Dublin Smart City Data Integration, Analysis and Visualisation

Hammad Ul Ahad
Technological University Dublin

Dublin Smart City Data Integration, Analysis and Visualisation

A Thesis Presented For the Award of Masters by Research by

Hammad Ul Ahad

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH

DUBLIN

TECHNOLOGICAL UNIVERSITY DUBLIN

Technological University Dublin – Tallaght Campus

Department of Computer Science

For Research Carried Out Under the Guidance of

Dr. Martin O'Connor

Dr. John Burns

Submitted to Technological University Dublin

December 2020

Declaration

I hereby certify that the material, which I now submit for assessment on the program of study leading to the award of a Master of Science (research), is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged with in the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Signature of candidate	Date

I/We hereby certify that to the best of my knowledge all the unreferenced work described in this thesis and submitted for the award of a Master of Science (research) is entirely the work of Hammad Ul Ahad. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Signature of supervisor	Date
Signature of supervisor	Date

Hammad Ul Ahad
December 2020

Acknowledgements

In the name of ALLAH the most gracious and the most merciful. All praises and glory to the almighty ALLAH, who gave me the strength and courage to complete this task, peace and blessings be upon Prophet MUHAMMAD.

I would like to express my gratitude to my principal supervisor Dr. Martin O'Connor (TU Dublin, Tallaght) for his careful guidance, tremendous patience, remarks, and eagerness to provide support throughout the learning process and for providing me a convenient direction to complete my research project. He gave me the opportunity to gain valuable skills by working on this project and encouraged me in sharpening my technical skills. He provided me consistent support in the face of uncertainty at any stage of my work on the thesis. I would also like to special thanks to my co-supervisor Dr. John Burns (TU Dublin, Tallaght) for his generous feedback, valuable corrections, and provision of the up-to-date information.

I would like to special thanks to Dr. Barry Feeney (Head of Computing Department, TU Dublin, Tallaght) for providing me a safe and supportive environment to finish my thesis and get valuable work experience. In addition, I would like to acknowledge the support received from all colleagues, lecturers, and technicians.

I want to say to my parents, Zaheer Ullah-(late) and Ishrat Sumayya, that my heart is full of love, respect and deep appreciation for all they gave me, for helping me to become a person I am today, for always being there to give me a hand. I also want to thank them for their ruthless judgments that gave me so much motivation and willingness to learn and grow. My gratitude also goes towards all the members of my family and relatives for all the sacrifices they had made to support me, and for their love and prayers.

A very-very special thanks to my parents, specifically to my beloved mother, as without her powerful prayers and support I would not be able to achieve anything in life.

To my beloved sisters Noor-un-Nisa, Najm-us-Sahar, Kokab Aseela, my younger brother Saad-ul-Samad, my brothers-in-law Alauddin Qureshi, Riaz Ahmed, and Hasnain Ali for their consistent support and prayers.

To my beloved aunt Frankie Kelleher, for always looking after me, loving me as her own son, and supporting me all the time.

To my close friends Dr. Shahzad Imar and Dr. Abdul Ghaffar for always giving me the best possible advice, constant support, encouragement, and motivating me whenever I am going through difficult situations in life.

Abstract

Data is an important resource for any organisation, to understand the in-depth working and identifying the unseen trends with in the data. When this data is efficiently processed and analysed it helps the authorities to take appropriate decisions based on the derived insights and knowledge, through these decisions the service quality can be improved and enhance the customer experience. A massive growth in the data generation has been observed since two decades. The significant part of this generated data is generated from the dumb and smart sensors. If this raw data is processed in an efficient manner it could uplift the quality levels towards areas such as data mining, data analytics, business intelligence and data visualisation.

The smart city idea is now strongly established, the proper and efficient management of public and private infrastructures will result in improved living standards. The Dublin City local authorities are providing over 200 open source datasets containing the captured data of the daily city life. However, in terms of processing this data there are some challenges available at the moment such as:

- The data silos are standalone and isolated in nature.
- The data formats in these data silos are also heterogeneous.
- There is often a lack of structure and meaningful context in the data.

Data Integration methods are used to integrate these heterogeneous data silos and provide users with data as a unified view. There is no automated procedure to integrate these datasets now, the integration step is the key enabler to transform these independent datasets into a smart city repository. The novel aspect of this research is the provision of an on-demand Extract, Load and Transform (ELT) framework that will perform the structural, contextual and augmentation enrichment towards the data streams and will generate the mapping rules between the source and target schema. Also, it will transform and integrate the selected datasets according to the selected facts and dimensions by the domain experts.

The primary objectives of the thesis consist in the development of an on-demand Extract, Load and Transform framework using the modern tools that will provide the integration services based on the dimensions and measures of the provided data sets and will fulfil the research demands. The designed algorithms for the ELT framework will perform structural, contextual as well as augmentation enrichment towards the provided data streams. In our work we have described our designed Application Programmable Interface (API) query service using the Python programming language. This API service is built to connect our Data Visualisation web application with our canonical data model. A user can communicate with our Dublin smart city repository by using our API service and could connect the API to any web or mobile application in the future. In the framework for visualisation we have developed an analytics web application for Dublin Bikes sharing scheme to demonstrate the new insights and bikes usage patterns, this application is connected to our smart city repository. In addition, based on our observations from the web application we have carried out a detailed analysis on Dublin Bikes data and developed an efficient economical model for Dublin bike sharing scheme and suggested a pairing strategy for replenishing.

Table of contents

List of figures

List of tables

List of Acronyms

API	Application Programming Interface
BSSs	Bike Sharing Systems
CMD	Canonical Data Model
CSV	Comma Separated Values
DBSS	Dublin Bikes Sharing Scheme
DSL	Domain Specific Language
DSM	Domain Specific Modelling
EPA	Environmental Protection Agency
ELT	Extract, Load and Transform
ETL	Extract, Transform and Load
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
ICT	Information and Communication Technology
JSON	JavaScript Object Notation
MDA	Model Driven Architecture
OWL	Web Ontology Language
RDF	Resource Description Framework
SOA	Service Oriented Architecture
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TFI	Transport For Ireland
XML	Extensible Markup Language

Chapter 1

Introduction

In today's world the current state of the art technology available is a result from the human nature of data collection, analysing it, building new knowledge and then improving the existing domains through the implementation of efficient decisions gained from new insights. The expeditious generation of digital data has been observed during the last decade. A huge portion of this data is generated by smart sensors or dumb sensors on a daily basis. The efficient processing on this raw data is important to further enhance the areas such as data mining, data analytic, data visualisation and business intelligence.
The concept of a smart city is now strongly established, the appropriate and efficient management of city's private and public services will contribute to the improvement of living standards. A smart city relies on the information kept in the city's core repositories which are required for the purpose of data integrating, analysing and sensing.
At the same time activities such as private and public services, environment safety, traffic management, industrial domains and commercial sectors can be improved by making them efficient. The architecture of a smart city is based on three main layers: perception layer, network layer and application layer, and to make the world intelligent, measurable and interconnected (Yong and Rongxu, 2010, p. 55). There are many new challenges arising in managing a city such as waste disposal, air pollution, resource allocation, health issues, traffic congestion, technical issues and many more.
A large amount of various categorical data can be accessed easily through internet for the purpose of collection and performing analysis. However, accessing data online in a few minutes through internet was quite difficult in the past. This available data typically belongs to the private and public sectors and various other city components and could be utilised for the benefit of the city itself. The applications that are involved for storing, integrating and processing city's data are known as Smart city applications.
However, the data generated from smart city applications are not always in a similar data

format which gives rise to a challenge of data stream analysis, transformation and integration. If these aspects are not dealt properly it can give rise to various problems such as data formats mismatch, missing data, inconsistent results and most importantly losing an opportunity to perform a cross domain analysis on city data repositories.
The smart city relies on the accurate decisions taken by decision making authorities of the city that is crucial for the further enhancement of the city services to improve the living standards of the citizens and give them the best possible service experiences. However, it is nearly impossible to take the effective steps and decision without analysing the raw data that is already stored in the city's repositories that needs to be retrieved to gain useful insights about the city life that has not been noticed before. The cross domain analysis becomes possible only when data is integrated in such a way that facilities the domain experts to view data in a unified manner across different data domains.

1.1 Background

The problem of merging the data from heterogeneous data sources is known as data integration, with an end goal of providing a unified data view to the end user. In real world applications the problem is related towards designing these integration systems, these problems are categorised by a number of issues that needs to be addressed before implementation. Some of these challenges are related to data integration, these mentioned issues specifically are designing an application for data integration, the processing of queries in these integration systems, handling the heterogeneous data sources and the effectiveness of these queries (Raghavan *et al.*, 2020, pp. 393-403).
The data integration is known as the method of merging various heterogeneous data sources with the aim of delivering a combined data view to the end user for data analysis and for further decision making (Fusco and Aversano, 2020, p. 257).
Research in the area of data integration for enterprise data sources with semi-structure, unstructured and structured data is ongoing for the last 40 years (Bergamaschi, Castano and Vincini, 1999, pp. 54-59). The traditional procedures for building multidimensional schemata or warehouses is based on common business requirements along with the enterprise databases to identify the specific subsets of data that are important for the construction of required datasets.
The challenges of integrating heterogeneous data sources is not new and exists for a long time. There are several issues that needs to be addressed, the data sources can modify their structures, new data sources can emerge, different users will be required to construct their demanded datasets, and the key challenge with smart city applications is exploiting the useful

information from the available data. For all these challenges an efficient, more flexible and an automated integration approach is required. It is crucial to have a methodology to analyse unknown data streams, which detects web-enabled data sources for the purpose of constructing required data models.

1.2 Motivation

Addressing a large amount of heterogeneous data is a time consuming and challenging task which needs an efficient data structure to ensure proper data processing and analysis. Thus, efforts are required to clean and transform the data by eliminating any unwanted semantics and outliers. The most effective method when dealing with heterogeneous data is the development of an Extract, Load and Transform (ELT) architecture.

ELT is a procedure to extract data from multiple data sources, load it in a data model and then transform it by adding useful context and removing unwanted semantics. The main goals of the ELT algorithms are to perform the data cleansing and data transformation in the required form.

The Dublin city local authorities are collecting and providing more than two hundred public datasets that are available open source (Dublinked, 2018). These datasets are containing variety of life aspects of Dublin city. Among these datasets some are providing real-time information to its citizens about transport such as Dublin Bus service, Luas (Dublin Tram-line service), Dublin Bikes (Bike Sharing Scheme), Dublin Air Quality and Weather information, and Dublin Motorways travel times. If utilised efficiently and effectively this data will lead the Dublin city to take further steps towards becoming a Dublin Smart City.

However, this data is in the form of heterogeneous data silos not even having matching data types. This information is stand-alone, separated and as isolated data silos they create a challenge to integrate them. An on-demand construction of Extract, Load and Transform (ELT) framework is the main component towards an automated integration service that contains several web-based isolated data silos into a repository and will facilitate domain experts to integrate these datasets according to their own required dimensions.

Thus, the construction of an automated and robust ELT service will facilitate the structural, contextual and augmentation enrichment for Dublin city data streams. Also it will allow us to perform querying on data by enable historical and predictive analysis and to present the results in an appealing format.

1.3 Research Tasks and Objectives

The first goal of the thesis is to address data integration in the context of heterogeneous data silos using the Extract, Load and Transform approach. The second goal is to develop an efficient resource allocation strategy for the Dublin Bikes Sharing Scheme. The hypotheses put forward in this dissertation are:

> Hypothesis 1: The increased automation of an Extract, Load and Transform framework from a non-automated or semi-automated approach can better facilitate the integration and contextual enrichment of diverse data sources.
>
> Hypothesis 2: The development of an efficient resource allocation strategy for the Dublin Bikes Sharing Scheme can facilitate in identifying the overflow and underflow bike stations and help minimise the number of lost customers at underflow bike stations and increase revenue at those stations.

To fulfill these goals we need:

- to elaborate the literature directly affiliated to our research work and adjacent fields.
- to design the data models and develop the algorithms to perform structural enrichment of real-time or near real-time web data streams and offline datasets.
- to develop the algorithms to perform the contextual and augmentation enrichment of the selected data streams.
- to developed the Canonical Data Model (CDM) holding the integrated data.
- to define the mapping rules among target and data sources.
- to design a fast and effective query service for the end user.
- to present the insights in a compelling visual format.
- to facilitate the design of an efficient economic model for the Dublin Bikes Sharing scheme.

1.4 Subject of Consideration

Due to the wide possibilities, time constraints and the broad area of our research work we primary focused on the cycling dataset which is Dublin Bikes Sharing Scheme. The cycle to work schemes are becoming popular around the world due to its non negligible benefits like health, easy access, and travelling at your leisure. These are some research papers supporting the novelty of research work we have done on Dublin Bikes Sharing Scheme.
(Caulfield, 2014, pp. 216-226) have presented the analysis performed on census data for the past 5 years data of Ireland's three largest cities i.e. Dublin, Cork and Galway. Dublin city was found to be the one with the highest number of workers using bicycles to reach workplaces or to travel the city.
(Rani and Vyas, 2017, pp. 43-55) illustrated an NS2 simulator to predict the available bikes for any bike station in the future by analysing the historical/previous data. The NS2 simulator is an event-driven simulation tool for the analysis of dynamic communication networks. There architecture provides the design, data flow, evaluation and techniques for the sensing network. The system will facilitate the customers to reserve a bike from any station from anywhere. To avoid the unavailability of the bikes in future the historical data was also analysed.
(Oliveira *et al.*, 2016, pp. 119-129) have performed a visual analysis to find out the frequent outage events meaning no bike is available and the station is empty. The identification of outage events are important to get an understanding of bike users behaviour and the minimisation of outage events are necessary to improve the service quality. It also helps to identify the importance of stations allocation in the network, the patterns throughout the day which could be important before designing a replenishing strategy.
(Caggiani *et al.*, 2019, pp. 117-126) have given an optimisation model that can allow the formation of spatio-temporal clusters based on the bike usage trends. The bikes stations during the service hours shows patterns like high availability to low availability and the reverse. In addition, there are times when the stations are full capacity or is completely empty.

1.5 Contribution

Our work will facilitate the schema generation operations, the integration of data models, the formation of a querying service and the construction of a visualisation framework for Dublin city to help create new knowledge and generating unseen insights. We have designed an automated and robust Extract, Load and Transform (ELT) framework that will perform

integration services after analysing the data streams. The ELT framework will automatically construct the schemata for both the online and offline data streams. After the formation of all schemata, the framework will again perform an analysis on the data sources and on the newly built schemata so as to construct the automated mapping rules that is necessary for populating these schemata with the data from the sources.
Then our ELT framework will generate the automated rules through the analysis of the schemata and data streams to construct our Canonical Data Model (CDM). In addition, the automated mapping rules will be produced that will populate the CDM with the data already stored in the data lake.
Then we created a query service for the Dublin Bikes dataset that will allow the users to communicate with our CDM for data retrieval. This query service is working as an intermediary service between our Canonical Data Model and the Visualisation application/Dublin Bikes Peak Times website. The website contains different tabs to demonstrate the Dublin bikes usage trends and present insights in a compelling visual format.

1.6 Thesis Organisation

We have arranged our thesis as follows: In Chapter 2 we present the state of the art literature review on data integration methodologies currently in use and also those that were used in the past. We also discuss the Bike Sharing Schemes around the world and the research done to enhance those bike sharing networks. In Chapter 3 we explain our adopted methodology and architecture for Extract, Load and Transform (ELT), we also explain the designed algorithms used for data modelling, data extraction, structural and contextual enrichment, and for augmentation enrichment. In Chapter 4 we discuss our designed query service for Dublin Bikes and the developed Dublin Bikes web application as our visualisation service. In Chapter 5 we conduct an analysis on Dublin Bikes dataset to identify the system lost users and suggest a cost metric to estimate the overall revenue loss during the service rush hours, we also suggest an efficient replenishing strategy. Finally in Chapter 6, we summarise our entire work and discuss the limitations, contribution and the possible future directions of our research work.

Chapter 2

Literature Review

In this chapter we present a literature review of the state of the art of our area of research. In Section 2.1 we begin by outlining a road-map for the structure of our literature review and analysis on the various data integration frameworks discussed in this thesis and we present the existing approaches to integrate the data for online and offline data warehouse architectures. In Section 2.2 we present the state of the art research performed on Bike Sharing Schemes in various smart cities with the overall goal to improve the service. In Section 2.3 we conclude with a summary of the advantages and limitations of the reviewed data integration frameworks and a summary of the analysis of the Bike Sharing Schemes. Also, we describe our vision for the possible enhancement of these models and for performing a resource allocation analysis on Dublin Bike Sharing Scheme.

2.1 Data Integration Architectures

In the modern world data integration is the key step enabling a door towards gaining new business insights for any organisation or business. These insights were not visible before due to the semantics and complexity available within the data. These insights are of great importance to enhance the overall organisational output and to increase system performance based on the decisions informed by newly created knowledge.

There are various data integration frameworks which are adopted widely by organizations and businesses around the world. The strengths and limitations of these integration architectures need to be considered before an actual implementation otherwise they may not provide the expected results. We discuss some of the popular data integration frameworks in this chapter and highlight their strengths and weakness in comparison to our framework. In section 2.1.1 we discuss the Cloud-based integration, the Multi-Level architecture in section 2.1.2, the CityPulse framework in section 2.1.3, the Online Data Integration Process framework in section 2.1.4, the Domain-Specific Automated ETL architecture in section 2.1.5 and the Automatically Generated Data Marts architecture in section 2.1.6.

2.1.1 Cloud-based Integration

Smart cities contain data from several private and public services such as transport, environment and health infrastructures which is collected from heterogeneous sources and then used by smart city applications but often used as an integrated source for decision making and information purpose (Raghavan *et al.*, 2020, pp. 393-403). To handle this quantity of data for integrating, analysing and to be used by different applications such as simulations, visualisation, future city models, decision making and provision of information and quality services to citizens is a challenge without implementation of appropriate techniques and tools. In Figure 2.1 the view of the main smart city thematic pillars is presented: smart economy, smart people, smart governance, smart environment and smart mobility which will provide resources that enables sustainability and flexibility against increasing demands.

The aim in creating this view is to consider an approach for smart cities which is capable of performing services such as data integration, acquisition of data, procedures for processing, and analysis that will facilitate integrating required information to increase the sustainability level of the city. Data management of thematic data domains in a cloud environment provides an integration opportunity to merge different data sources and process them in real-time frames. However, due to several challenges such as objects interoperability, flexibility, availability, adaptability, portability and other requirements adopting the cloud infrastructure for smart city applications is a difficult task (da Silva *et al.*, 2013, pp. 1722-1727). The authors at (Khan, Anjum and Kiani, 2013, pp. 381-386) addressed these issues by using ICT tools and software services for managing and analysing complex data of smart city, and by incorporating feasible cloud infrastructure such as proposed by (Khan and Kiani, 2012a; Khan *et al.*, 2012b; Mitton *et al.*, 2012).

Proposed Design for Analytical Processing Services

The Figure 2.2 shows the proposed architecture design for the cloud based integration, this architecture is divided into three phases that enables a unique aspect of knowledge base development.

The operation of the first layer is dedicated to collecting data from heterogeneous domains that further contain heterogeneous data formats. The metadata stores are filled by the metadata collected from these distinct data sources, containing mapping rules that are produced among the resources, links are produced and data is transformed in semantically operable and

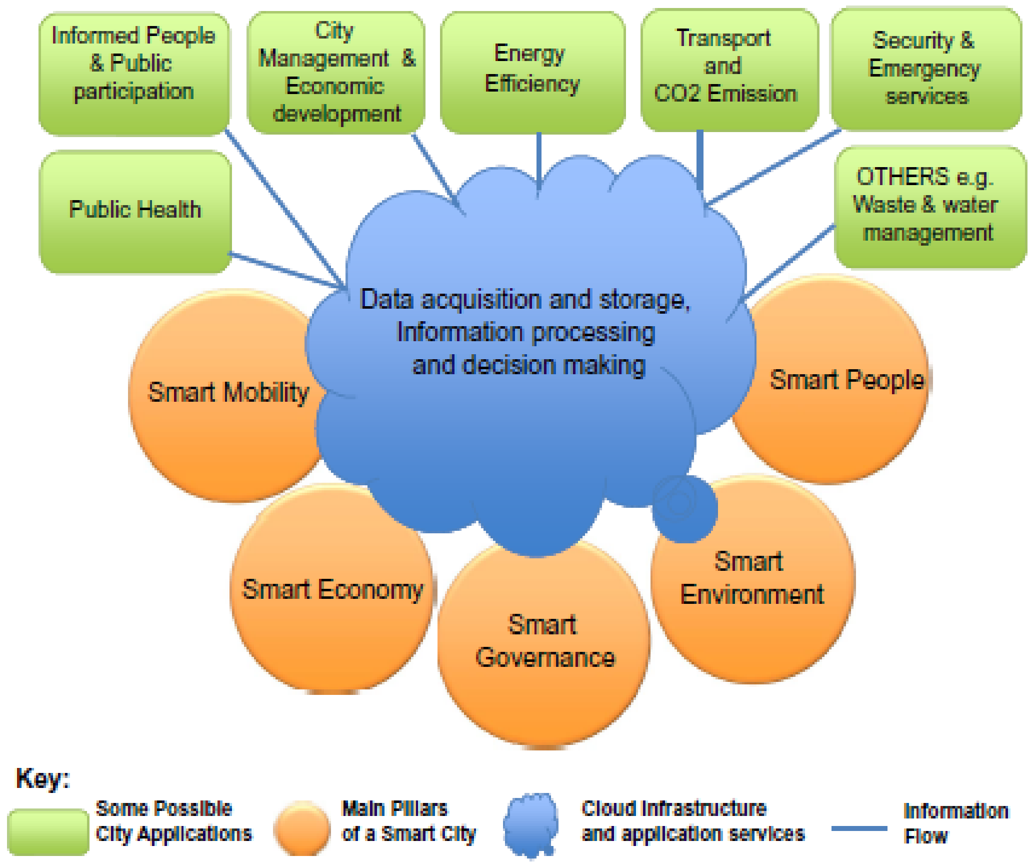

Informed People & Public participation
City Management & Economic development
Energy Efficiency
Transport and CO2 Emission
Security & Emergency services
Public Health
OTHERS e.g. Waste & water management
Data acquisition and storage, Information processing and decision making
Smart Mobility
Smart People
Smart Economy
Smart Governance
Smart Environment
Key:
Some Possible City Applications
Main Pillars of a Smart City
Cloud Infrastructure and application services
Information Flow

Figure 2.1 Cloud architecture for managing Cross-thematic data of smart city applications (Khan, Anjum and Kiani, 2013, pp. 381-386)

browse-able form. Then data is mapped through standard resource semantics: all the links established among the artefacts and resources will be stored in a Resource Description Framework (RDF) store. In required scenarios, the mashups and high level services can be composed for browsing and data usage purposes. The link phase will provide support to the work flows for building complex relationships among isolated data sources and facilitate in finding new scenarios. The queries will be served and databases can perform search operations if the linked data layer is present.

The cloud-based architecture by (Khan, Anjum and Kiani, 2013, pp. 381-386) provides an approach to process and analyse smart city data in the cloud. However, it has been observed that there is a need for the latest services and tools to facilitate efficient data processing and analysis. In addition, the identification for technical limitations and implications were not addressed for suggesting a viable future approach. In our research we have developed an Extract, Load and Transform (ELT) framework using the modern languages and tools

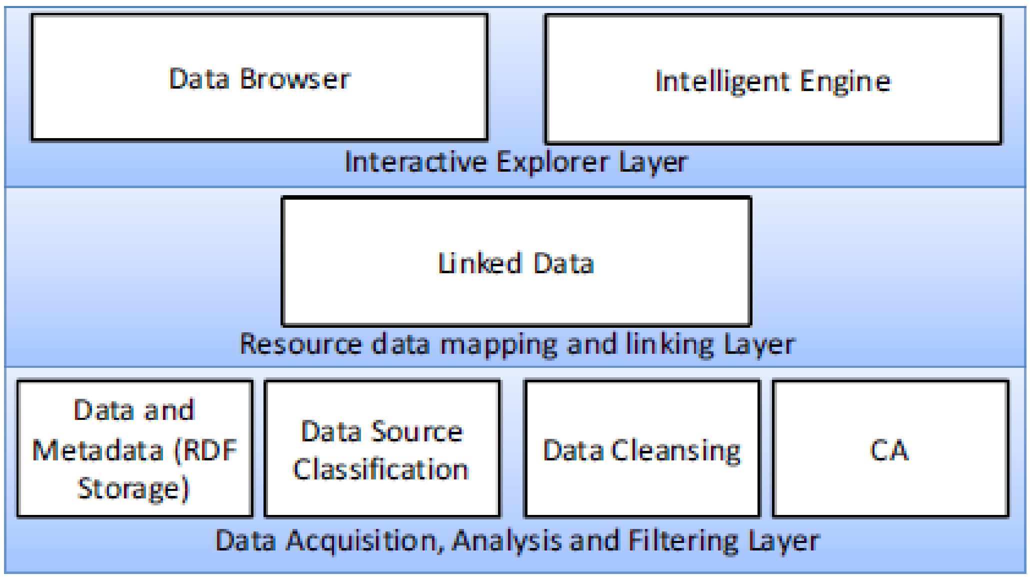

Data Browser
Intelligent Engine
Interactive Explorer Layer
Linked Data
Resource data mapping and linking Layer
Data and Metadata (RDF Storage)
Data Source Classification
Data Cleansing
CA
Data Acquisition, Analysis and Filtering Layer

Figure 2.2 Cloud-Based Integration Architectural Design (Khan, Anjum and Kiani, 2013, pp. 381-386)

to analyse and process the data efficiently. In the next subsection 2.1.2 we discuss the Multi-Level architecture for the smart cities.

2.1.2 Multi-Level Architecture for Smart City

In Figure 3.3, the authors explain about how their proposed architecture initially collects raw data which contains several heterogeneous data formats. These formats are then processed through semantic web technologies to convert different formats to a common one. Information collected is converted into common layout that is Resource Description Framework (RDF). The Resource Description Framework is an efficient way to integrate and share heterogeneous data sources and it uses a common technique for exchanging information. Then various tools can be used to utilise Resource Description Framework data for reasoning purpose. At the next stage the pre-processed RDF data may be exploited to retrieve high contextual information through semantic knowledge.

The integration process is done by using Web ontology language (OWL) for publishing ontologies. It is usually an RDF graph constructed from Resource Description Framework and ontologies together. It enables the concepts of classification by using the class data structure. It contains two characteristics: Data and Object Property, helpful in creating relationships between various classes. After the completion of data classification, uncertain reasoning and further knowledge enrichment can be performed by domain experts. Authors (Gaur *et al.*,

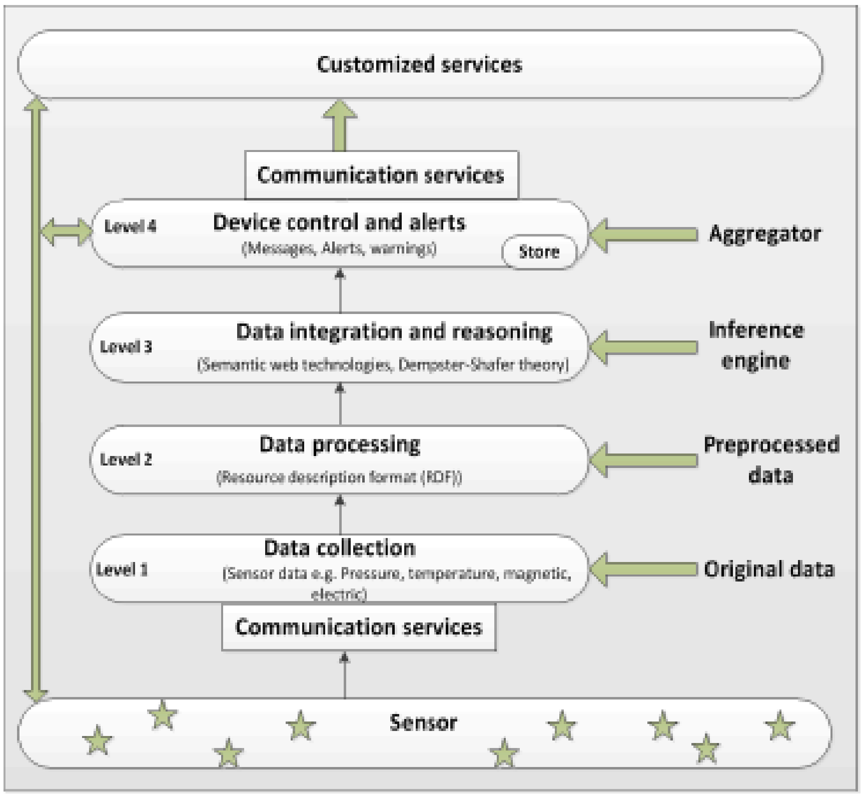

Customized services
Communication services
Level 4
Device control and alerts
(Messages, Alerts, warnings)
Store
Aggregator
Level 3
Data integration and reasoning
(Semantic web technologies, Dempster-Shafer theory)
Inference engine
Level 2
Data processing
(Resource description format (RDF))
Preprocessed data
Level 1
Data collection
(Sensor data e.g. Pressure, temperature, magnetic, electric)
Original data
Communication services
Sensor

Figure 2.3 Smart City Multi-Level Architecture (Gaur *et al.*, 2015, pp. 1089-1094)

2015, pp. 1089-1094) use the Dempster-Shafer framework for recognising activity, learning new rules and data integration from heterogeneous data sources. The Dempster-Shafer theory is also well known as the theory of belief functions, it is the generalisation of the Bayesian subjective probability theory (Frittella *et al.*, 2020, pp. 14-25). This permits the generation of new knowledge through uncertain reasoning and in constructing an intelligent smart system. SPARQL is a query language that is used for the query, retrieval and manipulation of records in RDF format. The SPARQL performs query and retrieves data, once the entire database is converted into RDF triples format. This level motivates the Low-level information fusion. The new rules learned can be stored during the process of extracting high level contextual information from raw data and used for building knowledge in the architecture.

The method of multi-level architecture approach of (Gaur *et al.*, 2015, pp. 1089-1094) uses the raw low-level information, then performs semantic enrichment by using customised intelligent applications in the city domain. Moreover, they have provided a fusion approach which depends on the knowledge of domain experts and a process of reasoning which uses the Dempster-Shafer evidence theory (Frittella *et al.*, 2020, pp. 14-25), this approach is beneficial for a smart city model when facing uncertainty in heterogeneous data. However, we observed that the data interoperability and scalability issues were not addressed by authors (Gaur *et al.*, 2015, pp. 1089-1094) in their approach of multi-level architecture design. In our research we have developed an Extract, Load and Transform (ELT) framework that performs the interoperability operations on the data such as contextual enrichment and augmentation enrichment to provide more sense to the data. Also, our designed Canonical Data Model (CDM) evolves/grows with the addition of new datasets. In the previous subsection 2.1.1 we discussed the Cloud-Based integration, in the next subsection 2.1.3 we discuss the CityPulse framework for smart cities.

2.1.3 CityPulse Framework

The CityPulse framework is depicted in Figure 2.4 along with its required components to integrate and process large quantities of data streams in a flexible and extensible manner. The exhibited open APIs by CityPulse components facilitates service creation and helps the application. This platform can be divided in two levels: The first level represents the modules that perform the processing of large data streams, it represents the tools that provide communication opportunity for application developers with smart city heterogeneous and unreliable domains. This level also permits processing, summarising and the discovery of data streams. The second level contains the modules for the robust decision supporting component, it represents the tools that provides various recommendations based on current city status and user context.

The first level components are useful to control and process data streams. The CityPulse applications are required by a cloud based components for service execution. In this manner the components can constantly monitor and process data streams. At this point an application can perform communication with components via APIs to get information of the current city status at any second. At the second level, when monitoring or recommendations of certain scenario/context are needed, the robust decision making components are executed. The second level components are not continuously running unlike the first level components which are continuously running.

The CityPulse architecture presented by (Puiu *et al.*, 2016, pp. 1086-1108) provides the advance services for the integration of different data streams, for the analysis, for the data interoperability, a scalable framework for smart city application development and for real-time analytics. The framework components are built as reusable APIs and available as open source. However, it is observed that a virtualisation method called Data wrappers is implemented to deal with the data heterogeneity. The collected data is not directly interpretable which makes it difficult to perform automated interpretation of the data. Therefore, semantic annotation is performed on the collected data to bring it in an interpretable format. The different

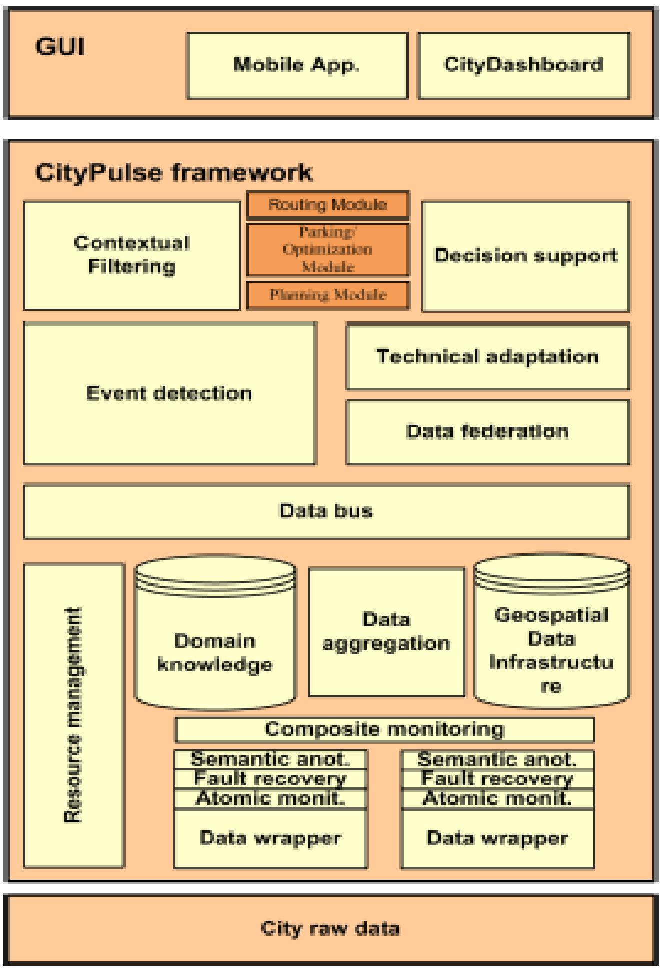


Figure 2.4 CityPulse Framework and APIs (Puiu *et al.*, 2016, pp. 1086-1108)

classification/quality of the collected data was examined from the start by giving the quality measures. The fault recovery and data aggregation methods were used to uplift the quality and processing of the datasets. In our research the developed Extract, Load and Transform (ELT) framework takes the data stream and performs the automated transformation, cleansing and augmentation operations on the data to provide quality to the data. In the previous subsection 2.1.2 we discussed the Multi-Level architecture, in the next subsection 2.1.4 we discuss the Online Data integration process framework for smart cities.

2.1.4 Online Data Integration Process Framework

The authors (Skoutas and Simitsis, 2007, pp. 1-24) applied a graph model for several distinct data sources to integrate them and store them in a single warehouse. The authors acknowledge a limitation whereby an ontology is required to be added for every single

graph structure created by the application designers for all source to target map production. However, the authors (Scriney, O'Connor and Roantree, 2017, pp. 23-35) have removed this limitation by developing automatic mappings by identifying the facts and dimensions. Also, a corresponding map file was generated for data source analysis.
In Figure 2.5 the authors have presented their proposed methodology using the Star graph model and explained their system process life cycle. There are different transport infrastructures for commuters to travel between work place and home such as: Bus, Bikes or Train.etc. Due to the increasing population it has become challenging for transport systems to increase system efficiency. The transport service providers and commuters both can get the benefits if the commute travel times are minimised. The authors have considered conceptually the city transport infrastructure as a graph (for example train stations, road junctions, and a bus stop). A departure or destination point is defined as the graph node, the edge between two graph nodes is considered as the path, and a route was a continuous sequence between two or more paths. The point of interest in this work was not only the travelled distance but also the distance covered in the minimal time. For this purpose the previous data was collected from the transport history to carry out the most efficient route at time ***t***.

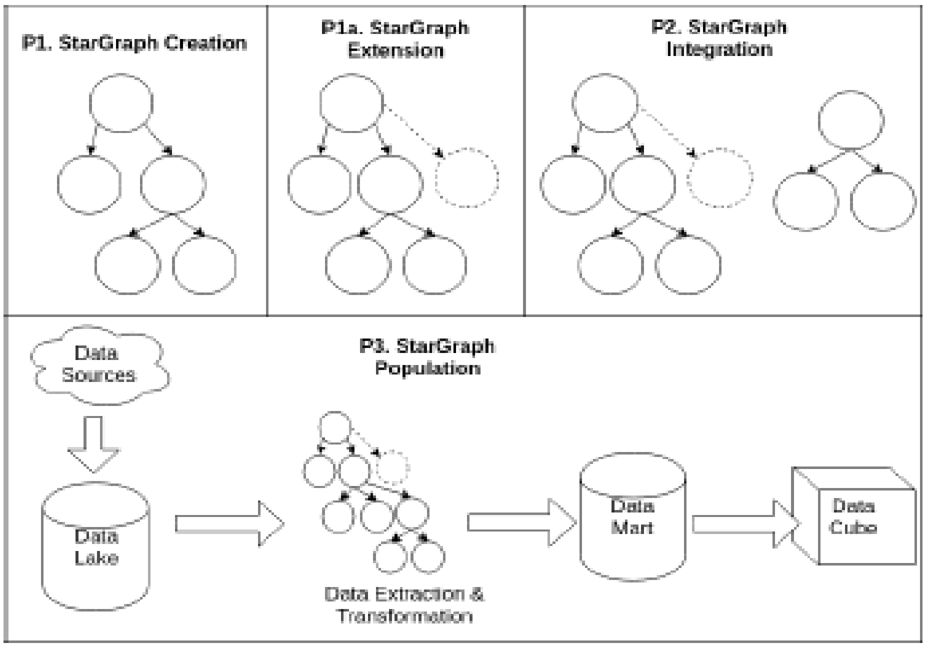


Figure 2.5 Extract Transform Load system based on StarGraph (Scriney, O'Connor and Roantree, 2017, pp. 23-35)

In Figure 2.5 the first phase constructs the StarGraph from the schema. The next phase is an optional phase which provides the user with an opportunity for editing the graph manually and adding information that is supplementary for further analysis. The next phase is divided into two parts: first part is the merging of the two graphs to construct ConstellationGraph, and the second part is then to integrate the StarGraph and the ConstellationGraph to gradually proceed towards the final schema. The final phase fills a fact table that is based on mapping rules, it will take two inputs: the mapping rules specific to a data mart and the data source streams.

The automated mappings approach of (Scriney, O'Connor and Roantree, 2017, pp. 23-35) presents the formation of ConstellationGraph using StarGraph, the ConstellationGraph is serving as a Data Mart for decision supportive systems. The authors have provided an efficient method for combining new data sources as these sources appear online and the appearance of these data sources in the data marts in almost real-time just after an update. The aim was to provide datasets more close to the real-time for travel/transport services. However, a time delay factor is observed during the integration operation, which happened because of the heterogeneous nature of the data sources containing different data formats. In our research we have minimised the time delay factor by using the Extract, Load and Transform (ELT) methodology in which first the data is extracted, loaded and then transformed instead of an Extract, Transform and Load (ETL) method where the data is extracted, transformed in the staging area and then loaded in the target schema. In the previous subsection 2.1.3 we discussed the CityPulse framework, in the next subsection 2.1.5 we discuss the Domain-Specific automated ETL architecture for smart cities.

2.1.5 Domain-Specific Automated ETL Architecture

The work presented by (Petrović *et al.*, 2017, pp. 425-460) uses a method called domain-specific modelling for automating processes of Extract, Transform and Load (ETL). Their focus was to implement the real formal specifications for processes of an ETL. An (ETL-PL) ETL platform is introduced which supports the two main activities: the first is the creation of ETL processes modelling according to the defined Domain Specific Language (DSL), second is the automatic generation of code for the models created in previous step. The code will be executed by the ETL platform environment.

ETL Conceptual Framework

The most crucial phases in developing the data warehouse process is the construction of a process which converts the business data into useful information which is possible using an

ETL (Martins *et al.*, 2020, pp. 609-619). The ETL conceptual architecture defines various Domain Specific Languages used for the various requirements by ETL processes. Basically, language that is for the data operations requirements, such as (ETL-O), the data and the language that will handle the order of execution is the (ETL-P) control flow.
There are three languages also provided which are supplementary: (ETL-T) the language for the templates transformation tasks, (ETL-E) the language for the requirements of several arithmetic and logical expressions and lastly the (ETL-D) language for the target and source data models. The defined DSLs concept which is related to the various ETL processes and they are completing all the previously mentioned needs related to the model languages.

The abstraction sets responsible for enabling several modelling languages integration will be provided by the Base ETL package. These Domain specific languages are explained as extension towards metamodel. We can also say that, every Domain Specific Language defines its own concepts through the extension of a base concept. The authors presented an example of metamodel in Figure 2.6.

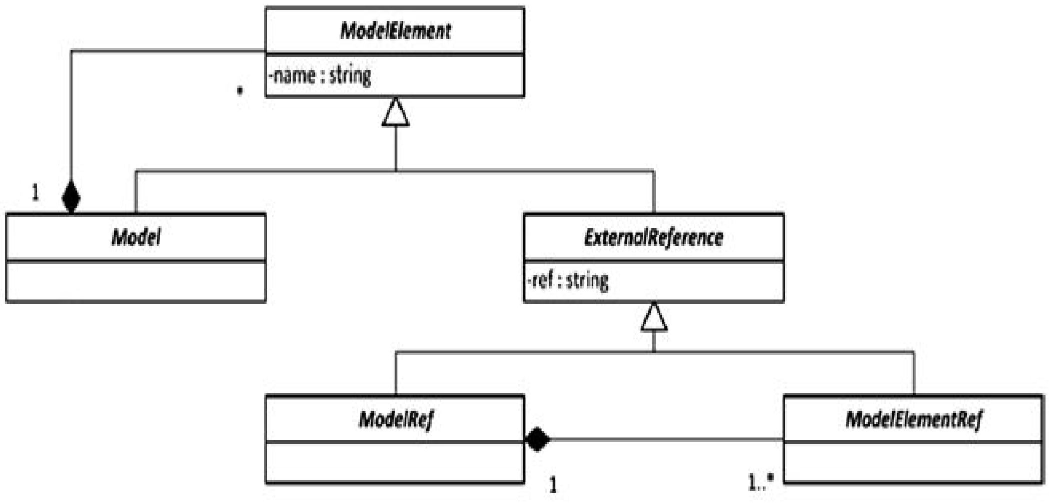


Figure 2.6 Base Metamodel of ETL (Petrović *et al.*, 2017, pp. 425-460)

Process Modelling of ETL

In this phase the separate models will be generated for each aspect of the ETL processes. To reduce the complexity of the ETL process modelling, various components will be divided into different complex models. Each model will be constructed by the concepts of Domain specific languages such as ETL-P to define the order of execution of these tasks, ETL-O to define the ETL data operations for process tasks. The construction order of these models are not predefined because every model is built independently. The designer needs to decide the order of modelling at the beginning.

The models constructed according to the proposed ETL-O requirements can be still very complex, based on the level of complexity of modelled ETL process, such as the number of required transformations. To help the construction of these models it needs to be done separately at several different abstraction levels and performs refinement of the previous steps by providing an in-depth description of ETL processes. To accomplish the hierarchical data process description it needs a set of diagrams to be introduced.

ETL Process Specifications Implementation

After the specification of a Domain Specific Language, the actual implementation is the next stage. This implementation will be achieved by automatically transforming the specifications into code that is executable. Model-Driven Architecture (MDA) relies on refining the models through successive transformations. This needs the extension of additional details towards automated constructed models manually, partial automation is achieved through this approach.

On other side, by ETL processes the demand is that some other requirements need to be completed by the software responsible for implementation of the proposed Domain Specific Languages:

- It should be able to automatically transform the ETL models into code which is executable during run-time.
- Contains the potential to rapidly change according to the proposed business needs by adapting easily the specification for ETL processes and can be executed quickly after changes such as extension, modification and even creation of new business requirements.
- The different versions execution should be allowed such as separate version of the same model.
- Easy scalability and deployment should be supported without conflicting with the operations of execution environment.
- It also should contain the feasibility of ETL process execution over distributed platform and the opportunity to perform parallel processing for different processes.

To meet the above goals, a Service Oriented Architecture (SOA) for developing software needs to be adopted, because it provides output as a very flexible and scalable solution along with supporting the distributive and parallel execution. To achieve the required flexibility the execution of the ETL processes, the ETL should be metadata driven such as: ETL-O or ETL-P models for execution and the models saved in the repository such as: ETL-D, ETL-E and ETL-T.

ETL Platform

To support the ETL specifications technologically and allowing automated ETL process development according to the Domain Specific Modelling approach a specific ETL platform is shown in Figure 2.7. The First layer which is the ETL Tools shows the environment for development, it contains the software tools that was developed to provide support towards building ETL processes. The second layer which is the ETL Transformation is responsible to transform models automatically into code, generators are supporting these transformations. The third layer which is the ETL Framework shows the environment for execution and it contains a number of services which are handling the management and execution of ETL processes.

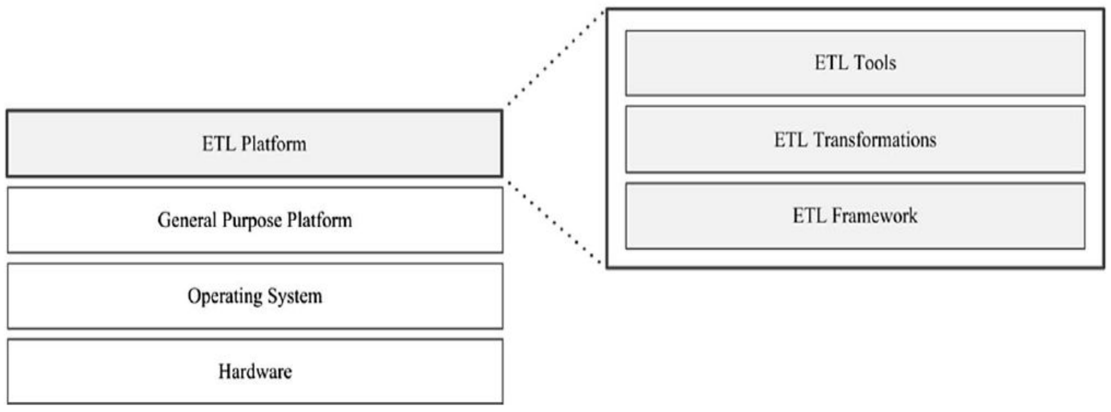


Figure 2.7 Proposed ETL platform Design (Petrović *et al.*, 2017, pp. 425-460)

The Domain-Specific Modelling (DSM) approach for software development is presented by (Petrović *et al.*, 2017, pp. 425-460), this approach is proposed for an automated and conceptual process development of ETL. Their work supported both the automatic ETL models development and the execution of the generated code from these models as well. There are only some methodologies available that allows the automatic ETL development using the Model-driven development techniques but these methodologies only deal with the ETL process development while, the approach of (Petrović *et al.*, 2017, pp. 425-460) deals with both the ETL process model construction as well as the execution code generation from these models. However, it was noticed that for the representation of heterogeneous sources in uniform source the reconciliation step was performed during the developmental process of the warehouse. Also, the construction of models are independent and their order of construction is not known in advanced. Thus, this depends on the designer to define that which aspect needs to be constructed first in the system, this depends on designer experience and knowledge about model construction. In our research a user does not need to be an expert for defining the order of construction in the system as our ELT constructs the data models and other data transformation operations automatically for the given data streams. In the previous subsection 2.1.4 we discussed the Online Data Integration process framework, in the next subsection 2.1.6 we discuss the automatically generated Data Marts architecture for smart cities.

2.1.6 Automatically Generated Data Marts Architecture

Authors (Scriney *et al.*, 2019, pp. 394-413) used a StarGraph model for every specific data source and an approach called constellation is used for integrating the data marts. Their methodology is based on the main four phases for the overall conversion and for several aspects in data transformation. The main process of automatically generated data marts architecture in StarGraph are mention below:

- Introduction of Stream
- Construction of StarGraph
- Construction of ConstellationGraph
- Materialising of Graph

StarGraph

In this section, we explain the working of a StarGraph. A star schema is capable of multidimensional data representation which is based on dimensions, relationships and facts. To represent the data captured from semi-structured data sources in dimensions and facts hierarchy a StarGraph is suitable or could be called a graph-based model. The first part will explain the StarGraph properties, while second part will define its functionality.
The StarGraph contains two tuple `S=(E,V)`, where `E` is the edges which is connecting vertices, and `V` is the nodes (vertices) in the graph. The node (vertex) is based on four tuples `v=(t,c,n,s)`, where `t` is the node datatype, `c` determines the node type, `n` is the node name

and `s` determines the data item location. The edge of StarGraph is based on three tuples `e=(r,y,x)` where, `(y,x ε v)` and `r` are the relationship among `(y,x)`.
The `s` in HTML/XML data for each node in StarGraph can be XPath and for JSON data it can be the dot-notation (Späth and Friesen, 2020; Bourhis, Reutter and Vrgoč, 2020). There could be five possibilities for the `c` attribute:

- **Dimension** that may mark the node as a dimension beginning.
- **Sub-dimension** will check if the node is sub-dimensional.
- **Dimension attribute** will determine the question node as an attribute from the parent node or not.
- **Measure** will determine whether a node is measure or not.
- **Container** will identify that whether a node is an instance which is holding another node.

Also, there could be five possibilities for *t* attribute:

- **String** will be any information in textual form.
- **Object** will be allocated to those nodes having no information but containing other nodes link, and during generation of StarGraph these nodes will be converted to Dimensions.
- **Number** will contain any data type related to numeric. Candidate nodes are representing by these nodes.
- **Date** will contain data types related to date.
- **Geo** will hold the nodes that contain the information of longitude and latitude (Geo-location)

The attribute relationship `r` in every StarGraph could be `(m-n, 1-n or 1-1)` between `e` Edges.

Introduction of Stream

This is the first process named as (P1) in this model which is responsible for adding new streams of data into the data lake along with the stream service process and it also adds newly appeared data source instances at a defined update time interval (Quix, Hai and Vatov, 2016, pp. 67-83). There are several things which the user needs to provide towards the data stream such as the URL, if available then schema of stream data, and the time update interval. This system used a low frequency for update which was weekly.
These sensor stream format is not provided to customers, in fact the customer should have the knowledge of stream frequency update, location and obtaining new instance as per update from the stream. Different formats such as reports are included in this definition which is sourced from private organisations and governmental statistical sectors. The update interval along with the source URL schema represented as a tree and stream metadata all were captured by a mechanism called Metabase.

Construction of StarGraph

There are a number of transformation steps involved in the process of StarGraph construction which initially create and then extends the StarGraph. This StarGraph construction mechanism is divided into five functions.

- **Annotation of Graph:** This function creates a Graph design from a data source, adds metadata along with the graph that explains the source. For data extraction from data source a query is also included in every single graph node. The queries are produced by documents traversal or (JSON/XML) object structure and node location is returned within the data source.

  The function uses the XPath queries for HTML or XML data. For CSV data files, the nodes hold the column numbers as a reference number and for data sources of JSON it uses the dot-notation. The edges between graph nodes are annotated with the relationship cardinality. After the construction of queries for all nodes, every node is assigned a data type. The queries used to extract data also determine these data types.

- **Classification of Nodes:** The second step after the nodes is to get their data types by classification of nodes. This step will assign the particular role to each and every node in graph to be performed in data source. This step will allocate the importance and position of nodes in the data mart. At the beginning the nodes in graph are scanned one by one, where classification function assign their roles.

- **Graph Editing:** This function is removing unnecessary nodes from the graph to perform the graph restructuring. The nodes determined as containers will be removed by the algorithm and the child edges associated with those container node will be re-linked to the parent nodes of containers.
- **Classification of Dimensions:** To get the multidimensional features of the graph by determining its dimensions and measures - this step is crucial. This function will evaluate all the measures determined as a candidate measures at the previous step. At a point where it is not possible to identify them as a measure, then they will be considered as a part of dimension object after reclassifying them to dimension-attribute.
- **Classification of Measures:** The measures are arrange into the same fact based upon their shared relationships. For instance, If two measure nodes having edges towards one dimension node, then they will be identified along with a single fact. It will be determine by inspecting the edges joining every single measure towards dimensions and gathering them into dependency sets.

Construction of ConstellationGraph

The user chooses the data sources using the ladder strategy used by the system to perform integration (Batini, Lenzerini and Navathe, 1986, pp. 323-364). To conclude, the initial two sources are integrated and then the next data source is integrated with the already integrated one schema. It will execute until there are no sources left behind to integrate. This results in an integration process that contains n-1 steps where n is the total number of data sources to integrate. The construction of the Constellation Graph is based on five steps which are described below:

- **TermMap:** This function solves the heterogeneous issue in graph by appending the canonical terms to graph for source attributes. A simple use of TermMap function is residing in the Geo dimension containing many different ways to represent a country name. For example, the terms 'Spain' and 'Espana' cannot be related using standard algorithm for string matching. However, they are referring to the same country Spain.
- **TypeMap:** For resolving concepts issue still remains that need to be treated of exact type, even their appearance is different.(e.g. 'Spain', 'Ireland' are two different values of same type country). Canonical types are assigned by type mapping function to every node by collecting concepts used for linking the nodes.

- **MetamodelCheck:** This function will check if there is integration required information available in the measure or not. This function checks the connections of measure towards nodes along with types Geo, Metric, Date, Unit and Item.
- **GranularityCheck:** This function performs the comparison among the graphs to check whether their granularity is of same level or not. It verifies that two graphs are having the exact hierarchy dimension to ensure that they are at the same hierarchical level.
- **DetermineStrategy:** This function is responsible for analysing the attributes of the metamodel related to the data sources to develop an integration model. This function will perform several steps in order to complete the analysis phase.
  - The first step will create the dependency sets for every single measure of all data sources.
  - The second step will choose how the dependencies will be integrated. It is possible in three different ways: no integration, row appending and column appending technique. The result will show how the combinations of sets will be integrated.
  - The third step will perform integration management through transitive closure which was not possible in the previous step. This step will produce a combination graph, and each node of the graph will be considered as a dependency set, while the nodes edges will be the integration procedures.
  - The last step is the graph collapse step in which the remaining nodes can be assigned to a fact in the last constellation. This will check all nodes that are sharing an edge of row append and will combine it. The new node inherits all of the edges from a collapsed node.

Materialising of Graph

The Materialising of graph is based on the logic of data model and the wrappers constructed for XML, CSV, JSON and HTML (Scriney, 2018, p. 76). It is not possible to provide a detail working of materialising of the graph in this chapter but you will see the produced mappings in Figure 2.8, showing how the fact containers and dimensions are properly extracted using wrappers.

The automating data mart generation from semi-structured data streams is presented by (Scriney *et al.*, 2019, pp. 394-413), this approach is capable of identifying measures, facts

```
{
"name" : "aim\_1,bb\_1",
"source_meta" : {
    "aim\_1" : {
        "DATE" : [
            {
                name:"2016-03-05",
                type:"WEEK",
                src:"//table/tbody/tr[2]/th"
            },{
                 name:"2016-03-05",
                type:"WEEK",
                 src:"//table/tbody/tr[3]/th"
            } ],
        "GEO" : [
            {
                name:"CANADA",
                type"Geo",
                src:""//table/tbody/tr[6]/th""
            },..
        ],
        "ITEM" : [
        {
            name:"PIGS",
            type:"Item",
            src:"STATIC"
        } ],
    }
```

Figure 2.8 Mappings for Final Constellation Graph (Scriney *et al.*, 2019, pp. 394-413)

and dimensions from not seen sources of data. The mappings between sources and targets are automatically performed. This system automatically builds the data marts so no existing warehouse is required. Their approach requires a lightweight ontology and a little interaction with the user to generate the data marts. However, the ontology used in this approach is used purely to facilitate this integration scenario in order to support a fast developing and deployment process, but if implemented to any other integration case, it may not give the same performance/results as it has provided in this scenario. In addition, the data stream update frequency in their system is low i.e. weekly but in our research we have used the sensors data stream having an update frequency in real-time.

In the work presented by (Scriney *et al.*, 2019, pp. 394-413) it is mandatory for a user to provide the URL of the data source, the update interval of the data source, and a schema of the data source (if exists). A user should have the information of the stream's location and update interval and required to acquire new instance according to the update. Based upon the idea and research of the authors, in our research we have created a robust and flexible

Extract, Load and Transform (ELT) framework to pull the data from live streams (Transport). The ELT produces the data models automatically for the provided data streams. Our ELT is able to produce the automated mapping rules through the identification of facts, measures and dimensions. Our ELT framework produces the automated mapping rules required for both: the generation of data models, and for the mapping rules required between the data source and the target schema. It also performs the structural, contextual and augmentation enrichment towards the datasets by using the data semantics taken from the original data streams. It is not mandatory for the user to have knowledge of the stream's location or update interval although a user can set the data extract interval according to requirement. In the next section 2.2 we have discussed some analysis methodologies performed on the Bike Sharing Schemes (BSS) around the world.

2.2 Bikes Sharing Schemes

The bikes sharing schemes are becoming popular in the modern day world for several reasons such as: you can rent a bike from any Bike station and lodge it to any Bike station with in the network, it gives you the opportunity to become healthy while commuting, and you can travel the city at your leisure.

2.2.1 Growth of Cycling in Dublin City

Dublin City is Ireland's capital city with a figure of 1.2 million population (Census, 2016). It is a suitable cycling city because of its relatively flat topography. The climate of Dublin city is mild type having an average rain per month of 61 mm, the average rain per month in Amsterdam is 64 mm, in Copenhagen city it's 44 mm and in Freiburg this average figure is 78 mm. According to the (Climate, 2019) due to the traditional reputation and climate make these cities most cycle friendly cities of the world. Dublin's topography and its climate is an important factor in the increase of the growth rate of cycling. To present a better image of cycling in Dublin city and to raise the growth rate of mode-share cycling there has a constant effort since the year 2008.

Authors (Caulfield, 2014, pp. 216-226) carried out research to find the impact on the cyclists rate through the demographics changes over time in Dublin city, for this support they took the census data of Dublin city of 2006 and 2011 to perform a comparative analysis. The data represents the regular routine trips of an individual to work; by the medium of transport (Cycle) they mentioned that they use the most often. However, for examining cycling this data is not very ideal, due to the seasonal variations in it e.g. as the demand/rate of cycling is increased during the summer season and decreased during the winter season, but the reason for selecting this dataset by the authors was being the only data sources available in Ireland to perform change comparison of the modal share over the time. The dataset showed 1.8 million work trips occurred in 2006 and 1.7 million trips in 2011, the Irish census data is for all individuals resident in Ireland.

To determine the factors that were impacting the Dublin city cycling growth (Caulfield, 2014, pp. 216-226) have used an estimated multinomial logit regression model. It would have been of interest to perform an analysis using a spatial multivariate model but the dataset provided doesn't have the capability to do it, because from each dataset matching records were not possible. Therefore, this task was performed using the electoral districts dataset. The dataset was divided into three sections: first those areas that have received an increase number in cycling, second the areas noticed a decrease in number and last those areas that were the same and have no changes experienced. Multinomial logistic regression analysis was performed to see the relationships between cycling rates and demographic factors. A Multinomial logistic regression also known as 'Multinomial Regression' is used to predict a nominal dependent variable given one or more independent variables (Multinomial, 2002).

Analysis on growth of cycling in Dublin City

Authors (Caulfield, 2014, pp. 216-226) presents the outcome of the analysis performed on census data, they have examined and presented the past 5 years data for the three largest cities of Ireland including Cork, the second largest city containing the population of 198,000 and Galway the third largest city with a population of 93,000. According to the results Dublin has the highest share of workers using cycle to work in Ireland. It also showed that in 2011 mode-share of Dublin cyclists has decreased as compared to the year 2006, while the total number of cyclists has expanded.

The volume of traffic entering the city is counted annually by Dublin City Council by the month of November. However, when interpreting these results it should be noted that cycling numbers are reduced during the month of November due to the seasonal variations. The authors (Caulfield, 2014, pp. 216-226) identified that instead of general policies to increase the growth rate of cycling, more targeted policies should be adopted to encourage cycling for commuters as their primary mode of transport. The findings depict the transformation of a city from low cycling towards a city that embraces cycling thus promoting an Eco-friendly environment and a feasible alternative compared to other transport modes.

However, we have observed that the analysis carried out cannot identify any specific measure responsible for the modal shift towards cycling. In Dublin, the analysis showed the impact of measures used. However, these results may not be directly reproducible at other cities. Also, the growth focus was on the city centre area and many other areas have showed no increase or even a decline. It is due in no small part to the major investment on cycle infrastructure in this area and the lowering of speed limits in city centre area. In our analysis we have identified the data trends of cycling through out the city and have suggested a pairing strategy that could uplift the service quality to facilitate the modal shifting towards cycling. In the next subsection 2.2.2 we discuss the Smart Bike Sharing System for smart cities.

2.2.2 Smart Bike Sharing System to make the City Smarter

There are many challenges in the daily lives of citizens in cities concerning the living culture and the management of city developments. According to the United Nation 2007 statistical data (Naphade *et al.*, 2011, pp. 32-39), 50% of the global population were living in cities. In the same report it is estimated that almost 60% of the global population will be living in cities by 2030. On top of all other challenges, we can highlight some major issues related to environmental pollution, air quality and transport congestion. Many transport issues in urban areas can be addressed by using bicycles as a mode of transportation in our daily lives. An increase in the usage of motor vehicles has led to corresponding increases in the problems relating to congestion, less space and amenities, accidents, cost, air and noise pollution, demand in energy consumption and have a negative impact on the natural Eco-system. In the coming years, the use of bikes for commuting should be a vital solution as bikes have no negative impact on environment (Rietveld and Daniel, 2004; Geels and Raven, 2006).

Smart Bike Sharing System Methodology and Analysis

The authors (Rani and Vyas, 2017, pp. 43-55) have presented an NS2 simulator to allow a registered user to reserve a bike from any particular bike station. It can also show a user their previous rides if any, and permit a view of the station map and in addition, can check the number of bikes available at any moment. They have presented a design, workflow and evaluation for their bike sharing system and the techniques for a sensor network. The system allows the user to reserve a bike at any particular station from their home/work without waiting if the bikes are not available. To avoid the unavailability issues in future the recent years data was analysed. The bike stations will be scattered throughout the city and enable a user to collect a bike from any particular station and return it at any desired station. The entire system will be automated and does not require human assistance on any of the hubs.

The bike sensors will be responsible to collect all the data through out the journey and will pass it to the nearest bike station.

However, there are some limitations to the presented work such as the data type identification. For instance the NS2 simulator system is not able to identify the type of data stored in the trace file (a type of logging file) received from the bikes. The format of the data in the trace file typically consist of: event, time, source and destination nodes, packet name, packet size, flags, flow id, source and destination address, sequence number and packet unique id. The bikes data is continuously updated in the database which leads to storage and analysis difficulties of over 17 million records. In addition, it is difficult to analyse the data for any specific country due to the inconsistency in the availability of data. Also, for the bike to transfer its sensor data it has to wait until it comes in contact with any nearby station and it will try to use the logic for multi-hop communication. In our research we have used the clean and transformed data to forecast the number of lost users and revenue from our CDM populated by our developed ELT. In the previous subsection 2.2.1 we discussed the growth of cycling in Dublin city, in the next subsection 2.2.3 we discuss the analysis on Citi Bike Sharing System of New York.

2.2.3 Visual Analysis of Citi Bike-Sharing System of New York

Nowadays. the sharing of public bikes is becoming popular and implemented in most of the countries, also called the public Bike-Sharing Systems (BSSs). The total set of bike stations are scattered across the town/city and providing bikes on rent. The user can withdraw bike from any nearest station, then travel for an allowed time period and return to any station nearby. There is s challenge with such a scheme as it gives less opportunity to the operational staff to control the distribution of bikes as the bikes are moved frequently by the commuters. This control is necessary to ensure the equal distribution of bikes at all stations, so the users can pick bikes from any station and return to any station. An outage event occurs when a station is either full (additional bikes cannot be returned) or empty (bikes cannot be rented). To prevent an outage event the station re-balancing is necessary. To move the bikes from one outage station to another lorries are used, this requires addressing issues such as best possible route, least time, minimise fuel consumption, traffic congestion and also some busier bike stations needs to be re-balanced more often than other stations. New York has one of the country's largest bike sharing system known as Citi Bike which was developed in May 2013, containing 330 stations scattered across the city, serving 6000 bikes from these stations and more than 11000 docks (CitiBike, 2013).

The bike sharing systems re-balancing operations around the world are mostly done during the night time when there are less active users or no service available. However, the

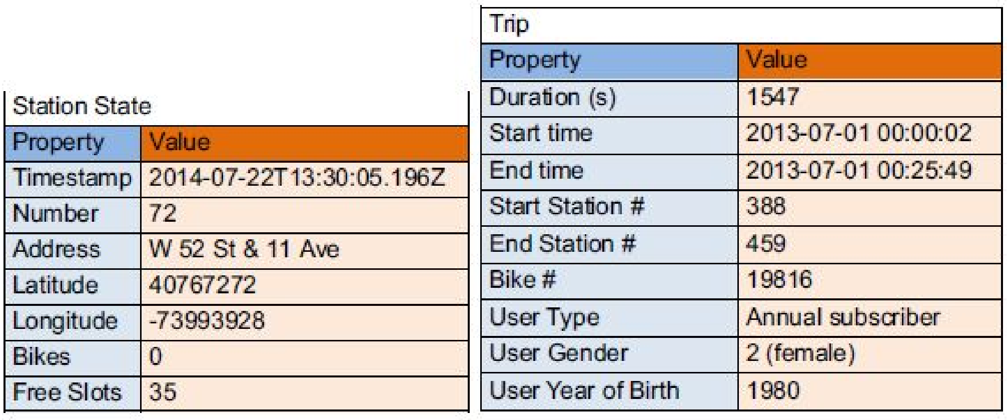
New York Citi Bike sharing system performs these re-balancing actions during the daytime to deal with the immense amount of usage demand (Oliveira *et al.*, 2016, pp. 119-129). The capturing of bike stations real-time status could help to understand the commuters behaviour and unbalanced stations. By performing an analysis on the bike usage data it could help to improve the re-balancing techniques, help in infrastructure improvement, and may facilitate the user to use the system more effectively.

Visual Analysis Methodology of Citi Bike-Sharing System

Authors (Oliveira *et al.*, 2016, pp. 119-129) have introduced an interactive visualisation system that facilitates the exploration of the dynamics of the bike sharing systems through the analysis of the historical dataset. The authors aimed to identify the bottlenecks in the bike sharing system such as frequent outage events which need to be addressed to achieve an efficient commuting experience and to identify the impact of city life diversity and behaviour events of the bike sharers. The popularity in bike sharing schemes have an impact in the city life routine. The authors sought to understand the importance of stations distribution roles, the changes in source and destination stations throughout the day, which maybe of importance while designing the bike sharing systems balancing strategies. Furthermore, to recognise any trends difference during the different time period since the Citi bikes service is deployed in New York.

The status of all the bike station data was fetched by querying the feed (CitiBike, 2019). The feed contains the current balance state of all the stations and updates the state whenever any changes occur in the balance of any bike station. The feed contains the bike station unique id, name of the station, total number of available bikes, number of free bike stands, station GPS coordinates, and the last update timestamp. They have recorded the variations in JSON feed at every 30 seconds for a period of 8 months since June 2013, more than 10 million updates were recorded for all 330 bike stations. In Figure 2.9 the trip and station status data is presented.

The update events occurs when a commuter withdraw or returns a bike at any station. The sequence of these events recorded at specified time interval leads to a time series. The usage details of bikes at a one station on any selected day is a sequence of n events. The bike return event is recorded whenever a number of increase is captured in the total current status of station. The total bike slots of each station is identified as the capacity of that station. The balance of a station is calculated using the difference between the number of available bikes and the total capacity available at the station. Because of the visualisation limitations the interval of the time series was reduced by (Oliveira *et al.*, 2016, pp. 119-129), they have performed an aggregation at an interval of 15 minutes to re-sample the series, because this



Station State

Property	Value
Timestamp	2014-07-22T13:30:05.196Z
Number	72
Address	W 52 St & 11 Ave
Latitude	40767272
Longitude	-73993928
Bikes	0
Free Slots	35

Trip

Property	Value
Duration (s)	1547
Start time	2013-07-01 00:00:02
End time	2013-07-01 00:25:49
Start Station #	388
End Station #	459
Bike #	19816
User Type	Annual subscriber
User Gender	2 (female)
User Year of Birth	1980

Figure 2.9 Trip and Station Status (Oliveira *et al.*, 2016, pp. 119-129)

interval duration is suitable for a complete 24-hour period and for the usage rate of stations. The aggregation interval for the trip data was 1 hour. The aggregated computed measures for trips starting or ending in every interval is: withdraw/return difference, total capacity, balance, total bike trips, the total trips incoming and out going, outage(full/empty) or no outage events, total trips, trip distance and duration, total number of outgoing destinations and incoming origins. All the trips were divided into three different categories: the cyclic, incoming trips, or the outgoing trips. The origin and destination stations are different in incoming and out going trips while in cyclic it is the same.

Analysis on Citi Bike-Sharing System of New York

Authors (Oliveira *et al.*, 2016, pp. 119-129) presented a visual analysis of the bike sharing system in New York city called the Citi Bike. They have leveraged some insights related to the Citi Bike usage for the initial 10 months time period. Their solution could be applied to other bike sharing schemes around the world. The authors visual analysis includes the comparison among various days in a week, among weekdays and weekends, and comparing the same days over different periods. Errors were encountered and easily recognised during different days and time due to the sudden transition of colours in various matrices. However, the authors assumed the reason for these changes are as a result of some special events going on in the city but they are unable to find any such events in the news that could demonstrate a relationship with these sudden colour transitions. Thus, the authors assumed these changes as an output from the operational issues. Furthermore, another issue is to identify the outage stations the authors have presented that arranging the rows brush-wise in the matrices by using various variables was useful to find out the places on the map that reflects the required behaviour. In addition, seasonal trends were observed with a noticeable decrease in the bike usage during the winter season. In our research we have developed a Canonical Data Model that contains both the weather data and the cycling dataset that will facilitate to perform cross domain analysis. In the previous subsection 2.2.2 we discussed the Smart Bike Sharing System, in the next subsection 2.2.4 we discuss the user satisfaction model for resource allocation in Bike-sharing systems.

2.2.4 User Satisfaction Based Model for Resource Allocation in Bike-Sharing Systems

In the past few years to commute within the city using a sustainable and healthy mobility public bike service also known as Bike Sharing Systems (BSSs) has gained popularity. The bike sharing system is a public service that allows users to rent a bike, it is based on different stations scattered across a city that allows user to pick a bike from the nearest departure location and then drop it to any station close to the destination. Each station may contain a set of bikes for customers to avail of.
The aim of a BSSs service is to provide users with a bike whenever they want and thereby eliminating the expenses and responsibilities of bike ownership, storage and security. In addition to these benefits, it results in a decrease in traffic congestion and harmful gas emissions, provides potential financial savings for an individual, helps to maintain a healthy life style by using the bike and also works as a bridge between users and reaching other transport services (Shaheen, Guzman and Zhang, 2010, pp. 159-167).
An issue of importance in bike sharing systems which needs to be addressed is the unavailability of bikes: this situation occurs when all the bikes from a station are taken and there are no available bikes remaining, and the opposite situation can be that a user cannot find an empty rack at any station to return the bike. The resource allocation of racks and bikes needs to be managed by the bike sharing system operators, to ensure that the service users are provided with sufficient resources so that the bike service can be efficient and reliable alternative to other transport means (Fricker and Gast, 2016, pp. 261-291).

Methodology of User Satisfaction Based Model

Authors (Caggiani *et al.*, 2019, pp. 117-126) have presented an optimisation model that can facilitate the formation of spatio-temporal clusters based on the bike usage frequency patterns data of bike sharing system stations in order to improve the user experience. The bikes availability at every station during the service hours fluctuates from many available to few available to many available again according to the demand at that station. However,

there are some time periods where a bike station becomes complete empty or full.
The authors have considered these two scenarios: a completely empty station or completely full station. These two scenarios were named *full-port time* when no parking space is available and *zero-vehicle time* when no bike is available. In order to increase the users satisfaction these two factors needs to be kept minimum, otherwise it will result in forcing the customer to take a bike from another bike station or to use an alternative mode of transport respectively. In Figure 2.10 the proposed model for increasing the customers satisfaction by re-balancing the bikes and racks in bike sharing systems is illustrated.

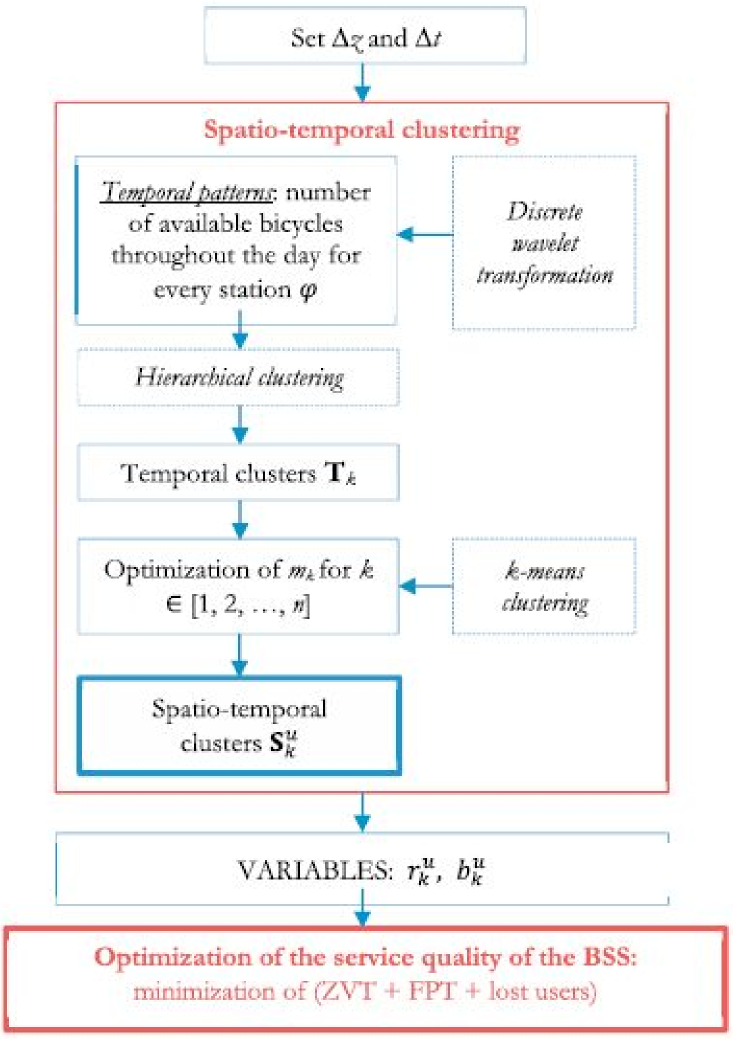

Set Δz and Δt
Spatio-temporal clustering
Temporal patterns: number of available bicycles throughout the day for every station φ
Discrete wavelet transformation
Hierarchical clustering
Temporal clusters $\mathbf{T}_k$
Optimization of m_k for $k \in [1, 2, \ldots, n]$
k-means clustering
Spatio-temporal clusters $\mathbf{S}_k^u$
VARIABLES: r_k^u, b_k^u
Optimization of the service quality of the BSS:
minimization of (ZVT + FPT + lost users)

Figure 2.10 Customer satisfaction model for resource allocation in BSS (Caggiani *et al.*, 2019, pp. 117-126)

In Figure 2.10 the work flow for user satisfaction model for resource allocation in bike sharing system is illustrated. Every station ϕ has its own data patterns of available bikes which change over time. The data for bikes at every station is gathered at time interval $\mathbf{\Delta t}$. The highest value that could be reached at each station during every time period is equal to the number of bike stands at that station (meaning station is full of bikes). On the other hand, if all the bikes are taken and the station becomes empty, the gathered number will be zero.

The authors selected a time period of BSS operation and named it $\mathbf{\Delta z}$. When the $\mathbf{\Delta z}$ is defined, then the required time period $\mathbf{\Delta t}$ data for each station ϕ is retrieved from the database: these are the temporal trends of the each stations which the authors propose to aggregate /cluster. The initial spatio-temporal analysis of this system rely on two main divisions of clusters: the temporal clusters **T** and spatial clusters related with every element **Tk** of the set **T**, and is called spatio-temporal clusters (set **Sk**).

Analysis on User Satisfaction Based Model for Resource Allocation in Bike-Sharing Systems

The authors (Caggiani *et al.*, 2019, pp. 117-126) presented a methodology to address the resource allocation issue: that is zero available bikes or zero available racks/bike stands. To overcome the zero available bikes scenario or full parking space situation they have performed a historical analysis on the data usage patterns of bike stations, smoothing the trends having fluctuation, and performed the initial spatio-temporal clustering method. The Spatio-temporal clustering is a method of forming groups of objects based on their spatial and temporal resemblance (Kisilevich *et al.*, 2010, pp. 855-874). The authors model has the potential to identify the need of adding or eliminating racks to every bike station, and allocating the optimal amount of bikes during the same time period, also the capability of eventual realisation step for the creation of additional bike stations. Then they applied the spatio-temporal clustering method on both a small test-case scenario and on a real-size network and presented their analysis in parallel with an analysis for sensitivity.

This method has been implemented on a station-based bike sharing system, and it could be implemented to a free-floating one (Pal and Zhang, 2017, pp.92-116), where kiosk machines and docking stations are not compulsory, as bikes can be locked to an ordinary rack before or near the destination bike station. The authors have applied the bike sharing system enhancement for quality service from the customer point of view: such as minimising the situation where a customer cannot find any available bike to ride or cannot find an empty rack to park an already taken bike, and also minimising the number of lost customers at the same time from the system.

The output results seems to provide a suggestion for an ideal path to invest the current allocated budget for the general enhancement of the existing system. In addition, it was noticed that this methodology also provides leverage to operator interests: ensuring that the zero available bikes and zero park space scenarios are minimised, which will also decrease the need of performing frequent bike relocation and will crucially decrease the management expenses. Furthermore, the reduction in the loss of customers would lead to an increase in revenue generation.

However, we have noticed that temporal clustering for the bikes stations on a daily basis (7 days a week) has not been performed as there is a difference in bikes usage trends between weekdays and weekends. This analysis could be useful if performed on each bike station at the start of the day (morning) and then can be measured according to the need/demand at each station for the entire week. In our research we have performed the analysis on the bike stations on a daily basis meaning for the entire week i.e. from Monday to Sunday which includes both the weekdays and the weekends. In our analysis we have identified the locations within Dublin city with high availability, medium availability and low availability bike stations with the help of a heat map. Also, we have developed an economical data model to increase the efficiency of the Dublin Bike Sharing System and suggested a pairing strategy to replenish the low availability bike stations with bikes from the nearest high or medium availability bike stations during the morning and evening rush hours. This strategy will help to minimise the lost customers and consequently lead to an increase in the overall revenue.

2.3 Summary

The main data generation sources of a smart city are smart phones, sensors, and citizens but then to integrate these data sources with the city repositories to produce information crucial for decision making and to perform analytical reasoning. This requires an integrated and uniform information model to be carefully constructed that will allow the development of various information services to support the smart city application having heterogeneous data.

We have divided this chapter in two parts: in the first part we have presented an in-depth background information about our research area i.e. data integration. We have presented a state of the art literature review and the detailed methodologies adopted by these integration frameworks. We have also discussed the strengths and weaknesses of these data integration architectures. The literature review allows us to identify the limitations of the existing integration frameworks such as: update services and tools in Cloud-based integration, the interoperability and scalability issues in the Multi-Level architecture, the limitation of automated interpretation in the CityPulse framework, the speed factor of Online Data integration process framework, the necessity of a designer in the Domain-Specific automated ETL architecture, and the limitations of having the complete knowledge of data streams in the Automatically Generated Data Marts architecture. The papers reviewed in the first part have inspired us with the ideas and basis to develop our ELT framework.

In the second part of this chapter we have presented the state of the art research done on the Bike Sharing Schemes around the world, and discussed some of the papers supporting the novelty of the research work we have performed. The literature review performed on these research works enabled us to identify the strengths and limitations of the current work such as: the lack of measures suggesting modal shifting towards cycling in Dublin city, the inability of data identification in the NS2 simulator, the assumption of colour transitions as special events in the New York Citi Bike-Sharing, and the limitation of temporal clustering on a daily basis in the User Satisfaction based model. The papers reviewed in the second part helped us to perform resource allocation analysis on Dublin Bikes dataset to suggest a viable replenishing strategy, because of the broad area of our research project we have focused on the cycling dataset in chapter 4.

In the next chapter, we present in detail the methodology of our Extract, Load and Transform (ELT) architecture, the developed data models, and the algorithms designed during our research to provide structural enrichment and data semantic enrichment and to address the current limitations of the existing frameworks.

Chapter 3

Extract, Load and Transform (ELT) Framework

In the previous chapter we presented a literature review of the state of the art in the area and have compared the strengths and limitations of these existing frameworks. In this chapter we explain in detail the overall architecture of our developed Extract, Load and Transform (ELT) framework. We discuss the model life cycle, the automated schemata formation steps, the algorithms used to provide data streams with structural and contextual enrichment and providing meaning to the data. Also we define and present algorithms used to perform various data augmentation enrichment.
In the first section, we provide a brief introduction of Dublin transport along with its current challenges and our proposed methodology. In section two we describe all the transport infrastructures one by one that were used to build our data lake. In section three, we discuss about our ELT framework and its working.

3.1 Experimental Setup

In Appendix.A we provide detailed steps outlining how to install the required tools and libraries to run our Extract, Load and Transform (ELT) framework on a vanilla computer.

3.2 Methodology and Architecture

Dublin city has several modes of public transport which include Dublin Bikes Service, Dublin Bus Service, Dublin Luas Tram Service, Dublin Motorways and Dublin Air Quality Service. These public infrastructures are providing real-time information to citizens through different websites and mobile applications. These applications are stand alone and isolated which makes them useful just within their limited domain itself. Moreover, these data sources are publishing data in heterogeneous data formats such as JavaScript Object Notation (JSON), Extensible Markup Language (XML), Comma-Separated Values (CSV), Hypertext Markup Language (HTML) and raw text files. The data lake is a central repository to keep all the unstructured data from the individual transport stream in a single place for further processing. These individual schemata will be stored into our repository i.e. data lake. The isolated nature of these data domains presents a challenge to integrate them, which if addressed properly can open many doors of opportunities through cross-domain analysis and help produce new knowledge and insights. To address this challenge of integrating the various Dublin Smart City isolated data silos we have developed an automated and robust Extract, Load and Transform (ELT) framework that will facilitate the generation of the Canonical Data Model (CDM). The Canonical Data Model is a single large data model that is constructed by integrating the individual data models of each individual transport stream. The Canonical Data Model can evolve with the addition of new transport streams when required and further contextual enrichment can be performed.

In Figure 3.1, you will see the proposed architecture of our project including the ELT. The CDM will enable users to perform querying and analysis operations over a number of domains available in our Data Lake or staging area. The CDM will perform the integration operation by using the required dimensions and measures for integration, then it will pull down the data from schemata residing in our data lake.

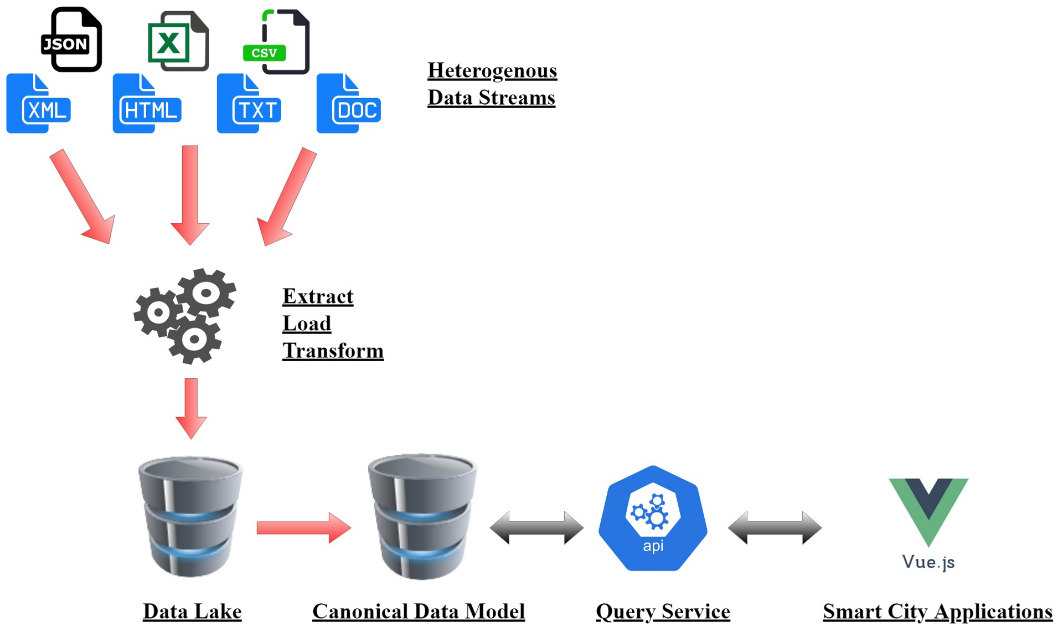


Figure 3.1 Project Architecture

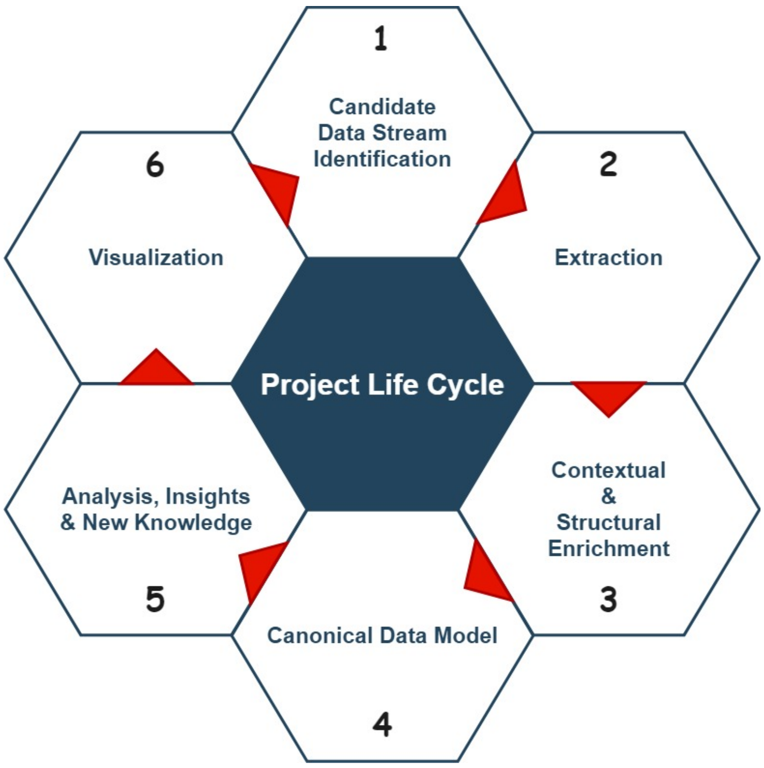

1
Candidate Data Stream Identification
2
Extraction
Contextual & Structural Enrichment
3
Canonical Data Model
4
Analysis, Insights & New Knowledge
5
6
Visualization
Project Life Cycle

Figure 3.2 Project Life Cycle

In Figure 3.2, we present our project life cycle which starts from the first phase the identification of candidate data streams and then proceeds towards the data extraction phase from those streams. The transformation phase consists of both structural as well as contextual enrichment towards data streams. In the structural enrichment we provide a schema or data model for the data to facilitate the querying of the data and in contextual enrichment we will add relevant and useful text to the available data from additional data sources in order to give a complete meaning to the data. We have performed contextual enrichment on almost all of our datasets which is explained in detail in their relevant sub-sections. In the fourth phase the data is ingested into our Canonical Data Model. In the fifth phase the new insights and knowledge will be obtained through data analysis. Lastly, this information will be presented in a compelling visual format. The Cycle will repeat itself from phase one if we wish to add additional useful datasets.

In Figure 3.3, you will see the overview of CDM architecture and how the schemata from the Data Lake will be ingested into the Canonical Data Model for further analysis.

In Figure 3.3 you can see how our data lake is containing all the automatically produced schemata at top level, their detailed formation is provided in section 3.3. The Canonical Data Model will pull down the data from these schemata residing in our Data Lake and will populate the tables in the CDM. In addition, our data lake will also have schemata

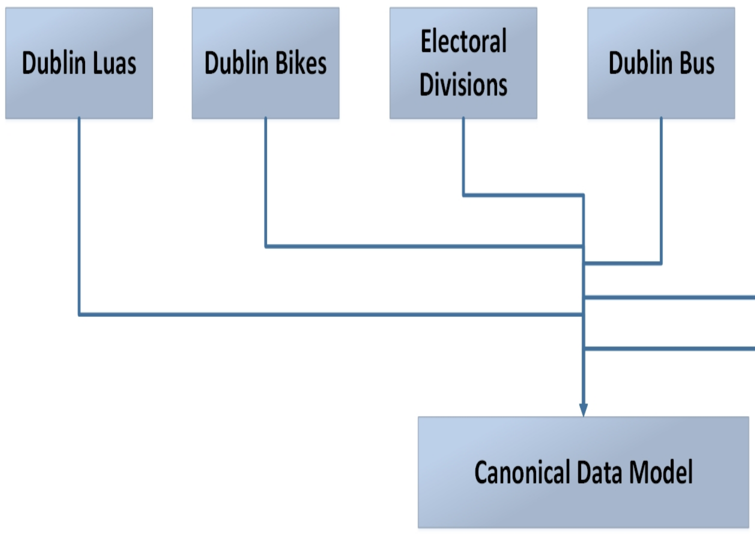


Figure 3.3 Canonical Data Model Architecture

which contains data pulled from various other data domains such as weather, air quality and electoral divisions (Census Data). The integration operations are performed using common measures and dimensions such as Geo-location, time, date and common identifiers.

3.3 Dublin Public Transport Infrastructures

Everyday the people of Dublin city use the public transport services to commute from their homes to their workplaces and later back home. The Dublin public transport infrastructure consists of a Bus Service, a Tramlines Service (Dublin Luas), the Dublin Bikes Service, the Dart (Railway Service) and the Motorways network. These modes of public transport are deployed in an interconnected manner that allow the commuters to reach from one destination to another where it may be difficult to reach by just using a single transport service. There are various mobile applications and websites available to users to keep track of public transport on any particular stops in real-time.

Several of the transport infrastructures providing information in real-time are mentioned below:

- Dublin Bus Infrastructure
- Dublin Bikes Infrastructure
- Dublin Luas Infrastructure
- Dublin Motorways Infrastructure

- Dublin Environment Infrastructure

3.3.1 Dublin Bus Infrastructure

Nearly half a million people are using Dublin Bus on a daily basis to commute between their home and workplaces or to reach anywhere within the bus routes network. The Dublin Bus network consists of more than one thousand buses (1010), almost one hundred and twenty (120) routes and nearly five thousand (5000) bus stops. These buses are covering their own specified routes every day and serve all the bus stops within their routes. Many of these stops contain a digital screen displaying the buses expected arrival time at the stop in real-time. The Dublin Bus Management Information System Team (MIS) also provides this real-time information through an open Application Programming Interface (API) in a JSON data format (SmartDublin, 2019b). The information of any bus stop can be retrieved by passing the bus stop number as an argument to Dublin Bus API which will return the number of buses arriving on that particular stop, their arrival time and so on.

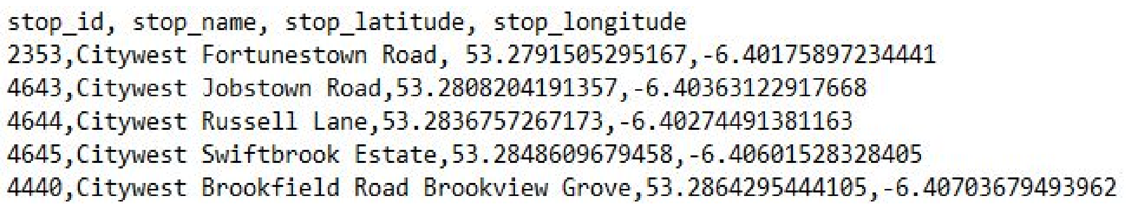
Automated Schema Formation of Dublin Bus

The Transport For Ireland (TFI) (Transport_For_Ireland, 2019b) provide several text files containing useful information about the Dublin Bus routes and all the bus stops including their numbers, names and the Geo-location coordinates of their location. These files will be analysed by our framework in-search of relevant and required textual or numeric information that will help in constructing an automated schema for the Dublin Bus.

The schema building process is divided into two parts: the first part will analyse the provided textual files to search the required information and data to define the make up the table attributes along with appropriate data types. In the second part the Dublin Bus API real-time data will be written to a JSON file and this file will be analysed to find the remaining attributes that was not found in the first part of the process and is necessary to complete the schema build up process.

Algorithm 3.1 Input Data File

In Figure 3.4 we present below some sample data of the Dublin Bus stops taken from the bus_stops.txt file which is provided by the Transport for Ireland. The file contains a header row showing the name of the column attributes and the remaining rows shows the actual data of all bus stops.



```
stop_id, stop_name, stop_latitude, stop_longitude
2353,Citywest Fortunestown Road, 53.2791505295167,-6.40175897234441
4643,Citywest Jobstown Road,53.2808204191357,-6.40363122917668
4644,Citywest Russell Lane,53.2836757267173,-6.40274491381163
4645,Citywest Swiftbrook Estate,53.2848609679458,-6.40601528328405
4440,Citywest Brookfield Road Brookview Grove,53.2864295444105,-6.40703679493962
```

Figure 3.4 Bus_Stop.txt file

The purpose of algorithm 3.1 is to analyse the bus_stops.txt file, which contains all bus stops data covering all the Dublin Bus transport network, to identify the columns required to construct the Bus Stop table and to then create that table in the database. The algorithm reads the data file and looks for the required attributes in order to declare variables with appropriate data types that will be passed to a Structured Query Language (SQL) query detailed in appendix D.1 to construct the Bus_Stops table in the schema. We omit low-level error-checking implementation details from the algorithm.

- ***(lines 1-6)*** The algorithm reads the header row from the bus_stops.txt file and assigns it to a variable called line. An if statement checks the availability of the text stop_id in the line variable. If the condition is true the required text is assigned to the variable Bus_Stop_ID and the appropriate SQL data type is set.

Algorithm 3.1: `Create_Table_Bus_Stops`

```
/* This algorithm analyses the Bus Stops file to identify the columns required to construct
   the Bus Stop table and then creates the table in the database.                        */
input  : bus_stops.txt - the file contains the required bus stop column names and data.
output : The Bus Stop table is created in the database.
1  begin
2      line ⟵ read the first (header) line of bus_stops.txt file;
3      if (stop_id in line) then
4          Bus_Stop_ID_Name ⟵ stop_id;
5          Bus_Stop_ID_Type ⟵ Numeric;
6      end
7      if (stop_name in line) then
8          Bus_Stop_Name_Name ⟵ stop_name;
9          Bus_Stop_Name_Type ⟵ Varchar;
10     end
11     if (stop_latitude in line) then
12         Bus_Stop_latitude_Name ⟵ stop_latitude;
13         Bus_Stop_latitude_Type ⟵ Numeric;
14     end
15     if (stop_longitude in line) then
16         Bus_Stop_longitude_Name ⟵ stop_longitude;
17         Bus_Stop_longitude_Type ⟵ Numeric;
18     end
19     Create Bus_Stop_Table (Bus_Stop_ID_Name, Bus_Stop_ID_Type,
20                           Bus_Stop_Name_Name, Bus_Stop_Name_Type,
21                           Bus_Stop_latitude_Name, Bus_Stop_latitude_Type,
22                           Bus_Stop_longitude_Name, Bus_Stop_longitude_Type);
23 end
```

- ***(lines 7-18)*** In a similar manner, three more conditionals are checked to verify the availability of the required column names and if present, to assign the column names and set the appropriate respective SQL data types.
- ***(lines 19-23)*** Finally, these variables are passed as parameters to a parameterised SQL query that will generate the Bus_Stops table as part of the Dublin Bus schema in the PostgreSQL database.

Dublin Bus Schema Architecture

In Figure 3.5 we present the Dublin Bus schema formed after an analysis of all required data sources i.e. the required text files. An almost identical algorithm to algorithm 3.1 is used to create the Bus_Direction table, Bus_Routes table, Bus_Routes_Direction table, Bus_Stops_Direction table, Bus_Stops_Distance table and Bus_Times table. This schema contains seven tables. We now outline the contents and purpose of each table.

1. The purpose of the **Bus_Direction** table is to hold the direction_name i.e. inbound or outbound with a unique direction_id.
2. The purpose of the **Bus_Routes** table is to store a unique route_id, the route departure location name and the destination location name.
3. The **Bus_Routes_Direction** table contains a unique routes_direction_id as a result from combining the unique direction_id from Bus_Direction table with the unique route_id from Bus_Routes table.
4. The purpose of the **Bus_Stops** table is to store the information about each bus station. This table contains the original unique Bus_stop_id, bus_stop_name and the GPS coordinates of bus station.
5. The **Bus_Stops_Direction** table contains the combined data from bus_stops table with the unique routes_direction_id taken from Bus_Routes_Direction table, this table shows the total stops and direction of these stops of particular bus route.
6. The **Bus_Stops_Distance** table contains the distance and the estimated travel minutes from one stop to another within a specific bus route calculated by using the Google Distance Matrix API. This table can be used to compare the delay at every bus stop.
7. The purpose of **Bus_Times** table is to store the real-time information retrieved from the Dublin Bus API. This table contains the Bus_Stop_ID, the Bus_Stop_Name,

the Arrival_Date_Time, the Arrival_Due_Time, the Departure_Date_Time, Departure_Due_Time, Scheduled_Arrival_Date_Time, Scheduled_Departure_Date_Time, Destination, Origin and Source_Timestamp. All the referential integrity constraints (primary key and foreign key rules) were enforced during the formation phase.

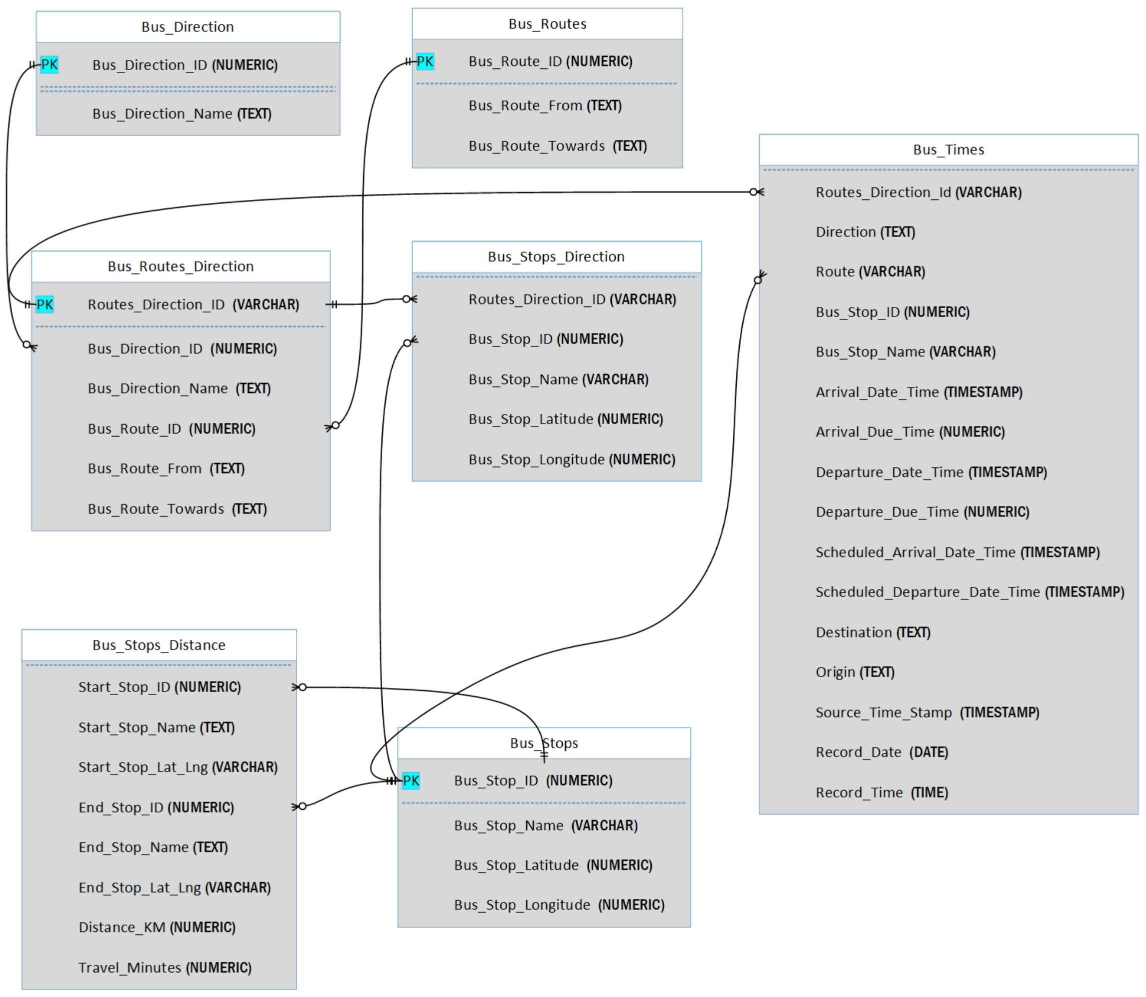

Bus_Direction
PK Bus_Direction_ID (NUMERIC)
Bus_Direction_Name (TEXT)
Bus_Routes
PK Bus_Route_ID (NUMERIC)
Bus_Route_From (TEXT)
Bus_Route_Towards (TEXT)
Bus_Times
Routes_Direction_Id (VARCHAR)
Direction (TEXT)
Route (VARCHAR)
Bus_Stop_ID (NUMERIC)
Bus_Stop_Name (VARCHAR)
Arrival_Date_Time (TIMESTAMP)
Arrival_Due_Time (NUMERIC)
Departure_Date_Time (TIMESTAMP)
Departure_Due_Time (NUMERIC)
Scheduled_Arrival_Date_Time (TIMESTAMP)
Scheduled_Departure_Date_Time (TIMESTAMP)
Destination (TEXT)
Origin (TEXT)
Source_Time_Stamp (TIMESTAMP)
Record_Date (DATE)
Record_Time (TIME)
Bus_Routes_Direction
PK Routes_Direction_ID (VARCHAR)
Bus_Direction_ID (NUMERIC)
Bus_Direction_Name (TEXT)
Bus_Route_ID (NUMERIC)
Bus_Route_From (TEXT)
Bus_Route_Towards (TEXT)
Bus_Stops_Direction
Routes_Direction_ID (VARCHAR)
Bus_Stop_ID (NUMERIC)
Bus_Stop_Name (VARCHAR)
Bus_Stop_Latitude (NUMERIC)
Bus_Stop_Longitude (NUMERIC)
Bus_Stops_Distance
Start_Stop_ID (NUMERIC)
Start_Stop_Name (TEXT)
Start_Stop_Lat_Lng (VARCHAR)
End_Stop_ID (NUMERIC)
End_Stop_Name (TEXT)
End_Stop_Lat_Lng (VARCHAR)
Distance_KM (NUMERIC)
Travel_Minutes (NUMERIC)
Bus_Stops
PK Bus_Stop_ID (NUMERIC)
Bus_Stop_Name (VARCHAR)
Bus_Stop_Latitude (NUMERIC)
Bus_Stop_Longitude (NUMERIC)

Figure 3.5 Dublin Bus Schema

The purpose of algorithm 3.2 is to analyse the bus_stops.txt file containing all the bus stops data and perform a cleansing operation on the data by removing unwanted special characters present in the data, and then finally to insert the clean data into Bus_Stops table using the SQL query detailed in appendix D.2.

- ***(lines 1-5)*** An empty list called bus_stops_list is initialised that will store the record associated with single bus stop. The algorithm reads the bus_stops file, skipping the first (header) line, and assign it to a variable called File_Data. Then, a for loop iterates

Algorithm 3.2: `Populate_Table_Bus_Stops`

```
   /* This algorithm reads the Bus Stops file and populates its data into the Bus_Stops table
      created by algorithm 3.1.                                                              */
   input  : bus_stops.txt - the file contains all the bus stops data.
   output : The bus stops data is inserted into Bus_Stops table.
1  begin
2      bus_stops_list ⟵ empty list;
3      File_Data ⟵ read all lines in the file except the first (header) line ;
4      for line in File_Data do
5          single_bus_stop ⟵ empty list;
6          for stop in line.strip(newline).split(,) do
7              single_bus_stop.append(stop);
8          end
9          bus_stops_list.append(single_bus_stop);
10         Insert into Bus_Stops_Table Values(single_bus_stop);
11     end
12 end
```

through the bus stops data by using another variable called single_line which will retrieve one data row at a time. An empty list will be declared called single_bus_stop.

- ***(lines 6-8)*** A nested for loop iterates through the single_line data retrieved by the outer loop and cleans the data from unwanted special characters. After cleaning of data it will append to the single_bus_stop list. Recall that each single_line contains the bus stop ID, the bus stop name and the bus stop GPS latitude and longitude coordinates.
- ***(lines 9-12)*** The single_bus_stop data will be appended to bus_stops_list forming a two dimensional array. The clean data will be entered into Bus_Stops_Table by using the parameterised SQL query, with the termination of the outer for loop the procedure will terminate.

Automated Contextual Enrichment of Dublin Bus

Before proceeding with an exposition of the contextual enrichment stage, it is important to clarify several concepts. A **path** is a link (edge) joining two adjacent bus stops (vertices). A bus route consists of all the paths from the starting bus stop to the destination bus stop. We designed algorithm 3.3 to perform the contextual enrichment by automatically creating the paths for any given bus route. Then, we designed an algorithm 3.4 to calculate the distance for all the created paths in kilometers and the estimated travelling time in minutes using the Google Distance Matrix API. The actual API call and sample output result of Google Distance Matrix for Dublin Bus stops is mention in appendix E.9. This contextually enriched data will be inserted into the `Bus_Stops_Distance_Table` by using an almost identical approach used to insert data into the `Bus_Stops_Table`.

Bus_Stops_Direction Table Data Source

We now outline the contextual enrichment performed by algorithm 3.3. We create a new identifier called the routes_direction_id. The purpose of the routes_direction_id is to uniquely identify a bus stop located on either an inbound or outbound bus route. The routes_direction_id is then combined with the bus_stops information and stored in a file called bus_stops_direction.txt. The first five digits of the route_direction_id indicate whether the bus stop is on the inbound or outbound direction of the bus route. For example the route_direction_id `10001` indicates an inbound bus route whereas the route_direction_id `10002` indicates an outbound bus route. The last two/three digits of the route_direction_id indicate the bus number serving that route. Recall that Dublin Bus operates 1010 buses across almost 120 radial, cross-city and peripheral routes in the city of Dublin and the greater Dublin area. Many bus stops participate in multiple bus routes. For this project, we have selected a sample of two bus routes: bus route no.27 and bus route no.49. The reason for choosing these two particular routes is their passage by TU Dublin Tallaght Campus and through Dublin City Centre. The order of the inbound and outbound bus stops of route numbers 27 and 49 can be obtained from the Dublin Bus Mobile Real-Time Passenger Information (RTPI) API web service (Dublin_Bus_RTPI, 2020), and the bus stops were arranged in that order in the bus_stops_direction.txt file.

In Figure 3.6 we present sample data from the file bus_stops_direction.txt containing a header row showing the name of the column attributes and the remaining rows shows the actual data of bus stops.

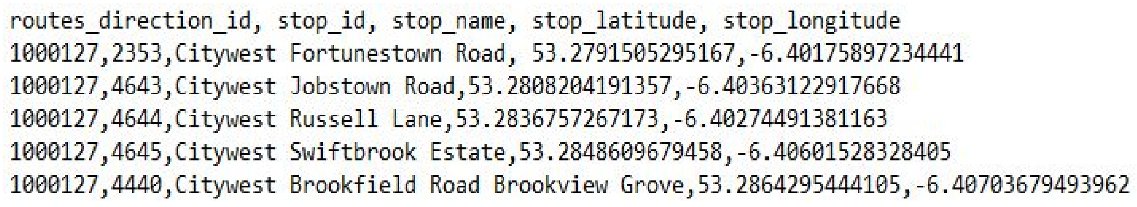

```
routes_direction_id, stop_id, stop_name, stop_latitude, stop_longitude
1000127,2353,Citywest Fortunestown Road, 53.2791505295167,-6.40175897234441
1000127,4643,Citywest Jobstown Road,53.2808204191357,-6.40363122917668
1000127,4644,Citywest Russell Lane,53.2836757267173,-6.40274491381163
1000127,4645,Citywest Swiftbrook Estate,53.2848609679458,-6.40601528328405
1000127,4440,Citywest Brookfield Road Brookview Grove,53.2864295444105,-6.40703679493962
```

Figure 3.6 Bus_Stops_direction.txt file

To construct the bus_stops_direction table, we employ an almost identical algorithm to algorithm 3.1 and then to populate the data into the table Bus_Stops_Direction using an almost identical algorithm to algorithm 3.2.

The purpose of algorithm 3.3 is to create a list of all paths that reside on a specific inbound or outbound bus route. The routes_direction_id specifies which stops are on any given bus

Algorithm 3.3: `Bus_Paths_Formation`

```
/* This algorithm creates the bus paths for the given bus route.                 */
input  : routes_direction_id - the unique bus routes_direction_id.
output : paths_list - a list of paths (two adjacent bus stops) for the given bus route.
begin
    stops_direction_list ⟵ retrieve all the stops data from the bus_stops_direction table using the unique
      routes_direction_id.
    length ⟵ compute the length/size of stops_direction_list
    path ⟵ empty list;
    paths_list ⟵ empty list;
    for (row_index, row_data in stops_direction_list) do
        if $(row\_index < (length - 1))$ then
            path ⟵ row_data + stops_direction_list(row_index + 1);
            paths_list.append(path);
        end
    end
end
```

route. The algorithm 3.3 retrieves the data from the bus_stops_direction table and uses the unique routes_direction_id to create the bus paths. The algorithm then reads the list and performs the paths formation operation on data by concatenating the current bus stop with the next nearest one in the route and stores it in paths_list.

- ***(lines 1-5)*** The stops_direction list is created and it contains the data retrieved from the bus_stops_direction table using the routes direction id. The length or size of the list is calculated and stored in the length variable. A path list is initialised that will store hold a single path at a time. Then a paths_list is initialised that will contain all the created paths.

- ***(lines 6-8)*** We use a for loop to enumerate through the stops_direction_list data by fetching one row at a time and storing it in the row_data variable and then storing the loop index value in the row_index variable. A condition will check whether the current row_index is less than the total list size. Then the current stop data is concatenated with the next stop and assign this to the paths_list.

- ***(lines 9-12)*** The path list is appended to paths_list forming a two dimensional list. When the loop is terminated the procedure will also terminate.

The purpose of algorithm 3.4 is to calculate the actual distance of each bus path generated by algorithm 3.3 and then to compute the estimated travel duration of each path in minutes. The algorithm 3.4 receives the paths_list and uses the starting and ending Global Positioning System (GPS) coordinates of each path in the list. These GPS coordinates are passed to the Google Distance Matrix API to compute the distance and the estimated travel time from one stop to another (path). We have mentioned the actual API call and sample output result of

Algorithm 3.4: `Calculate_Paths_Distance`

```
/* The paths_list from algorithm 3.3 is analysed to calculate the paths distance using the
   Google Distance Matrix API.                                                            */
input  : paths_list - the paths_list constructed by algorithm 3.3.
output : The calculated distance data of paths is inserted into Bus_Stops_Distance table.
1  begin
2      for path in paths_list do
3          Distance_Matrix ⟵ compute_distance(path.start_GPS_point, path.end_GPS_point);
4          distance_list ⟵ empty list;
5          for elements in Distance_Matrix do
6              for values in elements do
7                  travel_time ⟵ values.duration;
8                  distance_km ⟵ values.distance;
9                  meter ⟵ "m";
10                 distance_unit ⟵ distance_km.split('m');
11                 if (distance_unit == meter) then
12                     distance_km ⟵ values.distance/1000;
13                 else
14                     distance_km ⟵ values.distance
15                 end
16                 distance_list ⟵ path + distance_km + travel_time;
17                 Insert into Bus_Stops_Distance Values(Start_Stop_ID, Start_Stop_Name,
18                                    Start_Stop_Lat_Lng, End_Stop_ID, End_Stop_Name,
19                                    End_Stop_Lat_Lng, Distance_km, Travel_Minutes)
20             end
21         end
22     end
23 end
```

Google Distance Matrix for Dublin Bus stops in appendix E.9. Finally, this information is passed to the SQL query detailed in appendix D.3 to insert the data into Bus_Stops_Distance table.

- ***(lines 1-4)*** A loop enumerates through the paths_list and retrieve a single path at a time. The path GPS coordinates are passed as an input to compute the distance and estimated time using the Google Distance Matrix API then this information is stored in a Distance_Matrix variable. An empty list is initialised called distance_list to store the single path calculate distance and time.

- ***(lines 5-8)*** Then a for loop iterates through the information in the Distance_Matrix variable by using another variable called elements to retrieve the data from a granular structure. Then a nested for loop iterates through the variable called elements to retrieve one row at a time and to store it in another variable called values. The information of calculated time and estimated distance is assign to separate variables called travel_time and distance_km.

- ***(lines 9-14)*** A string character called 'm' is assign to a variable called the meter. The distance_km variable is split to store the unit of distance computed by API. Then meter variable is compared to the distance_unit variable to check if distance is in meters. If so, it will convert it to kilometers otherwise original value will be assigned as it is.

- ***(lines 15-23)*** The original path data along with the calculated distance and time is assigned to a distance_list and finally a parameterised SQL query performs the insertion operation in the Bus_Stops_Distance table.

The purpose of the algorithm 3.5 is to extract the real-time arrival information of any bus arriving at a specified Bus stop on a specified bus route using the Dublin Bus Official API. The algorithm 3.5 extracts the bus stop real-time information of the given stop number by using the unique stop_id. Then the information is filtered to only store the required buses on bus route_name such as: route 27 or 49.

- ***(lines 1-5)*** The stop_id is passed to the Dublin Bus API to retrieve the data of that stop, then the information is stored in a variable called realtime_data. Three empty lists are initialised called the bus_time to store the data associated with single bus stop, the bus_times_list contains the data of all bus stops and the stop_data is used store the bus stop details i.e stop id and stop name.

- ***(lines 6-8)*** Then a for loop iterates through the information stored in the variable called realtime_data, another variable row is used to retrieve a single row of data at

Algorithm 3.5: `Extract_Bus_Times`

```
/* The bus stop real-time information is extracted through Dublin Bus official API.   */
input  : stop_id, route_name - the unique bus stop_id and the selected bus route_name.
output : bus_times_list - The bus_times list containing all the particular route buses arriving at that stop in real-time.
1  begin
2      realtime_data ⟵ bus_API(stop_id);
3      bus_time ⟵ empty list;
4      bus_times_list ⟵ empty list;
5      stop_data ⟵ empty list;
6      for row in realtime_data do
7          stop_data ⟵ retrieve stop details from Bus_Stops table using the unique stop_id;
8          if (row[route] == route_name) then
9              if (row[arrivaltime] == 'due') then
10                 arrivaltime ⟵ 0;
11             else
12                 arrivaltime ⟵ row[arrivaltime];
13             end
14             if (row[departuretime] == 'due') then
15                 departuretime ⟵ 0;
16             else
17                 departuretime ⟵ row[departuretime];
18             end
19             bus_time ⟵ stop_data + realtime_data + arrivaltime + departuretime;
20             bus_times_list.append(bus_time);
21         end
22     end
23 end
```

a time from realtime_data. The stop details of that particular bus stop (Name, GPS coordinates) will be retrieved using the SQL query from bus_stops table using the unique stop_id, the actual SQL query is mention in appendix D.4. A condition will ensure to extract the required bus route data among all the different route buses arriving at that stop.

- ***(lines 9-18)*** In a similar manner, two more conditionals are checked to convert the arrival time and departure time to zero if received a arrival time is 'due', otherwise the original numeric value is stored.
- ***(lines 19-23)*** The stop_data along with the realtime_data is assigned to the bus_time and is appended to the bus_times_list making a two dimensional list. The termination of the loop will terminate the procedure as well.

3.3.2 Dublin Bikes Infrastructure

There have been over 3.8 million journeys made using the Dublin Bikes service in 2019. Thus, on average, there were over 10,400 journey each day of the year (Just_Eat, 2019). There are several reasons for the success of this service including affordability but foremost is being a source for most of the people to become or stay healthy through cycling while having their busy work routine. The Dublin Bikes network consists of one hundred and eleven (111) bike stations which are spread all over Dublin city. The maximum number of bike stands available at a bike station is 40 stands and the minimum bike stands available is 15 stands. On average, a typical bike station contains between thirty to forty (30-40) bike stands. All the stations allow payment options, some stations only allows credit card payment, some stations only allows the direct debit card payment and some stations allow the both payment options (Just_Eat, 2018). Dublin Bikes operate seven days a week between the hours of 5:00am and 12:30am. A bike may be returned 24 hours a day. Dublin Bikes is a self-service bike rental system open to everyone from 14 years of age (Dublin_City_Council, 2019). The Management Information System team of Dublin Bikes provides real-time information of bike stations through an open API in JSON data format (SmartDublin, 2019a). The information of all the bike stations can be retrieved by using your personal API key which is given after registering on the official website (JCDecaux, 2019).

The Dublin Bikes API provides the following data associated with each bike station: the unique bike station number, the unique bike station name, the station GPS coordinates, the total number of bike stands at the bike station, the total number of empty bike stands at the current time, the total number of available bicycles at the current time, and the record capture timestamp.

Automated Schema Formation of Dublin Bikes

The Transport For Ireland (TFI) (Transport_For_Ireland, 2019b) provides various text files containing useful information about the Dublin Bike stations such as the bike station numbers, the bike station names and their Geo-location coordinates. This file is analysed by our framework in-search of relevant and required textual or numeric information that will help in constructing an automated schema for the Dublin Bikes. We use the algorithm E.1 to construct the Bike_Stations table.

Dublin Bikes Schema Architecture

In Figure 3.7 we present the Dublin Bikes schema generated after the analysis of the required data sources i.e. the bike_station.txt text file and the official Dublin Bikes API. The algorithm in appendix E.1 is applied to create the Bike_Stations table. An almost identical algorithm is used to create the Bikes_Data table and the Bikes_Usage table. This schema contains three tables. We now outline the purpose and contents of each table.

1. The purpose of the **Bike_Stations** table is to store the information about each bike station. The **Bike_Stations** table contains a unique station_id, station_name and Geo-location coordinates of the bike station.

2. The **Bikes_Data** table contains the data about all bike stations. This information is retrieved via the Dublin Bikes API. Specifically, the data retrieved is: the bike station number, the bike station name, the GPS coordinates, banking "is credit card payment option available on that stop?" (Just_Eat, 2018), the total_stands (bike stands), available_stands (available bike stands), available_bikes, status (the status shows the state of the station open or closed) and last_update (the update time of this information).

3. The purpose of the **Bikes_Usage** table is to store the usage information about each bike station. The current data from the bike station will be compared with the data retrieved and stored five minutes earlier from the bike station to determine the changes that occurred at each station such as: the number of bikes withdrawn in the last five minutes, the number of bikes lodged in the last five minutes, the number of stands emptied, and the number of stands filled. The calculations are performed using the data stored in the **Bikes_Data** table. An appropriate textual message will also be generated according to the actions performed on each bike station such as: Bike withdrawn, Bike returned, Bikes are same, Stand emptied, Stand filled and Stands are same. All the referential integrity constraints (primary key and foreign key rules) were enforced during the formation phase.

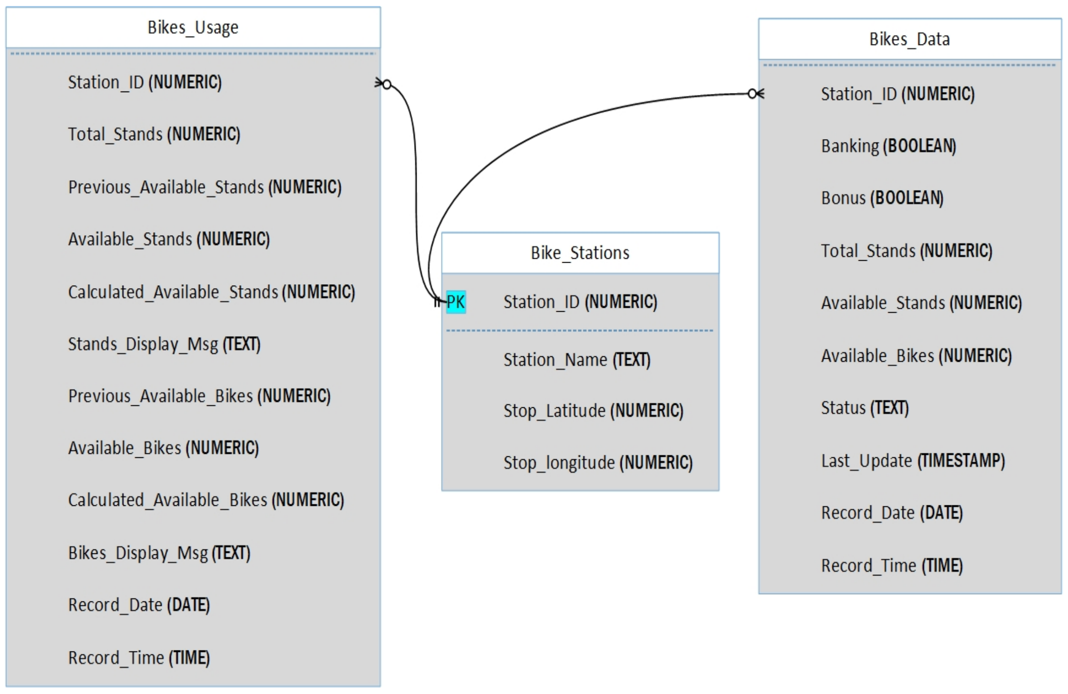


Figure 3.7 Dublin Bikes Schema

To populate the data in the tables of Dublin Bikes Schema using appropriate mappings we have adopted an approach mention in appendix in appendix E.2 that is used to populate the Bike_Stations table.

Automated Contextual Enrichment of Dublin Bikes

The layout of the Dublin Bike stations may be viewed as nodes on a graph. In a graph with N nodes, there are N-1 directed paths from a node to every other node. Thus there are N*(N-1) paths in total. Given that N = 111 (bike stations), there are 12,210 total paths possible in the Dublin Bikes network. However, the concept of path(s) was not applied on Dublin Bikes because although the concept of a path is valid in Dublin Bikes, there are no unique IDs assigned to each bicycle, thus there is no way to identify an individual bike journey from one specific bike station to another.

We have designed an algorithm 3.6 to perform the contextual enrichment by extracting the Bike stations information via Dublin Bikes API. The algorithm retrieves the Bike stations information via Dublin Bikes API and performs the required data cleaning and then populates it into the **Bikes_Data** table using SQL Query mention in appendix D.5.

The purpose of the algorithm 3.6 is to extract the real-time information of all the Dublin Bike stations provided by the Dublin Bikes Official API. The algorithm pulls all the Bike

Algorithm 3.6: `Extract_Bikes_Data`

```
   /* The bikes real-time information is extracted via the Dublin Bikes official API.   */
   input  : bikes_realtime_list - the real-time data from all bikes stations retrieved via the Dublin Bikes API.
   output : The bikes data is inserted into the Bikes_Data table.
1  begin
2      bikes_realtime_list ⟵ bikes_API(API_Key);
3      single_station_data ⟵ empty list;
4      bike_station_data ⟵ empty list;
5      bike_stations_dataset ⟵ empty list;
6      for (station in bikes_realtime_list) do
7          bike_station_data ⟵ retrieve stop details from Bike_Stations table using the unique bike_station['number'];
8          api_timestamp ⟵ station['last_update'];
9          api_converted_time ⟵ datetime.fromtimestamp(api_timestamp / 1e3); /* 1e3 mean 10 to the power of 3 */
10         transformed_time ⟵ api_converted_time(Hours:Minutes:Seconds);
11         single_station_data ⟵ bike_station_data + station + transformed_time;
12         bike_stations_dataset.append(single_station_data);
13         Insert into Bikes_Data Values(single_station_data);
14     end
15 end
```

stations real-time information at once. The information retrieved contains the number of bike stands available, the total number of bike stands at the station and the number of bikes available. Then after processing, this information will be passed to the SQL query in appendix D.5 to insert the data into Bikes_Data table.

- ***(lines 1-5)*** The bikes_API retrieves all the Bike stations data at once using the user API_Key and this information is stored in bikes_realtime_list. Three empty lists are initialised: single_station_data stores the record of one bike station at a time, bike_station_data stores the details of a bike station i.e. station id and station name retrieve from Bike_Stations table, and bike_stations_dataset stores the data associated with all the Bike stations.

- ***(lines 6-7)*** For each bike station in the Dublin Bikes Network (recall there are 110 bike stations) we used a loop variable called station to iterate through bikes_realtime_list to retrieve a single bike station data at a time. Then the details such as: bike station id and station name of each particular bike station are retrieved from the Bike_Stations table using SQL query mention in appendix D.5.1 by passing the station_id.

- ***(lines 8-10)*** The last update time from API is assigned to a variable called api_timestamp. This numerical value is converted to a human understandable timestamp format and is store in api_converted_time variable. The required time format is extracted from api_converted_time variable and is store in transformed_time variable.

- ***(lines 11-15)*** The bike_station_data (recall from Bike_Stations table) along with the real-time bike_station_data (recall number of available bikes and stands) and the transformed_time is assigned to single_station_data list. The single_station_data which is one dimensional array is then appended to the bike_stations_dataset forming a two dimensional array. Then the single_station_data list is insert into Bikes_Data table using a SQL query D.5. Finally, the termination of loop will terminate the procedure.

Automated Augmentation Enrichment of Dublin Bikes

The purpose of augmentation enrichment is to generate new useful data that is derived by performing additional computation on existing data. In this instance, we will augment our Dublin Bikes dataset by computing the number of bikes withdrawn and the number of bikes returned within the last five minutes. We have developed algorithm 3.7 to determine the number of bikes being used or returned within the last five minutes at each bike station. We have set the time difference of five minutes initially but this can be change according to the user requirements. Algorithm 3.7 calculate the number of bikes and bike stands being used by comparing the last five minutes record with the current data. The algorithm uses the data from Bikes_Data table to extract the previous and current information of every single bike station using its unique station_id. Then this calculated information is pass to a SQL query mentioned in appendix D.6 to insert the data in the Bikes_Usage table. Also, the algorithm will generate a text message according to the changes that has been occurred at each bike station during the last five minutes such as: Bike withdrawn, Bike returned, Bikes are same, Stand emptied, Stand filled and Stands are same.

- ***(lines 1-7)*** Three empty lists are initialised called the bikes_usage_list to store the bike usage information associated with each bike station, the previous_bikes list to store the previous five minutes data associated with a single bike station and similarly, the current_bikes list to store the current data associated with single bike station. Then a for loop will iterate through the bike_stations_dataset which is generated by algorithm 3.6. The previous and the current bike records will be retrieved from Bikes_Data table with the help of parameterised queries using the unique station_id and the retrieved results is store in the appropriate lists.
- ***(lines 8-19)*** A conditional checks that the previous number of bikes are greater than the current available bikes. Recall that the previous_bikes refers to the data captured five minutes ago. If the previous bikes (number of bikes) is greater than the current bikes, then the bikes are withdrawn. Thus, the withdrawn bikes are calculated and an appropriate message is generated. Similarly, the second condition checks that previous

Algorithm 3.7: `Calculate_Bikes_Usage`

```
   /* This algorithm calculates the number of bikes used or returned in the last five minutes for
      each bike station.                                                                      */
   input  : bike_stations_dataset - the bike_stations_dataset created by algorithm 3.6
   output : The bikes_usage data is inserted into Bikes_Usage table.
1  begin
2      bikes_usage_list ⟵ empty list;
3      previous_bikes ⟵ empty list;
4      current_bikes ⟵ empty list;
5      for row in bike_stations_dataset do
6          previous_bikes ⟵ retrieve the last five minutes record from Bikes_Data Table using the stop_id;
7          current_bikes ⟵ retrieve the current record from Bikes_Data Table using the stop_id;
8          if (previous_bikes[no.bikes] > current_bikes[no.bikes]) then
9              Calculated_Bikes ⟵ previous_bikes[no.bikes] - current_bikes[no.bikes];
10             Display_Message ⟵ "Bike_Withdrawn";
11         end
12         if (previous_bikes[no.bikes] < current_bikes[no.bikes]) then
13             Calculated_Bikes ⟵ current_bikes[no.bikes] - previous_bikes[no.bikes];
14             Display_Message ⟵ "Bike_Returned";
15         end
16         if (previous_bikes[no.bikes] == current_bikes[no.bikes]) then
17             Calculated_Bikes ⟵ current_bikes[no.bikes];
18             Display_Message ⟵ "Bikes_are_Same";
19         end
20         if (previous_bikes[no.stands] < current_bikes[no.stands]) then
21             Calculated_Stands ⟵ current_bikes[no.stands] - previous_bikes[no.stands];
22             Display_Message ⟵ "Stand_Emptied";
23         end
24         if (previous_bikes[no.stands] > current_bikes[no.stands]) then
25             Calculated_stands ⟵ previous_bikes[no.stands] - current_bikes[no.stands];
26             Display_Message ⟵ "Stand_Filled";
27         end
28         if (previous_bikes[no.stands] == current_bikes[no.stands]) then
29             Calculated_Stands ⟵ current_bikes[no.stands];
30             Display_Message ⟵ "stands_are_Same";
31         end
32         bikes_usage_list ⟵ previous_bikes + current_bikes + Calculated_Bikes + Calculated_Stands +
             Display_Message;
33         Insert into Bikes_Usage Values(bikes_usage_list);
34     end
35 end
```

bikes are less than the current bikes and the third condition will check for the equality scenario. If true, the appropriate calculation and message are set.

- ***(lines 20-31)*** A conditional checks for the number of previous bike stands are less than the current number of bike stands which means bikes are returned and stands were filled. Thus, the emptied stands are calculated and an appropriate message is generated. Similarly, the second condition checks that previous number of stands are greater less than the current stands and the third condition will check for the equality scenario. If true, the appropriate calculation and message are set.

- ***(lines 32-35)*** The previous and current records along with the calculated results and display message will be combined into bikes_usage_list. Then this list will be insert into Bikes_Usage table with the help of a SQL Query. Finally, when the loop will terminate the procedure will terminate as well.

3.3.3 Dublin Luas Infrastructure

The number of passenger journeys made on Dublin Luas was almost 42 million in the year 2018. Thus we can say that on average, over one hundred and ten thousand people travel on Dublin Luas (the Tram/light rail network) each day within Dublin city from one place to another or to their workplaces (National_Transport_Authority, 2019). Dublin Luas service consists of two tram lines: the Luas Green line and the Luas Red line which provide both inbound and outbound travel facilities. The Luas Red line is 20 km in length and has 32 stops. It runs from Tallaght to The Point and from Saggart to Connolly. The Luas Green line 24.5 km in length and has 35 stops. It runs from Brides Glen to Broombridge via the City Centre (Transport_For_Ireland, 2019a). Every stop has a digital display screen showing the arrival time and destination of the arriving Luas. All stops have payment options through leap cards, bank card or cash, some stops contain parking facilities as well, and some are near to the Dublin Bike stations. The Information Management Team of Dublin Luas provides real-time information of every Luas stop through an open API in XML data format (Dublin_Luas, 2020a), the API gives information about the estimated arrival times of Luas at any given stop. In Figure 3.8 we present the map of Dublin Luas (Transit_Maps, 2017).

Automated Schema Formation of Dublin Luas

The TFI (Transport_For_Ireland, 2019b) provide several text files that contains the useful information and data about the Dublin Luas stops including unique stop_id, stop_name, Geo-location coordinate, and stop abbreviation which is used to call the API. This file is analysed by our framework in-search of relevant and required textual or numeric information that will enable the process of automated schema generation for Dublin Luas. To construct the Luas_Stops table we have used an almost identical algorithm to that outlined in algorithm 3.1.The actual algorithm used to build Luas_stops table is mentioned in appendix E.3.

Dublin Luas Schema Architecture

In Figure 3.9 we present the Dublin Luas schema generated after the analysis of the required data sources i.e. the luas_stops.txt text file and the Dublin Luas official API. An almost identical algorithm to algorithm E.3 is used to construct the Luas_Line table, Luas_Direction table, Luas_Location table, Luas_Line_Direction table, Luas_Stops_Distance table, Luas_Times ta-

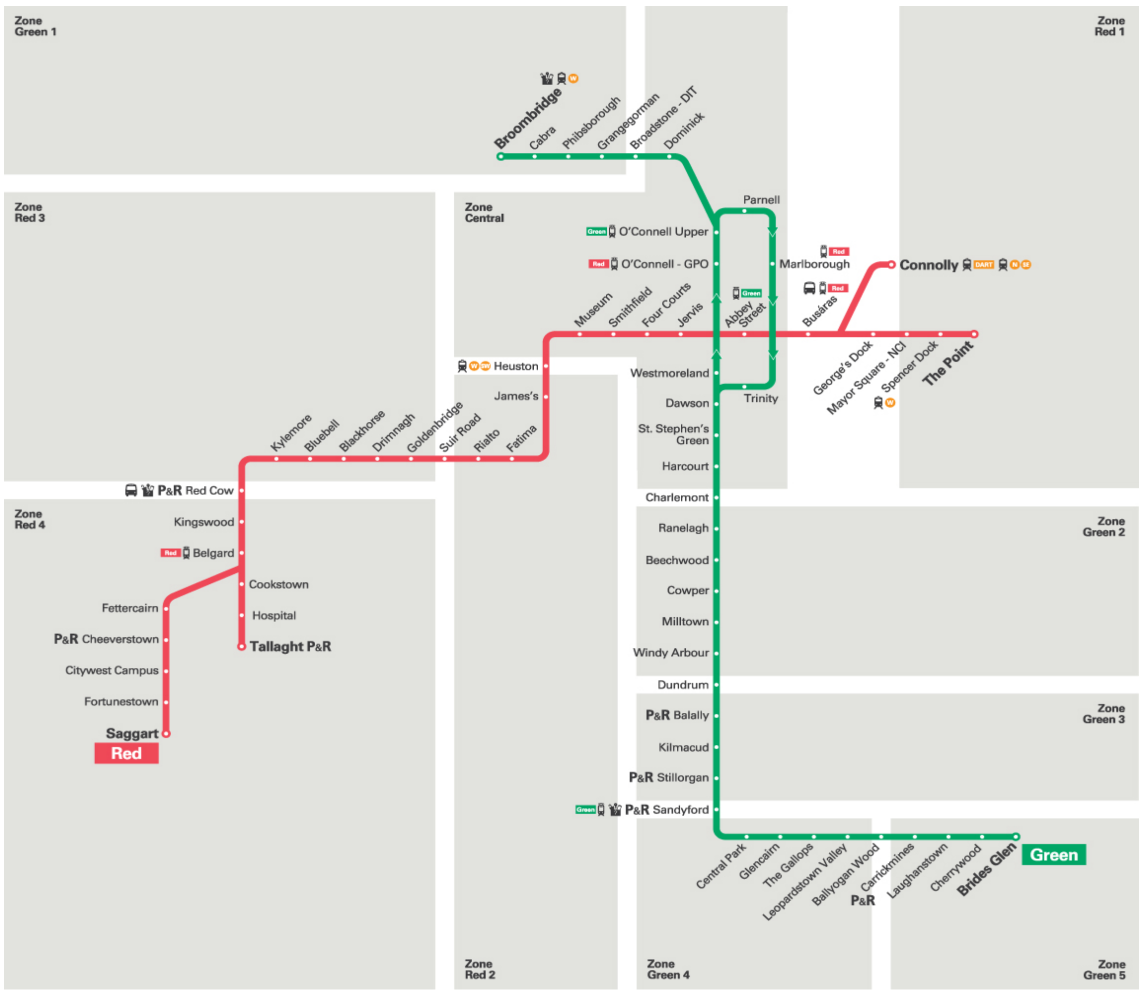


Figure 3.8 Dublin Luas Map (Transit_Maps, 2017)

ble, Luas_Days table, Luas_Frequency table and Luas_Delay_Frequency table. This schema contains ten tables. We now outline the purpose of and contents of each table.

1. The purpose of the **Luas_Stops** table is to store the information about all the Luas stops (Luas stations). This information contain a unique stop_id, the stop_name, the line_direction_id that shows the direction of the Luas line such as inbound or outbound, stop_abrev (stop abbreviation) which is input parameter of Luas API, the Geo-location coordinate of Luas stops, location_id, is_park_ride shows the parking facility, and is_cycle_ride shows cycle parking option.

2. The **Luas_Line** table contain the unique_line_id and line_name such as Luas red line or Luas green line.

3. The **Luas_Direction** table contain the unique direction_id and direction_name such as: inbound travel direction or outbound travel direction.

4. The **Luas_Location** contain the unique location_id, location_name and its GPS coordinates

5. The **Luas_Line_Direction** table contain the unique line_direction_id created by combining the unique line_id from the Luas_Line table with the unique direction_id from the Luas_Direction table. This table also contains the line_name and the direction_name.

6. The **Luas_Stops_Distance** table contains the start_stop_id, the start_stop_ name, the start_stop_lat_lng, the end_stop_id, the end_stop_name, the end_stop_lat_lng, distance_km and the travel_minutes of each path. The path is a distance between two Luas stops on a Luas line at inbound or outbound direction. The distance and estimated travel time is calculated using the Google Distance Matrix API.

7. The purpose of **Luas_Times** table is to store the information retrieved from the Dublin Luas API. This information includes the due_minutes that is the Luas arrival time, the destination stop of the arriving Luas and the source timestamp is the update time of this information.

8. The **Luas_Days** table contain the unique day_id and the day_name i.e. Monday to Sunday.

9. The purpose of **Luas_Frequency** table is to store the estimated frequency arrival times provided by Dublin Luas management for each Luas stop. The estimated frequency denotes the estimated arrival time between one Luas tram and the next Luas tram arriving on the same stop. In Figure 3.10 we present the hard-coded frequency provided for Tallaght Luas Stop. These estimated frequency is not same for the morning hours, afternoon hours, evening hours, night hours and then for the holidays.

10. The purpose of the **Luas_Delay_Times** table is to store the Luas calculated arrival frequency. The Luas calculated arrival frequency denotes the actual average arrival time of Luas trams at a stop for a given time period as opposed to the estimated expected hard-coded arrival frequencies provided by the Dublin Luas Management team. All the referential integrity constraints(primary key and foreign key rules) were enforced during the formation phase.

To populate the data in the tables of Dublin Luas Schema, we have used algorithm E.4. The actual algorithm to populate Luas_Stops table is mention in appendix E.4.

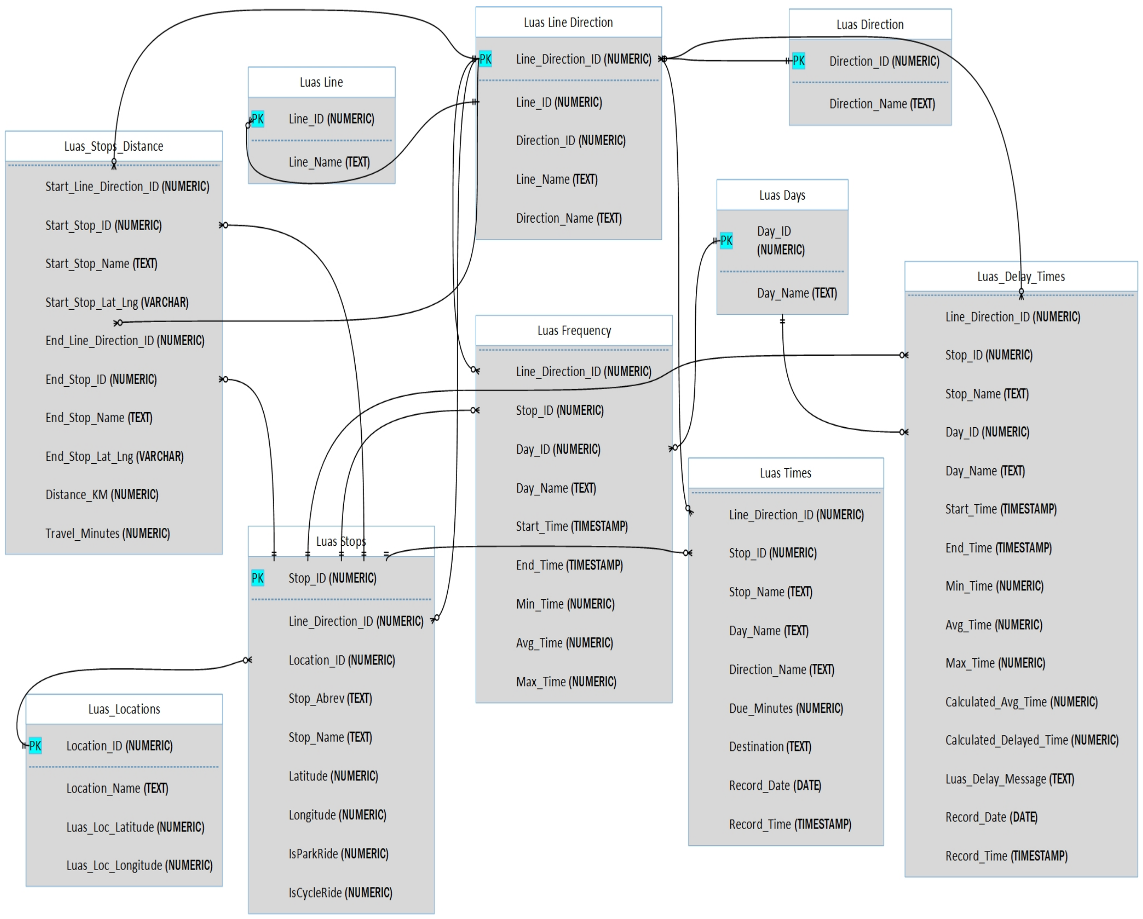


Figure 3.9 Dublin Luas Schema

Automatic Contextual Enrichment of Dublin Luas

The term path denotes the distance between two nearest Luas stops on an inbound or outbound direction for the Luas green or red line. To construct the Luas paths we have used algorithm mention in appendix E.5. After the **paths** has been generated the paths list is processed by another algorithm E.6 that will calculate the distance of the Luas paths in kilometers and the estimated travel time in minutes using the Google Distance Matrix API. We have mentioned the actual API call and sample output result of Google Distance Matrix for Dublin Luas stops in appendix E.8. Finally, this information is passed to the SQL query detailed in appendix D.7 to insert the data into **Luas_Stops_Distance** table.

The purpose of the algorithm 3.8 is to extract the real-time information of all Luas trams arriving at the given stop and their estimated due times. The algorithm processes one stop data at a time. The information contains all the Luas trams coming towards that stop,

Algorithm 3.8: `Extract_Luas_Times`

```
/* The Luas real-time information is extracted using the Dublin Luas official API.        */
input  : realtime_data - the realtime data from a Dublin Luas API i.e due minutes, destination, timestamp.
         line_direction_id - a unique id to show the travel direction of the Luas e.g inbound or outbound.
         direction_name - shows whether the Luas is travelling direction is inbound or outbound.
         stop_abrev - is the abbreviation of stops that is e.g 'TAL' for Tallaght stop.
output : luas_times_list -a list containing all the Luas stops data in real-time.
begin
    realtime_data ⟵ Luas_API(stop_abrev);
    luas_time ⟵ empty list;
    luas_times_list ⟵ empty list;
    stop_data ⟵ empty list;
    for row in realtime_data do
        if (row[destination] != 0) then
            stop_data ⟵ retrieve stop data from Luas_Stops table using unique direction_id and stop_abrev;
            if (stop_data != 0 && row[arrivaltime] == 'due') then
                arrival_time ⟵ 0;
            else
                arrival_time ⟵ row[arrivaltime];
            end
            luas_time ⟵ stop_data + realtime_data + arrivaltime;
            luas_times_list.APPEND(luas_time);
        end
    end
end
```

their estimated arrival time and their destination locations. The algorithm uses the unique stop_abrev and Dublin Luas official API to extract the information of that specific stop.

- ***(lines 1-5)*** The stop_abrev is passed to Luas API as an input to retrieve the data of that particular stop and to store this information in realtime_data variable. This information includes the due minutes of the arriving Luas (Trams), and their destination location name. Three empty lists are initialised called the luas_time that will store the real-time Luas stop information associated with a single Luas stop, luas_times_list will store the real-time information associated with all the stops and the stop_data will contain the information associated to the stop i.e. Luas stop_id and stop_name.

- ***(lines 6-8)*** Then we used a for loop to iterate through the information stored in the realtime_data variable , a loop variable row is used to retrieve a single row of data at a time from the realtime_data. An if condition will check for the availability of the destination information (Location name e.g Saggart or The Point stop.etc.), if satisfied the details of that specific Luas stop (stop id and stop name) is retrieve from Luas_Stops table and stored in stop_data. The data is retrieve by using a SQL query mention in appendix D.5.1 by passing the unique Luas stop_abrev and the line_direction_id.

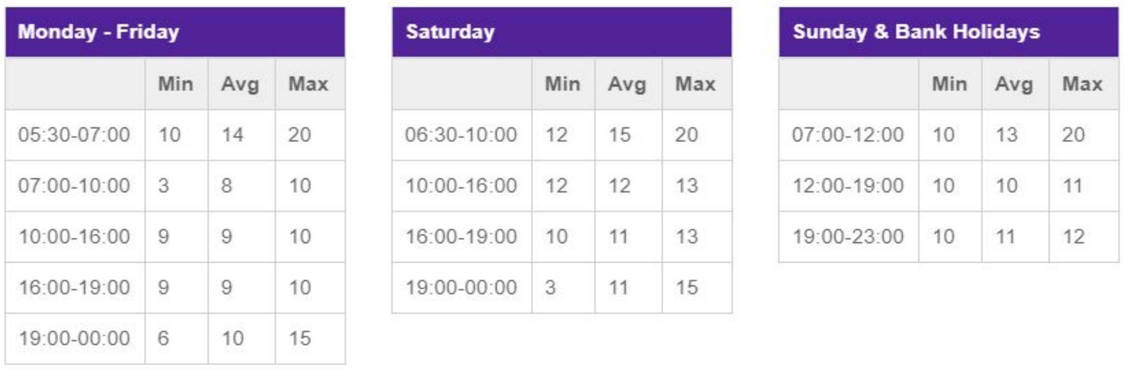
- ***(lines 9-13)*** A condition will check for the availability of the stop data and will convert the arrival time to zero if it's a string called 'due', otherwise will store the estimated arrival time in minutes to a variable called arrival_time.

- ***(lines 14-18)*** The stop_data along with the realtime_data and estimated arrival_time is assigned to the luas_time list, and then this list is appended to the luas_times_list forming a two dimensional list. The termination of loop will terminate the procedure too.

Automated Augmentation Enrichment of Dublin Luas

We designed algorithm 3.9 to determine the actual delay time in the Luas tram arrival frequency at each stop. The algorithm calculates the actual real-time Luas tram frequency delay for every stop and then compares the results with the estimated hard-coded frequency provided by the Dublin Luas management team. In Figure 3.10 we present the hard-coded frequency for Tallaght Luas stop provided by the Dublin Luas management team (Dublin_Luas, 2020b). The estimated frequency times is a tuple consisting of three values, a minimum, an average and a maximum. The minimum value shows the minimum estimated Luas tram arrival frequency, the average value shows the average estimated Luas tram arrival frequency and the maximum value shows the maximum estimated Luas tram arrival frequency. These estimated frequency values are different for the morning hours, for the afternoon hours, for the evening hours, for the night hours and for the bank holidays. The algorithm 3.9 will run every five (5) minutes and will also generate a display message according to the results after comparison such as: Better than the minimum frequency, Normal, Later than the average frequency, or later than the maximum frequency. The algorithm 3.9 will use the data from the Luas_Frequency table to extract the hard-coded frequency data provided by the Luas management team by using the unique day_id and the current time. The actual SQL Query to retrieve the frequency from Luas_Frequency table is mention in appendix D.8. The actual SQL query to compute the Luas real-time frequency from Luas_Times table is mention in appendix D.8.1.

- ***(lines 1-11)*** Four empty lists will be initialised that is luas_delay to store the calculated delay associated with a single Luas stop, luas_delay_list to store the calculated delay associated with all Luas stops, provided_frequency to store the given estimated frequency and the current_frequency is to store the actual calculated frequency. Then a for loop will use another variable called row to iterate through luas_times_list which is the output from algorithm 3.8. The provided_frequency is retrieved from the Luas_Frequency table and the current_frequency is calculated from the Luas_Times table. The provided minimum, average and maximum frequency is stored in separate variables for further comparison.



Monday - Friday			
	Min	Avg	Max
05:30-07:00	10	14	20
07:00-10:00	3	8	10
10:00-16:00	9	9	10
16:00-19:00	9	9	10
19:00-00:00	6	10	15

Saturday			
	Min	Avg	Max
06:30-10:00	12	15	20
10:00-16:00	12	12	13
16:00-19:00	10	11	13
19:00-00:00	3	11	15

Sunday & Bank Holidays			
	Min	Avg	Max
07:00-12:00	10	13	20
12:00-19:00	10	10	11
19:00-23:00	10	11	12

Figure 3.10 Tallaght Stop Luas Frequency (Dublin_Luas, 2020b)

Algorithm 3.9: `Calculate_Luas_Delay`

```
/* calculate the actual real-time delay for each tram arriving at each Luas stop.   */
input  : luas_times_list - the luas_times list calculated by algorithm 3.8.
         day - the current day name.
output : luas_delay_list - will return luas_delay list containing the calculated delay times of all stops.
1  begin
2      luas_delay ⟵ empty list;
3      luas_delay_list ⟵ empty list;
4      provided_frequency ⟵ empty list;
5      current_frequency ⟵ empty list;
6      for row in luas_times_list do
7          provided_frequency ⟵ retrieve provided frequency from Luas_Frequency table using the SQL Query D.8;
8          current_frequency ⟵ calculate the real-time frequency from Luas_Times table using the SQL Query D.8.1;
9          minimum ⟵ provided_frequency[minimum]
10         average ⟵ provided_frequency[average]
11         maximum ⟵ provided_frequency[maximum]
12         if (current_frequency < minimum) then
13             delay ⟵ 0;
14             Message ⟵ "Better than Minimum";
15         else if (current_frequency >= minimum) && (current_frequency <= average) then
16             delay ⟵ 0;
17             Message ⟵ "Normal";
18         else if (current_frequency > average) && (current_frequency <= maximum) then
19             delay ⟵ current_frequency - average;
20             Message ⟵ "Later than Average";
21         else if (current_frequency > maximum) then
22             delay ⟵ current_frequency - maximum;
23             Message ⟵ "Later than Maximum";
24         end
25         luas_delay = delay + Message;
26         luas_delay_list = APPEND(luas_delay);
27     end
28 end
```

- ***(lines 12-14)*** A conditional checks that the calculated current frequency is less than the minimum provided frequency. Recall that the provided minimum provided frequency refers to the earliest estimated arrival frequency of a tram at a Luas stop provided by the Dublin Luas Management. If the current frequency (actual due time) is less than the minimum estimated arrival frequency, then the tram is considered to arrive on time. Thus, the delay is set to zero and an appropriate message is generated

- ***(lines 15-17)*** If the previous condition check is not satisfied then the else part will be executed which checks whether the current frequency is greater than or equal to the provided minimum (earliest estimated arrival frequency) and is less than or equal to average estimated arrival frequency provided by the Dublin Luas management. If true, the appropriate message and delay are set. Note: we consider any arrival time less than or equal to the average estimated arrival time frequency provided by Dublin Luas management to be on time, that is to say there is no delay.
- ***(lines 18-20)*** In a similar manner, the next conditional checks whether the current frequency is greater than or equal to the provided average estimated arrival frequency and is less than or equal to maximum estimated arrival frequency provided by the Dublin Luas management. If true, the appropriate delay and message are set.
- ***(lines 21-24)*** Lastly, the final conditional checks whether the calculated current frequency is greater than the maximum estimated arrival frequency provided by the Dublin Luas management. If true, the appropriate delay and message are set.
- ***(lines 25-28)*** The message along with the calculated values is assign to the luas_delay. Then this list is inserted into Luas_Delay_Times table using a SQL Query similar to appendix D.6.

3.3.4 Dublin Motorways Infrastructure

Thousands of vehicles are using Dublin Motorways on daily basis to travel from one city to another or to reach from one corner to another with in the Dublin county (Transport_Infrastructure_Ireland, 2020). Dublin Motorways infrastructure is spread over six big motorways connected to each other. They all provide both inbound and outbound travelling services (TII_Road_Networks, 2019). All motorways contain several junctions which are connecting national roads to these motorways. There are digital display screens on motorways showing the travel times towards certain junctions or points. All motorways contain toll points which allows the driver to pay the toll tax, Some motorways facilitate the cash payment, direct debt and electronic toll but some motorways such as M50 only allow the electronic payment. The Information Management team of Dublin city traffic is providing real-time information of all motorways on a public website in an HTML format containing the travel times, signboards messages and weather data (TII_Traffic, 2020).

Automated Schema Formation of Dublin Motorways

The TFI (Transport_For_Ireland, 2019b) provide several text files that contains the useful information and data about the Dublin Motorways such as motorway_id, motorway_name, junction_names. This file in addition with the Dublin traffic official website (TII_Traffic, 2020) is analysed by our framework in-search of relevant and required textual or numeric information that will enable the process of automated schema generation for Dublin Motorways. To construct the Motorways table we used algorithm E.7 mention in appendix E.7.

Dublin Motorways Schema Architecture

In Figure 3.11 we present the Dublin Motorways schema generated after the analysis of the required data sources i.e. motorways.txt and the created text files from the live traffic website in order to search for the relevant textual and numeric information that will make the table columns. An almost identical algorithm to algorithm E.7 is used to construct the Junctions table, the Paths table, the Routes table, the Weather_Stations table, the Weather table, the Paths_Stations table, the Signboards_location table, the Signboards_Messages table, the Travel_Times table, the Travel_Times_Sum table, the Journeys table and the Journeys_Sum table. This schema contains thirteen tables. We now outline the purpose and content of each table.

1. The purpose of the **Motorways** table is to store the information about all the motorways. This information includes a unique motorway_id and motorway_name such as M50, M11 .etc.

2. The purpose of the **Junctions** table is to store the junctions information on a motorway. A junction is an interchange on a motorway. This table contain the unique junction_id and junction_name such as M50 junction 1, M50 junction 2 and so on.

3. The purpose of the **Paths** table is to store the paths information. The path is a distance between two adjacent junctions on a motorway. This table contain the unique path_id, motorway_id, junction_start_id, junction_start_name, junction_end_id and junction_end_name.

4. The **Routes** table stores the information about routes of motorway. A route is a combination of two or more adjacent paths. This table contain the unique route_id, motorway_id, start_path_id_id, junction_start_name, end_path_id and junction_end_name.

5. The **Weather_Stations** table contain the weather station_id and station_name. A weather station is providing the weather information of a certain geographical area under its coverage.

6. The **Weather** table contain the station_id, weather_updated_time, station_name, road_temperature, air_temperature and wind_speed.

7. The purpose of the **Paths_Stations** table is to show which weather station is near to or on which path of the motorway. This table contain station_id, station_name, motorway_id, path_id, junction_start_name and junction_end_name.

8. The **Signboards_Location** table contain the unique signboard_id, signboard_name, motorway_id, path_id, junction_start_name and junction_end_name. This table shows which electronic signboard is near to or on which path of the motorway.

9. The **Signboards_Messages** table contain the signboard_id, signboard_name, display_messages, record_date and the record_time.

10. The **Travel_Times** table contain all the motorway travel times information retrieved from traffic website. This include the free flow minutes, current travel time and the distance in kilometers.

11. The purpose of the **Travel_Times_Sum** table is to store the time calculated for routes. This table contain the calculated time for the routes using the path travel times data already stored in Travel_times table.

12. The purpose of the **Journeys** table is to store the information about a journey. A journey is a combination of two or more routes on the same or different motorways. This table contain the unique journey_id, journey_name, starting_route_id, starting_route_name, ending_route_id and ending_route_name.

13. The **Journey_Sum** table store the journey data calculated from the Travel_Times_Sum table. All the referential integrity constraints(primary key and foreign key rules) were enforced during the formation phase.

To populate the data in the tables of Dublin Motorways Schema using SQL query, we have used an almost identical approach to that outlined in algorithm 3.2.

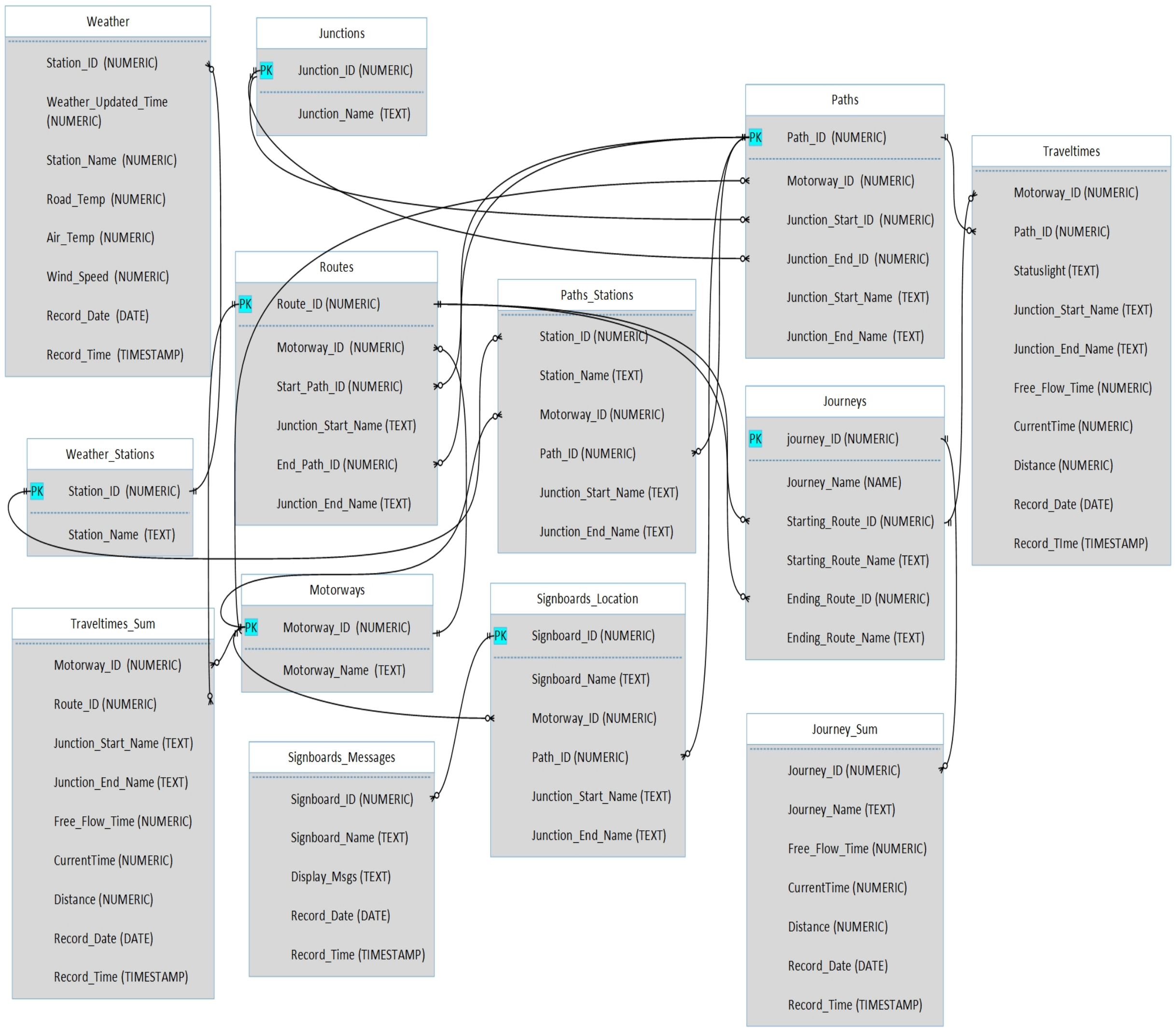


Figure 3.11 Dublin Motorways Schema

Automatic Contextual Enrichment of Dublin Motorways

Before proceeding with an exposition of the contextual enrichment stage, it is important to clarify several concepts. A path denotes the distance between two adjacent junctions on a motorway. A routes consists of two or more than two adjacent paths. A journey is a combination of two or more adjacent routes on the same or different motorways. Although, the traffic website is providing the travel time data already as Motorway paths (e.g from Junction 1 to Junction 2, Junction 2 to Junction 3 and so on) but also, the TFI (Transport_For_Ireland, 2019b) provide the paths.txt file so instead of recomputing the paths we simply populate the paths table. Then we have designed algorithm 3.10 to perform contextual enrichment by automatically extracting the data rows for all the motorway paths given on the Traffic

Algorithm 3.10: `Extract_M50_Times`

```
/* The M50 Motorway real-time travel information will be extracted through official traffic
   website.                                                                              */
input  : no.rows - the total number of rows of HTML table.
         no.cols - the total number of columns of HTML table.
         table_xpath - the Xpath location of rows in the HTML table.
output : M50_dataset - will return M50_dataset list containing the extracted travel times information of M50 paths merged
         with appropriate unique path_id.
begin
    path_data ⟵ empty list;
    M50_dataset ⟵ empty list;
    for (row in no.rows) do
        M50_dataset.append(empty list);
        for (column in no.cols) do
            path_data ⟵ table_xpath +[row + 1] + [column + 1];
            M50_dataset[row].append(path_data);
        end
    end
end
```

website (TII_Traffic, 2020).

The purpose of algorithm 3.10 is extract the paths travel time data for M50 motorway. The algorithm 3.10 will extract the M50 motorway junction names (paths), free flow travel time, current travel time and distance in kilometers. The total number of rows of the HTML table, the total number of columns and the total table size of HTML table is provide as input.

- ***(lines 1-3)*** The number of rows of HTML table, the number of columns and the Xpath location of rows in HTML table is passed to the function as an argument. The XPath is described as XML Path Language, It is a language use to find any element on the website with the help of XML path expression (XML_Path_Language, 2017). Two lists are initialised the path_data to store the data associated with a single path and the M50_dataset to store the data associated with all M50 paths.

- ***(lines 4-9)*** Then a for loop will execute through the number of rows and appends an empty list every time to M50_dataset list. A nested for loop will execute through the number of columns and appends the extracted data from the website to the path_data list. The path_data is then append to the M50_dataset list by using proper list index. With the termination of the outer for loop the function will also terminates.

Automatic Augmentation Enrichment of Dublin Motorways

Recall that a route is a combination of two adjacent paths on a motorway. We have designed algorithm 3.11 to calculate the routes time using the data extracted for paths and stored in the Travel_Times table. Then this calculated route times are insert into Travel_Times_Sum table.

Algorithm 3.11: `Calculate_Routes_Time`

```
/* The M50 routes time will be calculated from Travel times data from Algorithm 3.10 and
   Motorways Schema.                                                                    */
input  : dataset_length - the total number of rows is counted.
         start_path_id - the route start path id.
         next_path_id - the route second/next path id.
         start_row_index - the first row index of M50_dataset.
         next_row_index - the next row index of M50_dataset.
         M50_dataset - the M50_dataset list calculated by algorithm 3.10.
output : calculated_route_time - will return the calculated_route_time list.
1  begin
2      calculated_route_time ⟵ empty list;
3      for i in (dataset_length) do
4          if (start_path_id) in (M50_dataset[start_row_index]) then
5              if (next_path_id + i) in (M50_dataset[next_row_index + i]) then
6                  path_search ⟵ start_path_id + [next_path_id + i];
7                  sum_time ⟵ calculate the sum from Travel_Times table using the SQL Query D.9;
8                  route ⟵ retrieve route data from Routes table using the SQL query D.9.1;
9                  calculated_route_time ⟵ route + sum_time;
10             end
11         end
12     end
13 end
```

This algorithm will run every five minutes for all the the paths on M50 motorway.

The purpose of algorithm 3.11 is to calculate the time for all the routes on M50 motorway. The algorithm will perform the calculations by analysing the M50_dataset provide by algorithm 3.10.

- ***(lines 1-2)*** The dataset_length contain the length of M50_dataset, start_path_id contain the starting path id of the route, next_path_id contain the next path id of the route, the start_row_index contain the starting row index of M50_dataset, the next_row_index contain the next row index of M50_dataset, the M50_dataset is provided by algorithm 3.10. The calculated_route_time list is initialised to store the information associated with single route.
- ***(lines 3-5)*** A for loop will execute till the length of M50_dataset list. A conditional checks for the availability of start_path_id in the M50_dataset if satisfied a nested condition will check the availability of the remaining paths of that route in the M50_dataset list.
- ***(lines 6-13)*** A path search variable will be passed to the SQL Query mentioned in appendix D.9 to calculate the sum of the travel times data. In addition, the route information will be retrieved from the schema table Route by using SQL Query mention in D.9.1. The route information is combined with the calculated sum and is

then store in calculated_route_time list. When the loop is terminated the function will also be terminated.

3.3.5 Dublin Environment Infrastructure

The Environmental Protection Agency (EPA) of Ireland provide the air quality data for the entire country this data is updated at an interval of every thirty (30) minutes. The air quality monitoring stations are spread over the country and provide the data for that specific region/zone under its coverage (EPA_Stations, 2020). The air quality data represents the quality of air based on the presence of several particles in the air (EPA_Quality, 2019). On the other hand, the Ireland weather stations are providing the weather data, these stations are spread all over the country capturing the weather information of a specific geographical area. These weather stations provide the wind speed, the air temperature and the road temperature, and the update timestamp which is typically from every ten to fifteen (10-15) minutes (TII_Weather, 2020). The Information Management Team of Dublin Traffic provides the weather data in HTML data format (TII_Weather, 2020), while the EPA provides the real-time air quality information in JSON data format (EPA_API, 2020).

Automated Schema Formation of Dublin Environment

The public website by (TII_Weather, 2020) and the API provided by (EPA_API, 2020) are passed to our ELT framework that will analyse these data sources and will search for the required and relevant textual or numeric information i.e. location (station_name), updated (weather update timestamp), Road_Temp, Air_Temp and Wind_Speed. This information will allow the construction of an automated schema for the Dublin Environment. An almost identical algorithm to algorithm 3.1 is used to create the EPA_Weather_Stations table, the Air_Quality_Index table and the EPA_Weather table.

Dublin Environment Schema Architecture

In Figure 3.12 we present the Dublin Environment schema generated after the analysis of the required data sources i.e. the live Transport Infrastructure Ireland traffic weather website (TII_Weather, 2020) and the environment API (EPA_API, 2020). This schema contains three tables. We now outline the contents and purpose of each table.

1. The purpose of **EPA_Weather_Stations** table is to store the Weather stations information. This table contain the unique station_id, the station_name and the county_name e.g. county Dublin.

2. The purpose of **Air_Quality_Index** table is to store the quality index value along with the estimated range value of particles that are available in the air and upon which that index value is calculated (EPA_Quality, 2019). This table contains the unique quality_id, the quality_category, Ozone, Nitrogen_dioxide, sulphur_dioxide, particles_Pm2.5 and particles_Pm10 columns.

3. The purpose of the **EPA_Weather** table is to store the weather data in combination with the air quality data. This table contains the county_name, the station_id, the weather_updated_time, the station_name, the road_temperature, the air_temperature, the wind_speed, the quality_id, the quality_category such as: Good, Fair, Poor, the record_date and the record_time. All the referential integrity constraints(primary key and foreign key rules) were enforced during the formation phase.

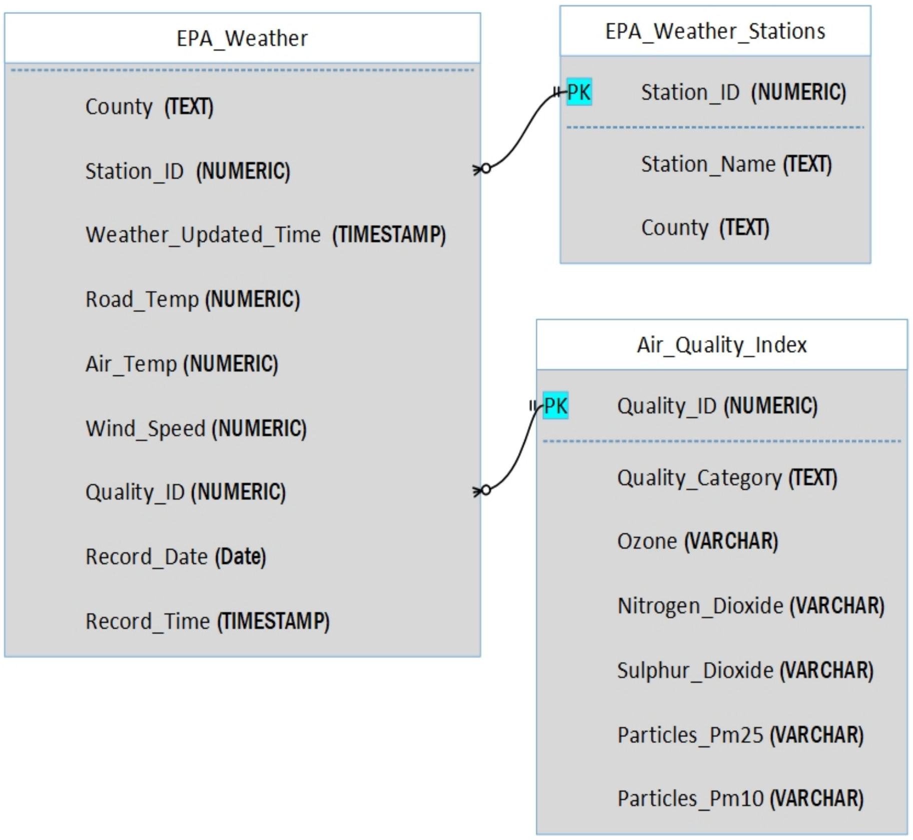


Figure 3.12 Dublin Environment Schema

Automatic Contextual Enrichment of Dublin Environmental Infrastructure

We have design the algorithms to perform the contextual enrichment through the automated extraction of the data rows from the weather website using the row indexes of the

Algorithm 3.12: `Air_Quality_Extraction`

```
/* The Air Quality real-time information is extracted via the official Environmental
   Protection Agency API.                                                              */
input  : realtime_data - the real-time air quality data from all the the Ireland is retrieved via the EPA API.
output : quality_dataset - a list containing the air quality data of the required region i.e. Dublin.
1  begin
2      realtime_data ⟵ EPA_API;
3      quality_data ⟵ empty list;
4      quality_dataset ⟵ empty list;
5      for (row in realtime_data) do
6          if row[region] == 'Dublin' then
7              air_quality ⟵ row[region_quality];
8              if (air_quality != null) then
9                  quality_data ⟵ retrieve quality data from air_quality_index table using air_quality value;
10                 quality_dataset.append(quality_data);
11             end
12         end
13     end
14 end
```

HTML table. Then another algorithm will extract the air quality data from the air quality API.

The purpose of algorithm 3.12 is to extract the information of air quality for Dublin county. The Algorithm 3.12 will extract the air quality data from the EPA official open API. The real-time data from the API is passed to the algorithm as input parameter. The algorithm will then extract the air quality data based on the selected region name, for example. Dublin.

- ***(lines 1-4)*** The realtime_data stores the API information. Two empty lists are initialise the quality_data to store the value of the quality data retrieved from the Air_Quality_Index table and the quality_dataset list.
- ***(lines 5-12)*** Then we used a for loop to iterate through the information stored in the realtime_data variable, a loop variable called row is used to retrieve a single row of data at a time from the realtime_data. A conditional checks for the availability of the string "Dublin" region, if satisfied will assign the current air index value to air_quality variable. Another nested conditional checks the availability of data in the air_quality variable, if satisfied will retrieve the quality index data from the Air_Quality_Index table using SQL Query mention in appendix D.10. This information is store in the the quality_data list which is then append to quality_dataset list.
- ***(lines 13-14)*** When the loop is terminated the function will also terminate.

The purpose of algorithm 3.13 is to extract the Dublin weather from an HTML website that provides information on wind speed, air temperature, road temperature and update time. The algorithm 3.13 will take the weather website as an input to analyse the required

Algorithm 3.13: `Dublin_Weather_Extraction`

```
/* The Weather real-time information will be extracted through official traffic website.    */
input  : columns_length - the total number of columns of HTML table.
         col_map - the Xpath location of columns in the HTML table.
         rows_length - he total number of rows of HTML table.
         row_map - the Xpath location of rows in the HTML table.
output : weather_dataset - will return weather_dataset list containing the weather information for the required region such as
         Dublin.
1  begin
2      weather ⟵ empty list;
3      weather_dataset ⟵ empty list;
4      for (row in rows_length) do
5          weather_dataset.append(empty list);
6          for (col in columns_length) do
7              weather ⟵ row_map + [row + 1] + [col + 1];
8              weather_dataset[row].append(weather);
9          end
10     end
11     return weather_dataset
12 end
```

information. The algorithm will extract the weather data using the Xpath mappings (recall the XPath is described as XML Path Language (XML_Path_Language, 2017), It is a language use to find any element on the website with the help of XML path expression).

- ***(lines 1-3)*** The columns_length, the rows_length, the Xpath location of the table row and table column of the weather HTML table is passed to the function as argument. Two lists are initialise the weather to store the data associated with single weather station and weather_dataset list to store the data associated with all the weather stations in county Dublin.
- ***(lines 4-8)*** Then a for loop iterates through the rows_length and will append an empty list every time to the weather_dataset list making it a two dimensional list. Then a nested loop will iterate through the columns_length to retrieve the data from the website and will store it in a variable called weather. Finally, the data of the single weather station stored in weather list is append to the the weather_dataset list.
- ***(lines 9-12)*** When the outer loop will finish execution it returns the weather_dataset list and the function will terminate.

3.4 Summary

In this chapter, we have presented our designed methodology of the Extract, Load and Transform framework. We have also discussed the automated formation of each single data model, we have explained the designed algorithms to perform the structural, contextual and

augmentation enrichment for each individual schema.
The clean and meaningful data after the transformation and contextual enrichment steps provides us the opportunity to derive some useful insights. Due to the reason of having a vast amount of data available in our data repository and most important because of the given time constraint for the research project we have decided to primarily focus on the Dublin Bikes data model. Our developed framework have the potential and the flexibility to implemented on the datasets of other smart cities such as: the New York Citi Bikes (CitiBike, 2019) and the London Santander Cycles (Santandar_Cycles, 2020). The smart cities are providing its live transport data through an open API using standard output formats i.e. JSON or XML. A little changes are required in our framework to implement it for other smart cities such as: providing the new API URL for New York or London and updating the schema column names in the scripts.

After, analysing the Dublin Bikes dataset we thought that it provides us a good opportunity to visualise the insights through a query and visualisation service. This motivation led us to investigate and design an API and Visualisation service which is explained in detail in the next chapter.

Chapter 4

Query and Visualisation Service

In this chapter we present the Query service and Visualisation application that we have developed. The development of the query service and visualisation application are a central aspect of the overall architecture. Our query service can form a foundational component upon which future smart city applications could be developed to allow end users to communicate with our Dublin Smart City data repository. In addition, the developed visualisation application for Dublin Bikes Sharing Scheme (DBSS) has key importance for demonstrating the newly derived insights and obtained knowledge.

4.1 Experimental Setup

In Appendix.A we have explained in detail how to install the required tools and libraries to run Extract, Load and Transform (ELT) architecture, the query service and the Vue.js on a vanilla modern desktop or laptop computer.

4.2 Application Programmable Interface (API)

A set of communication protocols, routines and tools necessary for developing software applications is known as Application Programming Interface (API). An API defines the interaction of software components. It is often used when programming the Graphical User Interface (GUI) components. An API is a communication protocol or interface between a client and server to facilitate the creation of client-side software. It has been described as a "contract" between the client and the server, such that if the client makes a request in a specific format, it will always get a response in a specific format or initiate a defined action (Braunstein, 2018, pp. 271-289).

4.3 Dublin Bikes Sharing Scheme (DBSS) Query Service

We have designed an Application Programmable Interface (API) or the query service for Dublin Bikes dataset using the Python Hug API framework (Python_Hug, 2019). The advantages of Python Hug API framework is discussed in detail in Appendix.B. This query service will allow the end users to communicate with the Dublin Bikes data stored in our Canonical Data Model (CDM) securely and efficiently. A user can retrieve the desired data from our Canonical Data Model by changing the parameters in the API call. In Figure 4.1 we have demonstrated the architecture of our Query Service.

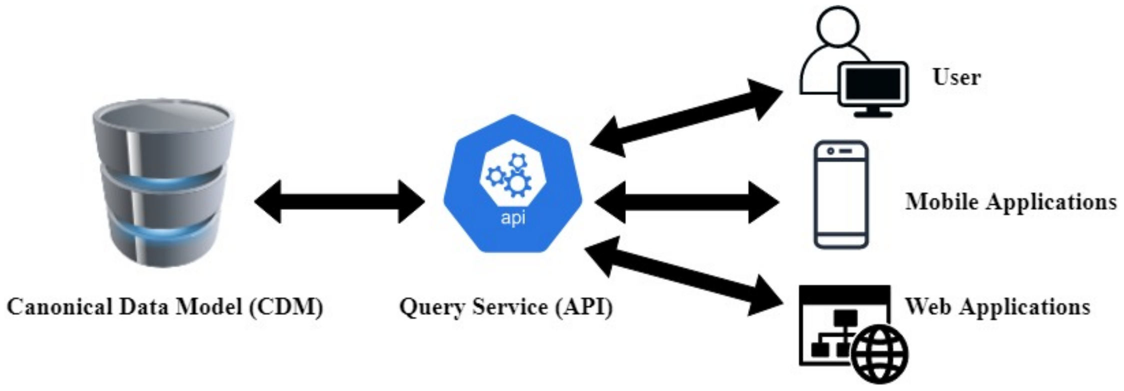


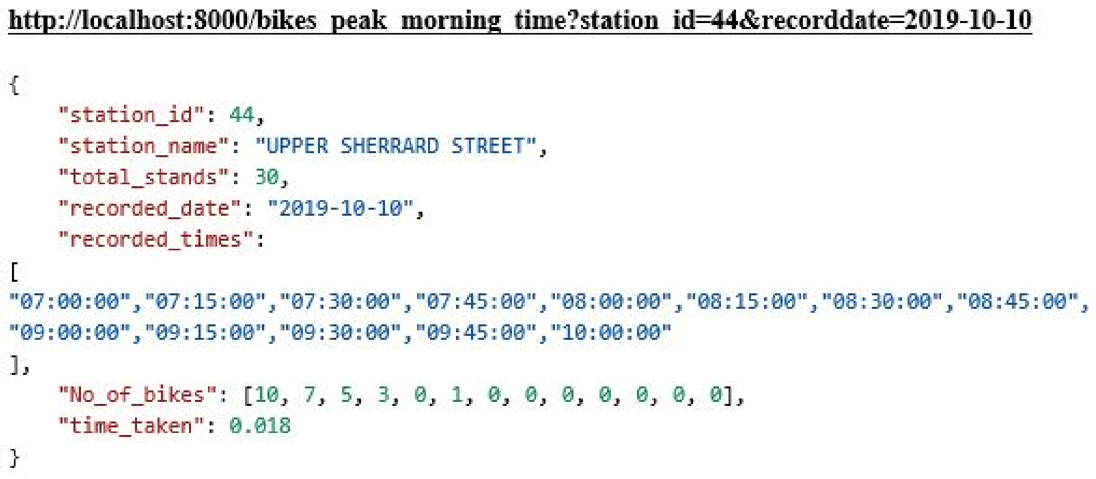
Figure 4.1 Query Service Architecture

A user can communicate with our Canonical Data Model by sending the required parameters to a specific API call such as date, time and bike station number (as explained in section 4.3.1). The data returned from by Python Hug API will be in JavaScript Object Notation (JSON) format. Smart city applications can be developed and connected to the CDM using our API service. Also, web and smart phone applications could be created to easily connect to our query framework due to its JSON output format. An advantage of our query service is it avoids users directly communicating with our CDM, thus providing a layer of security and facilitating a modular design.

4.3.1 DBSS Query Service Response

In Figure 4.2 we present the actual output response provided in JSON format after the API is called by sending the required parameters. We have used the built in server provided by the python hug library during the development phase which can we switch to any other server when deploying the service.

The URL section in Figure 4.2 contains the name of the API function which you are calling .i.e. bikes peak morning time, the next required parameter is station id which is a



http://localhost:8000/bikes_peak_morning_time?station_id=44&recorddate=2019-10-10

```
{
    "station_id": 44,
    "station_name": "UPPER SHERRARD STREET",
    "total_stands": 30,
    "recorded_date": "2019-10-10",
    "recorded_times":
[
"07:00:00","07:15:00","07:30:00","07:45:00","08:00:00","08:15:00","08:30:00","08:45:00",
"09:00:00","09:15:00","09:30:00","09:45:00","10:00:00"
],
    "No_of_bikes": [10, 7, 5, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    "time_taken": 0.018
}
```

Figure 4.2 Query Response as JSON Format

unique number for every bike station and the last parameter is the record date for the data. A user can change these parameters to see the different bike stations data during various hours on different recorded dates. The output data retrieved from our CDM will be provided in a JSON format.

The fetched results will contain the unique bike station id, bike station name, total number of stands at that bike station, recorded date of the data, recorded times and the number of available bikes on those recorded times such as: Morning Peak Hours, Midday Hours, Afternoon Hours or Evening Hours.

In Figure 4.2 we have called the bikes peak morning time function and passed the parameters .i.e. 44 for bike id and ‘2019-10-10’ for record date. In response we get the JSON output containing the station id ‘44’, station name ‘UPPER SHERRARD STREET’, total stands ‘30’, recorded data ‘2019-10-10’, recorded times at an interval of every 15 minutes from ’07:00 to 10:00’, the number of available bikes at the mention recorded times and the time taken to retrieve this data from the CDM.

4.4 Data Visualisation

Data visualisation is the (often dynamic) graphical representation of quantitative or qualitative data for exploration to facilitate the identification of underlying trends or patterns so as to provide deeper understanding and new insights. Due to the extensive number of visualisation methods sometimes it can be hard to choose an appropriate method to produce visualisations.

The human mind is more responsive towards visually illustrated information rather than the numerical figures. The visualisation allows to understand the insights without any difficulties.

When we use the term data visualisation, it means the graphical or image representation of data values. These graphical data presentations facilitates the higher management in decision making, through easy illustrations of complex data and ideas, it also allows identifying new patterns and data trends. The facility of interactive visualisation, allows us to hold more information from charts and plots, therefore can easily change the data already seen in chart and its processing as well. This method puts data forward and return them in a specific methodical structure which contains some properties and variables for bringing the information (Center, 2013, pp. 1-14). The insights discovery techniques provides the opportunity to organisation owners to create sources of entirely different data and can build custom analytical views (Khan and Khan, 2011, pp.1-14).

In the last few years, many methods for visualising data have been developed such as: bar graph, line graph, scatter plot, histogram, pie chart, flow chart, bubble chart, Venn diagram, multiple data series, area chart, data flow diagram, cone tree, time line, entity relationship diagram, parallel coordinates, semantic network and tree map (Khan and Khan, 2011, pp.1-14). These methods are best suitable for their specific case scenario. These techniques consists of properties like interactivity, layout features, usability and due to these characteristics they are easy to use. The reason for these methods being used to visualise is due to their evaluation mechanism (Khan and Khan, 2011, pp.1-14).

4.5 Dublin Bikes Peak Times

To discover new insights and obtain a deeper knowledge about the Dublin Bike Sharing Scheme (DBSS) we have developed a Data Visualisation website called Dublin Bikes Peak Times by using the modern languages and tools such as the Progressive JavaScript Framework Vue.js (Vue_Framework, 2020) and the ECharts JavaScript library (Apache_Echarts, 2020). The detailed architecture of Vue.js framework and its advantages over other existing frameworks is discussed in Appendix.C. The website is connected to our Canonical Data Model with the help of using our own developed query service. In Figure 4.3 we presented an overview of the Dublin Bikes website architecture.

We have used the ECharts library and several Vue.js built in chart plugins such as vuecharts and chart.js to visualise the data as interactive charts and depicted the information in a clear visual format so the end user can see and understand the data easily. These charts have tool-tips that shows real data values over the chart graphical points on the mouse hover event. In addition, these charts have legends as well to display or hide any specific data

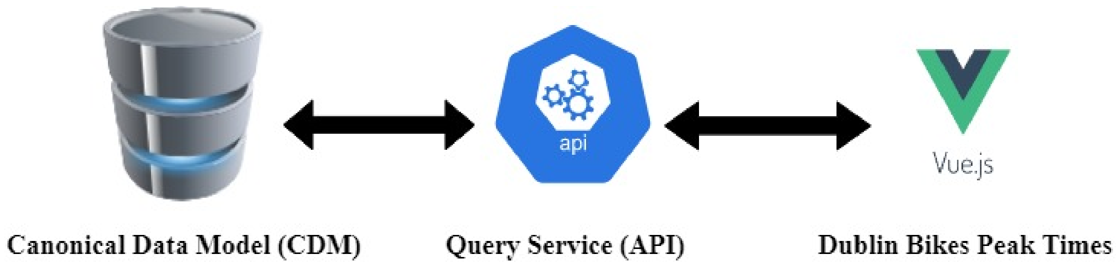
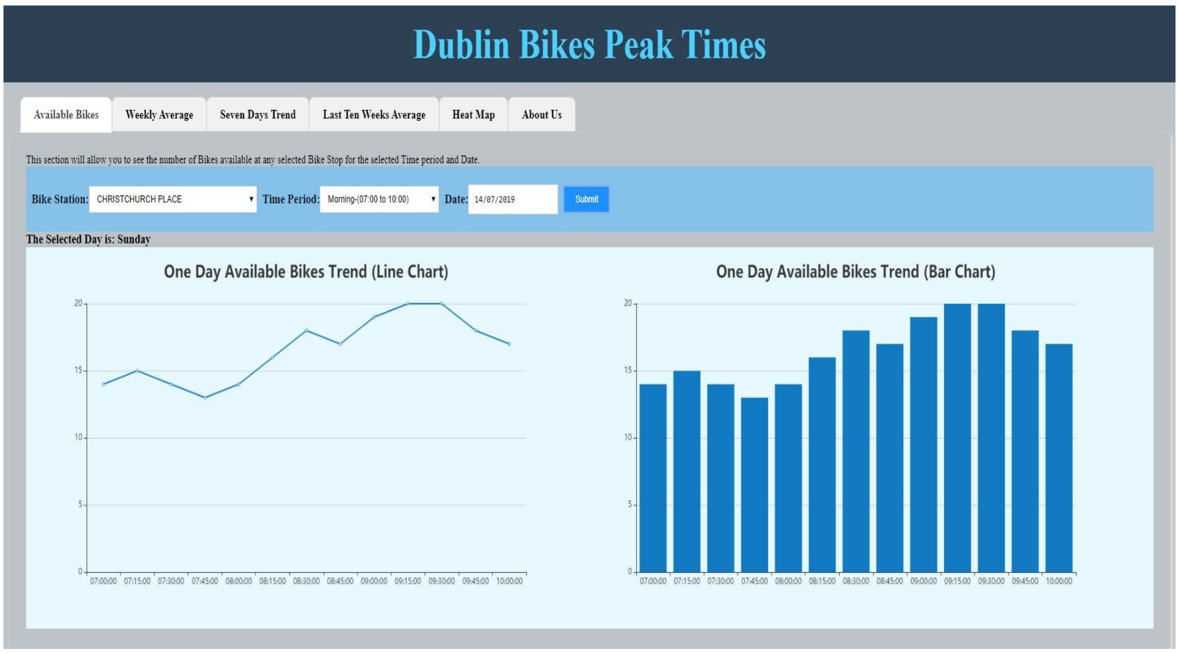


Figure 4.3 Dublin Bikes Website Architecture

on chart in real-time. The advantages of adopting ECharts over other popular charting frameworks is briefly discussed in Appendix C.

Also, we have created a heat map to illustrate the big picture of the usage trends across all the Dublin bike stations. The website shows the users with the number of available bikes in both line and bar chart to allow better understanding about the bikes usage trends. In Figure 4.4 we have demonstrated the design layout of our Dublin Bikes Peak Times web application.


Dublin Bikes Peak Times

Available Bikes | Weekly Average | Seven Days Trend | Last Ten Weeks Average | Heat Map | About Us

This section will allow you to see the number of Bikes available at any selected Bike Stop for the selected Time period and Date.

Bike Station: CHRISTCHURCH PLACE Time Period: Morning-(07:00 to 10:00) Date: 14/07/2019 Submit

The Selected Day is: Sunday

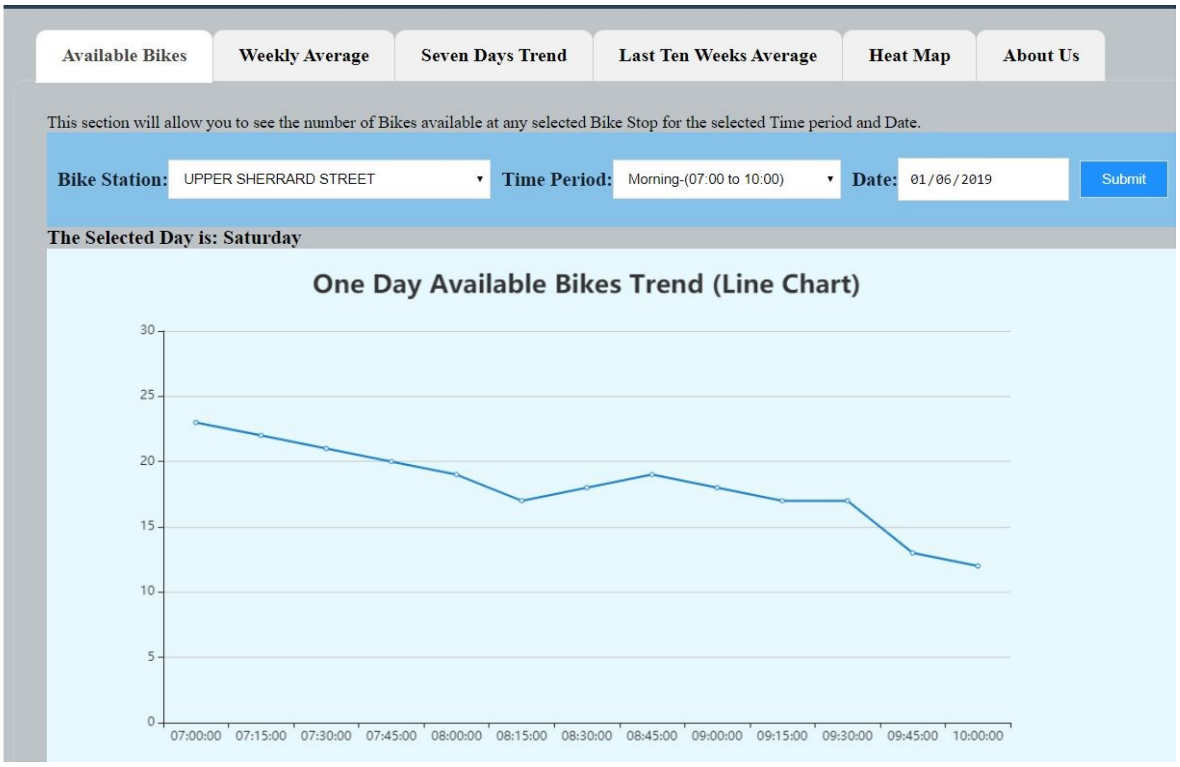
One Day Available Bikes Trend (Line Chart)
One Day Available Bikes Trend (Bar Chart)

Figure 4.4 Dublin Bikes Peak Times Web Application

Our Data Visualisation website contains six tabs, each tab providing a specific visualisation tool. In the following section, we present each tab of our web application including the available options to the end user and illustrated the output response being generated by user interaction. In addition, we explain the discovered patterns under each tab section as a sub section to separate the user guidance part from the observed trends part.

4.5.1 One day Tab

The first tab is the available bikes tab. This tab allows the user to see the number of available bikes for a single date. A user will be provided with three options to select under this tab: first is the bike station name which he wants to see, second is the time period such as: morning, midday, afternoon or evening and last is the date. An example of one day tab is depicted in Figure 4.5.



This section will allow you to see the number of Bikes available at any selected Bike Stop for the selected Time period and Date.

Bike Station: UPPER SHERRARD STREET

Time Period: Morning-(07:00 to 10:00)

Date: 01/06/2019

The Selected Day is: Saturday

Figure 4.5 One day Tab

In Figure 4.5 you will see the number of available bikes on bike station called the “UPPER SHERRARD STREET”, during morning time period from 7:00 to 10:00 on date 1st-June-2019 (Saturday). The information of available bikes is displayed as a line chart, the chart on y-axis is showing the total number of available stands on selected bike station and on x-axis shows the selected time period for which the data has been displayed.

Single Day Pattern Discovery

In the single day trends we have observed the bikes at station “UPPER SHERRARD STREET” have begun to withdraw sharply during the morning peak hours during the week and then a sharp rise in bikes lodgements have been noticed during the evening hours.

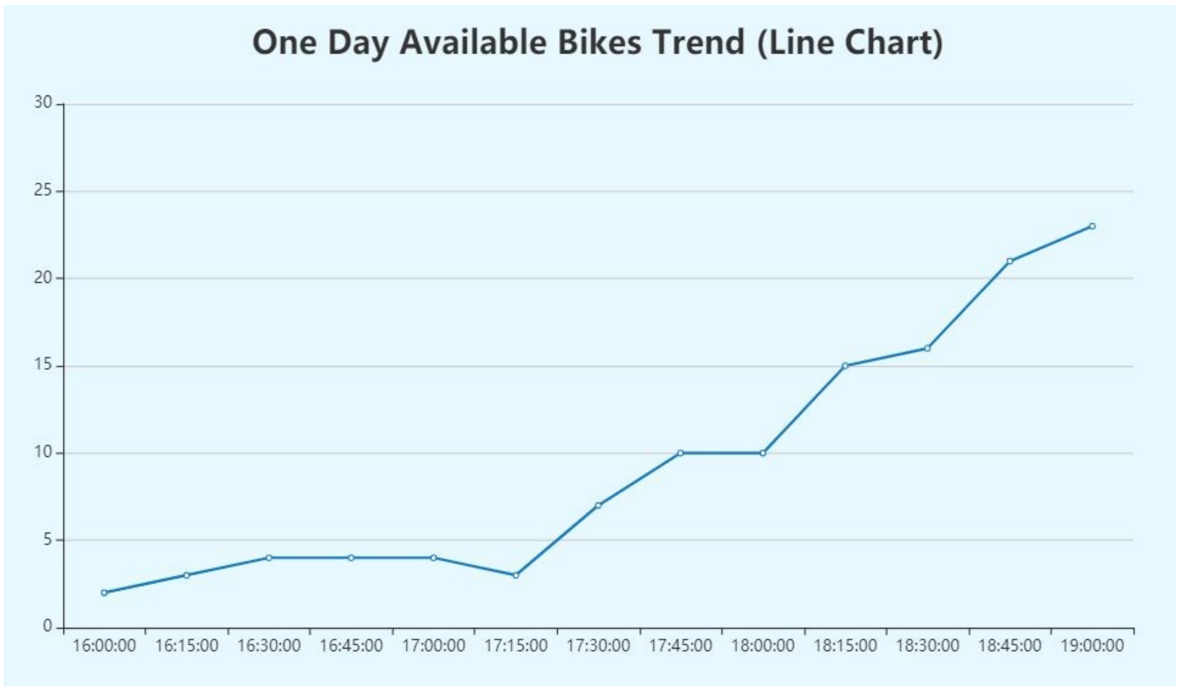
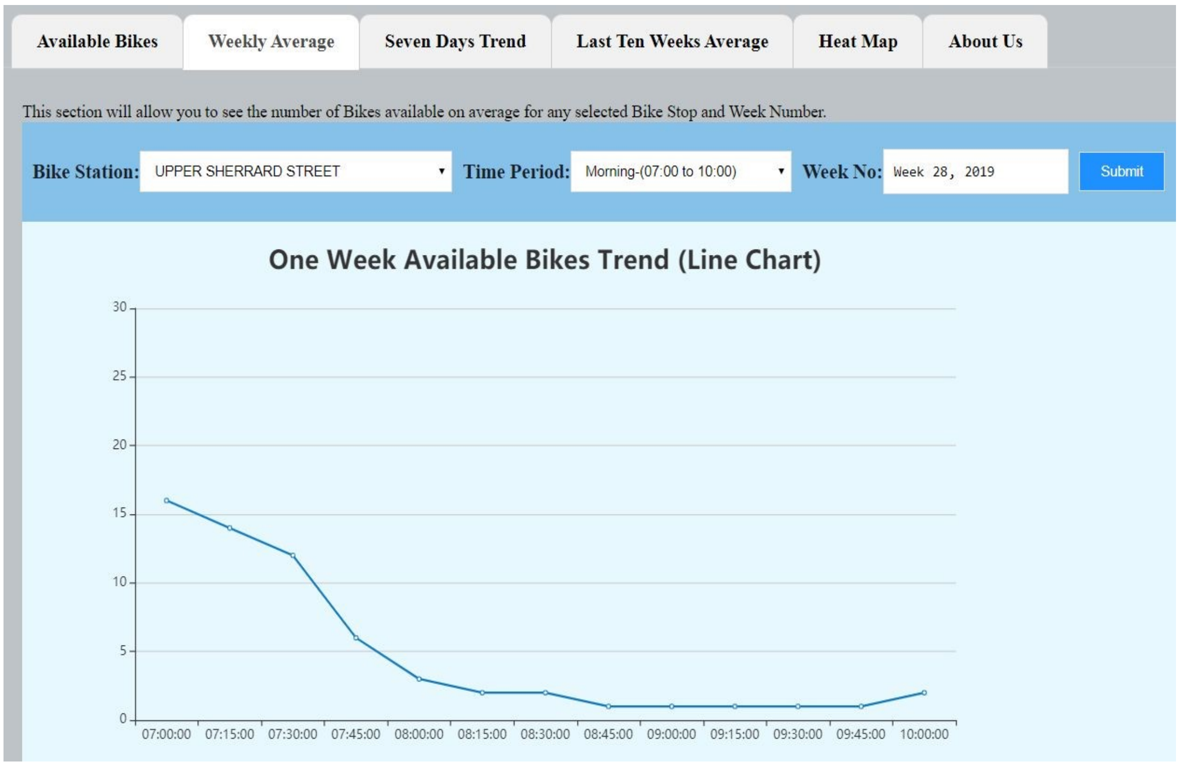


Figure 4.6 One day Trend Evening

In Figure 4.6 you can see the number of available bikes started to rise sharply in evening from 17:15 till 19:00 at the bike station called the “UPPER SHERRARD STREET”, the recorded date for the data is 24-July-2019 (Wednesday). The y-axis contains the total number of bike stands at the station while the x-axis shows the recorded times.

4.5.2 Weekly Average Tab

The second tab in our web application is to display the data on average for the whole selected week i.e. (Monday to Sunday). This tab have the same options as the previous one except the date which was replaced by the week number. A user will be provided with the opportunity to select the bike station, the time period, and the week number. An example of weekly average tab is shown in Figure 4.7.



This section will allow you to see the number of Bikes available on average for any selected Bike Stop and Week Number.

Bike Station: UPPER SHERRARD STREET

Time Period: Morning-(07:00 to 10:00)

Week No: Week 28, 2019

Figure 4.7 Weekly Average Tab

In Figure 4.7 you can see the average number of available bikes at bike station named "UPPER SHERRARD STREET", during the morning time period from 7:00 to 10:00 for the week 28, 2019 (Monday to Sunday). The query service will take the parameters from the website and send request to the CDM. Then the average number of bikes available at selected time periods for the entire week will be calculated and displayed on the chart. The x-axis and y-axis are containing the total number of bike stands and selected time period.

Weekly Average Pattern Discovery

The weekly average trends observed for the mornings and evenings peak hours were mostly same for the entire month of July 2019. The weekly pattern we had observed at the station "UPPER SHERRARD STREET" showed the withdrawal of bikes increase rapidly during the morning hours and a sharp growth in bikes lodgement during the evening hours.

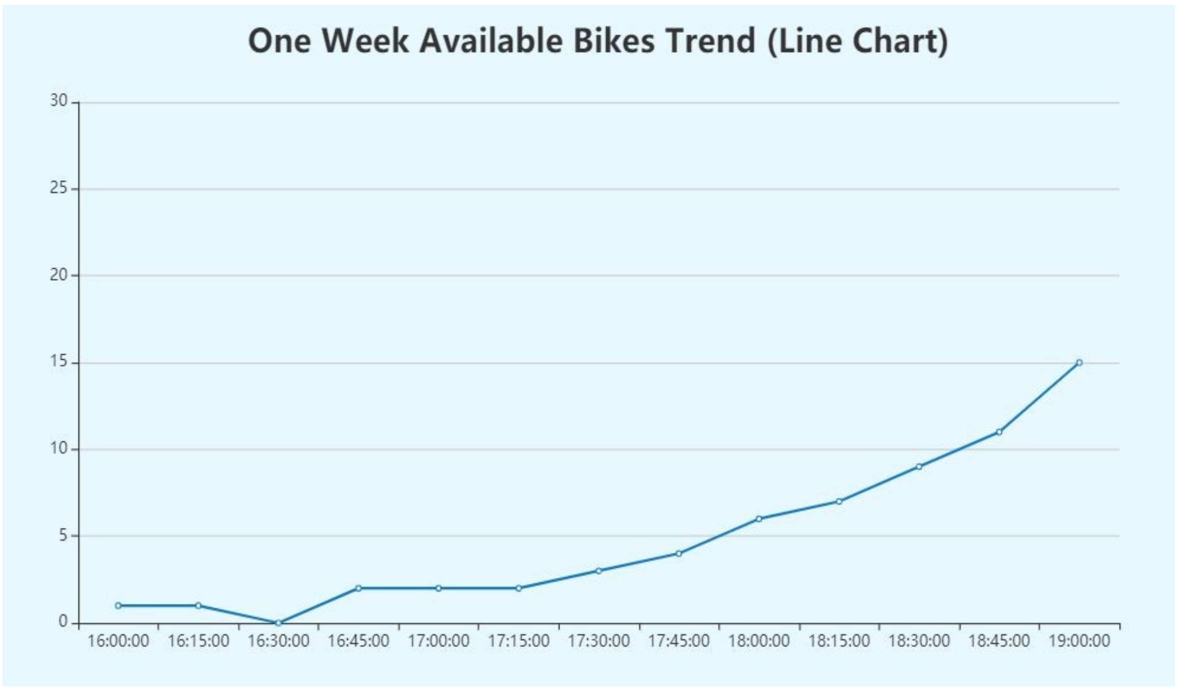


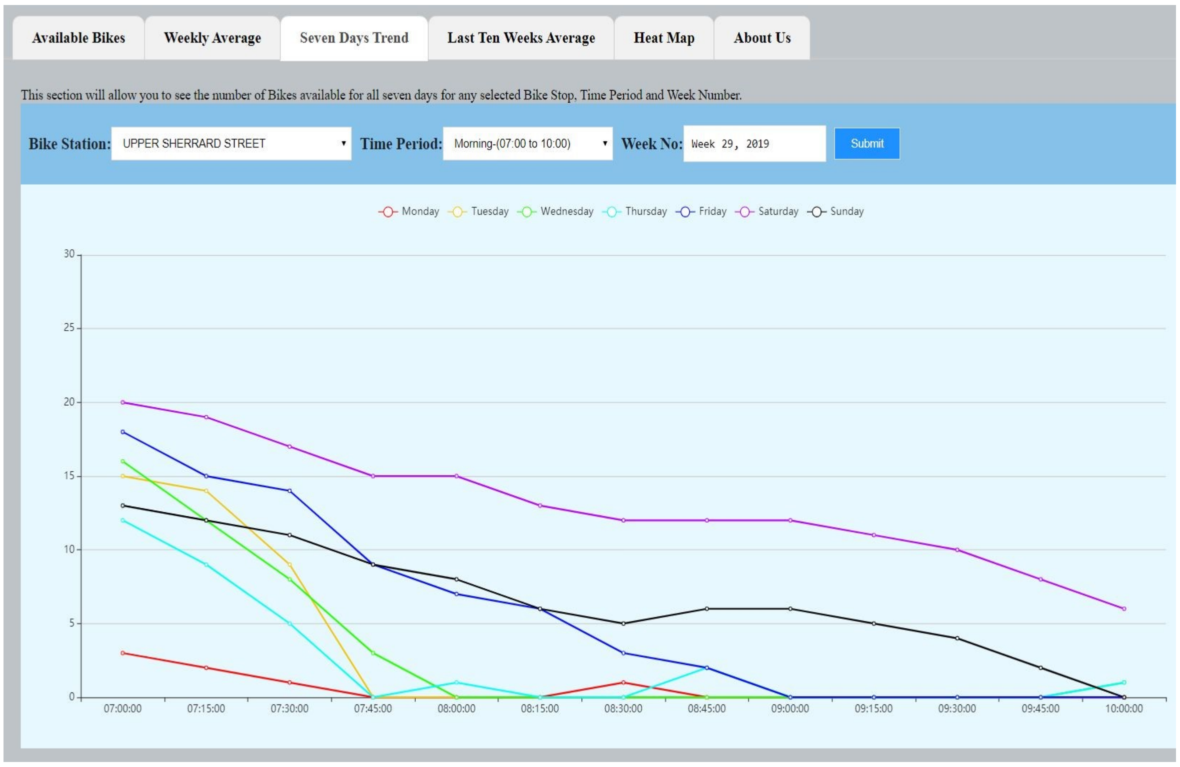
Figure 4.8 Weekly Average Trend Evening

In Figure 4.8 we have demonstrated the weekly average pattern at the bike station named "UPPER SHERRARD STREET" for the same week 28, 2019 (Monday to Sunday) but this time for the evening hours from 16:00 to 19:00. You can see a significant rise in the bikes availability after 16:00 and onward. However, this trend was found to be opposite during the morning hours means bikes are getting withdrawn in the morning and returned in the evening.

4.5.3 Seven Days Tab

The third tab is the called the seven days trend tab in our web application this will display the data for the whole selected week i.e. (Monday to Sunday) but separately for each day in different coloured lines. The red line shows the available bikes on Monday in the selected week, Yellow for Tuesday, Green for Wednesday, Sky Blue for Thursday, Dark Blue for Friday, Pink for the Saturday and Black for the Sunday.

The seven days tab contains similar options as the previous weekly average tab such as bike station name, time period and week number. In addition, the legends for each day is introduced on top of the chart to provide the user with the ability to display or hide unwanted days. For example to perform a comparison on various days in a selected week a user can display the weekdays (Monday to Friday) and can hide weekends (Saturday and Sunday) or do any desired selection. An example of seven days trend chart is shown in Figure 4.9.



This section will allow you to see the number of Bikes available for all seven days for any selected Bike Stop, Time Period and Week Number.

Bike Station: UPPER SHERRARD STREET Time Period: Morning-(07:00 to 10:00) Week No: Week 29, 2019

Figure 4.9 Seven Days Tab

In Figure 4.9 we have presented the seven days trend bikes available at bike station called "UPPER SHERRARD STREET" for the morning time period i.e. 7:00 to 10:00 for the week 29, 2019 (Monday to Sunday). Our designed query service will take the user selected parameters from the web application and will retrieve the data from our CDM for each day to display on chart. The data on y-axis contains the total number of stands at selected station and x-axis contains the selected time period for data display.

Seven Days Pattern Discovery

We have noticed some interesting trends for all the days (Monday to Sunday) at bike station called "UPPER SHERRARD STREET". The Seven days pattern showed the withdrawal during the morning hours and then an opposite trend of lodgement during the evening peak hours.

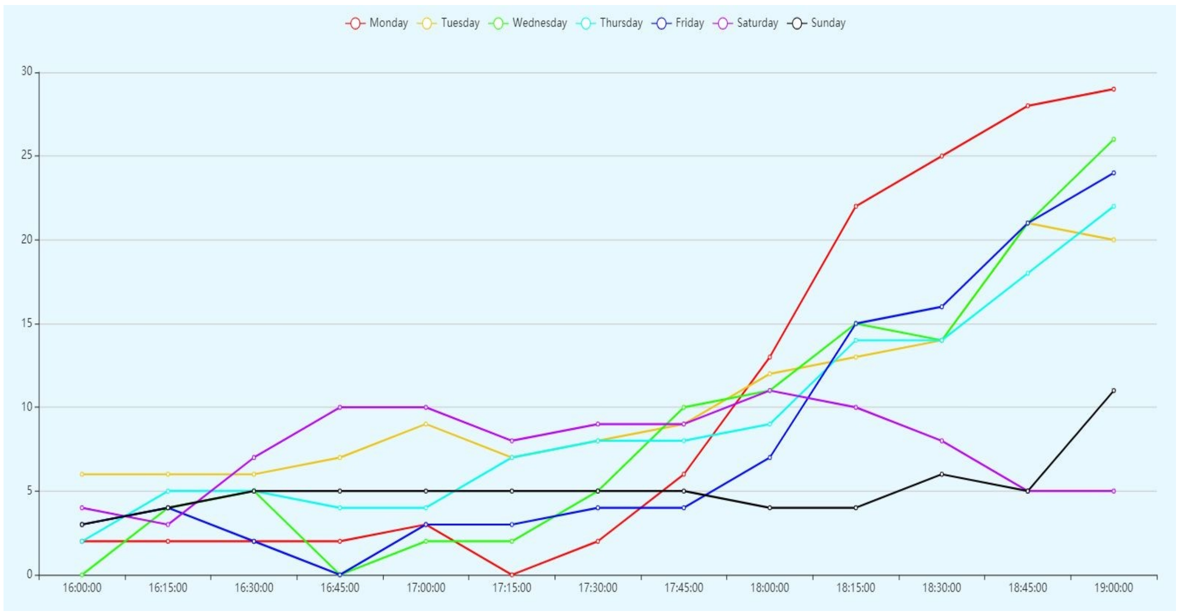


Figure 4.10 Seven Days Trend Evening

In Figure 4.10 you will see the pattern is totally opposite during the evening peak hours from 16:00 to 19:00 at “UPPER SHERRARD STREET” bike station for the week 29, 2019 (Monday to Sunday). You will noticed that during all the seven days a similar trend of withdrawal has noticed during the morning hours and then an opposite trend of lodgement has been observed during the evening hours.

In addition, our seven days tab will allow the user to see a full day trend of 16 hours at an interval of 30 minutes from 07:00 to 23:00 for all the seven days. In Figure 4.11 we have demonstrated the full day trend for the same bike station “UPPER SHERRARD STREET” for the week 30, 2019.

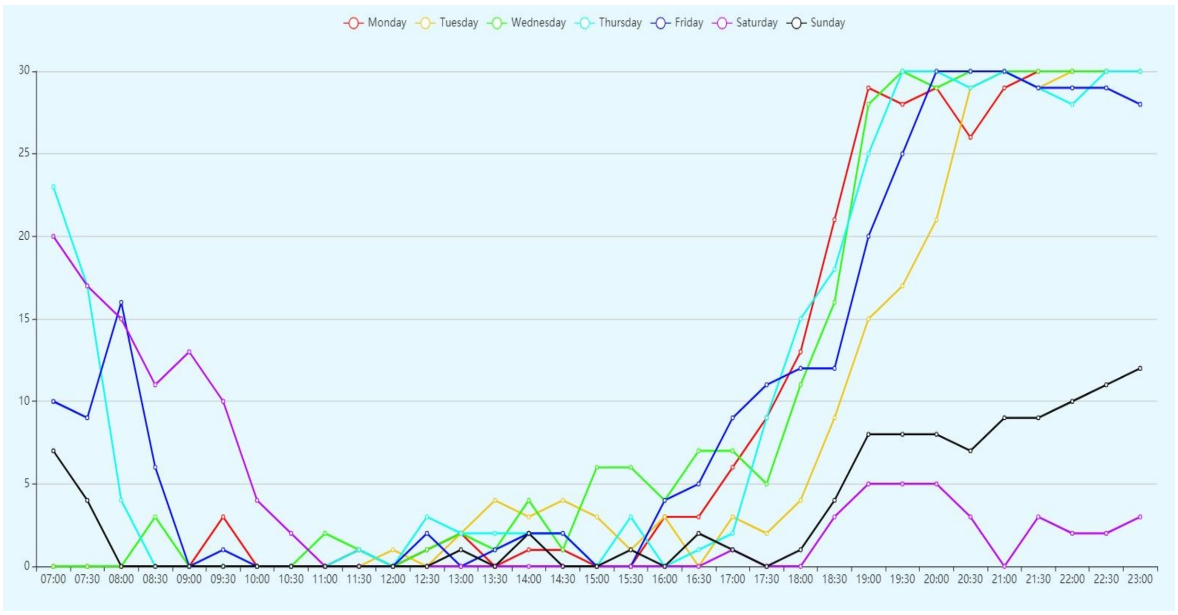
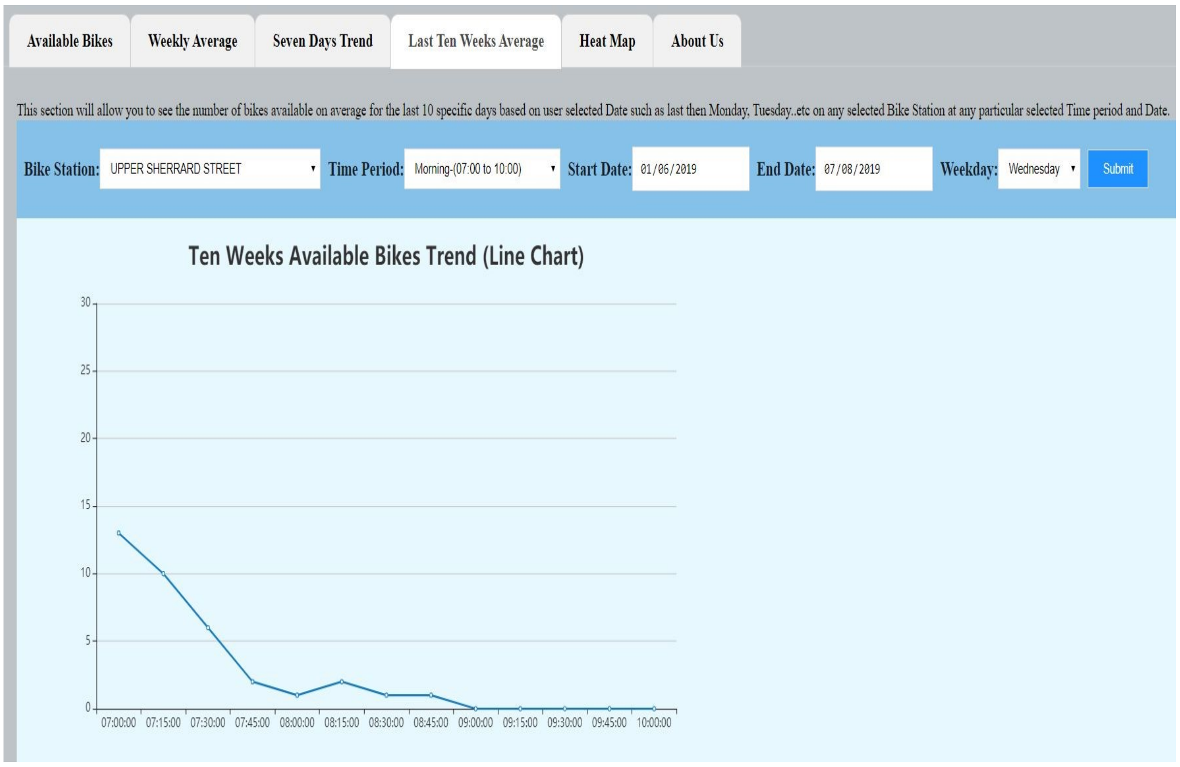


Figure 4.11 Seven Days Trend Full day

In Figure 4.11 we can see the peak withdrawal hours are the morning hours starting from 07:00 and then the station had a low availability of bikes after 09:00 for most of the days. The bikes available were less than 5 from 09:00 till 16:00 for the entire week and then a similar lodgement pattern can be observed after 16:00. However, the lodgement pattern for Weekends (Saturday, Sunday) was found to be less compared to the weekdays (Monday to Friday).

4.5.4 Last Ten Weeks Average Tab

The fourth tab in our Dublin Bikes Peak Times website is called the Last Ten Week Average, because it will display the average of the available bikes data for one specific day over the last ten weeks. We have provided some additional options in this tab such as the start date, end date and Weekday. A user can select the bike station name, the time period, the start date and then the end date but there should be 10 weeks difference between both dates otherwise it will generate an alert message to do appropriate selection. Lastly, the weekday option will allow to choose any particular day from Monday to Sunday to see the average available bike for that specific day over the selected 10 weeks. We have depicted the last ten weeks average tab in shown in Figure 4.12.



This section will allow you to see the number of bikes available on average for the last 10 specific days based on user selected Date such as last then Monday, Tuesday..etc on any selected Bike Station at any particular selected Time period and Date.

Bike Station: UPPER SHERRARD STREET
Time Period: Morning-(07:00 to 10:00)
Start Date: 01/06/2019
End Date: 07/08/2019
Weekday: Wednesday

Figure 4.12 Ten Weeks Average Tab

In Figure 4.12 you will see the bikes available on average at bike station "UPPER SHERRARD STREET" for the morning time period i.e. 7:00 to 10:00 for a total of 10 Wednesdays between 1st-June-2019 and 7th-August-2019. The query service will take the input parameters from the web application and will fetch the calculated average available bikes data from our CDM for selected day to display. The data on x-axis and y-axis contains selected time period and the total number of stands at selected bike station.

Last Ten Weeks Average Pattern Discovery

The last ten weeks average pattern showed similarity for all the days (Monday to Sunday) during the summer period 2019 at "UPPER SHERRARD STREET" station. The bikes are withdrawn during the morning peak hours and the station had a low availability of bikes till evening and then from 16:00 the bikes lodgement increased.

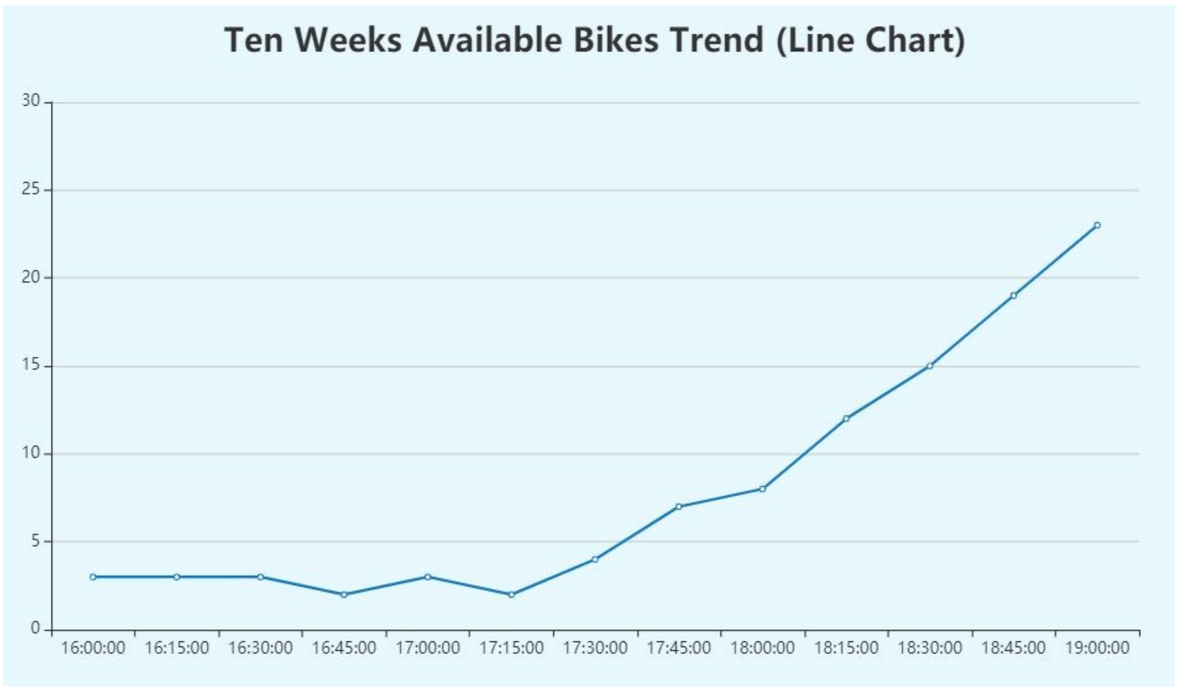


Figure 4.13 Ten Weeks Average Trend Evening

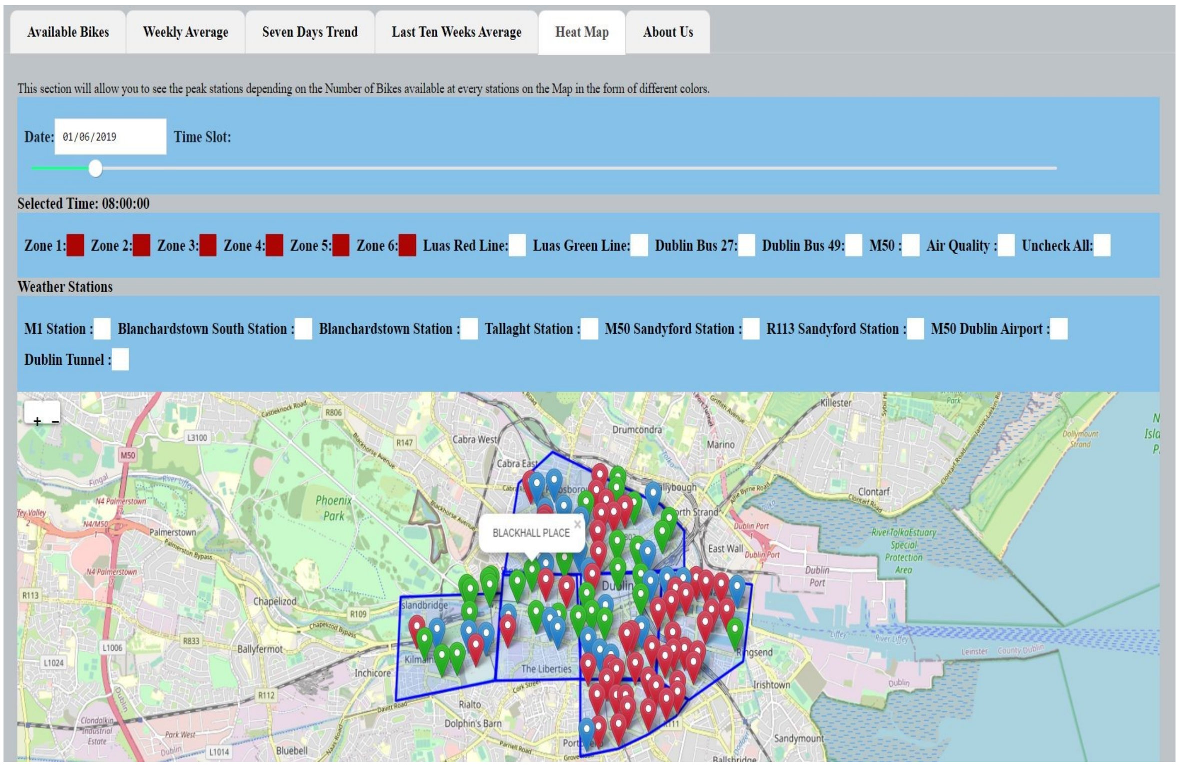
In Figure 4.13 you will see the ten weeks average evening pattern from 16:00 to 19:00 at "UPPER SHERRARD STREET" for 10 Wednesdays between 1st-June-2019 and 7th-August-2019. A sharp rise in bikes lodgement can be observed within the last ten weeks during the evening hours. However, this pattern was opposite during the morning peak hours and bikes are withdrawn fast and the station had a limited number of bikes after 09:00 but during the evening hours after 17:00 bikes are becoming high available.

4.5.5 Heat Map Tab

The last visualisation tab is called the Heat Map tab which contains a visual map and markers representing each bike station on the map based on their actual Global Positing System (GPS) coordinates. The heat map will display all 102 bike stations at the same time. Our heat will show the availability of bikes at all the bike stations simultaneously such as a Green colour for high availability, Red colour for low availability and Blue colour for medium availability at the stations.

We have introduced the option to select only one date at a time and then the user can slide to see the stations availability from 7:00 till 23:00 by using the slider control under the same tab. A user can select any marker to see the station name appear as a pop-up after click event. In addition, we have defined six zones each containing approximately the same number of bike stations. The six zones are Dublin City Centre, Dublin City North, Dublin

City South, South side, East-side, West-side. We have developed check-boxes to display the stations near the Dublin Luas lines, Dublin Bus route no. 27 and 49, and a checkbox for M50 Motorway. An example of heat map is shown in Figure 4.14.



This section will allow you to see the peak stations depending on the Number of Bikes available at every stations on the Map in the form of different colors.

Date: 01/06/2019 Time Slot:

Selected Time: 08:00:00

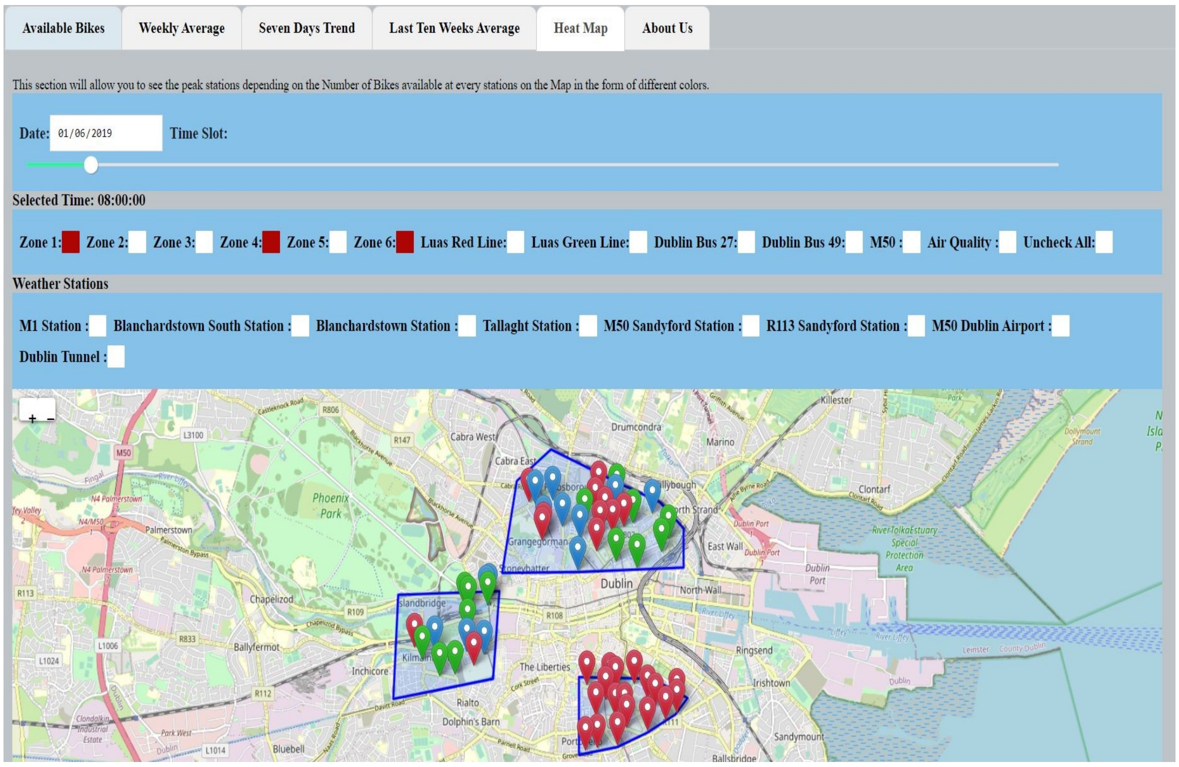
Zone 1: 🟥 Zone 2: 🟥 Zone 3: 🟥 Zone 4: 🟥 Zone 5: 🟥 Zone 6: 🟥 Luas Red Line: ☐ Luas Green Line: ☐ Dublin Bus 27: ☐ Dublin Bus 49: ☐ M50 : ☐ Air Quality : ☐ Uncheck All: ☐

Weather Stations

M1 Station : ☐ Blanchardstown South Station : ☐ Blanchardstown Station : ☐ Tallaght Station : ☐ M50 Sandyford Station : ☐ R113 Sandyford Station : ☐ M50 Dublin Airport : ☐ Dublin Tunnel : ☐

Figure 4.14 Heat Map Tab

In Figure 4.14 you will see the heat map showing the total stations and their availability at 8:00 in the morning on 1st-06-2019. All the zones are selected and appeared in blue polygons on the map. The bike stations in Red colour means that less than 1/3 bike are available, Blue colour shows that greater than (or equal to) 1/3 but less than 2/3 bikes are available, and the Green colour means that greater than (or equal to) 2/3 bikes are available. In Figure 4.15 we have presented the zone strategy and its advantage.



This section will allow you to see the peak stations depending on the Number of Bikes available at every stations on the Map in the form of different colors.

Date: 01/06/2019 Time Slot:

Selected Time: 08:00:00

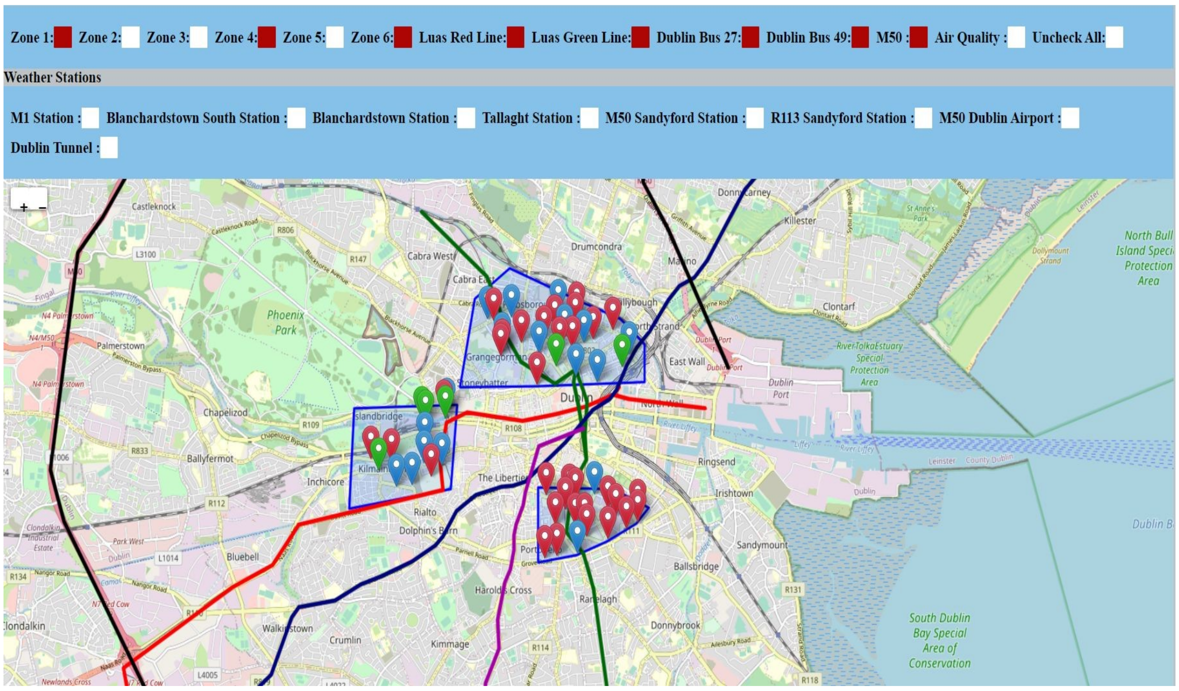
Zone 1: 🟥 Zone 2: ☐ Zone 3: ☐ Zone 4: 🟥 Zone 5: ☐ Zone 6: 🟥 Luas Red Line: ☐ Luas Green Line: ☐ Dublin Bus 27: ☐ Dublin Bus 49: ☐ M50 : ☐ Air Quality : ☐ Uncheck All: ☐

Weather Stations

M1 Station : ☐ Blanchardstown South Station : ☐ Blanchardstown Station : ☐ Tallaght Station : ☐ M50 Sandyford Station : ☐ R113 Sandyford Station : ☐ M50 Dublin Airport : ☐ Dublin Tunnel : ☐

Figure 4.15 Heat Map Zones

In Figure 4.15 we have only selected the zone 1, zone 4 and zone 6 to demonstrate how we can perform the bikes usage analysis by using the zones strategy. The zones technique will allow us to hide the unwanted zones and bike stations in those zones. By using desired zones the attention can be focused on only the desired zones to perform comparison among them and observe trends among them. In Figure 4.16 we have depicted how other transport services are intersecting the Dublin bike zones.



Zone 1: 🟥 Zone 2: ☐ Zone 3: ☐ Zone 4: 🟥 Zone 5: ☐ Zone 6: 🟥 Luas Red Line: 🟥 Luas Green Line: 🟥 Dublin Bus 27: 🟥 Dublin Bus 49: 🟥 M50 : 🟥 Air Quality : ☐ Uncheck All: ☐

Weather Stations

M1 Station : ☐ Blanchardstown South Station : ☐ Blanchardstown Station : ☐ Tallaght Station : ☐ M50 Sandyford Station : ☐ R113 Sandyford Station : ☐ M50 Dublin Airport : ☐ Dublin Tunnel : ☐

Figure 4.16 Heat Map Zones with other Transportation

In Figure 4.16 the check-boxes for Dublin Tramline service called the Luas Green line and Luas Red Line are checked to illustrate their actual coverage area on the map and to get a picture of the nearby bike stations to Luas lines. Also, we have demonstrated the two Dublin Bus routes i.e. 27 and 49 covering the Tallaght and passing from the Dublin city centre. The Blue line shows the Dublin Bus route number 27 and the pink line shows the route number 49. Whereas, the black line shows the M50 Motorway.

Heat Map Pattern Discovery

The heat map showed a matching pattern for most of the weekdays (Monday to Friday) throughout the year 2019. The heat map helped us understand the big picture of bikes flow patterns throughout the city throughout the day. The stations on the Dublin City North and West sides become low availability (Red) during the morning peak hours while, the South and East side stations become high availability (Green).

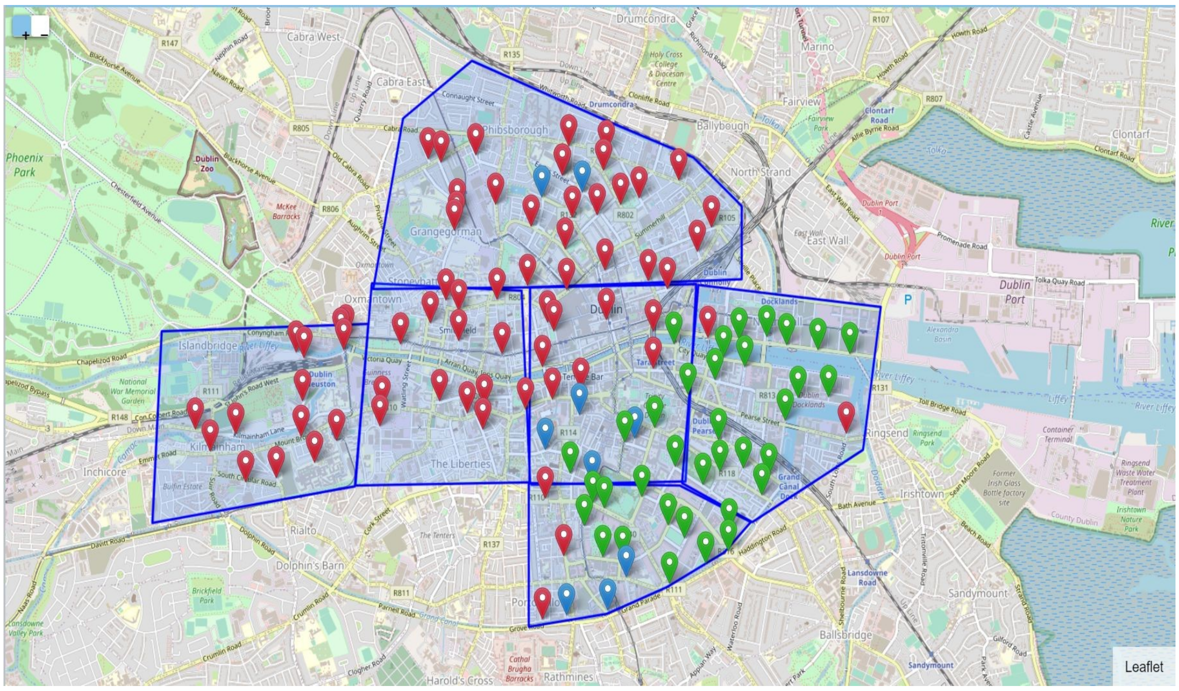


Figure 4.17 Heat Map Trend Morning

In Figure 4.17 you can see the situation of bikes availability on 8th-July-2019 at time 9:30 in the morning. The stations in red shows us the minimum bikes availability on the stations at Dublin City North and West side after the morning rush hours and at the same time the maximum bikes availability in green colour can be seen at the South and East side of the city.

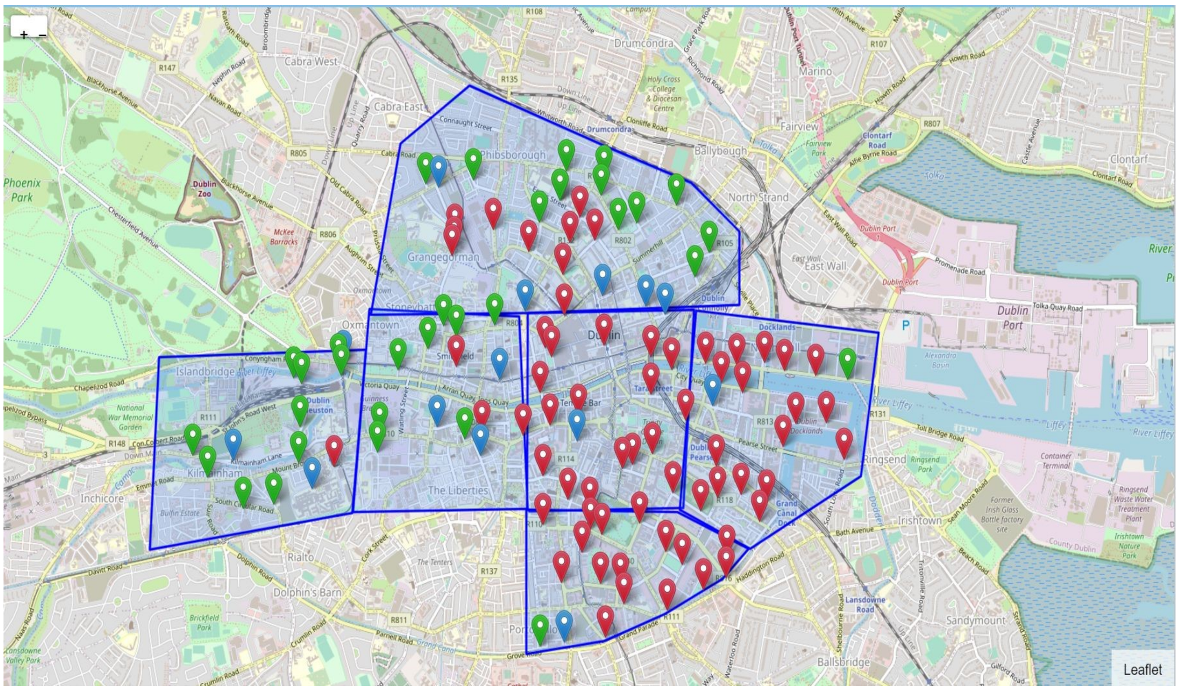


Figure 4.18 Heat Map Trend Evening

In Figure 4.18 we have demonstrated the bike availability situation on the same day which is 8th-July-2019 but at time 23:00. Now the pattern observed during the morning is changed to an opposite direction, the stations which had low availability (Red) in the morning are now high availability (Green) and the stations which were high availability (Green) are now low availability (Red).

These patterns helped us to understand the user behaviour in terms of bikes usage, it also showed that the bikes are withdrawn from the North and West during the morning hours and goes towards the centre, South and East side. While, during the evening time the bikes withdrawal is performed from the South and East side and goes back towards the North and West sides.

4.6 Summary

In this chapter, we have presented the architecture and working of our query service. We have also discussed our visualisation service in two different parts: the first part is about the user manual on how to use the application by exploring all the given options and how each option will work. Second part is the observed patterns in which we have explained the discovery of new insights and knowledge through observations from the data patterns while using the application.

These data patterns helped us to understand the usage behavior as well as left some motivational questions for us to find out by performing further analysis on the data such as:

- Finding the peak usage hours?
- Is there an efficient economic model available?
- What are the busiest locations?
- How miss opportunities can be calculated?
- How an efficient replenishing strategy can be developed?

However, after observing a visual representation of the data dynamics and by using the visualisation service we were able to identify the data trends for one day, one week, seven days, ten weeks and overall availability on the heat map. The unusual data patterns observed in Dublin Bikes data provides us the opportunity to develop an efficient economic data model. The unusual data patterns observed in this analysis has become the motivation for us to address these questions through the development of an economical model to bring efficiency in the system which is explained in detailed in the next chapter.

Chapter 5

Resource Allocation Analysis and Results

In this chapter we explain our performed analysis carried out on Dublin Bike Sharing Scheme (DBSS) stations in order provide the unseen insights demonstrating the issues related to the resource allocation or re-balancing of bikes which has a direct relationship to lost customers and decrease revenue.

5.1 Hypothesis Question

How to develop an efficient resource allocation strategy on DBSS that could facilitate in identifying the overflow and underflow bike stations, and help minimising the number of lost customers at underflow bike stations and increase revenue at these stations?

5.2 Experimental Setup

The data we have used in this experiment is taken from our own designed Canonical Data Model (CDM) which relies on our Extract Load Transform (ELT) framework to receive its data after being filtered and populated from real-time data streams at an interval of every five minutes since April 2019 until April 2020. The PostgreSQL statements are used to retrieve and sort only the required data from the CDM to carry out the designed analysis. After data retrieval step we save the data to a CSV file for further analysis.

The further analysis and calculation is performed manually on the generated CSV file by using Microsoft Excel and taking into consideration the features such as pivot tables and other built in functionalities of MS Excel.

5.3 Sampling of Dataset

The Dublin Bike scheme is providing bike rental services from 102 different bike stations spread across Dublin city centre covering the important locations and tourist spots. Each of these stations have a unique station identification number, some of these stations accepts credit card some do not. The total number of bike stands vary from station to station depending on the physical land allocated or the possible physical space available for that particular bike station. We have selected two bike stations from the Dublin north side, two from the city centre and two from the city south side.

We have selected bike station number 60 named as the "North Circular Road" and station number 44 called "Upper Sherrard Street" from the Dublin city north side. The selected stations from Dublin city centre includes bike station number 4 known as "Greek Street" and station number 6 called the "Christchurch Place". Lastly, the chosen station from Dublin city south side is bike station number 55 called the "Hatch Street" and station number 41 known as "Harcourt Terrace".

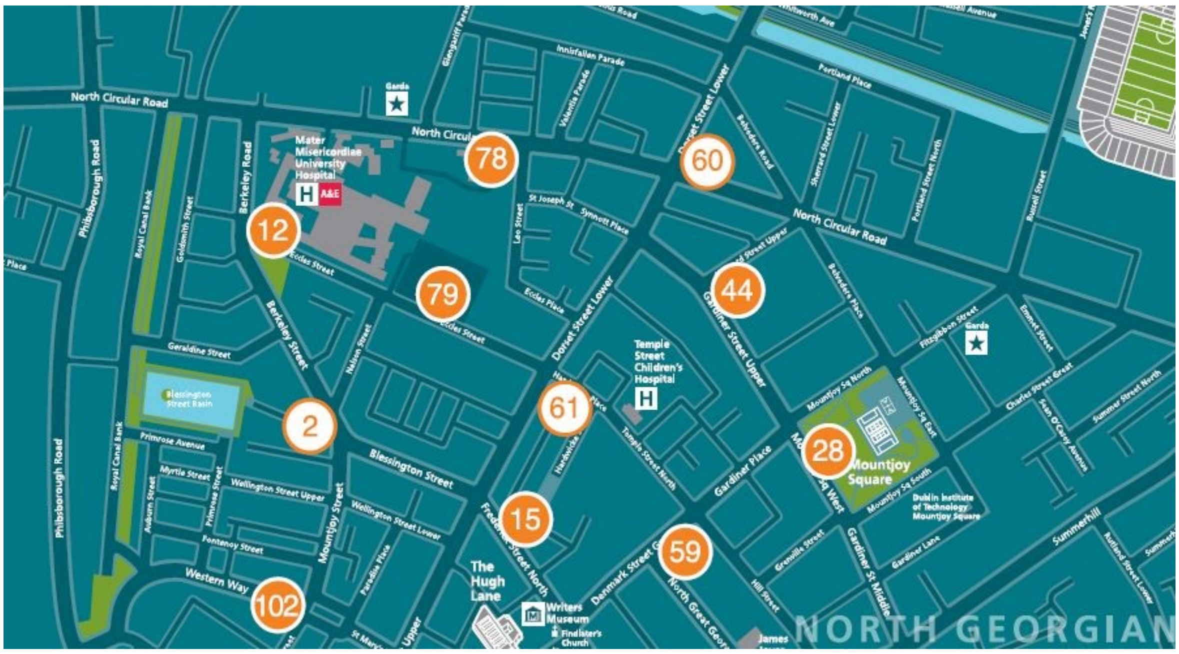


Figure 5.1 Dublin City North Bike Stations (Just_Eat, 2018)

In Figure 5.1 we have presented some of the bike stations from Dublin city north side and the reason for choosing bike station number 60 is due to it being the only station on the North Circular Road and for station number 44 is being located near to Temple Street Children Hospital and to Mountjoy Square.

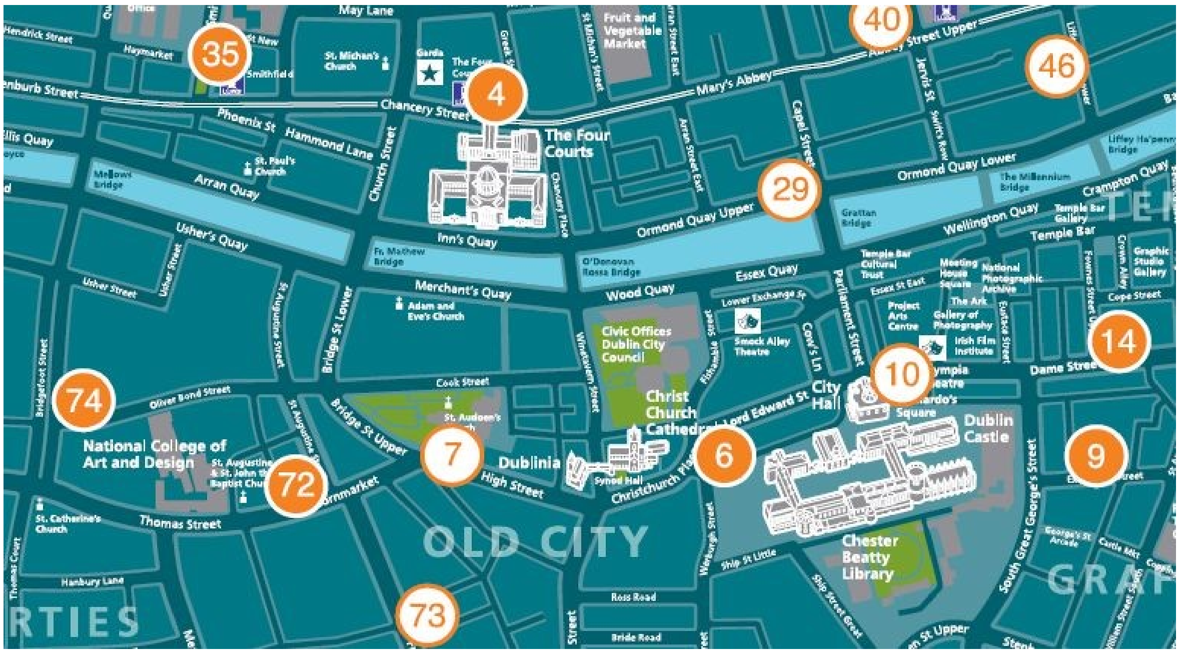


Figure 5.2 Dublin City Central Bike Stations (Just_Eat, 2018)

In Figure 5.2 we have illustrated some of the bike stations located around the famous spots for tourists and governmental organisations that people visit on daily basis. The reason for selecting station number 4 is due to being based near the Four Courts building and for station number 6 is because it is the only bike station near Christchurch Place and Dublin Castle.

In Figure 5.3 we have illustrated several of the main bike stations placed near the main attractions on Dublin city south side. We have selected station number 55 because it is located near Iveagh Gardens and the National Concert Hall. The reason for selecting station number 41 is due to its location being near to the Eye and Ear Hospital on Adelaide Road.

5.4 Analysis Methodology

We have aimed to model the peak usage hours to re-balance them in such a way that the morning peak hour bike stations reaching 0 or no available bike or even near to 0 are replenished by bikes from other stations that has reached or about to reach over flow status meaning no empty bike stand available. The selected morning peak-hours were from: 06:30 to 09:00 at an interval of every 10 minutes time period for the entire month of July 2019, as it is summer time and by taking advantage of long sunny days of summer Irish people and tourists prefer to visit historical places and spending time in Dublin city centre.

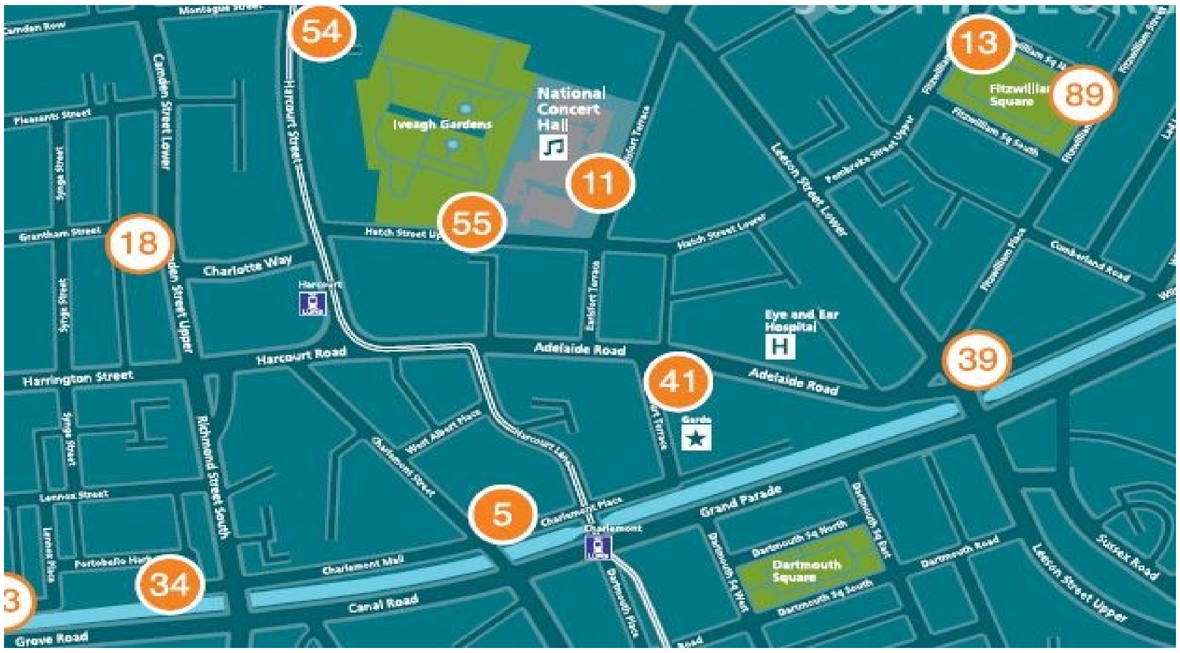


Figure 5.3 Dublin City South Bike Stations (Just_Eat, 2018)

We have calculated the withdrawal rate for the chosen sample bike stations using the formula below:

Withdrawal Rate Formula:

$$wr = \frac{\sum_{t=1}^{sf} (bw)_t}{sf} \tag{5.1}$$

Note:

- wr is the withdrawal rate to be calculated.
- bw is the number of bikes withdrawn.
- t is the time period.
- sf is the sample time frequency.

Using this formula we have calculated the withdrawal rates during rush hours and then added these rates to each other forming a total withdrawal rate for that station on a specific date.

Then we have calculated the miss opportunity or lost customers who did not find any available bike at the station during peak-hours using the formula mention below:

Missed Opportunity Formula:

$$mo = wr * t_n \tag{5.2}$$

Note:

- mo is the miss opportunity to be calculated.
- wr is the withdrawal rate calculate through Formula 5.1.
- t_n is the contiguous time period for which no bikes were available.

After the calculation of withdrawal rate we have calculated the miss opportunity for these bike stations or lost customers for example: the people who arrived during the time slots when no bikes were available and went back to take alternative transport such as Dublin Bus or Dublin Luas Service. Then we have calculated the total revenue/profit lost during peak-hours for every day (July 2019) for all six sample bike stations.

Lodgement Rate Formula:

$$lr = \frac{\sum_{t=1}^{sf} (br)_t}{sf} \tag{5.3}$$

Note:

- lr is the lodgement rate to be calculated.
- br is the number of bikes returned .
- t is the time period.
- sf is the sample time frequency.

We have calculated the lodgement rate or the bikes return rate for Dublin bike stations using the formula used similar to calculate the withdrawal rate but instead of bikes withdrawn the number of bikes added to the station was considered.

5.5 Calculations and Results

In this section, we present the in depth calculation that was performed on the actual data gathered from our CDM. Then using our defined analysis methodology we have drawn some insights that could be useful to develop a nearest stations pairing replenished strategy to minimise the miss opportunity rate and to increase the overall revenue that will allow the management later on to improve the overall service.

5.5.1 Withdrawal Rate

In Table 5.1 we have calculated and explained the withdrawal rate by using the withdrawal formula explained in in section 5.4. In column two you can see the actual number of bikes available at different time periods, then in column third we have calculated the withdrawn number of bikes in 10 minutes time frames. Then in column four you will see that total three withdrawals are performed whereas, the divisor 40, 20 and 10 are referring to the total number of minutes it took for these withdrawals such as 6:30 to 7:10 = 40 minutes, 7:30 to 7:50 = 20 minutes and 8:40 to 8:50 = 10 minutes. After, the total withdrawal calculation for the specific station the average withdrawal rate is calculated by dividing the total number of withdrawals occurred during 6:30 to 9:00.

Time Period	Available Bikes	Withdrawn Bikes	Withdrawal Rate	Average Withdrawal Rate
6:30:00	15	0		
6:40:00	10	5		
6:50:00	6	4		
7:00:00	1	5		
7:10:00	0	1		
7:20:00	0	0	(5+4+5+1) / 40 = 0.37	
7:30:00	5	0		
7:40:00	3	2		
7:50:00	0	3		
8:00:00	0	0		
8:10:00	0	0		
8:20:00	0	0	(2+3) / 20 = 0.25	
8:30:00	2	0		
8:40:00	0	2		
8:50:00	0	0		
9:00:00	0	0	(2) / 10 = 0.20	(0.37+0.25+0.20) / 3 = 0.27
Total	15	15	0.82	0.27

Table 5.1 Withdrawal Rate

5.5.2 Lodgement Rate

In Table 5.2 we present the lodgement rate calculated and explained in section 5.4. The second column of the table shows the actual number of bikes available at several time periods. In the third column we have calculated the number of bikes added or lodged during 10 minutes time interval to the bike station. In column four you will find the total three lodgements done thus, the divisor values 50, 20 and 40 are referring to the total number of minutes it took for each lodgement has performed such as 6:30 to 7:20 = 50 minutes, 7:40 to 8:00 = 20 minutes and 8:10 to 8:50 = 40 minutes respectively. Then in the last column the total average lodgement rate is calculated for the specific station on particular date by dividing the total number of withdrawals took place between 6:30 to 9:00.

Time Period	Available Bikes	Lodged Bikes	Lodgement Rate	Average Lodgement Rate
6:30:00	0	0		
6:40:00	1	1		
6:50:00	3	2		
7:00:00	6	3		
7:10:00	8	2		
7:20:00	9	1	(1+2+3+2+1) / 50 = 0.18	
7:30:00	9	0		
7:40:00	9	0		
7:50:00	12	3		
8:00:00	13	1	(3+1) / 20 = 0.20	
8:10:00	13	0		
8:20:00	17	4		
8:30:00	19	2		
8:40:00	22	3		
8:50:00	30	8		
9:00:00	30	0	(4+2+3+8) / 40 = 0.42	(0.18+0.20+0.42) / 3 = 0.26
Total	30	30	0.80	0.26

Table 5.2 Lodgement Rate

5.5.3 Missed Opportunity

The missed opportunity refers to the lost customers who arrived to a bike station and found zero or no bikes, the longer the station is empty the higher it will lose the customers. In Table 5.3 we have demonstrated the calculation technique to compute the lost customers or the missed opportunity. In Table 5.1 we have used the calculated average withdrawal rate to compute the missed opportunity if this withdraw rate continues even when the station is empty. The average withdrawal rate is multiplied with the stations zero or no bikes duration time period, meaning as long as a station stays empty till the next lodgement time, that duration is considered as a miss opportunity duration. In column six of Table 5.3 we have multiplied the average withdrawal rate with the zero bikes duration time periods which were from 7:10 to 7:20 = 10 minutes, 7:50 to 8:20 = 30 minutes and 8:40 to 9:00 = 20 minutes. The first missed opportunity shows that the station has lost 3 (2.7 rounded up) customers if the average withdrawal rate was continued even during the 10 minutes duration when the station became empty (zero available bikes). Similarly, the second missed opportunity shows the loss of 5 customers and the final one shows the loss of 5 customers as well.

Time Period	Available Bikes	Withdrawn Bikes	Withdrawal Rate	Average Withdrawal Rate	Missed Opportunity
6:30:00	15	0			
6:40:00	10	5			
6:50:00	6	4			
7:00:00	1	5			
7:10:00	0	1			
7:20:00	0	0	(5+4+5+1) / 40 = 0.37		(0.27) x 10 = 2.7
7:30:00	5	0			
7:40:00	3	2			
7:50:00	0	3			
8:00:00	0	0			
8:10:00	0	0	(2+3) / 20 = 0.25		(0.27) x 30 = 8.1
8:20:00	0	0			
8:30:00	2	0			
8:40:00	0	2			
8:50:00	0	0			
9:00:00	0	0	(2) / 10 = 0.20	(0.37+0.25+0.2) / 3 = 0.27	(0.27) x 20 = 5.4
Total	15	15	0.82	0.27	16.2

Table 5.3 Missed Opportunity

5.5.4 Alternative Public Transport Cost

The alternative public cost refers to the money spent by the system lost customers who arrived at the station and found no bikes and then took an alternative transportation option such as Dublin Bus service. The maximum fare of a single trip in Dublin Bus is 2.50 €. However, the fare could be less based on the customers' destination zone but we have considered the maximum fare because most of the commuters just tap their leap/travel cards which results in a maximum deduction of 2.50 €. In column six of Table 5.4 we have simply multiplied the calculated number of miss opportunities/ lost customers in column five with the maximum bus fare of 2.50 €.

Time Period	Available Bikes	Withdrawn Bikes	Withdrawal Rate	Missed Opportunity	Public Transport Cost (€)
6:30:00	15	0			
6:40:00	10	5			
6:50:00	6	4			
7:00:00	1	5			
7:10:00	0	1			
7:20:00	0	0	(5+4+5+1) / 40 = 0.37	(0.27) x 10 = 2.7	3 x 2.50 = 7.50
7:30:00	5	0			
7:40:00	3	2			
7:50:00	0	3			
8:00:00	0	0			
8:10:00	0	0			
8:20:00	0	0	(2+3) / 20 = 0.25	(0.27) x 30 = 8.1	8 x 2.50 = 20
8:30:00	2	0			
8:40:00	0	2			
8:50:00	0	0			
9:00:00	0	0	(2) / 10 = 0.20	(0.27) x 20 = 5.4	5 x 2.50 = 12.50
Total	15	15	0.82	16.2	40 €

Table 5.4 Public Transport Cost

5.5.5 Analysis Results and Observations

The results we have received after performing the proposed analysis on the selected bike stations for the entire month on sample bikes stations are explained below along with actual graphs.

Station no. 44 (Upper Sherrard Street)

In Figure 5.4 we have illustrated the withdrawal rates and lodgement rates calculated for station no. 44 which is located on the north side of Dublin city.

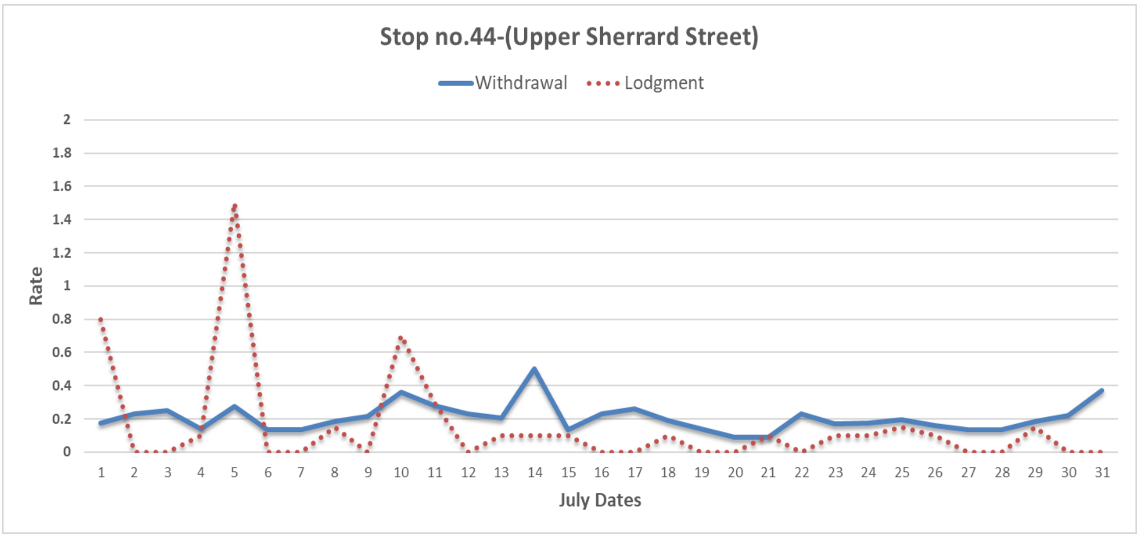


Figure 5.4 Station no.44 withdrawals and lodgements

The values on x-axis are showing all the dates from the 1st -July-2019 till 31st-July-2019, on the other hand the values on y-axis are showing the average withdrawal/lodgement rate per minute. In Figure 5.4 the blue continuous line series is showing the average withdrawal rate for the entire month and the red dotted line series is demonstrating the average lodgement rate during the morning rush hours for the entire month of July at Upper Sherrard Street bike station. However, the lodgement rate was high during the start of the month but then less after the first 10 days of the month which became the reason for most of the miss opportunities and revenue loss.

Station no. 60 (North Circular Road)

The station no. 60 is also located on the Dublin city north side and is close to the previous station no. 40. In Figure 5.5 you will see the withdrawal as well as the lodgement rates for the North Circular Road bike station.

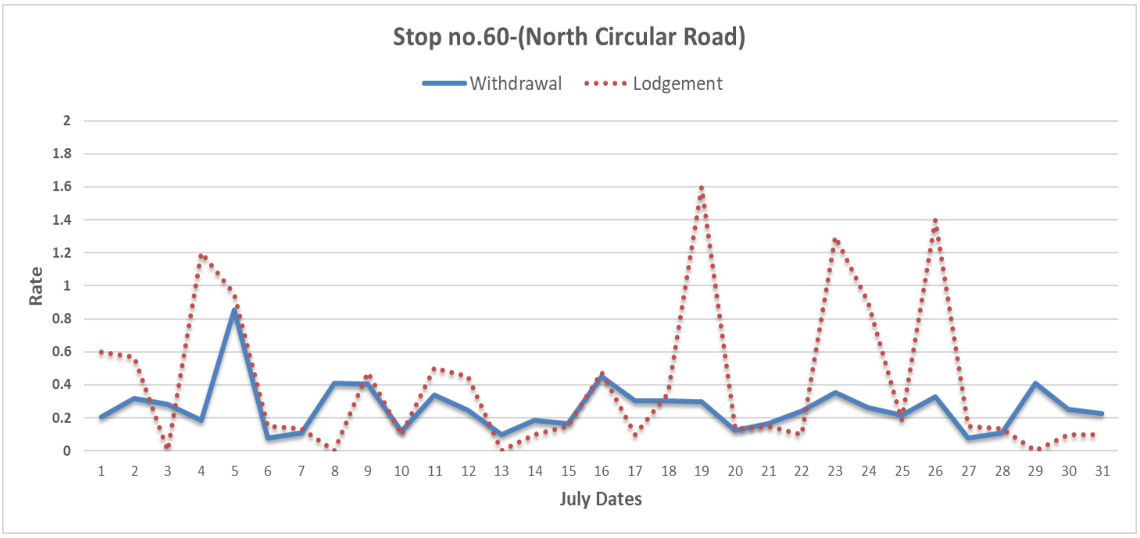


Figure 5.5 Station no.60 withdrawals and lodgements

In Figure 5.5 we have presented the withdrawal and lodgement rates for station no. 60, we have noticed that both of the rates were almost similar during the start of the month till the mid of July and then the lodgement rate becomes almost doubled on some of the days during the end of the month which means more bikes are available and caused less miss opportunity and revenue loss as compare to the nearby station no.44.

Station no. 6 (Christchurch Place)

This station is located in the Dublin city centre and is close to many popular tourist spots as well. In Figure 5.6 we have presented the average withdrawal and lodgement rates for the station no. 6 called the Christchurch Place bike station.

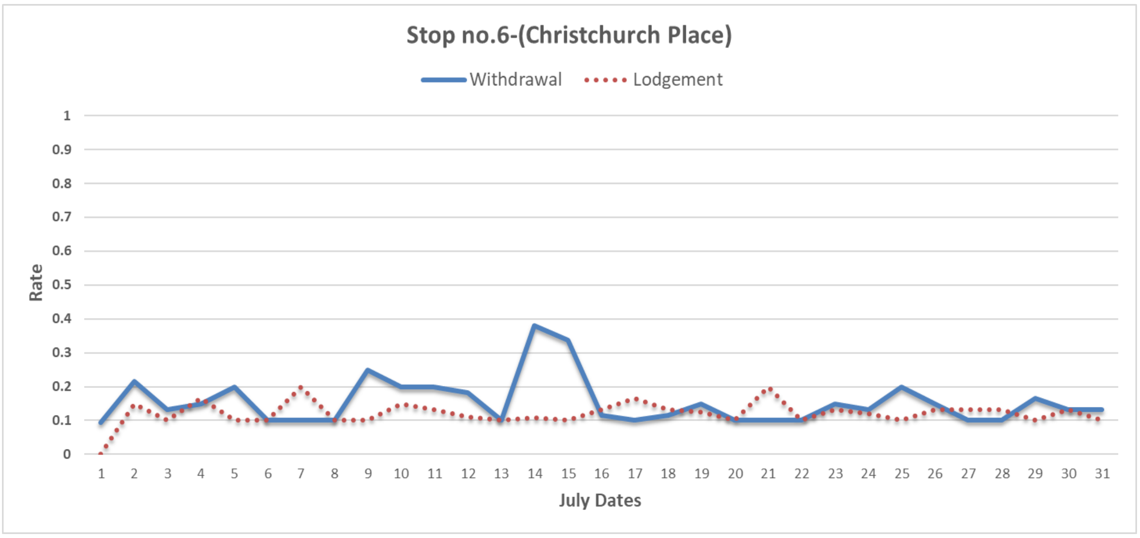


Figure 5.6 Station no.6 withdrawals and lodgements

In Figure 5.6 you will see the withdrawal and lodgement rates at station no.6 is quite different from the north side stations. We have observed that the trend is not same at this station, the withdrawal rates and lodgement rates was almost similar which shows that not many bikes are used at this station and less service customers were lost due to the availability of bikes most of the time.

Station no. 4 (Greek Street)

The station no. 4 is nearby the previous station no.6 in the same location of Dublin city centre. In Figure 5.7 you will see the average withdrawal and lodgement rates for the station no.4 also named as Greek Street bike station.

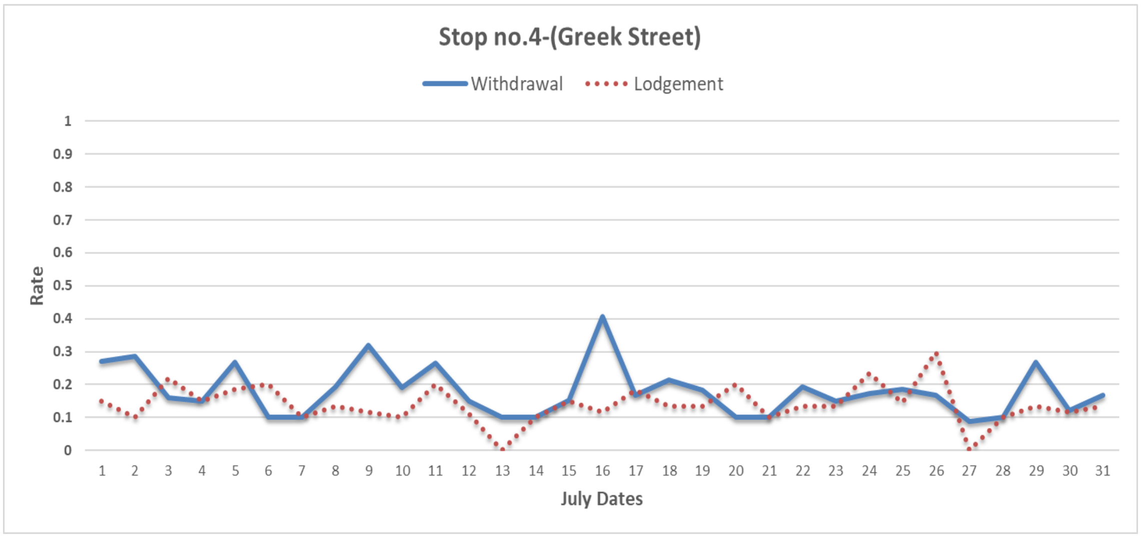


Figure 5.7 Station no.4 withdrawals and lodgements

The withdrawal rates and lodgement rates at this station are quite similar to each other. In addition, similar trend has been observed on station no.4 as it was observed on the previous nearby station no.6. Due to the reason of less withdrawal rate bikes were available mostly and there was minimum chance of miss opportunity and revenue loss.

Station no. 41 (Harcourt Terrace)

In Figure 5.8 we have presented the withdrawal and lodgement rates calculated for station no. 41 called the Harcourt Terrace bike station which is located at the south side of Dublin city.

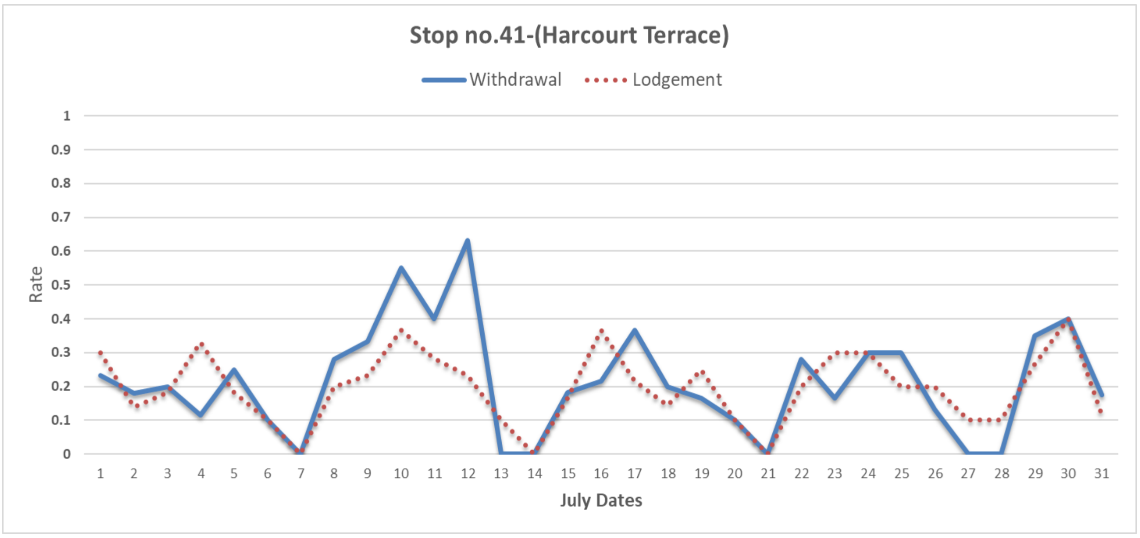


Figure 5.8 Station no.41 withdrawals and lodgements

The Harcourt Terrace bike station withdrawal and lodgement rates are very similar to each other. However, this bike station showed slightly different overall trend of withdrawals and lodgements in comparison to the city centre stations. The withdrawal rate on this station was high but at the same time lodgement rate was high too which became the reason of minimum lost opportunity and profit loss.

Station no. 55 (Hatch Street)

In Figure 5.9 you will see the station no. 55 called the Hatch Street bike station, the data of withdrawals and lodgements for the month of July is depicted, and this station is situated close to the previous station 41.

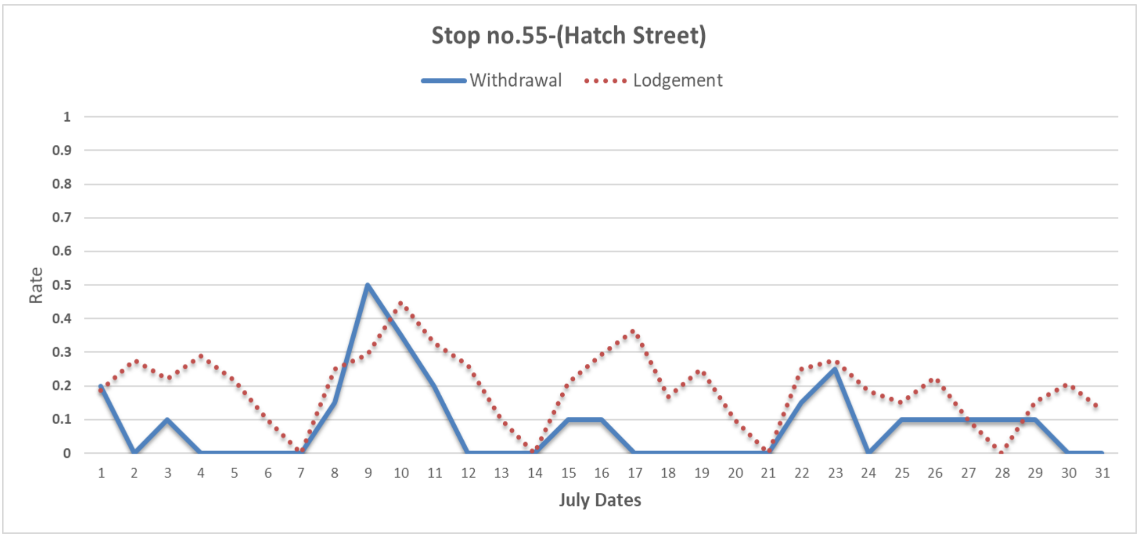


Figure 5.9 Station no.55 withdrawals and lodgements

The withdrawal rates and lodgement rates at this station no. 55 is following the same trend as it was observed at station no. 41. The least number of miss opportunity was found at this station due to the higher lodgement rate and lower withdrawal rate which became the reason of minimum users lost and less profit loss.

5.5.6 Overall Month Results

The average results we have received for the entire month of July 2019 after performing the proposed analysis on the selected bike stations are illustrated below in Table 5.5.

Station ID	Station Name	Peak Hours	Month	Average Withdrawal Rate-(per minute)	Average Lodgement Rate-(per minute)	Miss Opportunity	Lost Profit (€)
44	Upper Sherrard Street	6:30-9:00	July 2019	6.42	4.65	296.4	747.5
60	North Circular Road	6:30-9:00	July 2019	8.10	12.55	237.85	600
6	Christchurch Place	6:30-9:00	July 2019	4.78	3.76	110.29	280
4	Greek Street	6:30-9:00	July 2019	5.58	4.31	19.5	50
41	Harcourt Terrace	6:30-9:00	July 2019	6.61	6.08	10	25
55	Hatch Street	6:30-9:00	July 2019	2.6	6.03	2	5

Table 5.5 July 2019 Overall Results

In Table 5.5 we have presented the calculated the bikes data for the entire month of July 2019 for the chosen bike stations, these calculations include average withdrawal rate per minute, average lodgement rate per minute, total miss opportunities and lost profits.

Bike Station 44 and 60

These stations are located on the Dublin City North-side, the bikes withdrawal rate was higher at station 60 as compared to station 44 but most customers and profit was lost at station number 44 instead of station 60. However, the interesting thing we have noticed here is that in addition to a higher withdrawal rate the lodgement rate was almost doubled at station 60 as compared to the nearby station number 40 due to which less customers and profit is lost at station 60.

Bike Station 6 and 4

These stations are situated in the heart of the Dublin City, the withdrawal rate of bikes at station number 4 was higher than station number 6 but after having a high withdrawal rate most of the lost customers and profit was found at station number 4 having less withdrawal rate. The reason for this unexpected loss of higher customers and profit after still having less withdrawal rate at station 4 was due to the higher lodgement rate on nearest bike station number 6.

Bike Station 41 and 55

The stations 41 and 55 are located on Dublin City South-side, the bikes withdrawal rate was much higher at station 41 as compare to station 55 due to which more customers and profit was lost. However, we observed that the lodgement rate on both stations was almost similar but the withdrawal rate at station number 55 was also quite less which becomes the reason for least customers and revenue loss. The station number 41 has the highest withdrawal rate after the station number 60 located on the Dublin City North-side.

5.5.7 Conclusion

It has been observed that the bike stations located on the Dublin city north side are much busier in the morning as compared to the bike stations located in the centre and south side of the city. However, this morning trend at north stations was noticed to be opposite during the evening rush hours. Also, we have noticed that the flow of bikes during the morning rush hours is from north towards the centre and then later on to the south side. However, this movement starts to become opposite from the south side towards the north side during the evening hours.

Our derived insights regarding the bikes usage frequency during the rush hours could be sufficient to convenience the bikes management team to perform re-balancing strategies through implementing a pairing technique by creating pairs of the nearest mostly used and least used stations, and through performing the above analysis for all the bike stations for entire year.

Through this pairing technique it is possible to re-balance the busiest bike stations during the rush hours quickly and efficiently by replenishing bikes from the nearest least busy station, this technique will minimise the man power and fuel consumption required by re-balancing vehicles. The time efficiency will also be increased and will help to raise the overall revenue generation from those busiest stations that are not getting replenish during the peak hours, this maximum revenue generation will facilitate the service provider to enhance the overall quality of the service and user experience.

5.5.8 Limitations

Due to the time limitation for the project we were unable to perform the analysis for all the 102 bike stations over the period of 12 months/year and the analysis is only limited to the sample stations selected from the overall population dataset of bike stations. In addition, the analysed data is limited to the busiest summer month of July due to the project time restrictions.

5.5.9 Future Work

We will extend our work to perform the analysis across all the 102 bike stations for the entire period of 12 months. Our plan also includes to perform a comparison analysis of normal days usage with the lock-down days caused due to COVID-19 outbreak. In addition, we might pull another useful dataset to perform cross domain analysis on the data from other services such as Dublin Bus, Dublin Luas, Dublin Motorways and Environmental data stored in our Canonical Data Model to leverage some other new insights that allows to check the impact of these transport services on each other or the effect of weather on these services alone, this will also facilitate the management to further improve the quality by extending the services to the next level and stepping further towards building Dublin a Smart City.

Chapter 6

Summary

In this chapter we summarise the overall work performed in this research project. We discuss the addressed research hypothesis, the goals we set out to accomplish and the milestones achieved during this research project. Also, we explain the developed Extract, Load and Transform (ELT) framework, the query and visualisation service. Finally, we discuss the resource allocation analysis performed on the Dublin Bikes data and the limitations of the work done. In addition, we lay out our future research work plan for the possible direction of the current work done.

6.1 Research Work Overview

In this research work, we have addressed the problem of integrating isolated datasets of Dublin City sourced from various domains and containing heterogeneous data formats. We have developed a robust and automated Extract, Load and Transform (ELT) framework that performs the integration operations for the Dublin Smart City heterogeneous datasets through the identification of facts, measures and dimensions of these provided data sources.

Our Research work has addressed our proposed research hypotheses:

> Hypothesis 1: The increased automation of an Extract, Load and Transform framework from a non-automated or semi-automated approach can better facilitate the integration and contextual enrichment of diverse data sources.
>
> Hypothesis 2: The development of an efficient resource allocation strategy for the Dublin Bikes Sharing Scheme can facilitate in identifying the overflow and underflow bike stations and help minimise the number of lost customers at underflow bike stations and increase revenue at those stations.

The main problems addressed in this research work are:

1. The study of the algorithms and methods for the development of an Extract, Load and Transform (ELT) framework.
2. The study of the techniques and tools used for the data integration.
3. The development of a robust Extract, Load and Transform (ELT) architecture from scratch for Dublin Smart City which was not available before to facilitate a cross domain analysis.
4. The construction of automated data lake and Canonical Data Model (CDM).
5. The development of algorithms to perform the structural enrichment, contextual and augmentation enrichment towards the provided data streams..
6. The development the automated mapping rules between the data lake and actual data sources, and between the CDM and our data lake.
7. The development of a query service to facilitate secure and fast communication with our CDM.
8. The development of a Dynamic Data Visualisation Website called Dublin Bike Peak times to facilitate the exploration, identification and analysis of patterns and trends in the Dublin Bikes data.
9. The development of a resource allocation strategy to identify overflow and underflow bike stations in order to estimate the number of potential lost customers and revenue at bike stations.

The developed Extract, Load and Transform (ELT) framework will first generate the individual schema for each provided data source using the automated rules. Then the processes will keep all the constructed data models in parallel to form a data lake. After the construction of data lake the analysis is carried out on the original data sources and on the data lake in search of relevant and useful textual or numeric information required to construct the Canonical Data Model (CDM). After the development of the CDM with the help of the derived automated mapping rules the data is populated into our data lake and then from data lake to our CDM.

Moreover, the ELT framework will perform the structural enrichment and contextual enrichment of the data streams. As a part of our ELT framework we have designed several algorithms to perform augmentation enrichment by applying data aggregation operations on the provided data streams and calculating useful information. One of the major functionality of the designed ELT framework is to ingest the data models from the data lake into CDM based on the selected facts and dimensions. The ELT contains several algorithms each dedicated for specific task such as: structural formation, data population, data cleansing, contextual enrichment, data transformation, augmentation enrichment and data integration. We have implemented this ELT architecture specifically on the Dublin Transport domain that includes: Dublin Bus, Dublin Bikes, Dublin Luas (Tramline), Dublin Motorways and Dublin Weather infrastructures.

Given the availability of the huge size of data in our data repository we have decided to focus on the Dublin Bikes data because of the time limitation provided for the research and we have considered it a good opportunity to reveal some useful insights about this real life public transport service. Another reason is the bikes data is changing all the time and gave us the motivation to build an efficient economical model. In order to access the Dublin Bikes data securely and quickly we have developed a query service such as an Application Programmable Interface (API) service using the latest technology and tools. This service enables users to communicate with our Dublin Smart City data repository as well as with the smart city applications. The query service will prevent the direct communication with the CDM and will work as a transporter between the user and the data.

To observe the newly obtained knowledge and demonstrate it in a compelling visual format we have developed a visualisation service for Dublin bikes sharing scheme and name it **Dublin Bikes Peak Times** website to demonstrate the new obtained knowledge. This web application allows the user to perform data analysis on the various bike stations during different times. The application contains several tabs to see trends for one day, one week, for a specific day in last ten weeks, for all seven days together in a specific week and heat maps to see the stations bicycles availability. The website will communicate with the data in CDM by only using our query service (API). With the help of the web application we have identified some unusual data patterns. These data trends has facilitate us to get a better understanding of the user behavior on each bike station and help us to develop an economical model for efficiency and suggest a better replenishing strategy.

The resource allocation analysis conducted on Dublin Bikes has helped us to reveal the unseen insights in order to improve the replenishing strategy. While, it was observed that the bikes stations situated on the city North-side are in heavy demand during the morning rush hours and a reverse trend is observed on the South-side during the evening peak hours. For example, the total revenue lost in the month of July 2019 on the bike station no. 44 (Upper Sherrard Street) was 747.5 euro which is situated on the North-side and bike station no.55 (Hatch Street) was 5 euro which is on the South-side. After these derived insights we have suggested a pairing strategy to replenish the bikes from the most nearest less used bike station to minimise the revenue loss.

6.2 Scientific Novelty and Practical Contribution

In this work, we have presented the processes that will analyse the provided data streams and by the identification of facts and dimensions it will generate the schemata for the provided data sources. We have developed the processes for the automated construction of our Canonical Data Model (CDM) by analysing the already built data models along with the live data streams. We have specified the rules to perform the data transformation and the data cleansing operations in order to remove the heterogeneity available within the data types and any unwanted semantics.

We have presented the processes to perform the relevant and useful contextual enrichment towards the data streams. The automated mapping rules are specified between the data sources and schemata, and then the mapping rules between the schemata and the CDM. The algorithms are designed to perform the augmentation enrichment towards various datasets residing in our data lake. In addition, we have built a query service that will enable the communication between the user and the data residing in our Canonical Data Model. Our query service will provide an extra layer of security by eliminating the direct communication of users with our CDM. We have designed a visualisation service for Dublin Bikes to demonstrate the newly obtained insights and knowledge in a compelling visual format. Lastly, an economical model was introduced for Dublin Bikes Sharing scheme to bring efficiency in the system by suggesting a pairing replenishing strategy.

Our research work is performed on the live transport data streams (real-time) that are constantly changing every minute or even in seconds, For example: the bus arrival times, the tram arrival times or the number of bikes available at a bike station. The ELT framework has the advantage to ingest new dataset and will evolve our CDM. We can apply our framework to other smart cities in the world using their open datasets by applying only small changes to our existing framework. Our framework will allow the Dublin Smart City and other smart cities around the world to perform cross-domain analysis and will facilitate them to address the contemporary problems of the city. The developed economical model and pairing replenishing strategy can be applied to other smart cities and can facilitate them to bring efficiency in their systems.

6.3 Limitations

This research work contains some limitations due to the factor of given time period to complete this research project because of which it was not possible to carry out the proposed resource allocation analysis to all one hundred and two (102) bike stations located within the sharing scheme and then to extend this analysis to the entire year for each single bike station. So, it was only performed on the selected sample size drawn from the total population. Moreover, the analysis was only performed on the single busiest month of the summer of 2019, this analysis could be extended to the whole year.

In the given amount of time for this project it was not possible to expand the analysis to all the Dublin bikes stations we were able to apply the developed economical model to a few bike stations. If more time were available we could perform it on all the stations for every month.

6.4 Future Work

Our future plan is to expand the Dublin Bikes resource allocation analysis to all the Bike stations in the system for the entire year to provide a detailed picture on the usage trends over the year at every single station. There are several analysis that could be of great interest to us in the future research such as: we might pull some additional datasets from our repository to perform analysis on other data as well such as Dublin Bus service, Dublin Tramline service, Dublin Motorways and Dublin weather data from our data lake to derive new and additional insights and knowledge. We could observe the impact of these infrastructure on each other or the effect of weather on them to enable the authorities to further enhance the service quality to its best and will push the city a step forward to become a Dublin Smart City.

References

Apache_Echarts (2020) *Echarts* [Online]. Available at: https://www.npmjs.com/package/echarts (Accessed: 9 December 2019)

Batini, C., Lenzerini, M. and Navathe, S.B. (1986) A comparative analysis of methodologies for database schema integration *ACM computing surveys (CSUR)* **18**(4), pp. 323–364

Bergamaschi, S., Castano, S. and Vincini, M. (1999) Semantic integration of semistructured and structured data sources *ACM Sigmod Record* **28**(1), pp. 54–59

Bourhis, P., Reutter, J.L. and Vrgoč, D. (2020) Json: Data model and query languages *Information Systems* **89**, p. 101478

Braunstein, M.L. (2018) *Health Informatics on FHIR: How HL7's New API is Transforming Healthcare* Springer

Caggiani, L., Camporeale, R., Marinelli, M. and Ottomanelli, M. (2019) User satisfaction based model for resource allocation in bike-sharing systems *Transport Policy* **80**, pp. 117–126

Caulfield, B. (2014) Re-cycling a city–examining the growth of cycling in dublin *Transportation research part A: policy and practice* **61**, pp. 216–226

Census (2016) *Central Statistics Office* [Online]. Available at: https://www.cso.ie/en/media/csoie/newsevents/documents/census2016summaryresultspart1/Census2016SummaryPart1.pdf (Accessed: 13 November 2019)

Center, I.I. (2013) Big data visualization: Turning big data into big insights *White Paper* pp. 1–14

CitiBike (2013) *New York City's Bike Paths, Bike Lanes Greenways* [Online]. Available at: http://www.nycbikemaps.com/spokes/citi-bike-2013-summary/ (Accessed: 25 March 2020)

CitiBike (2019) *Citi Bike System Data* [Online]. Available at: http://www.citibikenyc.com/system-data (Accessed: 25 March 2020)

Climate, W. (2019) *Weather and Climate information for every country in the world.* [Online]. Available at: https://weather-and-climate.com/average-monthly-Rainfall-Temperature-Sunshine,Dublin,Ireland (Accessed: 13 November 2019)

Dublin_Bus_RTPI (2020) *Dublin Bus Mobile RTPI* [Online]. Available at: http://www.dublinbus.ie/DublinBus-Mobile/RTPI-Stops/?routeNumber=27&Towards=Jobstown&From=Clare%20Hall&Direction=I (Accessed: 5 March 2019)

Dublin_City_Council (2019) *Dublin City Council* [Online]. Available at: https://www.dublincity.ie/residential/transportation/cycling-dublin-city/just-eat-dublinbikes (Accessed: 7 March 2019)

Dublinked (2018) *Dublinked Open Data Store* [Online]. Available at: https://data.smartdublin.ie/ (Accessed: 21 October 2018)

Dublin_Luas (2020a) *Luas Forecasting API* [Online]. Available at: https://data.gov.ie/dataset/luas-forecasting-api (Accessed: 15 March 2019)

Dublin_Luas (2020b) *Tallaght Luas Stop* [Online]. Available at: https://luas.ie/tallaght.html (Accessed: 17 March 2019)

ECMAScript (2020) *ECMAScriptR©2020 Language Specification* https://www.ecma-international.org/ecma-262/ Accessed: January 21 2020

EPA_API (2020) *Environmental Protection Agency Air Quality API* [Online]. Available at: https://www.epa.ie/air/quality/dev/ (Accessed: 28 March 2019)

EPA_Quality (2019) *Environmental Protection Agency Quality Index* [Online]. Available at: https://www.epa.ie/air/quality/index/ (Accessed: 28 March 2019)

EPA_Stations (2020) *Environmental Protection Agency Stations* [Online]. Available at: https://www.epa.ie/air/quality/data/ (Accessed: 28 March 2019)

Fricker, C. and Gast, N. (2016) Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity *Euro journal on transportation and logistics* **5**(3), pp. 261–291

Frittella, S., Manoorkar, K., Palmigiano, A., Tzimoulis, A. and Wijnberg, N. (2020) Toward a dempster-shafer theory of concepts *International Journal of Approximate Reasoning* **125**, pp. 14–25

Fusco, G. and Aversano, L. (2020) An approach for semantic integration of heterogeneous data sources *PeerJ Computer Science* **6**, p. 254

Gaur, A., Scotney, B.W., Parr, G.P. and McClean, S.I. (2015) Smart city architecture and its applications based on iot. in: *ANT/SEIT* pp. 1089–1094

Geels, F. and Raven, R. (2006) Non-linearity and expectations in niche-development trajectories: ups and downs in dutch biogas development (1973–2003) *Technology Analysis & Strategic Management* **18**(3-4), pp. 375–392

JCDecaux (2019) *JCDecaux Developer* [Online]. Available at: https://developer.jcdecaux.com/#/opendata/vls?page=getstarted (Accessed: 10 March 2019)

Just_Eat (2018) *Just Eat dublinbikes station list / View stations / Stations / Dublin* [Online]. Available at: http://www.dublinbikes.ie/Stations/View-stations/Just-Eat-dublinbikes-station-list (Accessed: 10 March 2019)

Just_Eat (2019) *Just Eat dublinbikes - 2019 figures! / Reports / Magazine / Dublin* [Online]. Available at: http://www.dublinbikes.ie/Magazine/Reports/Just-Eat-dublinbikes-2019-figures (Accessed: 7 March 2019)

Khan, M. and Khan, S.S. (2011) Data and information visualization methods, and interactive mechanisms: A survey *International Journal of Computer Applications* **34**(1), pp. 1–14

Khan, Z., Anjum, A. and Kiani, S.L. (2013) Cloud based big data analytics for smart future cities in: *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing* pp. 381–386 IEEE

Khan, Z. and Kiani, S.L. (2012a) A cloud-based architecture for citizen services in smart cities in: *2012 IEEE Fifth International Conference on Utility and Cloud Computing* pp. 315–320 IEEE

Khan, Z., Ludlow, D., McClatchey, R. and Anjum, A. (2012b) An architecture for integrated intelligence in urban management using cloud computing *Journal of Cloud Computing: Advances, Systems and Applications* **1**(1), p. 1

Kisilevich, S., Mansmann, F., Nanni, M. and Rinzivillo, S. (2010) *Spatio-temporal clustering* pp. 855–874 Springer US, Boston, MA

Li, D., Mei, H., Shen, Y., Su, S., Zhang, W., Wang, J., Zu, M. and Chen, W. (2018) Echarts: A declarative framework for rapid construction of web-based visualization *Visual Informatics* **2**(2), pp. 136–146

Martins, A., Martins, P., Caldeira, F. and Sá, F. (2020) An evaluation of how big-data and data warehouses improve business intelligence decision making in: *Trends and Innovations in Information Systems and Technologies*, (Eds.) Á. Rocha, H. Adeli, L.P. Reis, S. Costanzo, I. Orovic and F. Moreira pp. 609–619 Springer International Publishing, Cham

Mitton, N., Papavassiliou, S., Puliafito, A. and Trivedi, K.S. (2012) Combining cloud and sensors in a smart city environment

Multinomial (2002) *Multinomial Logistic Regression using SPSS Statistics* [Online]. Available at: https://statistics.laerd.com/spss-tutorials/multinomial-logistic-regression-using-spss-statistics.php#:~:text=Introduction,with%20more%20than%20two%20categories. (Accessed: 1st November 2019)

Naphade, M., Banavar, G., Harrison, C., Paraszczak, J. and Morris, R. (2011) Smarter cities and their innovation challenges *Computer* **44**(6), pp. 32–39

National_Transport_Authority (2019) *National Transport Authority* [Online]. Available at: https://www.nationaltransport.ie/wp-content/uploads/2019/08/Bus_and_Rail_Statistics_2019.pdf (Accessed: 15 March 2019)

Oliveira, G.N., Sotomayor, J.L., Torchelsen, R.P., Silva, C.T. and Comba, J.L. (2016) Visual analysis of bike-sharing systems *Computers & Graphics* **60**, pp. 119–129

Pal, A. and Zhang, Y. (2017) Free-floating bike sharing: Solving real-life large-scale static rebalancing problems *Transportation Research Part C: Emerging Technologies* **80**, pp. 92–116

Petrović, M., Vučković, M., Turajlić, N., Babarogić, S., Aničić, N. and Marjanović, Z. (2017) Automating etl processes using the domain-specific modeling approach *Information Systems and e-Business Management* **15**(2), pp. 425–460

Puiu, D., Barnaghi, P., Toenjes, R., Kümper, D., Ali, M.I., Mileo, A., Parreira, J.X., Fischer, M., Kolozali, S., Farajidavar, N. *et al.* (2016) Citypulse: Large scale data analytics framework for smart cities *IEEE Access* **4**, pp. 1086–1108

Python_Hug (2019) *Embrace the APIs of the future* [Online]. Available at: https://www.hug.rest/ (Accessed: 2 January 2020)

Quix, C., Hai, R. and Vatov, I. (2016) Metadata extraction and management in data lakes with gemms *Complex Systems Informatics and Modeling Quarterly* (9), pp. 67–83

Raghavan, S., Simon, B.Y.L., Lee, Y.L., Tan, W.L. and Kee, K.K. (2020) Data integration for smart cities: Opportunities and challenges in: *Computational Science and Technology* pp. 393–403 Springer

Rani, M. and Vyas, O. (2017) Smart bike sharing system to make the city even smarter in: *Advances in Computer and Computational Sciences* pp. 43–55 Springer

Rietveld, P. and Daniel, V. (2004) Determinants of bicycle use: do municipal policies matter? *Transportation Research Part A: Policy and Practice* **38**(7), pp. 531–550

Santandar_Cycles (2020) *Santander Cycles London* [Online]. Available at: https://tfl.gov.uk/modes/cycling/santander-cycles (Accessed: 10 May 2019)

Scriney, M. (2018) Constructing data marts from web sources using a graph common model Ph.D. Thesis Dublin City University

Scriney, M., McCarthy, S., McCarren, A., Cappellari, P. and Roantree, M. (2019) Automating data mart construction from semi-structured data sources *The Computer Journal* **62**(3), pp. 394–413

Scriney, M., O'Connor, M.F. and Roantree, M. (2017) Integrating online data for smart city data marts in: *British International Conference on Databases* pp. 23–35 Springer

Shaheen, S.A., Guzman, S. and Zhang, H. (2010) Bikesharing in europe, the americas, and asia: past, present, and future *Transportation Research Record* **2143**(1), pp. 159–167

da Silva, W.M., Alvaro, A., Tomas, G.H., Afonso, R.A., Dias, K.L. and Garcia, V.C. (2013) Smart cities software architectures: a survey in: *Proceedings of the 28th Annual ACM Symposium on Applied Computing* pp. 1722–1727

Skoutas, D. and Simitsis, A. (2007) Ontology-based conceptual design of etl processes for both structured and semi-structured data *International Journal on Semantic Web and Information Systems (IJSWIS)* **3**(4), pp. 1–24

SmartDublin (2019a) *JCDecaux Dublin Bikes* [Online]. Available at: https://data.smartdublin.ie/dataset/dublinbikes-api (Accessed: 10 March 2019)

SmartDublin (2019b) *Real-time Passenger Information (RTPI) for Dublin Bus, Bus Eireann, Luas and Irish rail* [Online]. Available at: https://data.smartdublin.ie/dataset/real-time-passenger-information-rtpi-for-dublin-bus-bus-eireann-luas-and-irish-rail (Accessed: 1st March 2019)

Späth, P. and Friesen, J. (2020) Working with xml and json documents in: *Learn Java for Android Development* pp. 641–697 Springer

TII_Road_Networks (2019) *Ireland National Road Network* [Online]. Available at: https://www.tii.ie/roads-tolling/our-road-network/ (Accessed: 21 April 2019)

TII_Traffic (2020) *Dublin Motorways Travel Times* [Online]. Available at: https://www.tiitraffic.ie/travel_times (Accessed: 20 March 2019)

TII_Weather (2020) *Transport Infrastructure Ireland: Weather* [Online]. Available at: https://www.tiitraffic.ie/weather (Accessed: 26 March 2019)

Transit_Maps (2017) *Official Map – Luas Light Rail Network, Dublin, Ireland* [Online]. Available at: https://www.transitmap.net/dublin-luas-2017/ (Accessed: 15 March 2019)

Transport_For_Ireland (2019a) *About Luas - Dublin's Tram Service* [Online]. Available at: https://www.transportforireland.ie/getting-around/by-tram/about-luas/ (Accessed: 15 March 2019)

Transport_For_Ireland (2019b) *General Transit Feed Specification* [Online]. Available at: https://www.transportforireland.ie/transitData/PT_Data.html (Accessed: 1st March 2019)

Transport_Infrastructure_Ireland (2020) *Traffic Count Data* [Online]. Available at: https://www.tii.ie/roads-tolling/operations-and-maintenance/traffic-count-data/ (Accessed: 21 April 2019)

Vue_Framework (2020) *Vue.js* [Online]. Available at: https://vuejs.org/ (Accessed: 21 November 2019)

XML_Path_Language (2017) *XML Path Language (XPath) 3.1* [Online]. Available at: https://www.w3.org/TR/2017/REC-xpath-31-20170321/ (Accessed: 28 March 2019)

Yong, L. and Rongxu, H. (2010) About the sensing layer in internet of things [j] *Computer Study* **5**, p. 55

Appendix A

Instructions Manual to run Extract, Transform and Load (ELT) framework

A.1 Download and Install Database

- Download PostgresSQL from this link and select the 12.1 Windows x86-64 to download. https://www.enterprisedb.com/downloads/postgres-postgresql-downloads
- After downloading install the PostgreSQL on your system. While installing the database when asked to provide the server password for **postgres** account which is the main role remember to provide the password **'SmartDublin2020'** because all the python scripts will use this password to connect to the main database role **'postgres'** later on for schema building and for data insertion.

A.2 Download and Install Anaconda

- Once, the database is installed successfully then download Anaconda from this link https://www.anaconda.com/products/individual, and then select Python 3.8 (or higher) Version-(64 Bit Graphical Installer) to download on your PC.
- Create a folder with any name in C: or in any other hard drive where you want to install the anaconda software and Install Anaconda it in that directory

A.3 Install Python Libraries

A.3.1 Manual Graphical User Interface (GUI) Method

1. Once you have installed the Anaconda open the Anaconda navigator and click on the environments on the left hand side you will see all the libraries, on top you will see a drop-down menu select not-installed this will show all the not installed libraries.
2. There will be a search box beside type **’psycopg2’** if not installed selected the checkbox beside this library and click on Apply button it will take a few moments to install this library. Now do the same for these libraries too:
   - Selenium
   - Requests
   - Beautifulsoup4
   - Jsonlib
   - Urllib3
   - Logging
   - Threaded
   - Urllib
   - Numpy
   - Lxml
   - py-xml
   - Shapely
   - geometry

   After installing these libraries now close anaconda navigator.

A.3.2 Manual Command Prompt (CMD) Method

1. Open anaconda command prompt and type **’pip install psycopg2’** it will install the library if not already installed. Now do the same for these libraries too by typing these commands one by one:
   - pip install –U selenium
   - pip install requests

- pip install beautifulsoup4
- pip install jsonlib
- pip install urllib3
- pip install logging
- pip install threaded
- pip install urllib
- pip install numpy
- pip install lxml
- pip install py-xml
- pip install shapely
- pip install geometry.

After installing these libraries now close anaconda command prompt.

A.3.3 Automated Method

1. I have created a requirements.txt file which contains all the python libraries you just need to run this file in order to install all the python libraries on your PC by using this command **'pip install –user –requirement requirements.txt'**.

   For more information visit:
   https://medium.com/@boscacci/why-and-how-to-make-a-requirements-txt-f329c68 5181e

A.4 Executing the ELT framework

A.4.1 Folder Creation and Files Extraction

- Now unzip or extract the zip file naming 'Transport_Automated_Full_Version.zip' in C drive 'C:\'

A.4.2 Open Database

- From the windows start menu type **'pgadmin'** and click to open it will take a while to open your newly installed postgres database. Click on **'Servers'** and provide the password which you have provided earlier during the installation phase **'Smart-Dublin2020'**. Click on **PostgreSQL 12** and then on **Databases** your new database will appear here shortly.

A.4.3 Run Anaconda

- From start menu write anaconda prompt and you will see the black DOS-environment. Now change the directory to the newly extracted folder by using command:
  'cd\ Automated_Full_Version'.

  When the directory is changed type **'python CreateDatabase.py'** by executing this command the **'Dublin_Smart_City'** database will be created.

A.4.4 Create Environmental Schema

- Now change directory to:
  'cd\Transport_Automated_Full_Version\Dublin_Air_Pollution' when you see the directory is changed then type **'python Dublin_Air.py'** this file will execute the process of building schema for Dublin Air Quality and weather. The ERD folder in Air Pollution folder have the Entity Relationship Diagram (ERD) of Dublin Environmental schema that will be produced after running the script.

A.4.5 Create Bikes Schema

- Again change the directory to:
  'cd\Transport_Automated_Full_Version\Dublin_Bikes' when you see the directory is changed then type **'python Dublin_Bikes.py'** this file will generate the schema for Dublin Bikes. The ERD folder in Dublin Bikes folder contains an ERD of Dublin Bikes schema.

A.4.6 Create Bus Schema

- For Dublin Bus you have to change the directory to Dublin_Bus folder like
  'cd\Transport_Automated_Full_Version\Dublin_Bus'
  when the directory is changed to Dublin bus folder then type **'python Dublin_Bus.py'** this file will generate the schema for Dublin Bus. There will be a folder named ERD in the Dublin Bus folder representing the Dublin Bus Schema ERD.

A.4.7 Create Luas Schema

- Now change the directory to Dublin_Luas folder
  ‘cd\Transport_Automated_Full_Version\Dublin_Luas’
  and execute the file by running **‘python Dublin_Luas.py’** to construct the schema for Dublin Luas. The ERD folder of Dublin Luas folder will show you the ERD diagram of Dublin Luas schema.

A.4.8 Create Electoral Schema

- Then change the directory to Dublin_Electoral folder
  ‘cd\Transport_Automated_Full_Version\Dublin_Electoral’
  when the directory is changed to this location then run **‘python Dublin_ED.py’** this will create the schema for Dublin census data. When you will open the ERD folder in Dublin Electoral folder it will allow you to see the ERD produced as a result of running the above script.

A.4.9 Create Motorways Schema

- Change the directory to Dublin_M50 folder using
  ‘cd\Transport_Automated_Full_Version\Dublin_M50’
  command and then run **‘python Dublin_Traffic.py’** this script will create the schema for Dublin Motorway roads. After running the M50 schema script you will be able to navigate and understand the schema produced using the ERD in the same folder.

A.4.10 Schema Tables

- Now all the schemas are formed which can be seen in pgadmin by browsing PostgreSQL 12 → Databases → Dublin_Smart_City → Schemas → Public → Tables. Under Tables you will see all the generated tables.

A.4.11 Data Extraction

To extract the data change the directory to **‘cd\Transport_Automated_Full_Version’** and then run the file **‘python Scrape_Dublin_Data.py’** this script will populate the schemas and CDM as well by filtering and cleaning the data streams with any unwanted semantic.

A.4.12 Database backup

1. To restore the backup of the database open pgadmin and right click on Databases, select create new and provide the name 'TransportDatabase'. Then right click on the newly created database and select restore. A new window will open with following 4 options:

   - Format
   - Filename
   - Number of jobs
   - Role name

   select Tar in the format, navigate to the backup file path in Filename option, leave the number of jobs empty and lastly select the role name postgres.

2. It will take a while to restore all the data from backup file.

3. You can navigate through and check the data or by querying the tables.

A.5 Running the API Service

1. Extract the www.zip file to the C drive and then open anaconda prompt and type 'pip install hug' to install the hug library. Note: if you use method in A.3.3 Automated Method there is no need to manually install it.

2. Then navigate to the www folder by using the command **'cd\www'**. When the directory is changed type **'hug –f Bikesapi.py'** this will bring the API server Online and will allow communication between the API and the Database.

3. When you will see the hug logo and message inside the DOS prompt and serving at port 8000, then open any web browser such as Google Chrome, Firefox or Internet Explorer and write
   http://localhost:8000/ This will display the response of API call back from Postgres database.

4. Then write this link in a new tab to search data according to your own provided parameters http://localhost:8000/bikes_peak_morning_time?station_id=10&recordd ate=2019-10-10
   by changing the station id and date you can retrieve the data through passing parameters.

A.6 Dublin Bikes Peak Times

1. If you have node.js already installed, make sure to install the latest one by uninstalling the previous one and installing the latest one from the internet using the link https://nodejs.org/en/download/

2. To run the Dublin peak times website extract the **DublinBikes.7z** file to C drive and open anaconda prompt and change the directory to **'cd\DublinBikes'** when the directory is changed execute the command **'npm run dev'** which will make the node web server online in that directory and you will see the Dublin Bikes Peak Times web application automatically opened in web browser.

3. You might face an issue while interacting with the API through your web browser if you are using Google chrome in that case when you are not getting any response from the API even though when both API and web server is online then simple you need to add the extension 'Moesif Orign & CORS Changer' to your web browser. It is available at:
   https://chrome.google.com/webstore/detail/moesif-orign-cors-changer/digfbfaphojjndkpccljibejjbppifbc
   For more information visit:
   https://www.moesif.com/blog/technical/api-gateways/How-to-Choose-The-Right-API-Gateway-For-Your-Platform-Comparison-Of-Kong-Tyk-Apigee-And-Alternatives/

4. After installing it switch it on from the top right corner and check the API response in the web console.

Appendix B

API Framework

B.1 Python Hug Framework

The Hopefully Useful Guide (Hug) is a micro-framework to making standard internal Python APIs available externally (Python_Hug (2019)). It provides architects and developers an opportunity to define structure and logic, and then expose it cleanly to other means. The system requirements for hug is very minimum, it needs a local installation of Python 3.3+, and optionally inside a virtual environment (virtualenv). In addition, pip (the python package installer) is also required, but this is included by default with any version of Python 3.4 or greater. Some of the features of Hug API are mentioned below:

1. It creates HTTP REST APIs idiomatically and cleanly.
2. This micro-service makes easy the switching out of pure python calls for HTTP communication.
3. Expandable type system for simple parameters.
4. Support for fully integrated Marshmallow, an advanced deserialization, serialization and validation library. The Hug uses marshmallow fields and schemas as input types.
5. Python 3 type annotations for API specification and validation.
6. It provides directives for common required attributes.
7. Easily build new directives.
8. Provides Hug routing.
9. Supports any WSGI server that uses auto-reloading, e.g. uWSGI and Gunicorn.

10. Build upon high performance Falcon’s HTTP library.
11. Divides APIs over multiple files.

The hug micro-framework is a lightweight and powerful library to work with the python existing modules. It is easy to deploy without creating complex logic. However, the hug API micro-framework requires more detailed documentation when using any development language other than python. This documentation should explain the working of hug framework with other existing technologies and frameworks. In addition, to the above explained advantages one another reason for choosing the Hug API is because our Extract, Load and Transform (ELT) framework is developed using the Python programming language. So, to maintain the consistency of core programming language throughout the research project, Python language services were preferred where possible.

Appendix C

Vue.js Framework, Charts.js, D3.js, Echarts

C.1 JavaScript Framework

JavaScript is also known as JS, it is a high level object oriented programming language complied just in time. JavaScript contains syntax of curly-brackets, first class functions, dynamic typing and object-oriented based prototype. It also conforms to the ECMAScript language specifications (ECMAScript, 2020). In combination with web development languages such as HTML and CSS, JavaScript is among one of the core technologies of World Wide Web (WWW). It is an essential component of web applications and enables web page interactivity. The major web browsers contain a built in engine to run JavaScript codes because a majority of web pages use JavaScript.

It has a multi-paradigm structure which supports different styles such as imperative, event-driven and functional. It has Application Programmable Interfaces (APIs) to work with arrays, regular expressions, text, dates and the Document Object Model (DOM), but it doesn't contain any I/O, like storage, graphics and networking facilities. To provide these facilities JavaScript depends on the embedding host environment. In the beginning JavaScript was only embedded in client-side browsers, but now are embedded in many host software types, including databases and server-side web servers, in non-web programs like PDF and word processing software. It also includes run-time environments to facilitate the desktop and mobile applications development.

C.1.1 Vue.js Framework

Vue.js is a JavaScript open source Model-View-ViewModel (MVVM) framework used for developing single-page applications and graphical user interfaces. Unlike other inflexible frameworks, Vue.js is a progressive development framework which is designed with the intention to be gradually adoptable. The focus of the main library is on the view layer only, and integrating with other projects and libraries is easy. On other side, when Vue.js is used with other supporting libraries and modern tools Vue.js can effectively provide power to the single page applications. Some of the features of Vue.js over other popular web development frameworks are mentioned below:

1. It is a light-weight framework compared to Angular JS and provides a few more built-in features and options than React JS.

2. Vue.js is simple to learn and understand as compared to Angular and React JS. It maintains a well-outlined structure to keep the information, custom methods and life-cycle strategies separate.

3. It provides simple integration and is easy for developers to integrate Vue.js with other existing applications because of being based on JavaScript framework. It can also be used to replace jQuery.

4. Vue.js provides great flexibility to the user to write the template in HTML file, JavaScript file, and pure JavaScript file using virtual nodes. This flexibility gives easy understanding of Vue.js framework to the developers of React, Angular and other JavaScript frameworks.

5. The MVVM design allows two-way communications that makes it simple to handle the HTML blocks. Two way binding bounds the data from the Document Object Model (DOM) back to JS. Vue.js is quite different to libraries like React.js which provides method communication only.

6. It is highly reactive. This adds many getters and setters to every variable and this feature allows it to track modifications and updates in the DOM.

7. It provide a new Vue Command Line Interface (CLI) which is one of the simplest tool for JavaScript framework development. It permits the user to start new projects like State Store, Unit Testing, Routing, Typescript, and CSS pre-processors. It also provides a user interface to manage your projects.

8. It has learned from the React library and Angular framework taking the best components from each of them. Similar to the React library, Vue.js uses a virtual DOM that makes it quite fast. Similar to the Angular framework, Vue.js allows two way data binding and directives.

9. It is a progressive framework which makes it convenient for the user to adapt and learn quickly as compared to Node.js and other JavaScript frameworks.

10. It provides adaptability and hassle-free migration, reusable components, easy and efficient structure.

11. Vue.js provides a detailed documentation and a large community of contributors.

C.2 Data Charts

The presentation of data in a graphical format is called data visualisation. It may be in the form of images that provides the information about the data to the audience. A meaningful chart is a result from a systematic and structured mappings between the data figures and the graphical points on the chart. The demonstration of data values or new insights on chart is defined by these mapping rules, these mappings rules also manages the extent properties of graphical marks on chart, these properties are size or colour, change in them to reflect the change in datum value.

C.2.1 Chart.js

The chart.js is a JavaScript library used by the user to create several charts using the canvas element of HTML5. The canvas is an HTML5 element that is used via JavaScript to draw graphics, it is basically a graphics container. The chart.js uses canvas, in order to use it in an old browser you have to include canvas 5 Polyfill library. This chart.js library does not have dependencies and have file size of 11kb when served as zipped, minified and concatenated. However, this size can be reduced even more if not all of the six core chart types are used and just includes the required one only. Suppose you need to include the bar chart only in your web application, you can include the bar module, core features and save the bandwidth of the end user. In addition, another interesting feature of chart.js is having responsiveness and reactivity by default without writing code, they will adapt based on the screen size. Also, unlike other libraries there is a lot of clear and detailed documentation available that makes advanced options easy as well.

C.2.2 D3.js

The D3.js is a library developed in JavaScript to manipulate data and to visualise data using various types of charts appropriate to understand. The D3.js uses the Scalable Vector Graphics (SVG), HTML and CSS that helps you to bring life to your Web pages. The XML based SVG is used as a graphics container. D3.js is compatible with all the latest web browsers giving you the advantage of using robust visualisation components and a data-driven methodology to manipulate your Document Object Model (DOM). The D3.js allows you to bind your data to DOM and then implement the data-driven modifications to your document. For example, a user can use D3.js to produce an HTML table out of an array of digits or the user can use the same data to build an interactive SVG line chart with smooth interaction and transitions.

C.2.3 ECharts

ECharts is a robust, free visualisation and charting library allowing the users to add highly interactive, intuitive and customisable charts to any applications or commercial products. ECharts is purely written in JavaScript and is based on a lightweight canvas library called zrender. The zrender provides two dimensional charts for ECharts library. The ECharts is an easy-to-use framework which allows you to build interactive charts with convenient, interactive and high performance properties. The data can be visualised using ECharts by having minimum expertise in web development.

C.2.4 Comparison of Chart.js, D3.js and ECharts

These charts are well known and widely used JavaScript libraries for charting and visualising data. They allow the creation of line charts, bar charts, scatter plots and other graphs but they all have significantly different approach. The Chart.js and ECharts provides facility to use ready to go charts that can be configured and styled accordingly. While the D3.js provides an opportunity to create blocks that will merge together to create any data visualisation virtually. To create charts using D3.js a user should have experience and expertise as a developer to build the required data visualisations.

However, the chart.js uses pre-built charts and can be implemented by writing a few lines of code without having much coding experience. it requires little time as compared to D3.js to make a chart. The Charts.js is also responsive and includes hover pop-up, legend and toggling but unlike D3.js they can be configured by modifying chart elements easily. The chart.js uses HTML canvas element for rendering its charts instead of SVG element which results in better performance, especially when a huge quantity of data needs to be rendered.

Library	Extended Charts	Customs Interactions	Canvas
Charts.js	No	No	Yes
Highcharts	No	No	No
C3	No	No	No
D3.js	Yes	Yes	Yes
ECharts	Yes	Yes	Yes

Table C.1 Chart Comparison

Furthermore, the charts built in Chart.js and ECharts can be downloaded as an image file due rendered by canvas element.

The authors (Li *et al.*, 2018) has presented the ECharts as an efficient and structured rapid framework for web-based visualisations. It is created to facilitate the easy-to-use charts for users with minimum or no programming experience. The components, data, styles and interactions for ECharts are easy to configure and customise according to the user requirements. These designs provides an opportunity to decrease the workloads and to take full control of visualisation structures and construction procedures. However, due to the reason of being established on a high performance HTML5 canvas management and rendering system known as Zrender, its performance is much faster as compared to D3.js and chart.js. These performance benchmark tests were conducted and demonstrated by (Li *et al.*, 2018) to outline the response time and the behavior of these libraries in real world applications, also the comparison of these libraries is provided in Table. C.1.

Appendix D

SQL Statements

In this section we have mentioned the actual Structured Query Language (SQL) queries used in our Extract, Load and Transform (ELT) framework.

D.1 Algorithm 3.1 SQL Query

```
CREATE TABLE Bus_Stops (Bus_Stop_ID NUMERIC PRIMARY KEY,
Bus_Stop_Name VARCHAR, Bus_Stop_Latitude NUMERIC,
Bus_Stop_Longitude NUMERIC)
```

D.2 Algorithm 3.2 SQL Query

```
INSERT INTO Bus_Stops VALUES(Bus_Stop_ID, Bus_Stop_Name,
Bus_Stop_Latitude, Bus_Stop_Longitude)
```

D.3 Algorithm 3.4 SQL Query

```
INSERT INTO Bus_Stops_Distance VALUES(Start_Stop_ID, Start_Stop_Name,
Start_Stop_Lat_Lng, End_Stop_ID, End_Stop_Name,
End_Stop_Lat_Lng, Distance_km, Travel_Minutes)
```

D.4 Algorithm 3.5 SQL Query

```
SELECT Bus_Stop_ID, Bus_Stop_Name FROM Bus_Stops
WHERE Bus_Stop_ID = 123
```

D.5 Algorithm 3.6 SQL Queries

```
INSERT INTO Bikes_Data VALUES(Station_ID, Station_Name, Banking, Bonus,
Total_Stands, Available_Stands, Available_Bikes, Status,
Last_Update, Record_Date, Record_time)
```

D.5.1 Retrieve Bike Station details

```
SELECT Station_ID, Station_Name FROM Bike_Stations
WHERE Station_ID = 44
```

D.6 Algorithm 3.7 SQL Query

```
INSERT INTO Bikes_Usage VALUES(Station_ID, Station_Name,
Total_Stands, Previous_Available_Stands, Available_Stands,
Calculated_Available_Stands, Stands_Display_Msg,
Previous_Available_Bikes, Available_Bikes, Calculated_Available_Bikes,
Bikes_Display_Msg, Record_Date, Record_time)
```

D.7 Algorithm E.6 SQL Query

```
INSERT INTO Luas_Stops_Distance VALUES(Start_Stop_ID, Start_Stop_Name,
Start_Stop_Lat_Lng, End_Stop_ID, End_Stop_Name,
End_Stop_Lat_Lng, Distance_km, Travel_Minutes)
```

D.8 Algorithm 3.9 SQL Query

```
SELECT Line_Direction_ID, Stop_ID, Stop_Name, Day_ID, Day_Name, Start_Time,
End_Time, Min_Time, Avg_Time, Max_Time FROM Luas_Frequency WHERE
(Line_Direction_ID = 50200111 AND Stop_ID = 20012 ) AND (Day_Name = 90011
AND 10:30:00 BETWEEN Start_Time AND End_Time
```

D.8.1 Retrieve and Calculate Current Luas Frequency

```
SELECT AVG(Due_Minutes) FROM Luas_Times WHERE
(Line_Direction_ID = 50200111 AND Stop_ID = 20012)
AND (Record_Date = 21-10-2019 AND Record_Time = 10:30:00)
```

D.9 Algorithm 3.11 SQL Query

```
SELECT SUM(Free_Flow_Time), SUM(Current_Time), SUM(Distance)
FROM Travel_Times WHERE (Path_ID BETWEEN 50111 AND 50120) AND
(Record_Date = 21-10-2019 AND Record_Time = 10:30:00)
```

D.9.1 Retrieve Routes data

```
SELECT Motorway_ID, Route_ID, Junction_Start_Name,
Junction_End_Name FROM Routes WHERE
(Start_Path_ID = 50111 AND End_Path_ID = 50120)
```

D.10 Algorithm 3.12 SQL Query

```
SELECT Quality_ID, Quality_Category FROM
Air_Quality_Index WHERE Quality_ID = 2)
```

Appendix E

ELT Framework Algorithms and APIs

In this section we present additional algorithms that are similar to the algorithms explained in Chapter 3. In addition we have already explained the API calls.

E.1 Algorithm to create Bike_Station table

Algorithm E.1: `Create_Table_Bike_Stations`

```
/* This algorithm analyses the Bike Stations file to identify the columns required to
   construct the Bike Station table and then creates the table in the database.      */
input  : bike_stations.txt - the file contains the required bike station column names and data.
output : The Bike Stations table is created in the database.
begin
    line ⟵ read the first (header) line of bike_stations.txt file;
    if (station_id in line) then
        Bike_Station_ID ⟵ station_id;
        Bike_Station_ID_Type ⟵ Numeric;
    end
    if (station_name in line) then
        Bike_Station_Name ⟵ station_name;
        Bike_Station_Name_Type ⟵ Varchar;
    end
    if (station_latitude in line) then
        Bike_Station_latitude ⟵ station_latitude;
        Bike_Station_latitude_Type ⟵ Numeric;
    end
    if (station_longitude in line) then
        Bike_Station_longitude ⟵ station_longitude;
        Bike_Station_longitude_Type ⟵ Numeric;
    end
    Create Bike_Stations_Table (Bike_Station_ID Bike_Station_ID_Type
                                Bike_Station_Name Bike_Station_Name_Type,
                                Bike_Station_latitude Bike_Station_latitude_Type,
                                Bike_Station_longitude Bike_Station_longitude_Type);
end
```

E.2 Algorithm to populate Bike_Stations table

Algorithm E.2: `Populate_Table_Bike_Stations`

```
/* This algorithm reads the Bike Stations file and populates its data into the Bike_Stations
   table created by algorithm E.1.                                                          */
input  : bike_stations.txt - the file contains all the Bike Stations data.
output : The Bike Stations data is inserted into Bike_Stations table.
begin
    bike_stations_list ⟵ empty list;
    File_Data ⟵ read all lines in the file except the first (header) line ;
    for line in File_Data do
        single_bike_station ⟵ empty list;
        for station in line.strip(newline).split(,) do
            single_bike_station.append(station);
        end
        bike_stations_list.append(single_bike_station);
        Insert into Bike_Stations_Table Values(single_bike_station);
    end
end
```

E.3 Algorithm to create Luas_Stops table

Algorithm E.3: `Create_Table_Luas_Stops`

```
/* This algorithm analyses the Luas Stops file to identify the columns required to construct
   the Luas Stops table and then creates the table in the database.                          */
input  : luas_stops.txt - the file contains the required bike stop column names and data.
output : The Luas Stops table is created in the database.
begin
    line ⟵ read the first (header) line of luas_stops.txt file;
    if (stop_id in line) then
        Luas_Stop_ID_Name ⟵ stop_id;
        Luas_Stop_ID_Type ⟵ Numeric;
    end
    if (stop_name in line) then
        Luas_Stop_Name_Name ⟵ stop_name;
        Luas_Stop_Name_Type ⟵ Varchar;
    end
    if (stop_abrev in line) then
        Luas_Abrev_Name ⟵ stop_abrev;
        Luas_Abrev_Type ⟵ Numeric;
    end
    if (latitude in line) then
        Luas_Stop_latitude_Name ⟵ latitude;
        Luas_Stop_latitude_Type ⟵ Numeric;
    end
    if (longitude in line) then
        Luas_Stop_longitude_Name ⟵ longitude;
        Luas_Stop_longitude_Type ⟵ Numeric;
    end
    Create Luas_Stops_Table (Luas_Stop_ID_Name Luas_Stop_ID_Type
                             Luas_Stop_Name Luas_Stop_Name_Type,
                             Luas_Abrev_Name Luas_Abrev_Name_Type,
                             Luas_Stop_latitude Luas_Stop_latitude_Type,
                             Luas_Stop_longitude Luas_Stop_longitude_Type);
end
```

E.4 Algorithm to populate Luas_Stops table

Algorithm E.4: `Populate_Table_Luas_Stops`

```
/* This algorithm reads the Luas Stops file and populates its data into the Luas_Stops table
   created by algorithm E.4                                                                  */
input  : luas_stops.txt - the file contains all the luas stops data.
output : The luas stops data is inserted into Luas_Stops table.
begin
    luas_stops_list ⟵ empty list;
    File_Data ⟵ read all lines in the file except the first (header) line ;
    for line in File_Data do
        single_luas_stop ⟵ empty list;
        for stop in line.strip(newline).split(,) do
            single_luas_stop.append(stop);
        end
        luas_stops_list.append(single_luas_stop);
        Insert into Luas_Stops_Table Values(single_luas_stop);
    end
end
```

E.5 Algorithm to create Luas Paths

Algorithm E.5: `Luas_Paths_Formation`

```
/* This algorithm creates the luas paths for the luas stops.                                 */
input  : luas_stops_list - the luas_stops list is passed as an input.
output : paths_list - a list of paths (two nearest bike stops).
begin
    luas_stops_list ⟵ retrieve all the stops data from the luas_stops table using line_direction_id.
    length ⟵ compute the length/size of luas_stops_list
    path ⟵ empty list;
    paths_list ⟵ empty list;
    for (row_index, row_data in luas_stops_list) do
        if (row_index < (length − 1)) then
            path ⟵ row_data + luas_stops_list(row_index + 1);
            paths_list.append(path);
        end
    end
end
```

E.6 Algorithm to compute Luas paths distance

Algorithm E.6: `Calculate_Paths_Distance`

```
/* The paths_list from algorithm E.5 will be analysed to calculate the paths distance using
   google distance matrix API.                                                          */
input  : paths_list - the paths_list constructed by algorithm E.5 will be given as input.
output : The calculated distance data of paths is inserted into Luas_Stops_Distance table.
begin
    for path in paths_list do
        Distance_Matrix ⟵ compute_distance(path.start_GPS_point, path.end_GPS_point);
        distance_list ⟵ empty list;
        for elements in Distance_Matrix do
            for values in elements do
                travel_time ⟵ values.duration;
                distance_km ⟵ values.distance;
                meter ⟵ "m";
                distance_unit ⟵ distance_km.split('m');
                if (distance_unit == meter) then
                distance_km ⟵ values.distance/1000;
                else
                    distance_km ⟵ values.distance
                end
                distance_list ⟵ path + distance_km + travel_time;
                Insert into Luas_Stops_Distance Values(Start_Stop_ID, Start_Stop_Name,
                                         Start_Stop_Lat_Lng, End_Stop_ID, End_Stop_Name,
                                         End_Stop_Lat_Lng, Distance_km, Travel_Minutes)
            end
        end
    end
end
```

E.7 Algorithm to create Motorways table

Algorithm E.7: `Create_Table_Motorways`

```
/* This algorithm analyses the motorways file to identify the columns required to construct
   the Motorways table and then creates the table in the database.                      */
input  : motorways.txt - the file contains the required column names and data.
output : The motorways table is created in the database.
begin
    line ⟵ read the first (header) line of motorways.txt file;
    if (motorway_id in line) then
        Motorway_ID ⟵ motorway_id;
        Motorway_ID_Type ⟵ Numeric;
    end
    if (motorway_name in line) then
        Motorway_Name ⟵ motorway_name;
        Motorway_Name_Type ⟵ Varchar;
    end
    Create Motorways_Table (Motorway_ID Motorway_ID_Type
                   Motorway_Name Motorway_Name_Type);
end
```

E.8 Google Distance Matrix API for Dublin Luas

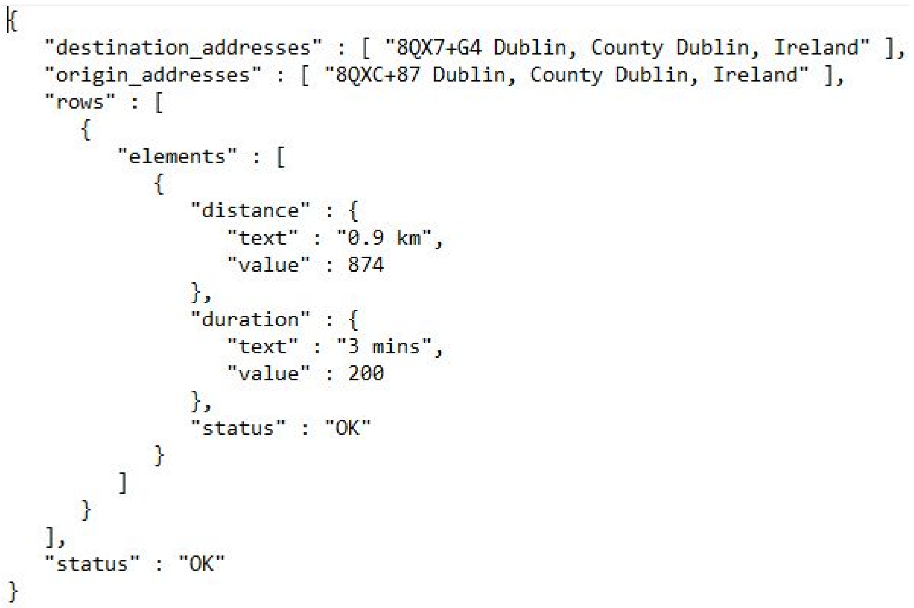
E.8.1 API for Dublin Luas

To calculate the Distance for Dublin Luas paths, we have used the mode **"tram"** and departure time 09:00 in the API URL. In Figure E.1 we present the sample output results from **"The Point"** luas station till the **"Spencer Dock"** luas station.

API URL for Dublin Luas

https://maps.googleapis.com/maps/api/distancematrix/json?units=metric&origins="53.34835,-6.22925833333333"&destinations="53.3488222222222,-6.23714722222222"&mode=tram&departure_time=0900&key=YOUR_API_KEY

API response for Dublin Luas



```
{
   "destination_addresses" : [ "8QX7+G4 Dublin, County Dublin, Ireland" ],
   "origin_addresses" : [ "8QXC+87 Dublin, County Dublin, Ireland" ],
   "rows" : [
      {
         "elements" : [
            {
               "distance" : {
                  "text" : "0.9 km",
                  "value" : 874
               },
               "duration" : {
                  "text" : "3 mins",
                  "value" : 200
               },
               "status" : "OK"
            }
         ]
      }
   ],
   "status" : "OK"
}
```

Figure E.1 API Response for Dublin Luas

E.9 Google Distance Matrix API for Dublin Bus

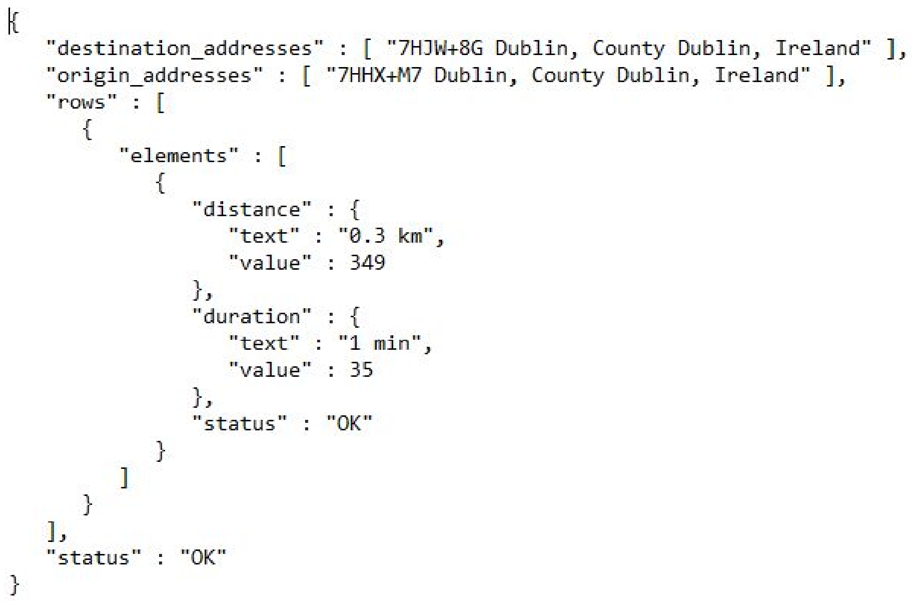
E.9.1 API for Dublin Bus

To calculate the Distance for Dublin Bus paths, we have used the mode **"bus"** and departure time 09:00 in the API URL. In Figure E.2 we present the sample output results from **"Citywest Fortunestown Road"** bus station till the **"Citywest Jobstown Road"** bus station.

API URL for Dublin Bus

https://maps.googleapis.com/maps/api/distancematrix/json?units=metric&origins="53.2791505295167,-6.40175897234441"&destinations="53.2808204191357,-6.40363122917668"&mode=bus&departure_time=0900&key=YOUR_API_KEY

API response for Dublin Bus



```
{
   "destination_addresses" : [ "7HJW+8G Dublin, County Dublin, Ireland" ],
   "origin_addresses" : [ "7HHX+M7 Dublin, County Dublin, Ireland" ],
   "rows" : [
      {
         "elements" : [
            {
               "distance" : {
                  "text" : "0.3 km",
                  "value" : 349
               },
               "duration" : {
                  "text" : "1 min",
                  "value" : 35
               },
               "status" : "OK"
            }
         ]
      }
   ],
   "status" : "OK"
}
```

Figure E.2 API Response for Dublin Bus