UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

ZURICH

Technology
Delivery Center
ServiZurich | Barcelona

Implementation of a data warehouse on the cloud

Lucas Cajal Treviño

Bachelor Thesis
Specialization in Computing

Director: José Luis Sánchez Ros
Ponent: Elvira Pino Blanco
GEP Tutor: Joan Sarda Ferrer

January 19, 2022

Acknowledgement

I would like to thank everyone at ServiZurich who has helped in the making of this project, specially my director José Luis, and my colleague and also roommate Arnau Soler, who has been a huge help in the final stages of the project.

També vull donar les gràcies a la Judit, per haver-me donat sempre un suport incondicional i haver estat al meu costat, sempre d'una forma tan especial.

A mi familia, que siempre me ha apoyado y ha aguantado mis monólogos en las comidas de fin de semana, a pesar de no entender demasiado lo que les explicaba.

Y por último a Roberto, a quien me hubiese gustado poder enseñar este proyecto y que estuviese aquí para verme acabar la carrera. Te echamos de menos, kabron.

Abstract

This project is centered around the creation of a system capable of extracting open data from a variety of public sources, that will be loaded into the company's data infrastructure. This infrastructure will analyze and process the data, structuring it to make it easily usable by the company's users and applications. One of these applications will be centered on the usage of data containing geographic information, and it will be implemented as a part of this project.

The final objective is the creation from start to end of a process capable of extracting data from sources external to the company, add it to the company's data infrastructure, and presenting them to a final user via an application that has filtering and visualization functionalities of data based on its geographic location.

Resumen

Este proyecto se centra en la creación de un sistema capaz de extraer datos abiertos de una variedad de fuentes públicas, que serán cargados dentro de la infraestructura de datos de la empresa. Esta infraestructura analizará y procesará los datos, estructurándolos de forma que sean fácilmente utilizables por usuarios y aplicaciones de la empresa. Una de estas aplicaciones estará basada en el uso de datos que contienen información geográfica, i será implementada como parte de este proyecto.

El objetivo final es crear un proceso de inicio a fin capaz de extraer datos de fuentes externas a la organización, añadirlas a la infraestructura de datos de la empresa, y que estos datos puedan ser presentados a un usuario final mediante una aplicación con funcionalidades de filtración y visualización de datos según su ubicación geográfica.

Resum

Aquest projecte se centra en la creació d'un sistema capaç d'extreure dades obertes d'una varietat de fonts públiques, que seran carregades dins la infraestructura de dades de l'empresa. Aquesta infraestructura analitzarà i processarà les dades, estructurant-les de forma que siguin fàcilment usables per usuaris i aplicacions de l'empresa. Una d'aquestes aplicacions es basarà en l'ús de dades que contenen informació geogràfica, i serà implementada com a part d'aquest projecte.

L'objectiu final és crear un procés d'inici a fi capaç d'extreure dades de fonts externes a l'organització, afegir-les a la infraestructura de dades de l'empresa i que aquestes dades puguin ser presentades a un usuari final mitjançant una aplicació amb funcionalitats de filtratge i visualització de dades segons la seva ubicació geogràfica.

Contents

List of Tables

List of Figures

1 Context and scope

1.1 Context

1.1.1 Introduction

Today, the world is driven by data. But the amount of available data dramatically exceeds the amount of used data. Almost every human activity which involves the usage of a digital device generates some kind of data: from GPS coordinates to the heart rate of a person. Everything is logged and stored, even when not used for any particular purpose. And this trend just keeps growing, as companies and public institutions realize the power and importance of data, and as a consequence install more sensors on devices, add data-collection features to software or organize data-collection campaigns.

A clear problem arises: how can we keep up with the increasing availability of data in order to be able to analyze it and use it with ease? How can we collect all this data, which is located in many sources, and be able to use it, considering it comes in different formats and structures? How can we classify it by subject, when most datasets do not have a description, or it is located in many places (the source's website, inside a README file, etc.)? Clearly, doing all of this manually is an impossible task. We need a way to automate the process of extracting data, classifying it and transforming it to a standardized format to allow its ease of use.

An example of this problem can be found in the insurance company Zurich. Being the 76th largest company in the world [1] and having presence in 215 countries and territories [2], the amount of data the company possesses is enormous. But with the company being so large, every business unit has its own collected data, stored in its own servers, and as a result there is no standardization between regions and business areas, preventing the sharing of data across business regions and limiting the potential usage of the available data. Most of the data usage is ad-hoc for specific use cases and very little reusability is achieved, even inside the same business unit. Additionally, data preparation is a costly process, and it is sometimes being performed several times for the same data (or of similar nature) for different use cases.

1.1.2 Terms and concepts

Before starting the definition of the project, some background on terms and concepts related to it must be given in order to properly understand the work.

Cloud computing: The increasingly famous term of cloud computing [3] refers to the usage of computing services, mainly storage and computing power, over the internet, without active management of the used resources by the user.

DevOps: They are a set of practices that aim to shorten and automate the development cycle of software, by integrating automated testing, continuous integration and deployment (CI/CD) of the product and other practices [4].

Data Warehouse: Data warehouses [5] are a type of data management systems that take data from a wide range of sources and stores it in a common format for later access. They process data in three different stages to achieve this:

- **Data ingestion:** The first step is extracting the data from the corresponding source and storing it in a landing database as is.
- **Data transformation:** Data then needs to be transformed to standardize its format. Every source and dataset has a different structure, and this needs to be changed to have a common format for all the ingested data.
- **Data loading**: Once the data has been transformed, it needs to be stored in a final trusted database where all cleaned data is available. This is where the data will be taken from by the end user.

Geospatial data: Geospatial data is data that has an explicit or implicit relationship with locations on the Earth.

Coordinate reference system (CRS): A coordinate or spatial reference system is a coordinate based system used to locate points and other geographical features in a geographic space. The WGS84 standard [6] defines the Earth surface as an ellipsoid. It is the most common standard in geospatial systems, and most common coordinate reference systems are based on it. It is the standard that will be used for geospatial operations in this project. Specifically, the following spatial reference systems will be used:

- **EPSG 4326 [7]:** This spatial reference system simply adds units to the WGS84 standard (the terms WGS84 and EPSG:4326 are commonly used in the same way), so it represents a curved surface, and uses degrees as units (latitude and longitude). The origin is the point where the equator and prime meridian (Greenwich) cross.

- **EPSG 3857 [8]:** This CRS is a projection of WGS84 into a flat surface. Its units are meters, and as with the previous CRS is centered where the equator and Greenwich meridian cross.

1.1.3 Problem to be resolved

This project aims to provide a solution to that problem. The goal is to implement a system capable of extracting data from multiple sources, both internal company sources and external public (or paid) sources, classifying the data by topic and location and transforming it to have a standard format for all datasets. There is also the need of providing a convenient way of accessing the data, so that it can be used by every business unit around the world.

1.1.4 Stakeholders

Before detailing the stakeholders of this project, some context on how the internal structure of Zurich works is needed. Inside the Zurich ecosystem, there is a subsidiary company named ServiZurich [9] in charge of developing technology products for the parent company. It works as an internal consulting firm, where the clients are Zurich's business units and regions. This subsidiary is where the product will be developed.

Context provided, the main stakeholders of this project are:

ServiZurich's EDAA team: Inside ServiZurich, the team responsible for developing the product is the *Enterprise Data Analytics and Architecture* (EDAA) team. It will be divided into different task groups, each one in charge of developing a specific part of the product: from internal or external data extraction to data transformation and availability. I will be part of the team in charge of developing methods for obtaining external data, classifying obtained datasets by location and subject, and finally implementing a system to provide access to the final transformed data for the clients.

Project manager: The project manager will be the one in charge of specifying the objectives which need to be met, list the features the product will have and supervising the development. This figure will also be in charge of meeting with the ServiZurich's clients, which are the different Zurich business units, in order to sell the product to them and accommodate it to their needs.

Zurich business units: Inside the Zurich Insurance company, all different business units around the globe will benefit from the use of this product. It will allow them to properly organize their own data and give them access to a huge collection of new datasets, both from other business units and from external sources.

Zurich clients: The last group which will be affected by the product are the Zurich clients, both companies and individuals. Although unknowingly, they will benefit from the data-driven decisions taken by Zurich and will have access to a better offering of insurance products.

1.2 Justification

1.2.1 Previous studies

Many companies and organizations have implemented similar solutions for this problem. There are public data portals and standards which aim to be a centralized source for public data (also known as OpenData), such as *Kaggle* [10] or CKAN servers [11], but these are limited to publicly available data and contain only a small portion of it.

Having said this, the usage of data warehouses is becoming increasingly popular, as companies and institutions realize the importance of data for their operations, and as a result many studies have been done on the topic. This means that, even though it will not be possible to investigate similar implementations of the product that is going to be developed, extensive public research has been done on the topic, and it can be used as a baseline for a good design. The specifics of the implementation will have to be done from scratch, but the design of each component of the product and the approaches used for their implementation will be based on previously done research.

1.2.2 Solution justification

The main reasons for developing the product internally are the usage of internal company data and the need to adapt the solution to the specific needs of Zurich Insurance. Although the general scheme of the product could be used for many cases, the company has very specific requirements regarding how the data must be collected, transformed and accessed. And due to the presence of internal data, both from the company and it's clients, we have to implement a system that has never been implemented before (as this is the first time Zurich will have a product like this) and that will not be used outside the company itself. This is why the project will be implemented from scratch.

1.3 Scope

As mentioned earlier, the project will be implemented by different teams, each one in charge of several components of the product. The scope of this thesis is implementing the ingestion of external data, classifying it based on subject and location and setting up a system that enables easy access to the final transformed data. The transformation and standardization of the data is part of the product, but out of the scope of this thesis.

1.3.1 Objectives

We will divide the requirements by project component:

External Data Hub

- **Automated ingestion**: Ensure a periodical and automated ingestion of data from the different sources.
- **Data updating**: Ensure the ingestion of new versions of already ingested datasets when an update is available.
- **No data duplication**: Avoid duplication of data that has been already ingested.
- **Classification**: Analyzing the data to classify it by location and subject.
- **Data reuse**: Allow for the reuse of data, both within the same business unit and globally.
- **Achieve global data licensing**: Allow for a global acquisition of licensed data in order to reduce costs.
- **Easy source addition**: The ingestion framework must allow an easy and fast addition of new sources.

Data access

- **Low-level API**: Create an API to provide access to the data using multiple methods.
- **High-level interface**: Implement an interface that, using the previously mentioned API, can be used to navigate through the data and view it using multiple filters.
- **Holistic view**: Create a holistic view on data already available in Zurich.

1.3.2 Laws and regulations

Something that needs to be considered is the legislation regarding data-protection of different countries. When loading data into the ZIH, an anonymization process should be executed to meet the required privacy standards, while also maintaining the usability of the data. For the scope of this project, all data that is being used is limited to open data sources, which can be freely accessed and used. Future scenarios that might make use of data that has special regulations will be exclusively managed by the ZIH, which takes cares of data anonymization and data protection policy, and therefore it is out of the scope of this thesis.

Regarding the licensing of used tools, the code and libraries that are being used are all under an open source license. Bibliography and material consulted is duly referenced in each section.

1.3.3 Potential obstacles and risks

The three main potential obstacles are:

- **Change of requirements**: A substantial change in the requirements of the clients that forces a redesign of the product's architecture.
- **Dependency on other teams**: As the project is divided in components that will be developed by different teams, it is very likely that at some point one of the teams takes longer than expected to finish a task and this propagates to other teams which have dependencies on that task.
- **Support required by other team**: Each team will be in charge of a different task, but it is possible that tasks need to be reassigned in order to balance the workload between teams, or a team requires support to develop some components.
- **Security related holds**: Security is a critical aspect of the product. Each component, tool, and resource used must meet strict security standards, which in some cases might slow down the development time for certain components.

1.4 Methodology and rigor

1.4.1 Methodology

As the product is made up of differentiated components, a modular design will be created. This will allow for easy development of each component without

having strong dependencies on other components, and will allow teams to work on tasks in parallel. As a consequence, if one of the tasks gets stopped for any reason, the team can immediately jump to a different task until the issues are resolved and minimize the risk of being in stand by due to a hard stop.

From a more practical standpoint, during the development of the project, the main tool that will be used is Microsoft's *Azure* platform [12]. The product will be implemented using the available services inside the platform, and *Azure DevOps* [13] will be used to organize the project, plan required tasks and implement a continuous integration and continuous deployment (CI/CD) [14] [15] infrastructure using *Azure Repos* [16] and *Pipelines* [17]. This will allow having an initial minimal working version of the product, which can be used as a proof of concept (POC) to showcase and sell the product to the clients, and allowing for an incremental upgrade in the quality of the deployed service.

1.4.2 Rigor

The use of *Azure DevOps* will allow for an easy planning of the steps to be taken, the precise monitoring of the tasks, their progress, and the completion rate, with the use of the *Boards* [18] tool. Other tools that will be used are *Azure Repos*, which will be used for version control [19], and then combined with *Pipelines* to allow the implementation of automated testing and code quality checks inside the previously mentioned CI/CD infrastructure.

Weekly meetings with the Project Manager will be done to periodically review the state of the development and the quality of the work being done, and a strict monitoring of both development time and costs will take place to quickly detect deviations in the estimations made in this document.

2 Planning

The start date of the project is September 16th, 2021. and the end date varies depending on the assigned day for the oral presentation, being the earliest possible one on January 24th, 2022. This accounts for a total of 86 working days, distributed in 18 weeks. The amount of hours of work per day will be 7 hours: 6 of them spent working at ServiZurich and an extra hour of personal work. This accounts for a total estimate of 602 total hours spent in the project, 516 of them working for the company and the other remaining 86 hours being for personal work.

The project will be developed using an Agile [20] methodology, more specifically Scrum [21].

2.1 Task definition

We will divide tasks into groups, the first one being the project management and the other two based on the two main features of the product to be implemented: the external data hub and the GeoIntel application.

2.1.1 PM. Project management (160h)

- **PM1. Defining the project's scope (15h)**: Definition of the project's scope and objectives, contextualizing it and providing a justification for its development. This initial writing phase will take approximately 15 hours of work, including both the writing of the document and the later correction of the document based on feedback.
- **PM2. Project planning (15h)**: Definition and description of the tasks, and estimation of the time of completion of each one. Planning of the steps to be taken to complete the project. This task can only start once the project definition (task PM1) has finished. They are both very similar tasks, and therefore have been assigned the same amount of time.
- **PM3. Budget and sustainability report (15h)**: A study and estimation of the project's cost must be evaluated to create a budget. A report has to be written as well to analyze and document the project's sustainability. To start this task, it is necessary to complete the project planning (task PM2), in order to calculate costs based on the needed resources. As with the previous two tasks, the amount of hours assigned is also 15, based on the similarity between them.
- **PM4. Final project documentation (60h)**: On the project's completion, all the previously mentioned documents must be combined, alongside with the documentation of the project itself, and correct any previous mistakes. To complete the document, task PM3 must be finished, as the final document will contain all the previously written information. This is the heavy writing task, as it comprises documenting the whole project, correcting mistakes and checking grammar and format. As this is a long and tedious task, a generous amount of 60 hours has been assigned to it.
- **PM5. Meetings (55h)**: Every week there will be a set of meetings to control the development of the product. They will be divided into 15 minute daily meetings with the developers from Monday to Thursday and a weekly 1.5 hours team meeting with the project manager to analyze the work in progress, monitor the work being done and adapting to

potentially changing or new requirements. As the project will be developed during 18 weeks, this accounts for a total of 45 hours in meetings. An extra 10 hours will be added to account for the meetings with the thesis supervisor and other unplanned meetings.

2.1.2 ED. External Data Hub (230h)

- **ED1. Research available sources (20h)**: For the external ingestion of data, we will first need to make some research on open available sources from which we can obtain our data. This includes both ensuring the quality of the source and studying possible ingestion methods, such as web scraping [22] or usage of a source's API. This will be a slow, manual and tedious task, and will take an estimated 20 hours of work.
- **ED2. Ingestor framework design (20h)**: Create an ingestion framework from which each data ingestor will be implemented. The behavior of each ingestor can only differ when obtaining the data, the rest of the process (input, data storing, exceptions, etc.) must be the same for all of them. This task will not only require a thorough thought process to design a robust and usable framework, but also an investigation of the possible tools to be used for each component of the framework. A total of 20 hours have been assigned to it to ensure that a good design is created.
- **ED3. DevOps (50h)**: Set up a CI/CD infrastructure to automatically test and deploy code. Every service must be saved in Azure Repos [16], and with the help of Azure Pipelines [17] the quality of the code must be checked with the help of *Pylint* [23], it has to be automatically tested using unit tests [24], a Docker [25] image has to be generated and pushed to an Azure Container Registry [26] and finally deployed to the corresponding resource in the cloud. This way all the process can be automated for all the services to be deployed. This can only be done once the ingestion framework has been designed, so task ED2 must be completed before starting. A total of 50 hours has been assigned to it, because a lot of effort must be put into it in order to later save time when developing the project component's, as it will end up automating a lot of manual tasks.
- **ED4. Ingestors (70h)**: An ingestor must be implemented for each source type, implementing the proper ways to extract the data from the corresponding source inside the ingestor framework. To develop the ingestors it is necessary to have completed the previously needed research on sources (task ED1), and the automated testing and deployment of the code has to be in place (task ED3). Although the boilerplate code for every ingestor will be the same, the data extraction algorithm will

change for every source. Therefore, every ingestor will have a different implementation and will take between 10 and 20 hours to code. As we will have multiple ingestors, the total expected development time is of 70 hours (between 4 and 7 different ingestors).

- **ED5. Dataset classification (60h)**: A method for classifying datasets by subject must be implemented. This will require some research and trials to determine the best approach to do so. The initial approach will be investigating the use of machine learning [27] classification models [28] using natural language processing [29] to classify each dataset. As with the ingestors, the infrastructure to automatically test and deploy the code (task ED3) has to be working before starting this task. Regarding the time estimations, finding a good machine learning classification model is a slow task, and therefore 60 hours of work have been calculated.
- **ED6. Automation of ingestors (10h)**: Once the data ingestion platform is set up, automatic event-based triggers will be implemented to automate the download of new available datasets and update already downloaded ones. This is the final step in the External Data Hub component, and can only be executed with tasks ED4 and ED5 completed. This final task is a simple task, and it will take about 10 hours.

2.1.3 DA. GeoIntel (210h)

- **DA1. Geocoding (20h)**: In order to access the data by location, a method for geocoding [30] addresses will be required. This will mean developing a tool capable of both parsing coordinates in multiple formats and translating an address to its corresponding coordinates. The geocoding component is a simple but critical component of the Data viewer. It is expected to take about 10 hours to develop it and another 10 hours to thoroughly test it, accounting for a total of 20 hours.
- **DA2. Auxiliary datasets (10h)**: In order to visualize the data and filtering it, a few manually selected datasets will be required. The main one will be a dataset containing addresses information from all around the globe, in order to visualize the data in a map, filtering datasets by location, etc. Other auxiliary datasets can be added in the future, such as population information. This task will be a fast one, as it will involve searching for a few datasets only. However, this search will be done manually, and the validation of the reliability of the data is critical to ensure a correct API functionality. It will take about 10 hours.

- **DA3. API development (100h)**: An API will be implemented to access all the information. This API will be the way of accessing the datasets, filtering them by location, time stamp, type of data, etc. To implement the API, both tasks DA1 and DA2 must be completed in order to allow data queries by the API. It also has a finish to finish dependency with the viewing website (DA4), as both tasks depend on each other: the website needs a functional API to access data, and the API needs all the routing of the website correctly implemented. This will be the main task in the data accessing component. The API is the backbone of user interaction with the product, and many functionalities and services must be implemented. Therefore, 100 hours have been assigned to its development.
- **DA4. Data viewing website (80h)**: A more user-friendly way of accessing the data will be developed as a web app. This app will have an interactive map and a menu to search and visualize data, and it will use calls to the previously mentioned API to access the data. As mentioned in the previous task, it has a finish to finish dependency with task DA3. Regarding the time estimation, a total of 100 hours might seem too high, but the team has very little experience with front-end developing and therefore a slower than usual development time is expected.

2.2 Time planning

2.2.1 Time estimates summary

Having defined the dependencies between tasks and their estimated time of completion in section 2.1, a summary of the tasks can be found in table 1, where they are listed with their identifier, name, expected completion time and the dependencies between them.

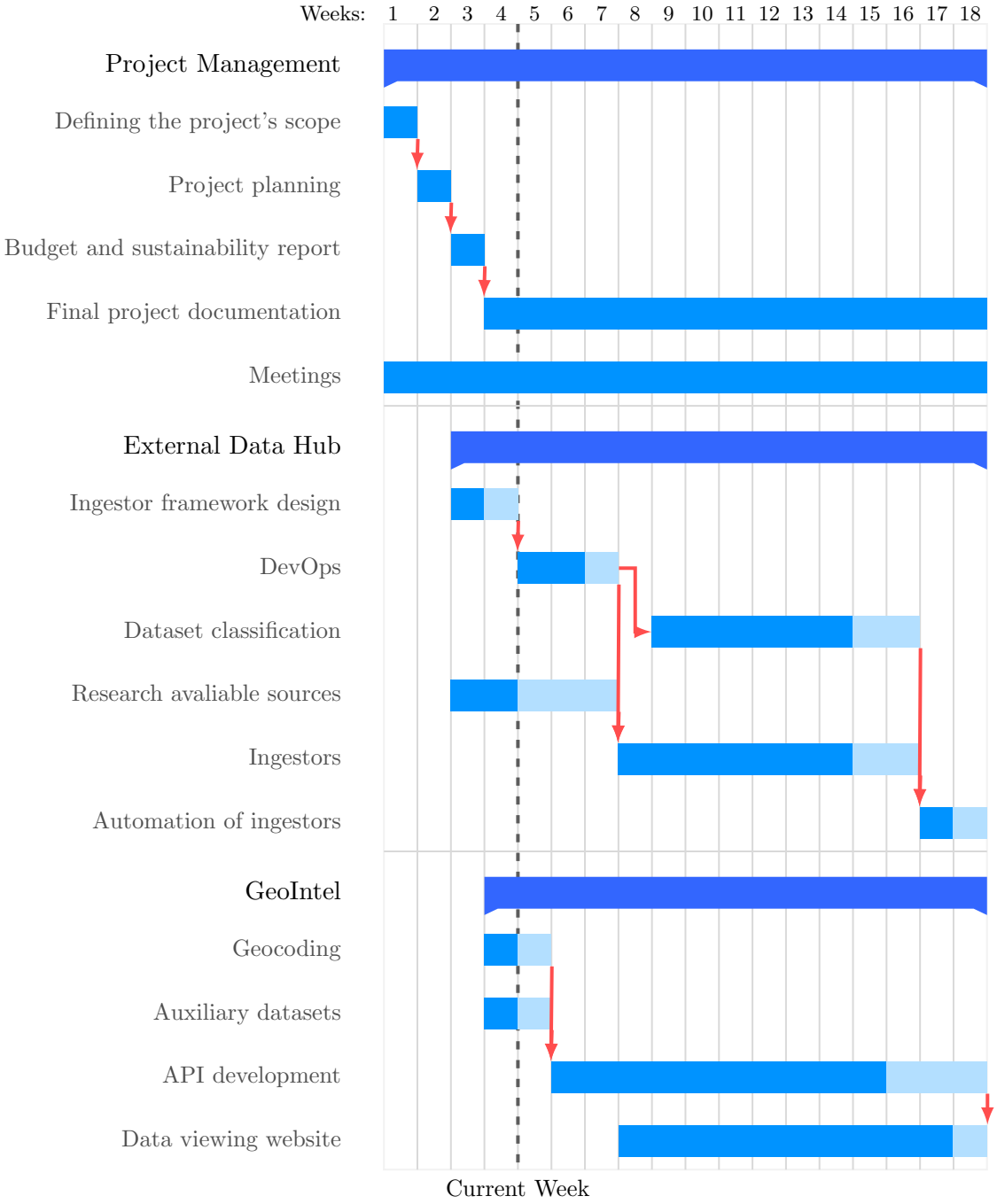
ID	Task Name	Time (h)	Dependencies
PM	Project management	160	-
PM1	Defining the project's scope	15	-
PM2	Project planning	15	PM1
PM3	Budget and sustainability report	15	PM2
PM4	Final project documentation	60	PM3
PM5	Meetings	55	-
ED	External Data Hub	230	PM
ED1	Research available sources	20	-
ED2	Ingestor framework design	20	-
ED3	DevOps	50	ED2
ED4	Ingestors	70	ED1, ED3
ED5	Dataset classification	60	ED3
ED6	Automation of ingestors	10	ED4, ED5
DA	GeoIntel	210	PM
DA1	Geocoding	20	-
DA2	Auxiliary datasets	10	-
DA3	API development	100	DA1, DA2
DA4	Data viewing website	80	DA3

Source: Own creation.

Table 1: Summary of tasks.

As explained in section 1, the two main task groups (External Data Hub and GeoIntel) are completely independent of each other, and therefore can be done in parallel. And the project management tasks need to take place alongside the development of the product, as meetings will control the execution of other tasks and adapt the strategy to accommodate changes, and the documentation of components will be done on the fly every time there is a new component finished. This allows for a high concurrency of task execution, as it can be seen in figure 1.

2.2.2 Gantt chart



Source: Own creation.

Figure 1: Gantt chart

2.3 Resources

2.3.1 Project management

- **Microsoft Teams**: We will use Microsoft Teams to communicate with the development team and the project manager, both through chat and remote meetings.
- **Outlook**: Alongside Teams, we will use Outlook for email communications.
- **Overleaf**: The online LaTeX text editor Overleaf will be used to write all the project's documents.
- **Microsoft Azure DevOps**: Microsoft's Azure DevOps, more specifically the Boards [18] resource, will be used to organize tasks and monitor the project's development, using the scrum [21] methodology.

2.3.2 Code development

- **Microsoft Azure**: The cloud computing service provided by Microsoft, Azure, will be the main platform for the deployment of the product. We will use many resources from the ones available. The following list contains the main components selected for use in the planning phase, however more resources might be added in the future:
  - **Azure Functions**: They will be the main workers for the project. Both serverless and stateless Azure Functions and state preserving Durable Functions [31] will be used to implement the Ingestors, the API backend and other critical components of the project.
  - **Azure App Service**: The App Service resource will be the tool used to implement both the API routing and the data visualizer as a website.
  - **Postgres SQL database**: With the help of the PostGIS [32] extension, it will be used to allow the API to run location queries and retrieve data based on geospatial information.
  - **Cosmos database**: This NoSQL [33] database will have the final cleaned data and from it the API will query the desired datasets.
  - **Azure Blob Storage**: It will be used as the storing place for the raw ingested data.
  - **Azure ML**: The dataset classifier will use this resource to deploy classification [28] models.
  - **Azure Maps**: It will be used for the Geocoding [30] phase

  - **Azure Container Registry**: Here is where the Docker images of different components will be stored, and later retrieved by each resource to create the running container for each specific component.
  - **Azure KeyVault**: It will be used to store critical application parameters, such as passwords, access tokens or API keys.

- **Microsoft Azure DevOps**: The DevOps platform created by Microsoft will be used for version control and setting up the CI/CD infrastructures for the project's components, using Azure Repos [16] and Azure Pipelines [17].

- **Postman**: The Postman API platform [34] will be used as a testing tool during development. It will mainly help for making queries to Azure Functions, APIs, etc., both when testing locally as when testing deployed components of the product.

- **Microsoft Visual Studio Code**: The IDE we will use is VS Code. The reason for this could be simply that it is the one we like the most, but in this case it will also be very helpful because we are working with Microsoft products, and it has a very good integration with Azure.

- **Docker**: The Docker tool will be used to create virtualized environments for each product component, in order to easily define each component's running environment and simplify the deployment process.

- **Swagger/OpenAPI**: When creating the API, it will be critical to clearly define its structure and document its usage. For this, we will use Swagger [35], a tool that uses either the JSON [36] or YAML [37] format to define APIs.

2.3.3 Human resources

- **Product Manager**: It is indispensable to have a product manager to coordinate the developer teams, communicate with the clients and supervise the product's development.

- **Developer teams**: Two main development teams will be required to implement the project. These can be further divided into smaller teams as needed to adapt to the project's requirements.

- **Infrastructure support**: An extra team of people is required to manage the infrastructure used by the developers. This team will be in charge of managing security, creating the needed Azure resources and providing support to the developer teams.

2.3.4 Other resources

- **Laptops**: Each person involved in the project will need a laptop to work on the project. Although computers with good performance are nice to have, they are not strictly required, as all heavy processing will be done on the cloud using the Azure Platform.
- **Internet connection**: It is critical to have a good and safe internet connection, as all work will be done remotely. Both for the project management and product development, almost all tools are online, so a secure and stable network is indispensable.
- **Computer peripherals**: To allow for a comfortable workflow and therefore improve the productivity of workers, peripherals such as chairs, mouses, monitors or headsets must be provided to all workers to create a practical working environment.

2.4 Risk management

During the development of the project, many factors can affect its execution and slow down the development process. These factors have been listed in the context and scope chapter, under section 1.3.3. After a study of each one of them, a few solutions have been proposed to mitigate their negative effects:

2.4.1 Change of requirements

In case the client requests a significant change in the requirements, the product's architecture will need a redesign.

- **Risk**: Low.
- **Mitigation**: The Project Manager will be in constant touch with the client to make sure that their requirements are in sync with the design of the product, and to improve the reusability of already developed code in case this scenario happens, a highly modular architecture will be put in place, in a way that the components have a low dependency on other parts of the project and can be easily adapted to the new architecture without having to reimplement their core functionality.
- **Solution and consequences**: If the architecture needs to be changed, extra working time for all people involved will be mandatory, and therefore increase the cost of human resources. The used Azure services will also change, and therefore the development costs. However, it cannot be

predicted in what way, as they can either increase or decrease, depending on what changes are made to the architecture.

2.4.2 Dependency on other teams

It is very likely that at some point in the development process, a task is halted as a consequence of a dependency on another team's work.

- **Risk**: High.
- **Mitigation**: To lower the consequences of this scenario, the same modular architecture mentioned before will be helpful, as the low dependency between components will minimize the risk of this happening. Furthermore, tasks related to code development will always be prioritized over other tasks: this will ensure that tasks that can be affected by this risk are done earlier, and therefore the risk is reduced, and that when this scenario does happen, other tasks that are not related to code development will be available to be worked on while waiting for the other team to finish their work.
- **Solution and consequences**: If the developers find themselves completely halted, they will be reassigned to help the teams that are slowing down the development to try to speed up the completion of their pending tasks. This will translate in extra working hours and an increase of human resources costs, and will increase the total development time.

2.4.3 Support required by other team

In case other teams require assistance to develop their products as a result of workload imbalance between teams,

- **Risk**: Medium.
- **Mitigation**: The organization of weekly meetings with the whole team will ensure that everyone is in sync with the work being done by every team and allow for a smoother transition of members between tasks, should the scenario arise.
- **Solution and consequences**: It will be critical to have a quick adaptation process in order to effectively jump to help in the development of a completely different component. Extra meeting time will be required to completely understand the details of what the other team is working on, and the cost of human resources will increase as a consequence.

Also, the time spent helping the other team will be time lost, and the product's development will slow down.

2.4.4 Security related holds

In all enterprise products, security is a highly important aspect. This is why the security of the project's infrastructure will be constantly reviewed. As a result, it is possible that a component needs to be reimplemented if a vulnerability is found. This will mean an increase of the development time.

- **Risk**: Medium.
- **Mitigation**: To avoid having to redesign the infrastructure or reimplement a component in order to meet security standards, it will be mandatory to have it in mind at all times. This means that, specially when starting new tasks, security aspects will not be ignored and, before using a new tool, a proper safe environment with secure access will be set up with the help of the IT administrator. Not only will it minimize this risk, but also ensure that the product is secure.
- **Solution and consequences**: In case a vulnerability is found, changes to the code and the infrastructure will have to take place. This will mean extra working hours for the developers and the IT manager, and possibly having to set up new Azure services (such as VPNs or firewalls). So we will see an increase in human resources and development costs.

3 Budget

In average, a year has 250 workdays. With 8 daily work hours, this accounts for a total of 2000 work hours for each year. We will use this information to calculate the hourly cost of each item.

3.1 Human resources

For the components of the product in scope of this thesis, a total of 6 workers will be needed: a project manager that leads and organizes the project, an IT manager in charge of providing support with the cloud infrastructure and security and 3 developers, which will be data engineers. The estimated total cost of each role is the following:

- **Project manager (3.348 €)**: In Spain, a project manager has a gross income of about 46.000 €per year [38]. This amounts to an hourly gross

wage of 23 €, which translates to a cost of 31 €after including the social security costs. The average time spent on the project by the project manager each week is of 6 hours. Having a total of 18 weeks for the development of the product, the total workload in hours for the project manager is of 108 hours. This translates to a total cost of 3.348 €.

- **IT manager (892 €)**: In the case of IT managers, the annual gross income in Spain is 33.300 €[39]. The equivalent hourly gross wage is 16,5 €, which after including the social security turns into 22,3 €. The expected worked time for the IT manager is of 40 hours, taking into account the time consumed setting up all the needed cloud resources and the time spent providing support to the developer team when needed. The total cost of the IT manager will be 892 €.
- **Data engineers (30.556 €)**: Lastly, a data engineer working in Spain has a mean income of 31.000 €per year [40]. Calculated hourly, this represents a gross wage of 15,5 €, or 20,9 €with the social security accounted for. For the development of this project a total of 3 developers will be required, with two of them working 5 daily hours in this project, and the third one (me, the author of this thesis) working 7 daily hours on the project. This accounts for a total of 17 daily hours of developer's work. Having 86 work days assigned to the project, the total amount of worked hours by the developers will be 1.462 hours. As a total, the cost of the developing team will be of 30.556 €.

These are the costs of the human resources by role, but in order to properly monitor the deviations in budget, they need to be translated to costs of each task. For that purpose, table 2 lists every task with the estimated hours of work of each person for that task, and shows the cost associated to it.

After adding up the cost of the three roles participating in the project, the total cost of the human resources is of 34.795,8 €. This is the total personnel costs per activity (PCA).

3.2 Development costs

The cost of the tools used for the development of the product are basically the costs of running the infrastructure on Azure. We will use Azure's Price Calculator [41], a tool provided by Microsoft, to estimate the costs of using the cloud service. After listing all the resources in section 2.3.2, the expected cost per month returned by the calculator is of 1.354,33$ (see table 3), which in euros translates to 1.171,35 €. As the project development will take 4 months, the final cost will be of 4.685,4 €.

It is important to note that the Azure costs are closely related to its usage.

Tasks		Hours worked by role				
ID	Cost (€)	P.M. 31€/h	I.T. 22,3€/h	D.E.1 20,9€/h	D.E.2 20,9€/h	D.E.3 (me) 20,9€/h
PM	8.329,3	88	15	46	46	160
PM1	499,5	6				15
PM2	541,3	6		1	1	15
PM3	499,5	6				15
PM4	1.874	20				60
PM5	4.915	50	15	45	45	55
ED	14.135,5	20	25	195	195	230
ED1	1.254			20	20	20
ED2	2.097	20	10	20	20	20
ED3	2.842,5		15	35	35	50
ED4	4.389			70	70	70
ED5	2.926			40	40	60
ED6	627			10	10	10
DA	12.331			190	190	210
DA1	418					20
DA2	627			10	10	10
DA3	6.270			100	100	100
DA4	5.016			80	80	80
TOTAL	34.795,8	108	40	431	431	600

Source: Own creation.

Table 2: Tasks cost estimation

Service type	Estimated monthly cost
Storage Accounts	$101,48
Azure Functions	$0,00
Azure Database for PostgreSQL	$634,12
Azure Cosmos DB	$375,40
Key Vault	$0,03
Azure Machine Learning	$198,56
App Service	$13,14
Azure Maps	$0,00
Container Registry	$31,60
Total	$1.354,33

Source: Azure Price Calculator [41].

Table 3: Azure cost estimation

This means that the costs shown here are calculated from the development view, as they reflect the expected usage during the implementation of the project. There is a real possibility that, during development, the client re-

quests to put in production one or more completed components of the product, and therefore scale up the usage and costs of resources. This will not be reflected here, as it is not a development-related costs, but rather a production cost that will be assumed by the requesting client.

3.3 Other costs

Here we will list all the remaining generic costs of the project. As a result of the COVID-19 pandemic, every worker of the company is working remotely, so the entirety of the product will be developed by workers working from home. This means that costs related to the usage of an office will not be considered, but other costs associated with working remotely do. It is estimated that the hourly cost of each worker working from home, regardless of their position, is divided into these categories:

- **Laptop (0,15 €/h)**: The current laptop provided to all workers by the company is a Dell Latitude 5401. The cost of this model when bought new was approximately 1.200€[42]. The laptops given by the company to the workers are renewed every 4 years, and are therefore used for 8.000 hours. The hourly cost of the laptop is of 0,15 €.
- **Peripherals (0.0305 €/h)**: Workers also need to be provided with peripherals to be able to work with comfort and be productive. They will need to be provided with headsets to participate in meetings, a mouse to use the laptop with more ease, and a chair and desktop to have an office space. Although workers might already have some or all of these peripherals as their personal property, the company must cover these costs as they are all work-related tools. Table 4 provides a quick summary of the cost of each of these peripherals:

	Price (€)	Life expectancy (years)	Hourly cost (€)
Headset	25	3	0,0042
Mouse	15	4	0,0019
Chair [43]	120	4	0,015
Table [44]	150	8	0,0094

Source: Own creation.

Table 4: Generic hourly costs

  The total cost per hour of these peripherals is 0.0305 €.

- **Internet (0,24 €/h)**: A good internet connection is a basic requirement for remote work. The average monthly cost of fiber optic internet in Spain is of 40 €, 480 €each year. The proportional hourly cost (with 8 daily worked hours) is of 0,24 €.

- **Electricity and gas (0,045 €/h)**: The last expense related to telework is the consumption of electricity and gas (heating during winter). Their estimated yearly cost due to remote work is of 90 €[45], which translates to 0,045 €per worked hour.

Adding all these costs together, for each worker the company has to spend an extra 0,4655 €for each worked hour. Going back to the work hours estimated for each role in the human resources costs (section 3.1), the total expected worked hours for all collaborators are 1610 hours. Therefore, the final generic cost for this project adds up to 749,45 €.

With the development costs and the remaining costs related to workspace tools calculated, the final generic costs (GC) can be obtained by adding them, which results in a total of 5434.85 €.

3.4 Budget deviations

3.4.1 Contingency

There is a clear risk of having a deviation in the project's budget. External factors can affect the development time, or a simply incorrect estimation of the difficulty or cost of a task can end up in higher costs than expected. To account for this phenomenon, a contingency fund will be put in place to provide a safety monetary margin and avoid over costs. This contingency will be the 15% of the estimated PCA and GC, which accounts for a total of 6.035 €.

3.4.2 Incidental costs

In section 2.4, we defined a list of risks that can affect the project. These risks can result in an increment of the cost of the project, and need to be taken into account. Table 5 shows a summary of these risks, their estimated cost, the risk of them occurring and the expected final cost.

3.5 Final budget

Having listed and estimated every cost related to the project, including contingency and incidental costs, the final budget can be laid out. Table 6 shows the summary of all costs and the final project cost.

Incident	Estimated cost (€)	Risk (%)	Cost (€)
Change of requirements	7.000	20	1.400
Dependency on other teams	1.500	70	1.050
Support required by other team	1.000	40	400
Security related holds	2.000	50	1.000
Total	11.500	-	3.850

Source: Own creation.

Table 5: Incidental costs

Activity	Cost (€)
PCA	34.795,8
GC	5.434,85
Total costs (CPA + GC)	40.230,65
Contingency	6.035
Incidents	3.850
TOTAL	50.115,65

Source: Own creation.

Table 6: Final budget

3.6 Management control

Having created a budget, it is crucial to keep track of the real costs of the project as it is implemented, in order to check if it differs from the budget and by how much. Keeping track of where deviations of the budget occur, and their magnitude, is a task that needs to be constantly done, specially after finishing each task.

Each cost group (human resources, development costs and other costs) will be monitored differently, in order to properly identify their potential deviations correctly based on the individual characteristics of each cost group.

- **Human resources**: The costs derived from the human resources are strictly limited to the hours worked by each member of the project's development. Therefore, we will keep track of the exact hours worked by each individual and use the following formula to calculate the cost deviations:

$$Cost\ deviation = \sum_{w \in workers} (w_{worked\_hours} - w_{estimated\_hours}) * w_{hourly\_cost} \tag{1}$$

- **Development costs**: The development costs can be very easily monitored. The Azure portal has a detailed monitoring tool that will do the

work, so the only thing that will have to be done is checking the costs shown by this tool periodically. The formula used will be:

$$Cost\ deviation = \sum_{t \in dev\_tools} t_{real\_cost} - t_{estimated\_cost} \tag{2}$$

- **Other costs**: Almost all of these costs are fixed. The costs of the peripherals, laptops, tables, and others will not change, because if one of them needs to be replaced it can be done without affecting the budget, as the company already expects to replace these elements periodically. The only cost that can change will be the compensations for working from home. If there is a relevant fluctuation in the price of energy, the compensation for workers might need to be reviewed, and the deviation in the cost monitored. The general formula used to monitor these deviations will be:

$$Cost\ deviation = \sum_{c \in other\_costs} c_{real\_cost} - c_{estimated\_cost} \tag{3}$$

A periodic review of the deviation of costs will be made, and with the closure of the development of the project a detailed report of cost deviations will be written to show where they have occurred and their magnitude.

4 High level architecture

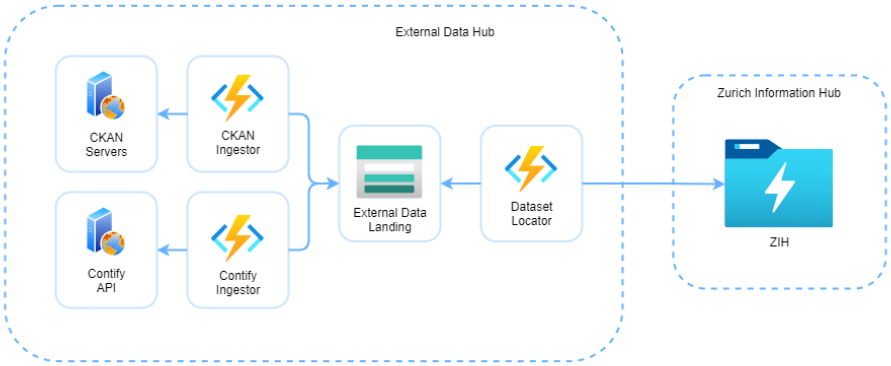
The first phase of the project is designing the architecture. The goal is to design a first solid version with the core components of the product, which will evolve as the development advances. It is important to design an architecture with decoupled components, that way the introduced changes and addition of new components will have a minimal impact on the already developed software.

The architectures of each of the two components developed as part of this thesis will be designed and explained separately, as they are not directly connected. However, data extracted by the EDH can we later used by the GeoIntel app.

4.1 External Data Hub

The External Data Hub will have a set of ingestion algorithms that will extract data from external (not company-related) data sources. An ingestor will be developed for each source type, allowing for the usage of a common configurable ingestor for different data sources that share the same extraction method. An example is the CKAN extractor, which consists of a configurable ingestor that can extract any type of data hosted in CKAN servers. These

ingestors will be developed as Azure Functions, which will be periodically time triggered. They will ingest the data and store it unprocessed in a landing data storage. This data will then be processed by another Azure Function that will be in charge of loading it into the Zurich Information Hub. Figure 2 shows the different components of the EDH and how are they related between them.



Source: Own creation.

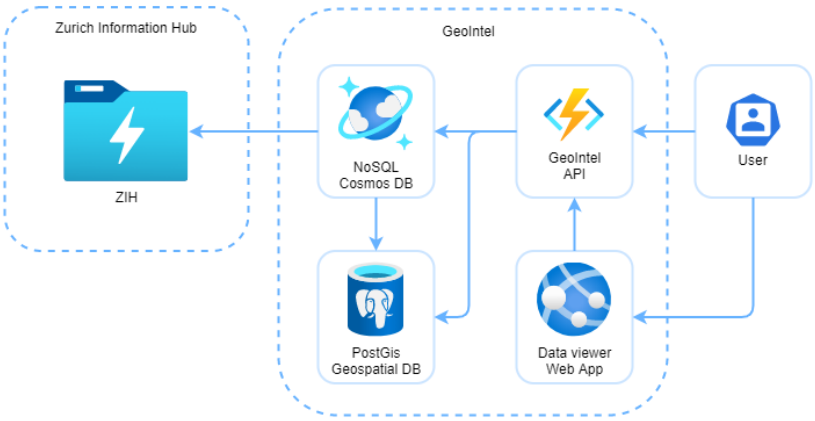
Figure 2: External Data Hub architecture

4.2 GeoIntel

The GeoIntel application will start by loading processed geospatial data from the Zurich Information Hub. The data will be divided in two databases: a Cosmos DB and a PostgreSQL DB with the geospatial PostGIS extension installed. For each dataset, the Cosmos DB will store all the data except the geospatial information, and will load into PostgreSQL the geospatial information alongside an ID acting as a foreign key, using either an azure function or a stored procedure (not designed yet).

These databases will be the source of data for the GeoIntel API. This API is the backbone of the service, and will be implemented using an Azure Function. This API will be defined with a swagger file, following the OpenAPI specification. The end user will be able to use this API directly, but a web app will also be created for a more user-friendly experience. This website will be implemented using the Flask [46] web framework, and deployed to an Azure App Service.

A representation of the GeoIntel architecture can be seen in Figure 3.



Source: Own creation.

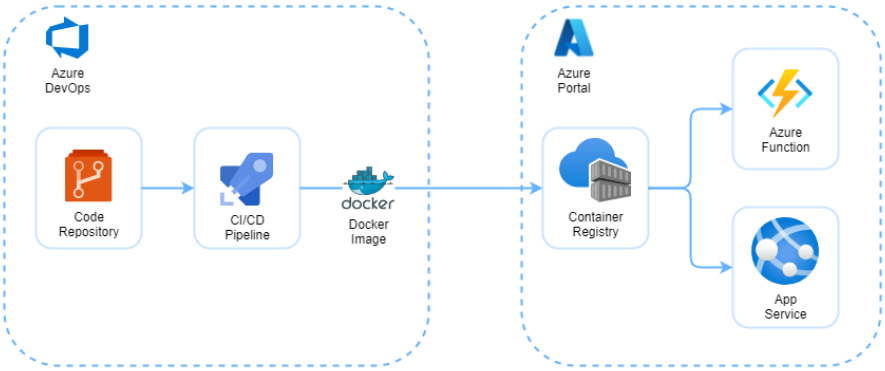
Figure 3: GeoIntel architecture

4.3 DevOps

Finally, a crucial step of the project is working with a CI/CD [14] infrastructure. Software needs to be deployed iteratively as the development advances, and once a first minimal version of the components is working, they must be always ready to be used, as the project manager will be showing it to current and potential customers. Updates and enhancement of the components will be progressively deployed as they are implemented.

As mentioned in section 1.4.1, several tools from the Azure DevOps [13] platform will be used for this purpose. Figure 4 shows the general architecture of this infrastructure.

The developed code will be saved and versioned with the help of repositories [16]. These repositories will have several branches for development, and a main branch that will be where the currently deployed code is. Whenever a new version of the main branch is detected, a pipeline [17] will be launched. This pipeline will perform a quality check of the code and run automated tests to ensure the correctness of the new code. If all checks pass, it will create a docker image and publish it to a container registry [26] hosted in Azure. As a final step, the corresponding Azure service will pull the new docker image and run it in a container.



Source: Own creation.

Figure 4: DevOps architecture

5 External Data Hub

5.1 Ingestors

The ingestors are the main component of the External Data Hub. They are in charge of extracting data from external sources and storing it inside the company's infrastructure, so that it can be accessed and used by business units and company's applications.

The implementation is done with Azure Functions written in Python 3, and although each ingestor has some specifics to it, they all share a common structure. All ingestors have two types of triggers: a time trigger and an HTTP request trigger. This way, data is extracted periodically in order to fetch updates, and can also be triggered manually when needed via the HTTP trigger.

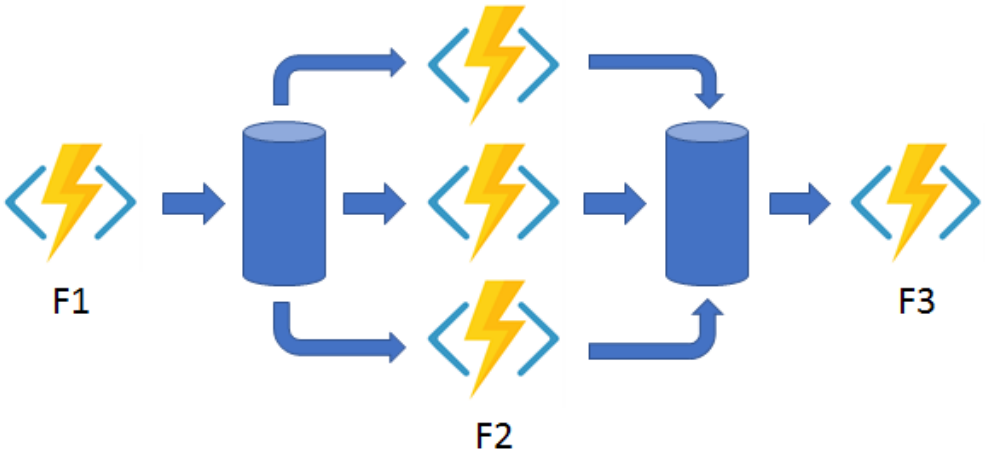
Once triggered, the function starts its execution. It connects to the corresponding data source, using credentials stored in the product's key vault (if they are necessary), and then uploads the extracted data to the External Data Landing storage.

- **CKAN**: CKAN is a type of open data server standard used by many companies and public organizations to publish their data openly to the public. The ingestor takes advantage of this open standard and therefore can ingest any data source that is hosted in a CKAN server. It is therefore a very powerful tool, as it is not source specific. Examples of organizations that use this standard are the humanitarian data exchange [47] or governments like the United Kingdom or Australia. Therefore,

with this single ingestor we can ingest thousands of dataset collections.

As the content of these servers can be very large, a traditional durable function cannot be used for the ingestion. They have an execution time limit, so ingestion of large amounts of data is not possible. To get around this, Azure Durable Functions will be used. These allow for stateful execution and are not limited in execution time. They also allow for the implementation of several application patterns.

For this particular scenario, we will use the fan-out/fan-in pattern [48]. It consists in having an orchestrator that receives the input and generates a list of tasks to be executed, which are then processed by workers in parallel. The orchestrator then collects the results from the workers, aggregates them if necessary, and returns a response (figure 5).



Source: Microsoft Docs. Durable Functions overview [48].

Figure 5: Fan-out/fan-in durable function pattern

This pattern does not only achieve parallel execution, it also simplifies the required code. The orchestrator will be triggered with an HTTP request, that will contain in its body the URL for the target server for the ingestion a and a list of datasets from the CKAN server to ingest. An example for the ingestion of a single dataset from the *United Kingdom Government's open data CKAN server*:

If the dataset list is empty, all datasets in the server will be ingested. The orchestrator will then generate an activity for each dataset that must be ingested, and wait for the completion of all the ingestion processes, which, as said before, will be executed concurrently.

- **Contify**: This ingestor uses the Contify News API [49] to extract data related to a given set of companies around the world. More specifically, it retrieves daily updates of news related to these companies. It is a more

specific ingestor than CKAN, so fewer datasets will be ingested with it, but the data obtained will be highly valuable for the company.

A list of companies and entities of interest for the company will be used to configure the ingestor. For every company that is in this list when the ingestion is triggered, its related news and events will be ingested from Contify and stored in the EDH landing storage as a JSON file. This list is mutable as the companies of interest for Zurich change over time. The ingestor will read it as its configuration and ingest the desired data, so no changes will need to be made to the product if the list changes, as the code will adapt automatically to it.

More ingestors will be added in the future, as new types of open data sources are required by the company. This can be easily done without affecting other ingestors or components of the EDH, and using the same code structure used by the above listed ingestors, with just an adaptation to the specific data source.

5.2 External data landing storage

The external data landing storage is where raw ingested data is placed before being sent to the ZIH for processing. It is saved in a blob storage, where a specific container for the landing data zone has been created (the storage has other containers that are automatically created and used by other Azure services, such as Durable Functions [31]).

Each ingested dataset is stored inside a specific path of the form:
`externaldata-landing/<data-source-name>/<dataset-name>/<files>`.

5.3 Dataset locator

Due to a change in the client's requirements, the dataset classification component has been changed to the dataset locator. Instead of classifying datasets into predefined categories, geospatial information will be extracted from them. More information on this requirement change can be found in section 8.2.

The data ingested will be analyzed by the dataset locator component to try to extract location information from the dataset, and therefore being able to store it in a geospatial database and use it in the GeoIntel product. If the dataset contains geospatial data, it will be processed and transformed to adapt it to the requirements of the geospatial database. More precisely, all available geospatial data will be combined into a single geometry object, which will be then used as the location of the dataset in the geospatial database.

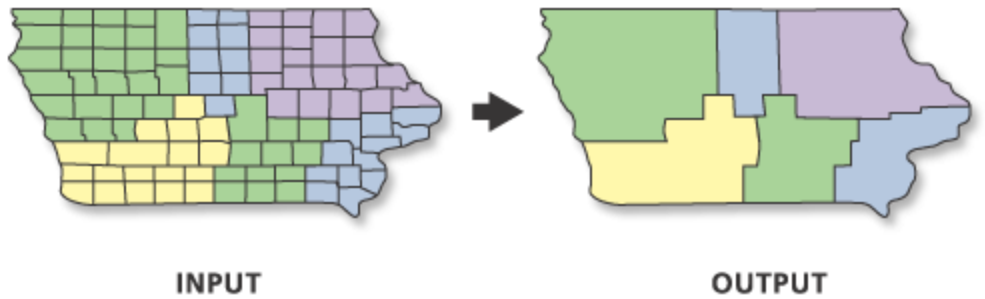
However, this will be made just to geolocate the dataset, and its contents will not be modified, and therefore the original individual geometry objects will continue being present in the dataset. For datasets that do not contain any explicit geospatial data, the dataset locator will try to generate it based on the contents of the dataset.

The dataset location function is divided into these two categories: geospatial data extraction and geospatial data generation. For the explanation of this application, the *0.1% annual probability extents* dataset is used as an example. It has been obtained with the CKAN ingestor from the *United Kingdom Government's open data CKAN server*. The results are plotted with the help of Folium [50], a Python map visualization library based on Leaflet.js [51].

5.3.1 Geospatial data extraction

Geospatial data (1.1.2) can be represented in a variety of file formats, such as GeoJSON [52], Google Keyhole Markup Language [53] or Shapefile [54], among others. The geospatial data extraction process consists in going through all the files inside the dataset and trying to load them into GeoPandas [55], which is an extension of Python's [56] data analysis library that adds support for geospatial data. All files that contain geospatial data are loaded using GeoPandas into a geospatial data frame, which is a regular data frame with a column containing geospatial data. The rest of files are ignored.

Once loaded as a data frame, all the contained geometries within the datasets are aggregated, as the goal of this functionality is extracting a summary of the contained geodata, not the whole available geodata. As an example, a geometry containing the polygons of all European countries can be dissolved into a single polygon of all Europe. Figure 6 shows a representation of this process. This step also in reducing the complexity of the computations for the function, as the granularity of the geometries is reduced.



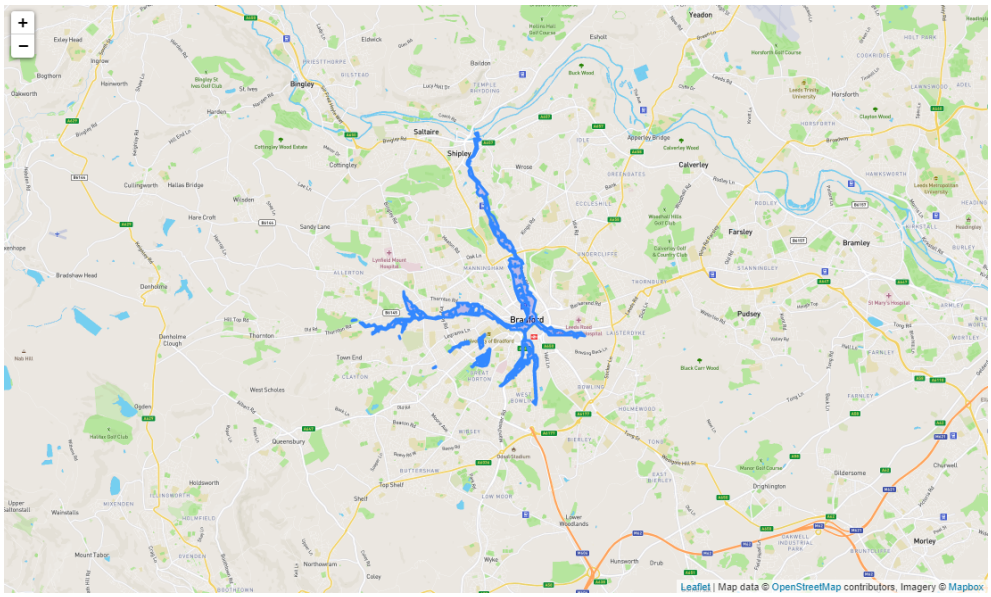
Source: EarthDatasSience.org. GIS in Python [57].

Figure 6: Geometry dissolving

Then, the geodata from each file has to be standardized into a common CRS

1.1.2, which will be EPSG:4326, and then a union of all the data is performed. This results in a single geospatial object, representing the union of all geometries present in the dataset. It is returned using the GeoJSON [52] standard.

In figure 7 the plotted GeoJSON generated for the example dataset can be seen.



Source: Own creation.

Figure 7: Extracted geodata

5.3.2 Geospatial data generation

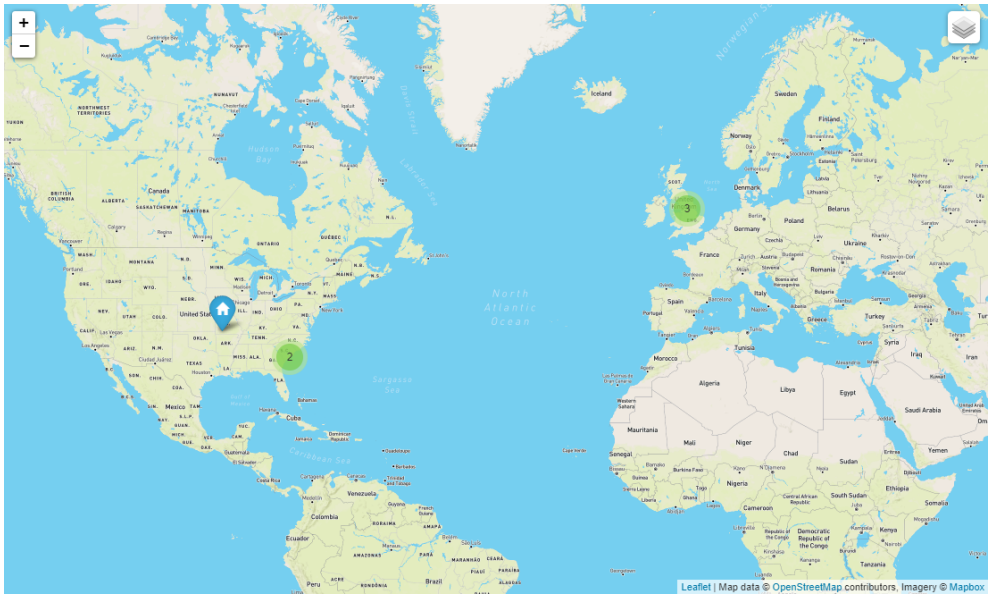
When a dataset does not contain explicit geospatial data in it, this functionality will attempt to extract implicit geodata from the dataset and generate a GeoJSON that geolocates the data.

For this task, a NLP [29] technique called Named-entity recognition [58] is used. It consists in using language models to extract key entities from a text, which are classified into categories. These entities can be a person, location, organization, etc. For this application, only entities that are categorized as locations are used. These are extracted from the text and then geocoded, and the resulting coordinates are used to generate the geospatial information that locates the dataset.

The first required step is extracting raw text from the data. For this, the tool used is Textract [59], which is a Python module that extracts raw text from a wide variety of file formats, even images and audio files. This allows the function to read almost any file format that contains text in it.

Once the text has been obtained from the file, the NER process starts. This is done using the spaCy [60] library, which is an NLP engine for Python. It has multiple language models available, and the one that is used for this task is the ***xx_ent_wiki_sm*** [61] model, because it is a multi-language model with very good NER capabilities. The datasets that will be analyzed by the application can come in many languages, so it is important to choose a multi-language NLP model. Using the NER feature, location entities will be retrieved from the dataset's extracted text.

Once the list of extracted locations is created, they are geocoded with the help of the GeoIntel API, using its geocoding functionality 6.3.1. As the list may contain repeated entities, a single geocoding call is made for each unique entity. However, the count of appearances of each entity in the dataset is kept, as it represents its weight in the dataset's locations set. In figure 8 the geocoded entities extracted from the example dataset are plotted.

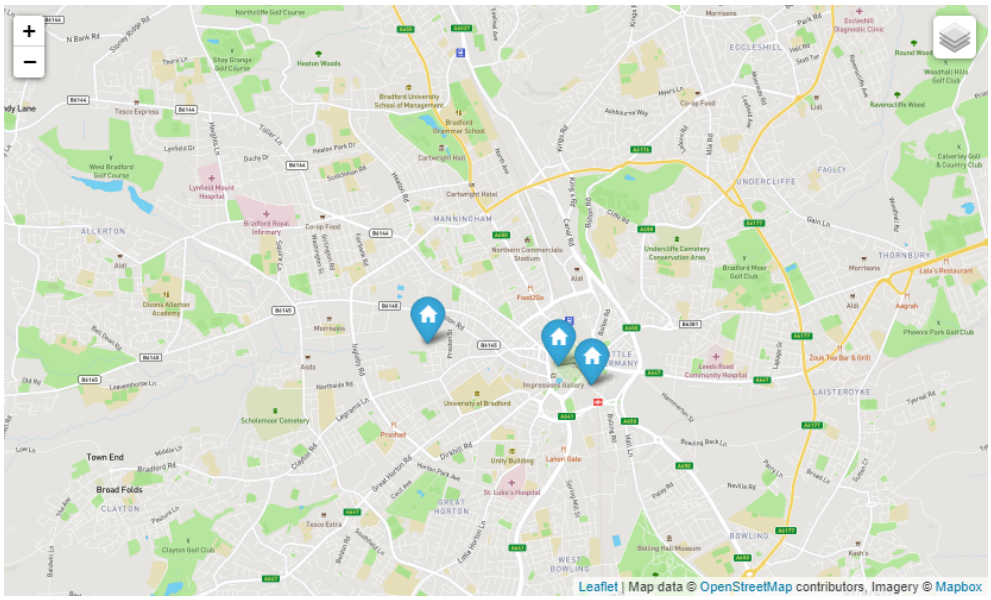


Source: Own creation.

Figure 8: Geocoded location entities

The extracted locations are potentially spread across a large area of a country and even the world. In order to reduce the area covered by the locations and obtain the most probable location of the dataset, clustering is used. It is important to note that, as said earlier, the count of appearances of each location in the dataset's text is highly relevant, as the clustering method not only uses the unique locations found, but the total amount of locations, including repeated ones. This ensures that a location that is present multiple times in the dataset's text has more weight in the clustering algorithm than a single isolated location. The OPTICS algorithm [62] is the used method to find clusters in our locations. It is an algorithm that finds density-based clusters in spatial

data. It's based on DBSCAN, but it is better suited for this application as, unlike DBSCAN, it works with data of varying density, which is the case for this scenario. The OPTICS algorithm is executed using the implementation available in the Scikit-learn package [63]. Once the algorithm has executed, the biggest cluster is selected, that is, the cluster with the most amount of locations in it, points not belonging to it are discarded. Figure 9 shows the plotted locations for the biggest cluster found for the example dataset.



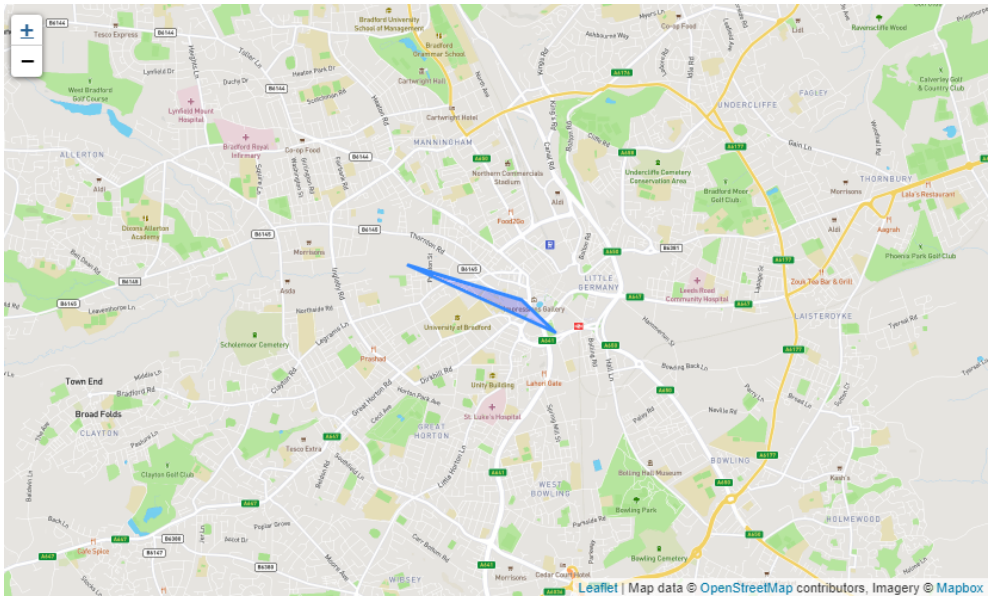
Source: Own creation.

Figure 9: Geocoded location entities (close-up)

The locations in the biggest cluster are considered as the final locations that geolocate the dataset. However, they are specific points, as the geolocation function returns a single point in Earth's surface, but the reality is that a dataset will contain data of an area, not specific points inside this area. Therefore, some geospatial transformations need to be done to the set of locations to generate an area that covers them and their surroundings.

Using GeoPandas [55] and Shapely [64] (a Python library that is used by GeoPandas to perform geospatial operations), a bounding polygon for the locations is generated. A bounding polygon is the smallest possible polygon that covers all the points, where each of its vertices is a point belonging to the set being covered. Figure 10 shows the bounding polygon found for the example dataset.

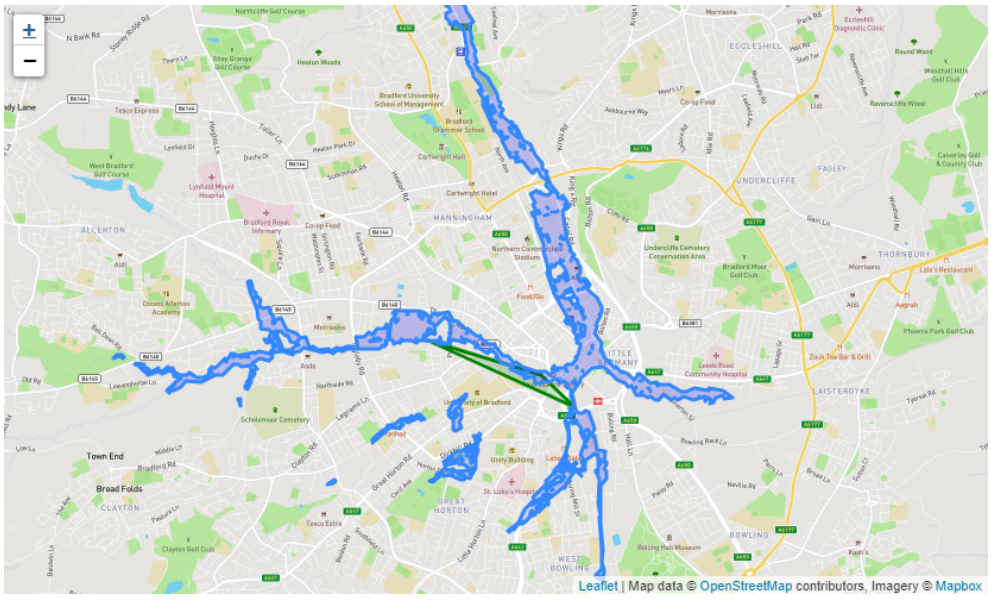
However, this polygon is the smallest possible polygon that covers all the points, and therefore only covers the area between all the points, but areas that are very close to the vertices of the polygon but don't fall inside it are not considered as part of the dataset. To illustrate this, it can be seen in



Source: Own creation.

Figure 10: Bounding polygon for cluster

figure 10 that the generated polygon covers a very small area of the city, and when comparing this polygon to the one extracted for the same dataset by the geospatial data extractor (figure 11), they barely overlap.



Source: Own creation.

Figure 11: Extracted geodata and generated geodata (bounding polygon)

Furthermore, imagine the case where the set of unique geocoded locations are

just one or two points. This would mean that the bounding polygon would be a single point or a straight line, and therefore cover no area at all. It is clear that a method must be implemented to transform this polygon into a bigger area that solves the previously mentioned issues.

The solution is the usage of the buffering technique. It is a geospatial operation that, given a geometry object, it generates a buffer polygon around it to a specified distance [65]. For this scenario, the distance for the buffering has to be proportional to the bounding polygon size.

In order to determine the size of the polygon, the maximum distance between two of its vertices is used. To measure these distances, the Haversine formula [66] is used as a metric, as it is required to account for the curved surface of the Earth. This formula states that the distance D between two coordinate points (X_{lat}, X_{lon}), (Y_{lat}, Y_{lon}) on Earth's sphere, is the result of:

$$a = sin^2(\frac{X_{lat} - Y_{lat}}{2}) + cos(X_{lat}) \cdot cos(Y_{lat}) \cdot sin^2(\frac{X_{lon} - Y_{lon}}{2})$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a})$$

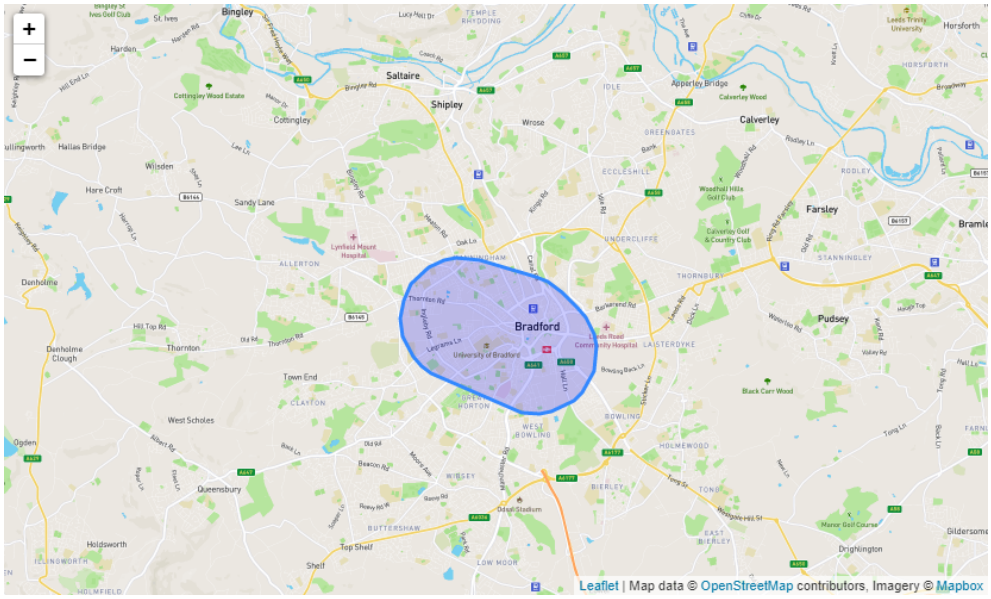
$$D = R \cdot c$$

where R is the radius of the Earth. As the units for the EPSG:3857 [8] CRS are meters, we will also use meters for the Haversine formula, and therefore $R = 6.371.000$. As there are n^2 pairs of vertices in the polygon, the computation of the maximum distance between two of its vertices has a complexity of $O(n^2)$.

Having calculated the size of the polygon, the final value set for the buffering distance is proportional to this distance and with an extra factor that increases the buffering in proportion to the amount of points in it, as a single point location would produce a point as a bounding polygon, which has a distance value of zero and therefore no buffering would be applied to it. In figure 12 the buffered polygon for the example dataset is shown, and figure 13 shows the comparison of the final generated geodata with the extracted data.

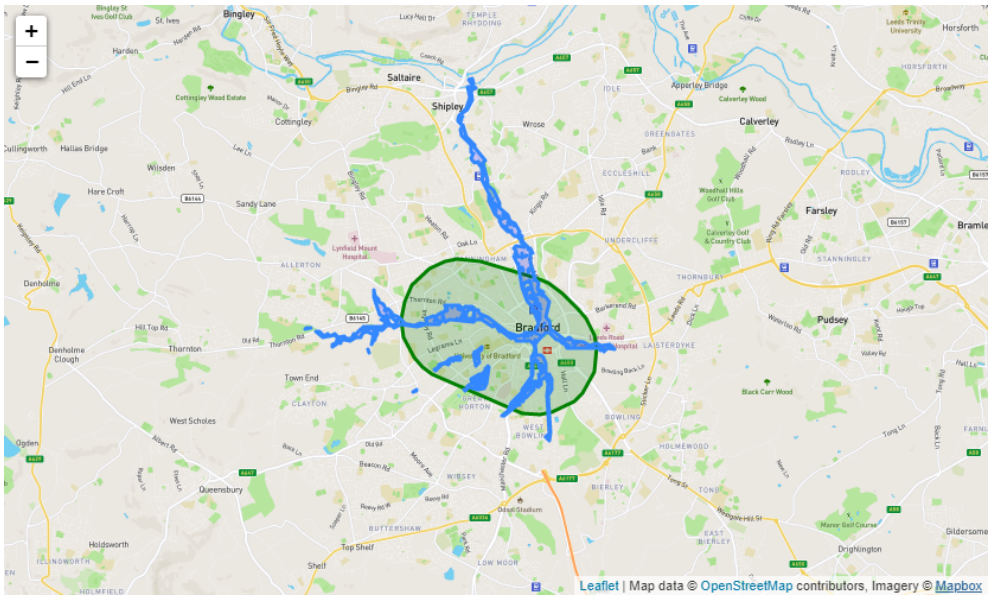
Compared with figure 11, the improvement can be clearly seen, as the intersecting area between the generated and extracted polygons is much greater.

The resulting polygon from the buffering step is the geospatial information that is assigned to the dataset. As with the geodata extraction method, it is returned using the GeoJSON standard.



Source: Own creation.

Figure 12: Generated geodata



Source: Own creation.

Figure 13: Extracted and generated geodata

5.3.3 Geospatial data extraction evaluation

Methodology For the geodata generation function, it is crucial to test its performance. The methodology used consists in iterating over ingested datasets from the *United Kingdom Government's open data CKAN server*

and running the geodata extraction method. Then, for each dataset that contains geospatial information, the geodata generation method is executed as well, and the geometries returned by the two methods are compared. If they intersect, it is considered as a match. The formula used to evaluate the performance will be:

$$\frac{|\{E_n \cap G_n \neq \emptyset\}|}{|\{E_n \neq \emptyset\}|} \tag{4}$$

where, for each dataset n, E_n is its extracted geometry and G_n its generated geometry. Furthermore, all datasets for which their extracted and generated geometries do not intersect are classified in two groups: datasets with generated empty geometries and datasets with non-empty generated geometries, which is in fact a classification between datasets with incorrect generated geometries and datasets with no generated geometries. This allows for a distinction between incorrect geodata generations and empty generated geometries. It is important to note that the NLP processing of the dataset does not extract information from the geospatial files. All the information used to generate the geodata comes from the text contained within the dataset.

Results A total of 44.022 datasets have been used for the test, from which 1.865 had geospatial data in them. For datasets having geospatial data, the generator correctly generated a geometry for 1.464 of them, and failed to do so for 401 . For the failed cases, 365 of them were incorrect generations, and the remaining 36 were the consequence of the generator not being able to find a location and generating an empty geometry as a result. Table 7 shows a summary of these results.

<table>
<tr><td colspan="4">Total datasets</td></tr>
<tr><td colspan="4">44.022</td></tr>
<tr><td>Without geodata</td><td colspan="3">With geodata</td></tr>
<tr><td>42.157</td><td colspan="3">1.865</td></tr>
<tr><td rowspan="3"></td><td>Correctly generated</td><td colspan="2">Incorrectly generated</td></tr>
<tr><td>1.464</td><td colspan="2">401</td></tr>
<tr><td rowspan="2"></td><td>Wrong generated geometry</td><td>Empty generated geometry</td></tr>
<tr><td>365</td><td>36</td></tr>
</table>

Source: Own creation.

Table 7: Geospatial data generation tests

Using formula 4 (and converting it to a percentage), we obtain a total of **78,5%** correctly generated geometries by the dataset locator. This is a very

good result that allows for the usage of the dataset locator component in the ingestion pipeline to geolocate datasets which do not contain any geospatial information in them.

6 GeoIntel

6.1 Cosmos Database

The data loaded from the ZIH is stored in a Cosmos DB. Each dataset has its own container (similar to a table in a relational database), where every data point is stored as a document. Each document is represented with the JSON format, and even though it is likely that documents within a same collection have the same fields, they represent unstructured data and therefore can have varying structures. It is important to note that the ID of each document is the same ID used in the geospatial database for the geospatial data of the document, and can therefore be used as a foreign key between the two databases.

6.2 PostgreSQL with PostGIS

A PostgreSQL database with the PostGIS extension is used to store the geospatial data of each dataset and to perform geospatial operations. This database has a table for each dataset, which contains the geospatial information of the dataset, the date (if available) of the represented data and an ID referencing the corresponding cosmos document (table 8). The database also has some extra auxiliary tables containing needed geospatial information to perform geospatial queries. They contain geometries (formed by polygons and multi polygons) representing countries, districts and neighborhoods (table 9).

Column name	Type
ID	text (UUID)
date	text (UTC)
location	geography

Source: Own creation.

Table 8: Dataset table structure

Column name	Type
name	text
location	geography

Source: Own creation.

Table 9: Polygons table structure

The PostGIS extension has two geospatial data types: geometry and geography. As their names indicate, the first type is used to work in a geometric space, while the geography type is used to work in the geographic space, that is, the surface of the Earth. As we are working with geospatial data, we will use the geography type. If we used geometry, the data stored would not be

represented accurately and geospatial operations would produce incorrect results, as we would be treating Earth's surface as a flat surface and not a curved one. Surely, this will upset some flat earthers.

Being able to run queries as fast as possible is mandatory, as the users will expect an immediate response from the API or the web app. To achieve this, indexation [67] is being used. In the case of the dataset tables, an index is constructed on both the date and location columns, with the goal of speeding up geospatial queries and also the filtering of data based on date. Indexation of the polygons in the auxiliary datasets is also being done to complete the indexation of all geospatial data in the PostgreSQL DB and therefore ensure maximum speed.

The indexation of geospatial information works differently from generic indexation. Instead of using hierarchical trees based on the values being indexed, the bounding boxes of geospatial objects are calculated and then indexed. When a geospatial operation or query is performed by the database, it is first run against the indexed bounding boxes, and then against just the geometric features that are covered by the matching bounding box [68].

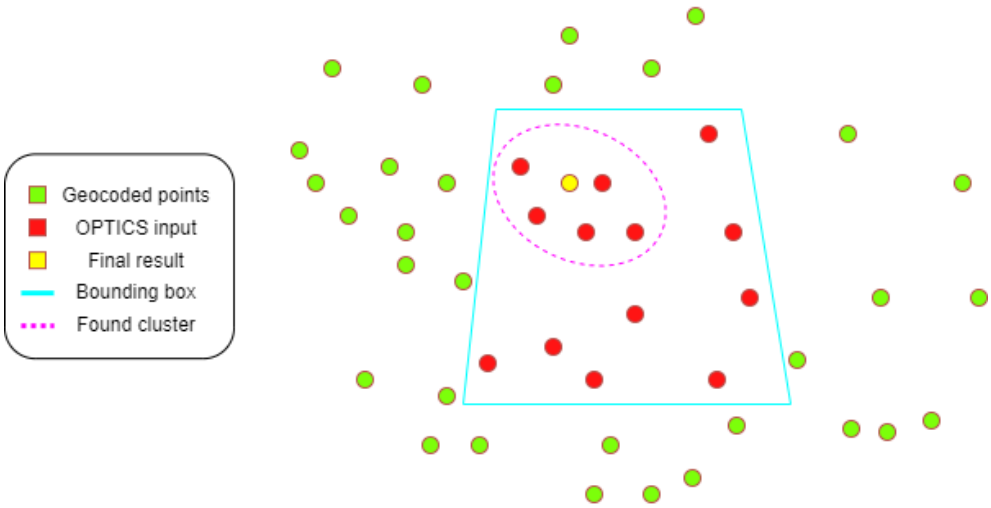
6.3 API

The API is the backbone of GeoIntel and the main feature of the product. It is seen as the main point of interest for clients and also serves data to the viewing website.

The API is implemented using an Azure function written in Python 3. It has several functions that have a variety of functionalities, which fetch data from the Geospatial databases and external geocoding APIs. The main functions provided by the API are:

6.3.1 Geocoding

This function provides the API with a geocoding functionality. It takes raw text as input, and the function returns a specific location (as coordinates) inferred from that text. It first checks if the inputted text are coordinates, in any notation. If they are, it parses them. Otherwise, the function analyzes the text to extract location information and then calls several geocoding APIs (such as Azure Maps, TomTom or OpenStreetMap) to geocode the addresses. The function can use a defined bounding box to reduce its search. If specified, all geocoded coordinates that fall outside this bounding box are discarded before moving to the next step.



Source: Own creation.

Figure 14: Geocoding illustration

Once addresses are geocoded, the OPTICS [62] clustering algorithm is used, just like for the geodata extraction process (5.3.1). The biggest cluster is selected, and the most representative point of the set is returned as the final geocoded location. This is the closest point to the centroid of the set. Figure 14 shows a representation of this process.

6.3.2 Data access

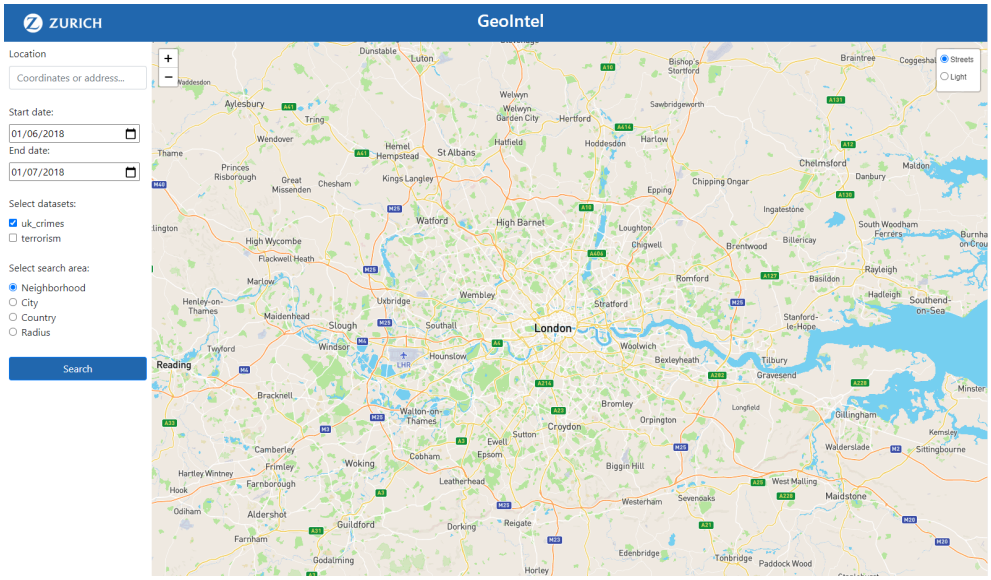
- **Data search**: This function returns GeoIntel data based on the given search parameters: a query with location information, date ranges for the search, the search area and wanted datasets. It returns all found data points matching the search criteria.
- **Data point information**: This function returns all available information of a specific data point, given its ID. It directly accesses the Cosmos DB and returns the corresponding document.
- **Dataset listing**: This simple function returns the list of available datasets in the GeoIntel app.

More functionality will be incrementally added to the API in the future as development evolves and more client requirements are given.

The credentials and access tokens for both the GeoIntel databases and external geocoding APIs are stored securely in an Azure Key Vault as secrets. The

API accesses the Key Vault when needed to fetch the credentials, so nothing is stored inside the Function App itself, ensuring secret confidentiality and security.

6.4 Data viewing web app



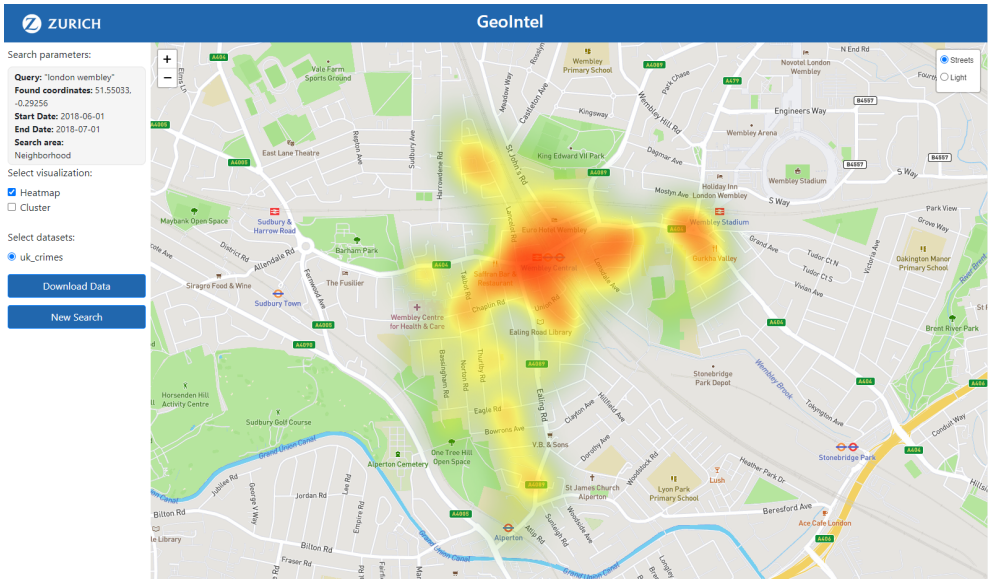
Source: GeoIntel app.

Figure 15: GeoIntel web main page

The data viewing app is implemented with the Flask [46] python web framework. It uses the JavaScript Leaflet [51] library to create an interactive map that displays the GeoIntel data retrieved via the API. The app is a simple querying and viewing tool: it has a side panel and a map (figure 15).

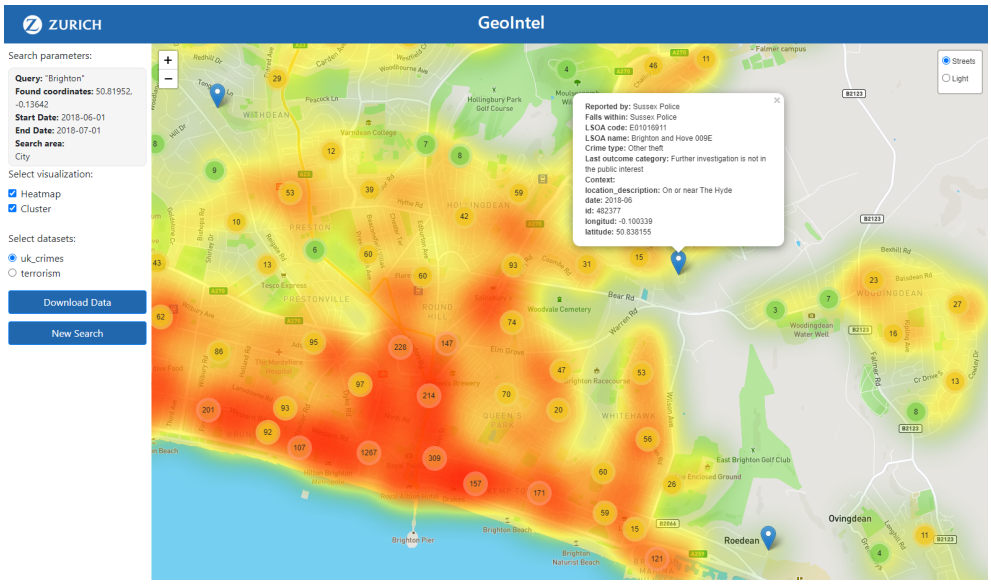
The side panel is used to make a search of GeoIntel data, and once a search has been made it is used to select the desired visualization methods and to download the obtained data. On the other side, the map is used to visualize the data and interact with it. It has two viewing methods: a heatmap and point clusters. As figure 17 shows, they can be overlapped, and with the cluster view the user can select a specific data point and view the information associated to it.

All the data accessed and viewed through the website can also be downloaded from the menu. This grants that less technically skilled users can interact with the data with an easy-to-use and intuitive portal, being able to view it directly on the app and then downloading the data segments they are interested in without having to interact with the API at all. And the data downloaded has the same format as the one served by the API, so if the user wants to interact



Source: GeoIntel app.

Figure 16: GeoIntel web heatmap view



Source: GeoIntel app.

Figure 17: GeoIntel web heatmap and cluster view

with it or process it with a different application, it can do so as well, still without the need to interact with the API to obtain it.

7 DevOps pipelines

The usage of pipelines allows the team to implement a CI/CD infrastructure. A pipeline is needed for each type of service that has to be deployed, but a single pipeline design is used for deployments of the same type. This common pipeline is configurable with parameters in order to control the specifics of each component. As we are developing code for two types of Azure services, two pipelines have been designed.

7.1 Azure Functions

The deployment of azure functions has four stages:

1. **Validation and testing**: The first stage is running quality checks and validations on the code, and then running automated tests. An environment with all the required libraries (such as Pylint and Pytest) is created, and then the code is checked, validated and tested.
2. **Publishing artifact**: Once the code has passed the previous stage, an artifact is created with the component's code.
3. **Dockerization**: Then, docker is installed and a docker image of the azure function is created.
4. **Deployment**: As a final step, the docker image is pushed to an Azure Container Registry and deployed to the corresponding Azure Function.

7.2 App Service

For the app service, the deployment is a simplification of the function deployment:

1. **Dockerization**: Docker is installed and a docker image of the app service is created.
2. **Deployment**: The docker image is pushed to an Azure Container Registry.
3. **App restart**: Once the new image has been pushed, the app service is restarted. After the restart, it automatically runs a container from the latest pushed image.

8 Project review

This section contains a summary of the development lifecycle of the project.

8.1 Encountered issues

During development, a few issues have come up. Fortunately, the planning and risk management done in the early stages of the project have minimized the impact of these problems, resulting in no significant impact on the development. The problems that were encountered are listed and below, grouped by category.

8.1.1 Azure functions limitations

During the implementation of the CKAN ingestor, limitations of the serverless mode of azure functions became apparent. For big data sources containing large amounts of data, the required time to ingest all the data exceeded the capabilities of stateless azure functions. This forced a change in how ingestors were implemented, and the solution was to still use the Functions service but with the durable functions [31] extension. They allow writing stateful functions in a serverless environment, and as a consequence can ingest large amounts of data in a single run without any time limitations.

8.1.2 Container Registry Firewall

In the previously explained DevOps architecture, the usage of a container registry to store docker images was explained. But the connection to this container registry raised some security related issues. When the CI/CD pipeline was developed and working, the platform team detected that the container registry was openly connected to the internet, which is a major security problem. A firewall was put in place, limiting access to other Azure services and connections through selected virtual networks.

This caused a connection error between the agent executing the pipeline. The DevOps agents are hosted in a Microsoft server, so they are not inside the company's network. Therefore, it could not push the docker image to the container. As a result, for two weeks, azure functions and the data viewer could not be deployed to Azure. Services could still be developed and tested locally as individual components, but the CI/CD infrastructure was stopped, so the progress being made was not being reflected in the deployed product, so the product manager was not able to showcase the product to clients for

this time period. The issue was finally solved by adding extra steps in the pipeline to modify the firewall rules and temporarily whitelist the agent's IP while the image was being pushed, and then reverting the changes to keep the firewall secure.

8.1.3 Geospatial database

The usage of a geospatial database has proven to be a more complex part of the project than expected. The initial architecture design was made with no sufficient knowledge on geospatial data and databases, and as a result it has been redesigned during development as a reaction to the found limitations. The three approaches that have been used are:

- **PostgreSQL with PostGIS**: The initial architecture used a single database, and PostgreSQL was the chosen one. Using the PostGIS extension, geospatial queries were performed to retrieve the data. However, the large amount of data that was being stored in the database significantly slowed down the performance of these queries. Also, as PostgreSQL is a relational database, we were not able to take advantage of unstructured data.

- **Cosmos DB**: The next idea was to use Cosmos DB to store the data, and keep PostGIS for just the geospatial information and operations. But when researching information on how to use cosmos, we found that it was compatible with geospatial data and queries. Although its capabilities are significantly more limited than the ones in PostGIS, it was estimated that they were sufficient for our use cases: *point*, *polygon* and *multi polygon* data types to represent our geospatial data, *ST_DISTANCE* and *ST_WITHIN* functions to make geospatial queries, and possibility of indexing geospatial data to ensure fast querying. As a result, the decision to use just cosmos was made, as it resulted in a simpler architecture.

  But after a while, some previously unknown limitations started to pop up. Cosmos DB has a document size limit of 2 MB, and also a polygon size limit of 4096 points. This meant that polygons of certain complexity could not be used. Various alternatives were discussed, such as simplifying the affected polygons or partitioning them into smaller ones and then combining their results. But they were all complex solutions that would slow down development time and produce a much more complex code.

- **PostgreSQL with PostGIS + Cosmos DB (current)**: The final chosen architecture makes use of the two databases. The idea is to combine the powerful geospatial capabilities of PostGIS (which are also

faster than in Cosmos DB) with the speed and ability to use unstructured data of cosmos. The geospatial information of the data will be stored in the PostgreSQL database, alongside a unique identifier referencing the cosmos document that contains the data. The reduction of data stored in PostgreSQL (and the usage of indexing) dramatically improves the speed of geospatial queries, and then retrieving the data from cosmos is almost instant.

8.1.4 Integration with ZIH

The *Zurich Information Hub* is a highly demanded product of the company, and although it is already usable it is in constant development and its integration with other products is still a complex process. There are currently other products being developed by the company that make use of the ZIH and have a higher priority than this project, and therefore the developers in charge of integrating the ZIH with other products are focused on other projects. Fortunately, the two components of this project can be developed and tested separately without the need of using the ZIH, and therefore this is not a development issue. It will only become a problem when moving from the development to the production stage, which is out of the scope of this thesis.

8.1.5 Reduction of developers

During the intermediate stages of the project, one of the products being developed by Zurich needed extra developers in order to meet the delivery deadline. As a consequence, the first week of October, the company relocated one of the developers working on this project to another team. Furthermore, three weeks later another developer left the company, so the team was reduced to a single developer. This continued for all of November, and it wasn't until the end of November that a new developer joined the team as a reinforcement. This means that the planned team size of three developers only existed during the first three weeks of the project, and for a full month all the development was done by a single developer. Furthermore, the onboarding of the new developer at the end of the project required extra meeting time to do a comprehensive onboarding.

The reduction in developers translated into a longer developing time for the project or a reduction of its size. The chosen solution was to reduce the amount of implemented ingestors for the data hub. A more detailed explanation can be found in section 8.3.1. Also, the impact of this issue on the cost of the project is detailed in sections 8.4.1 and 8.4.3.

8.2 Planning changes

During the development of the EDH, one of the business requirements changed. The classification of datasets had to be substituted with a geodata extraction process. The goal was to easily detect datasets that contained geospatial data in order to use them with ease in the GeoIntel product, and, for datasets that do not contain geospatial data, attempt to create a process capable of generating it based on the contents of the dataset. A more detailed explanation of this new component can be found in section 5.3.

This change in the requirements has been reflected by changing task **ED5** from **dataset classification** to **dataset geolocation**. This new task is similar to the initially designed process, as it also takes a dataset and tries to generate new information from its contents, and then load it into the ZIH alongside the original data. So the only change that will be necessary is the internal algorithm of the component, and the architecture of the EDH will remain untouched, thanks to its modular structure. This change of requirements occurred before starting the development of the dataset classification component, and therefore did not impact in any negative way the development, and no previously done work had to be discarded.

8.3 Task completion

For the tasks defined in section 2.1, table 10 shows their final status.

Also, the updated Gantt chart can be seen in figure 18, where the final timeline of the project can be viewed.

For tasks related to the product implementation, a more detailed explanation will be given. Tasks under the project management category will not be commented, as they are

8.3.1 ED. External data hub

- **ED1. Research available sources**: The research of sources was a simple but tedious task. Not only sources had to be listed, also the type of datasets they provided, the value these datasets can provide to the company and the requirements of each source to allow its ingestion. The task was considered completed after three weeks of its start, once a set of source types was selected. However, this list ended up being reduced to just two data source types.
- **ED2. Ingestor framework design**: The design of the framework

ID	Task Name	Status
PM	Project management	Completed
PM1	Defining the project's scope	Completed
PM2	Project planning	Completed
PM3	Budget and sustainability report	Completed
PM4	Final project documentation	Completed
PM5	Meetings	Completed
ED	External data hub	Simplified
ED1	Research available sources	Completed
ED2	Ingestor framework design	Completed
ED3	DevOps	Completed
ED4	Ingestors	Simplified
ED5	Dataset geolocation	Completed
ED6	Automation of ingestors	Completed
DA	GeoIntel	Completed
DA1	Geocoding	Completed
DA2	Auxiliary datasets	Completed
DA3	API development	Completed
DA4	Data viewing website	Completed

Source: Own creation.

Table 10: Task completion status.

was a process of first understanding the different services available in Azure and how they could be used, and then choosing how the ingestors would be implemented, where would the extracted data be stored and designing the architecture to connect all of these elements, keeping in mind at all times the goal of simplifying the ingestor addition process for future implementations. The task took one week more than expected and was finished on time.

- **ED3. DevOps**: The DevOps architecture has been properly implemented and has provided the software development process with a robust CI/CD infrastructure. This has impacted the overall development of the product, as tests and deployments have been completely automated from the beginning, allowing the team to focus on the development and bug fixing.

The usage of pipelines proved to be more difficult than expected for people not familiar with them, and therefore the task took a week longer than expected to be completed, but was finished before the deadline.

- **ED4. Ingestors**: This task's magnitude has been reduced in order to adapt to the reduction of developers. The decision made was to implement just two carefully chosen ingestors to provide a base functionality to the product that could be used by the company and serve as a good example for the future addition of new ingestors.

  The decision to implement the ingestors for the Contify API and CKAN servers was based on business use cases. The data provided by Contify is highly valuable for the company and was needed in the data science department. And the CKAN ingestor provides an ingestion function capable of obtaining data from an extremely wide variety of sources, and the internal structure of the extracted datasets is completely unspecified. This is ideal for testing other components of the product, such as the dataset geolocator, which is intended to work with any type of dataset. Simply put, the Contify ingestor provides the extraction of a very specific set of information from a single source that is highly valuable for the company, and the CKAN ingestor provides a generalized ingestion method that can be used for many data sources and can therefore ingest a huge amount of datasets for the company.

  As a bonus, the Contify ingestor uses a regular Azure Function, while the CKAN one uses a Durable Function, so they serve as examples for the two possible implementations for future addition of new ingestors.

  The work on this task started on time, and thanks to its reduction in complexity it was finished a week earlier than expected and three weeks before the initial deadline.

- **ED5. Dataset geolocation**: The component has been fully implemented and deployed. The implemented Azure Function is able to process a given dataset and extract an aggregation of all the geodata contained in it. If the dataset does not contain any geospatial data, the function tries to extract location entities from the contents of its files and generates a polygon based on them.

  The task was not initially planned, but based on the planned timeline for the task being replaced by it, it started with a two-week delay, which was caused by the redefinition of the task to adapt it to the business requirements and the reduction of developers in the team, as the focus was set on the ingestors. It also took an extra week to be developed than the expected for the original dataset classification task. The result is a completion one week later than the deadline for the original task.

- **ED6. Automation of ingestors** : The two implemented ingestors (CKAN and Contify) have been deployed and a daily time trigger executes them. Every morning, new datasets and updates on previously

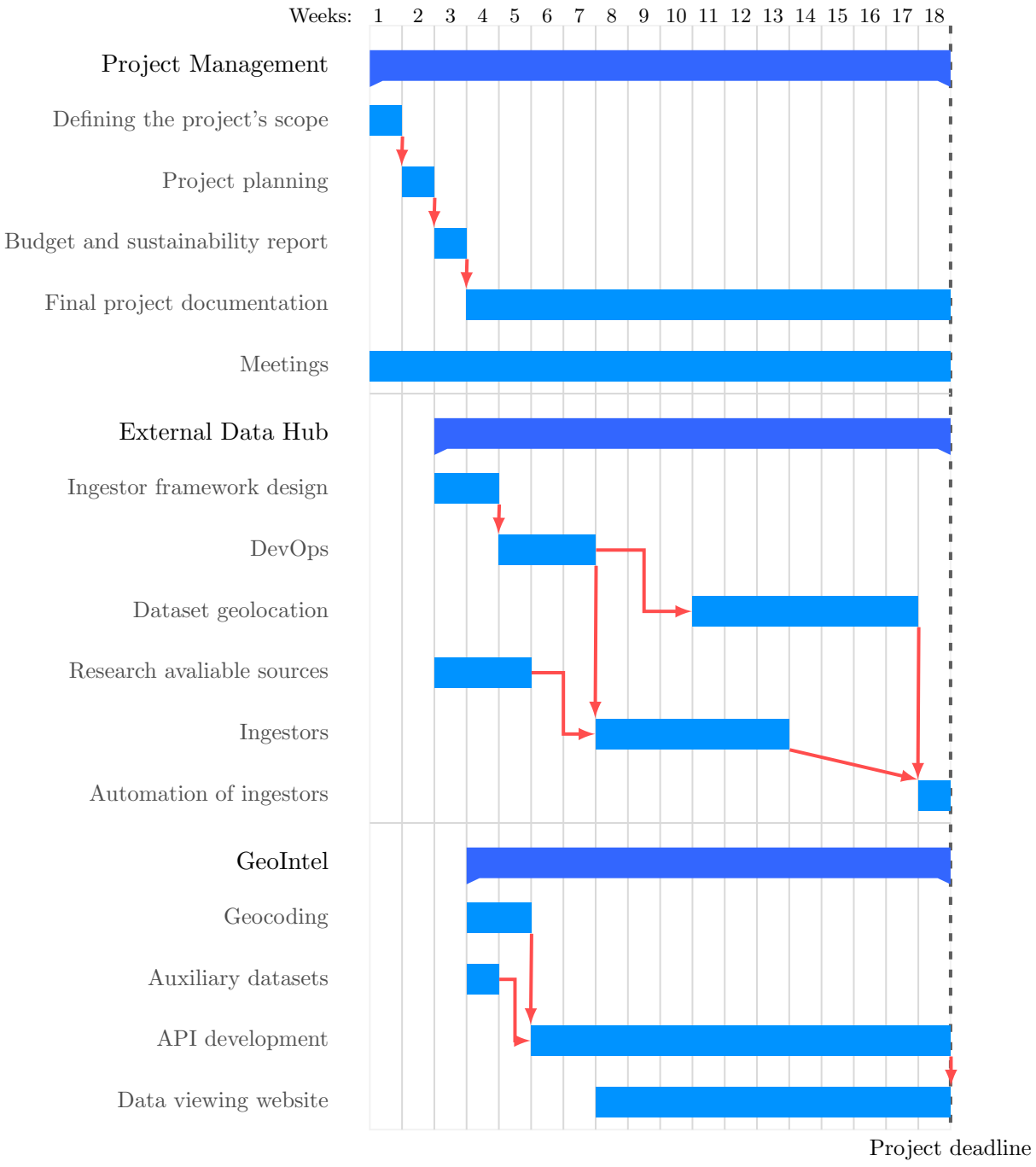
ingested data are available in the EDH landing storage (5.2). This task started a week later than expected due to a delay in the implementation of the dataset locator, but was completed on time.

8.3.2 DA. GeoIntel

- **DA1. Geocoding**: The geocoding functionality has been successfully implemented. Although it as taken a week more than planned to be implemented, the task has been completed before the deadline and with very satisfactory results.
- **DA2. Auxiliary datasets**: Auxiliary geospatial datasets have been easily found and uploaded to the geospatial database. This has been a straightforward and simple task and has taken the expected time to be completed.
- **DA3. API development**: The first release of the API is deployed. It has the needed endpoints needed by the viewing website, alongside with other endpoints that can be used by clients to access the data in a variety of ways, being able to retrieve filtered data and statistics. The API properly connects to the two available databases (geospatial PostgreSQL and Cosmos DB) and depending on the request being served it fetches information in either one of these databases or both if needed. Response times are fast, except for queries that return a big amount of data, which take longer to be downloaded by the requesting client.

  The development of the API has taken a longer than expected time to fully implement, with a 3-week difference with the planned time estimate. But as the deadline for this task had a 3-week margin from the estimated time, the task has been completed on time.
- **DA4. Data viewing website**: The website has the main features working, with a fully interactive map and search menu. Response times are fast, and the portal can be used not only as a graphical user interface for using the API, it also serves as a geodata viewing portal. It serves as a very good example of how the GeoIntel API can be used to implement geospatial applications that use data available in the ZIH. The task has finished on par with the API development, as expected. This is a week later than expected, but on schedule.

8.3.3 Final Gantt chart



Source: Own creation.

Figure 18: Final Gantt chart

8.4 Final cost

8.4.1 Human resources

The cost of human resources has seen an important deviation, as a consequence of the reduction in the amount of developers working on the project (issue 8.1.5). The total number of hours worked has been reduced, and most of this reduction comes from the simplification of task ED4. In table 11 are detailed the final worked hours by each data engineer on each task, alongside the final deviation in worked hours that tasks have suffered. The worked hours of the product manager and the IT manager are not listed because they have matched the planned ones.

Tasks	Hours worked by developer				Deviation
ID	D.E.1 20,9€/h	D.E.2 20,9€/h	D.E.4 20,9€/h	D.E.3 (me) 20,9€/h	Total hours
PM	21	29	35	160	-7
PM1				15	0
PM2	1	1		15	0
PM3				15	0
PM4				60	0
PM5	20	28	35	55	-7
ED	40	90	50	305	-115
ED1	20	20		20	0
ED2	20	20		20	0
ED3		50		70	0
ED4			20	80	-110
ED5			20	100	0
ED6			10	15	-5
DA	10	40	210	290	-20
DA1				20	0
DA2	10	10		10	0
DA3		30	80	160	-10
DA4			130	100	-10
Total	**71**	**159**	**295**	**755**	**-142**

Source: Own creation.

Table 11: Tasks final worked hours by data engineers

With the deviations in worked hours for each task, the deviation of their cost in human resources. As the deviations occurred within the same role, the deviations in the tasks costs can be calculated by multiplying the deviation in hours by the hourly wage of the data engineers. Table 12 shows the final cost for each task and the respective deviation. The total final cost, **31.828 €**, is the final personnel costs per activity (PCA), and the deviation (calculated with formula 1) is of **- 2.967,8 €**.

Tasks ID	Estimated cost (€)	Real cost (€)	Deviation (€)
PM	8.329,3	8.183	- 146,3
PM1	499,5	499,5	0
PM2	541,3	541,3	0
PM3	499,5	499,5	0
PM4	1.874	1.874	0
PM5	4.915	4.768,7	- 146,3
ED	14.135,5	11.732	- 2.403,5
ED1	1.254	1.254	0
ED2	2.097	2.097	0
ED3	2.842,5	2.842,5	0
ED4	4.389	2.090	- 2.299
ED5	2.926	2.926	0
ED6	627	522,5	- 104,5
DA	12.331	11.913	- 418
DA1	418	418	0
DA2	627	627	0
DA3	6.270	6.061	- 209
DA4	5.016	4.807	- 209
Total	**34.795,8**	**31.828**	**- 2.967,8**

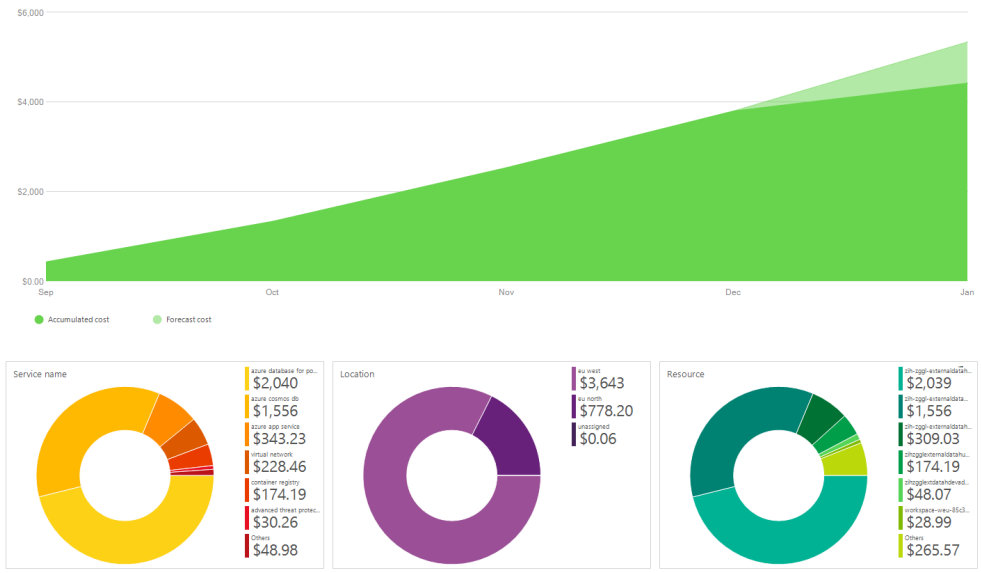
Source: Own creation.

Table 12: Tasks cost

8.4.2 Development costs

To obtain the development costs, the cost report generated by the Azure portal has been used. This tool simplifies the analysis of costs and provides a clear and interactive dashboard (image 19) where the user can review the costs of the used Azure services over a specified period of time.

Using this tool, the cost of each azure resource used over the development period of this thesis has been checked, and the average monthly costs can ve



Source: Azure portal.

Figure 19: Azure portal cost report

viewed in table 13, alongside the originally projected costs and the deviation between them. The values shown in the Azure portal are given in dollars, and when converting them to euros we obtain a total final cost of **3.819,22 €**, with a deviation that is calculated with formula 2 and amounts to **- 866,18 €**.

Service type	Estimated cost ($)	Real cost ($)	Deviation ($)
Storage Account	101,48	4,45	- 97,03
Azure Functions	0,00	0,5	+ 0,5
Azure PostgreSQL	634,12	509,4	- 124,72
Azure Cosmos DB	375,4	388,41	+ 13,01
Key Vault	0,03	0,01	- 0,02
Azure ML	198,56	0	- 198,56
App Service	13,14	85,74	+ 72,6
Azure Maps	0	0	0
Container Registry	31,6	43,5	+ 11,9
Virtual Network	-	57,11	+ 57,11
Threat protection	-	7,57	+ 7,57
Log analytics	-	7,27	+ 7,27
Total	**1.354,33**	**1.103,96**	**- 250,37**

Source: Azure Portal Cost analysis.

Table 13: Azure resources monthly costs

The biggest deviation can be seen in the Azure ML service, as it has not

been used. The other major deviations are seen in the Storage Account and PostgreSQL geospatial database. The reason for this is that the amount of ingested data has been less than expected, as a direct consequence of issue 8.1.5. Less ingestors means less ingested data. Lastly, the App service used for the data viewing webpage has had a big positive difference between the budgeted and real cost. This is probably due to an underestimation of the Azure Price Calculator, as the deployed web app consumes more resources than the average web app due to the large amounts of data passing through it and being displayed in the interactive map. Finally, a few extra security related resources that were not budgeted but have been put in place by the IT manager have increased the final development costs.

8.4.3 Other costs

The rest of the costs have not changed significantly during the development of the project, as it was expected. The only change that has occurred is a consequence of the reduction of hours worked by developers, explained in section 8.4.1, and has to do with the peripheral usage. In section 3.3 it was estimated that the cost of these peripherals is 0,4655 €for each worked hour. Using formula 3, the cost has gone down by **66,10 €**. The final cost is **683,35 €**.

Adding the development costs and other costs, the final generic costs (GC) end up being **4.502,57 €**.

8.4.4 Summary

In table 14 can be seen the comparison of the budget with the final cost.

Activity	Budget (€)	Cost	Deviation (€)	Deviation (%)
PCA	34.795,8	31.828	- 2.967,8	- 8,53
GC	5.434,85	4.502,57	- 932,28	- 17,15
Total costs	40.230,65	36.330,57	-3.900,08	- 9,69
Contingency	6.035	0	- 6.035	- 100
Incidents	3.850	0	- 3.850	- 100
TOTAL	50.115,65	36.330,57	-13.785,08	- 27,51

Source: Own creation.

Table 14: Final budget

Although at a first glance the deviations seem big, looking at the individual numbers things make more sense. First, the contingency and incidents budgets have not been used, as no incident has occurred that has forced to make use of

them. Therefore, if we remove these two items from the equation and look at the PCA and GC costs, the deviation is much lower (- 9,69 %). As explained in previous sections, the reduction of the development team has reduced both the costs of human resources and the development costs for some storage related Azure services. These, together with the discarding of the Azure ML service, account for the majority of the reduction of the costs.

As an overall view, the reduction in the project cost can be seen as a success, and not only for the obvious reason of costing less money than budgeted. The contingency and incidents budget items have not been needed, as no major issues have occurred during the project. Also, the only important issue faced has decreased cost proportionally to the decrease in implemented ingestors, all without having a major impact in the usability of the developed products. Therefore, it can be concluded that the deviations are not the result of an overestimation of the budget, but a consequence of the reduction of the team size.

8.5 Sustainability report

Examining the sustainability of this project is a very important task. Examining how every project done and their results its results will impact the world is a must, and the social, economic and environmental implications have to be analyzed. This also applies to this project. It is important to be conscious of how it will impact society, and the environmental implications it might have, as this is a shared responsibility for every member of society. This sustainability report examines the project's environmental, economic and social implications.

8.5.1 Environmental impact

The environmental impact of this project will come from energy usage and the carbon footprint of the used hardware components. As components are developed and deployed, their needed resources will be allocated, so they will only be consumed when needed. Also, the project will not only benefit from the environmental advantages of using any cloud platform (detailed in the next question), but also from choosing a service that is committed to reducing its environmental footprint [69].

There are many implementations of systems that do a similar task as the one that this project aims to implement. Although the specifics of how they are implemented (and therefore their environmental impact) is often not publicly available information, there are two main approaches to implement the project: creating an on-premise infrastructure by the company, or using a

cloud platform. The best approach from an environmental standpoint is using the cloud platform [70], as it ensures that the resources used are just the currently needed ones, and also ensures the reuse of hardware components, as they are shared among all cloud users. This is why the decision to use Azure's cloud service instead of setting up an on-premise infrastructure is a good decision from an environmental point of view.

Looking at the individual components being used, the tasks that do the majority of the processing are deployed in Azure Functions, which due to their serverless nature consume much less energy than regular served-based applications [71]. The other major energy-consuming components of this project are related to data storage. Storing data on the cloud consumes much more energy than storing in local hard drives [72], but this is the only possible approach for this product, as the files need to be available for all business units around the globe. Using storage solutions provided by cloud services is a better solution from an environmental standpoint than using a dedicated storage facility set up by the company, as, although the datacenters used by cloud providers are major carbon emitting facilities, they are designed to be as efficient as possible, and are often located near clean energy sources from where they obtain their energy. Furthermore, the ZIH is a centralized place intended to store all the company's data. This centralization avoids having duplicate data in different business units, so the total amount of data stored, and the emissions produced by it, are reduced to the possible minimum.

As a result, the project will end up reducing the carbon footprint of the company, as data will be unified in a single location, removing duplicates within the company and moving it from local storing sites to more efficient cloud data centers. It will be crucial to ensure that all business units move their data to the ZIH, and start to get more data through it, as if they fail to do so the environmental impact will increase.

8.5.2 Economic impact

Before starting the implementation of the project, an estimation of its costs, both human and material, have been analyzed and documented in section 3. Once the development has concluded, the final costs have also been studied and are listed in section 8.4. There, the details on the final costs and their deviations from the budget are analyzed.

As with the environmental implications, the usage of a cloud platform instead of an on-premise infrastructure results in economic benefits. If a cloud service was not used, all hardware components used for this project would need to be bought at full price, and an important part of them would not be used again once the product stopped being used. Using a cloud service ensures that

costs are strictly related to the usage of resources, and therefore provides for a significant cost reduction in the implementation and usage of the product. Also, once the project is completed, many tasks that were previously done manually will be automated, and therefore many human resources that are now being wasted on these tasks will now see their productivity increased and will be able to focus on higher value tasks. Therefore, the project will not translate in a cost reduction of human resources, but rather a boost in their productivity and profits of the company without increasing the spending. The usage of shared data among all business units will reduce the costs of obtaining this data, as the avoidance of duplicates will mean that costs of generating or extracting data are generated only once, and that licensing for paid data is paid only once and used by the whole company.

8.5.3 Social impact

As mentioned in section 1, there was an urgent need to develop this product, as it dramatically improves the knowledge of business units inside the company, reduces the amount of tasks that are now done manually and improves the productivity of their workers. Making data more accessible also opens up a wide range of opportunities to develop new products and improve business procedures in the future.

With the development of this product, users will be able to effectively find and analyze data that is currently being wasted or ignored. Customers of the company will see how the product will improve, having a better personalization of their insurances, offering better prices and better coverage for their specific needs. The only possible negative impact that the product can have on costumers is the increase in the price of the products they use, or the degradation of its quality, caused by the use of biased data that discriminates a certain user group. This is not a direct consequence of this particular product, but a possible scenario that commonly occurs when using data driven business decision taking. The upside is that the opposite effect can happen: the usage of data can remove human bias from the decisions taken and reduce inequality. Therefore, the company's data science department will have to do an exhaustive job in analyzing the data and models used to avoid using biased solutions.

For the people involved in its development, the project has made a big impact in our professional development, as we have used technologies that were not familiar to us, exploring solutions based on a cloud infrastructure and implementing information extraction methods for geodata that had to be designed and developed from scratch by the team. The project has been a very enriching experience overall.

8.6 Relationship with the degree

8.6.1 Justification of the specialty and related subjects

The implementation of this project builds on a lot of knowledge acquired from the computing specialty. Its main focus resides in the retrieval and processing of large quantities of data, alongside with the extraction of information from it. The knowledge and experience gained from the *Data Mining (MD)*, *Machine Learning (APA)* and specially *Massive Information Search and Analysis (CAIM)* subjects have been very helpful to develop this thesis. Also, concepts learned in *Programming Languages (LP)* and *Algorithmics (A)* have been helpful to develop efficient algorithms and complex data structures.

But not only knowledge from the computing specialization has been helpful. Both relational and non-relational databases have been used for the project, for which experience from *Databases (BD)* has been useful. And thanks to the *Parallelism (PAR)* subject, parallel computing has been successfully used in the scenarios where it could be applied to speed up processing time.

8.6.2 Technical skills

- **CCO 1.3 (in depth):** An in depth analysis of possible solutions using Azure Services has been done for the project. A robust architecture has been designed using the most appropriate resources, such as databases with different paradigms and capabilities, serverless computation and other components, where these components interact with each other inside a highly modular design.
- **CCO 2.1 (enough):** A fully automatized end to end system has been developed for the automatic ingestion, processing and visualization of data, removing any need for human interaction during the process.
- **CCO 2.2 (slightly):** The system properly analyzes the information and automatically generates or extracts geospatial information to categorize its contents in a location-based visual representation.
- **CCO 2.3 (slightly):** The data viewing website serves as an environment where data containing geospatial information can be viewed and interacted with.
- **CCO 2.4 (enough):** One of the implemented components of this project has been a function capable of extracting geospatial information from datasets, or generating it with the use of NER if no explicit geospatial information exists in the data.

- **CCO 2.5 (in depth):** An application in charge of the massive ingestion of open data has been designed and developed.
- **CCO 3.2 (slightly):** The programmed behavior of components deployed as different types of services, each one having specific running environments and capabilities, has been done keeping in mind the hardware behind them to obtain efficiently executed algorithms.

9 Conclusions

The project has been a success. A fully working end to end data processing pipeline has been deployed, where open data sources external to the company are ingested by the External Data Hub 5, saved to a landing storage and then processed by the Dataset Locator 5.3 to extract geospatial information from them. If no geodata is found within the dataset, the function is able to generate it using NER and generating a geometry, with a success rate of 78,5 % 5.3.3. This data is then sent to the Zurich Information Hub, where it is transformed and published for availability of use for Zurich employees and applications. One of these applications has been fully implemented. It serves as a geodata serving app, implemented both with an API that can be used by technically skilled users to build new applications on top of it, and an interactive portal where users can view, filter and download the data in a responsive and intuitive environment.

All the development process has been backed by a good DevOps architecture that automatically tests and deploys new features using a CI/CD infrastructure, allowing developers to focus on the development and removing human error from the process. The whole system is built with a modular architecture that allows for a very easy extension of the system: new ingestors can be easily added, the components are decoupled and easily modifiable and all of them can be connected to new applications. All of this in a robust and secure infrastructure.

Business requirements have been met, and the system is fully usable. The only major issue that has affected the project 8.1.5 has not impacted the final result in a meaningful way, thanks to a good planning and execution. It also has not negatively impacted the cost of the project 8.4.

10 Nomenclature

- **Continuous integration and continuous deployment (CI/CD):** Automatic testing and deployment of developed software elements as they are implemented. An automated pipeline is in charge of testing the developed code and deploying it, removing the need for manual interaction with the process.
- **Coordinate reference system (CRS):** Coordinate based system used to locate points and other geographical features in a geographic space. In this thesis, this space is Earth's surface.
- **External Data Hub (EDH):** Product developed in this thesis for Zurich, that aims to create a platform for open data ingestion.
- **Geocoding:** Process of converting a free text address to its corresponding coordinates.
- **Geodata:** Data containing geospatial information. Also referred to as geospatial data.
- **GeoJSON:** Open standard format based on the JSON format for representing geographical features along with non-spatial attributes.
- **Hypertext Transfer Protocol (HTTP):** Application-layer protocol for transmitting hypermedia documents. Used mainly for communication between web browsers and web servers.
- **Ingestor:** Component of the External Data Hub that extracts data from an open data source to a storage owned by the company.
- **JavaScript Object Notation (JSON):** Open standard format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays.
- **Named entity recognition (NER):** Subfield of natural language processing and information extraction that aims to recognize entities in free text. Entities can be locations, famous people, historic events, etc.
- **Natural language processing (NLP):** Field in artificial intelligence that aims to create tools, such as language models, that can be used by computers to analyze and use text as humans do.
- **Pipeline:** Automatically triggered and programmable process used to test code, check its quality and interact with cloud resources.
- **Zurich Information Hub (ZIH):** Product of Zurich designed to centralize all data that is possessed by the company, used to process, anonymize and transform ingested data.

References

[1] Andrea Murphy et al. *The Global 2000 2021*. May 2021. URL: https://www.forbes.com/lists/global2000.

[2] *A global insurer — Zurich Insurance*. URL: https://www.zurich.com/en/about-us/a-global-insurer.

[3] *What Is Cloud Computing? A Beginner's Guide — Microsoft Azure*. URL: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

[4] *What is DevOps? — Atlassian*. URL: https://www.atlassian.com/devops.

[5] *What Is a Data Warehouse — Oracle*. URL: https://www.oracle.com/database/what-is-a-data-warehouse/.

[6] *World Geodetic System 1984 (WGS84) - Qinsy*. URL: https://confluence.qps.nl/qinsy/latest/en/world-geodetic-system-1984-wgs84-182618391.html.

[7] *WGS 84 - WGS84 - World Geodetic System 1984, used in GPS - EPSG:4326*. URL: https://epsg.io/4326.

[8] *WGS 84 / Pseudo-Mercator - Spherical Mercator, Google Maps, OpenStreetMap, Bing, ArcGIS, ESRI - EPSG:3857*. URL: https://epsg.io/3857.

[9] *ServiZurich — Home*. URL: https://bcntdc.zurich.com/en/about.

[10] *Kaggle: Your Machine Learning and Data Science Community*. URL: https://www.kaggle.com/.

[11] *CKAN - The open source data management system*. URL: https://ckan.org/.

[12] *Cloud Computing Services — Microsoft Azure*. URL: https://azure.microsoft.com/en-us/.

[13] *Azure DevOps Services — Microsoft Azure*. URL: https://azure.microsoft.com/en-us/services/devops/.

[14] Isaac Sacolick. *What is CI/CD? Continuous integration and continuous delivery explained — InfoWorld*. URL: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html.

[15] *CI/CD for Azure Web Apps - Azure Solution Ideas — Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/azure-devops-continuous-integration-and-continuous-deployment-for-azure-web-apps.

[16] *Azure Repos – Git Repositories — Microsoft Azure*. URL: https://azure.microsoft.com/en-us/services/devops/repos/.

[17] *Azure Pipelines — Microsoft Azure*. URL: https://azure.microsoft.com/en-us/services/devops/pipelines/.

[18] *Azure Boards — Microsoft Azure*. URL: https://azure.microsoft.com/en-us/services/devops/boards/.

[19] Chuck Gehman. *What Is Version Control? Git Version Control? — Perforce*. URL: https://www.perforce.com/blog/vcs/what-is-version-control.

[20] *What is Agile? — Atlassian*. URL: https://www.atlassian.com/agile.

[21] *What is Scrum?* URL: https://www.scrum.org/resources/what-is-scrum.

[22] Colm Kenny. *What Is Web Scraping And How Does It Work? — Zyte.com*. URL: https://www.zyte.com/learn/what-is-web-scraping/.

[23] *Pylint - code analysis for Python — www.pylint.org*. URL: https://pylint.org/.

[24] Thomas Hamilton. *Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE*. Sept. 2021. URL: https://www.guru99.com/unit-testing-guide.html.

[25] *Empowering App Development for Developers — Docker*. URL: https://www.docker.com/.

[26] *Azure Container Registry documentation — Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/container-registry/.

[27] *What is Machine Learning? — IBM*. URL: https://www.ibm.com/cloud/learn/machine-learning.

[28] *Classification In Machine Learning — Classification Algorithms — Edureka*. URL: https://www.edureka.co/blog/classification-in-machine-learning/.

[29] *What is Natural Language Processing? — IBM*. URL: https://www.ibm.com/cloud/learn/natural-language-processing.

[30] *What is geocoding?—ArcMap — Documentation*. URL: https://desktop.arcgis.com/en/arcmap/latest/manage-data/geocoding/what-is-geocoding.htm.

[31] *Durable Functions Overview - Azure — Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp.

[32] *PostGIS — Spatial and Geographic Objects for PostgreSQL*. URL: https://postgis.net/.

[33] *NoSQL Databases – What They Are and Why You Need One*. URL: https://www.couchbase.com/resources/why-nosql.

[34] *Postman API Platform — Sign Up for Free*. URL: https://www.postman.com/.

[35] *API Documentation & Design Tools for Teams — Swagger*. URL: https://swagger.io/.

[36] *JSON*. URL: https://www.json.org/json-en.html.

[37] *YAML Tutorial: Everything You Need to Get Started in Minutes — Cloudbees Blog*. URL: https://www.cloudbees.com/blog/yaml-tutorial-everything-you-need-get-started.

[38] *Earning Power. Project Management Salary Survey Eleventh Edition*. Project Management Institute, 2020, pp. 251–257. URL: https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/pmi-salary-survey-11th-edition-report.pdf?v=b8dbbec0-0048-4b6d-99ca-389b36cad726.

[39] *Salario de un IT manager en España*. URL: https://es.indeed.com/career/it-manager/salaries.

[40] *Salario de un Data engineer en España*. URL: https://es.indeed.com/career/data-engineer/salaries.

[41] *Calculadora de precios — Microsoft Azure*. URL: https://azure.microsoft.com/es-es/pricing/calculator/.

[42] *DELL Latitude 5401 Specs, Reviews & Prices — Techlitic.com*. URL: https://techlitic.com/laptops/dell-latitude-5401/.

[43] *Sillas de Escritorio - Compra Online - IKEA*. URL: https://www.ikea.com/es/es/cat/sillas-escritorio-20652/.

[44] *Escritorios y Mesas de Ordenador - Compra Online - IKEA*. URL: https://www.ikea.com/es/es/cat/escritorios-hogar-20651/.

[45] *Esto es lo que pagas de luz de más por trabajar desde casa*. URL: https://www.eleconomista.es/economia/noticias/11285906/06/21/Este-es-el-coste-de-la-electricidad-por-trabajar-desde-casa.html.

[46] *Welcome to Flask — Flask Documentation (2.0.x)*. URL: https://flask.palletsprojects.com/en/2.0.x/.

[47] *Welcome - Humanitarian Data Exchange*. URL: https://data.humdata.org/.

[48] *Fan-out/fan-in scenarios in Durable Functions - Azure — Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-cloud-backup?tabs=python.

[49] *News API - Custom News Feed API for Business News — Contify*. URL: https://www.contify.com/news-feed-api/.

[50] *Folium — Folium 0.12.1 documentation*. URL: https://python-visualization.github.io/folium/.

[51] *Leaflet - a JavaScript library for interactive maps*. URL: https://leafletjs.com/.

[52] *GeoJSON*. URL: https://geojson.org/.

[53] *Keyhole Markup Language — Google Developers*. URL: https://developers.google.com/kml.

[54] *Shapefiles—ArcGIS Online Help — Documentation*. URL: https://doc.arcgis.com/en/arcgis-online/reference/shapefiles.htm.

[55] *GeoPandas documentation*. URL: https://geopandas.org/en/stable/.

[56] *Pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/.

[57] *How to Dissolve Polygons Using Geopandas: GIS in Python — Earth Data Science - Earth Lab*. URL: https://www.earthdatascience.org/workshops/gis-open-source-python/dissolve-polygons-in-python-geopandas-shapely/.

[58] *What is named entity recognition (NER) and how can I use it? — by Christopher Marshall — super.AI — Medium*. URL: https://medium.com/mysuperai/what-is-named-entity-recognition-ner-and-how-can-i-use-it-2b68cf6f545d.

[59] *Textract — textract 1.6.1 documentation*. URL: https://textract.readthedocs.io/en/stable/.

[60] *spaCy · Industrial-strength Natural Language Processing in Python*. URL: https://spacy.io/.

[61] *Multi-language · spaCy Models Documentation*. URL: https://spacy.io/models/xx#xx_ent_wiki_sm.

[62] *OPTICS algorithm - Wikipedia*. URL: https://en.wikipedia.org/wiki/OPTICS_algorithm.

[63] *sklearn.cluster.OPTICS — scikit-learn 1.0.2 documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html.

[64] *The Shapely User Manual — Shapely 1.8.0 documentation*. URL: https://shapely.readthedocs.io/en/stable/manual.html.

[65] *Buffer (Analysis)—ArcGIS Pro — Documentation*. URL: https://pro.arcgis.com/en/pro-app/2.8/tool-reference/analysis/buffer.htm.

[66] *Haversine formula - Calculate geographic distance on earth*. URL: https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/.

[67] *PostgreSQL: Documentation: 9.1: Index Types*. URL: https://www.postgresql.org/docs/9.1/indexes-types.html.

[68] *15. Spatial Indexing — Introduction to PostGIS*. URL: http://postgis.net/workshops/postgis-intro/indexing.html.

[69] *Azure Sustainability—Sustainable Technologies — Microsoft Azure*. URL: https://azure.microsoft.com/en-us/global-infrastructure/sustainability/#environmental-impact.

[70] *The carbon benefits of cloud computing*. Microsoft, WSP, 2020. URL: https://www.microsoft.com/en-us/download/confirmation.aspx?id=56950.

[71] *Adopting Azure serverless architectures to help reduce CO2 emissions – Part 1 - Sustainable Software*. URL: https://devblogs.microsoft.com/sustainable-software/adopting-azure-serverless-architectures-to-help-reduce-co2-emissions-part-1/?WT.mc_id=green-8664-cxa.

[72] *Carbon and the Cloud. Hard facts about data storage. — by Stanford Magazine — Stanford Magazine — Medium*. URL: https://medium.com/stanford-magazine/carbon-and-the-cloud-d6f481b79dfe.