A Minor project

On

Home Automation System

Submitted in partial fulfillment for the requirement of the award of the degree of

Bachelor of Technology

In

Computer Science Engineering

Department of Computer Science

HMR INSTITUTE OF TECHNOLOGY AND MANAGEMENT

Affiliated to Guru Gobind Singh Indraprastha University

Kashmere Gate, Delhi-110006(India)

Under the guidance of:

Ms. Archana Sharma

Lecturer

Submitted by:

Ritika Dhyani, Shalu Tyagi

Piyush Sharma, Suresh Joshi

DECLARATION

We hereby declare that the work presented in this project report entitled **"Homauto"** , which is a Home Automation System in partial fulfillment of the requirement for the award of Bachelor of Technology in Computer Science & Engineering., submitted to Guru Gobind Singh Indraprastha University, Delhi, is an authentic record of our own work carried out under the guidance of Ms. Archana Sharma, Lecturer, HMRITM. The work reported in this project report has not been submitted for the award of any other degree or diploma**.**

Date:

Names of team members :

Ritika Dhyani (05513303112)

Shalu Tyagi (07113303112)

Piyush Sharma (07313303112)

Suresh Joshi (08113303112)

CERTIFICATE

This to certify that Ms. Ritika Dhyani, Ms. Shalu Tyagi, Mr. Piyush Sharma and Mr. Suresh Joshi have successfully completed their project entitled 'Homauto' which is a Home Automation System for the award of Bachelor of Technology degree from Guru Gobind Singh Indraprastha University for the Batch 2012-2016.The project is based primarily on their own initiate and the work has been assessed and found satisfactory. The project fulfills the requirements as per the regulations of the university and in my opinion meets the necessary standards for submission.

DATE:

Project Guide

Ms. Archana Sharma

ACKNOWLEDGEMENT

We express our sense of gratitude to Ms. Archana Sharma and Mr. Ravi Sahu (EEE) for their valuable guidance and suggestions at all stages of this project. We are grateful and indebted to them for providing us all sorts of facilities and help needed for the project completion and clearing our concepts from time to time regarding the project. Their constructive criticism of approach to the problem and the result obtained during the course of this work has helped us to a great extent in bringing work to its present shape. It was only due to their motivation and encouragement that we could steer through the project on an honest course to the splendors of success.

DATE:

Team :

Ritika Dhyani (05513303112)

Shalu Tyagi (07113303112)

Piyush Sharma (07313303112)

Suresh Joshi (08113303112)

ABSTRACT

Home automation refers to the use of computer and information technology to control home appliances and features (such as windows or lighting). Systems can range from simple remote control of lighting through to complex computer/micro-controller based networks with varying degrees of intelligence and automation. Home automation is adopted for reasons of ease, security and energy efficiency .

Through this project we have tried to control the three LED lights using an android device connected to a LED switching device or HUB (i.e. Raspberry Pi) through Wifi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Introduction

1.1. Purpose

The purpose of this document is to present a detailed description of the Home Automation System. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the user clients and the developers of the system. It is a 'how to Operate the System' document for the clients and a guide for the developers.

1.2. Scope of Project

Home automation gives you access to control devices in your home from a mobile device anywhere in the world. The term may be used for isolated programmable devices, like thermostats and sprinkler systems, but home automation more accurately describes homes in which nearly everything -- lights, appliances, electrical outlets, heating and cooling systems -- are hooked up to a remotely controllable network. From a home security perspective, this also includes your alarm system, and all of the doors, windows, locks, smoke detectors, surveillance cameras and any other sensors that are linked to it.

Homauto aims at providing an intuitive and friendly interface that will be easy to use. It aims at enabling effective communication and collaboration among the Andoid device, LED lights. It provides Up-to-date information about the status of lights in real-time. It provides a Centralized database about the usage information of the lights.

Homauto virtually eliminates a common problem faced by other home automation systems which is that only one device can be connected to the hub. Through this system atmost 3 devices can be connected at a particular time. The device can have internet access while it is connected to our hub.User can access the data regarding the total time that the LEDs were used and can generate an estimated bill according to that usage data. Auser can choose to switch on a particular LED or all of the LEDs at the same time.

1.3. Technologies used

The entire HAS (Home Automation System) is built with the help of following three technologies:

1. JAVA for android application programming.
2. Raspberry pi as HUB or the interfacing device between the LEDs and Android Device.
3. Python with Flask as a Webserver for handeling the HTTP request from android device and switching the LEDs on/off accordingly.
4. SQL server management studio (Sqlite3)

1.3.1 JAVA

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2015, Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

Principal Design Features

Simple

Java was designed to be easy for the professional programmer to learn and use effectively. Assuming that you have some programming experience, you will not find Java hard to master. If you already understand the basic concepts of object-oriented programming, learning Java will be even easier. Best of all, if you are an experienced C++ programmer, moving to Java will require very little effort. Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java.

Object-Oriented

Although influenced by its predecessors, Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean, usable, pragmatic approach to objects. Borrowing liberally from many seminal object-software environments of the last few decades, Java manages to strike a balance between the purist's "everything is an object" paradigm and the pragmatist's "stay out of my way" model. The object model in Java is simple and easy to extend, while primitive types, such as integers, are kept as high-performance non objects.

Robust

The multiplatformed environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. Thus, the ability to create robust programs was given a high priority in the design of Java. To gain reliability, Java restricts you in a few key areas to force you to find your mistakes early in program development. At the same time, Java frees you from having to worry about many of the most common causes of programming errors. Because Java is a strictly typed language, it checks your code at compile time. However, it also checks your code at run time. Many hard-to-track-down bugs that often turn up in hard-to-reproduce run-time situations are simply impossible to create in Java. Knowing that what you have written will behave in a predictable way under diverse conditions is a key feature of Java.

Multithreaded

Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct smoothly running interactive systems. Java's easy-to-use approach to multithreading allows you to think about the specific behavior of your program, not the multitasking subsystem.

Architecture- Neutral

A central issue for the Java designers was that of code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was "write once; run anywhere, any time, forever." To a great extent, this goal was accomplished.

Interpreted and High Performance

As described earlier, Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. This code can be executed on any system that implements the Java Virtual Machine. Most previous attempts at cross-platform solutions have done so at the expense of performance. As explained earlier, the Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler. Java run-time systems that provide this feature lose none of the benefits of the platform-independent code.

Distributed

Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different from accessing a file. Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.

Dynamic

Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the Java environment, in which small fragments of bytecode may be dynamically updated on a running system.

Architecture

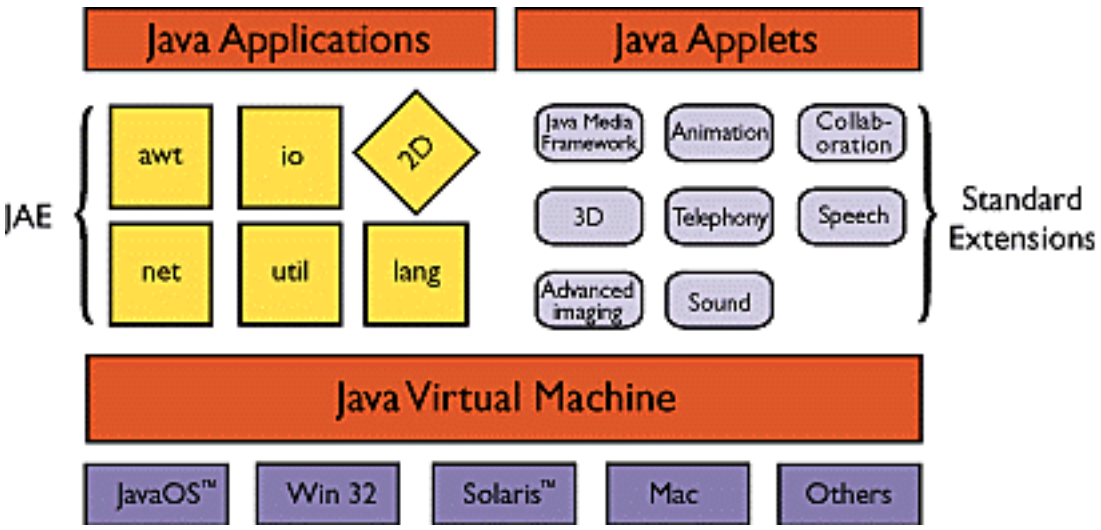

Fig1: .JAVA Architecture

1.3.2 Raspberry pi

Raspberry Pi is a small, single-board computer developed for computer science education. A United Kingdom (UK) charitable organization called the Raspberry Pi Foundation developed the device.

Raspberry PI is about the size of a credit card, has a 32-bit ARM processor and uses a Fedora distribution of Linux for its default operating system (OS). It can be programmed with Python or any other language that will compile for ARM v6.

The Raspberry Pi computer is essentially a system-on-a-chip (SoC) with connection ports. It can be operated by hooking up a USB keyboard and plugging the computer into a television. The Raspberry Pi Foundation has said it will be issuing two version of the computer: Model A will cost $25 and come with 256Mb RAM and one USB port but no network connection. Model B, which is already available and costs $35, has 256Mb RAM, comes with two USB ports and can be connected to an Ethernet network.

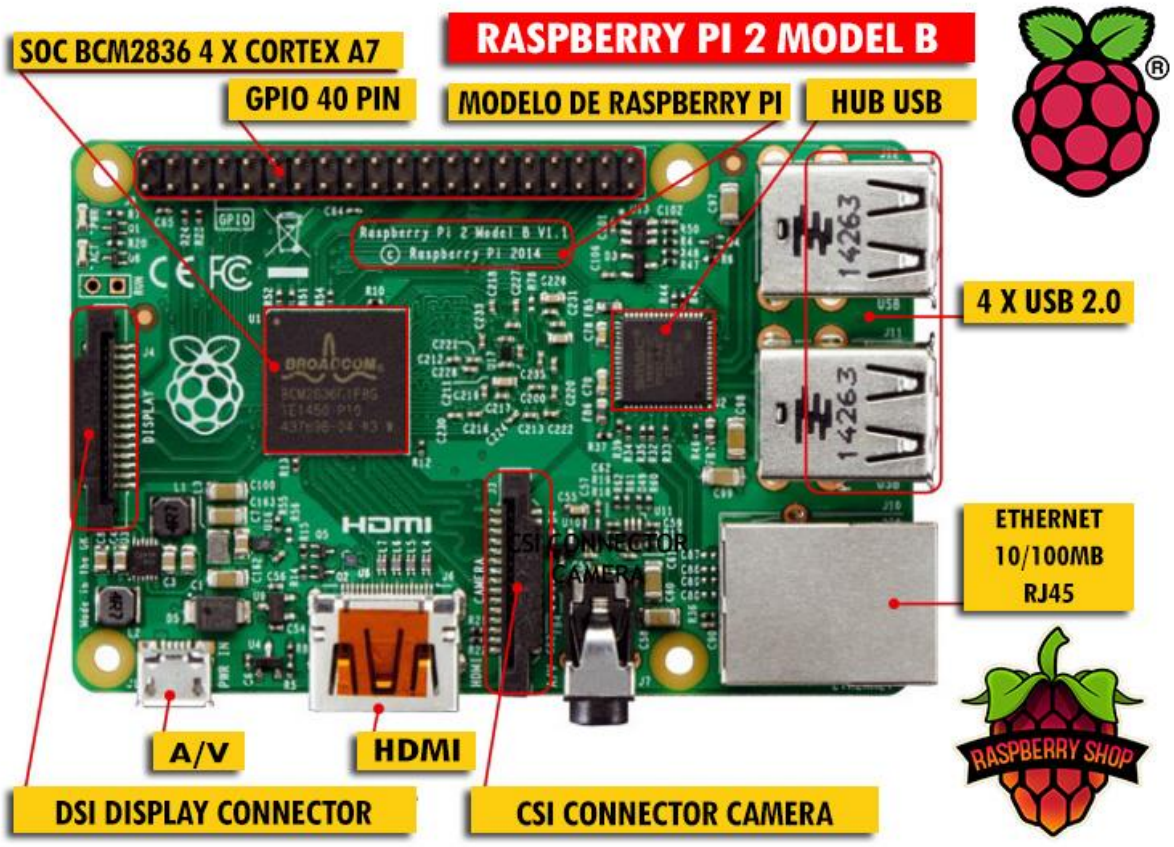

Fig2: Raspberry pi 2

1.3.3 Python (Flask)

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems. Using third-party tools, such as Py2exe or Pyinstaller, Python code can be packaged into stand-alone executable programs for some of the most popular operating systems, allowing the distribution of Python-based software for use on those environments without requiring the installation of a Python interpreter.

CPython, the reference implementation of Python, is free and open-source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

Flask is a micro web application framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. It is BSD licensed.

As of 2015, the latest stable version of Flask is 0.10.1. Examples of applications that make use of the Flask framework are Pinterest, LinkedIn, as well as the community web page for Flask itself.

Flask is called a micro framework because it does not presume or force a developer to use a particular tool or library. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Principal Features

Simple

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English (but very strict English!). This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the syntax i.e. the language itself.

Easy to Learn

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax as already mentioned.

Free and Open Source

Python is an example of a FLOSS (Free/Libre and Open Source Software). In simple terms, you can freely distribute copies of this software, read the software's source code, make changes to it, use pieces of it in new free programs, and that you know you can do these things. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and improved by a community who just want to see a better Python.

High-level Language

When you write programs in Python, you never need to bother about low-level details such as managing the memory used by your program.

Portable

Due to its open-source nature, Python has been ported (i.e. changed to make it work on) to many many platforms. All your Python programs will work on any of these platforms

without requiring any changes at all. However, you must be careful enough to avoid any system-dependent features.

You can use Python on Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC .

Interpreted

A program written in a compiled language like C or C++ is translated from the source language i.e. C/C++ into a language spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software just stores the binary code in the computer's memory and starts executing from the first instruction in the program.

When you use an interpreted language like Python, there is no separate compilation and execution steps. You just *run* the program from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your specific computer and then runs it. All this makes using Python so much easier. You just *run* your programs - you never have to worry about linking and loading with libraries, etc. They are also more portable this way because you can just copy your Python program into another system of any kind and it just works.

Object Oriented

Python supports procedure-oriented programming as well as object-oriented programming. In *procedure-oriented* languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In *object-oriented* languages, the program is built around objects which combine data and functionality. Python has a very powerful but simple way of doing object-oriented programming, especially, when compared to languages like C++ or Java.

Extensible

If you need a critical piece of code to run very fast, you can achieve this by writing that piece of code in C, and then combine that with your Python program.

Embeddable

You can embed Python within your C/C++ program to give scripting capabilities for your program's users.

Extensive Library

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI(graphical user interfaces) using Tk, and also other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the "batteries included" philosophy of Python.

Besides the standard library, there are various other high-quality libraries such as the Python Imaging Library which is an amazingly simple image manipulation library.

Architecture

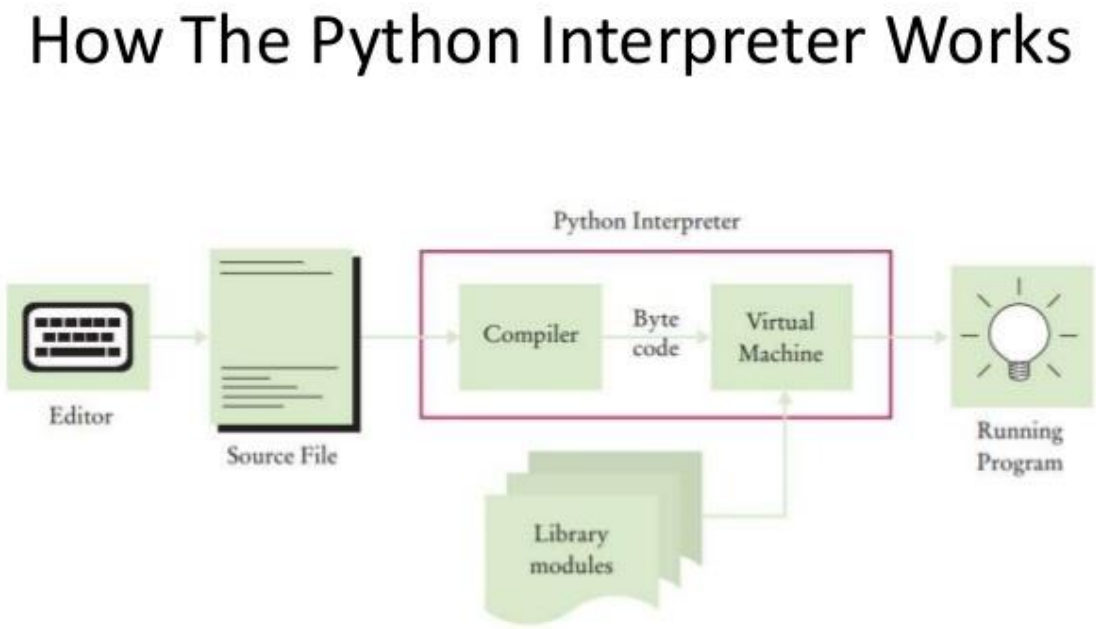

Fig3: Python Architecture

1.3.4 SQL

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most

of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

Language elements

The SQL language is sub divided into several language elements:

1. Clauses, which are in some cases optional, constituent components of statements and queries.
2. Expressions which can produce either scalar values or tables consisting of columns and rows of data.
3. Predicates which specify conditions that can be evaluated to SQL three-valued logic (3VL) or Boolean (true/false/unknown) truth values and which are used to limit the effects of statements and queries, or to change program flow.
4. Queries which retrieve data based on specific criteria.
5. Statements which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
6. SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
7. Insignificant white space is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

Queries

The most common operation in SQL is the query, which is performed with the declarative `SELECT` statement. `SELECT` retrieves data from one or more tables, or expressions. Standard `SELECT` statements have no persistent effects on the database. Some non-standard implementations of `SELECT` can have persistent effects, such as the `SELECT INTO` syntax that exists in some databases.

Queries allow the user to describe desired data, leaving the database management system (DBMS) responsible for planning, optimizing, and performing the physical operations necessary to produce that result as it chooses.

A query includes a list of columns to be included in the final result immediately following the `SELECT` keyword. An asterisk ("`*`") can also be used to specify that the query should return all columns of the queried tables. `SELECT` is the most complex statement in SQL, with optional keywords and clauses that include:

1. The FROM clause which indicates the table(s) from which data is to be retrieved. The FROM clause can include optional JOIN sub clauses to specify the rules for joining tables.
2. The WHERE clause includes a comparison predicate, which restricts the rows returned by the query. The WHERE clause eliminates all rows from the result set for which the comparison predicate does not evaluate to True.
3. The GROUP BY clause is used to project rows having common values into a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregation functions or to eliminate duplicate rows from a result set. The WHERE clause is applied before the GROUP BY clause.
4. The HAVING clause includes a predicate used to filter rows resulting from the GROUP BY clause. Because it acts on the results of the GROUP BY clause, aggregation functions can be used in the HAVING clause predicate.
5. The ORDER BY clause identifies which columns are used to sort the resulting data, and in which direction they should be sorted (options are ascending or descending). Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

1.5. Overview of Document

The next chapter, the Overall Description section, of this document gives an overview of the functionality of the product. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next chapter.

The third chapter, Requirements Specification section, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product.

Both sections of the document describe the same software product in its entirety, but are intended for different audiences and thus use different language.

2. Overall Description

2.1. Product Perspective

The objective of HAS is to develop a liable product for hustle free operation of LED using android device. It will enable to toggle the LED separately or altogether. It provides other functionality too like estimated bill generation, LED usage data, Real time status of LED HAS is intended to handle utmost 3 connections at a particular time.

2.1.1. System Interfaces

This android application will interact with database through SQL server, interact with LEDs through Raspberry pi interaction with web server will be through Python and Flask.

2.1.2. User Interfaces

The application will have user friendly and menu based interface.

The following screens would be provided:

1. A Splash Screen about Homauto.
2. A one-time login screen where username and password will be provided and access to different screens will be based on this.
3. Screen displaying the three switch buttons for a 3 different LEDs each, a bill estimate button and another switch button named cluster .
4. Screen for choosing the from date and to date ,three edittext box for inputting the watt value for each LED and Calculate bill button .
5. Screen displaying the information about the total date wise usage of LEDs and their monthly estimated bill.

2.1.3. Hardware Interfaces

1. Android device with android version of atleast 3.1 is required to run the application.
2. LEDs
3. Jumper wires
4. Bread board
5. Raspberry pi
6. Wifi Adapter supported by Raspberry pi and supporting Access point functionality.

2.1.4. Software Interfaces

1. Flask as Webserver.
2. Sqlite3 as DBMS for database.
3. Putty for accessing Raspberry pi and coding in python.
4. Android v3.1+
5. Eclipse IDE for coding in java
6. Raspbian as Raspberry pi OS.

2.1.5. Memory Constraints

1. **Processor** : Minimum 1GHz Cortex A7 Processor. Recommended 1.4 GHz Cortex A7 dual core Processor .
2. **RAM** : Minimum 1 GB RAM. Recommended 2 GB RAM.
3. **Hard Disk** : Minimum 4 MB of available space on installation device will be required for running the application.

2.1.6. Site Adaptation Requirements

The Android systems at client side will have to support the hardware and software interfaces specified in above sections.

2.2 Product Features

Use case description:

The functionality that is provided by this application is explained with the help of the following use cases:

2.2.1. SPLASH SCREEN

2.2.1.1. Brief Description

2.2.1.1.1. This use case describes Logo of Application.

2.2.1.2. Actors

Following actors participate in this use cases:

2.2.1.2.1. User

2.2.1.3. Flow of events

2.2.1.3.1. Basic Flow

2.2.1.3.1.1. This use case starts when the actor taps on the application.

2.2.1.3.1.2. The system checks whether the android device is connected to the Raspberry pi.

2.2.1.3.1.3. The system validates whether user has previously logged in or not.

2.2.1.3.2. Alternative flow

2.2.1.3.2.1. If the device is not connected to the Raspberry pi, it displays a alert dialog box.

2.2.1.4. Special Requirements

2.2.1.4.1. None.

2.2.1.5. Pre-Conditions

2.2.1.5.1. None

2.2.1.6. Post-Conditions

2.2.1.6.1. If the user had previously successfully Logged in then redirects him to the main activity if not then redirects him to login activity.

2.2.1.7. Extension Points

2.2.1.7.1. None

2.2.2. LOGIN

2.2.2.1. Brief Description

2.2.2.1.1. This use case is accessible only to those who have not previously logged in successfully.

2.2.2.1.2. System validates the user credentials .

2.2.2.2. Actors

Following actors participate in this use cases:

2.2.2.2.1. User

2.2.2.3. Flow of events

2.2.2.3.1. Basic Flow

2.2.2.3.1.1. This use case starts after the splash screen if the user has not previously logged in.

2.2.2.3.1.2. It checks whether the device is connected to the Raspberry pi and asks for id and password after clicking the login button.

2.2.2.3.1.3. It validates the credentials entered by the user

2.2.2.3.2. Alternative flow

2.2.2.3.2.1. If in the Basic Flow, the actor doesn't enters any value in either of the field it displays an alert box .

2.2.2.3.2.2. After clicking on login if the device is not connected to the Raspberry pi then it displays an alert dialog box .

2.2.2.4. Special Requirements

2.2.2.4.1. The user should not be prior logged in.

2.2.2.5. Pre-Conditions

2.2.2.5.1. The device must be connected to the Raspberry pi.

2.2.2.6. Post-Conditions

2.2.2.6.1. If the use case was successful, then he is re-directed to the main activity of the application, if not then alert dialog box is displayed informing about wrong credentials.

2.2.2.7. Extension Points

2.2.2.7.1. None

2.2.3. TOGGLE LED

2.2.3.1. Brief Description

2.2.3.1.1. This use case is accessible only to those who have previously logged in successfully.

2.2.3.1.2. System triggers the LED on or off.

2.2.3.1.3. System shows the status of the LEDs.

2.2.3.1.4. Redirects to the Date Pick Activity after the user taps on the Bill Estimate Button.

2.2.3.2. Actors

Following actors participate in this use cases:

2.2.3.2.1. User

2.2.3.3. Flow of events

2.2.3.3.1. Basic Flow

2.2.3.3.1.1. This use case starts after the login activity or directly after splash screen activity if the user has not previously logged in.

2.2.3.3.1.2. It regularly checks the status of the LEDs after every 6 seconds.

2.2.3.3.2. Alternative flow

2.2.3.3.2.1. None

2.2.3.4. Special Requirements

2.2.3.4.1. The user should not be prior logged in.

2.2.3.4.2. The device must be connected to the Raspberry pi

2.2.3.5. Pre-Conditions

2.2.3.5.1. None.

2.2.3.6. Post-Conditions

2.2.3.6.1. None

2.2.3.7. Extension Points

2.2.3.7.1. None

2.2.4. DATE PICK

2.2.4.1. Brief Description

2.2.4.1.1. This use case is accessible only to those who have previously logged in successfully.

2.2.4.1.2. System takes the 'from date' and 'to date' as the inputs from the user.

2.2.4.1.3. System takes the power consumption of each LED as an input from the user along with per unit cost.

2.2.4.1.4. Redirects to the Bill Estimate Activity after the user taps on the Calculate Bill Button.

2.2.4.2. Actors

Following actors participate in this use cases:

2.2.4.2.1. User

2.2.4.3. Flow of events

2.2.4.3.1. Basic Flow

2.2.4.3.1.1. System takes the ‘from date’ and ‘to date’ as the inputs from the user.

2.2.4.3.1.2. System takes the power consumption of each LED as an input from the user along with per unit cost.

2.2.4.3.1.3. Redirects to the Bill Estimate Activity after the user taps on the Calculate Bill Button.

2.2.4.3.2. Alternative flow

2.2.3.3.2.1. None

2.2.4.4. Special Requirements

2.2.3.4.1. The user should not be prior logged in.

2.2.3.4.2. The device must be connected to the Raspberry pi

2.2.4.5. Pre-Conditions

2.2.3.5.1. None.

2.2.4.6. Post-Conditions

2.2.3.6.1. None

2.2.4.7. Extension Points

2.2.4.7.1. None

2.2.5. CALCULATE BILL

2.2.5.1. Brief Description

2.2.5.1.1. This use case is accessible only to those who have previously logged in successfully.

2.2.5.1.2. System displays the LED usage history according to dates selected by user in Date Pick activity in Hour and minute basis along with the total bill till date and estimated bill for one month.

2.2.5.2. Actors

Following actors participate in this use cases:

2.2.5.2.1. User

2.2.5.3. Flow of events

2.2.5.3.1. Basic Flow

2.2.5.3.1.1. This use case is accessible only to those who have previously logged in successfully.

2.2.5.3.1.2. System displays the LED usage history according to dates selected by user in Date Pick activity in Hour and minute basis along with the total bill till date and estimated bill for one month.

2.2.5.3.2. Alternative flow

2.2.3.3.2.1. None

2.2.5.4. Special Requirements

2.2.5.4.1. The user should not be prior logged in.

2.2.5.4.2. The device must be connected to the Raspberry pi

2.2.5.5. Pre-Conditions

2.2.5.5.1. None.

2.2.5.6. Post-Conditions

2.2.5.6.1. None

2.2.5.7. Extension Points

2.2.5.7.1. None

2.2.6. UPDATE DATABASE

2.2.6.1. Brief Description

2.2.6.1.1. This use case acts as interface between the LEDs and Android Device through Http webserver created using Flask and Python and also this use case resides in Raspberry pi .

2.2.6.1.2. Handles the requests coming from the android device and gives response accordingly.

2.2.6.1.3. Updates and retrieve data from the sqlite 3 database.

2.2.6.2. Actors

Following actors participate in this use cases:

2.2.6.2.1. None

2.2.6.3. Flow of events

2.2.6.3.1. Basic Flow

2.2.6.3.1.1. Listens to the Http request and provides the response accordingly.

2.2.6.3.2. Alternative flow

2.2.6.3.2.1. None

2.2.6.4. Special Requirements

2.2.6.4.1. The webserver should be in start state

2.2.6.4.2. The device must be connected to the Android Device

2.2.6.5. Pre-Conditions

2.2.6.5.1. None.

2.2.6.6. Post-Conditions

2.2.6.6.1. None

2.2.6.7. Extension Points

2.2.6.7.1. None

2.3 User Characteristics

1. EDUCATION LEVEL: The user should be able to understand the Standard English language and should be a computer literate.
2. EXPERIENCE: The user must have the knowledge of how to work on Android Devices and Raspberry pi.
3. TECHNICAL EXPERTISE: The user must be comfortable with the use of Android Application on a mobile.

2.4 Operating Environment

1. Android Based Operating System having version 3.2 and above.

2.5 Design and Implementation Constraints

1. Due to limited small number of supported user which is 3. The application may become unresponsive or there may be delay in application.
2. SQL DBMS is used.
3. JAVA language is used for application Development.
4. Python Language is used with FLASK for webserver development
5. Raspberry pi is used as an interface between the Android device and LEDs.

2.6 Assumptions and Dependencies

1. Each user has access to an Android Device.
2. The user has basic computer knowledge.
3. The application used is of version greater then 3.0

3. Specific Requirements

3.1 Database Tables

The database include following Master tables:

1. switchtimedata: This table includes the specific and basic details about which LED was toggled on which date and till what time(in milliseconds).

3.2 Analysis

3.3.1 Context Diagram

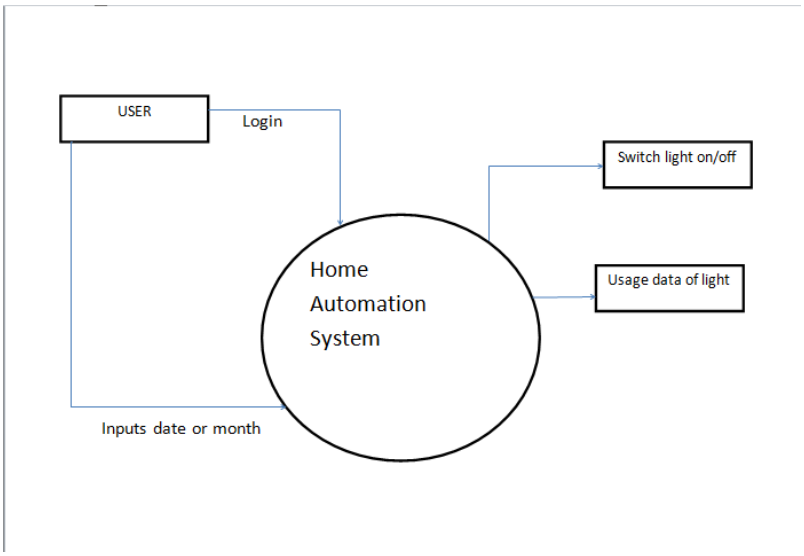

Fig4 : Context Diagram

3.3 Diagrams

3.3.1 Use Case Diagram

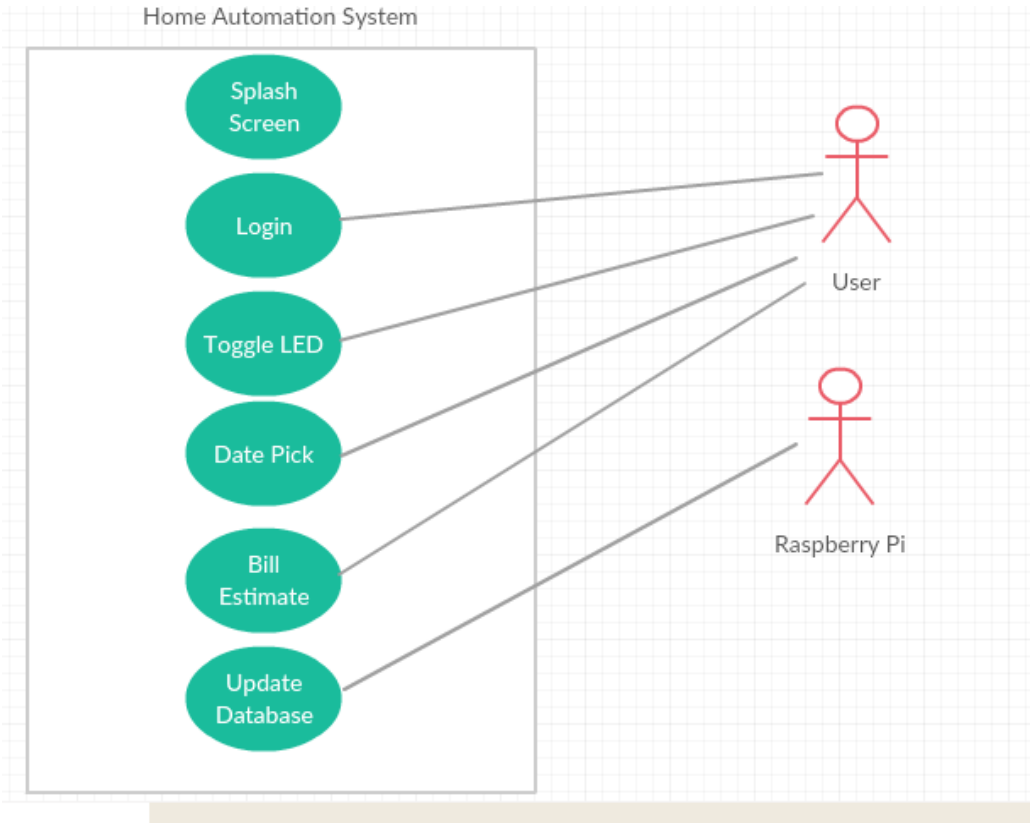

Fig5 : Use Case Diagram

3.3.2 Class Diagram

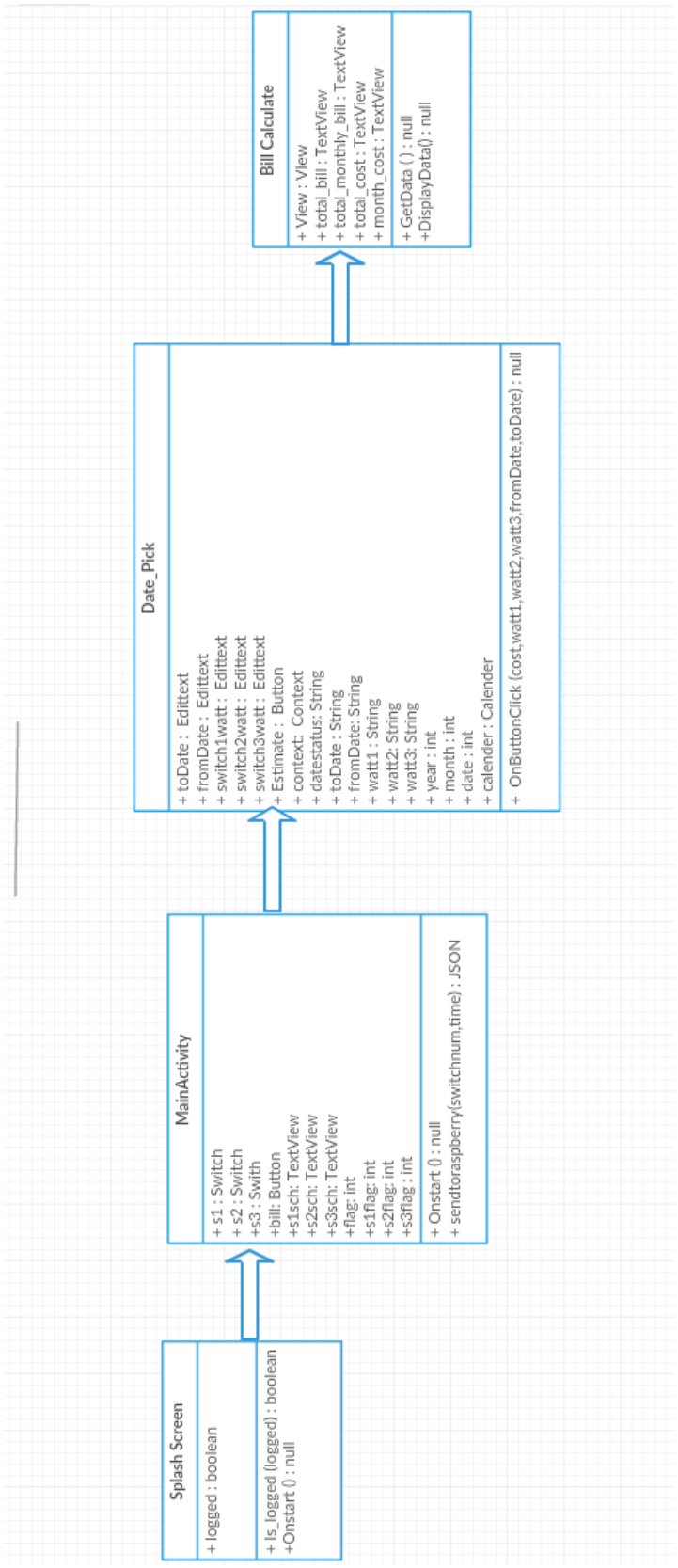

Fig6 : Class Diagram

3.3.3 Component Diagram

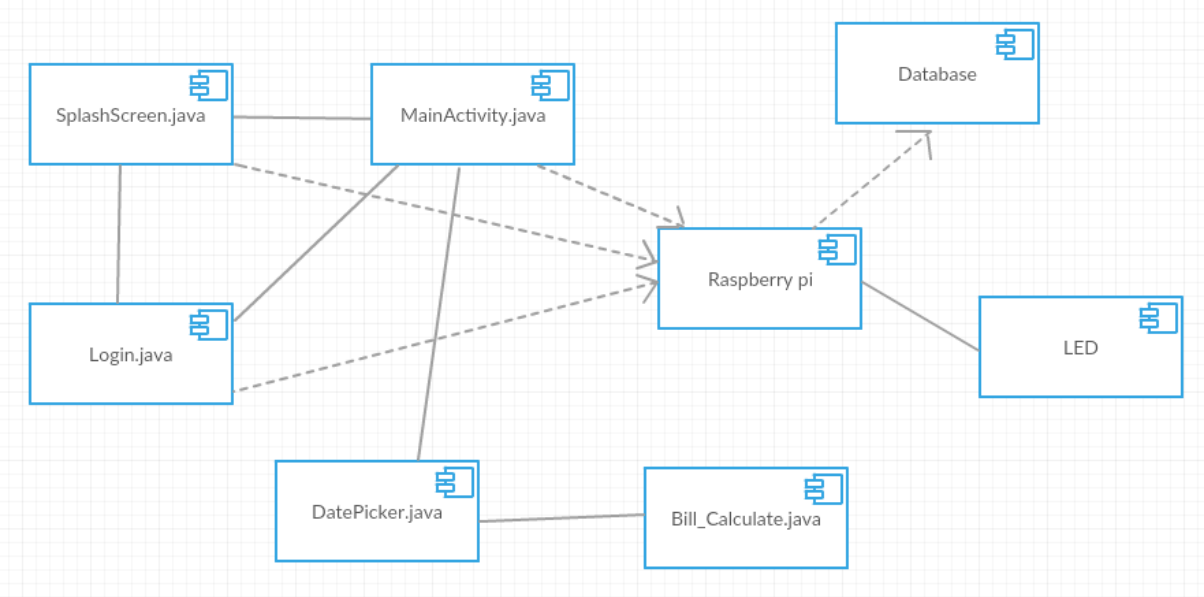

Fig7 : Component Diagram

3.3.4 Sequence Diagram

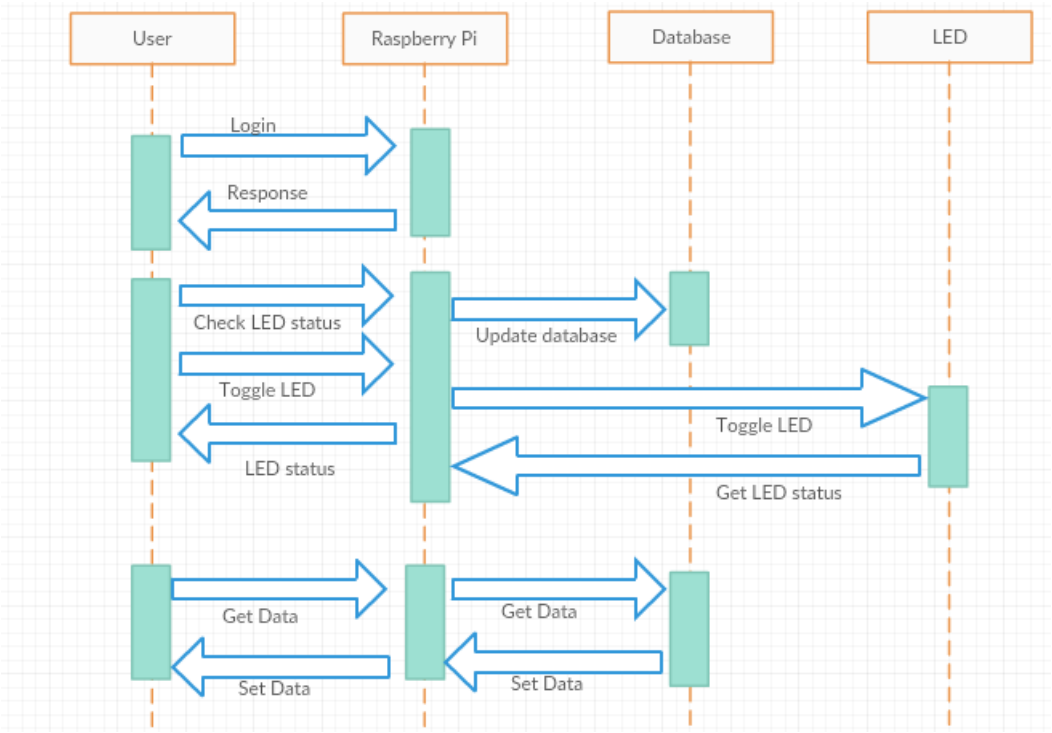

Fig8 : Sequence Diagram

3.3.5 Activity Diagram

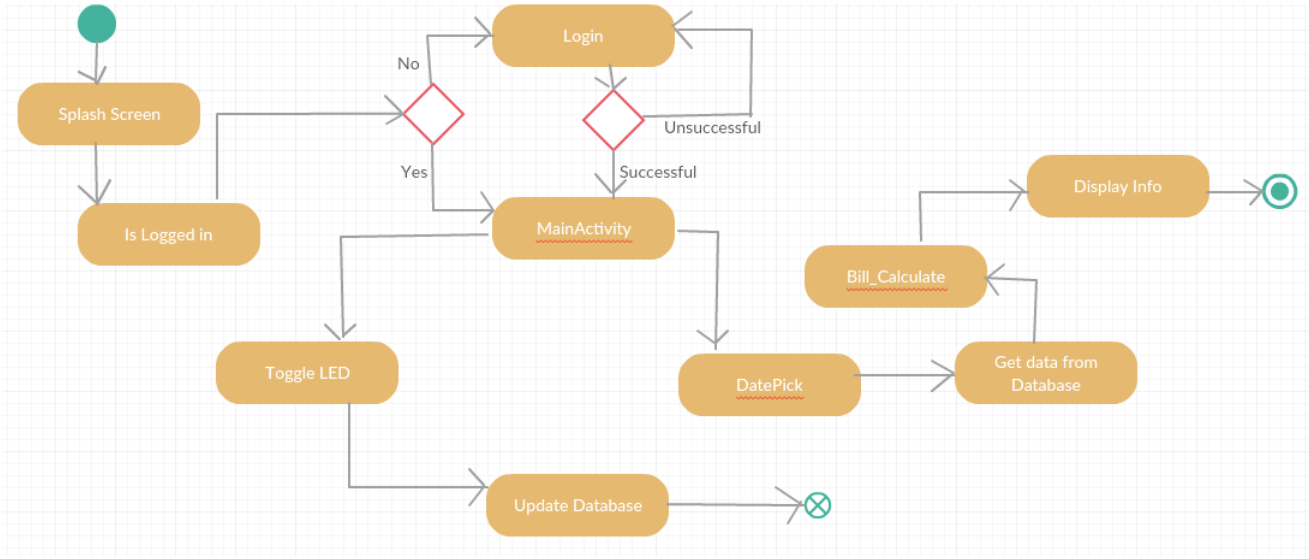

Fig9 : Activity Diagram

3.3.6 Deployment Diagram

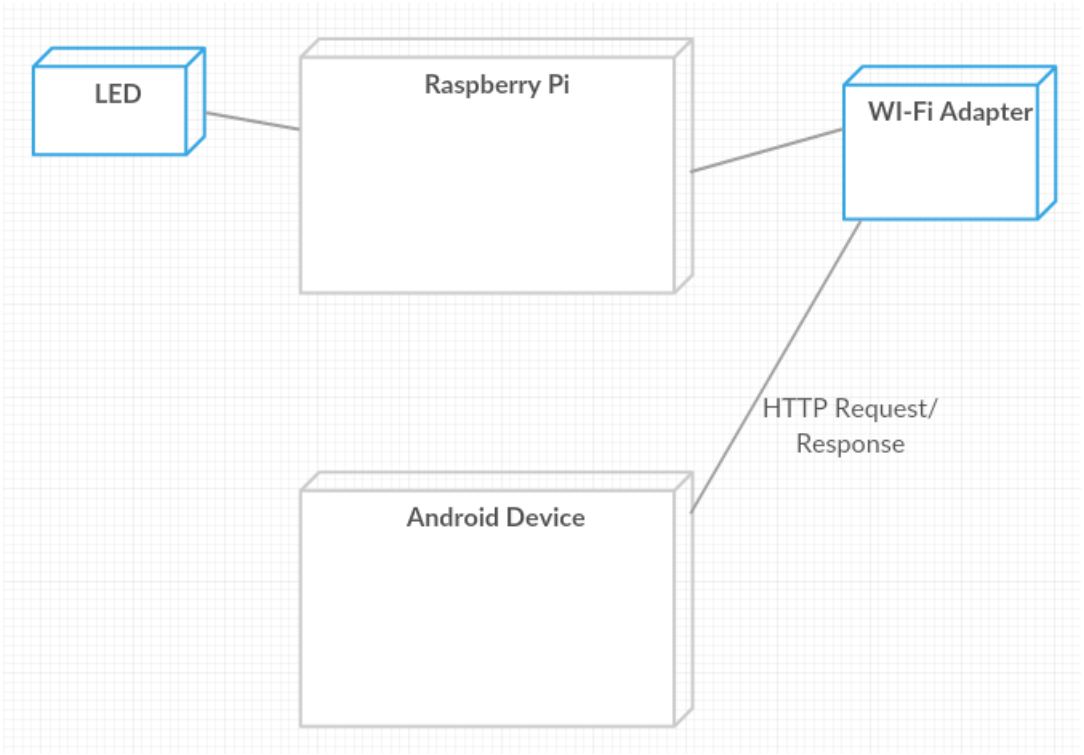

Fig10 : Deployment Diagram

DATABASE TABLES

Table1: switchtimedata

id	switchnum	date	time
56	switch1	20151103	20
57	switch1	20151103	10
58	switch2	20151103	10
59	switch3	20151103	40
60	switch3	20151103	50
61	switch1	20151103	50
62	switch2	20151103	50
69	switch3	20151103	7431
70	switch2	20151103	8315
71	switch1	20151103	9293
72	switch1	20151103	2779
73	switch3	20151103	581
74	switch1	20151104	1816
75	switch2	20151104	4802
76	switch3	20151104	1164
77	switch2	20151104	20004
78	switch2	20151104	43558
79	switch1	20151104	2736
80	switch2	20151104	1552
81	switch3	20151104	817
82	switch1	20151104	2700
83	switch2	20151104	2681
84	switch3	20151104	2678
171	switch2	20151106	36191592
172	switch3	20151106	4119
173	switch1	20151106	2454
174	switch2	20151106	2454
175	switch3	20151106	2454
176	switch1	20151106	1556
177	switch2	20151106	1556
178	switch3	20151106	1556
179	switch3	20151106	1692

4. Other Non-Functional Requirements

4.1 Performance Requirements

- More than 3 devices must not be connected at particular time.
- They must not toggle the switches at same time.

4.2 Security Requirements

- All areas of the application are password protected, completely confidential.

4.3 Software Quality Attributes

- **Maintainability**: It will be easy to incorporate new requirements into the existing modules and therefore it is easy to update it and maintain it for a longer time.
- **Portability:** The application is easily portable on any android based operating system.
- **Correctness:** The application is correct up to the developer's requirements.

5. SNAPSHOTS

SPLASH SCREEN

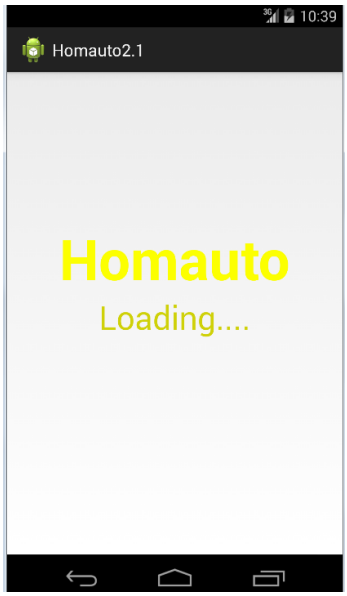

Fig11 : Splash Screen

MAINACTIVITY

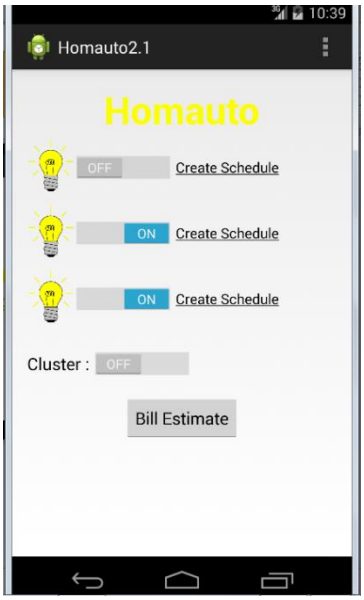

Fig 12 : MainActivity

DATEPICKER

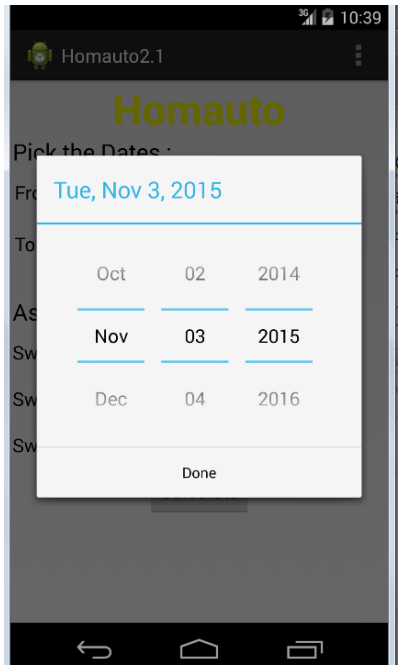

FIG13 : DatePicker

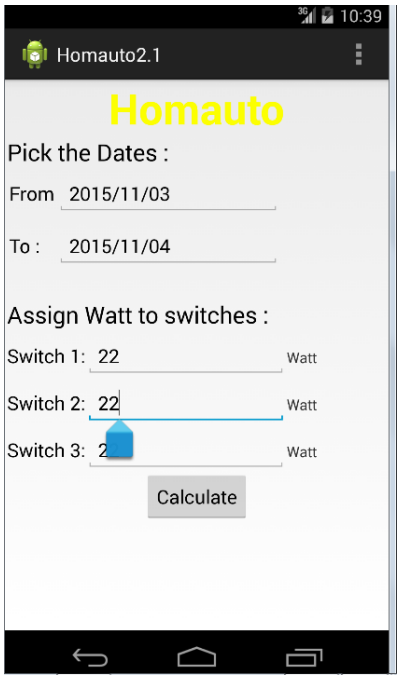

Fig14 : DatePicker

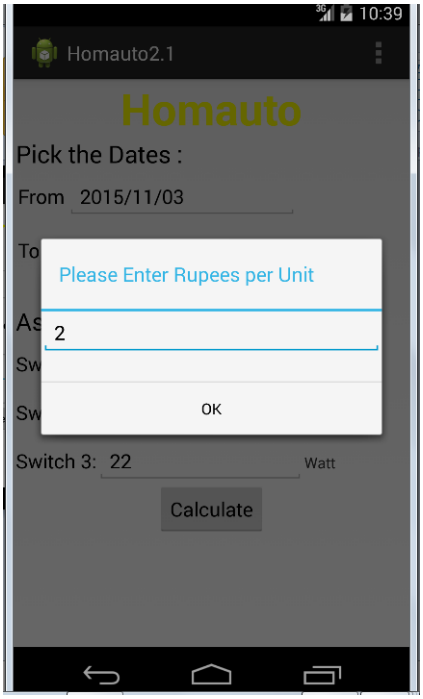

Fig15 : DatePicker

BILLCALCULATE

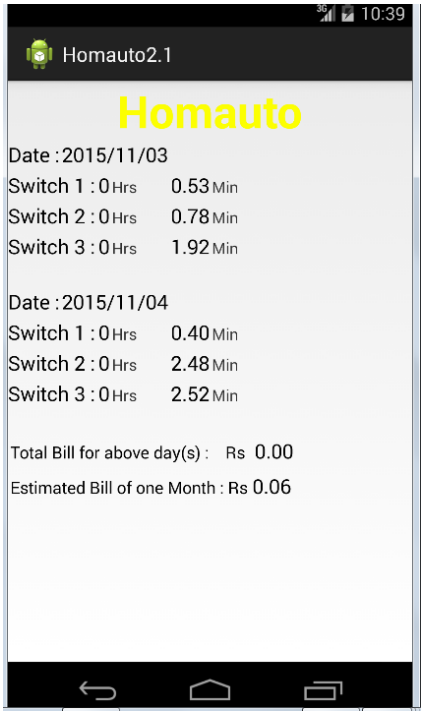

Fig16 : BillCalculate

6. Future Enhancements

6.1 Enhanced GUI.

6.2 A master App which could decide which switches are accessible by slave apps.

6.3 Scheduling of switches.

6.4 User could dynamically assign the name and switch number.

6.5 Graphically representing the usage of the LEDs.

6.6 Live implementation using the Solid State Relays with the real 220 volt appliances.

6.7 Use of Broadcast Receiver instead of timer.

6.8 Using a more secure method other than NAT for accessing the appliances over internet.

6.9 Using a more cheap hub.

6.10 Validations in Application.

7. References

7.1 visual-paradigm.com

7.2 Learning Python the Hard Way

7.3 Java the Complete Reference

7.4 pages.jh.edu

7.5 www.ibiblio.org

7.6 mattrichardson.com

7.7 en.wikipedia.org

7.8 whatis.techtarget.com