



ESTGOH

**Escola Superior de Tecnologia e Gestão
de Oliveira do Hospital**

Instituto Politécnico de Coimbra

Study and Development of Cross-Platform Cloogy Mobile Application for VPS – Virtual Power Solutions.

**Relatório apresentado com vista à obtenção do grau de Mestre no âmbito da
realização do ciclo de estudos de Mestrado em Informática Aplicada**

Autor:

Atidivya Kumar Patra

Orientador:

Luis Veloso

Coorientador:

Francisco Afonso

Supervisor:

Rita Carreira
Andre Oliveira

Data: October 2017

Acknowledgments

Firstly, I would like to thank my supervisors from the university, Professor Luis Veloso and Professor Francisco Afonso. Our discussions, their critical remarks, and, of course their overall support and encouragements helped me to complete my work on time.

Next, my gratitude goes to Rita Carreira and Andre Oliveira, my supervisors from Virtual Power Solutions, for their help and support. I sincerely thank for the time and effort Andre and Rita have put in during my internship period. Special thank you to Pedro Marques for providing small and vital inputs into my development, Mario Pereira for helping me out to deploy my prototype into Apple devices, and, David Fernandes, for helping me out throughout my internship period. I would also like to thank the other colleagues from VPS for making the time spent together very enjoyable, and I wish them all good luck with their own careers.

From Instituto Pedro Nunes, I would like to thank my new friends Celia Pedro, Mariana Coelho, Mariana Valenca, Ana Almeida and, Joel Jegundo for making my lunch breaks memorable.

I would like to thank my parents for supporting me throughout my stay in Portugal, for believing in me, and providing me with the means to study, even though it took a bit longer than anticipated.

And last, but certainly not least, I would like to thank my fiancé, Lamha Bhatnagar, for being there for me, listening to my endless, over detailed explanations of my problems and for eagerly waiting for me to come back to India.

September 2017, Coimbra

Atidivya Kumar Patra

Resumo

A energia renovável e a conservação de energia tornaram-se tópicos importantes nos últimos anos. As empresas têm realizado esforços para reduzir o consumo de energia através da otimização de dispositivos e da conscientização dos consumidores sobre o seu uso.

Para contribuir com este esforço, a Virtual Power Solutions (VPS) fornece uma solução onde os proprietários / utilizadores de edifícios obtêm visibilidade e controle em tempo real dos seus aparelhos elétricos instalados na sua residência. A VPS alcançou com sucesso a gestão de procura, e a tecnologia de automação de edifícios numa única aplicação móvel designada por Cloogy. Esta aplicação fornece aos consumidores de energia e aos seus parceiros a capacidade de verificar e controlar o consumo de energia em tempo real, permitindo reduzir o nível de consumo ao mínimo sem comprometer as operações do dia a dia.

Atualmente, a Cloogy tem suas aplicações móveis disponíveis para Android, iOS e Windows Phone com funcionalidades semelhantes. Deste modo, porem cada aplicação requer diferentes linguagens de programação para cada plataforma, o que envolve um custo para manter essas diferentes plataformas. Por esta razão, para a presente tese, a VPS apresentou o objetivo de desenvolver uma aplicação móvel híbrida, que se baseará numa base de código único e terá acesso a todas as APIs da plataforma.

Diferentes tipos de ferramentas de desenvolvimento estão disponíveis para construir uma aplicação híbrida. Depois de definir os requisitos funcionais e não-funcionais, um protótipo de aplicação híbrida foi construído usando o Ionic Framework, que consiste numa das Frameworks de código aberto os disponíveis para construir aplicações móveis híbridas. Com a ajuda desta framework, uma aplicação móvel pode ser criada usando um conjunto de tecnologias da web, como JavaScript, HTML e CSS, e implementada o aplicativo em todas as principais plataformas, como Android e iOS.

O protótipo construído nos permite-nos aceder a dados de consumo através do nosso smartphone ou tablet a partir de uma localização remota com a ajuda da iEnergy3 API da VPS. As principais características oferecidas pelo protótipo são a monitorização do consumo de energia através de registos e dados em tempo real, e a verificação dos indicadores de consumo como desempenho, média diária, previsões, etc. O protótipo também fornece pegadas ecológicas, conjuntamente com indicadores de consumo, e é capaz de controlar e agendar períodos de consumo de eletricidade a partir de um local remoto.

Palavras-chave

Android, API, Cloogy, CSS, Energia, HTML, Híbrido, iOS.

Abstract

Renewable energy and energy conservation have become important topics during the recent few years. Efforts have been made by enterprises to reduce energy consumption by optimizing devices and making consumer aware of their usage.

To contribute to this effort, Virtual Power Solutions (VPS) provides a solution where building owners/users get a real-time visibility and control over their electrical appliances installed in the residential house. VPS has successfully achieved bringing demand management and building automation technology together into a single mobile application named Cloogy. This application provides energy consumers and their partners with an ability to see and control their energy consumption in real-time by allowing them to reduce the consumption level to the minimum without compromising day to day operations.

Currently, Cloogy has its mobile applications available for Android, iOS and Windows Phone with similar functionalities. This way, each application requires different independent platform programming languages which involves a cost to maintain these different platforms. Therefore, in this thesis, VPS has come up with a goal to develop a hybrid mobile application which will be based on the single code base and will have access to all platform APIs.

Different types of development tools are available to build a hybrid application. After going through all functional and non-functional requirements, a hybrid application prototype was build using Ionic Frameworks, which is one of the open source frameworks to build hybrid mobile applications. With the help of this framework, mobile applications can be build using one set of web technologies like JavaScript, HTML, and CSS and deployed across all major platforms like Android and iOS.

The prototype built allows us to access consumption data through our smartphone or tablet, from a remote location with the help of iEnergy3 API from VPS. The main features offered by the prototype are monitoring energy consumption through logs and the real-time data, check consumption indicators like performance, daily average, forecasts, etc. The prototype also provides ecological footprints along with consumption indicators and it is capable of controlling and scheduling periods of electricity consumption from a remote location.

Keywords

Android, API, Cloogy, CSS, Energy, HTML, Hybrid, iOS.

Index

1.	Introduction	1
1.1.	Motivation	1
1.2.	Problem Statement	2
1.3.	Objectives.....	3
1.4.	Approach	3
1.5.	Structure of the Report.....	3
2.	State of the Art	5
2.1.	Cloogy	5
2.2.1.	Energy Consumption Management.....	5
2.2.2.	Installation and Operation	6
2.2.	Types of Mobile Applications.....	7
2.2.1.	Native Applications.....	8
2.2.2.	Mobile Web Applications	10
2.2.3.	Hybrid Applications	12
2.3.	Cross-Platform Development Tools.....	13
2.3.1.	The Criteria for Development Tool Selection.....	14
2.3.2.	Overview of Cross-Platform Development Tools.....	15
2.3.3.	The Selection of Ionic with Apache Cordova	18
2.4.	Summary	22
3.	Planning	23
3.1.	Initial Plan	23
3.2.	Final Plan	24
4.	Objectives and Methodologies	25
4.1.	Functional Requirements.....	25
4.2.	Use Cases	26
4.2.1.	Dashboard	27
4.2.2.	Electricity	28
4.2.3.	Plugs.....	29
4.2.4.	Settings.....	30
4.3.	Non-Functional Requirements	30
4.4.	RESTful Web Services.....	31
4.5.	Overview of iEnergy3 Platform	32
4.5.1.	iEnergy3 API.....	33
4.5.2.	iEnergy3 API Operations	34
4.6.	The Development Stack	36
4.6.1.	AngularJS.....	36
4.6.2.	Apache Cordova / PhoneGap	37
4.6.3.	Ionic Framework	38
5.	Developed Work	41
5.1.	Project Folder Structure	41
5.2.	Front-End	42
5.2.1.	Application Structure	43

5.2.2. Routes.....	48
5.2.3. JavaScript Libraries.....	54
5.2.4. AngularJS Filters.....	57
5.3. User Interfaces.....	60
6. Conclusions	65
6.1. Results	65
6.2. Strengths.....	65
6.3. Limitations	67
6.4. Future Work	68
Bibliography	69
Appendix A: Developers Manual.....	73
Appendix B: User Manual.....	75

Figures List

Figure 2-1 Features of Cloogy Application (from [4]).....	6
Figure 2-3 The Cloogy Functional Architecture (from [4]).....	7
Figure 2-4 Comparison between native, mobile websites, and hybrid application architectures (from [5]).....	7
Figure 2-5 Horizontal swiping on Financial Times`s web application (from [8])	10
Figure 2-6 Global Mobile OS Market share as of 2016 (from [11])......	14
Figure 2-7 Size break of a hello world application developed using Xamarin (adapted from [22]).....	19
Figure 2-8 Comparison Poll Results between Ionic, PhoneGap & Xamarin as of October 2017 (from [26])......	21
Figure 3-1 Initial Project Plan in a Gantt Chart for Cross Platform Cloogy Application.	24
Figure 4-1 Use Case Diagram for Cross-Platform Cloogy Application Prototype	26
Figure 4-2 iEnergy3 Platform Architecture (from [31])	32
Figure 4-3 Command response in POSTMAN to get the device resource by providing the device ID	34
Figure 4-4 Snapshot from POSTMAN where request headers has been provided along with the requests.....	35
Figure 4-5 Snapshot of request body provided while creating a session with an API	35
Figure 4-6 Response Body Snapshot from POSTMAN after successful creation of session with iEnergy3 API.....	36
Figure 4-7 AngularJS MVC design pattern (from [33]).....	37
Figure 4-8 Card view code snippet from Ionic Documentation (from [24]).....	38
Figure 4-9 Ionic Developer`s Community at a glance (from [24])	39
Figure 5-1 Default Project Structure created by Ionic	41
Figure 5-2 Project Folder structure of the front-end of the application for VPS.....	43
Figure 5-3 Snapshot of the index.html file which includes all JavaScript files and other plugin	44
Figure 5-4 outside.html represents the outside GUI of the application.....	44
Figure 5-5 login.html file where the login form is designed for the application.....	45
Figure 5-6 inside.html file presenting the tab view of the application.	45
Figure 5-7 Implementation of delegate methods in JavaScript file of plugs controller.	47
Figure 5-8 Implementation of ion-slide-box in the views of the application	47
Figure 5-9 config method where \$stateProvider and \$urlRouterProvider are injected as dependencies.	49
Figure 5-10 Ready-made tabs template from Ionic Framework (adapted from [24])	50
Figure 5-11 Route for electricity screen.....	51
Figure 5-12 StateParameters defined for the consumption module screen.	52
Figure 5-13 Usage of parameters as links while in a view of an application	53
Figure 5-14 Consumption view of water cooler where we can see value passed as a parameter along with URL.....	53
Figure 5-15 Live data transmitted from message server to the front-end (from [39]).	54
Figure 5-16 Dashboard Module from android and iOS respectively.	55
Figure 5-17 Consumption module of one of the plugs where chart.js has been implemented....	56
Figure 5-18 Consumption module of electricity module where it has been compared with historic data	56
Figure 5-19 Configuration method for ionic time and date picker.....	57
Figure 5-20 Create, view, edit and delete operations snippet from schedule module of the application.....	58
Figure 5-21 Calendar week bitmap plotting in schedule module.....	58
Figure 5-22 Usage of Angular Filter in schedule module of the application.	59
Figure 5-23 Custom application filter for week bit map	59
Figure 5-24 Decimal to binary conversion.....	60
Figure 5-25 Custom filter code for the week bit map using angular.filter().	60

Figure 5-26 Snapshot of the application prototype asking for change API address in both OS (left: Android, right: iOS).	61
Figure 5-27 Dashboard module from the application prototype (left: Android, right: iOS).	62
Figure 5-28 Electricity module from the application prototype (left: Android, right: iOS).....	62
Figure 5-29 Code snippet of implementation of flex container property of CSS3.....	63
Figure 5-30 Different functionalities snapshot of plugs module from iOS.....	64
Figure 5-31 Snapshot from iOS where Consumption module for electricity and plugs is presented respectively.	64

Tables List

Table 2-1 Comparison Table: PhoneGap, Titanium, and Xamarin (from [14]).....	18
Table 4-1Use Case for Dashboard Module	27
Table 4-2Use Case for Electricity Module.....	28
Table 4-3 Use Case for Plugs Module	29
Table 4-4 Possible Response Codes for a specific request to iEnergy3 API (from [31])	33
Table 5-1 Delegate Methods for ion-slide-box from Ionic Framework (from [36]).	46
Table 5-2 Attributes from ion-slide-box directive by Ionic Framework (from [36])......	48
Table 5-3 Nested states brief description in Angular Routing	51

Acronyms List

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CDN	Content Delivery Network
CLI	Command Line Interface
CORS	Cross-Origin Resource Sharing
CSS	Cascading Style Sheet
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JS	JavaScript
JSON	JavaScript Object Notations
JWT	JSON Web Token
kWh	Kilowatt hours
OS	Operating System
PC	Personal Computer
REST	Representational State Transfer
SDK	Software Development Kit
SVN	Subversion
UI	User Interface
URL	Uniform Resource Locator
VPS	Virtual Power Solutions
XML	Extensible Markup Language

1. Introduction

This chapter introduces the motivation behind the research that has been carried out, along with the structure of this thesis. It identifies the importance of having a cross-platform application prototype for VPS. The prototype will be based on single code-base and will have access to all platform APIs.

This chapter is further structured as follows: Section 1.1 presents the motivation of this work, Section 1.2 describes the state of the problem, Section 1.3 states the objectives of this thesis, Section 1.4 presents the approach adopted in the development and Section 1.5 outlines the structure of thesis by presenting an overview of the chapters.

1.1. Motivation

Cloogy mobile application is available on Android, iOS and Windows phone. These operating systems have their own SDK to create mobile applications. The OS vendor has their own preferred programming language that supports the development of native applications. For example, iOS application development requires Objective C and Swift which means the developer should have good knowledge of this preferred programming language for the development, whereas Java is the preferred programming language for Android mobile application development. Therefore, we can say that the applications created using Java or Objective C programming language using the official SDK from the vendor can be called native applications [1].

The main concern for an enterprise is to have a development team which is proficient in all preferred programming languages, to maintain their mobile applications across all platforms. This kind of multi-platform oriented team will involve higher costs for the VPS.

The customers of Cloogy product will have different types of mobile devices. Therefore, it is imperative for an enterprise like VPS to have a mobile application that would work across all platforms. As mentioned earlier in this section, we should also understand it involves higher costs to maintain an application across all platforms. Therefore, an ideal solution to lower the cost would be a cross-platform mobile application which will work on multiple operating systems with a single code-base.

This thesis is an effort to evaluate the possibility of having a cross-platform mobile application for VPS which will work across all platforms at a lower cost. This thesis will share all the experience and results from the development of cross-platform application prototype. The main objective would be to first work on the prototype and then, later on, the VPS management will be responsible for the final decision to develop the complete application based the experience and the result of this work.

1.2. Problem Statement

The specific research problem of this thesis is to examine if VPS can have a cross-platform mobile application for Cloogy which will serve a better option than having native application for Cloogy across all platforms. In other words, the possibility of replacing the native applications in Android, iOS and Windows Phone with a cross-platform mobile application for Cloogy. A few different variations of mobile applications exist which will be discussed in detail in section 2.2. There are different methods to develop mobile applications, and have different frameworks to develop mobile applications. Each type of application is believed to be better or less suited to solve the problem. In mobile application development, there are many frameworks available to develop mobile applications. Enterprises that need a mobile application, have three available options that are native application development, mobile web development or hybrid application development [2].

Native applications are built using the device's native programming language and they only run on their designated platforms. For example, if Android applications cannot run on iOS platforms and vice versa. Mobile web applications are web applications developed to mimic the native applications of the hosting operating systems by executing in a web browser on the host platform [3]. Mobile web applications run in the device's browsers and can operate across all platforms. For example, mobile web application works well on platforms like Android platform, iOS platform, Blackberry platform, and Windows Phone platform. When compared with native applications, the main difference is that the mobile web applications are not installed in the device itself [2]. Hybrid mobile applications are designed and developed using either web technologies like HTML5, CSS, JavaScript APIs, that run inside a native application container or they have a specific programming language based on the selected framework. For example, Xamarin, a hybrid application development framework requires C# as the programming language for the development. The key difference between hybrid applications from mobile web applications and native applications, is that they are developed using standard web technologies, but they have access to the native device APIs and hardware. Some of the well-known hybrid mobile frameworks are PhoneGap, Ionic, Appcelerator, Xamarin and Appresso [3]. Hybrid applications bridges the gap between different mobile applications like native applications, they can be uploaded in the application store, can have their own unique application icon, and can take advantages of the mobile device features like GPS, accelerometers, compass, list of contacts, battery icon and so on. Whereas web applications, they rely on the HTML being rendered in the browser but do not have access to the device features. With the hybrid application, the problem is solved as the application is composed in a native frame and HTML/JavaScript is executed in *WebView* creating a hybrid application of both native and mobile web applications [1] [2] [3].

1.3. Objectives

Based on the motivation and problem statement above, the objectives of this thesis can be defined as:

- Study the state of art of cross-platform mobile development frameworks.
- Overview of the VPS's data server API which is iEnergy3 API.
- Research about cross-platform mobile development frameworks available.
- Select a framework to develop a hybrid application for VPS based on the requirement of the existing native features.
- Development and deployment of the application prototype to emulators and real devices for iOS and Android.

1.4. Approach

To conquer our objectives, the following approach was taken:

- Investigate frameworks for cross-platform mobile development.
- Define the user and application requirements, based on the research and identified enterprise's goals, and derive the architectural elements from those requirements.
- Integrate the architectural elements into an application design that supports the desired functionality.
- Design and implement a proof of concept mobile application supported by the framework.

1.5. Structure of the Report

The structure of this thesis reflects the order in which these issues have been dealt with throughout the research and development process. This thesis is structured as follows:

- Chapter 2 gives the background of the research starting from introducing the Cloogy application, types of mobile applications in the market and at the end discussing about the development tools along with the selection of the tool.
- Chapter 3 presents the initial and final plan for the thesis with the help of Agile methodology for mobile applications.
- Chapter 4 presents the requirements and non-functional requirements for the development of the mobile application by following the principles of agile methodologies. It also introduces some concepts behind RESTful web services, iEnergy3 and its API operations. At the end of this chapter, the approaches of different programming languages are discussed as these languages will be utilized during the development phase.
- Chapter 5 represents the work, describing what kind of development has taken place during internship period. Moreover, a detailed overview of the work along with critical analysis of the results will be presented.
- Chapter 6 summarizes the work and presents its strengths and limitations.

2. State of the Art

This chapter presents basic concepts on types of mobile experiences and cross-platform mobile development tools available in the market.

Section 2.1 introduces the product Cloogy from VPS in detail, Section 2.2 discusses the different type of mobile applications, and Section 2.3 presents a brief introduction to the cross-platform development tools available for mobile applications.

2.1. Cloogy

Evaluating and measuring our energy consumption allows us to optimize it and reduce our monthly bills, without lowering our comfort level. This is where the Cloogy comes into play. It is a mobile application which provides user with an energy management solution. This management solution allows user to monitor and control energy consumption of a residential house. The main purpose of the application is to monitor electrical equipment consumption individually, and control their usage time through specially designed power plugs by VPS. The application combines data gathering devices and visualize in its user interface. The application provides a platform to the user to discover the areas where it can conserve energy. It provides ways to optimize electrical appliances which will help the end user to put an end to waste and unnecessary costs. In brief, below are the main features of the application [4].

- Determine the appliances working time frames, and turn them on and off.
- Know, in real-time, consumption and the savings achieved.
- With its intuitive applications and a web portal accessible from any computer, tablet or smartphone it is easy to manage the home appliances and their consumption.

Cloogy also offers an innovative Energy Management service which helps us to save money on the electric bill and reduce our carbon footprint [4].

2.2.1. Energy Consumption Management

VPS provides different platforms of Cloogy, where the user can check the global consumption of a residential house. The energy consumption management focuses on below features:

- **Monitoring:** Monitor the overall energy consumption and each appliance consumption
- **Control:** Control electrical appliance operation like turning the device on or off from a remote location.
- **Scheduling:** Perform schedule operations on electrical appliances like appliance operation period, eliminating standby consumption, and getting to know the obtained energy savings from the appliances.
- **Profile:** Cloogy creates the profile for each user. This energy profile provides an overview of the total energy consumption of the user in a dashboard.

- **Savings:** The user has an option to save and track energy consumption progress continuously.
- **Reports:** The application provides regular reports of energy consumption.
- **Community:** One of the benefits of the application is that it allows the user to compare performance within the community.

Below Figure 2-1 summarizes in brief the energy consumption management features of Cloogy.

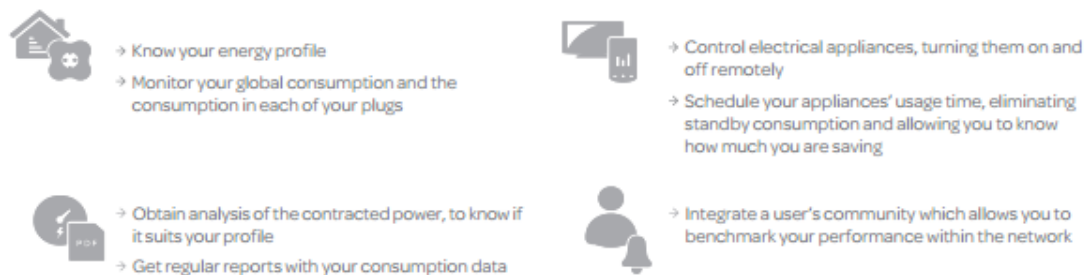


Figure 2-1 Features of Cloogy Application (from [4])

2.2.2. Installation and Operation

Installation of Cloogy follows simple steps. To connect Cloogy in a residential house, one of the first and foremost requirement is a good Internet connection with a free Ethernet port. The user must know that at present, the Cloogy system does not support 3G internet mobile broadband. The installation process begins with the installation of the clamp on the electricity meter as shown in the below Figure 2-2. This clamp is connected to a device called transmitter which sends the data collected by the Clamp to the Cloogy hub. This Cloogy hub receives the data from the transmitter and other Clamps, and then it is responsible for sending it to the main server of VPS. The hub stores the complete energy data and when requested by Cloogy interface, it transmits the data with the help of a router as shown in the below Figure 2-2. All appliances must be connected to the Cloogy power plugs for monitoring and scheduling operations. The power plugs interact with the hub for all the operations. There are different monitoring platforms for Cloogy like it can be personal computer, tablet, smartphone or it can be a simple Cloogy display which shows information about consumption in real time [4].

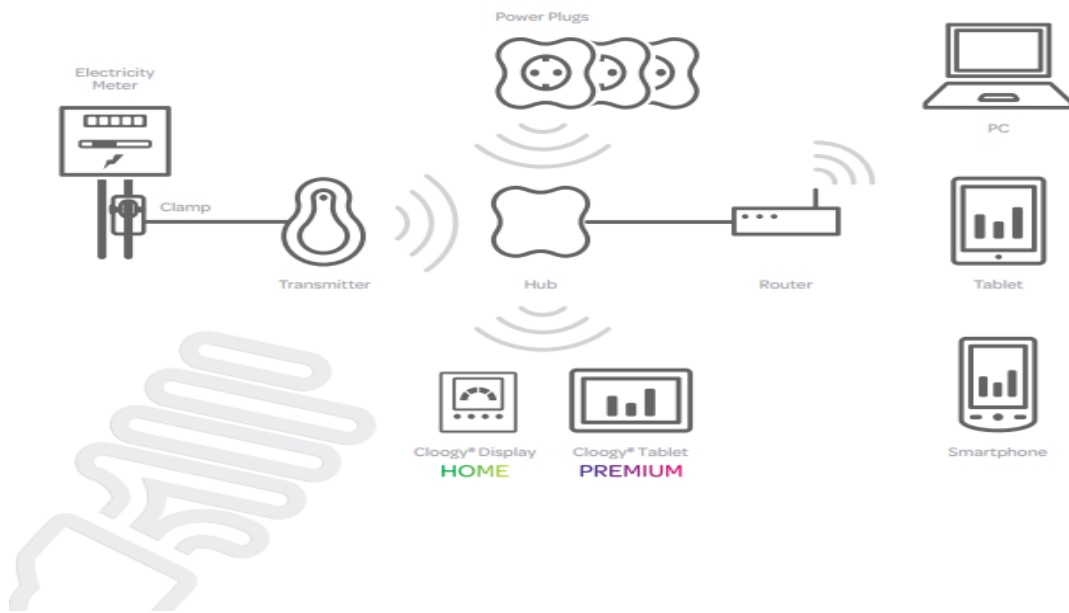


Figure 2-2 The Cloogy Functional Architecture (from [4])

2.2. Types of Mobile Applications

As discussed in the first chapter of the thesis, in Section 1.2, there are different methods to develop mobile applications which will be discussed in this section to better understanding. The types of mobile applications are as follows:

- Native Applications: *written in the native programming language.*
- Mobile Web Applications: *website designed for mobile phone.*
- Hybrid Applications: *single code source having access to all platform APIs.*

Below Figure 2-3 summarizes the comparison between native mobile and hybrid application architectures.

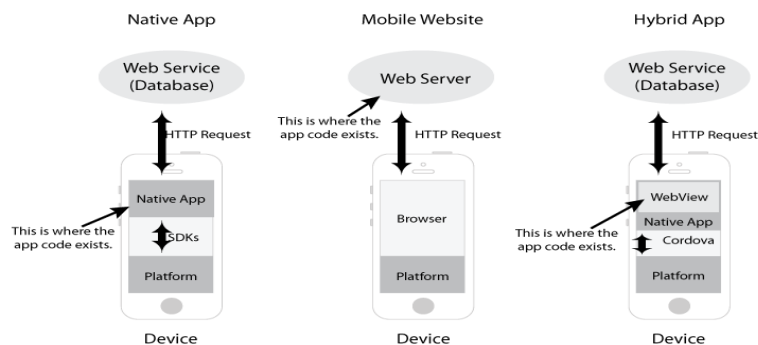


Figure 2-3 Comparison between native, mobile websites, and hybrid application architectures (from [5])

2.2.1. Native Applications

As described earlier in Section 1.2 of the thesis, native applications are built using default programming language according to the mobile platform. To build a native application for iOS, the developer is required to use Objective C or Swift, Java for Android, or C# or Visual Basic for Windows Phone. Then with the help of the platform SDK(API), the application can communicate with the platform to access device data or to load data from an external source using communication protocols [6].

To build a native application, one of the foremost requirement is the selection of the development IDE software application that will provide developers with comprehensive list of options to build software. It is mainly a code editor, or build automation tool with debugging facilities. The development environment can be official or unofficial based on the choice of the developer. Based on the comfort, and common practice, developer tends to use different types of frameworks in their native application to make the development process much easier [6].

Native application has series of features such as it can access to push notifications, have notifications for updates in the application which is considered one of the essentials for a mobile application. Native applications can be downloaded from the application store. For the usage and to make it easier, it has a designated icon on the menu of the mobile device. Native application resides in the device where it has good access to device functionalities such as GPS information, accelerometer, contacts, camera features, etc. It can also be said that native application gives developer more options to build features as they desire as they have deeper integration with the mobile device [6].

Moreover, native application works both online and offline, constantly been active at background irrespective of whether the application is currently used by the user or not. This help application to be up to date with new updates or modifications [6].

2.2.1.1. Advantages of Native Applications

Native application comes with several benefits when compared to the other type of mobile application. Most of the time, the benefits of the native application are integrated with the device platform.

- **Good quality of graphics:** Since the application is built using device`s native language and installed in the device itself, the application offers good quality graphics and animations. For business like game development, native applications are recommended ahead of other types of applications.
- **Application store distribution:** Native applications are always recommended to be distributed through their platform`s application store or the marketplace.
- **Device Integration:** Native application has full access to device`s hardware starting from GPS information, contract list, camera to the microphone, accelerometer. These access facilities are essential for applications that require geographical location or device position like Google maps. It is also important to know that hybrid application offers the same functionalities of device integration whereas mobile web applications offer partial integration.

2.2.1.2. Disadvantages of Native Applications

The disadvantages for a native application are generally the level of difficulties faced in developing and maintaining them.

- **No Portability:** Native applications are not portable from one platform to another. The mobile platforms are largely dominated by iOS and Android, while other platforms like Blackberry, Windows Phones have a minor share in the market. Therefore, for business purpose, building an application just for one platform exclude from all others. Building for all platforms requires time, resource and cost.
- **Platform Instability:** It has been seen in the recent years that popular platform today may disappear in just few years or may not be successful. For example, Blackberry which dominated the mobile industry for many years, is currently struggling to survive in the industry. Therefore, there is a risk factor of wasting time, and cost for the companies who choose the native approach.
- **Development Cost:** The development cost varies depending upon the requirement and application's complexity. It can be said that native application development is an expensive and time-consuming approach.
- **Development Time:** As mentioned in the above point, native application development is a time-consuming approach. If a business looks for native approach, time requirement increases depending upon the requirements and application complexity. For example, a business which prefer cross-platform development will save a lot of time when compared to business preferring building native application across all platforms.
- **Maintenance cost:** While all types of mobile application require regular updates and maintenance, native application requires most of the time when compared with other two types. With new platform releases, the native application must be regularly updated which increases the new work for the developers to duplicate every change or update across all platforms.

Overall the benefits of the native applications are in the areas of graphics, application store distribution, device integration whereas they are difficulties in the areas of portability which is one of the main issues for businesses. Native application requires a reasonable amount of time and cost for the development. In this scenario, hybrid application development has an advantage over native as they also offer application distribution and device integration which will be discussed in brief in the below Section 2.2.3 of this thesis. [3] [6] [7].

2.2.2. Mobile Web Applications

Mobile web applications are not real applications. They can be termed as websites which run on the mobile browsers and have an advantage of operating across all platforms. Mobile web applications are not installed on the device, they are distributed via web. Basically, a mobile web application is a static HTML/CSS/JavaScript page formatted for a use on a mobile device [7] [8].

Mobile web applications lack a lot of the benefits that native applications provide. However, they do have access to some essential features such as GPS, camera and phone calling, which are often sufficient for many applications. With the help of browser caching, it is possible to run the application offline. Other features such as push notifications are currently inaccessible via web applications. Web development with HTML5 gives all types of new features that can be implemented for developing a mobile web application. For example, in 2011, Financial Times newspaper withdrew its native application and launched web application for iPhone users with URL app.ft.com as shown in the below Figure 2-4. This web application has all basic features of a native application starting from swiping horizontally to move from one section to another, read newspaper offline with the help browser caching, and many other features [8].



Figure 2-4 Horizontal swiping on Financial Times's web application (from [8])

HTML5 plays a vital role in web development and is recommended from the W3C, which is the official, non-profit organization that develops and maintains web standards. HTML5 helps developers to modernize the capabilities of native web languages, so they offer all possible functionality to create a competitive mobile web application. In current days, dedicated mobile web applications and hybrid applications are dependent on HTML5 along with the mobile web browsers for rendering the content on the mobile device [3].

2.2.2.1. Advantages of Mobile Web Applications

Mobile websites enjoy several benefits, primarily in the level of work effort and compatibility on the devices across all platforms.

- **Stability:** As discussed in the above section 2.2.1.2, native applications have platform instability as there are new versions launched every year. However, mobile web applications eliminate this problem. In the present, where the mobile device technologies and operating systems keeps on changing, the web is the only constant. It is not controlled by any company and it won't disappear in few years. The mobile web application can be shared and operated on all mobile device operating systems.
- **Cross-Platform:** Unlike native applications, web applications can work across all mobile platforms and operating systems. There is no requirement for different applications to run on platforms like iOS, Android, Blackberry, and Windows Phone. It is the application which will work everywhere. Since mobile web applications runs on the browser of the device, it will run if there are new platforms developed in the future.
- **Lower Development Cost:** If a business already developed websites, they have the skills to develop mobile web applications. Unlike with native development, they do not require new developers with specific skillsets which would increase the development cost to the business.
- **Simple Maintenance:** Since the application runs on every platform, the maintenance consumes less time. One change in the single application will reflect across all platforms.
- **Instant Updates:** Since mobile web applications are not installed on the device itself, all updates instantly reflect in the application when changed at the backend of the application. Unlike native applications, the user is not asked to update the application frequently.

2.2.2.2. Disadvantages of Mobile Web Applications

As described in Section 2.2.2., mobile web applications run inside the browser of the device, which is one of the major cause of limitations and disadvantages for this type of mobile applications.

- **Limited graphics:** When compared with native applications, mobile web applications are not able to handle heavy graphics seamlessly even though they are suited for displaying every type of content on the application. For industries which are into gaming business, selection of mobile web application approach for their business will be a wrong choice as games need a good user interface.
- **Limited device integration:** Even with the rise of HTML5, mobile web applications cannot access the device's contact list or deliver push notifications. However, they have a capability of accessing hardware sensors like GPS, accelerometer, and so on with few limitations.

At the end, mobile web applications offer limited advantage over native applications like cross-platform, low-risk option for the companies. Unlike native applications, mobile web applications are not restricted to one specific platform. Features like cross-platform make mobile web applications simple and inexpensive option from the other two types of mobile application development options. Unless business is looking for a complete device integration or applications with heavy graphics, mobile web applications would be an ideal choice [2] [3] [7] [8].

2.2.3. Hybrid Applications

The combination of native and mobile web applications can be termed as a hybrid application. The development language for hybrid applications depends upon the selection of the framework. However, most hybrid applications are developed using JavaScript, HTML, and CSS. Frameworks like Xamarin from Microsoft requires C# for the development. It all depends upon the business for the selection of development tool. The hybrid application developed using web technologies uses the same code which is written for developing a website. One good advantage of having hybrid applications over mobile web applications is that they run as an application rather than a mobile web application running inside device browser. It is a combination of web and native application which can be deployed across all platforms. With this feature, hybrid applications can allow a large number of users to access the application. Hybrid applications usually rely on the HTML which is rendered in a browser, which is further wrapped in a platform-specific shell. With this shell, hybrid applications have native qualities like device integration, application installation, and platform's application store/market distributions [2] [7] [8] [9] [10].

2.2.2.3.1. Advantages of Hybrid Applications

- **Inexpensive cross-platform development:** Hybrid application development is not simpler when compared with mobile web applications. However, they are cheaper than building a native application which will be deployed across all platforms. The hybrid application has a web application which is wrapped in platform-specific native wrappers.
- **Native-like:** The hybrid code base is wrapped in platform-specific wrapper with the help of the framework. It gives native application look and feel as it runs as a mobile application in the *WebView* of device's browser. Like native, hybrid applications are installed on the device itself and work as a native application.
- **Device integration:** Hybrid development tools provide full device access including the native only features like camera, microphone, and address book. Business looking to deploy their applications across all platforms, will have access to all native-like features in every device.
- **Marketplace distribution:** It an advantage over mobile web applications, hybrid applications can be distributed to consumers through platform-specific applications store or marketplace.

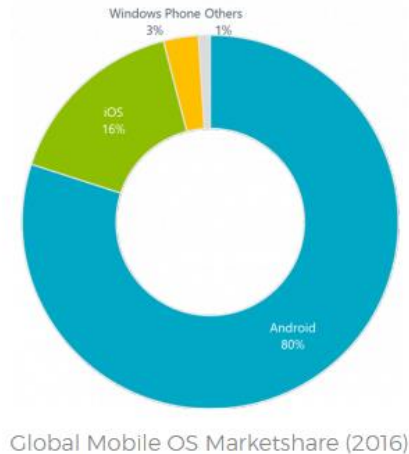
2.2.2.3.2. Disadvantages of Hybrid Applications

- **Limited graphics:** Like mobile web applications, hybrid applications have less graphical capabilities when compared with native applications. However, in recent years, there has been an improvement in the user interface designs with new development frameworks like ionic.
- **Web view limitations:** Since hybrid applications run on the device browser, the UI performance is tied with the version of the platform's browser installed in the device.
- **Increases dependency on third-party APIs:** If there is a new release of the mobile operating system, the developer is dependent on the supplier on the new API to be released. The devices will not be supported without new API, which will run the new operating systems.

While mobile web applications are the simplest solution for mobile application development, hybrid applications can be termed as the best option. Hybrid applications suit the requirement of the business who are looking to develop simple mobile applications across all platforms, combined with marketplace/application distribution and device integration. Hybrid applications will allow businesses to bridge the gaps between native and mobile web applications [2] [7] [8] [9] [10].

2.3. Cross-Platform Development Tools

Before moving forward to understand the development tools available for the cross-platform development, it is imperative to understand the current trend of technology usage. The tradition of using desktop computers is fading away with the rise of the mobile application. To build a cross-platform application, it is required to understand the degree of challenges where a single code should be developed which can be utilized or can be adopted to any native platform or operating system. Android is considered to have the most extensive OS market share as of 2016 [11]. The complete market share of mobile operating systems can be picturized in the below pie chart in below Figure 2-5.



Global Mobile OS Marketshare (2016)

Figure 2-5 Global Mobile OS Market share as of 2016 (from [11]).

There is a demand for mobile operating systems. Therefore, technology enterprises, such as Apple, Microsoft, Nokia, Symbian, and Google are working towards improving the performance of their respective operating systems. Each enterprise has its own product to fit in the market. The developers who are looking to build cross-platform mobile application face challenges because each operating system has its own language, different APIs, and different IDEs. To overcome this problem, there are cross-platform development tools available which provide developers with the possibility to develop source code once and run it across different platforms [12]. The benefits gained by developers working on cross-platform development tools are:

- **Reduced of required development skills:** There is reduction of required programming skills for developers as they will have common programming language across all platforms.
- **Reduction of coding:** The source code will be written once and will be compiled in all platforms. Hence, unlike native, there is reduction of amount of coding.
- **Decrement of API knowledge:** The API will be used by the selected tool. No requirement to know API of each OS.
- **Greater ease of development:** The development will be easier when compared to the effort required for native development.

2.3.1. The Criteria for Development Tool Selection

The criteria for development tool selection are based on predetermined requirements which can be decided by comparing all the development tools available in the market. Later, the selection of the framework can be done by analysing if the frameworks has fulfilled all predetermined requirements. This way, it helps a developer to understand which tool will be suitable for their desired goal [12] [13]. Below mentioned requirements are based on and have been influenced by various resources [2] [7] [8] [9] [10] [12] [13].

- **Platform Support:** This criterion examines if the framework supports all major operating systems. The application developed using this framework should be successfully deployed across major platforms like iOS, Android and Windows Phone.
- **License and Costs:** License and cost are one of the most importance criterion. It examines if the framework in question is an open source or is there any licensing cost involved to work on the framework.
- **Access to platform-specific features:** The framework should have access to platform specific features like GPS or camera, platform functionalities like contacts and notifications.
- **Look and feel:** The appearance of a mobile application is a continuous process which can be achieved even after the complete development of a functional application. However, it gives to the developer an added advantage if the framework inherently supports a native look and feel. Most of the businesses are seeking native features in their hybrid application.
- **Application speed:** At the end, consumers are looking for an application which will have good start-up time with good responsiveness to all application functionalities.
- **Application distribution:** The application should be easy to distribute through application store or platform designated marketplace. It is important as most of the user feel secure if they have an option to install the application from the official distribution channel.
- **Development Environment:** The framework should support good IDEs along with emulator/simulator, debugger and various other features and functionalities.
- **Code Reuse:** The more amount of time spent on changes/code rewrite should be less when compared with implementing same functionalities on different native platforms. The worse can be a situation where the selected framework does not stand by its name of cross-platform development tool if the code base cannot be deployed across all platforms.
- **Learning curve:** This criterion will examine if there is any requirement of additional learning of new technologies or programming language.
- **Documentation and Tutorials:** Detailed documentation with a good number of tutorials, sample examples, codes along with active framework community play crucial role while selecting the framework.

2.3.2. Overview of Cross-Platform Development Tools

During the research, the most common and popular frameworks identified were Appcelerator Titanium, Xamarin, and Apache Cordova. All these frameworks answer the questions raised and help us to develop a hybrid application for multiple platforms. It is imperative to look deeply into these frameworks about technical, business and philosophical aspects so that it helps to understand which framework will suit the requirement defined by VPS [7] [12] [14] [15].

In this section of the chapter, we will discuss in detail about each mentioned development tool for selecting one framework to develop our hybrid application for VPS.

Apache Cordova/Adobe PhoneGap

Apache Cordova which is formerly known as PhoneGap is a framework for mobile application development. Nitobi created this framework, but, later it was acquired by Adobe Systems in 2011. Adobe systems later released an open source version of the software with a name Apache Cordova. Cordova was created to help developers to build mobile applications using web technologies like JavaScript, HTML and CSS rather than using platform specific programming language [16]. Basically, Cordova helps developers to wrap applications written using web technologies into the native applications. Moreover, open-source software is built with the help of many components and extensions. Cordova is a hybrid application, which means it is a mixture of native and web-based applications. Cordova renders the UI of the application via a *WebView* of device browser whereas web-based applications lacks support of HTML in implementing some of the functions. Since Cordova is an open-source framework, there are possibilities for the programmers to develop their own plug-ins. There is no restriction on the usage of IDE (such as XCode for iOS, Android Studio for Android) to develop an application using Cordova. Developers are free to use their own IDEs Atom, Sublime, Visual Studio Code, etc. This feature of the software helps developer to develop the code-base on multiple platforms rather than restricting them to use specific PC's operating systems. There are cases where not all IDE's are compatible with PC's OS. The code base developed using Cordova is open for new changes according to the requirements which is an advantage as a Cross-Platform development tool [12] [17].

Cordova lacks in the UI quality as the application is rendered on the *WebView*. The quality purely depends upon the version of the browser installed in the device. Latest and updated browser versions provide better quality whereas the lower versions lack in this field. Different browsers on different platforms have their own limitations which affects the UI quality. There are security restrictions in place by platforms where some of the native functionalities cannot be implemented. For example, iOS does not allow Cordova to access iOS compass which restricts some development projects [17].

Titanium

Titanium framework was launched in 2006 with a beta version excluding Android and iOS. These two frameworks were added in 2009 and the full version was released. Applications can be developed using programming languages like JavaScript with additional support from PHP, Ruby, and Python. Like Cordova, Titanium allows developers to access native features, UI modules and other optional modules [12] [17]. According to the official document from the company, the re-usability percentage of code-base across all platforms is around 70-80% [18]. The rendering of the application is executed natively as most of the application is developed using native code. Titanium is recommended for building interactive applications. This framework is not suitable for building application with a heavy amount of graphics. The IDE for Titanium is its own development studio. There are extensions available for Eclipse IDE. The company claims that 90% of the native functionalities can be achieved and there is possibility of covering the remaining 10% for programmers to develop their own native APIs with this framework [19]. A good proficiency in JavaScript will increase the ease of use of the framework. Nonetheless, the framework lacks support in the developer's community. The number of coders are lesser when compared with other cross-platform development tools. The official documentation is far from complete and it is not equipped with the right amount of information. The documentation from the framework does not provide proper

information if coders wish to develop their own native API to access some restricted native functionalities. It is also recommended to be aware of the Titanium architecture before starting development process [12] [17].

Xamarin

Hybrid applications built using Xamarin are purely based on C# and .NET. According to the official documentation from Xamarin, 75% of the code-base can be shared across all platforms. If the UI design is built using *Xamarin.Forms*, the usability of code-base increases up to 100%. Xamarin is based on open-source Mono .NET framework, which allows Microsoft to deploy applications across all platforms like iOS, Android, and Windows Phone. Xamarin further provides two mobile development environments, Xamarin.iOS (formerly known as MonoTouch), and Xamarin.Android (formerly known as MonoDroid). Applications can be built using Xamarin’s own IDE which is known as Xamarin Studio or using Microsoft’s Visual Studio. To keep the application native, the UI layer is different for every platform, and the UI must be developed using the specific native API. Xamarin allows programmers to develop an iOS application on Windows platform using Visual Studio which is an advantage over other development tools. The IDE of Xamarin is compatible to add additional plug-ins and components. The fallouts of this framework start from debugging. For example, XCode simulator is required to debug iOS applications. Further, to design the UI for iOS applications, the XCode’s layout and storyboard editor are required. Resources, components, and libraries helps developers to add the required native functionalities in an easier way. However, most of these resources are not compatible with Xamarin which makes the development task quite tedious. For example, to implement simple slide menu library into the application, it requires right amount of code whereas, in other frameworks, they have inbuilt functions or ready-made templates. Here, the official documentation and online resources lacks behind from other frameworks [12] [17] [20] [21].

Table 2-1 summarizes and compares frameworks discussed in this section. It starts from the platform supported by them to the businesses who have adopted these frameworks to implement their respective hybrid applications.

	PhoneGap	Titanium	Xamarin
Platform Support	iOS, Android, Windows, Blackberry	Android, iOS & Blackberry	iOS, Android & Windows
Programming Language	JavaScript	JavaScript, HTML5, CSS	C#
Opensource	Yes	Yes	No
UI	Web UI	Native	Native
Web Standard Support	Yes	No	No
DOM Support	Yes	No	Yes

Native Performance	No	Yes	Yes
Access to Device API	Limited	Full	Full
Used By	IBM, Intel, Sony, Mozilla	Cisco, VMware, Safegaurd Properties, Mistubishi Electric	GitHub, Microsoft, Foursquare, Expensify, Dow Jones

Table 2-1 Comparison Table: PhoneGap, Titanium, and Xamarin (from [14])

2.3.3. The Selection of Ionic with Apache Cordova

The above overview of top three cross-platform development tools will help us to move forward selecting one of them to build our hybrid Cloogy application. Going through the research, we understand that each framework has its own unique advantage and disadvantage. For example, Cordova provides good access to device APIs across all platforms along with easy environment setup for the development process. Titanium helps to build native like application with JavaScript and if developers are comfortable with C# and .NET framework, Xamarin would be also one good choice to develop hybrid application. It has also been observed during the research that the cross-platform mobile application development is dynamic in nature. The rate of new updates is quite high in this field of mobile computing. One must be updated with all new features and research which constantly changes with respect to time. Moreover, it is essential to understand that the overall conclusion to select the framework can be only be based on the subjective analysis of the tutorial blogs, official webpages of the frameworks and the research papers. The overview of the frameworks does not include any practical conclusions on the frameworks. There is no practical implementation of each of the framework to decide the best possible option to develop Cloogy Hybrid application. Above research information in section 2.3.2, is collected from the developer’s community, and user experiences.

The best conclusion could only be drawn from the practical results. However, since the technology changes every month, it would not be advisable to spend a reasonable amount of time proving which framework will be best suited for Cloogy hybrid application. Therefore, based on the research and the results from different authors, and blogs, we can move forward with our final comparison to select the framework.

Going deep into the world of Apache Cordova ecosystem, we were introduced to wide range of new frameworks and tools like Ionic, Onsen UI, Telerik platform, App Builder IDE and more. As we already discussed in previous chapters about hybrid mobile applications, they are like web applications. Both use the same set of web technologies like HTML, CSS, and JavaScript. However, the slight difference between hybrid and

web-based application is in the way they are hosted on a mobile device. Hybrid application target the *WebView* which is hosted inside a native container instead of running on a mobile browser. Cordova provides APIs for accessing device's accelerometer, contacts, camera and more. Applications are built using HTML, CSS, JavaScript which are later packed by Cordova to target platform SDKs. After successful built, the mobile application run like any kind of native application on the device. In the later part, we also learn that Cordova is not an application framework. It can be termed as wrapping framework which can be used to create a hybrid application. The main functionality of Cordova is that it serves as a layer between JavaScript and the native functionalities of the mobile device.

On the other hand, Titanium is framework used to create real native applications using JavaScript code, which is recompiled into native code. When compared to Ionic, Titanium will create a real native application whereas Ionic creates hybrid application served in *WebView* of the device browser. However, Titanium suffers from memory leaks. The company never fixed this issue, even after several years [12] [17].

It also has been noticed that frameworks like Xamarin and Titanium have limited access to open source libraries. This way, they utilize platform specific APIs, which will not help to a great extend to build a single code base [12] [17].

It has also been learned that these frameworks are not suitable for applications with heavy graphics as the loading time is slighter slower compared to Cordova. Considering the type of framework and complexities offered by them, the applications built on Xamarin and Titanium have typically larger size than native ones. According to the below Figure 2-6, in native application development approach of Xamarin, we can notice that a simple hello world application can take up to 16MB of space where much of it is associated with libraries, content, runtime and base class library assemblies [22].



Figure 2-6 Size break of a hello world application developed using Xamarin (adapted from [22]).

Cordova requires much work to make an application look neat, user-friendly and more native like. To fill this gap, new frameworks have been released into the cross-platform development market. Some of the popular frameworks which uses Cordova as its cross-compiler are Ionic, Onsen UI, Famo.us and AppGyyer.

Within this race of popularity, Ionic is emerging as a winner to create native mobile applications. Unlike Titanium, Ionic framework utilizes AngularJS which is later wrapped into Cordova framework. This is an easier approach than using Xamarin or Titanium. Now according to the VPS requirement, the ideal framework for developing Cloogy prototype would be the one which has a simple approach for development, ease to develop with good speed and should be a single code base. In this case, Ionic uses the

power of Angular which is developed by Google, and Cordova by Apache respectively. The front-end code could be used to deploy on any platform which can adapt to Cordova features. Regarding the cost of development and the speed, Ionic wins the battle as it is an open source which means there is no cost involved whereas the discussion is still active regarding Xamarin as an open-source. It is believed that Xamarin is free along with a purchase of Visual Studio. The only thing here is that we need to have developers who are flexible with programming languages [23] [24].

As discussed in the section 2.3.2, debugging is an integral part of any software development cycle. Researching further about the debugging options, we can understand that, Xamarin takes more time compared to Ionic. We need to be patient to debug or test codes in Xamarin whereas Ionic is fast with the help of a ripple emulator. For Xamarin, pushing out code into an iOS device requires several seconds of compilation time and further, it takes more time to deploy the application on the device whereas Ionic's ripple emulator provides zero-compilation, sub-second feedback times. This feature alone significantly increased the development speed [15] [25].

Angular from Google is a very popular framework for creating both mobile and web applications. Features of Angular like extending the syntax of HTML to include components and data binding gives a free hand to a developer working on a cross platform application [15] [25].

At last, what is more important for a developer is a great support from the developer community. No developer is born genius who do not wish to be part of a worldwide developer community while working on a project. Developer communities help the developer to grow, share experiences and most important work/fix/contribute to issues/bugs in the frameworks. It is a great way to be in touch with the world of developers in the times when technology changes every month, every day or even every minute.

Ionic offers great support for its own services and development tools. Ionic also has the biggest community in comparison to other frameworks. Ionic benefits are discussed more in detail in the coming chapters of the thesis [15] [25]. To get an overall decision for the selection of the framework, it would be good idea to check results from a recent poll that was consulted from stackshare.com as shown in the below Figure 2-7. The poll provides results of popularity between Ionic, PhoneGap & Xamarin.

Study and Development of Cross-Platform Cloogy Mobile Application for VPS










	 Ionic	 Xamarin	 PhoneGap
	Cross-Platform Mobile Development	Cross-Platform Mobile Development	Cross-Platform Mobile Development
	Favorites ★ 181	Favorites ★ 61	Favorites ★ 18
	Stacks 1.12K	Stacks 254	Stacks 296
	I Use This	I Use This	I Use This
	Fans: 1.05K Votes: 1.5K Jobs: 49	Fans: 290 Votes: 546 Jobs: 32	Fans: 231 Votes: 85 Jobs: 26
Hacker News, Reddit, Stack Overflow Stats	Y: 669 V: 369 J: 4.47K	Y: 993 V: 728 J: 26.2K	Y: 439 V: 579 J: 750
GitHub Stats	31.6K stars 9.03K forks about 22 hours ago	No public GitHub repository stats available	4.19K stars 1.01K forks about 21 hours ago
Description	A beautiful front-end framework for developing hybrid mobile apps in HTML5. Best friends with AngularJS.	Create iOS, Android and Mac apps in C#	Easily create mobile apps using HTML, CSS, and JavaScript
Why people like using this tool	<ul style="list-style-type: none"> ▲ 221 Allows for rapid prototyping ▲ 198 It's angularjs ▲ 196 Hybrid mobile ▲ 161 Free ▲ 159 It's javascript, html, and css ▲ 94 Ui and theming ▲ 69 Great designs ▲ 65 Mv* pattern ▲ 62 Reuse frontend devs on mobile ▲ 56 Extensibility 	<ul style="list-style-type: none"> ▲ 92 Power of c# on mobile devices ▲ 60 Native apps with native ui controls ▲ 57 Native performance ▲ 52 Sharing more than 90% of code over all platforms ▲ 51 No javascript - truely compiled code ▲ 32 Ability to leverage visual studio ▲ 30 Mvvm pattern ▲ 30 Many great c# libraries ▲ 27 Amazing support ▲ 25 Powerful platform for .net developers 	<ul style="list-style-type: none"> ▲ 44 Javascript ▲ 12 Backed by Adobe ▲ 10 Free ▲ 8 Easy and developer friendly ▲ 4 Support more platforms ▲ 2 It's javascript, html, and css ▲ 1 Runs on mobile browser ▲ 1 Not bound to specific framework ▲ 1 Powerful Framework ▲ 1 Common code base across all mobile platform
Companies using this service			
Integrations			
Latest News	<ul style="list-style-type: none"> Updates for all: Ionic-angular 3.7 and more! (September 28, 2017) Built with Ionic: Microsoft Flow app (September 27, 2017) iOS 11 Checklist (September 21, 2017) See more news	<ul style="list-style-type: none"> Xamarin at Local Developer Events this October (September 29, 2017) Xamarin.Forms Stable Comes to .NET Standard 2.0 (September 28, 2017) Webinar Recording Exploring UrhoSharp 3D with Xama... (September 27, 2017) See more news	

Figure 2-7 Comparison Poll Results between Ionic, PhoneGap & Xamarin as of October 2017 (from [26])

2.4. Summary

Based on the research results, author's contributions, research papers, user and developer communities' experiences, Ionic could be one of the suitable framework to develop Cloogy hybrid application. The following are the reasons for Choosing Ionic framework:

- Beautiful front-end framework for developing cross-platform mobile applications in HTML, JavaScript, and CSS
- Well supported by Apache Cordova and Google's Angular JS.
- It is an open-source software licensed under MIT.
- The right amount of resources for UI and theme design.
- Supports MVC pattern.
- Suitable for enterprise applications with heavy graphics.
- Code re-usability across all platforms.
- Easy debugging options across all platforms.
- Good worldwide community across the globe.

3. Planning

Project planning is an important and fundamental task. It will help developers to identify necessary resources, to fulfill project requirements and accomplish goals within the timeframe available.

Top-down approach is one of the popular and standard approaches for project planning. Here, project requirements are defined and later these requirements are divided into smaller tasks/modules. Small modules will help the development team to estimate the time required to complete the project. This way, the software developers are measured on the ability to predict the future and be right on their predictions [27]. Project planning works around two questions and they are as follows:

1. What are the tasks?
2. How long will the tasks take?

By answering the above two questions, we can implement the top-down approach. It means to break down problems/tasks into smaller tasks to encounter the problem [27].

In this chapter, Section 3.1 presents the initial plan of the project and section 3.2 presents the final plan of the project in a Gantt Chart using Microsoft Project 2016 software.

3.1. Initial Plan

In this section, the initial plan is presented using Scrum from agile methodology framework. By following scrum, the inputs were taken from VPS who acts as a product owner. With the help of the inputs, a product backlog is created as shown in the below Figure 3-1. The product backlog is a list of required features, functionalities and other aspects of the hybrid application. After developing the product backlog, a sprint planning is scheduled with VPS development team and supervisors. In this meeting, sprint backlog is created by dividing the tasks into smaller modules. And then, the development process starts with the first module [28].

There is a constant interaction within the VPS development team and with the supervisors of the organization for which the application is developed. The supervisors/development team of VPS has the flexibility to request modifications even at the later stages of development which is quite important to meet expected UI of the Cloogy application in both Android and iOS. This methodology allows breaking down many requirements into manageable modules, and thus enables maximum utilization of resources. Therefore, according to the initial review of the existing Cloogy application both in android and iOS, we have defined our application development cycle into four main modules.

These stages can be termed as follows:

- Application Requirements.
- Application Design.
- Application Development.
- Application Demonstration.

Based on the above four mentioned stages, the Gantt Chart is prepared to create an initial plan for the development of the application which is shown below in figure 3-1.

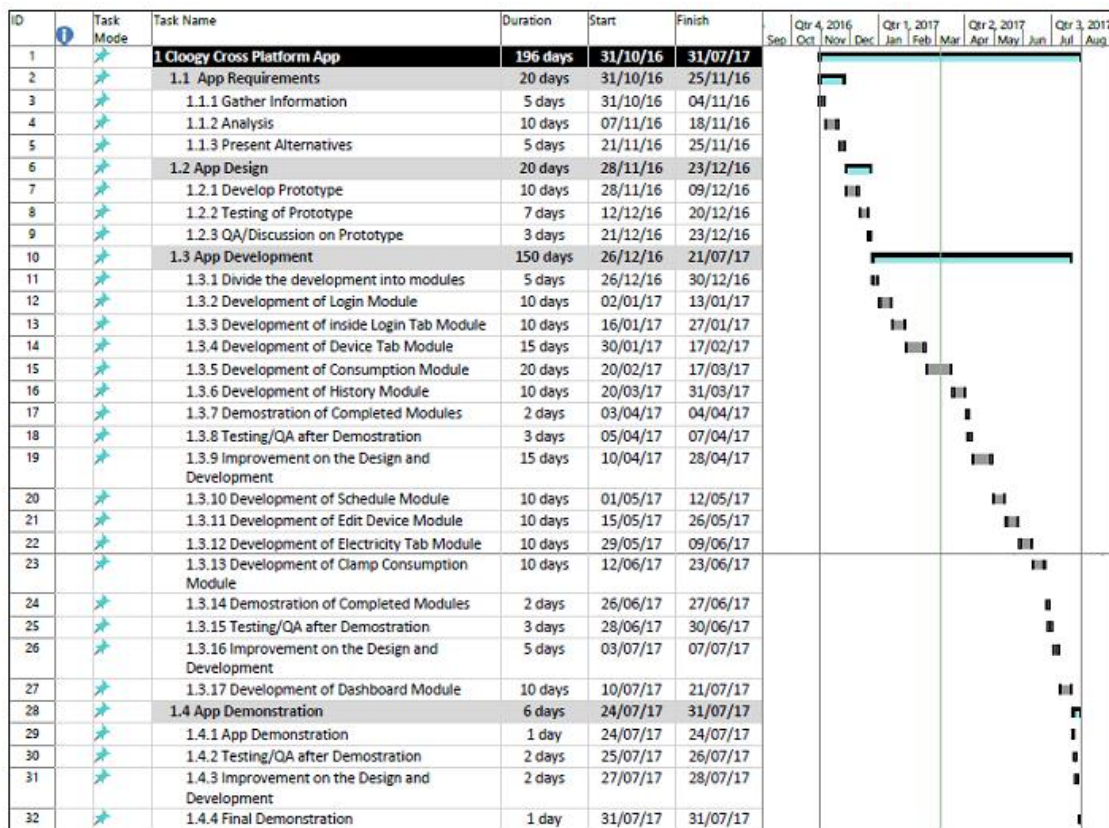


Figure 3-1 Initial Project Plan in a Gantt Chart for Cross Platform Cloogy Application.

3.2. Final Plan

The project was developed according to the initial plan proposed in above figure 3-1. The final demonstration of the prototype was planned on July 31, 2017, as seen in the above figure 3-1. However, due to professional commitments of supervisors at VPS, the final prototype was demonstrated on August 2, 2017, a delay of 2 days from the initial plan.

4. Objectives and Methodologies

This chapter clarifies the goals of the internship and how they are achieved by understanding the requirements for the development discovered using Agile SDLC model. These requirements are extracted from the use case scenarios of the existing Cloogy application. These scenarios describe the usage of the current Cloogy application available in Android, Windows Phone, and iOS.

The application consists of two active components, the iEnergy3 API, and the mobile application which acts as an interface to the user.

iEnergy3 API conforms to the RESTful principles of the HTTP protocol. It provides a friendly, robust and predictable development environment. In this chapter, we will understand the RESTful principles of HTTP protocols in brief. Also, there will be a section in this chapter where we will go through iEnergy3 API and its operations. Moreover, this chapter will provide brief information about a set of systematic methods, frameworks, programming languages in this project.

Section 4.1 discusses the functional requirements of the project, whereas section 4.3 discusses the non-functional requirements. Section 4.2 presents the use cases based on the functional requirements discussed in section 4.1. Section 4.4 introduces RESTful web services, section 4.5 provides an overview of the iEnergy3 platform, and at the end, section 4.6 gives an overview of the web technologies to be utilized in this project.

4.1. Functional Requirements

Regarding the functionality of the Cloogy application, several use case scenarios have been identified. These are recognized by analyzing the existing Cloogy native application, which has been developed separately for Android, Windows Phone and iOS.

The native application of Cloogy is designed to evaluate and measure household energy consumptions. It allows residents to optimize and reduce monthly bills, without lowering the comfort level. By analyzing the native application of Cloogy, we defined the below use cases for our cross-platform prototype. The prototype will be built from the start, by first implementing the existing native features into the cross-platform prototype.

- The user can monitor the global consumption and the consumption of the electrical appliances that are connected to the Power Plugs.
- The user can control the electrical appliances by turning them on and off remotely.
- The application eliminates standby consumption by allowing to schedule the usage time of the appliances.
- The user can check their energy consumption profile.
- It has a built-in feature to present contracted tariff analysis to know if it suits the profile of the user.
- The user can check consumption reports about how much energy has been saved and will be saved in future.
- The user can become part of a user`s community which allows them to benchmark their performance within the network.
- One of the features of the application is that it let the user to define alerts for unusual consumption.

4.2. Use Cases

Before the actual implementation, it is important to know what are the use cases that we must implement. Since the application is built from starting, below are the use cases defined to understand the existing functionalities of the native application. The below use case illustrated in the figure 4-1 is drawn based on the scenarios projected in section 4.1

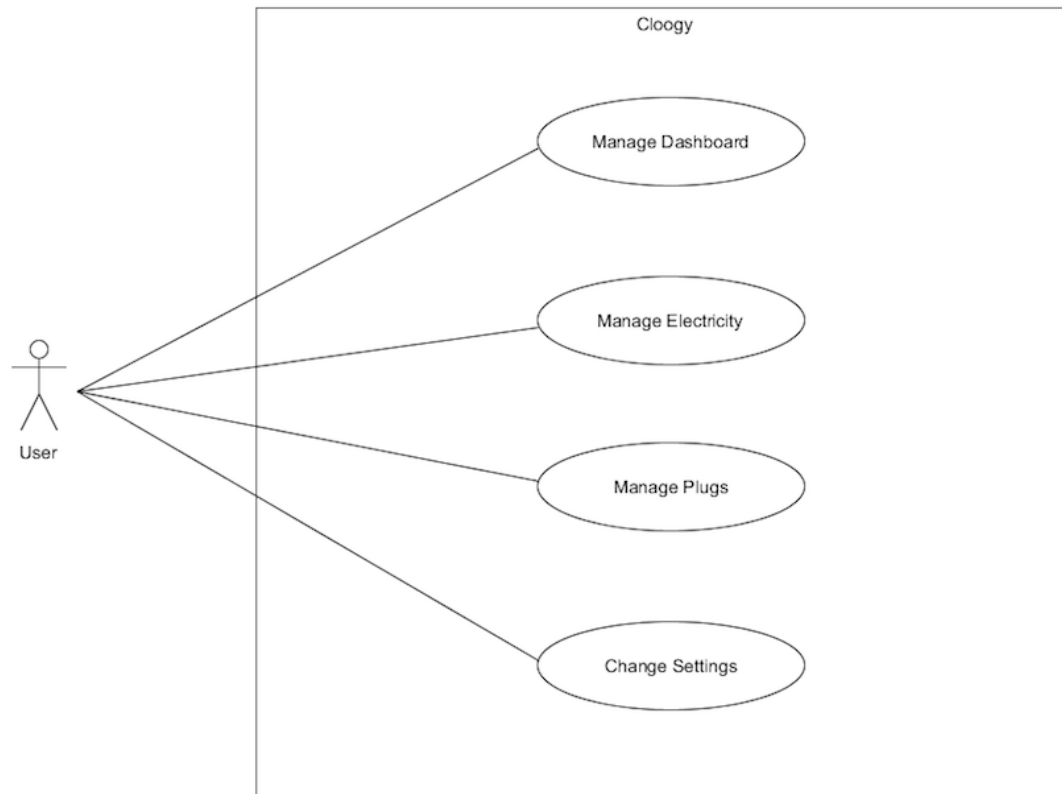


Figure 4-1 Use Case Diagram for Cross-Platform Cloogy Application Prototype

4.2.1. Dashboard

The dashboard is the first interaction area for the user after logging into the application with valid user credentials. The dashboard includes the most elementary function of the Cloogy application that is to provide summarized information on what seeks. Therefore, use case was defined in the below table 4-1 for dashboard module.

Table 4-1Use Case for Dashboard Module

Identifier	UC - 01
Name	Dashboard Module Management
Actor(s)	User
Pre-condition	User logs into the application
Post-Condition	The user has valid user credentials
Main Flow(s)	<ol style="list-style-type: none"> 1. The user is presented to the dashboard of the application. 2. The application connects to the iEnergy3 API and presents total power in use in terms of kilowatt at the top. At the bottom, it presents currency utilised for today and for the month. 3. The user has an option to set currency utilisation goal for the month.
Alternative Flow(s)	<ol style="list-style-type: none"> 1. The user is asked to provide correct user credentials if enters incorrectly. 2. The can click on register user to create new Cloogy account. 3. Forget password hyperlink can be clicked if user forgets the password 4. The application is not able to connect to iEnergy3 API. User is asked to try login in after some time.

4.2.2. Electricity

This is the second tab of the application where the user can view the electricity consumption details for today, for the week, for the month and the year. Each tab will have specific information about the current electricity consumption value captured by the clamp connected to the electricity meter. The consumption values are presented for the current date with a forecast projecting the consumption value by the end of the day. Similarly, consumption values for the week, month and year are presented in this module of the application. Therefore, below table 4-2 was defined for electricity module.

Table 4-2 Use Case for Electricity Module

Identifier	UC - 02
Name	Electricity Module Management
Actor(s)	User
Trigger	The 'electricity icon' in the tab menu of application is selected.
Pre-condition	<ol style="list-style-type: none"> 1. The application has started and is connected to the iEnergy3 API from the server. 2. The `electricity icon` is active.
Post-Condition	The application presents the electricity consumption for the day, week, month and year.
Main Flow(s)	<ol style="list-style-type: none"> 1. The application presents the user with electricity consumption details retrieved from iEnergy3 API. 2. The user selects `chart icon` to view the consumptions in form of a bar chart. 3. The user has an option to view the charts in terms of now, daily, weekly, monthly and yearly. 4. The user has an option to compare the consumption values from today`s date with corresponding last week date in a line chart. Similar comparison can be drawn for corresponding current value, weeks, month and year. 5. The user can view the bar chart in terms of kilowatt consumption or currency consumption for now, today, week, month and year.
Alternative Flow(s)	The application is not able to connect to the iEnergy3 API to render the charts. Loading spinner is shown until the connection has not been established with the API.

4.2.3. Plugs

The plug tab of the application displays a list of devices that are connected to Cloogy power plugs. All the devices are connected to the power plugs so that the user can manage, monitor and control the devices remotely with the help of the application. Each device list has various information regarding energy consumption, schedules of the device, the history, a toggle button to switch on or off the device, a display of current energy consumed by the device and the total energy consumed for the day. Therefore, below table 4-3 was defined for plugs module.

Table 4-3 Use Case for Plugs Module

Identifier	UC - 03
Name	Plugs Module Management
Actor(s)	User
Trigger	The `cloogy power plug icon` in the tab menu of the application is selected.
Pre-condition	<ol style="list-style-type: none"> 1. The application has started and is connected to the iEnergy3 API from the server. 2. The `cloogy power plug icon` is active.
Post-Condition	The application presents a list of devices that are connected to the Cloogy power plugs.
Main Flow(s)	<ol style="list-style-type: none"> 1. The application presents the list of the devices that are connected to Cloogy power plug from the iEnergy3 API. 2. Each UI card design of plug presents today`s consumption value in terms of currency and current power utilised by the device. 3. The user toggles the device from power on to power off or vice versa. 4. The user views consumption, history, schedule for each of the devices in the list. 5. The user selects `chart icon` to view the consumptions for the device in form of a bar chart. 6. The user views the charts in terms of now, daily, weekly, monthly and yearly. 7. The user compares the consumption values from today`s date with corresponding last week date in a line chart. Similar comparison can be drawn for corresponding current value, weeks, month and year. 8. The user views the bar chart in terms of kilowatt consumption or currency consumption for now, today, week, month and year. 9. The user clicks on `history` button which will present timestamp for the activation (the time when device was powered on or off). 10. The user clicks on `timer` icon to add future activation/deactivation schedule for each of the devices in the list.

	11. The user clicks on `edit` icon to change the icon or the device name on the list.
Alternative Flow(s)	The application is not able to connect to the iEnergy3 API to render the charts. Loading spinner is shown until the connection has not been established with the API.

4.2.4. Settings

The settings tab of the application is the last tab, where the copyrights and version details of the application are shown on a list card. The same requirement was implemented in our hybrid prototype.

4.3. Non-Functional Requirements

Non-functional requirement is a systematic approach to build quality into the software applications. The applications must exhibit software qualities like performance, security, maintainability, etc. [29]. Below are the defined non-functional requirements for the cross-platform Cloogy mobile application prototype.

Performance

If not like a native application, the prototype should be good enough providing no performance issues.

Maintainability

It should be easier for the developer to make changes to the application in the background and later, make it available for the users. It should allow developers to maximize efficiency, reliability and work on new native functionalities.

Security

The application will contain sensitive information about user`s energy consumption information. The application should be capable of securing the information when exchanging information, and through terminating the in-active sessions and asking the user to login again to create a new session.

Usability

According to the defined goals, the hybrid applications should have the same appearance and behavior of existing native applications of VPS. The user should not get a feeling of using different application of Cloogy in terms of native functionalities.

Portability

The hybrid code base should allow developers to re-use the code of the framework on other platforms.

4.4. RESTful Web Services

Before proceeding further, it will be useful to go through basic concepts of RESTful web services. The front-end hybrid application will be purely based on iEnergy3 API which will conform to RESTful web services.

REST is defined as an architectural principle which helps developers to design web services. These web services focus on system resources, which includes addressing of resource states and transfers over HTTP protocol written in different programming languages. Over the years, REST has been the front runner in web services. Famous protocols and standards of REST which are widely used in the industry, are SOAP and WSDL. These models are simpler in style and widely used by developers [30].

The REST web service follows four basic design principles which are as follows [30]:

- Use HTTP methods explicitly.
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JavaScript Object Notation(JSON), or both.

By following REST, developers need to use HTTP methods in a way that it should be consistent with the protocol definition. The design principle of REST is based on one-to-one mapping between CRUD operations and HTTP methods. To create a resource on the server, developers need to use POST operation. To retrieve a resource from the server, GET operation should be used. To change the state or update resource, PUT operation should be utilized. Moreover, to remove or delete a resource, DELETE operation should be utilized. Together, all the above four operations are known as CRUD operations of REST principles [30].

4.5. Overview of iEnergy3 Platform

The following architectural diagram of the iEnergy3 platform represents the interaction between the server with the services channel and the other devices such as iHub, Cloogy, iMod and Cloogy Home Appliance as shown in the below Figure 4-2.

According to VPS, communications servers are open, standards-based computing systems that operate as a carrier-grade common platform for a wide range of communications applications and allow equipment providers to add value at many levels of the system architecture. The communication server follows HTTP protocol to interact with the APIs which provides RESTful data service requests to the user interfaces [31].

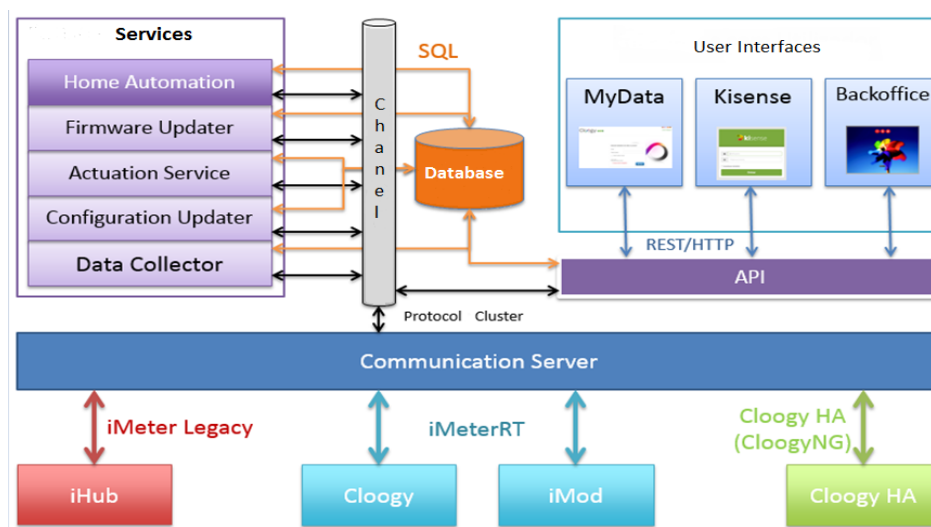


Figure 4-2 iEnergy3 Platform Architecture (from [31])

The user interfaces include MyData which is a user interface dedicated for domestic clients where all the data reports about the energy consumption are displayed,

Kisense is more oriented to business clients like buildings, industries etc. to analyses energy consumption data in real time, providing vital information and knowledge so that our company's decision-making concerning energy management is easy and quickly adapted. and the Back office to the iEnergy3 platform.

The services by the iEnergy3 platform are as follows:

- **Data collector** – Service responsible for querying the remote devices (hubs) for historical data and store it on the database
- **Configuration Updater** – Service responsible for configuring the parameters and variables on the remote devices
- **Firmware updater** – Service responsible for updating the firmware (i.e. software) that runs on the remote devices.

4.5.1. iEnergy3 API

The iEnergy3 API is provided by the iEnergy3 server of VPS to applications for usage. The API conforms HTTP protocol which helps developers to build a system with machine friendly, robust, and predictable interface. The data transmitted from the server uses JSON data format. The server of iEnergy3 of VPS has four main concepts that are central to data server platform [31]. These are as follows:

- **Local:** Local correspond to the physical location of the house or building where the electricity metering hardware is installed.
- **Unit:** It is the hardware responsible for communicating with the server through a TCP/IP connection. It is located at user’s residential house and is responsible for receiving data reading from the devices.
- **Device:** This is the hardware which is being monitored. Devices read several parameters starting from current, power, etc. and are responsible for sending the collected data to the governing unit. Unit can have more than one device associated with. Devices can be a socket plug which can be used to monitor consumption of an appliance. It can also be a clamp which can be used to monitor current on a specific wire. The communication between unit and device can be wired or wireless.
- **Tag:** It is responsible for storing data readings of a specific device. There are devices which measure current and power. For these devices, there will be two tags associated with, one for current and another for power. Therefore, if a device has more than one tag, then it can be said that it measures on more than one parameters.

Most of the operations must be invoked using the HTTPS protocol. Once invoked, an HTTP authorization header like “Authorization: ISA <session-token>”. The session token is received as response after the successful session is created by the applications [31].

Each request to iEnergy3 API gives specific response code in return. Below Table 4-4 lists the possible response code with their meaning.

200 OK	Successful request. Response body contains the resource representation requested
201 Created	Resource was successfully created.
204 No Content	Resource was successfully updated and its representation didn't change compared to what was sent by the client. Or, resource was successfully deleted.
304 Not Modified	Your cached resource is still up-to-date.
400 Bad Request	The request contains invalid data, check error details.
401 Unauthorized	Insufficient credentials to access resource.
404 Not Found	Resource not found.
405 Method Not Allowed	The verb used in the request is incorrect.
500 Internal Server Error	General server side error. Please try again later.

Table 4-4 Possible Response Codes for a specific request to iEnergy3 API (from [31])

4.5.2. iEnergy3 API Operations

To perform API operations, we need to understand the basic implementation methods of RESTful principles, where REST provides a way to access resources from an environment [32]. In our scenario, if our front-end application needs any of the resources from iEnergy3 API, then it needs to send a request to the server to access these resources. To get the resources from the server, let us understand the key elements of RESTful implementations which are as follows:

- **Resources:** The first key element is the resource itself. For example, the URL to get list of devices connected to the power plugs is <http://origin.isaenergy.pt:6600/api/1.4/devices>. Now to access device with id 1568 via REST, the application can issue the command as <http://origin.isaenergy.pt:6600/api/1.4/device/1568>. This command tells the web server to provide the details of a device whose ID would be 1568. The successful response result can be seen in the below Figure 4-3. The result is presented using POSTMAN which is a REST Client application where API's can be called and responses can be seen.

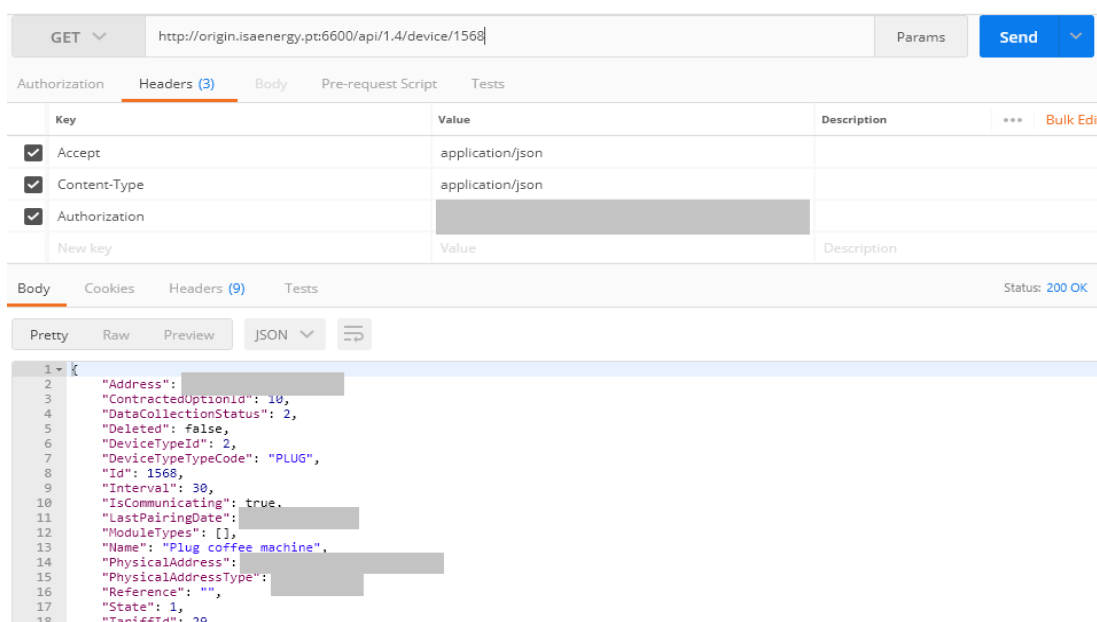


Figure 4-3 Command response in POSTMAN to get the device resource by providing the device ID

- **Request Method:** These describes what we want to do with the resource. An application can issue a GET verb to instruct the endpoint it wants to get the data from the server. However, there are many other CRUD operations available as discussed in section 4.4 of this chapter. In the above case of the example, <http://origin.isaenergy.pt:6600/api/1.4/device/1568>, the application is issuing a GET method because it wants to get device details from the server for a device with id 1568. The same can be seen in the Figure 4-3.

- Request Headers:** These are the additional instructions which may be required to be sent along with the request. For example, according to the requirement, we need to describe how to process our message. To achieve this requirement, Internet media types (IMT) which are previously known as Multipurpose Internet Mail Extensions(MIME) can be used to make the messages self-descriptive.

Key	Value
<input checked="" type="checkbox"/> Accept	application/json
<input checked="" type="checkbox"/> Content-Type	application/json
New key	Value

Figure 4-4 Snapshot from POSTMAN where request headers has been provided along with the requests.

- Request Body:** Here a dataset is sent with the request. For example, according to the requirement, to create a session on the API, the *CreateSession* method logs a user in the API by providing username and password in request body as seen in the below Figure 4-5.

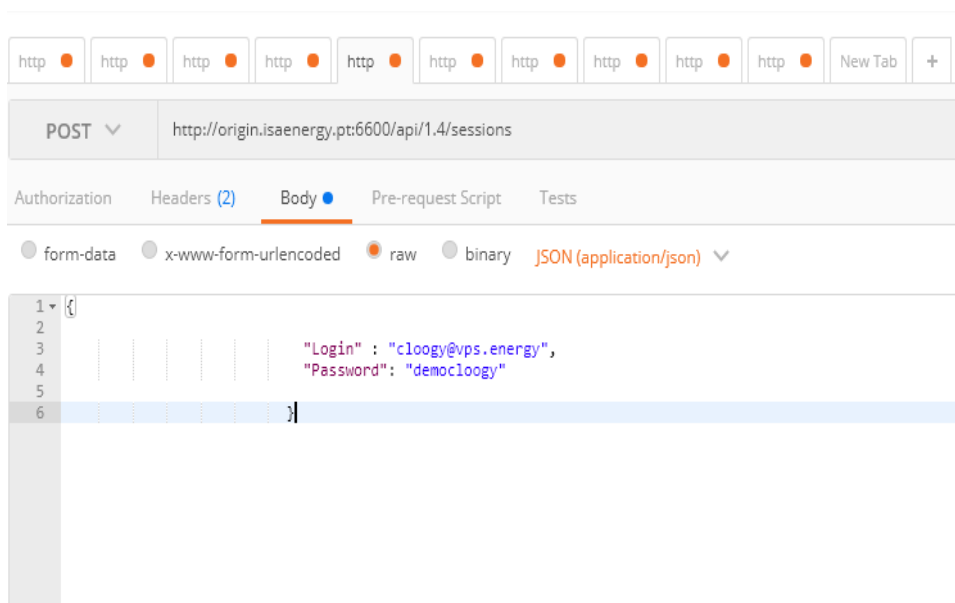


Figure 4-5 Snapshot of request body provided while creating a session with an API

- **Response Body:** This is the main body of the response. For example, after successful authentication, the server returns a token which is later utilized to query the server.

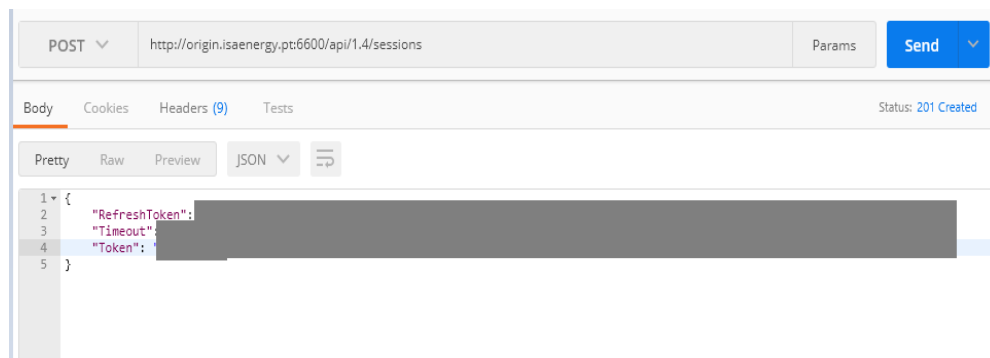


Figure 4-6 Response Body Snapshot from POSTMAN after successful creation of session with iEnergy3 API

4.6. The Development Stack

As we move forward understating more about the World Wide Web(WWW), we come to know that it has a vast amount of resources, which are identified by URI's and interlinked by hypertexts links. These web resources are mainly text documents formatted with Hypertext Markup Language (HTML). Later, these HTMLs are styled with Cascading Style Sheets (CSS) and can be programmed with JavaScript to enhance the user experience.

Since the development work is mainly focused on building cross platform mobile applications sharing the same code base, it was imperative to carry out detailed research about the selection of the framework. In section 2.3 of the second chapter, we understood and analyzed different types of development tools available for the cross-platform development and later, in the chapter, it was decided to develop our application using Ionic Framework. Going forward, we will discuss the development stack in briefly which will be utilized for the development process of the application. The selection of the framework was based on the facts like popularity, community and contributor activities (open source projects), learning curve and time and effort that is required to build a fully functional cross platform mobile application.

4.6.1. AngularJS

AngularJS is one of the famous structural framework for developing web applications. To build single page applications, AngularJS is one of the widely used framework. This framework is maintained by Google and the community of AngularJS developers. There are many advantages of using AngularJS like being able to structure the client-side codes systematically using the Model View Controller (MVC) design architecture, data binding, dependency injection, HTML extension via directives, etc. From the below figure 4-7, we can understand that when the user starts the application, the controller selects the suitable view to perform the requests. These requests invoke the methods created in service or factory to retrieve data. In the other side, the service or factory calls WCF

RESTful services using HTTP protocols. After a successful request, the service or factory provide a response in JSON to the caller which is later visualized in the view. One of main advantages of using AngularJS is being able to organize the codes in smaller modules. This enables the developer to maintain the code easily and reuse the same module in multiple applications by injecting the modules in an application where it is needed [33].

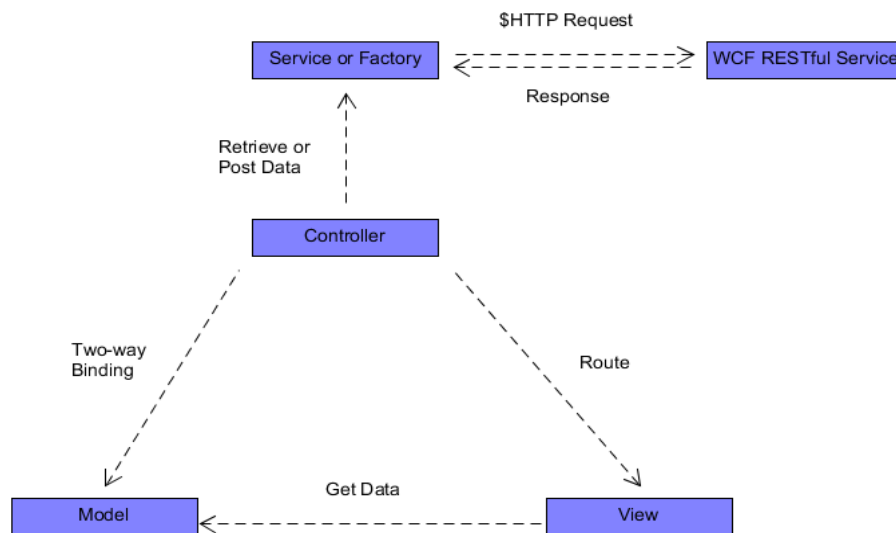


Figure 4-7 AngularJS MVC design pattern (from [33])

4.6.2. Apache Cordova / PhoneGap

The applications developed using web technologies like HTML, CSS and JavaScript, are wrapped depending upon the platform with the help of Cordova. The functionalities implemented in the mobile project using JavaScript and CSS are extended and implemented in the real mobile device with the help of Cordova. In the end, the final application can be termed as a hybrid application. It means neither the application will be truly native or web-based, it will be a mix of both technologies where the layout of the application is rendered via *WebView* instead of platforms' native UI framework. Cordova is the ecosystem for many new frameworks which helps developers to build hybrid applications. All these frameworks are built on top of Cordova. For example, frameworks like Visual Studio, Framework7, Intel XDK, Ionic and many more [16].

4.6.3. Ionic Framework

When it comes to the development of mobile applications using web technologies like JavaScript, HTML and CSS, Ionic is one of the popular framework. This framework provides all the required SDKs for developing a mobile application. Ionic has online documentation which has sample codes to create mobile application designs. Ionic is platform independent whereas native application development requires respective development tools based on the platform. Unlike native, Ionic development process consumes less time, and have direct access to device APIs in Cordova. To work with Ionic, it is imperative to have good knowledge of AngularJS framework. Most of the aspects of development in Ionic deals with AngularJS basic concepts like directives, controllers and scope [34].

Ionic documentation guide for developers has a good number of templates like the one in Figures 4-8, which helps developers to customize these templates according to the requirement. This kind of documentation help developers to spend less time in designing the application UI and make it easier to develop hybrid applications. Going through the benefits of this framework, it is imperative to say that it is a 100% free and open source project which is licensed under MIT [35]. Ionic helps to build cross-platform application with one single code base. It helps to deploy the application across all major platforms without any significant changes in the UI. Ionic offers around 70 native device features starting from Bluetooth, Finger Print Authorization, and more with the help of Cordova plugins. When it comes to empowering application with efficient accelerated transitions and touch-optimized gestures, we notice that application built using Ionic do not face any challenges in terms of speed and performance. As seen in the Figure 4-8, Ionic mobile application development is clean and simple. It works well on all current mobile devices and platforms.

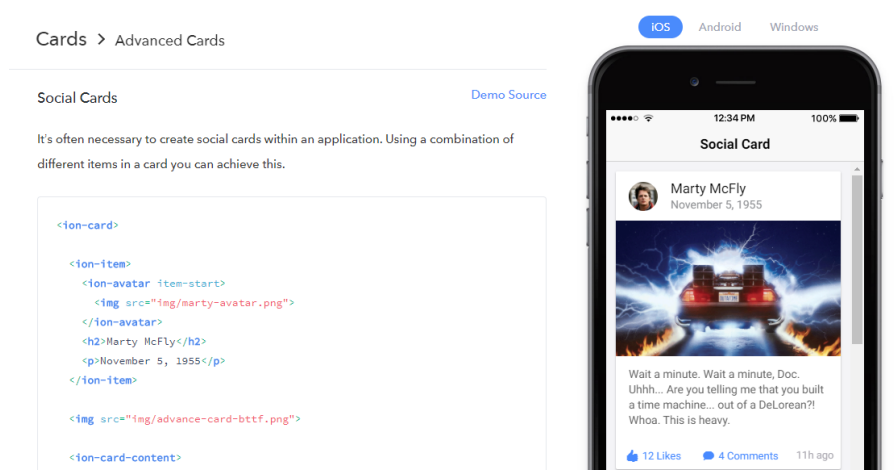


Figure 4-8 Card view code snippet from Ionic Documentation (from [24])

One of the main advantages of this framework is that it uses AngularJS MVC architecture for building single page applications which are optimized for the mobile devices. Functionalities of the application are implemented using JavaScript along with CSS components to provide aesthetic look and feel to the UI. To build, test, and deploy the application, Ionic has its own CLI with simple commands powered by NodeJS.

Compiling and redeploying applications is a tedious task when it comes to web development. Ionic comes with a feature called live reload, which re-compiles the application automatically once the application has been saved in the development environment. Selection of the framework is a matter of choice of the developer and popular frameworks do have a good developer community. These kind of developer forums helps developers to grow and share their roadblocks with the other developers. StackOverflow and GitHub are filled with common questions regarding development. These two places are the best place to reach out for answers for development roadblocks and frequent questions. Figure 4-9 summarizes the development community of this framework.



Figure 4-9 Ionic Developer's Community at a glance (from [24])

5. Developed Work

This chapter describes the implementation of the functional requirements defined in the previous chapter. Moreover, it contains first impressions on the resulting application of cross platform Cloogy mobile application.

As discussed in earlier chapters, the primary objective of this thesis is to replicate the existing Cloogy application of VPS in Android, iOS and Windows Phone into a cross platform mobile application prototype which will work across all platforms having a single code base.

The development work of the thesis is to be able to leverage all technologies discussed in chapter 4, to create the Cloogy application prototype.

The application will implement the concept of decoupled architecture where the back-end and front-end are entirely detached and unaware of each other. They will communicate with each other using iEnergy3 API which will try to conform to the RESTful web services, thus providing a machine friendly, robust, and predictable interface to the user. One important point to remember is that all data will be transmitted using the JSON data format.

5.1. Project Folder Structure

After setting up the environment for Ionic, below project directory shown in the figure 5-1 is created by default by the framework. It is essential for a developer to understand the purpose of every directory and files before starting the development process.

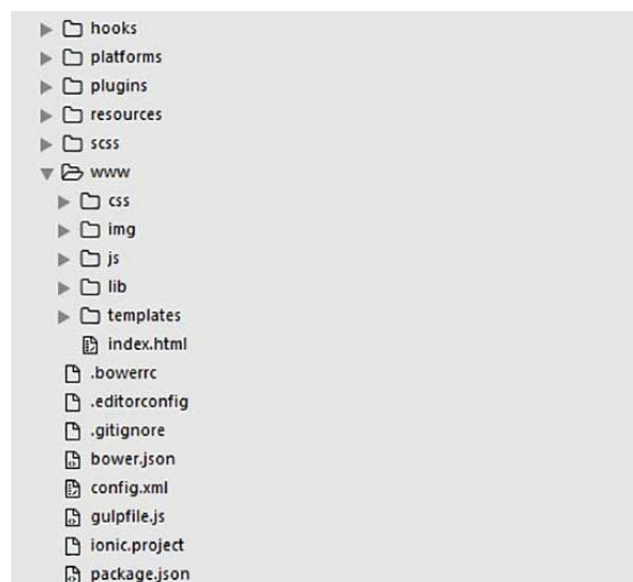


Figure 5-1 Default Project Structure created by Ionic

The default project directory has many folders starting from hooks, to platforms, plugins, resources, scss, and at the end, the www folder. Each folder has files associated with its special purpose. Below is the overview of the folders in the project directory:

- **hooks:** this folder has the scripts which are triggered during the build process of the application. These scripts have Cordova commands to build the application.
- **platforms:** platform specific folders are created inside this folder. If there is any requirement to change some line of code for a specific platform, that operation can be performed by searching the specific platform folder. The folder names are based on the platform such as Android or iOS.
- **plugins:** this folder stores all Cordova plugins that will be used in the application.
- **resources:** any resources that are added to the application like icon or splash screen are stored in this folder.
- **scss:** Ionic is usually built with sass. This folder will have the sass files.
- **www:** this is the main folder of the project where 99% of the development work is carried out. Inside this folder we will have the below sub folders.
 - **css:** files with CSS styling rules are written inside this folder
 - **img:** this folder is responsible for storing the images of the project
 - **js:** will have all JavaScript files.
 - **lib:** library files are placed inside this folder
 - **templates:** this folder will store the HTML files of the project
 - **index.html:** this HTML file is the starting point of our hybrid application.

Along with folders mentioned above, and *index.html*, there are a few more files that are created by default by Ionic. These are as follows:

- **bowerrc:** this file is the bower configuration file.
- **.editorconfig:** is the file responsible for editor configuration
- **.gitignore:** Some part of the application can be hidden before pushing into Git repository with the help of this file.
- **gulpfile.js:** automated tasks can be created in this file with the help of gulp task manager
- **config.xml:** this is a xml file for Cordova configurations.
- **Package.json:** this JSON file has information about application, plugins, and dependencies that are installed time to time with the help of NPM package manager.
- **package.json** contains information about the application, dependencies and plugins that are installed using NPM package manager.

5.2. Front-End

As discussed earlier in section 5.1 of this chapter, **www** is the main folder, where 99% of work is done on this project. It is the folder which will have our front-end code. Figure 5-2 presents the project folder structure created for our hybrid application for VPS. By going through the use cases and the default angular project structure, below sub folders are created under **www** folder.

The sub folders under **www** are as follows:

- controllers
- css
- img
- services
- views

along with **index.html**, **manifest.json** and **service-worker.js**

In **controller** folder, we have all the angular code required for the views (HTML files) of the application.

The different views are created and stored in **views** folder of our project. Whereas **css** and **img** folders contains the stylesheets and images required for the application.

The **lib** folder is responsible for storing all the plugins which will be utilized.

In the end, the **service** folder is created for service or factory files which will help to get the data via \$http requests.

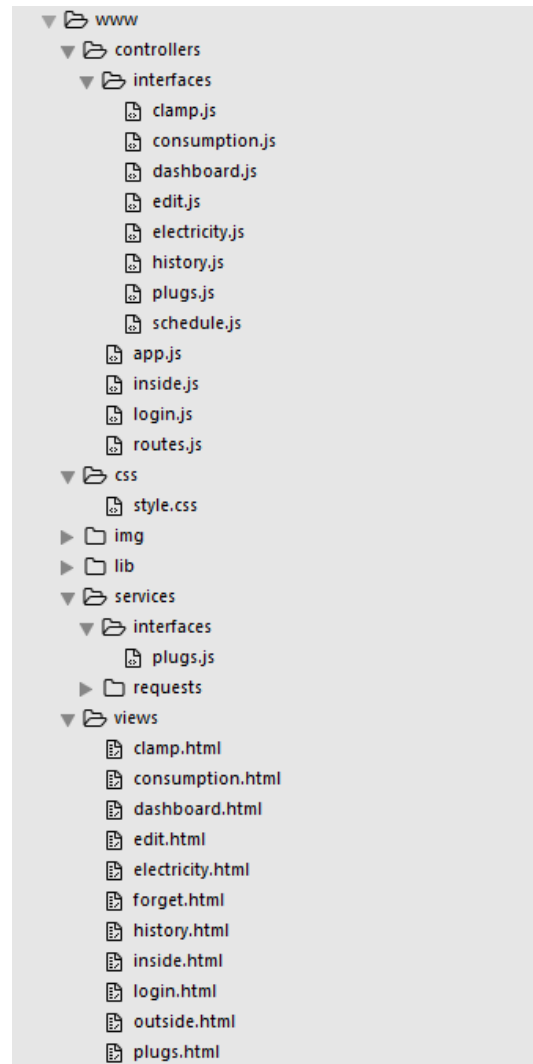


Figure 5-2 Project Folder structure of the front-end of the application for VPS.

In this chapter, we will discuss the developed work by breaking down the application in several parts. To start with, one first aspect of the development is to create our views of the application.

5.2.1. Application Structure

Index

The *index.html* file is the main file of the project which will serve different views and will be the controller above everything else. All JavaScript and CSS files are linked to the project through *index.html* file as shown in the below figure 5-3.

```

35 <!-- your app's js -->
36 <!-- JavaScripts for the plugins -->
37
38 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
39 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="
sha384-Tc5IQib027qvyjSMfHj0MaLkfuWVxZxUPnCA712mCWNIpG99mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>
40 <script src="https://use.fontawesome.com/80d786e3ff.js"></script>
41 <script src="https://cdnjs.cloudflare.com/ajax/libs/angular/storage/0.3.10/ngStorage.min.js"></script>
42 <script src="https://d3js.org/d3.v3.min.js"></script>
43 <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-svg-round-progressbar/0.4.8/roundProgress.min.js"></
script>
44
45 <!-- plugins -->
46 <script src="lib/moment/min/moment.min.js"></script>
47 <script src="lib/chart-plugins/Chart.js"></script>
48 <script src="lib/chart-plugins/angular-chart.min.js"></script>
49 <script src="lib/chart-plugins/angular-ui-router.min.js"></script>
50 <script src="lib/ionic-timepicker/dist/ionic-timepicker.bundle.min.js"></script>
51 <script src="lib/ionic-datepicker/dist/ionic-datepicker.bundle.min.js"></script>
52 <script src="lib/angular-oboe/dist/angular-oboe.min.js"></script>
53 <script src="lib/oboe/dist/oboe-browser.min.js"></script>
54 <script src="lib/ng-oboe-master/dist/ng-oboe.js"></script>
55 <script src="lib/jquery/plugin.js"></script>
56 <script src="lib/jquery/percircle.js"></script>
57
58 <!-- controllers -->
59 <script src="controllers/app.js"></script>
60 <script src="controllers/routes.js"></script>
61 <script src="controllers/login.js"></script>
62 <script src="controllers/inside.js"></script>
63
64 <!-- controller interfaces -->
65 <script src="controllers/interfaces/dashboard.js"></script>
66 <script src="controllers/interfaces/electricity.js"></script>
67 <script src="controllers/interfaces/clamp.js"></script>
68 <script src="controllers/interfaces/plugs.js"></script>

```

Figure 5-3 Snapshot of the index.html file which includes all JavaScript files and other plugin

The *index.html* is also responsible linking Cordova and other useful plugins in the project. There are plugins for rendering charts in the application, improve the UI of the application, and others which are added to the project as seen the figure 5-3 from code line number 38. Some of the plugins use CDN to have a secure path like Bootstrap plugin at code line number 39. The path for the controllers and the services starts from code line number 59.

Outside

The *outside.html* file is the boilerplate view for a navigation stack outside of the logged in area of the application. The code snippet for *outside.html* can be seen in below figure 5-4.

```

1
2 <ion-nav-bar class="bar-positive nav-title-slide-ios7">
3   <ion-nav-back-button class="button-clear">
4     <i class="ion-arrow-left-c"></i>
5   </ion-nav-back-button>
6 </ion-nav-bar>
7
8 <ion-nav-view animation="slide-left-right">
9   <!-- Center content -->
10 </ion-nav-view>

```

Figure 5-4 *outside.html* represents the outside GUI of the application.

Login

The next view of the application is the *login.html*, which holds the input fields according to the requirement along with *Sign In* button and two hyperlinks for *forget password* and *create a new account* as shown in the below figure 5-5.

```

1 <ion-view hide-nav-bar="true" view-title="Cloogy" >
2   <ion-content class="padding">
3     <div class="form-heading">
4        <on-hold="changeAPI()">
5       <script id="popup-template.html" type="text/ng-template">
6         <input type="text" ng-model="storedAPI.url" name="message">
7       </script>
8     </div>
9     <div class="row row-center">
10      <div class="col text-center">
11        <div class="login-list">
12          <div class="list">
13            <label class="item item-input">
14              
15              <input type="text" name="email" placeholder="Email" ng-model="user.Login" required/>
16            </label>
17            <label class="item item-input">
18              
19              <input type="password" name="password" placeholder="Password" ng-model="user.Password"
20                ng-minLength="8" required/>
21            </label>
22          </div>
23          <button class="button button-full button-positive" ng-click="login()">
24            Sign In
25          </button>
26          <button class="button button-positive button-clear button-full" ui-sref="outside.forget"><u>Forget
27            Password</u></button>
28          <button class="button button-dark button-clear button-full" ui-sref="outside.register"><b>I'm new
29            here!</b> <u>Create me a account.</u></button>
30        </div>
31      </div>
32    </div>
33  </ion-content>
34 </ion-view>

```

Figure 5-5 *login.html* file where the login form is designed for the application

Inside

Once the user logs into the application with successful authentication, the next view of the application is presented using *inside.html* file. In this HTML file, the tab UI is created by utilizing Ionics's `<ion-tab>` HTML tags. The tabs that are designed for the application prototype are presented under `<ion-view>` tag as shown in the below figure 5-6.

```

1 <ion-view>
2   <ion-tabs class="tabs-icon-top tabs-stable {{root.hideTabs}}">
3     <ion-tab title="Dashboard" icon-on="dashboard-activated" icon-off="dashboard-deactivated" href="#/inside/dash">
4       <ion-nav-view name="dash-tab"></ion-nav-view>
5     </ion-tab>
6     <ion-tab title="Electricity" icon-on="electricity-activated" icon-off="electricity-deactivated" href="#/inside/electricity">
7       <ion-nav-view name="electricity-tab"></ion-nav-view>
8     </ion-tab>
9     <ion-tab title="Plugs" icon-on="plugs-activated" icon-off="plugs-deactivated" href="#/inside/plugs">
10      <ion-nav-view name="plugs-tab"></ion-nav-view>
11    </ion-tab>
12    <ion-tab title="Settings" icon-on="settings-activated" icon-off="settings-deactivated" href="#/inside/settings">
13      <ion-nav-view name="settings-tab"></ion-nav-view>
14    </ion-tab>
15  </ion-tabs>
16 </ion-view>

```

Figure 5-6 *inside.html* file presenting the tab view of the application.

As discussed in chapter 4, there are four main components of the application which are dashboard, electricity, plugs and settings. According to the requirement by VPS, these four components were created using tab-view for each component as shown in the above figure 5-6 from code line 4 to 18.

Using Slide Box

Once the user logs into the application, four different views should be presented to the user. These views are:

- **Dashboard:** first interaction screen for the user providing summarized information about energy.
- **Electricity:** screen is presenting energy consumption captured by the clamp.
- **Plugs:** screen is presenting energy consumption for each of the devices.
- **Settings:** the last screen of the application presenting the build version along with copyright information of the application.

The mentioned views should be implemented in a slide-box UI according to the requirement of the VPS. To accomplish the task, the *ion-slide-box* directive from Ionic was implemented in our project. It is a simple directive that helps the developer to create slide-box UI in a mobile application. This slide box will contain pages that can be changed by swiping the content screen of the application. The *ion-slide-box* directive has few delegate methods for controlling the UI behaviour. The behaviours are listed in the below table 5-1. To implement the UI design, *<ion-slide-box>* container and *<ion-slide>* tag are placed inside the container view as shown in the below figure 5-8 [36].

Method	Parameters	Type	Details
<code>slide(parameter1, parameter2)</code>	to, speed	number, number	Parameter to represents the index to slide to. speed determines how fast is the change in milliseconds.
<code>enableSlide(parameter1)</code>	shouldEnable	boolean	Used for enabling or disabling sliding.
<code>previous(parameter1)</code>	speed	number	The value in milliseconds the change should take.
<code>stop()</code>	/	/	Used to stop the sliding.
<code>start()</code>	/	/	Used to start the sliding.
<code>currentIndex()</code>	/	number	Returns index of the current slide.
<code>slidesCount()</code>	/	number	Returns total number of the slides.
<code>\$getByHandle(parameter1)</code>	handle	string	Used to connect methods to the particular slide box with the same handle. <code>\$ionicSlideBoxDelegate.\$getByHandle('my-handle').start();</code>

Table 5-1 Delegate Methods for ion-slide-box from Ionic Framework (from [36]).

The *ion-slide-box* directive is injected into the controller which will help to add additional delegation methods to our view. In below figure 5-7, the *\$ionicSlideBoxDelegate* is injected into our application controller to control the UI navigation for the list of devices. With this, a list of devices connected to the Cloogy power plugs are presented in a slide UI design. This UI is implemented keeping the requirements as a priority.

```

3 app.controller('PlugsCtrl', function($scope,$ionicLoading,$http,$ionicSlideBoxDelegate,deviceList,oboe){
4
5     $scope.next = function(){
6         $ionicSlideBoxDelegate.next();
7     };
8     $scope.previous = function(){
9         $ionicSlideBoxDelegate.previous();
10    };
11
12

```

Figure 5-7 Implementation of delegate methods in JavaScript file of plugs controller.

To implement the slide box for the view of the application, the device list is created with the help of *<ion-slide-box>* along with attributes provided by the framework.

```

25 <ion-slide-box delegate-handle="image-viewer" on-slide-changed="slideChanged(index)" auto-play="false"
26 does-continue="true" show-pager="false" style="position:relative; ">
27     <ion-slide ng-repeat="plugModel in plugModels">
28         <div class="device-container">
29             <div class="card device-card">
30                 <div class="item device-avatar">
31                     <div class="row">
32                         <div class="col col-25">
33                             <h2 style="position:absolute;top: 40%; left: 5%;><b><font color="white">
34                                 {{plugModels[$index].plugInfo[0].Name}}</font></b></h2>
35                         </div>
36                         <div class="col col-25">
37                             
40                         </div>
41                         <div class="col col-25">
42                             
44                         </div>
45                         <div class="col col-25">
46                             
48                         </div>
49                     </div>
50                 </div>
51             </div>
52         </div>
53     </ion-slide>
54 </ion-slide-box>

```

Figure 5-8 Implementation of ion-slide-box in the views of the application

Each attribute helps the developer to control the behaviour as discussed above in this section of the chapter. The below table 5-2 adapted from the tutorial blog [36], provides a brief description of each attribute provided by the framework.

Attribute	Type	Details
does-continue	Boolean	Should slide box loop when first or last box is reached.
auto-play	Boolean	Should slide box automatically slide.
slide-interval	number	Time value between auto slide changes in milliseconds. Default value is 4000.
show-pager	Boolean	Should pager be visible.
pager-click	expression	Called when a pager is tapped (if pager is visible). \$index is used to match with different slides.
on-slide-changed	expression	Called when slide is changed. \$index is used to match with different slides.
active-slide	expression	Used as a model to bind the current slide index to.
delegate-handle	string	Used for slide box identification with \$IonicSlideBoxDelegate .

Table 5-2 Attributes from ion-slide-box directive by Ionic Framework (from [36]).

5.2.2. Routes

Applications built using Ionic framework have a feature to track navigation history. This feature helps application to enter and exit views of the application correctly. The transition between the views adopts platform's specific transition style. Each platform has their own UI design and style. Ionic quickly adapt the platform transition style and there is no requirement to change the code base for specific platforms. Since Ionic is based on the Angular ecosystem, it uses Angular UI Router module where application interfaces are organized into various states. Angular UI router has a powerful UI state manager which helps developers to the bound nested and parallel views. This helps to add more than one template on a single page. In this feature, it is not compulsory to bound each state with URL. Additionally, adding more flexibility to routes of the application, data can be pushed to each state with Angular UI router manager [24].

Usage

In this section of the chapter, we will understand how the router works in our project. As discussed in the earlier sections, Ionic comes along with AngularUI router, thus we can directly use *\$stateProvider* and *\$urlRouterProvider* in our *config* method to make use of the AngularUI router.

The *\$stateProvider* object features the state method that allows us to define granular application states that may or may not coincide with changes to the URL. The *\$urlRouterProvider* is an object that gives you control over how the browser's location is managed and observed. In the context of the UI Router, *\$urlRouterProvider* is used to help define a catch-all navigation scenario.

After setting up the workspace, we notice that the first JavaScript file which is created by Ionic by default is the *app.js*. However, it is the best practice to create separate JavaScript files for each view following the MVC architecture by Angular. In our project, the routing of the application is developed under new JavaScript file called *routes.js* under the path *www/controllers/routes.js*. A *config* method is created where *\$stateProvider* and *\$urlRouterProvider* are injected as dependencies as shown in the below figure 5-7.

```
155 .config(function($stateProvider, $urlRouterProvider) {
156
157     $stateProvider
158     .state('outside', {
159         url: '/outside',
160         abstract: true,
161         templateUrl: 'views/outside.html'
162     })
163     .state('outside.login', {
164         url: '/login',
165         templateUrl: 'views/login.html',
166         controller: 'LoginCtrl'
167     })
168 })
169
170     .state('outside.forget', {
171         url: '/forget',
172         templateUrl: 'views/forget.html'
173     })
174 })
175
176     .state('outside.register', {
177         url: '/register',
178         templateUrl: 'views/register.html'
179     })
180 })
181
182     .state('inside', {
183         url: '/',
184         abstract: true,
185         templateUrl: 'views/inside.html',
186         controller: 'InsideCtrl'
187     })
188 })
189
```

Figure 5-9 *config* method where *\$stateProvider* and *\$urlRouterProvider* are injected as dependencies.

The state method of *\$stateProvider* is used to declare the routes. The first argument of the state method is the name of the state and second argument contains the router configuration. This router configuration has URL which renders the template when URL is triggered. Each state of the application is defined by providing a unique name which informs the framework to find its markup for the view. From the code line number 158, present in figure 5-9, we see that the *outside* state is defined by providing the root location of the URL and a value for the *templateUrl* property. Angular UI routing provides a feature called state-centric routing, which helps the application to load the contents of

outside.html into the *ui-view* placeholder whenever the user navigates to the root of the application.

Moving ahead, we observe that in the above code in the figure 5-9, we can see that the first state of the application is the outside which is the navigation stack outside of the logged in area of the application prototype. Therefore, in the project, we have the below main views that were created according to the use cases of the prototype:

- **Outside:** outside of the logged in area of the application
- **Login:** allow users to provide their user credentials for authentication
- **Inside:** inside of the logged in area of the application

Nested States

As discussed in chapter 4, the four main components, dashboard, electricity, plugs, and settings are to be designed as tab views. Ionic helps in this requirement by providing a template. The tab view is designed according to the requirement, and VPS provides the icons used in the tab view. The Ionic template for tab view can be seen in the below figure 5-10.

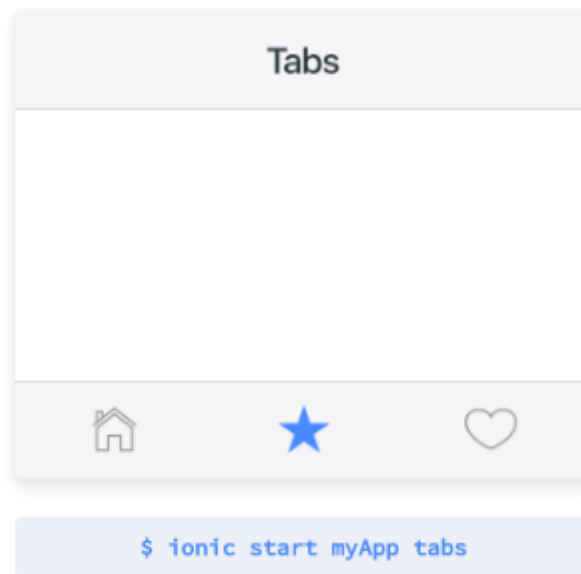


Figure 5-10 Ready-made tabs template from Ionic Framework (adapted from [24])

To have user to navigate through these tabs, nested states from angular routing can be applied to the application so that the user can easily navigate from one view to another. For example, the *inside.electricity* state is set up to load the markup of the *electricity.html* page into the *ui-view* placeholder. The stance includes a dot(.) between inside and electricity to denote a parent and child relationship among the states. Beyond doing a basic operation, *ElectricityController* is also associated to the view scoped at the level of the DOM element as shown in the below figure 5-11 that hosts the *ui-view* directive.

```

79     .state('inside.electricity', {
80       url: 'inside/electricity',
81       views: {
82         'electricity-tab': {
83           templateUrl: 'templates/electricity.html',
84           controller: 'ElectricityCtrl'
85         }
86       }
87     })
88

```

Figure 5-11 Route for electricity screen

To summarize the above-mentioned points, below table 5-3 provides the breakdown of each part of the state represents.

Table 5-3 Nested states brief description in Angular Routing

Object Key	Description
url	The URL route that can be accessed via href properties
templateUrl	The path to view template HTML file
controller	The controller to be used in the view

URL Parameters

Understanding the energy consumption of a residential house is the major outcome of the Cloogy application. To visualize energy consumptions in forms of charts, present current energy consumptions and future energy consumption predictions, it requires a unique technology which will permit to retrieve all important data from a power plug and later to visualize it in the front end of the application.

Monitoring electricity consumption (logs and the real-time data), check consumption indicators (performance, daily average, forecast) and other functionalities are the critical and main functionality of the application. Each hardware associated with Cloogy has its unique ID. These unique identifiers help developer implement specific functionalities to each hardware connected to Cloogy. For example, a Cloogy power plug, which is connected to a residential device will have the following IDs:

- Device ID
- Tag IDs
 - Active Energy Tag ID
 - Actuation State Tag ID
 - Active Power Tag ID

These IDs will help the front-end application to query the server with HTTP requests to get the data. This data in the form of JSON can later be visualized in the front end to carry

out all the functionalities. In Angular, *\$stateParams* is an object having a key URL parameter. This object helps to provide individual parts in navigation URLs to controllers or services [37]. With this feature of *\$stateParams*, our objective to pass unique actuation IDs is accomplished. For example, when the user tries to view the consumption details of water cooler device, it was required to pass the unique actuation ID of this device. This actuation ID received at the controller as a *\$stateParams*, and will be used to query iEnergy3 API to get consumption data for the water cooler.

Basic Parameters

To pass actuation IDs to the views, basic parameter method from Angular UI routing is implemented in our project. URLs can have dynamic parts which are called parameters. Angular provides various options to define necessary parameters [37]. One of them is implemented in our application as shown in the below figure 5-13.

```
238     .state('inside.consumption', {
239       url: 'inside/consumption/:energyId',
240       views: {
241         'plugs-tab': {
242           templateUrl: 'views/consumption.html',
243           controller: 'ConsumptionCtrl'
244         }
245       }
246     })
247
248     .state('inside.schedule', {
249       url: 'inside/schedule/:actuationId',
250       views: {
251         'plugs-tab': {
252           templateUrl: 'views/schedule.html',
253           controller: 'ScheduleCtrl'
254         }
255       }
256     })
257
```

Figure 5-12 StateParameters defined for the consumption module screen.

The defined URL `~/inside/consumption/:energyID``, contains: *energyID* which will capture the energy tag ID for the device. This will be later passed to the controller as an object as a key, which will be captured by *\$stateParams* service in the controller, and later could be used to query the server API.

Using Parameters in Links

The URL link along with parameters are created by state name which acts as a function. Then this URL is passed as an object with parameter names as keys [37]. Executing this action, a proper *href* will be generated. For example, using the above state which has specified: *energyID* as a parameter (in code line number of 239 in below figure 5-12), a link is created as shown in the code line number 35 of figure 5-13 below.

```

34     <div class="col col-25">
35         
36     </div>
37     <div class="col col-25">
38         
39     </div>
40     <div class="col col-25">
41         
42     </div>

```

Figure 5-13 Usage of parameters as links while in a view of an application

After successful routing, parameter value will be captured and passed through the URL. The value for: *energyID* will be the tag ID which will be passed in the URL as shown in the below figure 5-14.

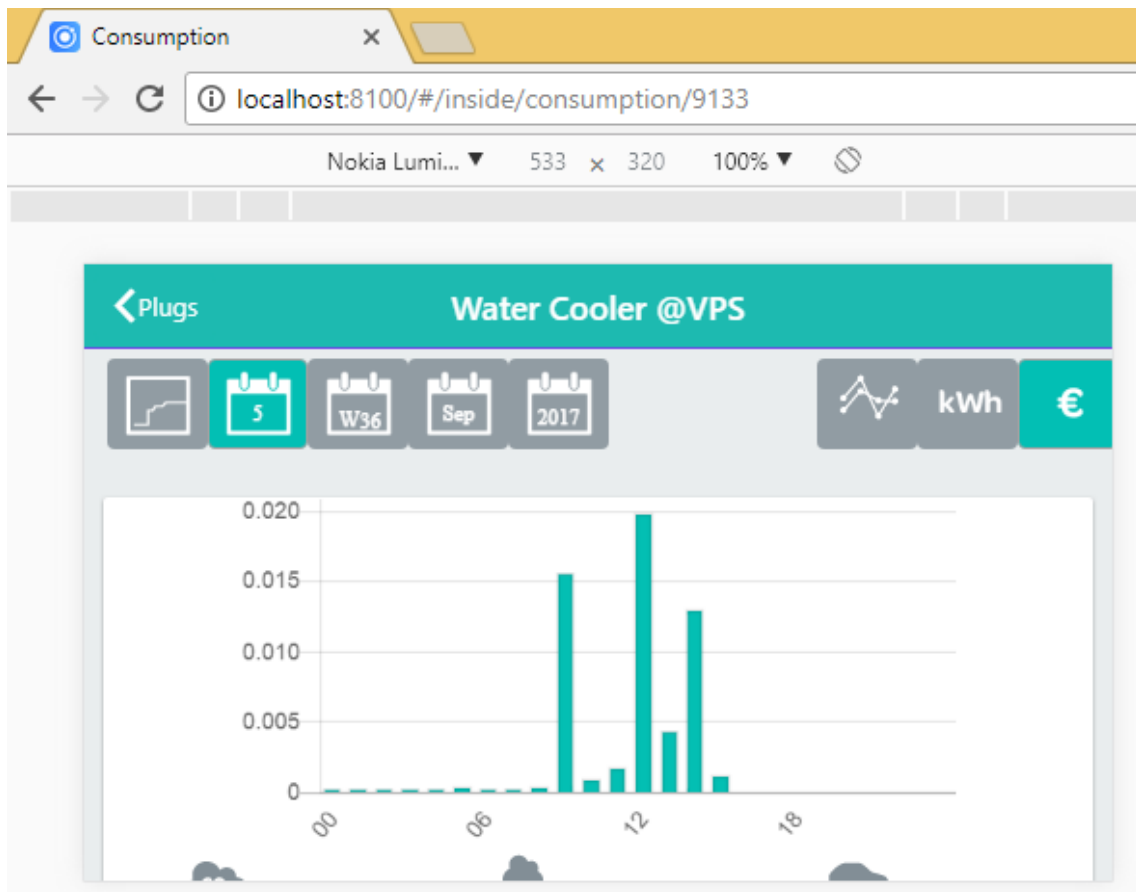


Figure 5-14 Consumption view of water cooler where we can see value passed as a parameter along with URL.

5.2.3. JavaScript Libraries

As discussed in chapter 3 of the thesis, an initial plan was created for the complete development process with the help of agile methodology. This approach promotes adaptive planning with continuous improvements. One of the main ideas behind this approach was to define project tasks starting with the project goal and then to break down into smaller planning chunks which will be called modules. Since we had four major views of the application prototype, the modules were defined as:

- Dashboard Module;
- Electricity Module;
- Plugs Module and;
- Setting Module.

Each of the above defined modules was defined separately and developed in different time interval keeping the functional requirements in place. Each module required specific JavaScript library to design the UI part of the application. These libraries were very helpful to visualize the JSON response into the front-end of the application.

Oboe.js

Dashboard module is the first interface with which the user is going to interact after successfully logging into the prototype. One of the main aspects of this interface was the ability to display active power received from the clamp. As discussed in the second chapter of the thesis, a clamp is installed on the electricity meter of the residential house, which is responsible for gathering and sending data to the transmitter. From there, it is the transmitter responsibility to forward the gathered data to the hub, which later will forward the data to the front-end applications via a router.

The dashboard module of the application will have a doughnut pie chart, which will help to visualize active power data from clamp in real time. To design a doughnut pie chart and implement JavaScript functionalities to render live data, oboe.js [38] JavaScript library was utilised. The HTTP response for active power data from iEnegy3 API is a continuous JSON response that never completes. The response starts as a JSON message chunk having reading data for active power and the response continues to send new active power reading as they happen. This kind of handshake of HTTP protocol is designed by VPS to capture and represent active power to the front-end of Cloogy applications. The implementation method of oboe.js can be summarized in the below figure 5-15, where we can see that a live JSON stream response is sent from the message server to the front-end application.

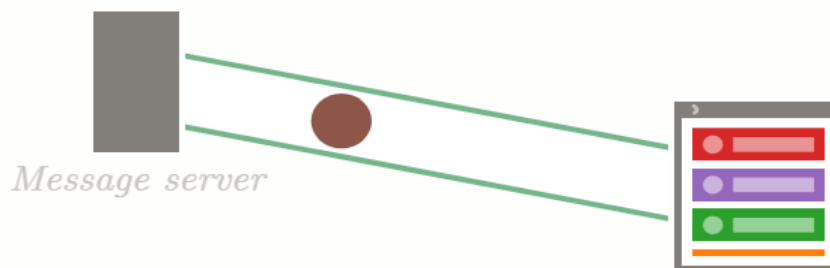


Figure 5-15 Live data transmitted from message server to the front-end (from [39]).

To capture live stream received from the clamp, it was required to utilise this new JavaScript library called oboe.js. This library is an open source JavaScript library which helps to load JSON streams by combining DOM with speed and fluidity of SAX. It can parse any JSON into a stream. This library helps developers to start using JSON response before HTTP response has been completed, or even if it never completes [39]. Hence, in the end, with the help of Oboe.js, the dashboard module was designed and developed, and later successfully deployed in both Android and iOS as seen in the below figure 5-16.

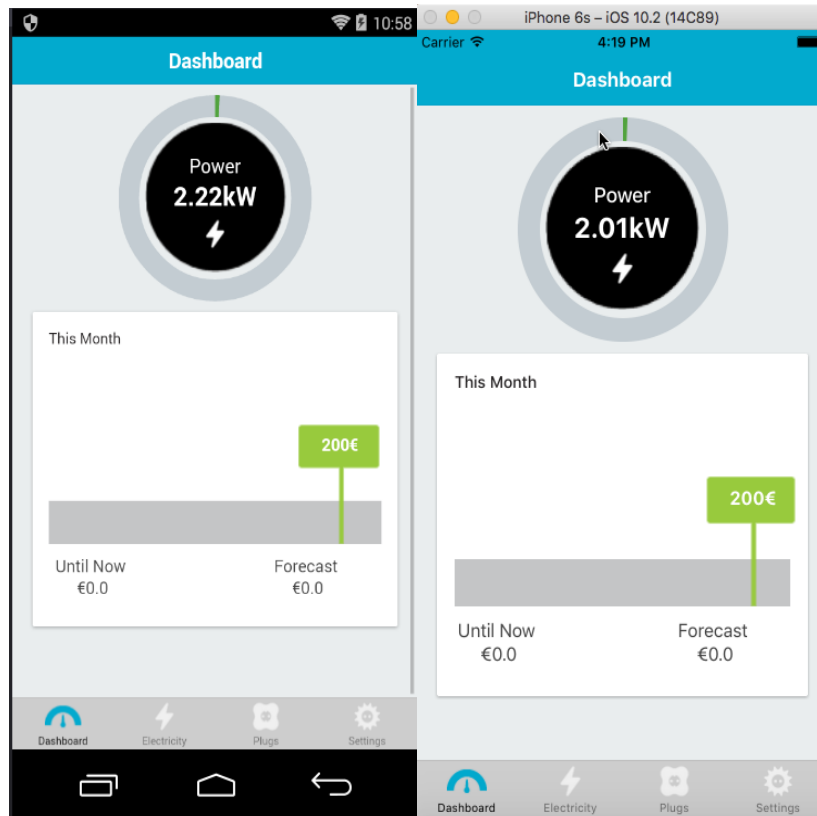


Figure 5-16 Dashboard Module from android and iOS respectively.

Chart.js

Both electricity and plugs module require an interface where the user is presented with consumption data in the form of charts. These charts log real time data in an interval. These interfaces also have consumption indicators to check the performance, daily average, forecast and other operations. For example, to view energy consumption by a residential device like a water cooler, the user can view all the information in the form of charts for daily, weekly, monthly and yearly. The performance of the device can also be compared with historical data with the help of line charts.

To design an interface for charts, an open source chart library for JavaScript named as Chart.js was selected. It is an open source library to visualize, and to animate interactive graphs or charts. In other words, it is a simple and engaging HTML5 JavaScript chart library. This library uses HTML5 canvas element to render charts in the applications. It is a responsive feature which helps to implement this library across systems with different screen sizes. Moreover, to create a chart in an application, the script for the chart.js library should be included in *index.html* which is the route file of the application. Next, by

providing a single *<canvas>* node, the charts are rendered into the views of the application. According to the library documentation, it is used with ES6 modules having plan JavaScript and module loaders [40] [41].

The below figure 5-17, presents a snapshot of the prototype where the user can monitor electricity consumption for one of the devices that is connected to the power plug. At the end part of the charts, there are ecological footprints for carbon, trees and cars.



Figure 5-17 Consumption module of one of the plugs where chart.js has been implemented

In the below figure 5-18, the same functionality of chart.js is implemented in the electricity module where an user can see the consumptions data received from the clamp with consumption indicators, and ecological footprints.

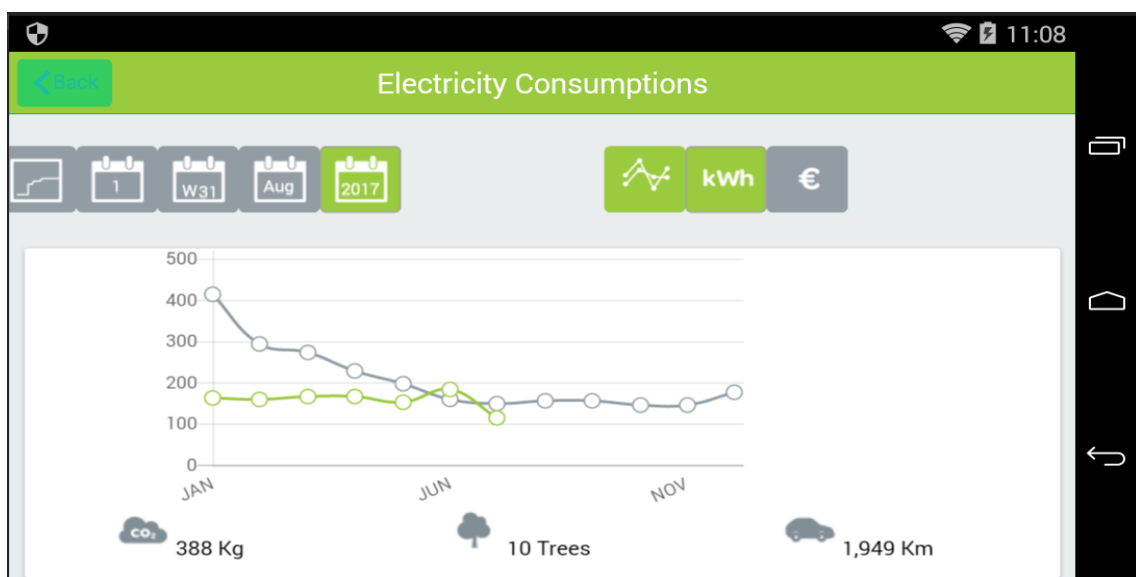


Figure 5-18 Consumption module of electricity module where it has been compared with historic data

Ionic Time and Date Picker

The Cloogy application comes with a feature that permits to schedule operating periods of any electrical equipment that is connected to the power plug. To mark the time interval to turn equipment on and off from a distance, it was required to present a calendar UI to the user. To implement this requirement, ionic-timepicker [42] and ionic-datepicker [43] bower components were implemented in the design. The plugins were installed into the project with the help of bower and the path for these plugins were specified in *index.html* file. Then the dependencies were injected into the schedule module where the controller utilized it.

Additional modifications to the time and date picker can be configured in the methods of *config* file of the plugin. The new configuration based on the requirement of VPS looks as below in the figure 5-19.

```

120 .config(function (ionicTimePickerProvider) {
121   var timePickerObj = {
122     inputTime: (((new Date()).getHours() * 60 * 60) + ((new Date()).getMinutes() * 60)),
123     format: 12,
124     step: 15,
125     setLabel: 'Set',
126     closeLabel: 'Close'
127   };
128   ionicTimePickerProvider.configTimePicker(timePickerObj);
129 })
130
131 .config(function (ionicDatePickerProvider) {
132   var datePickerObj = {
133     inputDate: new Date(),
134     titleLabel: 'Select a Date',
135     setLabel: 'Set',
136     todayLabel: 'Today',
137     closeLabel: 'Close',
138     mondayFirst: false,
139     weeksList: ["S", "M", "T", "W", "T", "F", "S"],
140     monthsList: ["Jan", "Feb", "March", "April", "May", "June", "July", "Aug", "Sept", "Oct", "Nov", "Dec"],
141     templateType: 'popup',
142     from: new Date(2012, 8, 1),
143     to: new Date(2018, 8, 1),
144     showTodayButton: true,
145     dateFormat: 'dd MMMM yyyy',
146     closeOnSelect: false,
147     disableWeekdays: []
148   };
149   ionicDatePickerProvider.configDatePicker(datePickerObj);
150 })
151
152

```

Figure 5-19 Configuration method for ionic time and date picker

5.2.4. AngularJS Filters

As discussed earlier, the Cloogy application allows users to control and schedule operating periods of electrical equipment. According to the requirement, in the schedule module, the user can create, view, edit and delete any schedules for an electrical equipment, as shown in the below figure 5-20.

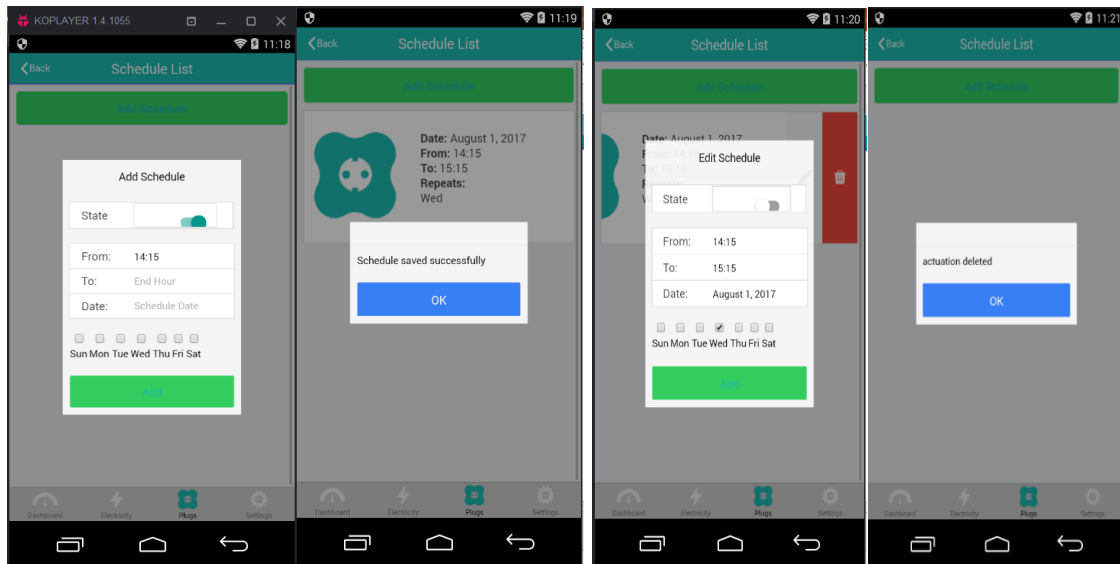


Figure 5-20 Create, view, edit and delete operations snippet from schedule module of the application.

The input fields which are mandatory for the user to add a schedule are the following:

- **State:** Whether user wish to turn on or off the electrical equipment,
- **From & To:** The time interval the user wishes to turn on/off the equipment,
- **Date:** The date the user wishes to perform the future actuation schedule, and
- **Week:** Multiple selection options to select the week days for the actuation schedule.

According to the requirement, the week selection should return the value in binary which should be later converted into decimal. Then this decimal value along with other input field values are utilized to query the API to perform schedule actuations on the server. Similarly, the JSON response to already scheduled actuations of electrical equipment has a decimal number which represents the weeks selected by the user. To visualize this information on the front-end, the application should plot this number against a calendar bitmap which converts the decimal number to binary. Lastly, the binary number is plotted in a week`s table to provide the output for the week days. The bitmap plotting is shown in the below figure 5-21.

Calendar BitMap Plotting								
	64	32	16	8	4	2	1	
WeekBitMap Number From Server	Saturday	Friday	Thursday	Wednesday	Tuesday	Monday	Sunday	((template View))
1	0	0	0	0	0	0	1	Sunday
8	0	0	0	1	0	0	0	Wednesday
62	0	1	1	1	1	1	0	Mon, Tue, Wed, Thu, Fri

Figure 5-21 Calendar week bitmap plotting in schedule module

For example, if the JSON response has a week bitmap number of 62, then the front-end applications should display Monday, Tuesday, Wednesday, Thursday, and Friday on the interface. The decimal number 62 is plotted on the map in a way that this decimal number of 62 is converted into binary which will be 0111110. Next, the desired result will be obtained if we plot the binary number 0111110 into a week calendar map starting from Saturday until Sunday, it will give the desired result. According to the requirement, the 1st bit of binary bitmap should start as Sunday, and ends on the 64th bit of bitmap as Saturday.

To implement the above functionality in angular, custom filters from the framework was utilized. In our prototype, the week bitmap value rendered on *schedule.html* as shown below in figure 5-22, on code line number 63.

```

55     <div class = "row">
56         <div class = "col">
57             <div></div>
58         </div>
59         <div class = "col">
60             <b>Date:</b> {{actuationLog.DateLimitBegin | date : "MMMM d, y" }} <br>
61             <b>From:</b> {{actuationLog.Time | date:'HH:mm' : '+0100'}} <br>
62             <b>To:</b> {{actuationLog.RelatedActuationRule.Time | date:'HH:mm' : '+0100'}} <br>
63             <b>Repeats:</b> {{actuationLog.WeekBitmap | weekBitConverter}}
64         </div>
65     </div>

```

Figure 5-22 Usage of Angular Filter in schedule module of the application.

To create bitmap functionality, custom filter from Angular JS has been implemented in our hybrid application. The steps to create custom filter starts with custom filter name and a function as input parameters to *app.filter()*. By this, *app.filter()* will return a function which can take various optional input parameters. The custom filter code will return the expected output in the returned function. In our hybrid application, a custom name `weekBitConverter` was created for the week bitmap as shown in the above figure 5-23, code line number 63. This filter name was passed as input parameter to the *app.filter()* which can be seen in the below figure 5-23.

```

55     .filter('weekBitConverter', function(){
56         return function(number){
57             if(isNaN(number) || number <1){
58
59                 return number;
60             }else {
61

```

Figure 5-23 Custom application filter for week bit map

Since the *app.filter()* returns the decimal value, our next step was to convert this decimal value into the 64 bit binary number. This can be seen in the below figure 5-24.

```

62     function pad(number,size) {
63         var s = number + "";
64         while(s.length <= size){
65             s = "0" + s;
66         }
67         return s;
68     }
69
70     number = pad(number.toString(2),6);

```

Figure 5-24 Decimal to binary conversion

After conversion, it was required to plot the binary number to get the selected week days. All the above functionalities objectives were obtained with the help of *angular.filter()*. The input parameters are defined in the *app.filter()* as shown in the below figure 5-25 which will implement our bitmap operations in our application.

```

72     var days = [];
73
74     if(number[0] == "1"){
75         //return 'Sat'
76         days.push("Sat");
77     } if(number[1] == "1"){
78         //return 'Fri'
79         days.push("Fri");
80     } if(number[2] == "1"){
81         //return 'Thu'
82         days.push("Thu");
83     } if(number[3] == "1"){
84         //return 'Wed'
85         days.push("Wed");
86     } if(number[4] == "1"){
87         //return 'Tue'
88         days.push("Tue");
89     } if(number[5] == "1"){
90         //return 'Mon'
91         days.push("Mon");
92     } if(number[6] == "1"){
93         //return 'Sun'
94         days.push("Sun");
95     }
96
97     var newDays = days.toString();
98     return newDays.replace(/^\s+|\s+$/g, '');
99
100     if(number){
101         return newDays
102     }
103

```

Figure 5-25 Custom filter code for the week bit map using angular.filter().

5.3. User Interfaces

The user interface starts with the login page asking for valid API URL along with valid username and password. Once the user has obtained the required authentication credentials by registration on the Cloogy application, the user initiates the usage of the prototype by starting with providing the API URL.

According to the given requirement, the input field to fill up the API URL is designed in such a way that the user must long press the Cloogy icon designed at the center of the login page. Once long pressed, it will have a popup input field to provide the API URL or to change the existing URL. The slow-motion pop up cannot be captured in an image.

However, in the below figure 5-26, we can see the popup hovering at the top of the login page, while the backside view has been darkened with the styling.

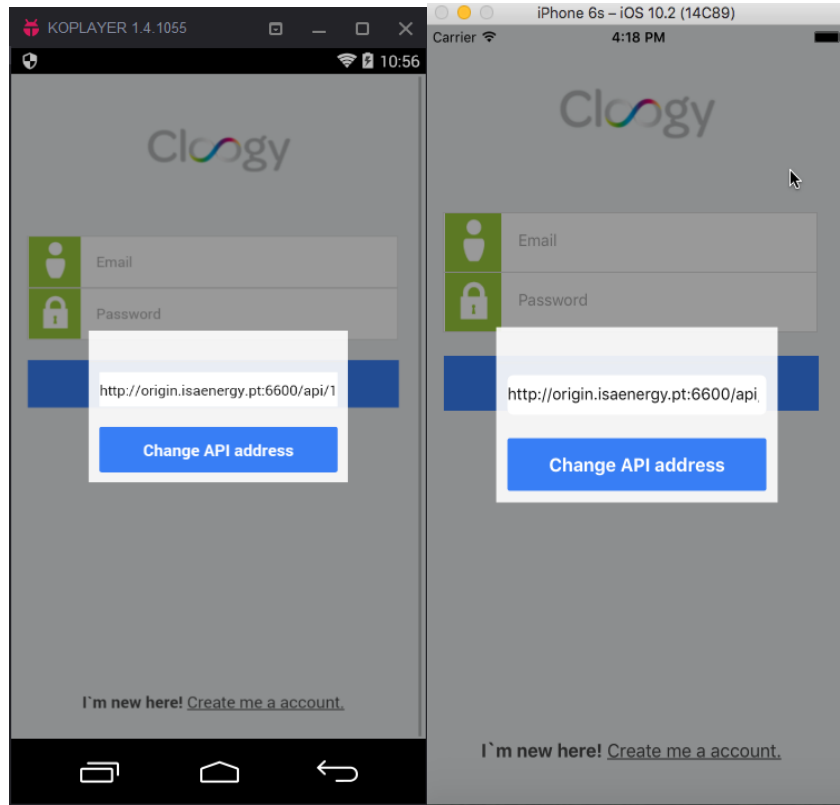


Figure 5-26 Snapshot of the application prototype asking for change API address in both OS (left: Android, right: iOS).

Once the user has successfully logged into the application, the first screen which will be presented to the user will be the dashboard, as shown in the below figure 5-27. The dashboard is responsible for providing an overview of monthly consumption in terms of currency visualized in a progress bar at the bottom of the application. It also allows the user to get a monthly goal for the energy consumption. At the top of the dashboard, the user can see doughnut pie chart progress bar which streams live active power from the clamp in terms of a kilowatt.

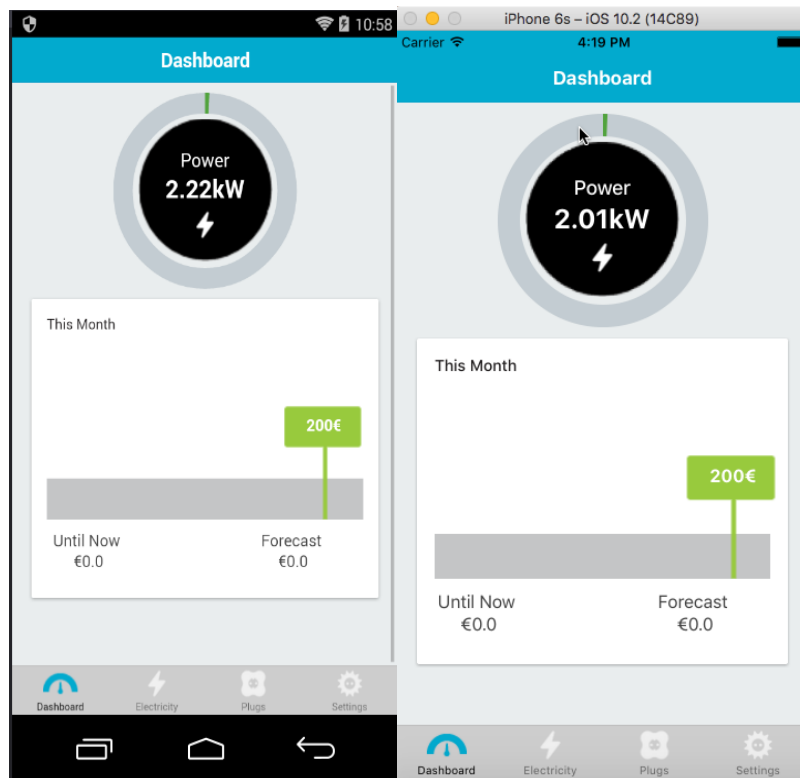


Figure 5-27 Dashboard module from the application prototype (left: Android, right: iOS).

As discussed earlier in this chapter, there are four main views of the applications which are dashboard, electricity, plugs, and settings. Moving forward, the user can switch to the electricity tab, where brief electricity consumptions for a day, week, month and year will be presented along with current and forecast values. The ecological footprints are also presented at the bottom of the card as shown in the below figure 5-28.

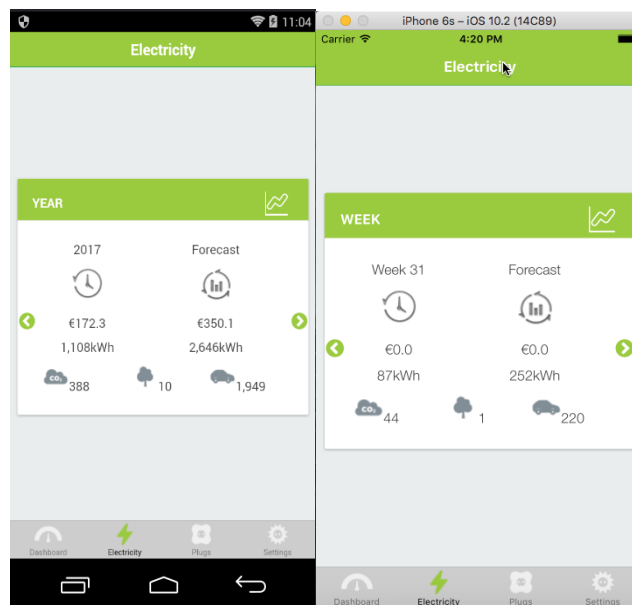


Figure 5-28 Electricity module from the application prototype (left: Android, right: iOS).

Both electricity and plugs tabs are designed using `<ion-slide-box>` directive from Ionic according to the requirement and the UI is designed using CSS property cards [44] from the Ionic framework.

As discussed in the section 4.6.3, Ionic provides a good number of UI design templates. These templates make developer's work little easier. Card template from Ionic is widely used in recent years. Cards in the mobile application helps developers to contain and organize information. There will be always be a good amount of content that developers intend to display at once, and often the screen size will be small enough to have contents. In this scenario, cards are helpful UI designs to customize the contents in one place. There are big companies like Google, Twitter, and Spotify which utilize cards as their design patterns. Cards helps to fit the same amount of information across all types of devices with different screen sizes. Cards provides useful features like animation, more control over the native functionalities, and are flexible in nature. In our hybrid prototype, cards are placed in a slide-box, where users should swipe the cards from left to right, or vice versa [44].

Since our requirement was to build a cross platform application, it was expected that there might be challenges making the user interface responsive across all platforms. There were also little deviations seen in different versions of the Android, where the UI does not follow CSS properties in older versions of Android older than 4.0. To make the UI more responsive, it was imperative to use CSS Flexbox property which is the new layout mode in CSS3. The usage of flexbox helps HTML elements to adapt if there is any change in the screen size. Flexbox accommodates the page layout according to the different screen sizes of different devices. The flex container's margins collapse with margins of its contents when there is any change is the page layout of the device. However, to start implementing the technology, the developer should start designing the UI by containers. To implement flexbox in our design, it is imperative to start flexbox properties with a flex container and its items. The flex container is declared having the *display* property of an element. The *display* property can be set to *flex* or *inline-flex*. After the container, the next task will be to define the properties of the items inside container [45] [46]. In the below figure 5-29, the *justify-content* property at the code line number 261, horizontally aligns the flexible container's items when the items do not use all the available space on the main-axis of the mobile device. Similarly, the other property *align-items* at the code line number 260, vertically aligns the flexible container's item when the items designed do not use all the available space on the cross-axis of the mobile device.

```
256 .device-container{
257     display: flex!important ;
258     height: 100% !important;
259     width: 100% !important;
260     align-items: center;
261     justify-content: center;
262
263 }
264
```

Figure 5-29 Code snippet of implementation of flex container property of CSS3

The below figure 5-30 provides a summary of the functionalities that the prototype offers in plugs module. Plugs modules display the list of devices that are connected to the Cloogy power plugs in a residential house. With the help of this prototype, the user can

turn on or off the power of the device from a remote location along with creating future actuation schedules, viewing the actuation history log.

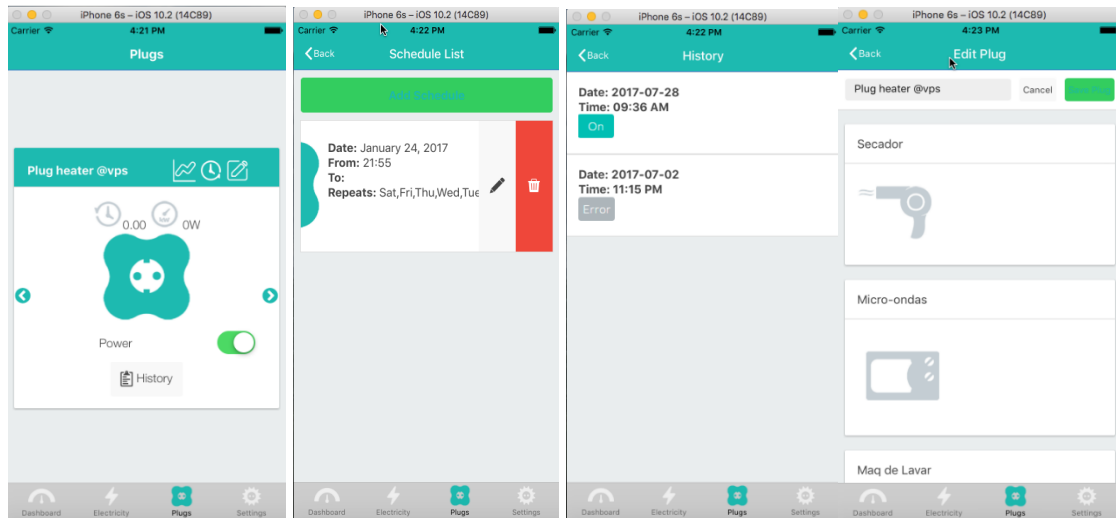


Figure 5-30 Different functionalities snapshot of plugs module from iOS

In plugs module, the user can view plug history, view energy consumption, schedule actuation operations, and change the name of the power plug along with the icon as seen in the above figure 5-30. The energy consumption for plugs will have indicators providing performance report in bar/line charts as shown in the below figure 5-31.



Figure 5-31 Snapshot from iOS where Consumption module for electricity and plugs is presented respectively.

6. Conclusions

This chapter discusses the conclusion of the thesis, focusing on strengths and limitations. The chapter is about the outcome of the thesis objectives and the results are analyzed in detail.

This chapter starts Section 6.1 with project final results. Section 6.2 will describe the strengths of the work, stressing what sets it apart from the other approaches and demonstrating how the initial problem is currently solved. Section 6.3 discusses few problems encountered during the development phase and at the end, section 6.4 provides a brief overview of improvements required in the project by providing a proposal for what should be the following steps for the project and what should be achieved in the future.

6.1. Results

Keeping the requirement of the project as top priority, the technical and implementation of the prototype has proven to achieve satisfactory results. With minor glitches at the end of the deployment, a successful prototype for Cloogy was built and deployed in iOS and Android.

The objectives defined by VPS at the beginning of the internship were accomplished. The first objective of this thesis was to study and research about the best possible solution for the cross-platform Cloogy application. To accomplish this task, different cross-platform development tools were analyzed by going through the framework documentations, user and developer experiences, online community, ease of access to the online components, plug-ins and support for the development. Simple prototypes were built to test the framework. The frameworks which were tested with simple prototypes were Meteor, React Native, Xamarin with Visual Studio 2015, Appcelerator Titanium studio and at the end, Ionic with Apache Cordova. The practical test results of the prototype were shared and discussed with VPS. Based on the test results, the Ionic framework was selected.

The second objective was to a develop cross-platform Cloogy prototype with single code-base which was accomplished using the Ionic framework. Finally, the last objective was to maintain the similar GUI in Android, and iOS. This objective was accomplished by successfully built the application to Android, and iOS. The built application was successfully deployed in both emulators and real devices. There was pitfall while deploying into Windows Phone. This will be discussed in the later section 6.3 of this chapter.

6.2. Strengths

Following the rapid growth in the field of mobile application development, there has been the urgency of having the best feasible development option for mobile application in a quick time and keeping the budget. Today, each organization is looking for moving their solutions into mobile applications. It has become a `must have` element for every organization, regardless of its size. It is crucial that the mobile application supports across all platforms. With limited time and budget, Ionic turns out to be a very good choice.

AngularJS is the backbone of Ionic. Therefore, to build a robust hybrid application with Ionic, it requires proper in-depth knowledge of AngularJS framework. For developers who are good with Angular, will be comfortable creating fully cross-platform hybrid applications. The code will be single code-base which will run across all platforms.

This cross-platform prototype of Cloogy was developed using the first version of Ionic framework along with AngularJS 1.4 and Apache Cordova to provide the native part of the application. When we think of building a cross-platform application, we face challenges regarding lack of mobile UI set of components, due to the singularities of each platforms whether it is iOS or Android or even if it is to be deployed in Windows phone. For instance, Android devices have tabs designed at the top of the screen. However, devices running on iOS, the tabs are placed in the opposite way. This is one of the example where Ionic counter this problem solution by providing single code base. There is no requirement to tweak the code base for platform specific UI design. The framework adapts by itself when deployed in the specific platform. Other frameworks do lack in these little features. Ionic in the background, takes care of all the little differences that platforms have between each other. Alternatively, all other frameworks do not bring such full front-end solutions when compared with Ionic.

Considering all the requirements of the enterprise starting from keeping the Cloogy UI in place, cost-efficiency, flexibility and one of the most important code reusability, Ionic fitted the choice. When tested with other well-known cross platform frameworks like Xamarin and Titanium, Ionic emerged as a winner when compared with open source libraries, using platform specific APIs, usage of heavy graphics, application loading time etc. Considering other factors like type and complexity, applications built on Xamarin and Titanium are typically more substantial than the native ones. One of the major factor for selecting the Ionic, was that it is a completely free and open source and it has beautiful UI that is very easy to customize for a developer who has robust web development skill set. These functionalities easy integrate with native functionalities with the aid of Cordova. As discussed in Section 4.6.3, Ionic is an open sourced framework which is licensed under MIT. This license grant permission, free of charge, to any developer to obtain the copy of the software without any restrictions [35] [24]. Keeping frameworks as open source helps to bring thousands of developers to work on the software which not only improves the framework, but also fixes minor challenges faced by the framework. Ionic has a simple approach to the development process. The CLI of the framework helps developers to perform application related tasks like running, debugging, and packaging the application with simpler commands. The team behind Ionic is closely working on building its own IDE which will help developers to create applications by visually dragging and dropping native functionalities [24].

There are times in the development process, when a smallest coding error can take whole day fixing it. This situation may arise when there isn't sufficient documentation or forums around the internet about the framework where developer look for help. This kind of situation was very rare with Ionic as it provides very detailed developer friendly documentation with tutorials across the internet to get started. Ionic also hosts different developer forums like in Stack Overflow, Ionic forums, Slack [47] where all the coders around the world gather in one place just to fix one bug on their codes. Cordova itself has a great community for developers and plugins which helps the coders in great extend to access native functionalities of platform on which they wish to run the application. Along with Cordova, Angular itself is support By Google and is also considered to be most popular single page application framework.

Setting up the development environment is one of the tedious tasks, but in the case of Ionic, its CLI allows developers to set up and test on any platform with quick, and simple commands. About the UI design which is one of the important factors in this thesis, Ionic is helpful as it is pre-loaded with easy-to-customize UI components, where most of the mobile application`s elementary things are included in CSS and JS components. With this feature, it consumes less time, resource and effort when compared to the other frameworks.

6.3. Limitations

Though there are new tools available in the development world for creating native like cross platform applications, the current state of having perfect cross platform application is far from complete. Since user interface(UI) and the experience design (UX) of iOS and Android are entirely different from each other, it is not an easy task to create an uniform GUI wrapper on top of it. If there is a requirement with does not fit with the framework vision of development, then it requires much effort to implement and write specific code.

By analyzing the limitations encountered during the development of the cross-platform application using Ionic, we notice that some of the JavaScript libraries do not support Android versions which are lower than 4.0. This limitation is for the phones which runs older versions of browsers, because they cannot support new JavaScript libraries. As already mentioned in Section 2.2.3, Cordova packages the web application in a device installer format and runs the application in the *WebView* of the phone`s browser. The *WebView* runs in a full-screen mode inside the native container of the device. Therefore, it is imperative to have the browser updated so that new libraries of web technologies can be implemented in the hybrid application. Ionic was always focused on building for modern web standards and modern mobile devices. Thus, for Android, Ionic supports from Android 4.1 and up, and for iOS, it supports iOS7 and up.

There is a workaround to fix the above encountered limitation by adding an additional plugin called crosswalk [48]. It is a plugin by Cordova where the application starts using crosswalk *WebView* instead of mobile device`s *WebView*. However, there are chances of increase in the memory footprint, and increase in the size of the application which is not advisable.

Since Ionic also supports windows phone, the *appxupload* file was built successfully for the window. However, the application was not deployed on both emulator and real device due to security restrictions by Microsoft. While deploying the application, it gives the below runtime error

“0x800c001c - JavaScript runtime error: Unable to add dynamic content”

It is a security feature that Windows runtime gives us if we dynamically try to inject content into our HTML. The run time error informs that third party libraries are not allowed to be executed in windows phone environment. It is being restricted by Microsoft to enhance the security feature of the windows phone operating system. This security feature blocks properties like *innerHTML* and *outerHTML* which may lead to unsafe handling of untrusted data. This kind of blockade are categorized under common security issues by Microsoft. This limitation has a workaround by adding a JavaScript file called

winstore-jscompat.js [49] by Microsoft Open Technologies. This JavaScript library allows to run code lines are flagged as unsafe by Microsoft.

There are security rules in place where it blocks unwanted access to window runtime whenever the application has run a malicious script. It has been observed that Windows store applications developed using AngularJS in Visual Studio IDE returns runtime errors. This runtime error informs developers that it cannot add inject dynamic contents which might be unsafe. To unblock these runtime errors, Microsoft Open Technologies (MS Open Tech) has released the JavaScript Dynamic Content for applications developed for Windows phone environment. It helps to relax the security checks performed by the Windows security model with a minor impact on the performance of the application. The performance of the application depends upon specific usage, time, and frequency. However, it does achieve the fundamental goals set by Microsoft Windows phone development environment [49].

6.4. Future Work

Finally, this thesis is intended to create a cross platform for Cloogy using Ionic framework. However, it does not rule out the possibility of creating a hybrid application of Cloogy with other cross-platform frameworks available. Ionic was chosen considering the time, and budget factors. Furthermore, other frameworks may provide other fruitful results with good native functionalities in the hybrid frameworks.

Nevertheless, taking Ionic as our cross-platform development framework, there are specific areas where there is room for improvement starting with UI design of the application. The cross-platform prototype of Cloogy was developed independently without any references from the existing Cloogy code base for Android, iOS and Windows phone. UI design of any mobile application developed using native or hybrid approach will always have some scope of improvement. Therefore, the UI design of Cloogy hybrid prototype can be improved if well-paid libraries are utilized as they were used in the native approach of Cloogy.

Registering new user and generating new password functionalities from the existing Cloogy application was not implemented as this is only a prototype for the enterprise to examine the possibility of having a cross-platform application.

There can be minor bugs when the application is deployed in real device especially with the JavaScript Libraries. According to VPS, the existing native functionalities uses paid libraries like Kendo UI or d3.js, whereas in this prototype, free, open-source libraries like chart.js, angular charts have been utilized for some of the modules where energy consumptions are visualized in form of charts. Hence, if enterprise agrees to move forward with Ionic to develop the cross-platform application, it would be recommended to use the paid versions libraries which enterprise is already using for their other software solutions.

Finally, the prototype code base cannot be termed as an optimal code base for a cross-platform since this is being developed with minimal experience of web technologies programming languages. There is always a scope of improvement in the future. The current code base was developed to have a prototype which will help the enterprise to examine the possibility of implementing Cloogy as a hybrid cross-platform application.

Bibliography

- [1] R. Patil, "Pros and Cons of Cross-Platform Mobile Application Development," InfoQ, 13 Sep 2016. [Online]. Available: <https://www.infoq.com/articles/mobile-cross-platform-app-development>.
- [2] M. P. C. Ketan Anant More, "Native Vs Hybrid Apps," *International Journal of Current Trends in Engineering & Research*, vol. 2, June 2016.
- [3] W. Jobe, "Native vs Mobile Web Apps," *International Journal of Interactive Mobile Technologies*, vol. 7, no. 4, p. 6, November 2013.
- [4] Cloogy, "CLOogy - Smart Living," Virtual Power Solutions, [Online]. Available: <https://www.cloogy.pt/en>.
- [5] J. Wiken, "Figure 1 - Native apps, mobile websites, and hybrid app architectures compared side by side.," 15 April 2015. [Online]. Available: <https://dzone.com/articles/three-types-mobile-experiences>.
- [6] J. Wilken, "The Three Types of Mobile Experiences," 15 April 2015. [Online]. Available: <https://dzone.com/articles/three-types-mobile-experiences>.
- [7] MRC - Michaels, Rose & Cole Ltd, "Why businesses should think twice before building native mobile applications," *Native Mobile Applications : The wrong choice for business?*, pp. 1-11.
- [8] Nielsen Norman Group, *Native, Web and Hybrid Apps*, 2013.
- [9] P. Cho, "The Benefits of Native vs. Hybrid Mobile Apps," 3 August 2015. [Online]. Available: <https://www.phase2technology.com/blog/the-benefits-of-native-vs-hybrid-mobile-apps/>.
- [10] F. M. K. & J. Jahid, "Comparing native and Hybrid Applications with focus on Features," Faculty of Computing, Blekinge Institute of Technology, Sweden.
- [11] WaveMaker, "What is cross-platform mobile application development?," [Online]. Available: <http://www.wavemaker.com/cross-platform-mobile-app-development-tool/>.
- [12] I. S. A. C. Manuel Palmeri, "Comparison of Cross-Platform Mobile Development Tools," *16th International Conference on Intelligence in Next Generation Networks*, no. 12, pp. 179-186, 2012.
- [13] S. Amatya, "Cross-Platform Mobile Development," Department of Computer Science, Linnaeus University, Sweden, 2013.
- [14] C. Infotech, "PhoneGap or Titanium or Xamarin - Which Cross-Platform Framework Should You Choose?," [Online]. Available: <http://www.cygnnet-infotech.com/blog/phonegap-or-titanium-or-xamarin-which-cross-platform-should-you-choose>.
- [15] M. Rajput, "Why You Should Choose Ionic Framework To Build Your Next Mobile App?," 17 October 2016. [Online]. Available: <http://www.mindinventory.com/blog/why-you-should-choose-ionic-framework-to-build-your-next-mobile-app/>.
- [16] Wikipedia, "Apache Cordova," [Online]. Available: https://en.wikipedia.org/wiki/Apache_Cordova. [Accessed 20 October 2017].
- [17] E. V. Alireza Pazirandeh, "Evaluation of Cross-Platform Tools for Mobile Development," Gothenburg, 2013.

- [18] imgZine, "On its Titanium-powered app platform, imgZine lets publishers and corporations build white-label Real Time Social Magazines that can increase readership by 400%," *Case Study : imgZine*, pp. 1-6.
- [19] Appcelerator Titanium, "Axway Appcelerator Blog," [Online]. Available: <https://www.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap/>. [Accessed 29 October 2017].
- [20] Microsoft, "Xamarin Platform," [Online]. Available: <https://www.xamarin.com/platform>. [Accessed 29 October 2017].
- [21] estaun.net, "Some thoughts after (almost) a year of real Xamarin use," 4 April 2014. [Online]. Available: <http://www.estaun.net/blog/some-thoughts-after-almost-a-year-of-real-xamarin-use/>.
- [22] altexsoft, "The Good and The Bad of Xamarin Mobile Development," 26 June 2017. [Online]. Available: <https://www.altexsoft.com/blog/mobile/the-good-and-the-bad-of-xamarin-mobile-development/>.
- [23] StackOverflow, "Is Xamarin free in Visual Studio 2015?," [Online]. Available: <https://stackoverflow.com/questions/30313302/is-xamarin-free-in-visual-studio-2015>. [Accessed 29 October 2017].
- [24] "Ionic Framework," [Online]. Available: <http://ionicframework.com/>.
- [25] L. P. Richardson, "Xamarin vs Ionic: A Mobile, Cross Platform, Shootout," 28 March 2017. [Online]. Available: <https://www.codeproject.com/Articles/1079101/Xamarin-vs-Ionic-A-Mobile-Cross-Platform-Shootout>.
- [26] stackshare, "Stackshare: Ionic vs. PhoneGap vs. Xamarin," 03 March 2017. [Online]. Available: <https://stackshare.io/stackups/xamarin-vs-phonegap-vs-ionic>.
- [27] A. Makar, "Top-Down vs. Bottom-Up Project Management Strategies," 27 January 2015. [Online]. Available: <https://www.liquidplanner.com/blog/how-long-is-that-going-to-take-top-down-vs-bottom-up-strategies/>.
- [28] H. Kniberg, *Scrum and XP from the Trenches*, InfoQ Enterprise Software Development Series, 2007, p. 142.
- [29] B. A. N. E. Y. J. M. Lawrence Chung, *Non-Functional Requirements in Software Engineering*, vol. 5, Springer US, 2000.
- [30] A. Rodriguez, "RESTful Web services: The basics," 06 November 2008. [Online]. Available: <https://www.ibm.com/developerworks/library/ws-restful/index.html>.
- [31] Virtual Power Solutions, "iEnergy3 - API Data Provisioning," Virtual Power Solutions, Coimbra, 2015.
- [32] guru99, "RESTful Web Services," [Online]. Available: <https://www.guru99.com/restful-web-services.html>.
- [33] W. Cheung, "Angular Single-Page Applications - Cool Privilege Control System," 29 March 2016. [Online]. Available: <https://www.codeproject.com/articles/1088570/angular-single-page-applications-cool-privilege-co>.
- [34] A. G. H. Arush Gupta, "Hybrid Application Development using Ionic Framework & AngularJS," *International Journal of Innovative Research in Computer Science & Technology (IJIRCST)*, vol. IV, no. 2, pp. 62-64, 2016.

- [35] Open Source Initiative, "The MIT License," [Online]. Available: <https://opensource.org/licenses/MIT>. [Accessed 20 October 2017].
- [36] TutorialPoints, "Ionic - Environment Setup," [Online]. Available: https://www.tutorialspoint.com/ionic/ionic_environment_setup.htm.
- [37] G. -. A. J. -. A. UI, "URL Routing," [Online]. Available: <https://github.com/angular-ui/ui-router/wiki/URL-Routing#url-parameters>.
- [38] J. Higson, "Oboe.js - GitHub," [Online]. Available: <https://github.com/jimhigson/oboe.js>.
- [39] J. Higson, "Why Oboe.js?," [Online]. Available: <http://oboejs.com/why>.
- [40] "Chart JS - GitHub," [Online]. Available: <https://github.com/chartjs>.
- [41] "Chart.js Official Documentation," [Online]. Available: <http://www.chartjs.org/docs/latest/getting-started/usage.html>.
- [42] R. Patlolla, "Ionic Timepicker," [Online]. Available: <https://github.com/rajeshwarpatlolla/ionic-timepicker>.
- [43] R. Patlolla, "Ionic Datepicker," [Online]. Available: <https://github.com/rajeshwarpatlolla/ionic-datepicker>.
- [44] I. Framework, "Version 1 CSS layouts of Ionic Framework," [Online]. Available: <http://ionicframework.com/docs/v1/components/#cards>.
- [45] C. Coyier, "A Complete Guide to Flexbox," 20 July 2017. [Online]. Available: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>.
- [46] w3schools, "CSS3 Flexible Box," [Online]. Available: https://www.w3schools.com/css/css3_flexbox.asp.
- [47] Slack, "Team Messaging Group for Ionic Developers," [Online]. Available: www.slack.com.
- [48] Cordova, "cordova-plugin-crosswalk-webview," [Online]. Available: <https://github.com/crosswalk-project/cordova-plugin-crosswalk-webview>.
- [49] Microsoft, "JavaScript Dynamic Content shim for Windows Store apps," [Online]. Available: <https://github.com/Microsoft/winstore-jscompat>.
- [50] LinkedIn, "Virtual Power Solutions," 2015. [Online]. Available: <https://www.linkedin.com/company/virtual-power-solutions>.
- [51] Anesco, "Anesco partners with Virtual Power Solutions to offer Smart Microgrid Solutions," 4 June 2015. [Online]. Available: <http://anesco.co.uk/anesco-partners-with-virtual-power-solutions-to-offer-smart-microgrid-solutions/>.
- [52] vividus, "Benefits of Mobile Apps," [Online]. Available: <http://vividus.com.au/insights/benefits-of-mobile-apps/>.
- [53] Tony, "What are the advantages of hybrid apps for your enterprise," [Online]. Available: <https://fliplet.com/blog/advantages-of-hybrid-apps/#>.
- [54] Wikipedia, "Non-functional requirement," [Online]. Available: https://en.wikipedia.org/wiki/Non-functional_requirement.

- [55] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," *UNIVERSITY OF CALIFORNIA, IRVINE*, 2000.
- [56] "w3schools," [Online]. Available: <https://www.w3schools.com/>.
- [57] AngularJS, "AngularJS API Docs," [Online]. Available: <https://docs.angularjs.org/api>.
- [58] Prat, "codecondo.com," 10 May 2016. [Online]. Available: <http://codecondo.com/9-cross-platform-mobile-development-tools-technologies-and-platforms/>.
- [59] J. Bristowe, "What is a Hybrid Mobile App?," 25 March 2015. [Online]. Available: <http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/>.

Appendix A: Developers Manual

Installation

This Cloogy cross-platform prototype is built using Ionic Framework. To install and run Ionic on your machine, there are few prerequisites that you should have in your machine (system) before you start installing Ionic Framework. The prerequisites are:

- **NodeJS:** This is the base platform needed to create mobile apps using Ionic.
- **Android SDK:** If you wish to deploy and test the application android, then you should have Android SDK setup on your machine.
- **Xcode:** If the user wishes to deploy and test this app in iOS, then you should have XCode setup on your machine.

After successful installation of the prerequisites, then we can move forward and start installing Ionic along with Cordova. It is required to have the below-mentioned versions for installations without any runtime error:

Cordova CLI: 7.0.1
Gulp version: CLI version 3.9.1
Gulp local:
Ionic CLI Version: 2.1.4
Ionic App Lib Version: 2.1.2
OS: Windows 8.1
Node Version: v7.0.0

Open the command window to install Cordova and Ionic by providing the below command

➤ `npm install -g cordova ionic`

Once we have installed Cordova along with Ionic, we can start developing new application by utilizing ready-made templates from Ionic using the command in the below figure A-1.

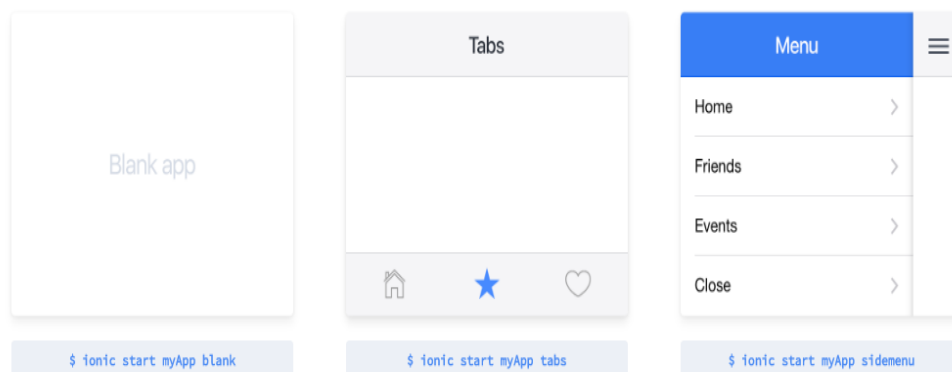


Figure A-1 Ionic Commands to install default templates

If the user wishes to run the existing ionic project, then you just need to give the below commands to run the application

- cd projectName
- ionic serve

After the user has provided the ionic serve command, the project will be hosted your localhost with port 8100.

It is recommended to install Ionic CLI version 2.1.4 to access the below commands. Newer version of CLI may have a different set of commands. If the user wish to deploy the application for android and iOS, then please follow the below commands:

iOS

- cd projectName
- ionic platform add ios
- ionic build ios
- ionic emulate ios [to run on emulator]
- ionic run ios [to run on real device]

Android

- cd projectName
- ionic platform add android
- ionic build android
- ionic emulate ios [to run on emulator]
- ionic run android [to run on real device]

Appendix B: User Manual

Introduction

The Cloogy cross-platform prototype mobile application is developed to help us to understand the feasibility of having a hybrid application with a single code base that can be deployed across all platforms of mobile OS.

The first version of the application can successfully be deployed in iOS and Android. After installing the Cloogy application, the user will be able to:

- View active power value on the dashboard
- Set monthly goals
- View electricity consumptions for a day, week, month and year which can also be viewed in form of bar charts.
- Compare the electricity consumptions in the form of line charts
- View device lists which are connected to the power plugs
- View consumption, perform actuation schedules, view history and edit the power plugs information.

Login

Before using this prototype, the user should have a valid username and password for Cloogy along with the API URL.

Once you have obtained the above required information, the user should follow the below steps.

- Long press on the Cloogy icon on the login page of the application to change the API address [if required] as shown in the below figure B-1.

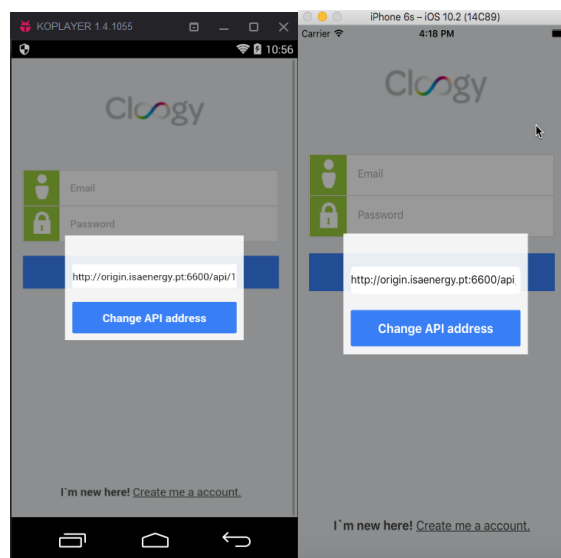


Figure B-1 Android(left) and iOS(right) login screen snapshot

- After setting the API address, please provide valid username and password to login into the application.

Usage Overview

This hybrid Cloogy prototype works like the existing Cloogy application available in Android, Windows Phone, and iOS, as the main goal was to retain the same functionalities along with GUI.

There are three main tabs in this application which are as shown below for Android and iOS.

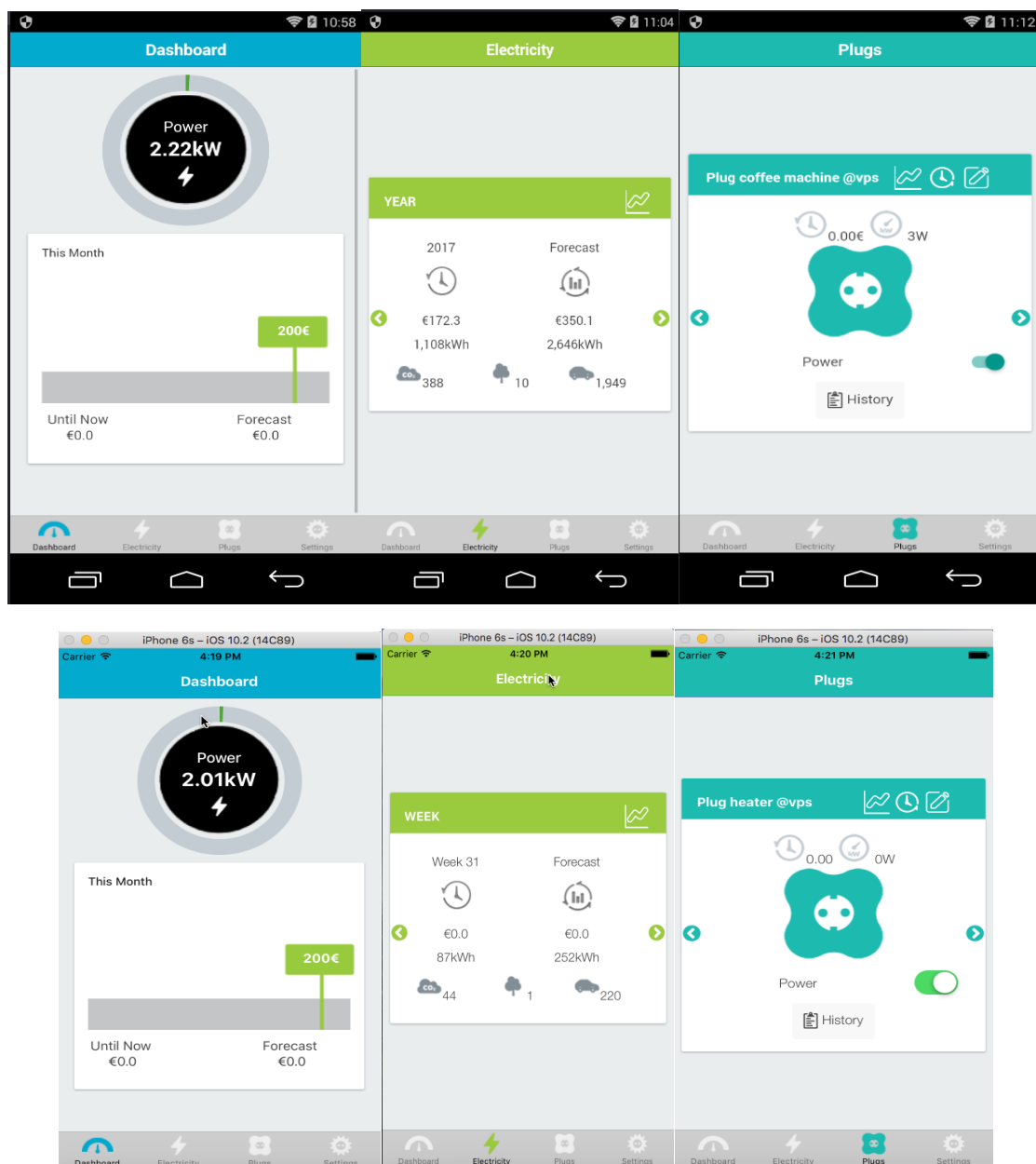



Figure B-2: Android(up) and iOS(down) usage overview of dashboard(left), electricity(centre), and plugs(right) modules.


Use Cases

Using this prototype, there are four main use cases in this mobile application. These are:




Dashboard



In the dashboard, the user is presented with the active power information along with monthly consumption details. The user has an option to change the monthly goal by clicking on the green flag  on the top of the monthly consumption progress bar.

Electricity

In the electricity tab, the user can view electricity consumption for a day, week, month and year. By clicking on the graph icon , the user can view consumptions in form of charts.

Plugs

In the plugs tab, the user can view consumptions , schedule actuations (create, edit, view, and delete) , edit plug icons  and view history .

In the front of the plug tab, the user can toggle the power button   to turn on or off the device from the application.

Settings

The last tab which is the settings tab provides an overview information about the prototype along with the copyright information. In the end, if the user wishes to logout of the application, he/she may do so by clicking on the logout button through the setting tabs.