Space Light
Hardware and Software system to monitor power consumption of various household appliances

Sanita Tifentale

Sanita.Tifentale@student.dit.ie

Supervisor: Dave Carroll

2nd Reader: Mark Deegan

March 26, 2015

This Report is submitted in partial fulfillment of the requirements for the award of Bachelor of Science (Hons) in Computer Science of the School of Computing, College of Sciences and Health, Dublin Institute of Technology.

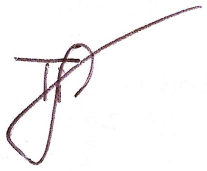
Abstract

In the past years energy costs have increased. There is greater demand in energy and concerns about global warming have generally contributed to a global push for energy preservation. Electricity represents 17.7% of the world's overall energy resources consumed in 2011 and demand for electricity is set to continue to grow faster than any other final form of energy. Between 2010 and 2035 the demand will increase by 70%, or 2.2% per year on average. In Ireland from 2008 up to 2013 a domestic consumption electricity fuel prices have drastically increased by 18.3% and it is the fourth highest price in EU.

The aim of this project is to develop a hardware and software system to monitor power consumption of various household appliances and provide the user with a feedback on their electricity consumption. The web application would not only allow the user to view, compare and contrast their electricity usage but also would motivate them to conserve the energy by setting monthly targets. This will not only conserve the energy but also minimize the household bills as now days the most beneficial way to spend less on electricity bills is to keep constantly switching supplier.

Declaration

I **Sanita Tifentale** hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.



Signed _____
Sanita Tifentale

Acknowledgements

I would first off like to thank my supervisor, Dave Carroll for his continued assistance and help throughout the project. His encouragement to keep up with a good work and motivation to not be afraid of extra work.

I also would like to express my gratitude to Mark Deegan for not only being my second reader but also for being my mentor and supporting me throughout the project. For providing me with invaluable knowledge beyond the scope of the project and always assisting me.

This project would not have succeeded without two best mentors that I had, it was my pleasure to work with both of you.

I would like to thank my family and friends for comforting me throughout the college. I also would like to express my appreciation to my beloved one for invaluable support, patience and care thought the college.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"In 2012, in Ireland an electricity consumption fell by 2.9% to 24 TWh, but energy-related emissions from electricity generation grew by 6.6%. Energy use in buildings fell by 3.0% and accounted for 31% of final demand. Energy consumption per household was 5.5% lower in 2012 than in 2011 and residential energy use fell by 4.2%. In Ireland from 2008 up to 2013 a domestic consumption electricity fuel prices have drastically increased by 18.3% and it is the fourth highest price in EU due to heavy dependence on imported oil and gas, and the introduction of carbon tax." [33] [27] [65]

Facts stated above clearly indicate that Ireland's electricity consumption is slowly reducing but the emissions from the electricity generation grow and fuel prices have drastically increased. Solutions that would help electrical conservation need to be introduced in order to save more energy within residential buildings. This would lead to lower household bills, informed and motivated users who would conserve and perhaps care more about the environment.

1.1 Project Objectives

The objective of this project is to develop a hardware and software system to monitor the power consumption of various household appliances. The gathered data then can be represented in the various graphs and statistics, it can be used to indicate what appliances cost more to run and could be candidate for replacement. The project can be broken down into three main components: data acquisition and transfer to the collection service, data storage and user interface.

The main objectives of this project are as follows:

- Research energy management systems
- Research hardware components of the project
- Research technologies and development techniques

- Research human computer interaction guide
- Develop data collector and transmitter
- Set up all the cloud services
- Design and create database
- Develop an API
- Design and develop user interface
- Deploy the web application

1.2 Project Challenges

The key challenge of the project is to ensure that all the project components are linked together to create a complete system. This is a huge challenge because the project consists of many components and all the components are critical to ensure a complete system implementation. Second key challenge is to choose the right technology for each project component implementation to ensure cohesive system. Another critical aspect of the project is to create a user friendly, intuitive and cross-platform user interface that will provide the users with the detailed household electricity usage and statistics. Provide the user with the information and motivation to conserve the energy and lower their household bills.

1.3 Report Structure

An overview of the upcoming chapters:

1.3.1 Chapter 2 – Research

This chapter will include an overview of the energy management systems and several already existing solutions similar to the proposed project will be discussed. It will also include a discussion on the hardware components of the project and human computer interaction guidelines.

1.3.2 Chapter 3 – Technologies

The technologies chapter will discuss all the technologies used within the scope of the project during all the stages of the project, including the project management tools.

1.3.3 Chapter 4 – Design & Architecture

This chapter will discuss the architecture and design aspects of the project. It will also examine the chosen methodology, the use cases, the presented database and the user interface design.

1.3.4 Chapter 5 – Implementation

The implementation chapter will discuss how each component was implemented and integrated into a complete hardware and software system.

1.3.5 Chapter 6 – System Validation

This chapter will discuss a different testing approaches undertaken to ensure project validation and accuracy. Validation will be achieved by system, regression testing and accuracy by the end-user feedback.

1.3.6 Chapter 7 - Conclusion & Future Work

This chapter will evaluate on the project completeness, changes made during the design or implementation stage. It will include a future work, learning outcomes and a personal reflection on the experience gained from the project.

Chapter 2

Research

2.1 Introduction

The main focus of the project is directed towards the need to monitor household electricity usage to lower the household electricity bill as much as possible. This chapter covers the overview and technological advancement on energy conservation in buildings. It also includes the overview of different energy monitoring tools, selection of low cost computing device for the project and alternative solutions. Furthermore the human and computer interaction is discussed.

2.2 Energy Management System Overview

"In recent years, rising energy costs, finite energy resources, increased energy demand, and concerns about global warming have collectively contributed to a global push for energy conservation. The use of electricity in commercial and residential buildings represent a significant portion of overall energy consumption. These buildings consume approx. 40% of the total electricity energy expenditure in Europe [Energy-Book 2010] and the U.S. [EIA 2013]." [39] The household electricity monitoring system would be a great solution to reduce the electricity consumption, example would be Building Energy Management Systems (BEMS). "BEMS are an integral part of so-called intelligent building [So and Chan 2012]." [39] Figure 2.1 illustrates the summarized technological advancements on energy conservation in buildings.

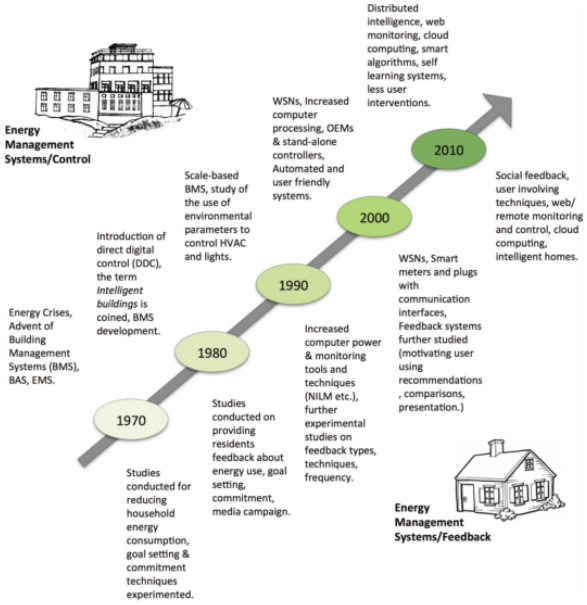


Figure 2.1: Technological advancements on energy conservation in buildings [39]

2.3 Generic BEMS Architecture

The main three key components are the sensor, the collector and the application, see Figure 2.2. Sensor layer is a physical-level which uses sensor nodes deployed inside a building that measures and transmits the measured data to the collector. The Computation layer consists of database which stores all the transmitted data with the timestamps and the computation unit which analyses the data. The Application layer provides the user with feedback system or energy control system. Feedback system presents real-time and historical energy usage information, this can be achieved via web or any web active device, e.g. mobile device. The energy control system controls the devices and appliances within the building via any device that is networked to the system. [39] The Figure 2.2 illustrates the generic architecture diagram of BEMS. This architecture model is the base model for this project too, the only difference is that this project will mainly concentrate on the feedback system rather than the control system in the application layer.

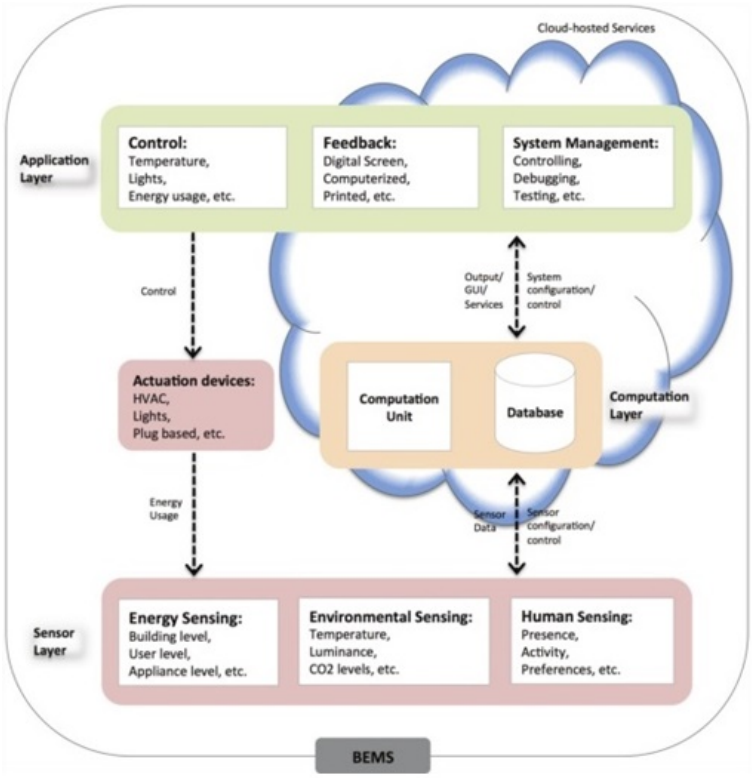


Figure 2.2: Generic architecture model of building energy management system [39]

2.4 Energy Monitoring Tools

"Traditionally, meters installed in buildings between the baseline and the external power line fail to provide sufficient detailed data on energy usage, nor do they offer a communication interface by which one could obtain such data in real time." [39] Two major approaches to monitor energy usage are "single-point monitoring (NILM)" and "distributed monitoring (ILM)." NILM – Non-Intrusive Load Monitoring [12] would be installed on the fuse box to measure the overall electrical usage of a building but ILM would be attached to each appliance within the house, to measure each particular appliance's usage at a time. [39] This project will mainly concentrate on distributed monitoring approach as certain household appliances need to be monitored but also single-point monitor is essential to the project for knowing the overall household usage.

System name	Year	Appliance Level	Building Level	Wireless	Clip-on	Display
SeeGreen	2001	✓				✓
Plug	2007	✓		✓		✓
Smart-Socket	2008	✓		✓		✓
ACme	2009	✓		✓		✓
iSensor	2009	✓		✓		✓
ECD	2009	✓	✓	✓	✓	✓
Distributed power meter	2010	✓		✓	✓	✓
Circuit-Panel meter	2010	✓	✓	✓	✓	✓
Low-Cost appliance state sensing	2012	✓		✓		✓

Figure 2.3: Smart Meters and Outlets [39]

Figure 2.3 provides the overview of smart meters and outlets comparison over past 10 years. Brief description of a few smart meters and outlets: "Acme meter main difference between other outlets are that it uses the IPv6/6LoWPAN stack with router to connect to other IP networks. Plug sensor differs from other smart meters as the node harnesses a range of embedded sensors to obtain information on various environmental parameters, this improves the accuracy of appliance recognition. SeeGreen is one of the earliest smart outlets design, this node uses power line networking to monitor and control all attached electrical appliances. iSensor measures current consumption of a plugged appliance and transmits data via ZigBee to a monitoring unit similar to other outlets; however, it does not provide control over appliances to switch them on or off". [39]

There is a wide range of monitoring tools and devices that could be used for this project, but DIT Lecturer Mark Deegan's monitoring tools were selected, Siemens 7KT1 530 [55] and XTM18SA [26]. Siemens and XTM are single phase energy meters, which work in the same way, but at a different rate. Mark Deegan's comment on the monitoring device: Each generates a fixed number of pulses every Kilo Watt Hour (KWh). More power is being consumed, the faster the device will pulse. It might, for example pulse every 40 seconds when the powered device is consuming 1kW and would pulse every 20 seconds when the device is consuming 2kW. The pulse rate is indicate on the device, each pulse equates to a TTL (Transistor Transistor Logic 5v) patterns as follows where the signal goes from 5v to 0v for a short period of time. This can be viewed on the meter display and LED flashes every time there is a pulse, which is a great indicator.

Figure 2.4: Pulse

By counting the pulses, we can know how much energy is being consumed, by measuring the time between pulses we can calculate the power of the device. A more powerful device leads to more frequent pulses, which leads to more energy consumed in a given time. To count the pulses Arduino Uno sets up an interrupt routine to handle the falling edge, when signal falls from 5v to 0v, counter is incremented. Figure 2.5 displays the electricity meters and how they are connected to the Arduino Uno.

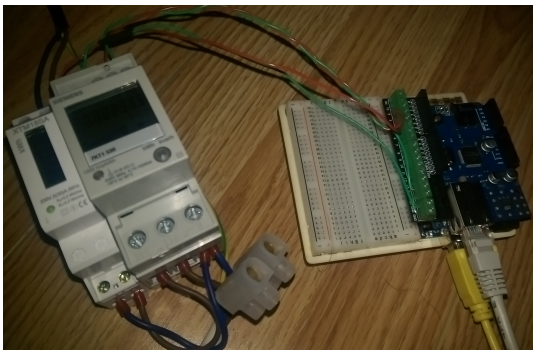


Figure 2.5: Electricity power monitors with Arduino Uno

Tweet-A-Watt power monitoring device is wireless and can be made from commercial Kill-A-Watt monitoring device and a XBee module running a ZigBee protocol. This device will send the data directly to the Internet. There is a starter kit available for creating a Tweet-A-Watt power monitoring device. This could be an alternative solution which could make this project wireless and would be a better solution fora commercial product.[63]

2.5 Low cost computing devices

To ensure that right low cost computing device is selected for the project, the Raspberry Pi, Arduino Uno and BeagleBone Black has been researched and compared. The functional overview comparison of selected micro-controllers can be viewed in Table 2.1.

Functional overview	Arduino Uno	Raspberry Pi	BeagleBone Black
Revision	3	B+	C
Price	20 Euro [6]	33.27 EUR [50]	44 EUR [14]
Size	68.6 x 53.4 mm [4], [20]	85 x 56 x 17 mm [42]	86.36 x 53.34 x 4.76 mm [14], [22]
Processor	16 MHz [4], [20]	Broadcom BCM2835 ARM11 700Mhz [42]	Sitara AM3358BZCZ100 1GHz [14] , [22]
RAM	SRAM 2KB [4], [20]	512 MB RAM [50], [42]	512 MB RAM [14], [22]
Storage	EEPROM 1KB, 32KB Flash memory [4], [20]	MicroSD [50], [42]	microSD 3.3V, on board flash storage – 4GB [14]
Video Connection	N/A	Integrated Videocore 4 Graphics GPU [42]	SGX530 3D, 20M Polygons/S [8]
Supported Resolution	N/A	Full 1080p HD Video [42]	1280 x 1024 [14], [22]
Audio	N/A	4 Pole Stereo [42]	Via HDMI Interface, Stereo [14]
OS	Arduino Tool	Raspian, OpenELEC, RaspBMC, Linux [50], [42]	Debian, Android, Ubuntu, Windows [22]
Power Consumption	50mA [36]	600mA [42]	mA@5V [22]
GPIO Capability	14 i/o pins & 6 analog inputs [4]	40 pins [50], [42]	2 x 46 pin headers [22]
USB	1 USB	4 USB ports [50], [42]	1 USB port [22]
Ethernet	No – shield or Arduino Ethernet [4]	10/100Mb Ethernet Port [42]	10/100, RJ45 [14]
HDMI	No – can use RP as a shield [20]	Yes [50], [42]	Yes [14], [22]
Weight	25g [4]	18g [50]	39.68g [22]
Power Supply	9V [53] and others	5V 2A [42]	5V 2A [22]

Table 2.1: The comparison table of different micro-controller boards

2.5.1 Arduino Uno

The Arduino Uno is a micro-controller board, which has 14 digital input/output pins, 6 analog inputs, a USB connection and some other components. Arduino team defines it: "Arduino is an open-source electronics prototyping

platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and everyone interested in creating interactive objects or environment. Arduino can sense the environment by receiving input from variety of sensors and can affect its surroundings by controlling lights, motors and other actuators." [6] It is also the best solution for the beginners as it is less complex in comparison to other micro-controllers. Arduino has a very large user community, there are many samples tutorials and sample projects available online. Figure 2.6 illustrates the Arduino Uno basic architecture.

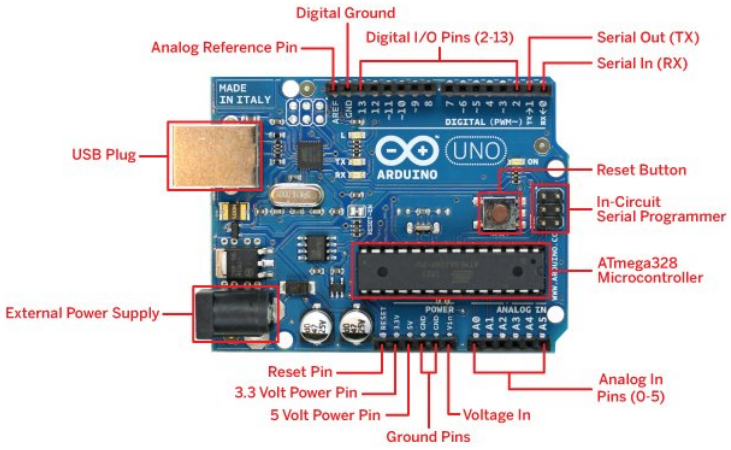


Figure 2.6: Arduino Uno Architecture [45]

To connect Arduino Uno to the Internet the Ethernet Shield or WiFi Shield can be used. Both shields have been used and WiFi Shield have been selected for this project as it is more convenient and more flexible. It allows for Arduino Uno to be placed anywhere in the household. Figure 2.7 shows how Ethernet shield sits on the Arduino Uno the same way as WiFi shield would.

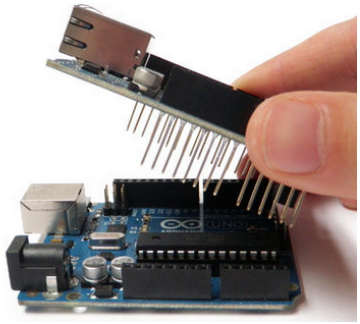


Figure 2.7: Arduino Ethernet Shield on top of Arduino Uno [5]

2.5.2 Alternative Solutions

Raspberry Pi

Raspberry Pi would be a good solution for the project but it is more complex and it might take longer to learn and adapt to it. It is great for networking & graphics projects as it supports full HD graphics.dfr[32] [7]

BeagleBone Black

BeagleBone Black is the least known of all three micro-controllers but it has its own advantages. It fully supports Linux, it has good input and output features and it is easy to connect to the network.[32] [7]

2.5.3 Conclusion

Arduino Uno in comparison to other boards might not seem powerful enough but it is a good solution a project of this scale . It has many advantages compared to other two low cost computing devices. Not only it is a good tool for learning but it has the lowest power consumption of all three boards and it can work with wide range of input voltages, different type of batteries. "It is a flexible platform with great abilities to interface to most anything", any type of hardware. Arduino comes in different sizes and types which makes it very versatile. Arduino Uno is just one of many boards available, there are Arduino Yun, which has already built in Ethernet, Wi-Fi and other useful components. Other examples would be Arduino Micro, Arduino Esplora and many more. [7]

2.6 Alternative Project Solutions

2.6.1 Energy Feedback Systems Overview

Energy Consumption and Cost is usually provided within the feedback systems to help the users to make financial savings. Energy consumption can be reported in kWh along with energy cost. Also other characteristics such as appliance-level energy consumption, comparison with past usage, and energy consumed by neighbours or other users. Other energy feedback systems provide the user with appliance-state information, this can allow the user to switch off the appliance if not used. Environment impact is one of motivational factors for the users as they can lower the carbon emissions for greener environment, to achieve this the feedback systems inform the users on their carbon footprints.

Feedback Systems can be broken down into different types:

- **Space Specific system** is achieved by dividing a building into smaller areas, division can be classified by rooms or appliance types. [39]
- **Appliance Specific system** information is considered as highly enriched feedback targeting energy estimation at appliance level. [39]
- **Time-Specific system** provides the feedback on different timescales, real time data and past data on hourly, daily, weekly or monthly bases. This system also informs the user on future predictions on the user's usage over a certain time. [39]
- **User-Specific system** or personalized feedback system can help individuals to keep track of their own energy usage. [39]
- **Service-Specific system** is recently presented [Bates et al. 2012]. A service is defined as an activity or combination of activities performed within a building, such as heating, lightning or cooking. Usage of one or more appliances can structure a service in the context of everyday life. [39]

Looking at all available feedback system types, this project will consist of Appliance-Specific components, which concentrates on appliance usage in the household and suggest the appliances for the replacement. The project will also include the Time-Specific components that will give the users the real time feedback and the historical usage overview. Then it can be used to compare with other users usage and look up how the progress of conservation has been made. Instead of offering the users future predictions the system will provide the user with the ability to set their own goals, this would motivate them to reduce electricity usage. Service-Specific system is a great idea because it could give the overview of the daily routine to the user. This information might help users to change their daily lifestyle by giving detailed information on their household activities / services. Below are listed few interesting feedback systems that have been proposed, more feedback systems can be found in the journal article [39] and [64].

2.6.2 HBCI: Human-Building-Computer Interaction

Their implementation approach is interesting as they use QR codes as a tags to identify objects for the implementation. "QR codes can be easily printed and attached to any object, reducing the cost of adding an object into the HBCI database. They use ZXing decoder on the Android mobile OS. The advantage of high decoding speed, small size and high density make QR codes an ideal choice for storing any URI metadata."[34] Then they have their application "mPAD" (Mobile Personal Appliance Dashboard), which can be installed on any Android OS phone. Once the user has logged in their app they can scan the QR code which encodes the URI, then the user can view the data on the object, choose service or action and then see the

visualization (graph). Users can also remotely turn on or off the appliances by using Acme plug-load meter and sMAP an application layer protocol to retrieve data streams from sensors. The system also provides the user with a time trigger that can turn off the light or appliance at any given time. The daily user's summary is displayed to increase awareness and energy reduction.[34]

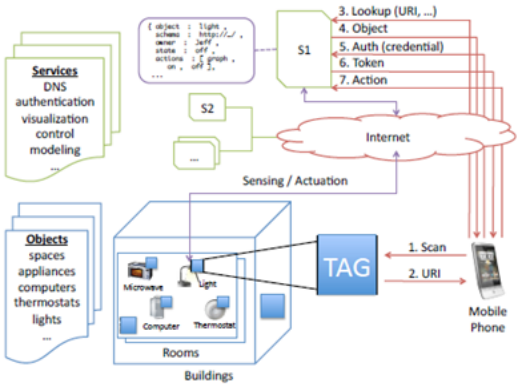


Figure 2.8: Process flow of the HBCI Architecture [34]

2.6.3 Yupik

"Yupik, a system that helps users respond to real-time electricity prices while being sensitive to their context and lifestyle. Yupik is essentially a planner that uses variable hourly prices and computes optimal appliance usage schedules for the next planning horizon (e.g. a day or two, or a week). It uses jPlugs, appliance level energy metering devices, to sense household's appliance usage patterns. Yupik's analytics engine analyses consumption data collected from jPlugs. " [11]

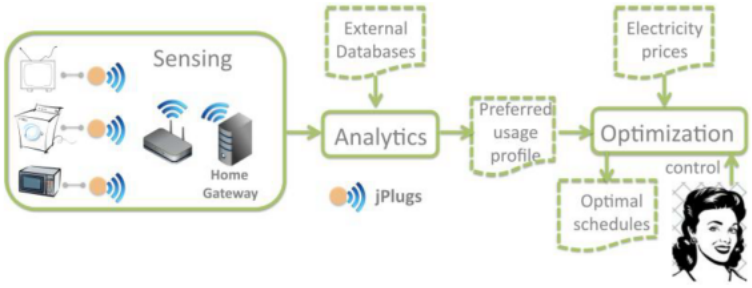


Figure 2.9: High level architecture diagram of Yupik [11]

2.6.4 Wireless Sensor Network for Single Phase Electricity Monitoring System via ZigBee protocol

"This paper proposes a star topology wireless sensor network for single phase electricity monitoring system based on Energy Meter IC ADE7753. Zigbee is used as open - source wireless protocol which provides many benefits such as its low cost, its low power and its low data rate. For web-server monitoring system, Ext-JS framework is used as client-side programming which is integrated with IDE Arduino. The system is divided into 3 clusters; there are main-device, base-station, and web-server. " [2]

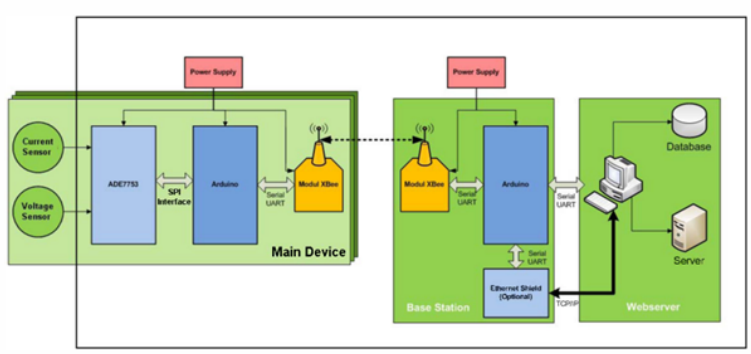


Figure 2.10: Wireless sensor network for single phase electricity monitoring system via ZigBee protocol concept design [2]

2.6.5 Insight into Home Energy Consumption in India

"Primary objective for this deployment was to bring forth the differences in the Indian context, as compared to the context of developed countries along the dimensions of - 1. Grid and network reliability; 2. Energy and water consumption patterns; and 3. The ecosystem of available sensing options that restrict the possible deployments." [13] For electricity monitoring at meter level they used "EM6400 Smart Meter" [13], at circuit level they used "Split-core CTs, clamped to individual MCBs" [13] and at application level "jPlug" [13] was used. This project involved not only electricity monitoring but also water and motion monitoring. Picture below shows "different granularity of measuring electricity consumption in home: meter, circuit and appliance and water consumption in home." [13]

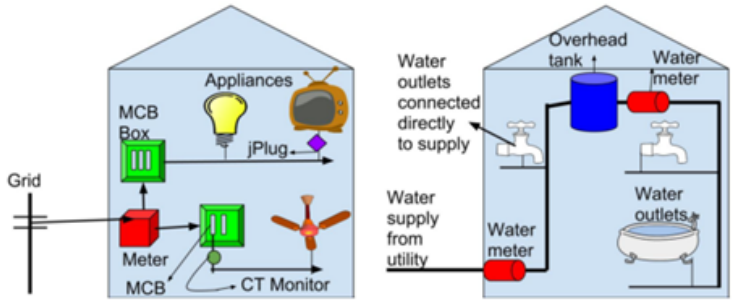


Figure 2.11: Granularity of measuring electricity and water in homes [13]

2.6.6 Ubiquitous Smart Energy Management System for Residential Homes (USEM)

"System is to allow long-term measurement and real-time monitoring of electricity consumption, and to provide the necessary mechanisms for intelligent and efficient control of power usage. USEM has to be developed as a ubiquitous system that supports adding, changing and removing numerous power sockets, devices, sensors, control switches and actuators. It also requires that user is able to interact with, and control the system using a range of tools from a central unit, at the device level, or remotely across the Internet. System also should schedule tasks, such as turning on the washing machine when the electricity price is low. To achieve this, an intelligent algorithms would be used. This system also would be design to work across a range of devices." [40]

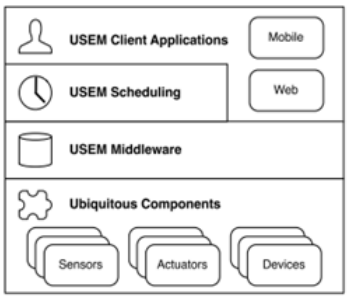


Figure 2.12: Architecture layers of USEM [40]

2.7 Human Computer Interaction

The goal of HCI is to design system that minimizes the barrier between a human and computers by enabling computer to satisfy the user's needs in a feasible manner.

2.7.1 The Eight Golden Rules of Interface Design

1. **Strive for consistency** "Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent color, layout, capitalization, fonts, and so on should be employed throughout." [56]
2. **Cater to universal usability** "Recognize the needs of diverse users and design for plasticity, facilitating transformation of content. Novice to expert differences, age ranges, disabilities, and technological diversity each enrich the spectrum of requirements that guides design."[56]
3. **Offer informative feedback** "For every user action, there should be system feedback. For frequent and minor actions, the response can be modest, whereas for infrequent and major actions, the response should be more substantial. Visual presentation of the objects of interest provides a convenient environment for showing changes explicitly."[56]
4. **Design dialogs to yield closure** "Sequences of actions should be organized into groups with a beginning, middle, and end. Informative feedback at the completion of a group of actions gives operators the satisfaction of accomplishment, a sense of relief, a signal to drop contingency plans from their minds, and an indicator to prepare for the next group of actions."[56]
5. **Prevent errors** "As much as possible, design the system such that users cannot make serious errors. If a user makes an error, the interface should detect the error and offer simple, constructive, and specific instructions for recovery. For example grey out disabled or unused fields, do not clear entire registration form if invalid email address was entered." [56]
6. **Permit easy reversal of actions** "As much as possible, actions should be reversible. This feature relieves anxiety, since the user knows that errors can be undone, and encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data-entry task, or a complete group of actions, such as entry of a name-address block." [56]
7. **Support internal locus of control** "Experienced users strongly desire the sense that they are in charge of the interface and that the interface responds to their actions. They don't want surprises or changes in familiar behavior, and they are annoyed by tedious data-entry sequences, difficulty in obtaining necessary information, and inability to produce their desired result." [56]
8. **Reduce short-term memory load** "Humans' limited capacity for information processing in short-term memory (the rule of thumb is that we can remember "seven plus or minus two chunks" of information) requires that designers avoid interfaces in which users must remember information from one screen and then use that information on another

screen." [56]

2.7.2 Nielsen's heuristics

1. **Visibility of system status** "The system should always keep users informed about what is going on, through appropriate feedback within reasonable time." [1]
2. **Match Between system and the real world** "The system should speak the user's language, rather than system-oriented terms, jargon. Follow real-world conventions, making information appear in a natural and logical order." [1]
3. **User control and freedom** "Provide user with clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo." [1]
4. **Consistency and standards** "Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions." [1]
5. **Error prevention** "Carefully design the system, which would eliminate the occurrences of error messages. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action." [1]
6. **Recognition rather than recall** "Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another." [1]
7. **Flexibility and efficiency of use** "Take in consideration expert and inexperienced users. Allow users to tailor frequent actions." [1]
8. **Aesthetic and minimalist design** "Dialogues should not contain information which is irrelevant or rarely needed." [1]
9. **Help users recognize, diagnose, and recover from errors** "Error messages should be expressed in plain language, precisely indicate the problem, and constructively suggest a solution. Avoid using codes." [1]
10. **Help and documentation** "Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large." [1]

Chapter 3

Technologies

3.1 Introduction

This chapter describes technologies used during the design and development stage of the project. Technologies are grouped into relevant groups.

3.2 Programming Language

3.2.1 Python

Python and Java are used for web application development but the project's main programming language will be Python. It has easy syntax, no brackets, pure indentation which keeps the code clean and clear and easy to understand. "Python has a lot of third party libraries" [18]. Flask or Django frameworks allow to develop fairly complex web applications very quickly. "Python is faster than Java because it is not running inside the virtual machine" [38]. It is open source, robust, flexible [19] and as portable as Java. "Python uses dynamic typing so variable type can be changed at any time but Java uses static typing, which forces user to declare the variable type and it can't be changed later in the program" [38]. "Python programs are typically 3-5 times shorter than equivalent Java programs" [23], so in my opinion it should be faster to complete. Figure 3.1 which could be found below shows that Java is the leading programming language but it is decreasing in demand. Python programming language might not be number one but it is slowly moving up when other languages are moving down.

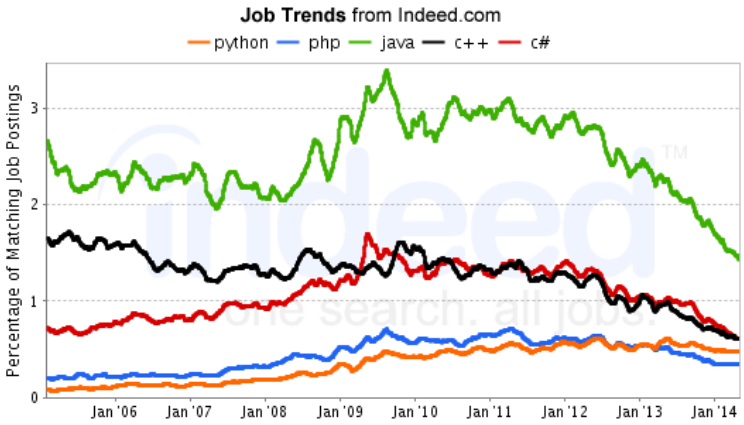


Figure 3.1: Job Trends from Indeed.com [49]

The programming language used to teach introductory courses in top 39 U.S. computer science departments. Figure 3.2 below shows that Python is number one programming language used to teach the introductory course, even though "Java was the dominant over the past decade". [48]

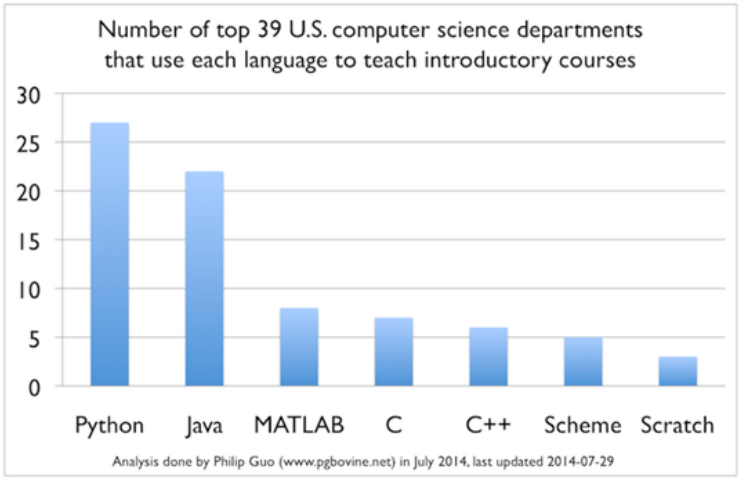


Figure 3.2: Programming Languages used to teach introductory courses in top 39 U.S. computer science departments [48]

3.3 Data Acquisition & Upload

3.3.1 Arduino Language

"Arduino programs can be divided in three main parts: structure, values (variables and constants), and functions." [4] It is based on C/C++ programming languages, in this project the program written in Arduino is used within the Arduino Uno, low cost computing device, to obtain data from the electricity power meters and to connect to the MQTT server to transmit obtained data to the server.

3.3.2 MQTT Protocol

"MQTT stands for MQ Telemetry Transport. It is publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements while also attempting to ensure reliability and some degree of assurance of delivery." [28] This is an ideal solution for the project because it is simple to setup and use. MQTT message consists of client, topic and some specific value, in this case pulse count. This provides the ability to have multiple topics, which allow multiple sensors transmitting data to Arduino Uno and then to the MQTT server, where each topic could represent a household and each subtopic could represent an appliance in the household. The client can be predefined for each meter which allows the direct connection to the specific meter and specific user.

3.3.3 MQTT Message Log Plug-in

"The MQTT Message Log is a very useful HiveMQ Plug-in for debugging and development purposes. It provides the possibility to follow up on any client communication with the broker on the terminal (and standard HiveMQ log files)." [43] HIveMQ plug-in is used to obtain the client communication with the broker (MQTT server hosted on Amazon EC2) via log files. Once log files have been obtained the necessary data has been extracted - time stamp, topic, client and the pulse count via python module. Then SQL insert statements are created and data is inserted in to the database, hosted on Amazon RDS. Alternative solution to this could be to develop a Plug-in which will directly insert data into the database.

3.4 Cloud Computing Services

3.4.1 Amazon AWS

"Amazon AWS is a collection of remote computing services, also called web services that together make up a cloud computing platform by Amazon.com since 2006. The most central and well-known of these services are Amazon EC2 and Amazon S3. The service is advertised as providing a large computing capacity (potentially many servers) much faster and cheaper that building a physical server farm." [10] Different Amazon AWS services was used for different parts of the project, they are listed in the section below.

Amazon EC2

"Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides re-sizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers." [8] Amazon EC2 is used to host MQTT server, API and the User Interface, three separate instances have been made to separately host the individual components.

Amazon AMI

"Amazon AMI (Amazon Machine Image) is a special type of virtual appliance that is used to instantiate a virtual machine within the Amazon EC2. It serves as the basic unit of deployment for services delivered using EC2." [8] Amazon AMI was used to provide other students who has related projects with the ability to use this projects MQTT server hosted on Amazon EC2 instance.

Amazon RDS

"Amazon RDS (Relational Database Service) is a web service that makes it easy to set up, operate, and scale a relational database in the cloud." [9] This project uses Amazon RDS to host PostgreSQL database, which allows the user to access the database from anywhere at any time.

Amazon Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. It provides the user with ability to register a domain name or transfer from other domain name systems. Any Route 53 domain names can be mapped to the EC2 instance, Amazon S3 buckets and other AWS web services.

3.5 Data Storage

3.5.1 PostgreSQL

PostgreSQL is the worlds most advanced open source database. Mainly the database will be filled with time series data, time stamp and the value associated with it. "PostgreSQL is effective and fun way of storing this type of data" [15]. It has different powerful functions such as "datetrunc" which is used with sum aggregate to count the value for particular time period (min, hours). PostgreSQL also have Window functions which are high-powered as you can apply aggregates over windows [15]. PostgreSQL can be used as SQL DB or NoSQL DB which means it can adapt to different types of situations. It can store structured data as it can use JSON and Hstone to handle documents [52]. This is great feature when data is in JSON format, because machines can easily parse and generate it. Database is flexible as you can add or remove columns on the fly. It is fast and scalable as scalability can be achieved by Proxy PL or Master Slave replication. PostgreSQL is highly optimized and it is compatible as SQL is an industry standard and many vendors support SQL. [52] "NoSQL market is fragmented and has no standards" [52], this means the project DB will be using SQL rather than No SQL.

3.6 Back-End Web Service Framework

3.6.1 Flask

"Django is the most popular of all python frameworks to date with 80 000 stack overflow questions where Flask has only 5 000 questions. On GitHub they have nearly identical number of stars 11 300 for Django and 10 900 for Flask." [17], [25] Pyramid used to be the number one framework but in 2011 [25] it has drastically fallen and Django became the leading python framework. This project back-end web service framework is Flask even if it is the youngest of all frameworks but it has been able to learn from frameworks that have come before and have set its sights firmly on small projects. Flask is good for small projects as it is fast to develop and should require little configuration. It also provides a simple API error/exceptions which provides the users with human readable error messages, a status code for the error message and the description of the status code. Flask has blueprints which are used to scale up the application and group related endpoints together which allows for bigger web interface development. [66]

3.7 Front-end Web App MV* Frameworks

AngularJS, Backbone and Ember have a lot in common: they all are open sourced and they try to solve the problems of creating Single Page Web Applications using the MV* design pattern. They all have the concept of views, events, data models and routing. [54]

3.7.1 AngularJS

Was born in 2009 as a part of a larger commercial product, called GetAngular, developed by Google. Unique and innovative features are a two-way data binding, dependency injection, easy-to-test code and extending the HTML dialect by using directives. Examples of users would be Google, YouTube, Nike, and others. It includes templates, HTML with binding expressions baked-in, which are surrounded by double curly braces. [54], [3]

Pros:

- Once you learn it, the user can be very productive
- Two way data binding saves a lot of boiler point code
- Categorises application building blocks
- Automatic dirty checking (no need for getters and setters)
- Written with testability in mind

Cons:

- Difficult to learn
- Follow certain structure
- Complexity of the Directives API
- Scope hierarchy uses Prototypal inheritance which can be confusing for developers coming from OOP
- Angular expressions are abused in the View layer which makes it hard to understand
- Misspelling a directive name or calling an undefined scope function are ignored, makes it hard to find
- It is easy to forget to call $digest() when running in non-Angular context
- Angular becomes really slow when page contains a lot of interactive elements. One page should only contain 2,000 interactive elements

3.7.2 Alternative Solutions

Backbone.js

is a lightweight MVC framework, born in 2010, it quickly grew popular as a lean alternative to heavy, full-featured MVC framework such as ExtJS. It was developed by Jeremy Askkenas and DocumentCloud. It doesn't include

templating, so it has a default Underscore templating but it is very basic and usually have to be used with JavaScript, which makes it more complex. Examples of users would be Twitter, LinkedIn Mobile, Soundcloud and others. [54], [3]

Pros:

- Easy to learn
- Flexibility
- Lightweight
- Fast
- Has a small memory footprint
- It has a great documentation
- Has annotated version of the code which explains how it works in detail
- It is used as a foundation to build your own framework

Cons:

- Has a lot of boiler point code due to lack of support for two-way data binding
- Lack of view lifecycle management it can have memory leaks unless you take care of cleaning it yourself
- A lot of decision making required as so many plugins are available
- Views manipulates DOM directly, making it really hard to unit-test, less reusable and fragile

Ember's

roots go way back to 2007, starting its life as the SproutCore MVC framework, originally developed by Sproult. It has Handlebars template engine, extension of Mustache templating engine. They want to make a transition to HTMLBars which will make the templating more appealing. Examples of users would be Yahoo, Groupon, Zendesk and others. [54], [3]

Pros:

- Automated controller create if not defined
- Includes excellent router and optional data layer called ember data
- Integrates with Ruby-on-Rails backend or any other Restful JSON API
- Performance – Run Loop

Cons:

- Has a lot outdated content and examples that no longer work, makes it confusing for developers to start.
- Handlebars pollutes the DOM with script tags, which can break the CSS styling or integration with other frameworks (jQuery). This will be resolved when HTMLBars will be introduced.

Metric	AngularJS	Backbone.js	Ember.js
Stars on Github	27.2k	18.8k	11k
Third-Party Modules	800 ngmodules	236 backplugs	21 emberaddons
StackOverflow Questions	49.5k	15.9k	11.2k
YouTube Results	~75k	~16k	~6k
GitHub Contributors	928	230	393
Chrome Extension Users	150k	7k	38.3k

Figure 3.3: Community overview in 2014 [54]

3.7.3 Conclusion

AngularJS is a good solution for people who are familiar with web development as it is an innovative approach for extending HTML. It has a large community and Google support, which means it's here to stay and grow and it can be used for any type of project. Backbone is an ideal solution for small projects, lightweight projects. Ember is the oldest of all three and it's a good solution for people who are familiar with MVC programming in Ruby, Java, Python or any other OOP language. This project will be using AngularJS as it is here to stay and it's popularity is increasing.

3.8 HTML 5

"HTML5 is a markup language of the Internet used for structuring and presenting content for the WWW. As of October 2014 this is the final and complete fifth revision of the HTML standard of the W3C". [35] Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices. It is used to create a responsive website, a feedback system for the end user.

3.9 CSS3

CSS is a cascade style sheet language used to layout the markup language components, give structure and design. CSS3 is divided into the modules, these modules extends or enhance the capabilities of CSS2 and it is backward compatible. It is used to give the feedback system its responsive design and appeal to the end user.

3.10 Project Management Tools

3.10.1 Yeoman

"Yeoman helps you to kick-start new projects, prescribing best practices and tools to help you stay productive. The Yeoman work-flow comprises three types of tools for improving your productivity and satisfaction when building a web app: the scaffolding tool (yo), the build tool (Grunt) and the package manager (Bower). " [62]

Node.js

"Node.js® is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices." [44] Node.js environment and package manager npm was used to set Yeoman work-flow tools and create idea web application development environment.

Yo

"Yo scaffolds out a new application, writing your Grunt configuration and pulling in relevant Grunt tasks and Bower dependencies that you might need for your build." [62] It provides the developer with basic web app project structure, it lays out all the necessary folders and configuration files for an easy start, for example bower and grunt configuration files. It provides developer with opportunity to deal with simple project structure, with no complexity of project start up and less time wasted on set up.

Bower

"Bower, the Package Manager is used for dependency management, so that you no longer have to manually download and manage your scripts." [62] "Web sites are made of lots of things — frameworks, libraries, assets, utilities, and rainbows. Bower manages all these things for you." [16] It finds, downloads, saves and installs packages from everywhere. It keeps track of all the imported packages / dependencies in the bower.json file. This file is used in project movement, deployment to ensure no manual insertion, simple automation takes place. Simple Bower command is –bower install lib_name –save. –save is used to save the dependency and grunt then can pick it up and insert it in the correct place.

Grunt

"The Grunt is task runner and its ecosystem is huge and it's growing every day. With literally hundreds of plug-ins to choose from, you can use Grunt to

automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Grunt plug-in to npm is a breeze." [30] If we specify bower tags within the index file, we then can run grunt command and it will insert the dependency in the appropriate place, no manual insertions. It provides the developer with hundreds of plug-in which can be used within the project. Grunt can provide with many tasks, automated testing, server, dependency builder and many more.

3.10.2 GitHub

GitHub is web based Git repository hosting service, which provides code sharing, publishing, deployment and many more. It provides the developer with private and public repositories. GitHub supplies developer with easy deployment, no need for backup, version control, separates development from the production and it can be used for team work.

3.10.3 Mendeley

Mendeley is a free reference manager which stores any reference in any academic format you desire. It creates users own fully-searchable and scalable library that can be accessed from any device. It also provides a desktop application for fast and direct reference access. Mendeley also offers "Save to Mendeley" button that is added to the bookmark bar, once you find an article, a PDF file or a book, then click the button and the reference details are created, review the details and save the reference to your own library. This saves a grate amount of time, as 90% of the time details are automatically populated for you, in some cases manual insertion is required. The library can be exported or plug-in can be used to directly cite in Microsoft Word document.

3.10.4 ShareLatex

ShareLatex is an online LaTeX editor which allows for real time collaboration and online compiling of projects to PDF format. Interim report and final year report was written using ShareLatex. It had a learning curve but once you familiarize yourself with it and it will takes care of everything. All you need to do is provide the text.

3.11 Deployment tools

3.11.1 Nginx

Nginx is an open source HTTP and reverse proxy server with many features, like serving static page, load balancing, fault tolerance and many more. It

is also mail proxy server, supports SSL, uses POP3, IMAP, SMTP authentication methods and other features. It is a power-full web server, that serves static content quickly, it is known for it's speed.

3.11.2 Gunicorn

Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy. As Nginx, the web server can't directly communicate with Flask, web service framework, Gunicorn is used as intermediate module between Nginx and Flask. It is written in Python and is easy configurable.

Chapter 4

Design & Architecture

4.1 Introduction

This chapter describes the methodology chosen for the project. It also describes the architecture and design of the project.

4.2 Approach and Methodology

The project will be using Iterative & Incremental development also known as "Evolutionary" approach, many view it as a modern practice but its application dates as far back as the mid-1950s. The journal article "Iterative and Incremental developments, a brief History" written by C. Larman and V. Basili has a written comment from Gerald M. Weinberg, who worked on the project and he wrote a comment: "We were doing incremental development as early as 1957, in Los Angeles, under the direction of Bernie Dimsdale [at IBM's Service Bureau Corporation]." [41] This is the earliest statement / comment regarding the use of the IID.

The other interesting comment from this article was regarding the IID practice, the evolutionary project management and how evolution term was described. The term evolution description was "It is a technique for producing the appearance of stability. A complex system will be most successful if it is implemented in small steps and if each step has a clear measure of successful achievement as well as a "retreat" possibility to a previous successful step upon failure. You have the opportunity of receiving some feedback from the real world before throwing in all resources intended for a system, and you can correct possible design errors... " [41] This comment describes that software should be broken down in to the small sections, in this case the project is broken down into smaller steps, requirements. One requirement will be implemented at a time and then tested before moving on to the next one. This will ensure that implemented functionality is working and will not bring issues to the next implementation.

It's like putting together a jodsaw puzzle, all small pieces should come together and make one complete picture. Iterative approach gives an opportunity to redesign if not all possible pitfalls are anticipated because "one of the essential motives for iterative development is to reduce risk by validating the proposed design as early as possible". [58] This will allow for any changes to occur at any stage of the project, which provides great flexibility. An interesting fact in article "Using Both Incremental and Iterative Development" by Dr. Alistair Cockburn. He stated: "Sadly, iterative development has come to mean either incremental or iterative, indiscriminately. This was an unfortunate turn for our industry since each serves a different purpose and need to be managed differently. " [21]

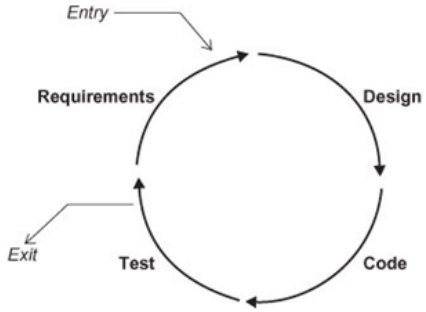


Figure 4.1: Iterative & Incremental Development model [57]

4.3 Use Cases

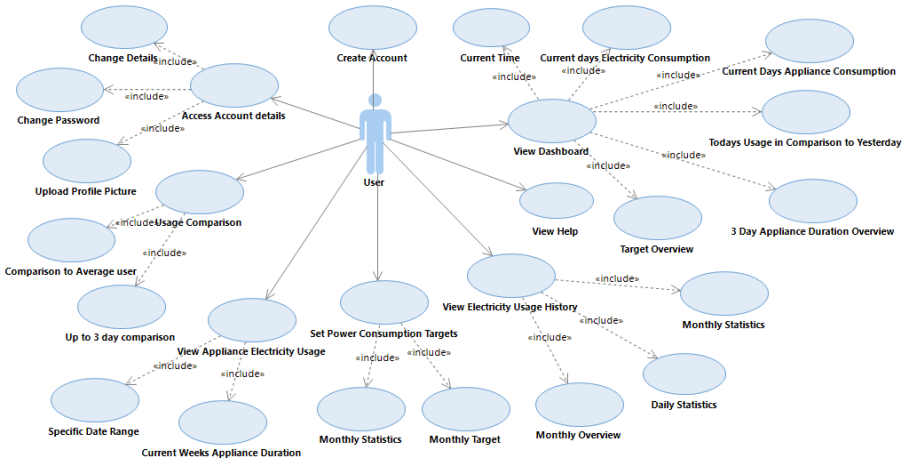


Figure 4.2: Use cases

4.3.1 Create Account & Access Account Details

Figure 4.3 illustrates the create an account and access account details use cases. User can create an account on the registration page, then log-in to the application and access the account details. Within the profile view the user can change his / hers details, profile picture and password at any time as long as they are logged in.

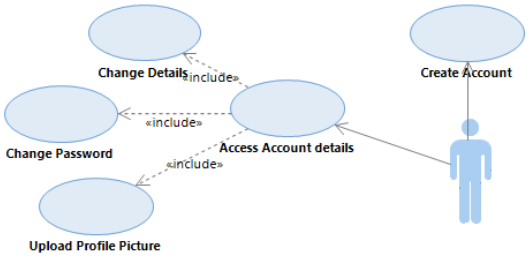


Figure 4.3: Create Account & Access Account Details use cases

4.3.2 View Dashboard

Figure 4.4 illustrates the view dashboard use case, which includes several use cases. Dashboard allows the user to view date time, current day's hourly usage, appliance usage, monthly target progress overview, appliance duration overview of past three days and today's and yesterday's comparison. To access the dashboard user need to log in and it will appear on the screen.

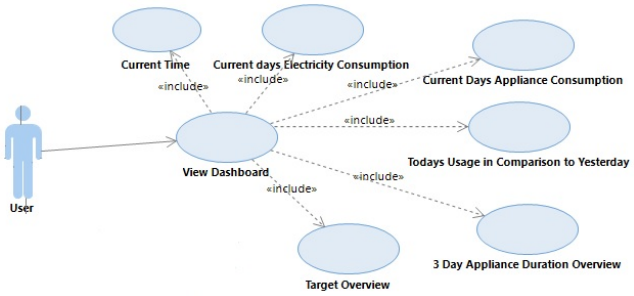


Figure 4.4: View Dashboard use case

4.3.3 View Electricity Usage History

Figure 4.5 illustrates the history use case, which includes three other use cases. History overview allows the user to view monthly usage, monthly and daily statistics.

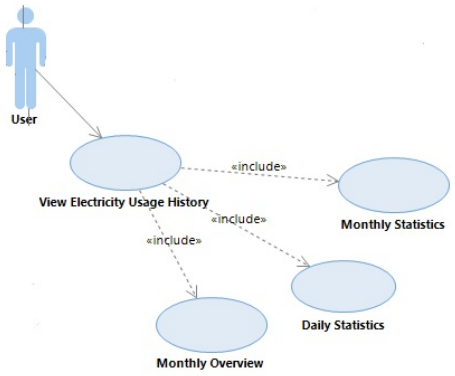


Figure 4.5: View Electricity Usage History use case

4.3.4 View Appliance Electricity Usage

Figure 4.6 illustrates the appliance electricity usage use case, which includes other two use cases. Appliance view allows the user to view specific date range electricity usage and appliance duration within the current week.

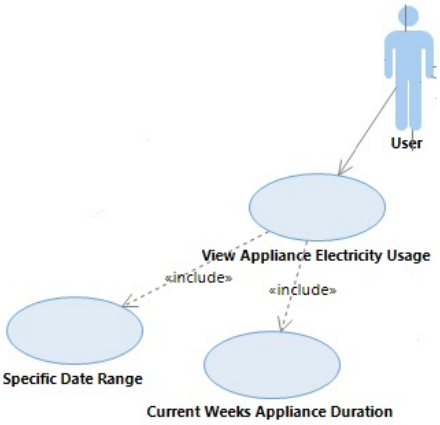


Figure 4.6: View Appliance Electricity Usage use case

4.3.5 Set Power Consumption Targets

Figure 4.7 illustrates the goal setting use case, which allows the user to set a monthly target, reset it at any time and view monthly statistics and usage progress.

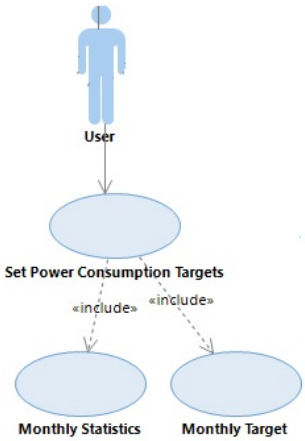


Figure 4.7: Set Power Consumption Targets use case

4.3.6 View Help

Figure 4.8 illustrates the help use case, this use case provides the user with the brief description of each component of the web application.

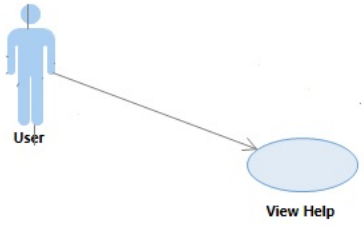


Figure 4.8: View Help use case

4.3.7 Usage Comparison

Figure 4.9 illustrates the comparison use case, which allows use to view their usage in comparison to the average users usage. It also allows for up to three day usage comparison.

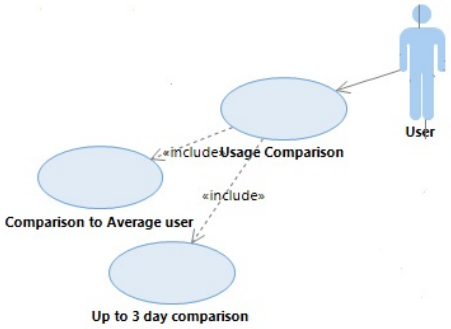


Figure 4.9: Usage Comparison use case

4.4 Database Design

This section will cover the database design evolution over the design and development stages of the project. There are a number of key factors to consider before designing a database. The main key aspect of the database design would be to sore obtained data from electricity power meters and be able to do calculations, analysis with it and then provide the user with a feedback via web application. This means there will be an API between database and user interface. The database should be designed so that API could easily query database, do calculations and respond to the user requests. User, meter and appliance details should also be taken in consideration when designing database.

4.4.1 Initial Database Design

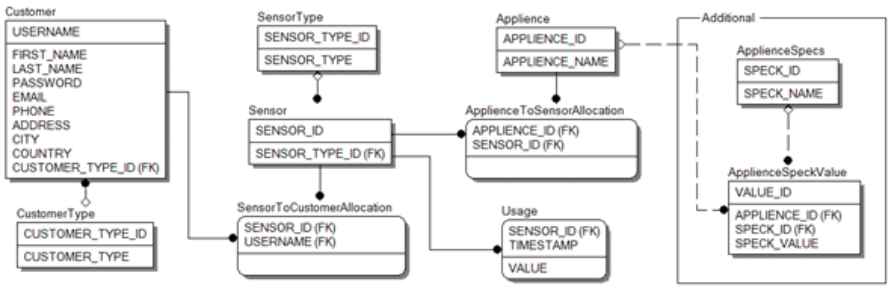


Figure 4.10: Initial ERD Diagram

Figure 4.10 represents the initial database model, which did change in the later stage of the project, due to having excess tables. Customer table stores all the users details, Sensor table stores the list of all the sensors, Usage table holds the obtained data from the electricity power meters. Usage table is

the fastest growing table within the database, it will contain thousands of records as data will be inserted every few seconds. Customer to Sensor Allocation table allocates meters to a particular customer and identifies what meter is in use. Appliance to Customer Allocation table allocates the appliance to a customer, the idea behind this was to allow the user to select an appliance and allocate a particular meter to it. This was removed in later stages of the development because it was unnecessary. Additional two tables where added to allow the ability to store additional details about different appliances.

4.4.2 Current Database Design

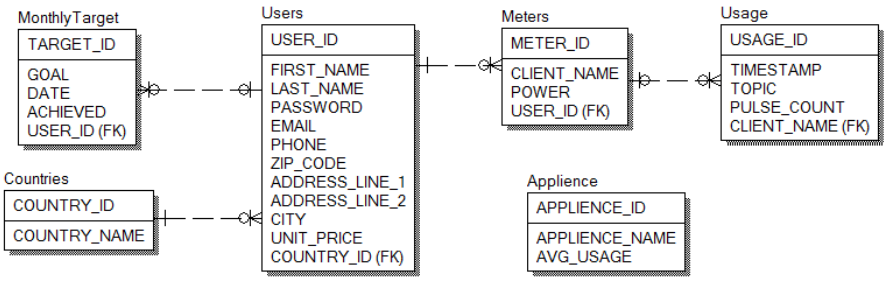


Figure 4.11: Current ERD Diagram

Figure 4.11 represents the current database model and in comparison to previous model has a significant number of changes. This model has fewer tables, which means less query joints and faster query execution. It still have Users / Customer table, which has additional attribute to store electricity unit price, Usage table has Usage_ID as a primary key, Appliances table has a new Average_Usage attribute, which indicates the average appliance usage. Sensor table is now Meters table, which has a Client_Name, Power and User_ID. Client_Name is the client name used to connect to the MQTT server and transmit obtained data from the power meters. It is an unique value, which identifies meter and user. Each meter can have only one user so, User_ID is added to the Meter table to indicate that meter is in use and assigned to a particular user. This eliminates need to have Client to Sensor Allocation and Sensor Type table. Power attribute indicates the consumption of electricity for one meter pulse, e.g. 1 pulse is 0.5 Wh or 1 pulse is 1 Wh. Newly added Goal Setting table allows the user to set a monthly targets and keep track of their progress.

4.5 Technical Architecture Diagram

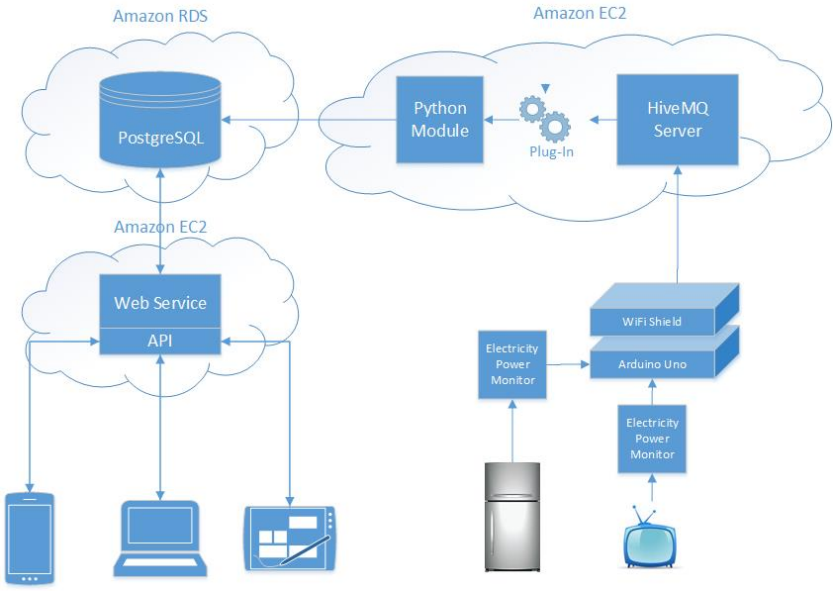


Figure 4.12: Architecture Diagram, created with Visio 2013

4.5.1 Electricity Power Monitors

The electricity power monitors are measuring household appliances electricity power consumption and transmit it to the Arduino Uno.

4.5.2 Arduino Uno with WiFi Shield

An application for Arduino Uno was developed to count the incoming pulses from the electricity power meters. This was achieved by using interrupts, a Interrupt Service Routine (ISR) was set up to handle pulse falling edge, so each time a falling edge is detected the pulse counter is incremented. The application also connects to the MQTT server via WiFi shield and transmits the pulse count every 30 seconds or any other specified time period. The application is not resetting the counter after each data transmit, but continuously incrementing. In case of server crush the data will still be in the application and will be transmitted to the server when server is back up and running.

4.5.3 MQTT Server

The MQTT server is hosted on Amazon EC2, which accepts the published data from the Arduino Uno and broadcast the raw data to the subscribed

clients, to the specific topic. The MQTT Message Log plugin stores the client communication details in the log files.

4.5.4 Python Module

Is an application written in python, which reads log files and extracts necessary data: time-stamp, client, topic and pulse count. From extracted data an insert statement is created and inserted in to the PostgreSQL database. Python module is a constantly running application, which constantly reads log files and fills up the database.

4.5.5 PostgreSQL Database

Is a permanent storage for the incoming data from the MQTT server which is hosted on Amazon RDS. The database will not only store the usage of household electricity consumption but also customer details and any other necessary data.

4.5.6 Web Service

An API written in python by using a Flask micro framework, which will handle client calls and interact with PostgreSQL database.

4.5.7 The Client

The user will be able to access the feedback system via any device, developed by using AngularJS Material design library and any other supporting libraries.

4.6 User Interface Mock-Up

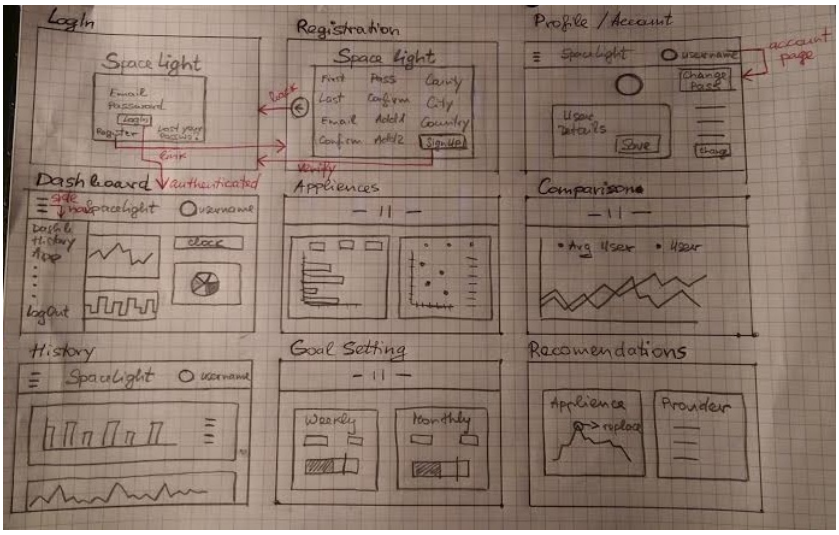


Figure 4.13: User Interface Screen Mock-Up

Figure 4.13 illustrates high level design of all user interface screens. User will be able to register, log in and then have an access to the application content. The application will mainly consist of graphs and statistics.

4.7 Conclusion

This chapter described the chosen methodology, use cases, database design, architecture diagram and user interface mock-up. Next chapter will describe the implementation of mentioned project components.

Chapter 5

Implementation

5.1 Introduction

This chapter describes the implementation of each project component and how they are combined together to create a fully functional and complete system. It will be divided into the following sections: Data Acquisition, Data Transfer, Data Storage, API and The User Interface.

5.2 Data Acquisition

To measure household appliance electricity power consumption, an electricity power meters were used: Siemens 7KT1 and XTM18SA. At first both meters were joined together and simultaneously measuring the same appliance. At later stage of the project meters were split and this allowed for more appliances to be concurrently measured. Figure 5.1 illustrates electricity meters after splitting them apart and making them more secure because they were placed within the sealed boxes.

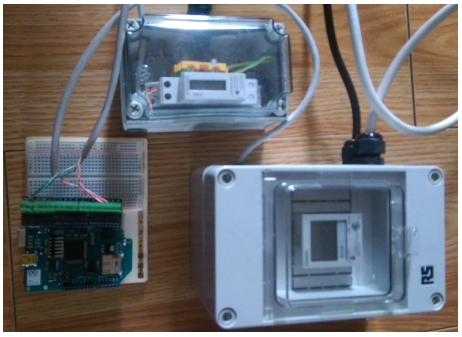


Figure 5.1: Both meters with Arduino Uno

5.2.1 Arduino Environment

To gather and transport electricity power consumption of household appliances an Arduino Uno micro controller was used. To allow Arduino program development and upload to Arduino board, an Arduino IDE was downloaded from official Arduino website and installed. Arduino board was connected to the computer via USB cable and driver installation process began. Windows was supposed to start the driver installation automatically but it didn't, so manual driver installation took place. On Windows 8 Control Panel ->Device Manager was opened and under Other devices section "Update Device Software" was selected, then computer was Browsed for the driver. It was located in Program Files (x86) ->Arduino Uno ->Drivers folder. Once the folder was selected the driver was installed. It can be viewed in Control Panel ->Device Manager under Ports (COM & LPT) section, in this case port name was Arduino Uno (COM3). This indicates that driver was successfully installed, now Arduino IDE can be opened and sketch development can start.

5.2.2 Arduino Sketch

To obtain electricity power consumption of various household appliances previously mentioned meters were attached to the Arduino Uno board and an Arduino sketch / program was written to detect and record incoming pulse from the meters. Arduino pin 2 and pin 3 were set to be an input pins to receive a pulse from the meters. Both pins are pulled up, to a known value, this ensures that pins are in the same state if no changes have occurred, otherwise they might float to low or high value and be unstable. Interrupt Service Routine (ISR) is set up to handle pulse falling edge. So every time an interrupt occurs the pin is pulled down and pulse is recorded. Interrupt 0 is set for pin 2 and interrupt 1 is set for pin 3. Every time an interrupt occurs a pulse counter is incremented accordingly to the interrupt. LED Pin on Arduino Uno board is set to be an output pin and works as an indicator, on the incoming interrupt the LED will blink. Figure 5.2 illustrates how pins mode and ISR is set up.

```
// the setup routine runs when powered on or when you press reset:
void setup()
{
  // set the LEDPin as an output
  pinMode(LEDPin, OUTPUT);

  // Sets up two ISRs for pulse detection
  // set pinMode for interruptPin to INPUT and pull high
    pinMode(interruptPin, INPUT_PULLUP);
    pinMode(interruptPin2, INPUT_PULLUP);

  // set up the Interrupt Service Routine (ISR) to handle pulse falling edge
  // interrupt 0 is on Arduino Uno Pin 2
  // interrupt 1 is on Arduino Uno Pin 3
  // every time a falling edge is detected the funtcion processPulse will be called
    attachInterrupt(0, processPulse, FALLING);
    attachInterrupt(1, processPulse2, FALLING);
```

Figure 5.2: Pin and ISR set up

Once ISR is set up, the WiFi shield or Ethernet shield is added to the Arduino Uno to connect to the Internet. For both shields a solution was implemented, which led to two separate Arduino programs and the only difference between the two sketches was the method of connection to the Internet.
PubSubClient library was added to both programs. This library provides a client for Arduino to do simple publish/subscribe messaging with MQTT server. In this project the client is set up to publish obtained pulse count from the electricity power meters. To create a client a host name / IP address, port and client name is necessary. Port number is 1883 or for web socket connection it is 8000. Once Arduino is connected to the Internet and MQTT client is set up, the connection to the MQTT server can start. To send the pulse count to the server, a topic need to be added to the message, in this applications a topic is an appliance name, that is plugged into the meters power strip. A message to the server is send every 10 seconds but default connection duration is 15 seconds, afterwards connection is dropped. Each time message is delivered the pulse count is reset back to 0, this allows for easy calculations in the later stages of the project. To ensure secure counter reset, the message delivery is checked before a counter is reset. The message is send for each meters pulse count. "Each message can be published with one of three quality of service levels (QoS). These levels are associated with different guarantees. A message send with level 0 doesn't have a guarantee at all, it implies fire and forget. Level 1 guarantees that the message will at least arrive once, but can arrive more than once. Level 2 is the most sophisticated choice, which guarantees that the message arrives at the destination exactly once. The choice of QoS is a trade-off, between protocol overhead and the guarantee that the message arrives, because ensuring QoS 2 is using more bandwidth than QoS 0." [31] Arduino client only supports QoS 0.

Figure 5.3 illustrates connection to the MQTT server and meter 1 pulse

count publication, in this case it was Kettle pulse count in last 10 seconds.

```
void uploadData()
{
  // MQTT connectiom - by Sanita
  if (!client.connected())
  {
    client.connect("131313");
    client.publish("FYP", "Connected");
    if (!client.connected())
    {
      Serial.println("Not connected");
    }
  }
  // pulse count upload to the server - by Sanita
  message = dtostrf(totalPulses,0,0,message_buffer);
  boolean meter1_response = client.publish("Kettle", message);
  // check if message was published, if yes, reset the counter - by Sanita
  if (meter1_response)
  {
    Serial.println("Meter 1 data published!");
    totalPulses = 0;
  }
```

Figure 5.3: MQTT connection and message transfer

Once Arduino program is complete and verified it is uploaded to Arduino Uno via USB cable, then meters are connected to the Arduino Uno and appliances are plugged into the meters power strip. Then appliances are switched on and every time an interrupt occurs the counter is incremented and every 10 seconds the message of pulse count is send to the server. Siemens 7KT1 one pulse represents 1 Wh and XTM18SA one pulse represents 0.5 Wh. This meter information is stored in the database and is used to calculate correct appliance power consumption in a later stage of the project.

Figure 5.4 illustrates the raw data / pulse count of Kettle and Fridge on the phone, this is achieved by having Android MyMQTT app on the phone. This app allows the user to publish or subscribe to any MQTT broker. MQTT inspector app can be obtained for iOS phones. HiveMQ web socket client can also be used to subscribe or publish messages to a particular topic.

Figure 5.4: Raw data viewed on the phone

5.3 Data Transfer

To transfer data from Arduino Uno to the server and then to the database, MQTT protocol was used. It is a machine-to-machine (M2M) connectivity protocol. "It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers." [28] To use MQTT protocol a HiveMQ MQTT server/broker was supposed to be installed on AWS EC2 with enabled Web-sockets. This allowed the communication between many MQTT devices as MQTT broker is the central server for the M2M communication so therefore all the messages go through it.

5.3.1 EC2 instance setup

Undertaken steps to set up EC2 instance:

1. Create an AWS account on the official website.
2. After logging in select Services and then choose EC2. This will open EC2 dashboard, displays the overview of the resources.
3. Press "Launch Instance" button to create new instance
4. Select an AMI from the list, in this case Ubuntu Server was selected
5. Select an instance type, configure an instance, add storage and tag an instance.
6. Configure the security groups as illustrated in Figure 5.5

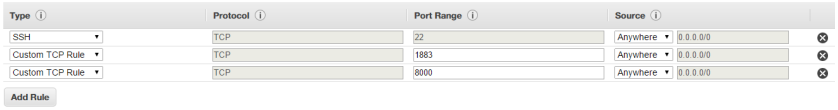


Figure 5.5: EC2 instance security configuration screen-shot

7. Review and Launch EC2 instance, Figure 5.6 illustrates the launched instance overview.

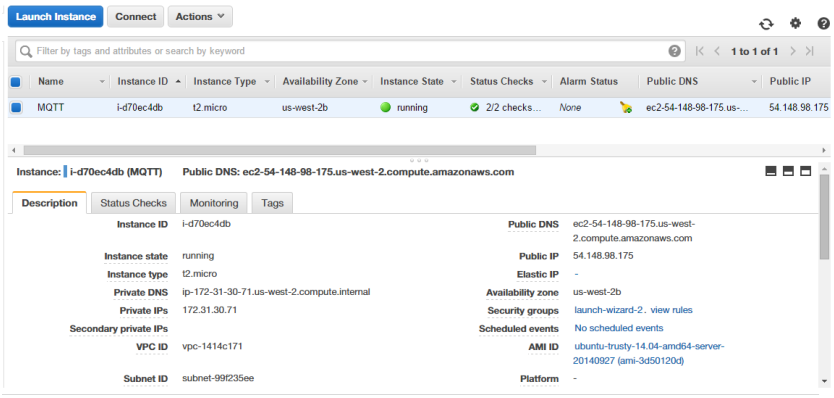


Figure 5.6: Launched EC2 instance overview

8. Save Amazon AWS key pair, a .pem file

5.3.2 PuttyGen and Putty setup

Putty is used as a secure way of connection to the newly created EC2 instance. "PuTTY is a free implementation of Telnet and SSH for Windows and Unix platforms, along with an xterm terminal emulator." [47] PuttyGen is used to convert .pem file obtained from AWS into a Putty private key. Figure 5.7 shows the PuttyGen main screen. Putty and PuttyGen can be downloaded from the official website Open PuttyGen and simply upload the (.pem) file and select "Save private key", this will generate (.ppk) file - the private key, keep it safe.

Figure 5.7: PuTTyGen main screen

Once private key is saved open Putty and perform steps:

1. In Category section under Connection select SSH
2. Select Auth and then "Browse" for the saved private key (.ppk) file
3. Go back to Session screen
4. Enter Host Name - EC2 instance Public IP
5. Enter port 22
6. Ensure connection type is SSH
7. Connect to the instance by pressing "Open" button
8. User-name: Ubuntu
9. Password: security key password

Figure 5.8 illustrates the main screen of the Putty and the projects MQTT server EC2 instance Public IP address.

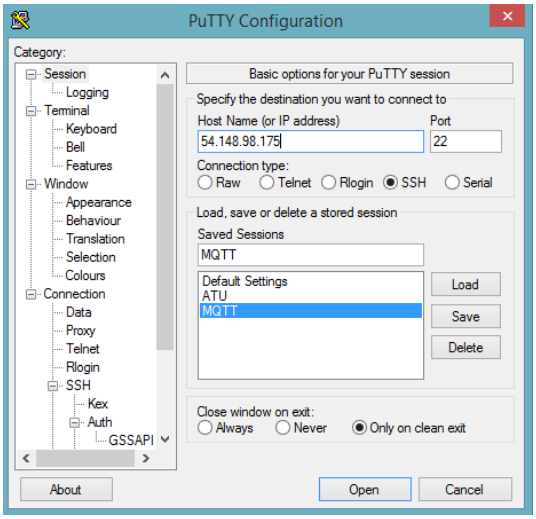


Figure 5.8: Putty main screen

5.3.3 Install HiveMQ on EC2 instance

1. First install Java

   sudo apr-get update

   sudo apt-get install openjdk-7-jre-headless unzip

2. Check if java is installed

   Java -version

3. Install HiveMQ

   wget –content-disposition
   http://www.hivemq.com/downloads/releases/latest

   unzip hivemq-1.x.x (x - is version)

   cd hivemq-1.x.x - go to the HiveMQ directory

4. Configure HiveMQ so that websockets would be enabled

   Open conf/configuration.properties file in vim

   Set websockets.enabled = true

   Set websockets.port = 8000

   Save the file

5. Run the HiveMQ server in the background

   ./bin/run.sh &

Figure 5.9: HiveMQ configuration.properties file code snippet to illustrate where to enable websockets

Figure 5.9 illustrates HiveMQ configuration file where to enable websockets. At first MQTT server was running as background process but that didn't ensure 24/7 service so MQTT server was set to run as a service.

5.3.4 Run HiveMQ as a service rather than a background process

1. Reinstall HiveMQ in /opt directory
2. Execute below commands in the sequence:

   cd /opt

   sudo wget –content-disposition http://www.hivemq.com/downloads/releases/latest

   sudo unzip hivemq-1.x.x.zip

   sudo ln –s /opt/hivemq-1.x.x /opt/hivemq

   sudo useradd –d /opt/hivemq hivemq

   sudo chown –R hivemq.hivemq /opt/hivemq-1.x.x

   sudo chown –R hivemq.hivemq /opt/hivemq

   sudo cp /opt/hivemq/bin/init-script/hivemq-debian /etc/init.d/hivemq

   sudo chmod +x /etc/init.d/hivemq
3. Start HiveMQ daemon

   cd /opt/hivemq

   /etc/init.d/hivemq start
4. Test if daemon is running

   netstat -tapn

Before running the daemon ensure that port 8000 and 1883 are available. Guide that was used but modified can be viewed on HiveMQ official website. [31]

"Opt directory usually describes as for optional add-on software packages source, or anything that isn't part of the base system. Only some distributions use it, others simply use /usr/local." [29]

5.3.5 Created HiveMQ Server AMI

AMI was created by following a guide to create an AMI. [24] The image was created to provide other related projects with already set up HiveMQ server. For others to obtain the image, their AWS account number need to be added to the permission list as it is a private image. Once permission is granted AMI can be found in EC2 console in AMI's section. Simply filter down to private images and launch the instance. This will provide the user with fully functional Ubuntu with MQTT server on it. The documentation about creating EC2 instance, installing HiveMQ and running it as a service rather than background process was created and provided to the related project owners. The documentation is attached in the Appendix C

5.3.6 Installed MQTT Message Log Plug-in

To log all the HiveMQ server incoming and outgoing messages the MQTT Message Log plug-in was installed on EC2 HiveMQ server instance. It was downloaded from official source, [43] then it was unzipped in HiveMQ plug-in folder and server was restarted to ensure successful plug-in installation. The messages appeared on the Putty screen, to store the messages and be able to process them the messages were redirected to the log file - run.sh >>test.log &). This was done when HiveMQ was running as a background process, once HiveMQ was installed as a service, this was unnecessary as all log files were stored in /opt/hivemq/log directory.

5.3.7 Additional component

An additional component needed to be developed which would read the log files, extract necessary data and insert into the database. At first this component was written in bash script and consisted from 2 files. Bash script which was reading log files, extracting data: time stamp, client name, topic and the message itself. Once data was extracted SQL insert statement were created and added to the SQL file. Then the database connector red the file and added data to the database. This was all done manually, no automation was implemented. Figure 5.10 illustrates how the bash script was executed and connection to database was made.

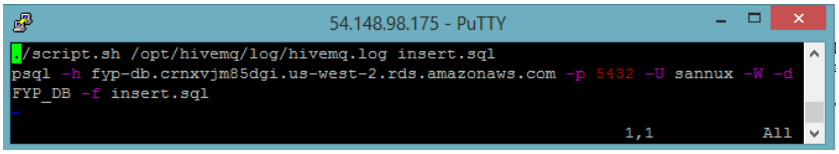


Figure 5.10: Script execution and database connection

In the later stage of the project automated python module was implemented, it was constantly reading log files by waiting for the new entries into the log files. File reading was rotated once the day changes and that's when file cursor was set back to 0, beginning of the file. Figure 5.11 illustrates the code that is constantly reading the log files and checking if current time is greater than today then it mean new day has started which indicates that cursor need to be reset back to 0, beginning of the file. It reads line by line and checks if line has string "sent a message to topic" and not contains "Connected" then the line is processed. The process method extracts the data into the dictionary and then data is inserted into the database. Figure 5.12 illustrates how the data is extracted from the obtained line. First split the line by spaces, then add necessary data into the dictionary. This code is very delicate as there are a specific rules that need to be followed otherwise the data extraction wont work and it can break the application. e.g. client name must be 6 characters long and it is an unique meter id in the database. Then the users are allocated to the meters, not other way around as there can be more than one meter for one user. Other rule would be the value must be an integer as meters will provide with the pulse count and nothing else. Once the data is extracted it is inserted in to the database. Models.py file contains the Usage table create used if tables doesn't exist in the database and init method that initializes the class object. Run.sh script contains the log file path, environment source and nohum, Linux command used to start the application and not let it to terminate if parent process is terminated.

Figure 5.11: Constant log file reader

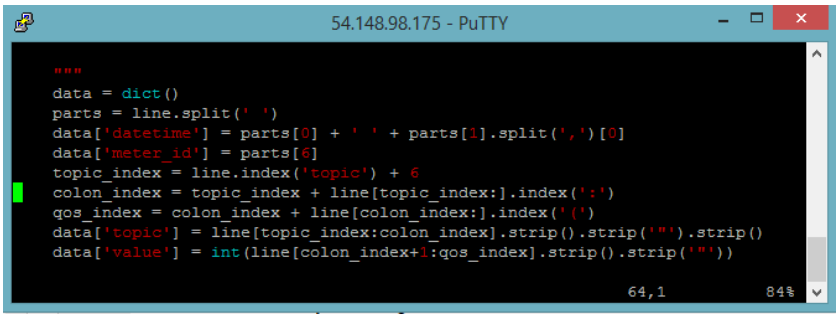


Figure 5.12: Extract necessary data from the log file obtained line

Python module environment was set on MQTT server:

1. install pip - sudo apt-get install python-pip
2. install environment - sudo pip install virtualenv
3. installed app dependencies - sqlalchemy, datetime and any other necessary libraries.
4. write run.sh script
5. run script ./run.sh

5.4 Simulator

Not all data could be obtained from the meters as only two meters were at my disposal for the project. The simulator was written in bash script to simulate more data and ensure that enough data is in the database to display various graphs and statistics. Figure 5.13 demonstrates the bash script that simulates a particular appliance. Script accepts appliance name,

its power consumption and the power meter hourly rate. e.g. ./simulator.sh Kettle 2000 1000, where Kettle is the appliance name, 2000 is the appliance power consumption in Wh and 1000 is power meter rate in Wh, which means one pulse equal to 1 Wh. This script then calculates how ofter the pulse should be repeated, in this case it will be every two seconds, just like the real Kettle if it would be plugged in to the power meter power strip. In this scenario the application will sleep every two seconds and then send a pulse to the named pipe also known as FIFO. This is used because simulator will send the data to the pipe and connector bash script will read the pipe and send the incoming pulses to the MQTT server. This ensures that rest of the project stays consistent and no specific changes need to be changes to accommodate the simulator. The simulator simply replaces the meters and the connector replaces the Arduino Uno.

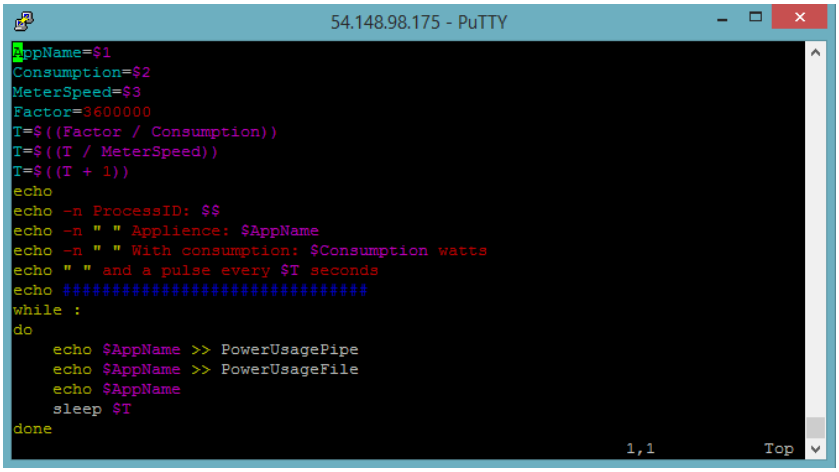


Figure 5.13: Simulator.sh

Figure 5.14 demonstrates the connector.sh script that is used to read the named pipe and transmit pulse count to the MQTT server. The script is constantly reading the pipe, incrementing the pulse count and using mosquito client to publish a message to the server. Mosquito pub is a simple MQTT client that publishes the message to the server and exits. If message was successfully send to the MQTT server, then a pulse count is reset to 0 and next message from the pipe is processed and transmitted.

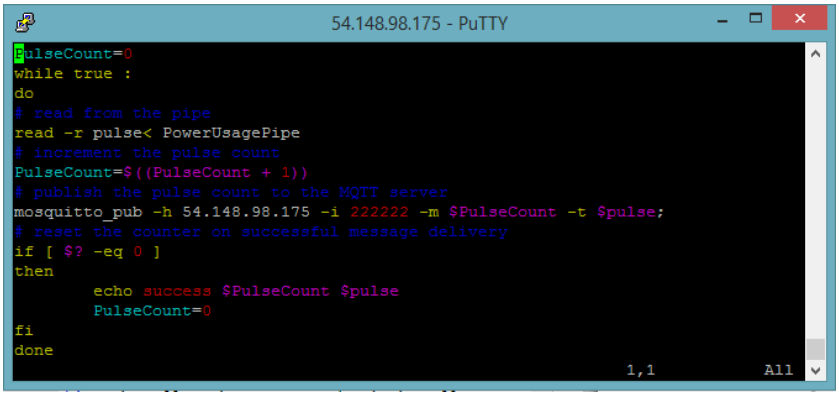


Figure 5.14: Connector.sh

5.5 Data Storage

For the project PostgreSQL database was used and it was hosted on Amazon RDS (Relational Database Service) and pgAdmin III client application was used to connect to the Amazon RDS instance from the Windows 8 machine.

5.5.1 Amazon RDS database set up

Undertaken steps to launch database instance:

1. Log into the Amazon AWS account
2. Select Services and then choose RDS. This will open RDS dashboard.
3. Press "Launch DB Instance" button to create new instance
4. Select PostgreSQL database
5. Select free RDS usage tier, so this means no production support
6. specify database details - free RDS usage tier components were selected
7. add advanced settings if necessary
8. launch the instance

pgAdmin III client application was used to connect to the instance, the test table were created, dummy data was inserted and select statements were executed to familiarize with the application. Usage table was the first table created and it was instantly tested by dummy data send from Arduino Uno via MQTT server hosted on Amazon EC2. The scripts inserting data into the database were executed manually but at later stages of the project this was made to be automated, it has been mentioned before. Tables were not created via the pgAdmin III client app but via the API, this ensures that API and database work well together and tables can be created on the go if the database changes. This will be discussed in greater detail in the API section as it is a part of API implementation.

RDS instance was used in production and for data storage, for development purposes a local database was created via pgAdmin III client application. This was done to ensure fast database connection, data insertion and selection, this database contained a dummy data but it had a production database structure. It ensured that once API is pointed to the production database there wouldn't have any differences.

5.6 API

An API was implemented by using a Flask, Python micro framework, which handle client requests and interact with PostgreSQL database.

5.6.1 Project Structure and Environment

JetBrains PyCharm IDE was used to develop API as it is one of the best Python and web development code editors. It provides the web development framework support, it has great build in development tools, examples would be SQLAlchemy, Git and others. At first the project didn't have a specific structure as it consisted from only a few python files but once other files got added to the project, a circular dependency was introduced during the module imports. Circular dependency means that two modules depend on each other. e.g.

module x - import y

module y - from x import test

To avoid circular dependency extract test component and move it to another module or place the import y at the end of the module x. To resolve the issue a project structure was applied, virtual environment was created and new directories was added. Figure 5.15 illustrates an API project structure and all it's components.

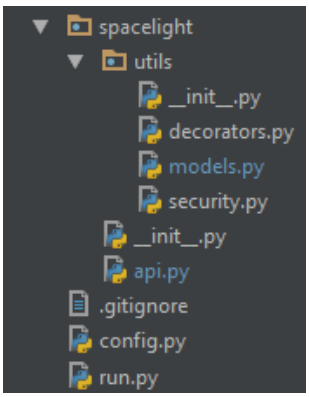


Figure 5.15: API project structure

__init__.py files are added by the default to any newly created directory, they are used to initialize the package. Spacelight directory init file sets up the application by importing Flask and SQLAlchemy packages, configuring the application and importing models and controllers. config.py file keeps the Flask applications configuration for different stages of the project, for example production, development, testing and others. This file also stores the secret key that is used to generate tokens. run.py file is created to execute the application by using a Flask_Script extension, it includes running a development server, scripts to set up the database and other command line tasks that belong outside the web application itself.

5.6.2 Models

Model represents the permanent storage of the data used in the MVC system. In the case of the API, the models are used to represent tables from PostgreSQL database. models.py file declares all the models and each declared model has class methods, which are used to query data from the database, for example create a new user, select a particular user based by the user ID, perform an update on a particular record withing the database. Figure 5.16 illustrates the usage model declaration, this is the fastest growing table in the database, it contains more than hundred thousand records, as new record is inserted every few seconds. It also illustrates every models basic declaration structure: provide table name, add all the table columns, assign each column with data type and length, if applicable.

```
142  class Usage(db.Model):
143      __tablename__ = 'usage'
144      id = db.Column(db.Integer, primary_key=True)
145      topic = db.Column(db.String(256))
146      client_name = db.Column(db.String(256))
147      timestamp = db.Column(db.String(256))
148      pulse_count = db.Column(db.Integer)
```

Figure 5.16: Usage model declaration

If the model contains a foreign key, then this would be an example of foreign key declaration:

user_id = db.Column(db.Integer, db.ForeignKey('users.id'))

users = db.relationship('Users', backref=db.backref('posts',

lazy='dynamic'))

This example illustrates how ID from users table is a foreign key in the meters table. Backref is a simple way of declaring a new property in Users class, that can be used to directly access the user by the user ID. Lazy is defined when SQLAlchemy loads data from the database.

Overall API have a six models: Users, Usage, Meters, Goal, Appliances and Countries. Each model represents a particular database table, and

database structure has been discussed in the design and architecture section. The upcoming section will give a brief description of each model and the list of class methods.

Users model

Users model represents the users table in the database, this table contains all the user details and class methods allow for user detail querying. The model contains nine class methods:

- Create a new user
- Get user by token - the web application developed as part of this project uses token-based authentication that will be discussed in the token-based authentication section of the report.
- Update user details
- Change user password
- Upload, get and delete profile picture
- Get and set electricity unit price

Usage model

Usage model represents the usage table in the database, the table contains the data obtained from the measured household appliances or simulated appliance usage. This model contains sixteen class methods and all of the methods perform some type of select statement on the usage table. Examples of usage class methods would be: select current years appliance usage, current month overall usage, particular date range appliance usage, hourly overall usage breakdown and others. The Usage model contains the list of class methods that are used to obtain data for various user interface graphs and statistics.

Goal model

Goal model represents the monthly targets table in the database, the table contain all the user monthly target history. The table contains only one user target each month therefore there can't be more than one target each month. The model contain six class methods:

- Set, reset and get target
- Get previous month target for particular user
- Check if target is achieved
- Get target statistics for particular user - obtain all the user target information

Meters model

Meters model represents the meters table in the database. The table contains the list of all the meters, each meter can be assigned to exactly one user. The model doesn't have any class method implementation, it only consists from model declaration, class object initialization and printable data representation.

Appliances model

Appliance model represents the appliances table in the database. The table contains the list of typical household appliances and their average power consumption obtained from Electric Ireland appliance calculator app and from the Daft Logic - Appliance Power Consumption. [46] Data was manually inserted into the database.

Countries model

Countries model represents the countries table in the database, the table contains the list of all countries, obtained from Umpirsky Software Development Blog [59] in CSV file format. This file was modified by removing a few countries that didn't support UTF-8 as they contained unsupported characters and by adding an ID column to the file. This modified file was imported into the countries table via pgAdmin III client tool. The model contains one class method, which gets the list of all countries.

5.6.3 Database table creates

PostgreSQL database was hosted on Amazon RDS but tables were never created via simple SQL create statements to ensure that application can accept any database connection and initialize it based by the declared models within the Flask application. To issue a CREATE statement create_all() function is executed via Python console. This method first checks if the table is already created, if not it will create the table otherwise it will move to the next table and so on until all the tables are created. This way both development and production databases were initialized.

5.6.4 API calls

api.py file contains all 32 API call implementation. Table 5.1 and Table 5.2 contains the list of all the API calls developed for the project. Both tables provide API call, HTTP method, brief description and parameters, request body content. All related API calls to obtain user data, user usage, goal setting or anything related to the user requires a request header should contain a token. Token-based authentication is discussed in the following section.

API call	HTTP methods	Description	Parameters
/login	POST	Verifies user login details	Email Password
/register	POST	On valid user details create a new user	Registration form input fields
/countries	GET	Obtains the list of all countries	-
/user	POST	Update user data	All account form details
/user	GET	Obtain user data	-
/password	POST	Change the user password	Old Password New Password
/img	POST	Upload profile picture	Picture
/img	GET	Obtain profile picture	-
/img	DELETE	Delete profile picture	-
/goal	POST	Set target	Target Month
/goal	GET	Get current and previous month target	-
/reset_target	POST	Reset the current target	New Target
/check_target	GET	Check if user has achieved monthly target	-
/target_stat	GET	Get target statistics for particular user	-
/current_hourly_usage	GET	Current overall usage per hour	-
/hourly_usage	POST	Hourly usage for particular day	Date
/current_appliance_usage	GET	Current day appliance usage, group by appliance	-
/appliance_duration	POST	3 day appliance duration and avg. user comparison	From Date To Date
/appliance_usage	POST	Appliance usage in particular date range	From Date To Date
/current_day_usage	GET	Current day overall usage	-

Table 5.1: First part: API call list

API call	HTTP methods	Description	Parameters
/yesterdays_ usage	POST	Yesterdays usage	From Date To Date
/daily_ monthly_ usage	POST	Usage for particular month grouped by days	Date
/monthly_ usage	POST	Particular month overall usage	Date
/avg_user_ monthly_ hour	POST	Average hourly consumption for particular day	Date
/user_ monthly_ hour	POST	Monthly usage grouped by hour and date	Date
/year_ appliance_ usage	GET	Current years appliance usage grouped by date	-
/weekly_ appliance	POST	Particular weeks appliance usage	Weeks day 1 Weeks day 7
/price	GET	Obtain users electricity unit price	-
/price	POST	Set users electricity unit price	New price
/monthly_ statistics	GET	Obtain users monthly statistics	-
/daily_ statistics	GET	Obtain users daily statistics	-

Table 5.2: Second part: API call list

5.6.5 Security: password hashing

Security usually is applied last and not enough time is allocated for it but it is very essential as it keeps the user data safe and protected. It is very hard to develop a secure application as there is always someone smarter and willing to find weak points within the application and exploit them. To add a security to the application, password hashing was applied when new user is registered, all the passwords stored in the database are hashed. To do password hashing Python package PassLib was used, SHA-265 hashing algorithm was used to encrypt the password. PassLib official website provides with details how pbkdf2_sha256() function encrypts the password. Figure 5.17 illustrates how password is encrypted and verified, encryption is applied on the user registration and verification on the user log-in.

```python
# use SHA-256 encryption
hashing = CryptContext(['pbkdf2_sha256'])


# encrypt password function
def encrypt_password(password):
    return hashing.encrypt(password)


# check if password is correct
def verify_password(encrypted_password, password):
    return hashing.verify(encrypted_password, password)
```

Figure 5.17: Encrypt the password and verify the password functions

5.6.6 Token-Based Authentication back-end implementation

Token-Based Authentication general concept is to all the user to enter lo-gin details, email / user-name and password in order to obtain a token which allows them to access / request a specific resource, without using their lo-gin credentials. Token is generated for a specific time period and is used to access a specific user related resource, from the remote site. Token need to be delivered in some sort of way, for example within the header, GET or POST request or in a form of cookie, then the remote site can extract token and determine if access is granted to the resource or not. AngularJS web application is using Token-Based Authentication, a token is generated in the API. Python package PyJWT is used to generate JSON Web Token (JWT). It is a compact URL-safe means of representing claims to be transferred between two parties and it is encoded as a JSON object that is digitally signed using JSON Web Signature (JWS). The token is generated on /login API call when user enters a valid log-in details. Each generated token represents a specific user, token is generated by using user id and secret key, stored in config.py file. Figure 5.18 illustrates two methods: generate token by using user id and secret key, and other method to decode the token to obtain the user id and verify the user.

```python
#  generate users token
def generate_token(payload, secret_key):
    return jwt.encode(payload, secret_key)


# decode token - used to verify user
def decode_token(token, secret_key):
    try:
        return jwt.decode(token, secret_key)
    except (jwt.ExpiredSignature, jwt.DecodeError):
        return None
```

Figure 5.18: Token generation and decode

Decode method is used to verify each API call made by the user, to ensure the token is in each request header a decorator was made. Decorator essentially is a wrapper that dynamically alter the functionality of a function, method or class without having to directly use subclasses. This is a good way of creating a function that never changes and apply it as many times as necessary. Decorator changes the behaviour of the function that it decorates without the need to modify the function itself. The decorator is essentially an ideal solution to ensure that each user resource request header contains a valid token. Figure 5.19 illustrates the @token_required decorator declaration.

```
def token_required(func):
    """
    create token decorator, used to ensure user request contains header with token
    otherwise request wont be fulfilled
    """
    @wraps(func)
    def decorated(*args, **kwargs):
        verify_token()
        return func(*args, **kwargs)

    return decorated
```

Figure 5.19: Decorator declaration

Figure 5.20 illustrates the function used to validate the header and it must be declared before decorator declaration. At first function obtains the request Authorization header, then it checks if the header exists, if not 401 error is raised indicating that the header is missing. Then header body is split in to two parts as it consist from jwt and token itself with space in between. If first part isn't jwt then unsupported authorization type error is raised. If header contains only one part or more than two parts then error is raised because there can only be exactly two parts, one for jwt and one for the token itself. If header passes all the validation the user is validated, if user is not found then invalid token error is raised otherwise the API call can be processed.

```python
# verify request header and token
def verify_token():
    header = request.headers.get('Authorization', None)

    # check if header exists
    if header is None:
        abort(401, description="Header missing")

    # split header into two parts by space - token cant contain spaces
    header_components = header.split()

    # check if header first part is jwt - JSON web token
    if header_components[0].lower() != 'jwt':
        raise abort(401, description="Unsupported authorization type")
    # check if both parts are in the header jwt and token itself
    elif len(header_components) == 1:
        raise abort(401, description="Token missing")
    # check if there is only two parts in header, if not then token is invalid as it cant contain spaces
    elif len(header_components) > 2:
        raise abort(401, description="Token contains spaces")

    # get the user by token from the request header
    user = Users.get_user_by_token(header_components[1])

    # if user is empty then tokan is invalid
    if not user:
        raise abort(401, description="Invalid token")

    # keeps objects for other methods
    g.user = user
```

Figure 5.20: Decorator function to validate request header

Figure 5.21 illustrates how @token_required decorator is applied to the API call to ensure valid token is passed before request is processed. The decorator is applied to all API calls required the valid token before it can process the request.

```python
@app.route('/user', methods=["GET"])
# cross_origin ensures the correct headers are added to the request
@cross_origin(headers=['Content-Type', 'Authorization'])
# token_required decorator ensures that token is added to the header and it is valid before request is processed
@token_required
def get_user_data():
    """ API call to get the user data and populate Account view
    """
    # extract token from the header and get request data
    token = request.headers.get('Authorization', None).split()[1]
    user = Users.get_user_by_token(token)
    user_data = jsonify({"first_name": user.first_name, "second_name": user.second_name, "email": user.email,
                         "add1": user.address_line1, "add2": user.address_line2, "city": user.city,
                         "zip_code": user.zip_code, "country": user.country})
    return user_data, 201
```

Figure 5.21: Decorator added to the API call

@cross_origin is predefined decorator obtained from the Flask-Cors package, it ensures that correct headers are added to the request. 'Authorization' header is needed for the token and 'Content-Type' header indicates the media type of the entire-body.

5.6.7 Conclusion

This section presented a details about the API implementation by the presented project, a Token-Based Authentication back-end implementation was also included within the section. In the User Interface section AngularJS

Token-Based Authentication front-end implementation will be discussed. Both implementations are separated because user interface project structure and basic concept need to be discussed first.

5.7 User Interface

AngularJS web application was implemented to provide the user with user friendly, responsive and intuitive user interface.

5.7.1 Project Structure and Environment

JetBrains Web Storm IDE was used to develop AngularJS web application because it is the smartest and most advanced Java-Script IDE. It is an ideal solution for client-side development and provides an excellent AngularJS support. At first the project didn't have a specific structure as it consisted from only a few HTML files, style sheet and a java script file. At that stage of the project index, log-in, registration and dashboard views were implemented by using only HTML, the layout and design was applied by the CSS style sheet. Java Script file contained the AngularJS app main module and it was used within the index.html page to identify the AngularJS application content. Nothing was right about the beginning of the web application implementation because there was no structure, no environment, conflicting libraries and it was not an AngularJS application. To fix all the issues and ensure correct project structure and environment, a Yeoman work flow was used. It comprises of 3 main tools Yo, Bower and Grunt. Yo provided with the basic structure of the project, it created the folder structure, added initial files and configurations. Bower was used to find and install any package / library that was necessary for the project. Grunt provided with task runner services, the commonly used commands was, grunt build - inserted all the dependencies installed by the bower within the index.html file. grunt serve command grants the access to the HTTP server and it can be accessed by http://localhost:9000/, the application was viewed within this page.

Set up Yeoman work flow

1. install node.js from official website
2. install npm - node.js package manager
3. open node.js cmd for Windows machine
4. npm install -g yo grunt-cli bower - install all the Yeoman tools
5. npm install -g generator-angular - installs AngularJS generator
6. yo angular - will create a new project

Figure 5.22 illustrates the web application structure provided by Yeoman. Overview of main components:

- app directory is the main project directory where are all the project components are located: images, scripts, views and styles. This is the directory containing the static app, that was developed by the project owner.
- bower.json file contains all the dependencies installed by the bower, file is used on deployment.
- bower_components directory contains all the installed dependencies
- node_modules directory contains all the modules of which the project depends on
- package.json file helps npm to manage packages
- Gruntfile.js configures project, it's tasks and plug-ins
- Karma is testing framework - not used within this project scope
- .gitignore file contains the list of project components that not supposed to be pushed to the repository: node_modules, .tmp, .sass-cache, bower_components and .idea
- dist directory contains processed files and is used as a root directory for web server

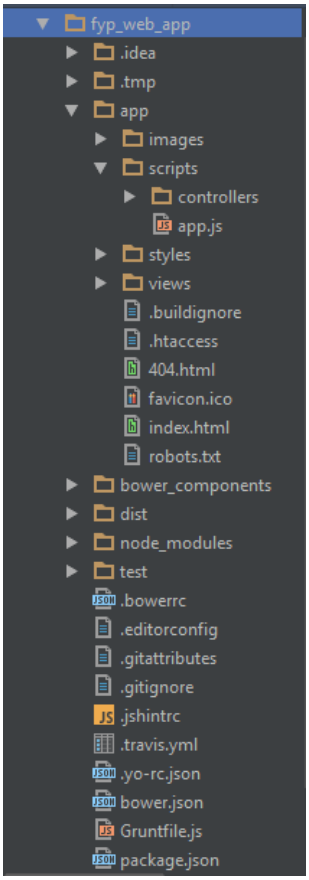


Figure 5.22: Web application structure

5.7.2 Single-Page Application (SPA)

A SPA concept was applied to the AngularJS web application to allow for more fluid user experience and removes the necessity to duplicate code applied to similar HTML pages. The project code is either loaded with a single page load or loaded depending on a user action and dynamic communication with API happening in the background. SPA concept is introduced within the index.html, the core HTML page of the application, as all the scripts, libraries and style sheets are loaded within this page and no other project HTML page contains any dependencies. This means all the scripts, libraries and style sheets are loaded only once, when the web application is opened or when web browser is refreshed. This decreases the performance on the application loading but once loaded it's overall performance is increased and reduces the render time. Figure 5.23 illustrates the index.html SPA concept, where div will load different views depending on the state of the application.

```
<!-- application content -->
<div ui-view layout="column" style="height:100%;width:100%"></div>
```

Figure 5.23: index.html page content div

index.html page changes three views depending on the state of the application: log-in page, registration page and the main page. The main page itself is using SPA concept because it changes six views determined by the state of the application. Each view will be discussed in the View section.

5.7.3 Client-side Dependencies

Bower was used to find any package / dependency necessary for the project. The command is bower install [name of the package] - -save, the save part ensures that it is saved and grunt build command can pick it up and insert dependency in to the appropriate place in the index.html page. The list of all the packages used within the web application:

- **angular** is front end web development MV* framework used to develop AngularJS application.
- **angular-animate** is animation library used on different components of the web application, for example ngView, ngHide, ngShow and others.
- **angular-aria** library provides support for common ARIA attributes: ngHide, ngShow, ngDisabled, ngClick, ngMessage, ngDblClick, ngModel
- **angular-image-crop** library was used to upload profile image, crop it and save it in the database.
- **angular-local-storage** provides the access to the local storage where token is stored.

- **angular-material** library was used to design the whole application, the library provides with set of tags that are used to replace the HTML tags or improve upon them. This library provides the developer with all the necessary features to create a professionally looking and responsive web application.
- **angular-messages** is a directive that is designed to show and hide messages based on the state of a key/value object that it listens on. Within the project scope it was used on the registration and log-in form to ensure correct input value was entered.
- **angular-ui-router** was used to allow for multiple views within the application routing need to be implemented and this library is used to do that.
- **angular-validation-match** was used to verify input fields matched with confirmation input field, for example on registration form to obtain a correct email address and password.
- **d3** is a library for manipulating documents based on data, within this project scope the library was used to support dimple graph library as it works based on d3.js.
- **dimple** is a simple charting API that uses d3 objects to create flexible and neat looking charts with very few lines of code.
- **material-design-icons** library is used to obtain the list of icons.
- **satelizer** is token-based authentication module for AngularJS with built-in support for Google, Facebook, LinkedIn, Twitter, Yahoo, Windows Live authentication. Attempt was made to do Google authentication but it wasn't successful as it clashed with application authentication.

5.7.4 Routing

To allow for multiple views the application needed to be configured to route HTML pages accordingly of the state of the application. stateProvider contains all the different states that application can be in: login, register, main.dashboard, main.history, main.appliances, main.goal, main.comparison, main.profile and main.help. Figure 5.24 illustrates few states of the application, on any other state of the application $urlRouterProvider will route the application to the log-in page. Each state has url - indicates what route needs to be added to the applications main URL to indicate the state of the application, templateUrl - indicates the path to the view, authenticate - indicates if authentication is necessary to access the state, controller - indicates what controller controls the view.

```
$urlRouterProvider.otherwise('/login');

$stateProvider
  .state('login', {
    url: '/login',
    templateUrl: 'views/login.html',
    authenticate: false,
    controller: 'LoginCtrl'
  })
  .state('register', {
    url: '/register',
    templateUrl: 'views/registration.html'
    authenticate: false,
    controller: 'registerCtrl'
  })
  .state('main', {
    url: '/main',
    templateUrl: 'views/main.html',
    authenticate: true,
    controller: 'mainCtrl'
  })
  .state('main.dashboard',{
    url: '^/dashboard',
    templateUrl: 'views/dashboard.html',
    authenticate: true,
    controller: 'dashboardCtrl'
  })
```

Figure 5.24: Few states of the web application

5.7.5 AngularJS Token-Based Authentication front-end implementation

The AngularJS service was implemented that is responsible for registering a new user, log-in and log-out registered user and store the generated token in the local storage. The service factory function is used to create an AngularJS service for the whole application. Declared factory can be injected into any component (controller, service, filter or directive) and it specifies a dependency on the service. Within this project scope the factory will be injected in to the log-in controller and registration controller. Figure 5.25 illustrates the factory initialization and the registration service declaration.

Figure 5.25: Authentication service factory declaration code snippet

Figure 5.26 illustrates the log-in service declared within the factory. User entered log-in details are sent to the API for verification and on successful response the token, user email, first and second name are added to the local storage via localStorageService. On error response the log out service is called and access to the application is denied.

Figure 5.26: Log-in service declaration withing the factory

Figure 5.27 illustrates how all the services are added together and returned as one service. This is done at the end of the factory, when all the services are declared.

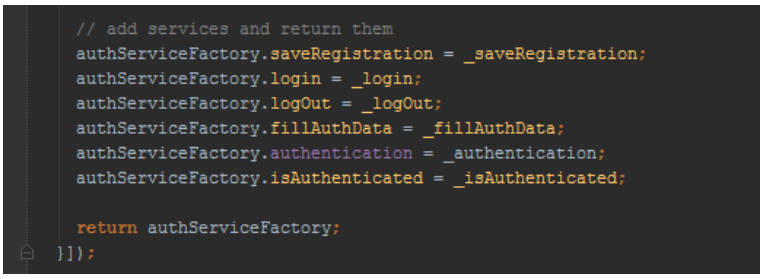


Figure 5.27: All services within the factory are added together and returned

The sample code for the Token-Based front-end Authentication was obtained from Bit of Technology website, the tutorial was written by Taiseer Joudeh. [37]

API call header

JWT token is embedded inside the API call, specifically within the Authorization header when user is requesting a resource. Figure 5.28 illustrates the header declaration used withing all user requests.

```
// header for the API calls
$scope.header = {"headers" :{"Content-Type" : "application/json",
  "Authorization": "jwt " + $scope.authentication.token}};
```

Figure 5.28: Request header declaration

5.7.6 MVC software architecture pattern

Model–View–Controller (MVC) is a software architectural pattern for implementing user interfaces. It divides the application into three components: model, view and controller, they are separate components as they have their own task to fulfill. Model represents any permanent storage, it can be database or file, the purpose of the model is to prepare data for other components to use or to insert data into the permanent storage. The models of the project are declared within the back-end of the web application, models.py file in the API. They have been discussed in the API section 5.6 under Models subsection 5.6.2.

Controller is the only component in the MVC software architecture pattern that interacts with other two components. It takes in an input and converts it to the command for the model or view. A controller obtains the data from the view and sends the request to the model to change the state of the model, for example by adding a new user on registration. On the other hand the controller can receive the response from the model and inform the view to show the confirmation or the information message.

The web application consists of nine controllers, each represents a view. An exception is main.js as it controls not only main.html but also help.html view because it is used within a dialog. Each view is either a static HTML page or generated response to a controller action. The web application consists of ten views.

Log-in view

Figure 5.29 illustrates the web application log-in view containing a simple log-in form used by user to verify their log-in details and access the application. Form contains email and password input fields, both fields are

validated and set to accept according input value. Figure 5.30 illustrates the validation of email input field. All the application input fields are validated in the same manner but each has it own valid input type.

On the log-in button click user entered details are passed to the authentication service, discussed in the section 5.7.5. Log-in service sends a request to the API, it validates the request and sends back the response. If details are valid the response will contain a token which then is placed in the local storage and access to the application is granted.

Figure 5.29: Log-in view

```
<md-input-container>
  <label>Email</label>
  <input type="email" name="email" required="" data-ng-model="user.email">
  <div ng-messages="loginForm.email.$error">
    <div ng-message="required">*</div>
    <div ng-message="email">Wrong Email Format</div>
    <div ng-message="match">{{emailMessage}}</div>
  </div>
</md-input-container>
```

Figure 5.30: Email input field validation

Registration view

Figure 5.31 illustrates the AngularJS web application registration page containing the registration form. All the forms fields are required and validated.

Figure 5.32 illustrates the data being obtained from the registration form and inserted into the dictionary. This dictionary then is forwarded to the registration service discussed in the section 5.7.5. Registration service then adds the dictionary to the body of request and sends it to the API, it validates the request and sends back a response. If details are valid the user will be routed to the log-in page to reenter their log-in details and authenticate them self's and grant the access to the application. Across the whole application the request body structure is in the dictionary / JSON format only.

Figure 5.31: Registration view

```
// get data from the form
$scope.registration = {first_name:$scope.first_name, second_name:$scope.second_name, email:$scope.email,
  password:$scope.password, address_line1:$scope.address1, address_line2:$scope.address2, zip_code:$scope.zipCode,
  city:$scope.city, country:$scope.country.country, account_type:$scope.account};
```

Figure 5.32: Data obtained from the registration form placed within the dictionary

Dashboard view

Figure 5.33 illustrated the web application dashboard view, it is the first page the user sees when entering the application. Dashboard displays the current day electricity usage overview in various graphs. The line chart illustrates the current days hourly usage, pie chart displays all the appliances and their percentage of overall usage. Figure 5.34 illustrates the code necessary to display a pie chart. Dimple.js is very simple and clear charting library powered by d3.js. All the web application graphs are created by using Dimple charts.

Date and time is also displayed within the dashboard. The bullet chart illustrates the current month usage in comparison to the set monthly target. Today in comparison to yesterday line chart illustrate the two day comparison. It informs the user if today's hourly usage has been increased or decreased in comparison to yesterday.

The three day appliance comparison bar chart is animated and displays duration of the appliance being switched on within the particular day. It also displays the appliance usage in comparison to the average appliance usage. This gives an user indication if the appliance consumes more than average appliance and can be a candidate for the replacement.

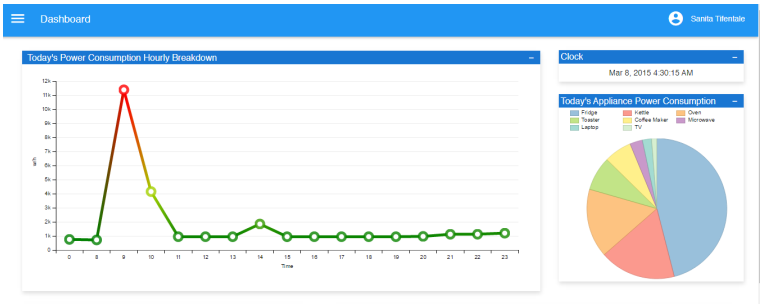


Figure 5.33: Dashboard view

```
/**
 * pie chart begins
 * obtained from: http://dimplejs.org/examples_viewer.html?id=pie_standard
 * changed to support the data
 */
//  responsive graphs width
var width = parseInt(d3.select('#pieChart').style('width'), 10) - 20;
var svg = dimple.newSvg("#pieChart", width, width);

var myChart = new dimple.chart(svg, $scope.appliances);
myChart.setBounds(40, 0, width - 60, width + 50);
myChart.addMeasureAxis("p", "Usage (Wh)");
myChart.addSeries("Appliance", dimple.plot.pie);
myChart.addLegend(10, 5, width, 200, "left");
myChart.draw();
$scope.pieChartLoading = false;
/**
 * pie chart ends
 */
```

Figure 5.34: Dimple pie chart

Side Navigation and Toolbar view

Figure 5.35 illustrates the side navigation and the toolbar. Side navigation routes the user across the whole application and the toolbar informs the user on the state of the application by displaying the selected page name within the toolbar. Toolbar also contains the user full name and the profile picture, which essentially is a link to the user account view. The side menu can be opened when menu icon, on the toolbars left hand side, is clicked otherwise it is hidden. To be able to toggle the side navigation the service was implemented (factory), this service contains the open, close and go services. Go service means change the view of the application. Figure 5.36 illustrates the partial code of side navigation service declaration and toggle side navigation service implementation.

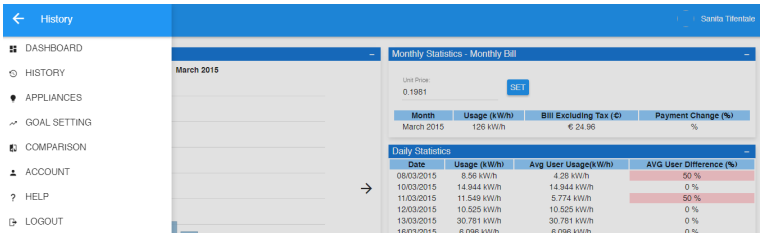


Figure 5.35: Side Navigation and Toolbar view

```
app.factory('navService', ['$log', '$state', '$rootScope',
  function($log, $state, $rootScope) {

    var navService = {};
    // open side navigation
    var _open_nav = function(mdSidenav) {
      mdSidenav('left').toggle()
        .then(function(){
          $log.debug("toggle left is done");
        });
    };
```

Figure 5.36: Code snippet to illustrate side navigation service declaration and toggle function

History view

Figure 5.37 illustrates the users overall usage history overview page. This page contains the bar chart which displays the current month daily usage but the user can also view the history by selecting the left hand side arrow. The arrow is disabled when there is no data found and the information message is displayed to inform the user.

The tables on the right hand side of the screen informs the user about their monthly and daily statistics. Within the monthly statistics user can set electricity unit price, which can be obtained from the electricity bill. Based by this unit price the monthly bill is calculated, the input field is validated by only accepts the numbers and floats with 0.4f precision. The unit price can be reset at any time and monthly statistics table will update the monthly bill amount. Payment change percentage indicates for how much the current month costs increased or decreased in comparison to previous month.

Daily statistics table displays the users usage and the average usage obtained by adding up all the users usage within that day and dividing the sum by the number of users. Percentage indicates for how much more or less user consumes than an average user.

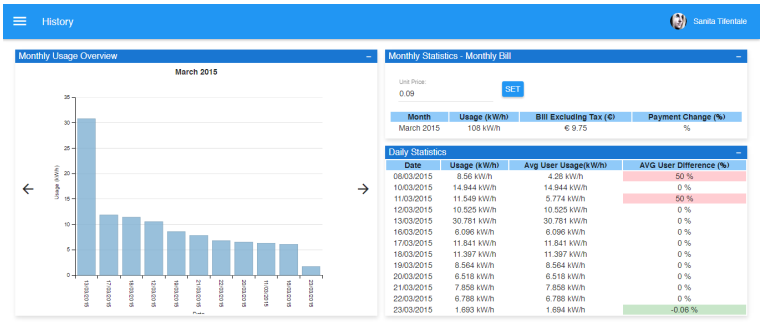


Figure 5.37: History view

Figure 5.38 illustrates the monthly statistics table implementation by using ngRepeat AngularJS directive. A template tables is created by adding necessary table headings and rows, then ng-repeat is set to loop through the passed in scope and assign cell values based by the key (acts as an index) and value {{stat.month}}. All tables within the application is created in this manner because it is dynamic way of creating table with unlimited number of rows.

```
<!-- table to display monthly statistics -->
<table flex layout-fill ng-show="!statLoading">
  <tr>
    <th>Month</th>
    <th>Usage (kWh)</th>
    <th>Bill Excluding Tax (€)</th>
    <th>Payment Change (%)</th>
  </tr>
  <tr ng-repeat="stat in statistics" class="text_middle">
    <td>{{ stat.month }}</td>
    <td>{{ stat.usage }} kWh</td>
    <td>€ {{ stat.payment }}</td>
    <!-- percentage highlights green when current month payment is lower than last month or red when
      other way around -->
    <td ng-class='{not_achieved_messages : stat.stat > 0, achieved_messages: stat.stat  < 0}'>
      {{ stat.stat }} %
    </td>
  </tr>
</table>
```

Figure 5.38: Monthly statistics table

Appliance view

Figure 5.39 illustrates the appliance view containing two graphs. Particular date range appliance usage graph allows the user filter appliance usage based by the selected date range. By the default the graph displays the current days appliance usage. The user can't select a future date, the calendar dates are disabled after the current day, to have a user friendly calendar. Current weeks appliance duration stacked bar chart displays the duration of each appliance being switched on every day of the week.

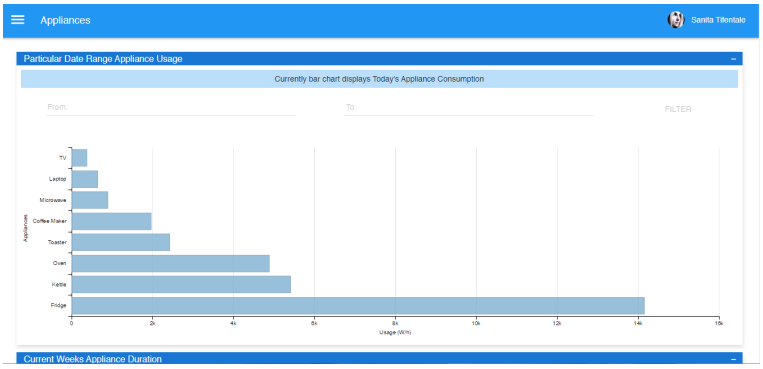


Figure 5.39: Appliance view

Figure 5.40 illustrates the code snippet of current weeks appliance duration graph container structure. All the graphs, tables or statistics are within this type of container across the whole application. The container is divided into two parts the header and the body. The header contains the heading and the toggle button, the body contains the loading bar, informative message panel and the inner data container. Information panel appears when there is no data found or if a specific message need to be displayed to inform the user about the state of the application. Inner graph container is identified by the ID, this ID is used when declaring an SVG panel to ensure graphs correct location within the application.

```
<!-- appliance duration charts container -->
<div flex class="flex-design">
  <div flex class="flex-header" layout="row" layout-align="space-between center">
    <!-- header title -->
    <div  class="flex-title size16">Current Weeks Appliance Duration</div>
      <button  class="flex-button" ng-click="isOpen = !isOpen">
        <i class="mdi mdi-minus" ng-show="!isOpen" ></i>
        <i class="mdi mdi-plus" ng-show="isOpen" ></i>
      </button>
  </div>
  <div flex class="flex-container" ng-show="!isOpen">
      <!-- graph container -->
    <div id="storyBoard" layout-padding>
      <!-- loading bar -->
      <md-progress-linear md-mode="indeterminate" ng-show="animeChartLoading"></md-progress-linear>
      <!-- information message when no data found -->
      <div layout="row" layout-fill layout-align="center center" ng-show="animeChartData"
           class="error_messages" layout-padding>
        {{animeChartMessage}}
      </div>
    </div>
  </div>
</div>
```

Figure 5.40: Code snippet of Current weeks appliance duration graph

Goal Setting view

Figure 5.41 illustrates the goal setting view, it includes monthly target setting and statistics. Current Month Goal Setting allows to set a target, which should motivate the user to reduce electricity consumption. Target can be set by typing the kWh or by dragging slider both ways. Once the target is

selected, "Set" button will set the target. "Reset" button can be used at any time to reset the target. If there has been a previous target then previous target slider will display the previous month target. Current month slider displays the current month overall usage. Once a target is set or reset a bullet chart is displayed to visualize the data.

Goal Setting Statistics table displays the information of all the monthly target statistics. If the target was achieved, by how much current month usage was over or below the set target. Percentage and achieved columns are changing colour from green to red or vice versa indicating positive or negative target state.

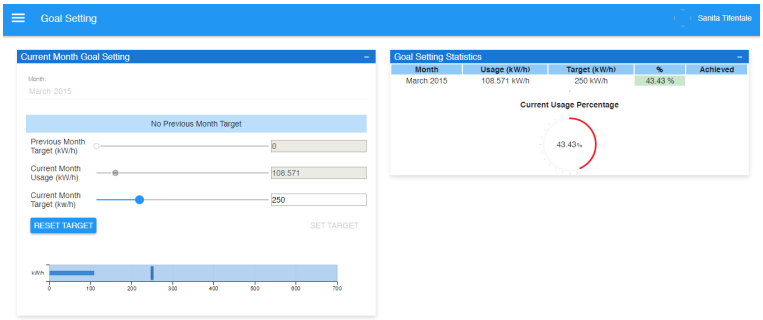


Figure 5.41: Goal Setting view

Figure 5.42 illustrates a code snippet of the function checking if the monthly target was achieved. This function is triggered every first day of the month. e.g. if today is the 1st of March 2015, then the function will be executed and it will request the API to check if the target has been achieved. The response will be displayed on the screen via informative message to inform the user if the target has been achieved. The statistics table also will be updated and Achieved column will indicate if the target was achieved or not.

```
// get the first day of the month
var firstMonthDay = new Date();
firstMonthDay.setDate(1);

// if first day of the month is current day then check if previous month target was achieved, notify user on the
// result, if there was no targets do nothing
if ($scope.current_month.getDate() === firstMonthDay.getDate()){
  $http.get($rootScope.url + 'check_target', $scope.header)
    .success(function(response){
      if (response.message == "Previous Month Target Not Achieved"){
        $scope.notAchievedMessage = response.message;
        $scope.notAchieved = true;
      }
      if (response.message == "No Data Found"){
        $scope.achieved = false;
      }
      else
      {
        $scope.achievedMessage = response.message;
        $scope.achieved = true;
        // obtain user target statistics
        $http.get($rootScope.url + 'target_stat', $scope.header)
          .success(function(response){
            if (response.message != null || response.data == ""){
              // inform user and remove loading bar
              $scope.statLoading = false;
              $scope.statMessage = "No current or previous targets";
              $scope.statData = true;
            }
            else{
              $scope.statistics = response.data;
              $scope.statLoading = false;
              $scope.show_table = true;
            }
```

Figure 5.42: Code snippet of the function checking if the monthly target is achieved

Comparison view

Figure 5.43 illustrates the comparison view containing two graphs. Line chart displays the users monthly usage in comparison to average users monthly usage. When Comparison view is loaded, the current month usage comparison is displayed. Arrows are provided to move from one month to another, when there is no data found the information message is displayed and arrow is disabled. Bar chart displays up to three day hourly usage comparison in Wh. The user can select any three days and compare their usage.

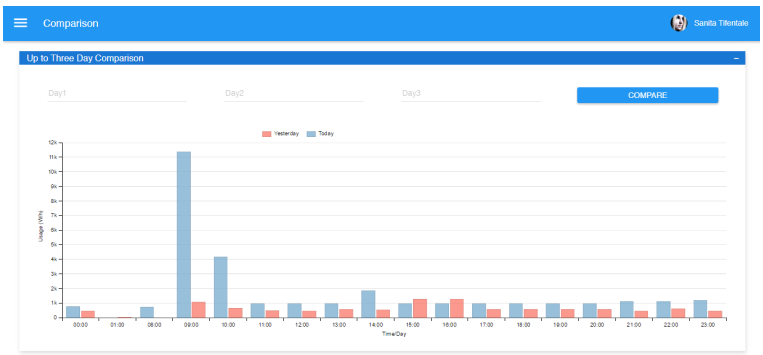


Figure 5.43: Comparison view

Figure 5.44 illustrates the function used to convert a date time to a date format is needed for the application. This function was obtained from the Stack Overflow and has been used across the whole web application.

```
// function to convert any timestamp to date without time and timezones
// obtained from Stack overflow
$scope.dateConverter = function(date){
  var dd = date.getDate();
  var mm = date.getMonth()+1;
  var yyyy = date.getFullYear();

  if(dd < 10) {
    dd = '0' + dd
  }

  if(mm < 10) {
    mm = '0' + mm
  }

  return dd+'/'+mm+'/'+yyyy;
};
```

Figure 5.44: Code snippet of the date converter

Account view

Figure 5.45 illustrates the account view containing the user details and allows for the user details to be changed at any time. Email field is disabled because email can't be change. This view also offers to change the password, upload or delete a profile picture. To change the password user need to provide the old and it must be valid to change the password to the newly entered password.

"Upload Profile Picture" link opens the form, which allows for the file upload. Once the image is selected, "Crop" the picture. If you are satisfied with the selected image then "Save" image, else "Reset" it. "Reset" will remove the picture and allow for new picture upload. Pictures can be changed at any time.

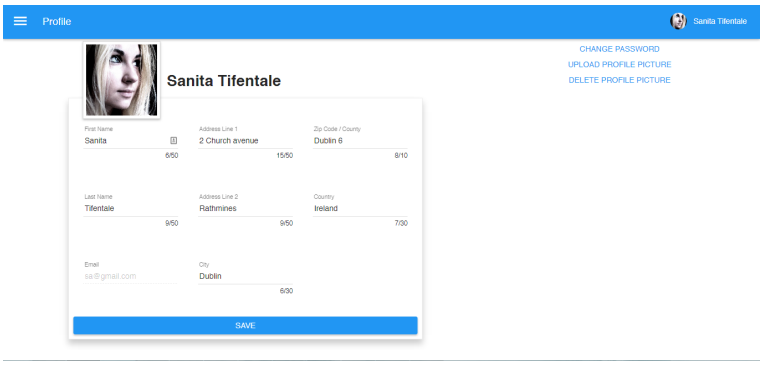


Figure 5.45: Account view

Help dialog view

Figure 5.46 illustrates the help dialog, it contains the information of each graph or component of the application.

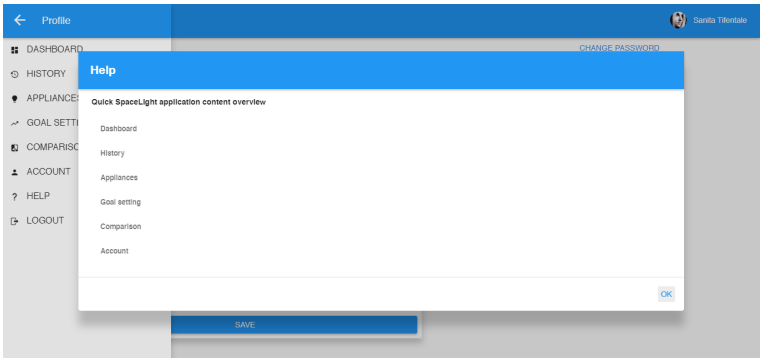


Figure 5.46: Help dialog view

5.8 Deployment

Two new EC2 instances were created, the same way as described in the section 5.3.1 and putty was used to access both instances also discussed in the section 5.3.2. Ensure both instances have HTTP security group added, to ensure port 80 is exposed.

5.8.1 AngularJS web application deployment

One of the created EC2 instance was used to host the web application. Steps undertaken to deploy the AngularJS web app:

1. Update the repository
   sudo apt-get update
2. Installed Node.js, development files and npm
   sudo apt-get install nodejs nodejs-dev npm
3. Install Yeoman
   sudo npm install -g yo - if this gives an error at the end then do
   sudo ln -s /usr/bin/nodejs /usr/bin/node - this will fix the issue and then execute previous command again and Yeoman should be install correctly
4. Install generator-angular
   sudo npm install -g generator-angular
5. Install git
   sudo apt-get install git
6. Clone the web application repository to the root
   git clone [clone URL]

7. Install Bower
   sudo npm install -g bower
8. Go to the web app directory and install all the dependencies
   npm install
   bower install
9. Install Nginx - web server
   sudo apt-get install nginx
10. Configure Nginx - set up reverse proxy
    cd /etc/nginx/sites-enabled/ - change the default file or add your own configuration. Figure 5.47 illustrates how Nginx web server is configured: Nginx listens to the port 80 and web app grunt serve runs it on the local port 9000, to forward to port 9000 upstream is declared.

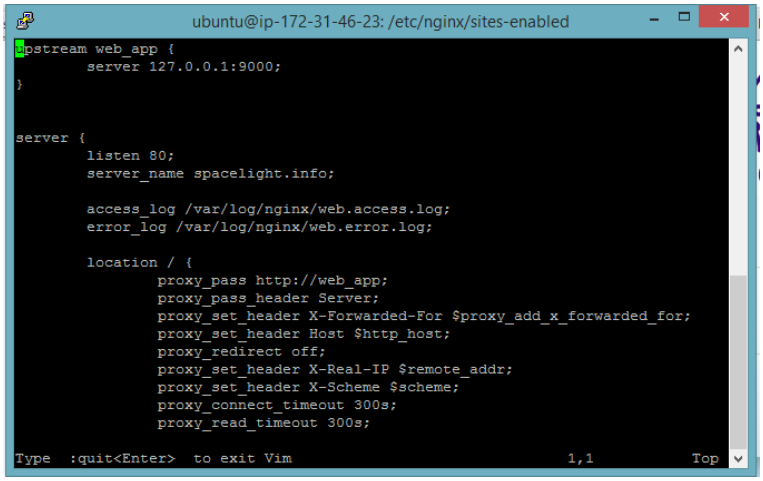


Figure 5.47: Nginx reverse proxy configuration

11. Restart the server
    sudo service nginx restart
12. Run grunt task runner in the background
    nohup grunt serve &
13. Enter your public IP or domain name (spacelight.info) and app should be displayed in the browser

Note: to obtain these exact commands few EC2 instances were recreated, because corrupt packages or incompatible node.js and npm versions were installed. Eventually this sequence of command were obtained and successfully deployed the web application.

If any changes were made then simply pull from the git repository and find the running process via ps aux — grep grunt - obtain the pid and kill that process via kill [pid]. To restart the task execute nohup grunt serve &

command again.

If any directory needs permission use this command to grant the permission to the user: sudo chown -R $USER [directory path].

If any file is in read mode and write mode is needed then use this command to give all the permissions: sudo chmod 777 [file name]. This could be necessary when Nginx configuration is mad eon the default page as it will be in the read mode.

5.8.2 API deployment

Other EC2 instance was used to host an API. Steps undertaken to deploy the Flask app:

1. Update the instance
   sudo apt-get update
2. Install all the packages
   sudo apt-get install -y python python-pip python-virtualenv nginx gunicorn
3. Create and activate virtual environment
   sudo virtualenv env
   source env/bin/activate
4. Install Flask
   sudo pip install Flask==0.10.1
5. Install Git and clone the API repository
   sudo apt-get install git
   git clone [repository URL]
6. Install postgresql database
   sudo apt-get install postgresql-9.3 postgresql-server-dev-all
7. Install all the dependencies from the requirements.txt file
   pip install -r /path/to/requirements.txt - to create a requirements document command: pip freeze >requirements.txt
8. Start Nginx server
   sudo /etc/init.d/nginx start
9. Configure Nginx as before and restart
   sudo service nginx restart
10. Create run.sh - Figure 5.48 illustrates the run.sh script
    ./run.sh - execute the script
11. Now go to the web application (spacelight.info) and try to log in. If successfully logged in the API is up and running.

Note: to obtain these exact commands few EC2 instances were recreated, because not all the dependencies of the applications were installed and some packages were in conflict with each other, to ensure clean start new EC2 instances were used. Ensure PyJWT is installed

not JWT package, it can get messy if some packages are not correctly installed.

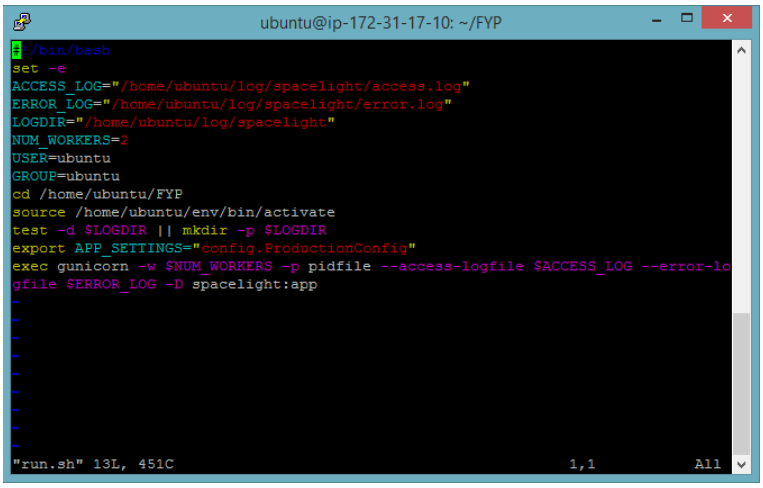


Figure 5.48: API run.sh script

5.8.3 Register a Domain Name

Amazon Route 53 web services were used to register a domain name. Steps undertaken to register a domain name and associate it with EC2:

1. Log-in to Amazon AWS account
2. Select Route 53 services
3. Select domains
4. Click Register Domain button at the top of the screen
5. Select a domain name - in the scope of this project a spacelight domain name was searched for and available generic top-level domain was selected - spacelight.info. It was $12 a year, it was a good deal because spacelight.io was $32 a year.
6. Fill in contact details, review and purchase
7. The domain name will be registered within three working days
8. To associate a domain name with EC2 instance an elastic IP address need to be used. Elastic IP address is a public IP address that is allocated to particular account and it is there until you decide to release it.
9. Go to EC2 console
10. On the left hand side under the Network & Security select Elastic IPs
11. Select Allocate New Address
12. Select Associate Address and select EC2 instance
13. To create a Record Set: go to Amazon Route 53 Console
14. Create a hosted zone or it will be allocated with newly registered domain name

15. Select Create Record Set
16. Select Type ->A-IPv4 address
17. Select no Alias
18. Set TTL to 60 seconds
19. Enter the IP address and create the record set
20. To obtain Domain Name from Load Balancers: Open EC2 console
21. Select Create Load Balancer
22. Give it a name
23. Select EC2 instance and create it
24. Now instead of using Public IP address spacelight.info domain name can be used

5.9 Conclusion

This chapter presented the implementation of each project component and how web application was deployed. The next chapter will discuss the system validation.

Chapter 6

System Validation

The following chapter describes the testing methods used in the presented project, as well as obtained testing results.

6.1 Unit Testing

"Unit testing is a test, executed by the developer in a laboratory environment, which should demonstrate that the program meets the requirements set in the specification." [51] Unit testing should be made on the small parts of the software for example methods or classes, to ensure their functionality. The strength of unit testing is to show that the individual parts of the software are correct. Unit testing also allows the programmer to refactor code at a later date, which will help in making sure the module still works correctly. By testing smaller parts first and then combining them, integration testing will become much easier. The most essential part of unit testing is to help the developer to reduce the number of bugs and not spending an an hours on debugging and looking for what caused the bug, and therefor contributes to a more stable software solution.

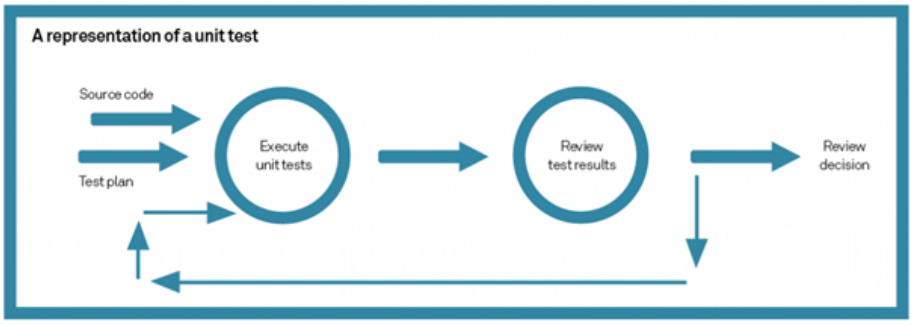


Figure 6.1: Unit Test representation [61]

Unit testing was supposed to be utilised by the project but due to time

constraint unfortunately it was omitted, instead each API call was tested by using Advanced Rest Client application before API was connected to the web application. This ensured that each API call performs the task accurately and responds with correct data / response message before it is connected to the web application and causes unnecessary errors.

Sample API calls made to the API and their response:

API call:	http://52.11.71.127/countries	Response
HTTP method:	GET	{ - countries: [259] -0: { country: "Afghanistan" }, -1: { country: "Albania" } }
Header:	Content-Type: application/json	
Body:	-	
Result:	Pass	

(a) API call to obtain countries

API call:	http://52.11.71.127/user	Response
HTTP method:	GET	{ add1: "2 Church Avenue" add2: "Rathmines" city: "Dublin" country: "Ireland" email: "sa@gmail.com" first_name: "Sanita" second_name: "Tifentale" zip_code: "Dublin 6" }
Header:	Content-Type: application/json Authorization: jwt eyJTkb1...	
Body:	-	
Result:	Pass	

(b) API call to obtain user details

API call:	http://52.11.71.127/reset_target	Response
HTTP method:	POST	{ message: "Target Successfully Reset" }
Header:	Content-Type: application/json Authorization: jwt eyJTkb1...	
Body:	{ "new_target": 300 }	
Result:	Pass	

(a) API call to reset target

API call:	http://52.11.71.127/hourly_usage	Response
HTTP method:	POST	{ -data: [17] -0: { hours: 0 usage: 751 } -1: { hours: 8 usage: 714 } }
Header:	Content-Type: application/json Authorization: jwt eyJTkb1...	
Body:	{ "date": "13/03/2015" }	
Result:	Pass	

(b) API call to obtain hourly usage of particular day

API call:	http://52.11.71.127/hourly_usage	Response
HTTP method:	POST	{ -data: [17] -0: { hours: 0 usage: 751 } -1: { hours: 8 usage: 714 } }
Header:	Content-Type: application/json Authorization: jwt eyJTkb1...	
Body:	{ "date": "13/03/2015" }	
Result:	Pass	

(a) API call to obtain appliance usage in particular date range

API call:	http://52.11.71.127/monthly_statistics	Response
HTTP method:	GET	{ -data: [1] -0: { month: "March 2015" payment: 24.13 stat: "" usage: 121 } }
Header:	Content-Type: application/json Authorization: jwt eyJTkb1...	
Body:	-	
Result:	Pass	

(b) API call to obtain monthly statistics for the History view

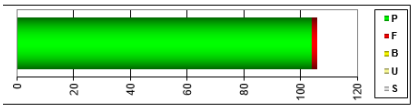
6.2 Integration Testing

"Integration testing may also include the testing of the integration of software and hardware." [60] Hardware and software integration were tested by attaching electricity power meters to the Arduino Uno, then WiFi shield was added and written Arduino sketch was uploaded to the Arduino Uno. The appliance was plugged into the meter power strip and pulse count was sent to MQTT server and raw data was viewed on the phone screen via MyMQTT app on Android phone. At the later stage of the project python module was introduced and automated data insertion into the database was implemented, to test the newly added component the data was transmitted from the meters up to the database and record count was checked within the database pgAdmin III client application. Once API and AngularJS web application was implemented the all component integration was tested by log-in into the application and viewing the transmitted data from the meters in the user interface via graphs and statistics. Once all the components integrated together the integration testing was successful.

6.3 System Testing

System testing also knows as a functional testing, which usually follows the integration testing. "System testing is performed using the complete set of test cases, which are developed from the system specification." [60] A set of test cases were created, to be precise 107, each test case was tested on the development environment, two test cases failed others passed. Figure 6.5 illustrates the template used to create all the test cases, this template is also used by my coworker at work when Quality Assurance (QA) team is testing the software. It is a great template as it automatically counts the passed, failed or non-applicable test cases and provides the result via user friendly graph at the top of the excel sheet. Figure 6.5 also illustrates the sample test cases that was applied during the System Testing stage.

1 - A



Test Case Results			
U	0	0%	0.0 h
P	104	98%	0.0 h
F	2	2%	0.0 h
S	0	0%	0.0 h
B	0	0%	0.0 h

07/03/2015 16:00

Created by: Sanita

TC#	Test Execution Steps	Expected Result	Test Result	Date Tested	Tester	TC Time	Comment / (or Requirement xref)
	LogIn Screen						
1	Input fields to obtain user login credentials	Email and password input fields	P		Sanita		
2	Valid email input field	Only correct email format is allowed	P		Sanita		
3	Valid password input field	Only valid password allowed	P		Sanita		
4	Required field identifiers	Star at each input field	P		Sanita		
5	Error messages if login details are incorrect	Wrong email or password error messages	P		Sanita		
6	Check if user can login with valid credentials	User should be able to login with valid credentials	P		Sanita		
7	Check if user can't login with invalid credentials	Deny access to the system	P		Sanita		
8	Provide user with registration link	Registration link should allow user	P		Sanita		

Figure 6.5: Test Case template with sample test cases and overall outcome

Test cases are based by the prior testing experience at work, when web diary was tested. Web application same as web diary had a log-in, registration, account forms, password change and form validation. This knowledge helped me to create a good set of test cases to ensure all the components of the web applications were tested, from the small component like input field up to the whole page view. The full list of all the test cases will be submitted separately.

6.4 Regression Testing

Regression testing is the final development testing stage, which ensures that all modules are working cohesively. It also ensures that no new bugs have been introduced in already existing system by adding a new functionality to it. Regression testing was applied on the deployed web application because System Testing was successful on the development environment and there was no need to test it again as there was no bug fixes to make. Regression testing was not fully successful because new issues were introduced which don't appear in the development environment.

High priority issues:

1. "Last 3 Day Appliance Usage Overview" graph in the Dashboard is not displayed due to the warning given in the development environment and crush in the development environment. Should have addressed the issue on the first time the warning was given, could have eliminated the bug.

2. "Current Weeks Appliance Duration" graph in the Appliance view is not displayed due to the same warning as previous graph because they both used the same SQL query to obtain the data from the Database.

3. Profile picture is not displayed in the Account view or on the toolbar but it is working on the development environment. Issue could be solved by not storing the image in the database but moving to the file system and store images in the protected folder on the server and store the URL to the picture in the database. This was supposed to be implemented in the first place, it is a huge mistake and caused a high priority issue. By storing the images in the file system would increase the performance and not cluster the database with huge files.

6.5 Cross-Platform Testing

Browser Platform	Fully Supports	Minor Issues	Major Issues	Issues
Chrome	✓			-
Opera	✓			-
Mozilla Firefox		✓		* Dashboard view boxes are distorted
Internet Explorer			✓	* Dashboard view boxes collide with each other * Help view is distorted
Safari			✓	* Layout is ignored, the content is aligned to the left hand side
Mobile phone		✓		* Functionality works and it is responsive but some views seems a bit overfilled
Tablet		✓		* Functionality works and it is responsive but on portrait view some views seems a bit overfilled

Table 6.1: Browser and platform support overview

6.6 User Testing

User testing or also known as Acceptance testing is the final testing stage, it involves the potential users as they need to use the web application and provide with the feedback. This is an essential testing stage because if users don't like the application then there is no potential business but if users

had a good first impression then there is a potential. To inform on the web application and the state of if, a documentation was created. It gave a brief description of what is Space Light used for, how electricity is measured within the household, explained why meters are not used as part of the user testing and that not all data is from the meters. Documentation also included the supporting browsers and the advised browser would be Google Chrome, the high priority issues also were mentioned to inform the users. To access the application and view the data the log-in details were provided, otherwise they would see only an informative message stating that there are no data to view. Application and survey link also was included within the documentation. The document is added withing the Appendix B and blank survey form is added within Appendix A.

6.7 Nielsen's heuristics Evaluation and survey response

The survey was based on the Nielsen's heuristics because it is a usability inspection method for computer software that helps identify problems in the user interface design. Additional questions were added to the survey to obtain the user opinion / feedback and any issues they may have noticed. Different type of people were asked to participate in the project usability testing, skilled people working within the industry, students, parents and people with basic computer knowledge. This exposes the application to the variety of different people, different opinion. The survey response is listed below.

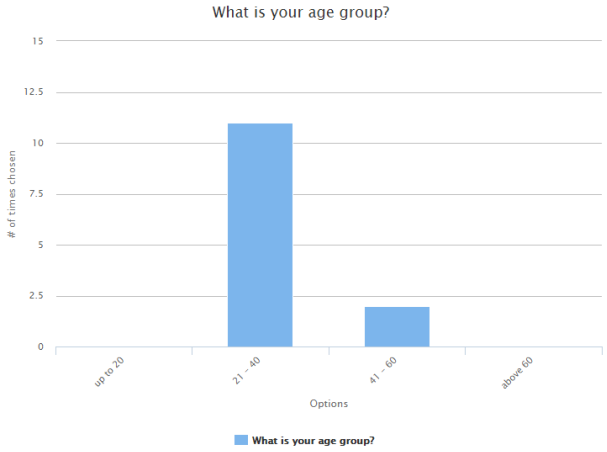


Figure 6.6: Question 1

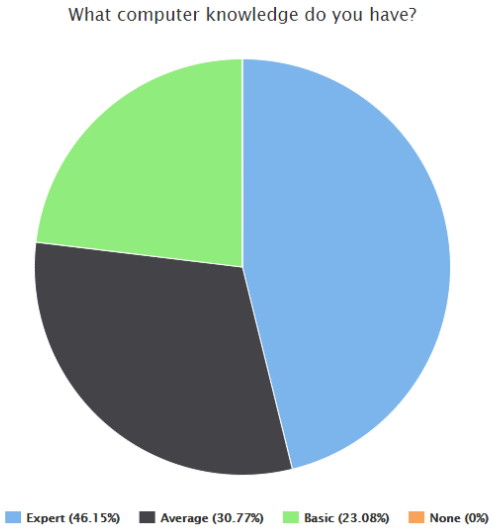


Figure 6.7: Question 2

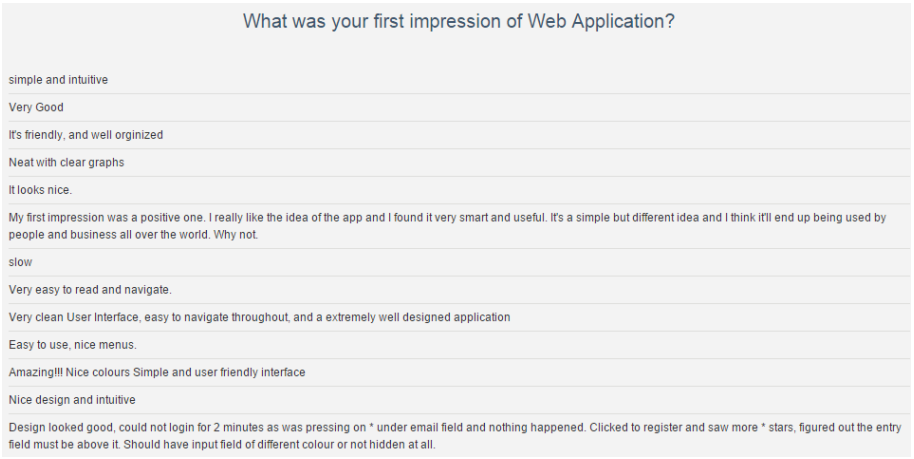


Figure 6.8: Question 3

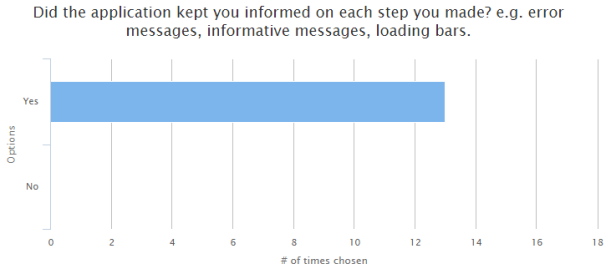


Figure 6.9: Question 4

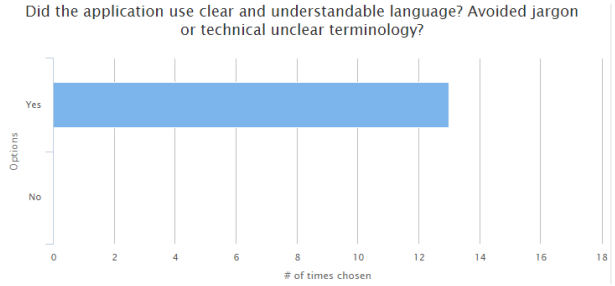


Figure 6.10: Question 5

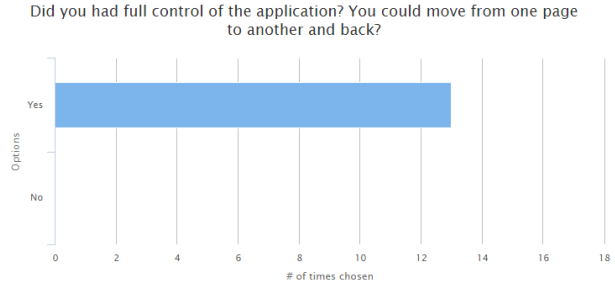


Figure 6.11: Question 6

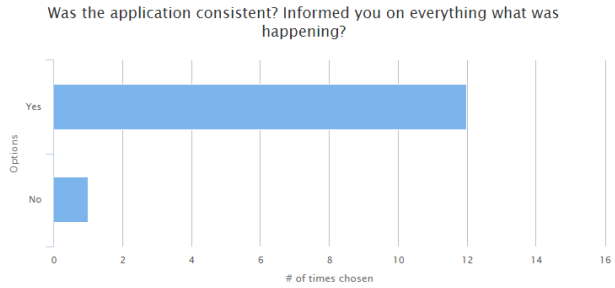


Figure 6.12: Question 7

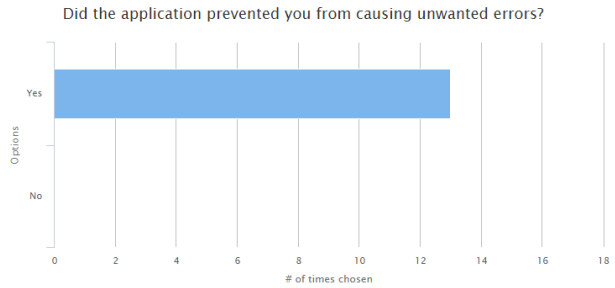


Figure 6.13: Question 8

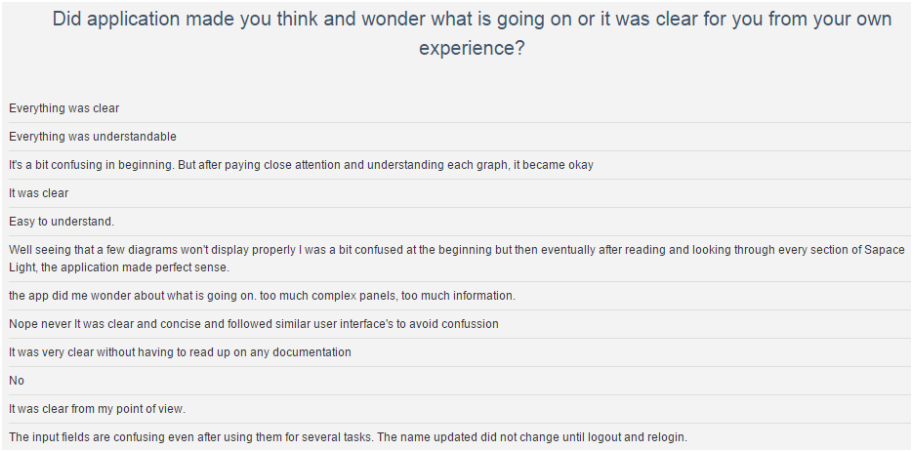


Figure 6.14: Question 9

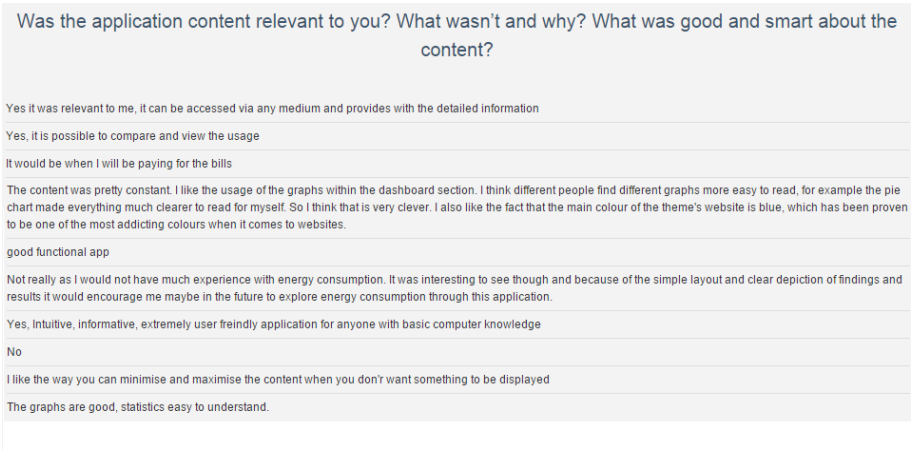


Figure 6.15: Question 10

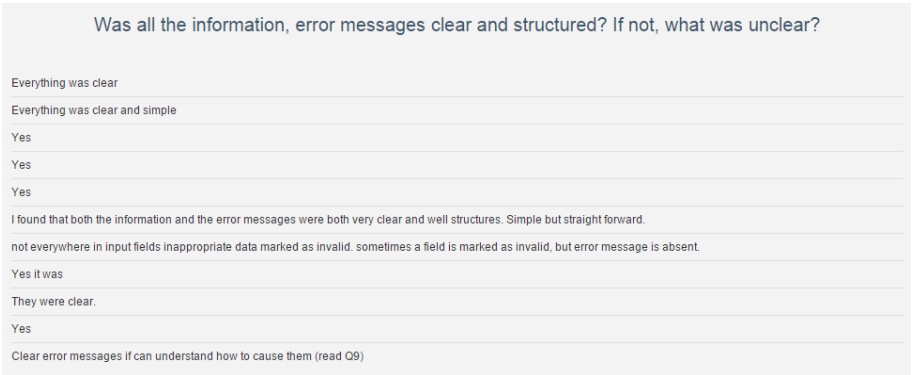


Figure 6.16: Question 11

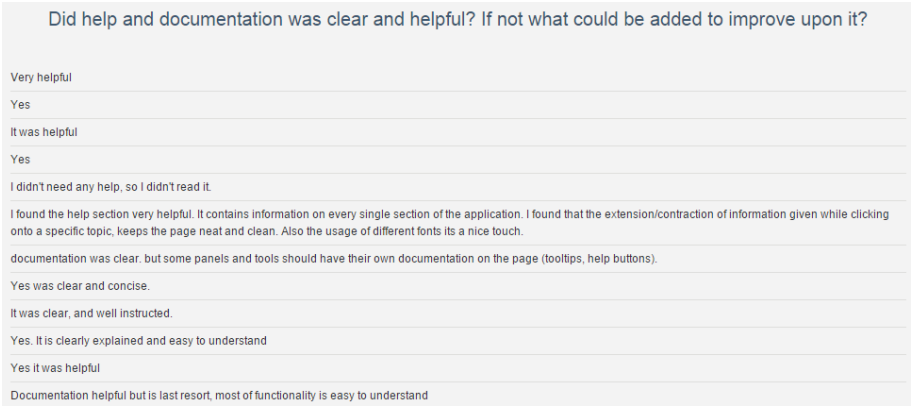


Figure 6.17: Question 12

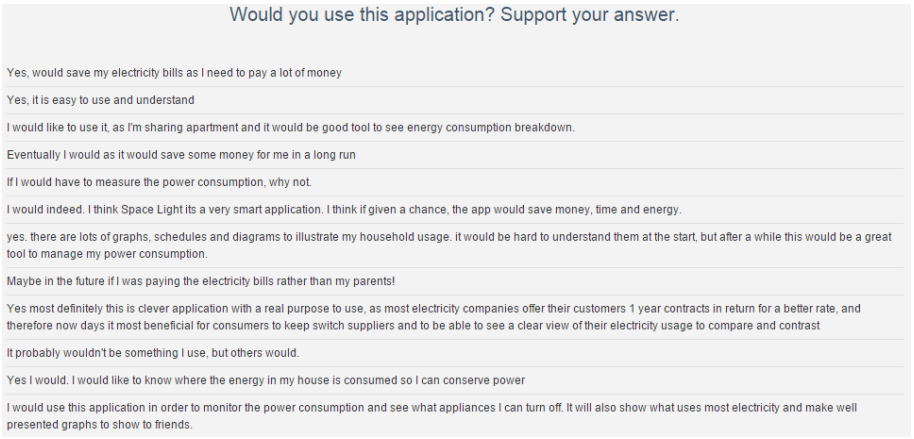


Figure 6.18: Question 13

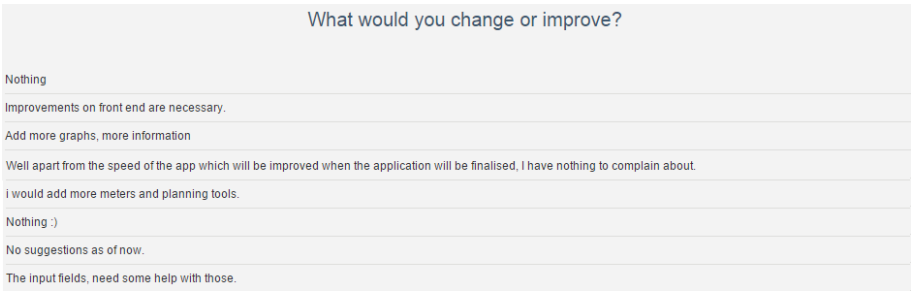


Figure 6.19: Question 14

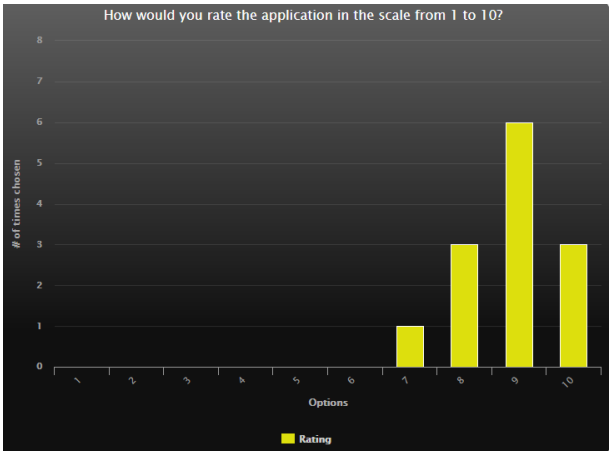


Figure 6.20: Question 15

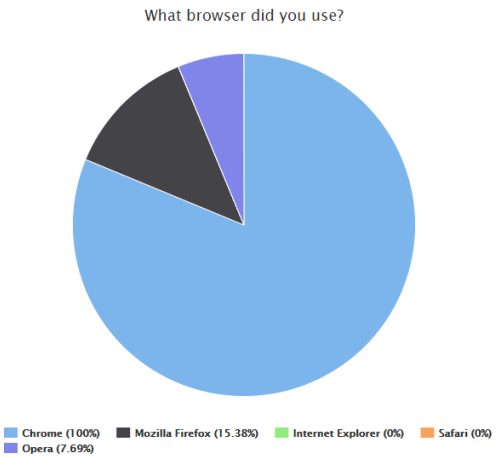


Figure 6.21: Question 16

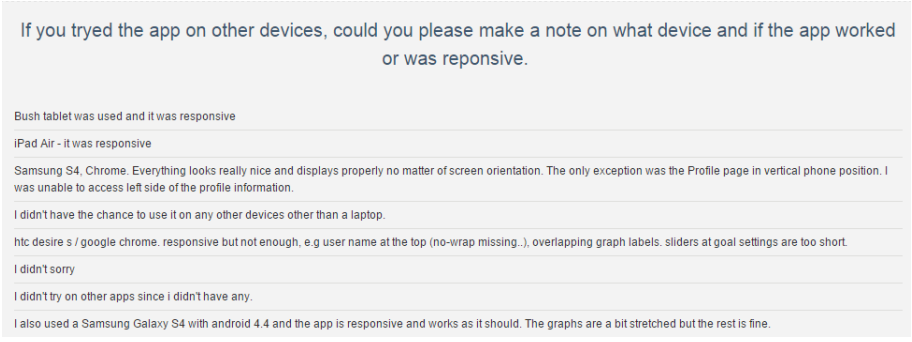


Figure 6.22: Question 17

6.8 Issues found by test users

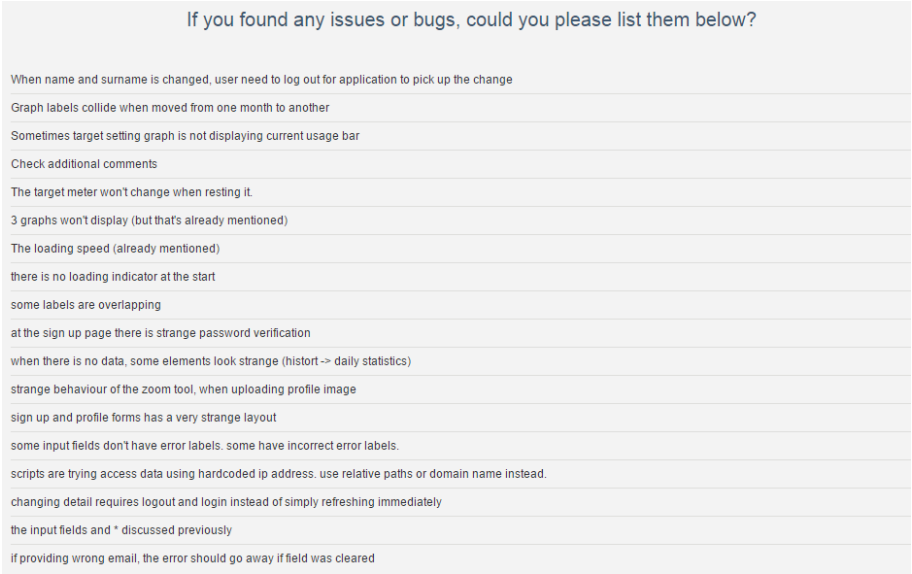


Figure 6.23: Question 18

6.9 Addressed issues

The following user testing issues were resolved:

- In the History view a monthly chart was ordered by the usage, it was changed to order by date.
- In the History view a scroll bar was added to the daily statistics table to ensure the box is not stretched across the whole web application.
- In the History view a monthly chart "Date" label is no more half hidden, the graphs container heights was increased.
- In the Goal Setting view a reset target was not fully working but not it is fixed and fully functional.
- In the History view on a price set the information message was showing "U" instead "Unit Price Successfully Set"

6.10 Conclusion

The results obtained were generally positive but some constructive criticism was also given, which helps to understand were the application can be improved and made better. Using the provided feedback some minor changes were made to ensure one full testing cycle was completed. Due to time constrain a second cycle of test could not be completed and some issues were unfortunately not resolved.

Chapter 7

Conclusion & Future Work

7.1 Introduction

This chapter analyses project planning, the strength and the weaknesses of the project and includes the future work. Personal thoughts and learning outcomes are also stated within the chapter.

7.2 Project Plan

Figure 7.1 illustrates the project initial Guntt chart. Not everything happened as it was planned, the red parts of the Guntt chart indicate the not fulfilled tasks and it was pushed back to the later stage of the project. The exam preparation and exams themselves were not taken in consideration during the planning stage of the project, this lead to few tasks being pushed back. Eventually time lost was regained and project was going forward.

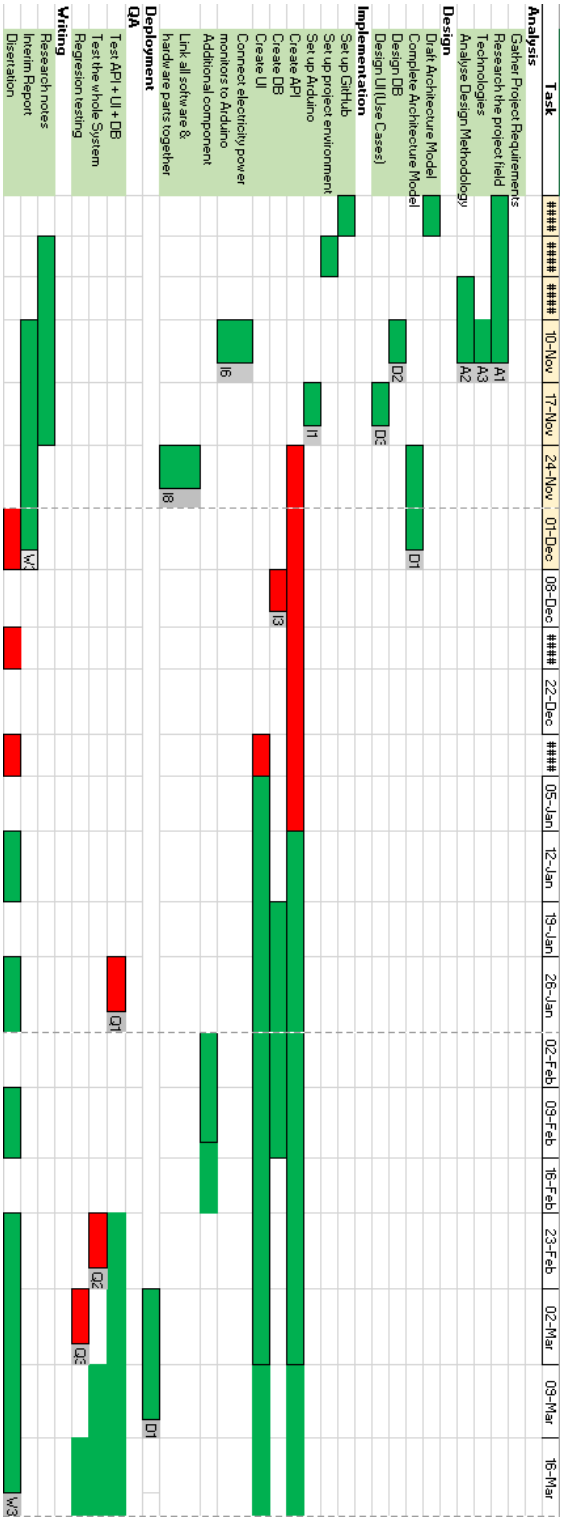


Figure 7.1: Gantt Chart

7.3 Future work

7.3.1 Bug fixes

The first thing that should be done is fixing all the bugs that were omitted due to time constraint. Improve on the design and ensure the web application works and looks the same across all the platforms.

7.3.2 Analyse the power consumption of many appliances on one electricity stream

For now the application identifies the power consumption of one appliance at a time but in the future this can be changed to multiple appliances by analysing the incoming pulse stream. This can be achieved because each appliance has its own power consumption rate and by knowing the household appliances and variety of combination this could be eventually achieved.

7.3.3 Providers

To bring the application to the next level a provider selection could be introduced to allow the user select their provider and the monthly bill would be calculated based by the provider charges and not by the user electricity unit price. This would make the application more user friendly and automated. During the project development stage I have asked for people opinion to what they would like to see within the application and the provider selection was one of the first things they have mentioned. It is because these days provider switching ensures better deals and this application could suggest what provider to choose next and remind the when to switch.

7.3.4 Recommendations

This project didn't really implemented any recommendations but only provided with the comparison and indication on what consumes more power. A recommendation section would make the application more smarter because it would suggest what appliance need to be changed or what is the best time to do a particular task.

7.3.5 Switching the appliances on or off via mobile phone

To switch the appliance on and off via mobile phone would increase the demand for the application and this could be turned into the profitable business. However this would require an enormous amount of labour hours and better hardware components. Power meter are wired to the Arduino Uno but wireless meters can be introduced and make it more modern energy monitoring system.

7.4 Project strength

The project has a number of strengths due to well structured project plan and enormous amount of time put in to the research. Not only in the background research but in technology selecting which was a crucial part of the project because it consisted of many parts and all needed to be integrated in the cohesive manner. The main project objective was to develop a hardware and software solution to monitor household appliances and visualize the gathered data via various graphs and statistics. This was achieved because all the components were integrated and the web application was deployed. The overall feedback from the web application survey was positive and first impressions were good, the users liked the ease of use and professional looking user interface.

7.5 Project weaknesses

Like any other project there are weaknesses. The presented project weakness are the three unresolved deployment bugs: the image not displaying and two graphs not appearing due to database error. The profile picture issue can be resolved by implementing file system rather than storing an image in the database. Graphs can be fixed by spending more time by looking in to the issue. Other weakness is slow performance when loading an application, this can be resolved by minimizing the web application which wasn't done in the deployment stage of the project. There are small design flaws but with extra time they could have been eliminated.

7.6 Learning Outcome

The project scope was enormous and provided with invaluable learning opportunities, not only in the computer science field but also in personal development. By undertaking this size of project and completing every stage of it shows how complex and challenging it can be but with the right motivation and willingness to succeed, nothing is impossible. The skills obtained from this project are irreplaceable. These skills include planning, time management, organising, problem solving, designing, testing, researching, coding and many more. I have never used many of the technologies required by this project and it was challenging to learn but it has been a valuable learning experience. The technical skills and knowledge obtained from this project include cloud computing via Amazon AWS services, better Linux understanding, new programming languages learned: python and Arduino, improved on my web development skills: AngularJS, java script CSS, HTML5. Implemented my first API by using Flask framework and deployed my first web application. Bought my first domain name and hosted my first

database via Amazon RDS. The amount of technologies used within this project shows my ability to learn, adapt and the willingness to discover new things.

7.7 Conclusion

Looking back at the project, I realize that many things could have been done differently and more efficiently but overall I am pleased with my project and my commitment to it. This project has been a valuable experience and I have thrived as a better developer and person. I must admit there were times when it was exhausting and power draining but the taught of completing the project kept me going and there is no better feeling than producing a valuable piece of work and being proud of it. Due to time constraint, there are flaws within the web application but overall system functionality seems to work just fine. With the future work in mind these flaws can be eliminated and fully complete system can be produced. While immense amount of work and effort has gone into this project to successfully complete and solve as many issues as possible, the invaluable knowledge was gained.

Bibliography

[1] *10 Heuristics for User Interface Design: Article by Jakob Nielsen*. URL: http://www.nngroup.com/articles/ten-usability-heuristics/.

[2] Muhammad Fajri Bayu Anbya et al. "Wireless sensor network for single phase electricity monitoring system via Zigbee protocol". In: *Proceedings of 2012 IEEE Conference on Control, Systems and Industrial Informatics, ICCSII 2012*. 2012, pp. 261–266.

[3] *AngularJS vs Backbone.js vs Ember.jsChoosing a JavaScript Framework [Part 2]*. URL: http://blog.fusioncharts.com/2014/08/angularjs-vs-backbone-js-vs-ember-jschoosing-a-javascript-framework-part-2/ (visited on 11/29/2014).

[4] *Arduino - ArduinoBoardUno*. URL: http://arduino.cc/en/Main/arduinoBoardUno (visited on 11/13/2014).

[5] *Arduino Ethernet Shield Tutorial*. URL: http://www.instructables.com/id/Arduino-Ethernet-Shield-Tutorial/ (visited on 11/20/2014).

[6] *Arduino Store - community and electronics*. URL: http://store.arduino.cc/product/A000066 (visited on 11/13/2014).

[7] *Arduino Uno vs BeagleBone vs Raspberry Pi — MAKE*. URL: http://makezine.com/2013/04/15/arduino-uno-vs-beaglebone-vs-raspberry-pi/ (visited on 11/02/2014).

[8] *AWS — Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting*. URL: http://aws.amazon.com/ec2/\#functionality (visited on 11/28/2014).

[9] *AWS — Amazon Relational Database Service (RDS)*. URL: http://aws.amazon.com/rds/ (visited on 11/28/2014).

[10] *AWS — What is Cloud Computing - Benefits of the Cloud*. URL: http://aws.amazon.com/what-is-cloud-computing/ (visited on 11/28/2014).

[11] T Bapat et al. "User-sensitive scheduling of home appliances". In: *Proceedings of the 2nd ACM SIGCOMM workshop on Green networking*. ACM. 2011, pp. 43–48.

[12] Karim Said Barsim, Roman Streubel, and Bin Yang. "An Approach for Unsupervised Non-Intrusive Load Monitoring of Residential Appliances". In: (), pp. 1–5.

[13] Nipun Batra et al. "It's Different: Insights into Home Energy Consumption in India". In: *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings* August (2013), 3:1–3:8. DOI: 10.1145/2528282.2528293. URL: http://doi.acm.org/10.1145/2528282.2528293.

[14] *BeagleBone Black Rev C - 4GB Flash - Pre-installed Debian ID: 1876 - $55.00 : Adafruit Industries, Unique & fun DIY electronics and kits*. URL: http://www.adafruit.com/product/1876 (visited on 11/13/2014).

[15] Robert Berry. *Querying Time Series in Postgresql*. 2014. URL: http://no0p.github.io/postgresql/2014/05/08/timeseries-tips-pg.html (visited on 11/13/2014).

[16] *Bower*. URL: http://bower.io/.

[17] Ryan Brown. *Django vs Flask vs Pyramid: Choosing a Python Web Framework*. 2014. URL: http://www.airpair.com/python/posts/django-flask-pyramid (visited on 11/14/2014).

[18] *Browse : Python Package Index*. URL: https://pypi.python.org/pypi?:action=browse\&c=533\&show=all (visited on 11/12/2014).

[19] Calvin. *Why We Choose Python — Six Feet Up, Inc.* URL: http://www.sixfeetup.com/blog/why-we-choose-python.

[20] *Can I output a HDMI signal with an Arduino? - Electrical Engineering Stack Exchange*. URL: http://electronics.stackexchange.com/questions/67098/can-i-output-a-hdmi-signal-with-an-arduino (visited on 11/13/2014).

[21] A Cockburn. "Using both incremental and iterative development". In: *STSC CrossTalk (USAF Software Technology ...* May (2008), pp. 27–30. URL: http://scholar.google.com/scholar?hl=en\&btnG=Search\&q=intitle:Using+Both+Incremental+and+Iterative+Development\#0.

[22] G Coley. "BeagleBone Black System Reference Manual". In: (2013). URL: http://dl.btc.pl/kamami_wa/beaglebone_black_revc.pdf.

[23] *Comparing Python to Other Languages — Python.org*. URL: https://www.python.org/doc/essays/comparisons/ (visited on 11/12/2014).

[24] *Create an AMI from an Amazon EC2 Instance - AWS Toolkit for Visual Studio*. URL: http://docs.aws.amazon.com/AWSToolkitVS/latest/UserGuide/tkv-create-ami-from-instance.html (visited on 03/25/2015).

[25] *Django, pyramid, flask Job Trends — Indeed.com*. URL: http://www.indeed.com/jobanalytics/jobtrends?q=Django,+pyramid,+flask\&l= (visited on 11/14/2014).

[26] *Electronic Single Phase Din Rail Energy Meter XTM18SA from Yueqing Xintuo Electronic Science&Technology Co.,Ltd, China*. URL: http://www.ec21.com/product-details/Electronic-Single-Phase-Din-Rail--8779616.html (visited on 11/30/2014).

[27] *EU energy in figures, Statistical Pocketbook 2014*. Luxembourg: Publications Office of the European Union, 2014. ISBN: 978-92-79-29317-7. DOI: 10.2833/24150. URL: http://ec.europa.eu/energy/sites/ener/files/documents/2014_pocketbook.pdf.

[28] *FAQ - Frequently Asked Questions — MQTT*. URL: http://mqtt.org/faq (visited on 11/15/2014).

[29] *Filesystem Hierarchy Standard - 3.12 /opt : Add-on application software packages*. URL: http://www.pathname.com/fhs/2.2/fhs-3.12.html.

[30] *Grunt: The JavaScript Task Runner*. URL: http://gruntjs.com/.

[31] *HiveMQ 1.4.0 - User Guide*. URL: http://www.hivemq.com/docs/hivemq/1.4.0/.

[32] *How to Choose the Right Platform: Raspberry Pi or BeagleBone Black? — MAKE*. URL: http://makezine.com/magazine/how-to-choose-the-right-platform-raspberry-pi-or-beaglebone-black/ (visited on 11/02/2014).

[33] Martin Howley and Mary Holland. *Energy in Ireland 1990 - 2012*. Tech. rep. URL: http://www.seai.ie/Publications/Statistics_Publications/Energy_in_Ireland/Energy_in_Ireland_1990_-_2012_Report.pdf.

[34] Jeff Hsu et al. "HBCI: human-building-computer interaction". In: *Proceedings of the 2nd ...* (2010), pp. 55–60. URL: http://dl.acm.org/citation.cfm?id=1878444.

[35] *HTML5 specification finalized, squabbling over specs continues — Ars Technica*. URL: http://arstechnica.com/information-technology/2014/10/html5-specification-finalized-squabbling-over-who-writes-the-specs-continues/ (visited on 11/29/2014).

[36] Igor. *Arduino Power Consumption - Gadget Makers' Blog*. 2013. URL: http://gadgetmakersblog.com/arduino-power-consumption/ (visited on 11/13/2014).

[37] Taiseer Joudeh. *AngularJS Token Authentication using ASP.NET Web API 2, Owin, and Identity - Bit of Technology*. URL: http://bitoftech.net/2014/06/09/angularjs-token-authentication-using-asp-net-web-api-2-owin-asp-net-identity/ (visited on 03/25/2015).

[38] Kasia Mikoluk. *Python vs Java: Key Differences*. 2013. URL: https://www.udemy.com/blog/python-vs-java/ (visited on 11/12/2014).

[39] Aqeel H. Kazmi et al. "A Review of Wireless-Sensor-Network-Enabled Building Energy Management Systems". In: *ACM Transactions on Sensor Networks* 10.4 (June 2014), pp. 1–43. ISSN: 15504859. DOI: 10.1145/2532644. URL: http://dl.acm.org/citation.cfm?doid=2633905.2532644.

[40] Michael Kugler and Florian Reinhart. "Architecture of a ubiquitous smart energy management system for residential homes". In: *Proceedings of the 12th ...* (2011), pp. 101–104. URL: http://dl.acm.org/citation.cfm?id=2000770.

[41] C. Larman and V.R. Basili. "Iterative and incremental developments. a brief history". In: *Computer* 36.6 (June 2003), pp. 47–56. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1204375. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1204375.

[42] *ModMyPi — Raspberry Pi - Model B+ (NEW)*. URL: https://www.modmypi.com/raspberry-pi-model-b-plus (visited on 11/13/2014).

[43] *MQTT Message Log*. URL: http://www.hivemq.com/plugin/mqtt-message-log/ (visited on 11/26/2014).

[44] *Node.js*. URL: https://nodejs.org/ (visited on 03/20/2015).

[45] *Pololu robotics forum - View topic - A4988 Arduino Uno AC Power Adapters*. URL: http://forum.pololu.com/viewtopic.php?f=15\&t=9042 (visited on 11/20/2014).

[46] *Power Consumption of Typical Household Appliances*. URL: http://www.daftlogic.com/information-appliance-power-consumption.htm (visited on 03/25/2015).

[47] *PuTTY: a free telnet/ssh client*. URL: http://www.chiark.greenend.org.uk/~sgtatham/putty/.

[48] *Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities*. URL: http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext (visited on 11/12/2014).

[49] *python, php, java, c++, c# Job Trends — Indeed.com*. URL: http://www.indeed.com/jobanalytics/jobtrends?q=python\%2C+php\%2C+java\%2C+c\%2B\%2B\%2C+c\%23\&l= (visited on 11/12/2014).

[50] *Raspberry Pi B+ Desktop (700MHz Processor, 512MB RAM, 4x USB Port): Amazon.co.uk: Computers & Accessories*. URL: http://www.amazon.co.uk/Raspberry-Pi-Desktop-700MHzProcessor/dp/B00LPESRUK/ref=sr_1_1?ie=UTF8\&qid=1415046688\&sr=8-1\&keywords=raspberry+pi+b\%2B\#productDetails (visited on 11/13/2014).

[51] P. Runeson. "A survey of unit testing practices". In: *IEEE Software* 23.4 (July 2006), pp. 22–29. ISSN: 0740-7459. DOI: 10.1109/MS.2006.91. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1657935.

[52] Hans-Jurgen Schonig. "PostgreSQL vs NoSQL Why structure matters". In: (2013).

[53] Tom Igoe (Editor) Scott Fitzgerald (Editor), Michael Shiloh (Editor). *Arduino Projects Book*. Torino: Arduino LLC, 2013.

[54] URI SHAKED. *AngularJS vs. Backbone.js vs. Ember.js*. 2014. URL: https://www.airpair.com/js/javascript-framework-comparison.

[55] PD Sheet. "Product Data Sheet". In: *J. Biol. Chem* (2013), pp. 1–3. URL: http://premiermundo.com/files/Product/Support/ServiceManuals/MP-0956-1.sm.pdf.

[56] Shneiderman, B (Editor), Plaisant, C (Editor). *Designing the User Interface: Strategies for Effective Human-Computer Interaction: Fifth Edition*. Reading, MA: Addison-Wesley Publ. Co., 2010.

[57] *Software Testing & QTP: Software Testing Development Models*. URL: http://aboutqtp10.blogspot.ie/2011/05/software-testing-development-models.html (visited on 11/14/2014).

[58] D. Sotirovski. "Heuristics for iterative software development". In: *IEEE Software* 18.3 (2001), pp. 66–73. ISSN: 07407459. DOI: 10.1109/52.922728. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=922728.

[59] Sasa Stamenkovic. *List of All Countries in All Languages and All Data Formats — Umpirsky Software Development Blog*. URL: http://dev.umpirsky.com/list-of-all-countries-in-all-languages-and-all-data-formats/ (visited on 03/25/2015).

[60] Real-time Embedded Systems and Michael D Lewis. "19971008 065". In: ().

[61] *The art of unit testing explained — Linux User & Developer - the Linux and FOSS mag for a GNU generation*. URL: http://www.linuxuser.co.uk/tutorials/the-art-of-unit-testing-explained (visited on 11/14/2014).

[62] *The web's scaffolding tool for modern webapps — Yeoman*. URL: http://yeoman.io/.

[63] *Tweet-a-Watt! A safe and simple wireless power monitor*. URL: http://www.ladyada.net/make/tweetawatt/ (visited on 11/22/2014).

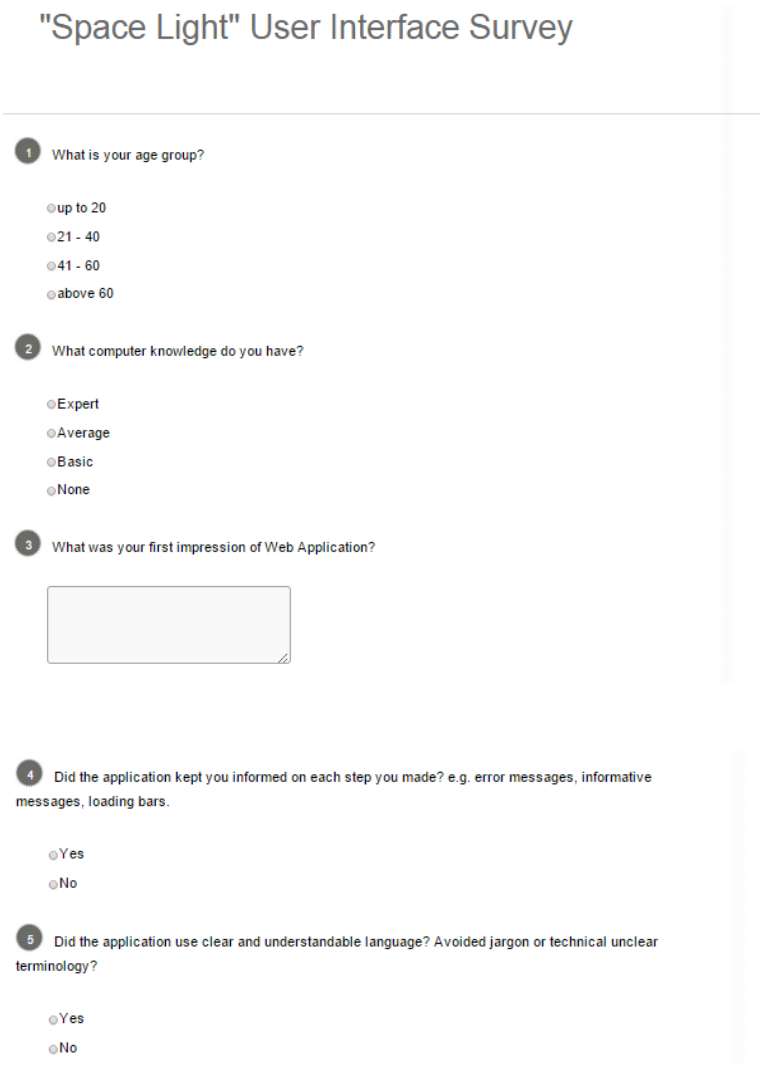
[64] Karla Conn Welch and Cindy K. Harnett. "A review of electricity monitoring and feedback systems". In: *2011 Proceedings of IEEE Southeastcon*. IEEE, Mar. 2011, pp. 321–326. ISBN: 978-1-61284-739-9. DOI: 10.1109/SECON.2011.5752958. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5752958.

[65] *World Energy Outlook 2012*. 75739 Paris Cedex 15, France: International Energy Agency, 2012. ISBN: 978-92-64-18084-0. URL: http://www.iea.org/publications/freepublications/publication/WEO2012_free.pdf.

[66] Matt Wright. *How I Structure My Flask Applications*. 2013. URL: http://mattupstate.com/python/2013/06/26/how-i-structure-my-flask-applications.html\#s2b (visited on 11/14/2014).

Appendix A

Survey



"Space Light" User Interface Survey

1 What is your age group?

- up to 20
- 21 - 40
- 41 - 60
- above 60

2 What computer knowledge do you have?

- Expert
- Average
- Basic
- None

3 What was your first impression of Web Application?

4 Did the application kept you informed on each step you made? e.g. error messages, informative messages, loading bars.

- Yes
- No

5 Did the application use clear and understandable language? Avoided jargon or technical unclear terminology?

- Yes
- No

6 Did you had full control of the application? You could move from one page to another and back?

- Yes
- No

7 Was the application consistent? Informed you on everything what was happening?

- Yes
- No

8 Did the application prevented you from causing unwanted errors?

- Yes
- No

9 Did application made you think and wonder what is going on or it was clear for you from your own experience?

10 Was the application content relevant to you? What wasn't and why? What was good and smart about the content?

11 Was all the information, error messages clear and structured? If not, what was unclear?

12 Did help and documentation was clear and helpful? If not what could be added to improve upon it?

13 Would you use this application? Support your answer.

14 What would you change or improve?

15 How would you rate the application in the scale from 1 to 10?

Rating

16 If you found any issues or bugs, could you please list them below?

1.
2.
3.
4.
5.
6.
7.
8.

17 What browser did you use?

- Chrome
- Mozilla Firefox
- Internet Explorer
- Safari
- Opera

18 If you tryed the app on other devices, could you please make a note on what device and if the app worked or was reponsive.

19 Any additional comments?

Appendix B

Test user Space Light web app documentation

What is Space Light?

It is an AngularJS application that visualizes the power consumption of various household appliances. Space Light provides the user with various graphs and statistics, to inform the user of his / hers usage. The application also provides the user with goal / target setting features to motivate user to consume less.

How it works?

The user would be provided with electricity power meters, household appliances would be plugged into the meter power strip. The meters would be connected to the Arduino Uno, the microcontroller, which gathers data from the meters and transmits it to the server. Then data is stored in the database and Space Light app is the user Interface to view the gathered and analysed data in the user friendly way.

Is all data obtained from the meters?

Unfortunately not all data is obtained from the meters as I only have two meters, most of the data is simulated the same way as it would be obtained from the meters. Simulated data is transmitted the same way as if it would be data obtained from the electricity power meters.

Why meters are not tested on the users?

This type of set up would take time to prepare for it and unfortunately I don't have time for it. Also user interface is the main part that need to be tested on the real life users.

How will I view the data if I don't have the meters to obtain the data?

You will be provided with the login details for one of the test users, to view their household usage. Test users data is simulated to ensure there is enough data to be able to test all the graphs and features provided by the application.

If I don't understand something what should I do?

The application itself has a help page that gives a brief explanation of each graph or page. If there is a major issue, you could always contact me and I will be glad to help.

What browsers Space Light supports?

It is mainly designed for Google Chrome, so please first of all do test it on Chrome. Opera also fully supports the app. Internet Explorer and Mozilla FireFox have an issue with Dashboard layout, but other than that the application is working just fine. The app was also tested on Safari and it doesn't support app layout but functionality works. Safari is not advised at all.

Is the app responsive?

Yes it is, if you are testing it on the phone or tablet make sure you use Google Chrome to ensure app is displayed in the correct format.

Performance issue

My apologies for the performance issue, the application is slow due to not being minimized during the deployment stage and having more than 100k record in the database. I hope it won't cause too much trouble, for the future reference, the app will be minimized. **Note**: loading can take more than 1 min.

Issues:

One of the dashboard graphs is not working and it will constantly show that it is loading. Also appliance second graph also is not working and will show that it is constantly loading. Profile image for some reason also is not displayed. These issues do not appear on the local environment.

Tasks:

- Login with provided user details to the app.
- Use the application, view and test all the app functionalities.
- Do register with your own details to ensure a user can actually register, no data will be displayed for you but appropriate information messages should be displayed.
- While logged in with your details do basic app check, profile picture upload, change user details, set target and other app features that don't require data obtained from the meters.
- Once you used the application, please fill in the survey, the link will be provided in the section below.

App URL: http://spacelight.info
Login details:
Email: sa@gmail.com
Password: test123!
Survey URL: http://freeonlinesurveys.com/s.asp?sid=db798dc5uuak2zs633094

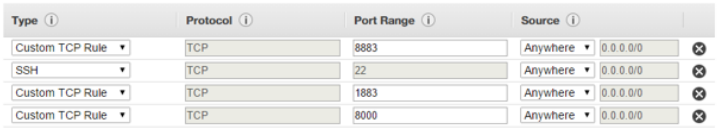
Thank you for the participation and feedback!

Appendix C

MQTT documentation

Steps to install HiveMQ on Amazon EC2 with enabled sockets:

1. Login into Amazon AWS,
2. Once you have logged in select Services and choose EC2. This will open the EC2 dashboard where you can see the overview of your resources.
3. To create new instance press "Launch Instance" button.
4. Choose an AMI. In my case I chose Ubuntu Server.
5. Choose instance type, configure instance, add storage and tag instance.
6. Configure the security groups, ensure you have:



7. After you have configured and launched EC2 instance, can install HiveMQ. For this we will need Putty and PuttyGen, can be downloaded: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
8. Once both installed, open PuttyGen and convert the .pem file (Amazon AWS Key pair) into a Putty private key.
9. Load the saved Amazon AWS key pair and select “Save private key”.
10. Once the key is saved open Putty and in Category section under Connection -> SSH -> Auth select “Browse” button and select just created private key (.ppk)
11. Go back to the Session and enter Host Name or IP (EC2 instance IP) and port number 22. Make sure connection type is SSH. And connect to the instance by selecting “Open” button.
12. Login as: Ubuntu and password is security key password

13. Once successfully logged in install Java :
    a. sudo apr-get update
    b. sudo apt-get install openjdk-7-jre-headless unzip
14. Check if Java is installed
    a. Java –version
15. Install HiveMQ
    a. wget --content-disposition http://www.hivemq.com/downloads/releases/latest
    b. unzip hivemq-1.x.x (x - version)
    c. cd hivemq-1.x.x (x - version)

16. Configure HiveMQ by editing conf/configuration.properties file. To change the file use vim filename
17. Change following values:
    a. websockets.enabled = true
    b. websockets.port = 8000
18. Save the file and run
    a. ./bin/run.sh & (& - run the process in background)
19. HiveMQ should start up and run 24/7

I followed the steps in http://forkbomb-blog.de/2013/installing-a-hivemq-mqtt-server-on-aws-ec2-with-enabled-websockets but there are not all steps.

Steps to run HiveMQ as a service:

1. Install HiveMQ in /opt directory
2. Do
    a. cd /opt
    b. sudo wget --content-disposition http://www.hivemq.com/downloads/releases/latest
    c. sudo unzip hivemq-1.x.x.zip
    d. sudo ln –s /opt/hivemq-1.x.x /opt/hivemq
    e. sudo useradd –d /opt/hivemq hivemq
    f. sudo chown –R hivemq.hivemq /opt/hivemq-1.x.x
    g. sudo chown –R hivemq.hivemq /opt/hivemq
    h. sudo cp /opt/hivemq/bin/init-script/hivemq-debian /etc/init.d/hivemq
    i. sudo chmod +x /etc/init.d/hivemq
3. Start HiveMQ daemon
    a. cd /opt/hivemq
    b. /etc/init.d/hivemq start
4. Test if daemon is running
    a. netstat –tapn (if something is running in port 8000 or 1883, then HiveMQ is running)

Note: to run HiveMQ as a service ensure the previous HiveMQ process is killed, to free up the port, for daemon to run.

I followed the steps in http://www.hivemq.com/docs/hivemq/1.4.0/ but I had to use sudo to execute the commands as a super user and use hivemq-x in line x.