Project No.01

University Room Reservation System

Presented by

1. Mr. Tulathorn Sripongpankul	58070503412
2. Miss Yosita Sitthiporn	58070503424
3. Mr. Arnan Hirunratanakorn	58070503438

Advisor

Dr. Sally E. Goldin.

"I've read and approved the content of this report"

…….

(……)

Advisor

University Room Reservation System

Mr. Tulathorn Sripongpankul
Miss Yosita Sitthiporn
Mr. Arnan Hirunratanakorn

A Project Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Engineering
Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi
Academic Year 2018

University Room Reservation System

Mr. Tulathorn Sripongpankul
Miss Yosita Sitthiporn
Mr. Arnan Hirunratanakorn

A Project Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Engineering
Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi
Academic Year 2018

Project Committee

…… Advisor
(Sally E. Goldin Ph.D.)

…… Committee
(Asst. Prof. Sanan Srakaew, M.S.)

…… Committee
(Assoc. Prof. Naruemon Wattanapongsakorn, Ph.D.)

…… Committee
(Asst. Prof. Phond Phunchongharn, Ph.D.)

Project Title	University Room Reservation System
Project Credit	6 Credits
Project Participant	Mr. Tulathorn Sripongpankul
	Miss Yosita Sitthiporn
	Mr. Arnan Hirunratanakorn
Advisor	Dr. Sally E. Goldin
Degree of Study	Bachelor's Degree
Department	Computer Engineering
Academic Year	2018

Abstract

This project tries to improve the existing manual room reservation system at KMUTT. We integrate information to develop a web application that makes room reservation easier and more convenient for everyone. Currently, the room reservation system takes a lot of effort and time to reserve a room. Our web application is introduced to improve the convenience and performance of reservation system.

Our university room reservation system provides the ability to reserve the room via a web application without coming to book at the university. The users will be able to select the time slot and the system would show the available rooms for booking. The users are allowed to edit or cancel the reserved room anytime they want.

For the analog door lock, the users need to get the key from IOT key box which is installed in front of each room and return the key when finished unlock the door. Each room will have their own key box installed in front. Moreover, our system is designed to be able to connect to the digital door lock that is provided to unlock the room with the application at the reserved time.

The web application can be separated into two parts which includes server and client. On the server side, the server receives the data when the client reserves any rooms and the data is sent to be stored in MariaDB database. These data can be used for monitoring and analyzing the users' behavior. Also, our database stores all information about the ongoing uses of the rooms for courses and it also needs some basic info about rooms, such as whether there are labs with computers, and how many students they can hold. On the client part, a client only needs to connect to the server in order to make a query (see rooms) or a request (reserve room). When the room reservation is completed, the client sends the data to the server and shows the pop-up window with "Reservation Completed" message and the notification will be sent to admin.

หัวข้อโครงงาน	ระบบการจองห้องเรียนในมหาวิทยาลัย
หน่วยกิตของโครงงาน	6 หน่วยกิต
จัดทำโดย	นายตุลธร ศรีพงษ์พันธ์กุล
	นางสาวโยษิตา สิทธิพร
	นายอานันท์ หิรัญรัตนากร
อาจารย์ที่ปรึกษา	Dr. Sally E. Goldin
ระดับการศึกษา	วิศวกรรมศาสตร์บัณฑิต
ภาควิชา	วิศวกรรมคอมพิวเตอร์
ปีการศึกษา	2561

บทคัดย่อ

โครงงานนี้ เป็นโครงงานทดลองทำการศึกษา เกี่ยวกับระบบการจองห้องเรียน เพื่อใช้งานใน สำหรับระบบการจองห้องเรียนออนไลน์ล่วงหน้า เพื่อเพิ่มความสะดวกสบาย และประหยัดเวลา ในการจองห้องเรียน เนื่องจาก ระบบดั้งเดิม ผู้จองจะต้องติดต่อเพื่อจองห้องเรียน จากผู้ดูแลและทำตามกระบวนการการจอง ณ เวลาที่จะใช้งาน ซึ่งในบางครั้งผู้ใช้งาน ก็เลือกที่จะหาสถานที่อื่น เพราะการจองห้องเรียนในระบบดั้งเดิมนั้น ใช้เวลามาก ที่สำคัญ คือมีหลายขั้นตอน ดังนั้นผู้จัดทำโครงงาน จึงพัฒนาระบบจองห้องเรียนออนไลน์ เพื่อให้ผู้ใช้งาน สามารถจองห้องเรียน ได้อย่างสะดวกและรวดเร็ว รวมถึงการวางแผนเพื่อจองห้องเรียนล่วงหน้าอีกด้วย

ระบบการจองห้องเรียนในมหาวิทยาลัย ถูกสร้างขึ้นในรูปแบบของเว็บแอพพลิเคชั่น สามารถใช้งานได้ในทุกแพลตฟอร์ม ทำให้ผู้ใช้งาน สามารถจองห้องเรียนในมหาวิทยาลัย เพื่อการใช้งานต่าง ๆ ได้ผ่านทางเว็บไซต์ โดยไม่ต้องผ่านกระบวนการที่ยุ่งยาก ที่สำคัญ ผู้ใช้งานสามารถเลือกเวลาและห้องที่จะจองได้ และ สามารถแก้ไขหรือยกเลิกได้ทุกเมื่อ

หากผู้ใช้งานได้ทำการจองห้องเรียนไว้ เมื่อถึงเวลาใช้งาน ผู้ใช้ สามารถรับกุญแจที่ตู้ฝากกุญแจ ซึ่งถูกติดตั้งอยู่หน้าห้องเรียนแต่ละห้อง เพื่อทำการปลดล๊อกห้อง และคืนกุญแจเมื่อปลดล๊อคเรียบร้อย มากกว่านั้น ตัวระบบของโครงงาน ได้ถูกออกแบบมารองรับสำหรับห้องที่ล็อกด้วยระบบดิจิตอล ในอนาคต ผู้ใช้สามารถปลดล็อกได้จากเว็บแอพพลิเคชั่นได้ทันที

Acknowledgement

Our project will not be completed without the support from another person. We would like to say something to all of these people which are the important part of our success.

First of all, we would like to acknowledge our advisor, Dr. Sally E. Goldin. She shared her time to guide us for months helping our project run smoothly. She also helps us with the infrastructure by provide one server to run our system. Moreover, her experience is super important to us which let us learn many things, also her trust in our idea. We are very pleased to thank her very much here.

For assisting with LDAP, Mr. Ekachai, one of KMUTT computer center staffs, he helped us to provide KMUTT LDAP. We would also like to thank for our friends for their help and support. We would not able to finish our work without them

We also really want to thank for KMUTT for the projects room, library and computer center, which is where our project is developed. Moreover, we would like to thank to every committee for their guideline.

Last but not least, we want to thank you to Stackoverflow for helping us solving problem as same as developer blog.

Contents

Page

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Problem Statement and Approach

Computer Engineering students and professors need to use rooms in the department for exam preparation, make-up classes and lab quizzes. During exam week, students demand places for reviewing and preparing for upcoming tests. As available places in the university are limited, students need to ask for more seats. The best solution for Computer Engineering students is to open a room for tutoring in their department. However, the process of asking for a room requires several steps, for example, writing a letter to the department, and finding the staff to open the room. At the same time, professors who need rooms for extra classes or exams confront the same complicated process. Thus, they also have trouble with booking process.

Despite the intention to manage rooms by using a manual timetable, there are many problems such as the staff booking a room for a teacher at the same period as student's reservation of the same room.

1.2 Project Type(s)

- Addressing the needs of a specific group of stakeholders
- Possible commercial product

1.3 Objectives

- Provide more convenient room booking service in the university.
- Make class planning easier for teachers or staff
- Avoid reservation conflicts and mistakes.
- Save effort for the staff who are responsible for looking after room reservations in every building.

1.4 Scope

1.4.1 Core feature

1. Web application for university room reservation system has two systems inside which include:
    a. Booking system: Allow teachers and university students to book the rooms
    b. Admin system: Allow admin to manage the room usage such as opening and closing times, reschedule, view room using history etc.
2. Database for collecting and recording all of data in the system
3. Application programming interface (API) to allow each service to communicate with others.
4. Report generation subsystem to produce various views of the database (currently handled by different spreadsheets) such as courses and rooms for one professor, courses and rooms for one student section, booked times for a particular room during each day of the week.
5. The entire user interface is bilingual for supporting Thai and English.

1.4.2 Component: Kiosk key box

We see the problem that most of the room in our department are still using analog locks (that is a lock that still needs a physical key) so we've planned to make a kiosk key box installed next to each room for borrowing and returning a key in short time. Each kiosk key box serves one specific room. This option adds more scope such as

- Hardware for key box (Wire, motor, sensor, metal, etc.)
- Microcontroller + network module for controlling the box and communicating with the booking system via its API
- RFID tag and receiver for tracking the key
- Unlocking the kiosk in front of the room via secure code.

1.5 Tasks and Schedule

1.5.1 Deliverables for Term 1

- System Architecture Design
- Database Implementation
- API Implementation
- Web Application User Interface design and mockup
- Kiosk Design (hardware and software)
- Implement kiosk Prototype

1.5.2 Completed task for Term 1

- System Architecture Design
- Database Implementation
- Kiosk Design
- User Interface Design

1.5.3 Deliverables for Term 2

- API Implementation
- Kiosk Key Box Implementation
- Database Implementation
- Web Application Implementation
- End to End test

1.5.4 Gantt chart for term 1

Table 1.1 Gantt Chart for Term1

Task No.	Task name	Responsibility	AUG		SEP				OCT				NOV				DEC	
			3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
1	System Architecture Design																	
1.1	System Architecture Overview	Tul, Hon, Jane																
1.2	Design integration test	Tul, Hon, Jane																

1.3	API Function Design	Tul																	
1.4	Database Schema	Hon																	
1.5	Create Integration test driver	Tul																	
2	Database Implementation																		
2.1	Create Table via ER Diagram	Hon																	
2.2	Add mockup data	Jane																	
2.3	Test and Refactor	Hon, Jane																	
3	API Implementation																		
3.1	Write API Test script (Unit test)	Tul																	
3.2	Write API function	Tul																	
3.3	Implement with database	Tul, Hon																	
3.4	Integration test	Tul, Hon, Jane																	
4	User Interface Design																		
4.1	Design Pages	Jane																	
4.2	Design Use Flow	Jane																	
5	Kiosk Design																		
5.1	Kiosk box design	Jane																	
5.2	Kiosk material research	Jane, Hon																	
5.3	Kiosk circuit design	Jane, Hon																	
5.4	Kiosk budget plan	Jane, Hon, Tul																	
6	Documentation and Presentation																		
6.1	Prepare & Rewrite proposal	Tul, Hon, Jane																	
6.2	Prepare & Rewrite report	Tul, Hon, Jane																	
6.3	Term presentation	Tul, Hon, Jane																	

1.5.5 Gantt chart for term 2

Table 1.2 Gantt Chart for Term 2

Task No.	Task name	Responsibility	JAN		FEB				MAR				APR				MAY	
			3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
1	Web Application Implementation																	
1.1	User Interface Implementation	Jane																
1.2	Use Flow Implementation	Jane																
1.3	Plug to API	Tul, Jane																
2	Kiosk Key Box Prototype Implementation																	
2.1	Equipment Preparation	Hon																
2.2	Circuit Implementation	Hon, Jane																
2.3	Create 3D Printed Model	Hon, Jane																
2.4	Plug to API	Tul, Hon																
3	Database Implementation																	
3.1	Create Database	Hon																
3.2	Plug to API	Tul, Hon																
4	API																	
4.1	API Implementation	Tul																
4.2	Plug to Database	Tul, Hon																
4.3	Plug to Web Application	Tul, Jane																
4.4	Plug to Kiosk Key Box	Tul, Hon																
5	Test and Fix bug																	
5.1	Test the system	Tul, Hon, Jane																

5.2	Fix Bug	Tul, Hon, Jane	
6	Login and Authentication Integration		
6.1	Building Login function	Tul	
6.2	Build up authentication	Tul	
6.3	Integration and test	Tul	
7	Documentation and Presentation		
7.1	Prepare and Rewrite Report	Tul, Hon, Jane	
7.2	Term Presentation	Tul, Hon, Jane	

Chapter 2
Background, Theory and Related Research

2.1 Related work

Gonzaga University has created online room reservation system [1] as an in-house system for providing the ability to reserve the room online and pick up the key at the reserved time from the staff. Our University Room Reservation System also provides online room reservation system but we provide access to the rooms via a custom-made kiosk key box that allows to open analog door lock (door that still needs a key to unlock) by getting the key from Kiosk key box after enteri a secure code.

Another example is room and resource booking system by Ecobook [2]. This is a commercial product for providing the ability to connect the booking system to Outlook, Office365, Yahoo, and Gmail account. The rooms can be booked manually or when the user adds to the calendar a task that needs to use the room, the room is automatically reserved. The system books the room matched with the size of business. Booking process is confirmed by an email. Admin can set the operating hours, room level visibility for different group of users. You need to contact Ecobook for enterprise pricing information. You can also submit a request for a free trial to see if the software is a good fit for your organization. Compared to our project, Ecobook room and resource booking system can only use with digital door lock but cannot use with the analog door lock. It does not capabilities suitable for our university where most of the rooms still need a key to unlock the door.

Another example is SIT Booking System which is an inhouse system by KMUTT School of Information Technology, provided for students to reserve the room within a faculty. It is the web-based booking system which provides the feature to book the room and see the time table of each room. The system can reserve a room as daily reservation, weekly reservation, monthly reservation. Also, their system is bilingual, offering user interaction in either Thai and English. The system needs SIT account for login, so this means it can only be used by the staff or student in the faculty. However, they still need to get the key from the staff to open the room. Our University Room Reservation System has included all the features that they provide. In addition, our system can be accessed by anyone with a KMUTT account. Our system can reserve a room as recurring reservation for specific user such as, class for fourth year student section A and B. Also, the Kiosk Key Box is provided in our system for the convenience of the user to open the door.

We focus on developing the software to make efficient and user-friendly software. Also, we have studied the user needs to develop software so we can better satisfy the needs of our users. We do not specifically focus on hardware because kiosk key box can always be changed in material, size, etc. in order to match with the specific key.

From this survey, we have found that existing university room reservation systems provide convenience and comfort for the users those reserved the rooms with digital door lock but there is no information confirmed that the users prefer the online room reservation system for analog door lock due to the fact that they still need to pick up the key to open the room.

2.2 Core concepts

In this section, we discuss core technical concepts behind this project in order to provide theoretical information.

2.2.1 HTTP Protocol

HTTP protocol is an application layer protocol that allows a program to fetch resources such as HTML documents. It is a foundation of data exchange for the World Wide Web. Requests are initiated by the recipient, usually the web browser. A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, image, video, script, etc.

HTTP consists of nine request types:

1. GET – Requests resource, should only retrieve data
2. HEAD – Ask the response identical to the GET request, but no body.
3. POST – Using to submit an entity specified resource, often causing a change in state or side effect to the server.
4. PUT – Replace all current representations of the target resource with the request payload.
5. DELETE – Remove specified resource.
6. CONNECT – Establishes a tunnel to the server identified by the target resource.
7. OPTIONS – Describing the communication options for the target resource.
8. TRACE – Performing a message loop-back test along the path to the target resource.
9. PATCH – Using to apply partial modifications to a resource.

The most commonly request types used are GET and POST. All requests in the HTTP protocol are expressed as plain text. Thus, HTTP is platform-neutral and can be used with any computer hardware, operating system and web server.

2.2.2 HTTPS Protocol

Hypertext Transfer Protocol Secure (HTTPS) is the secure version of HTTP. The data sent in HTTPS between browser and web server will be encrypted. HTTPS web pages normally use one of two secure protocols to encrypt the data which are Transport Layer Security (TLS) or Secure Sockets Layer (SSL). Both protocols use an 'asymmetric' Public Key Infrastructure (PKI) system. An asymmetric system needs two keys to encrypt the communications, which are public and private key. The data encrypted by public key can decrypted by private key and vice-versa.

To use HTTPS, it has two best practices for implementing HTTPS

1. Use robust security certificates
   The website has to be obtained a security certificate as a part in order to enabling HTTPS, which is issued by certificate authority (CA). When setting up the certificate, keep in mind that you have to choose 2048-bit key for ensuring a high security level. To get started:

- Get your certificate from a reliable CA that offers technical support
- Decide the kind of certificate you need:
  - Single certificate for single secure origin (e.g. www.example.com)
  - Multi-domain certificate for multiple well-known secure origins (e.g. www.example.com, cdn.example.com, example.co.uk).
  - Wildcard certificate for a secure origin with many dynamic subdomains (e.g. a.example.com, b.example.com). The domain that have wildcard certificate shown that all the subdomain that is in the domain will be secured.

2. Use server-side 301 redirects instead of 200

2.2.3 JavaScript

JavaScript is a high-level interpreted programming language. It is also known as dynamic, weakly typed, prototype-based and multi-paradigm. JavaScript is a core web technology supporting the World Wide Web. It allows a website to be interactive without the need to access server resources or code in order to allow a website to be interactive.

Since it is a multi-paradigm language, it supports functional, event-driven, and imperative programming styles. It has APIs for working with dates, text, basic manipulation of the DOM (see section 2.2.4), arrays, and regular expressions. However, the language does not include any I/O, such as storage, networking, or graphics facilities.

At present, JavaScript is used not only on the client-side, but also on the server-side. JavaScript can now be executed by embedded engines in web servers and databases, and in non-web programs such as word processors and PDF software. Moreover, the runtime environments for JavaScript are also available for writing mobile and desktop applications, including desktop widgets.

2.2.4 Document Object Model (DOM)

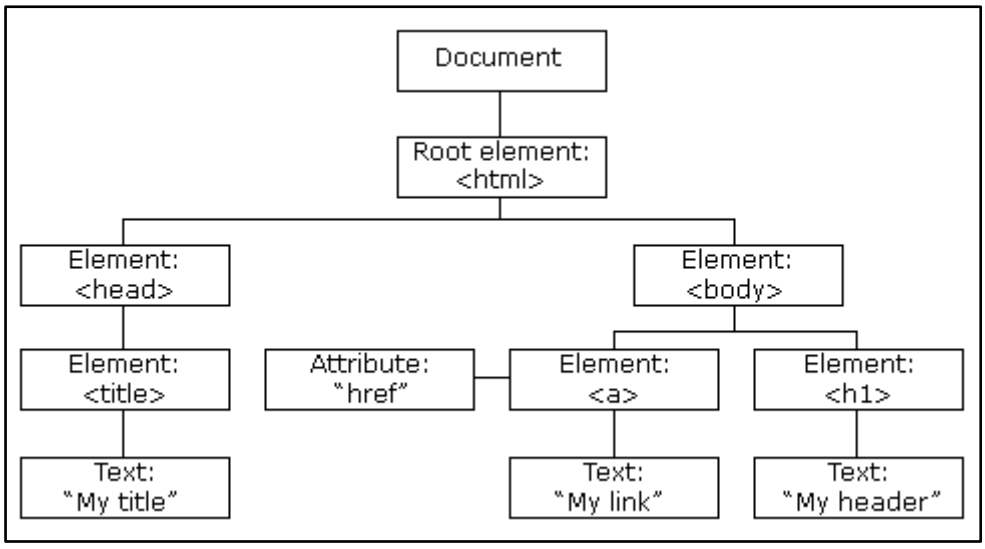

Figure 2.1 DOM representation of the example table [15]

Document Object Model is a data structure that allows programs to interface with HTML and XML documents. The DOM represents the document as nodes in a tree, as shown in Figure 2.1. Programming languages can access and change document information by specifying the appropriate node. The current DOM standard is named W3C DOM and has been implemented in all modern browsers. In HTML the DOM defines:

- HTML elements as an object.
- Properties of HTML elements.
- The events for all HTML elements.

In other word, we can say that the HTML DOM provides a standard representation for getting, changing, adding or deleting HTML elements.

JavaScript has the ability to access and modify the DOM elements You can use the JavaScript DOM API for the document or window elements to manage the document itself or to get to the children of that document, which are the diverse elements in the web page.

2.2.5 Container vs. Virtual Machine

Virtual Machines and Containers are two approaches for isolating and abstracting applications. Figure 2.2 provides a graphical summary of the two approaches.

A virtual machine (VM) runs software on top of physical servers to emulate a particular hardware system. A hypervisor, or a virtual machine monitor, is software, firmware, or hardware that creates and runs VMs. It is what sits between the hardware and the virtual machine and is necessary to virtualize the client machine. Virtualizing the machine means that the software provides all the capabilities, systems and subsystems that would be available on a physical computer.

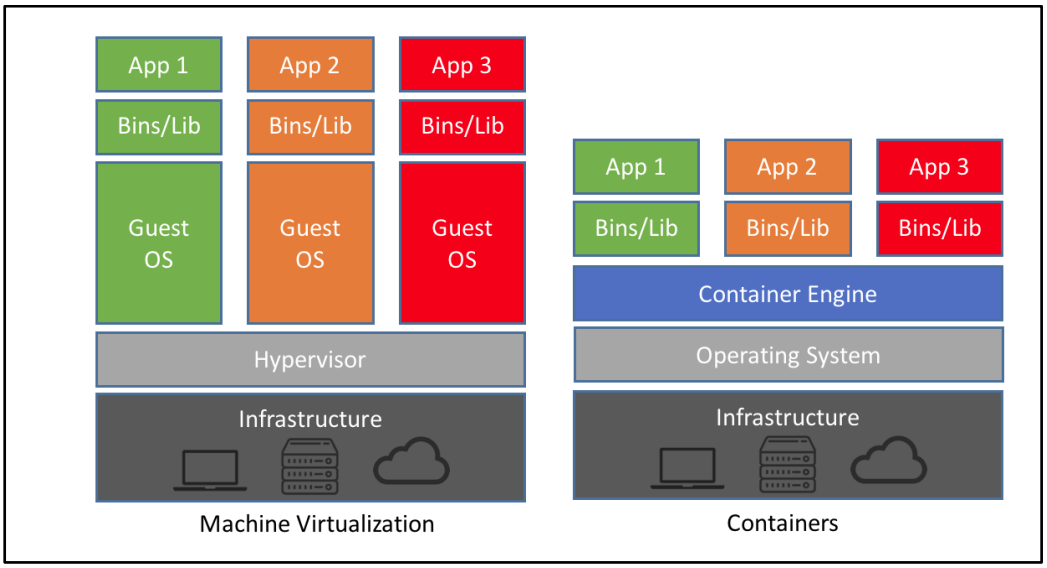

Figure 2.2 Comparing the architecture between Virtual Machine (Machine Virtualization) and Containers [8]

Each virtual machine has its own OS, called Guest OS, which is on top of Host OS. That means each VM has its own binaries, libraries, and applications. A VM may be many gigabytes in size.

Containers are a more lightweight approach to application abstraction. A container is designed to sit on top of Host OS and provides isolated systems that run on a single server or host OS. Each container shares the host OS kernel and, usually, the binaries and libraries, too. Shared components are read-only. Container sizes tend to be only megabytes and take just seconds to start, versus gigabytes and minutes for a VM. Containers also reduce management overhead. A container will be based on an operating system, which mean bug fixed, update, etc. need to do on the operating system. On the other hand, containers do not provide the OS-independence offered by virtual machines.

2.2.6 RESTful API

A RESTful API (Representational State Transfer) is an application programming interface that uses HTTP requests to GET, POST, PUT and DELETE data. The RESTful API is based on REST which is an approach to communicate in web services development.

In a RESTful web service, requests made to a resource's URI will receive a response with a payload formatted that can be either HTML, XML, JSON, or some other formats. The response can confirm that some alterations have been made to the stored resource, and the response can provide hypertext links to other related resources or collections of resources.

The advantages of RESTful API are:

1. Simple to build and adapt.
2. Low use of resources.
3. Process instances are created explicitly.
4. Client does not require routing information.
5. Clients can have a generic ‘listener’ interface for notifications.

The alternative to REST is SOAP (Simple Object Access Protocol) which is mostly used in enterprise software. The big wall of SOAP contains lot of limitation such as being limited only to request or to response model and requiring that all content be wrapped in XM.

2.2.7 Relational database

A relational database is a database based on the relational model of data. Relational means that it stores information in a set of tables which can be linked together (related) using shared data fields. The software system that used to maintain relational databases is a relational database management system (RDBMS). Typically, relational database management systems use Structured Query Language (SQL) for maintaining and querying the database.

In the relational model, all data must be stored in tables. Each table includes rows and columns. Each table has one header and a body. The header is the list of columns in the table. The body is the set of data that belong to the table. One set of data is called row.

Each table in a relational database usually represents one "entity type", or category of information. Each row in the table is considered to be one instance of the category (one entity). Columns, which are also called attributes, define the properties of entities in each row. The rows usually include one or more unique values (the primary key), which identifies the entity in the row. The relationship between tables can constructed with foreign keys which is an attribute that links to the primary keys of another table.

Atomicity, Consistency, Isolation and Durability (ACID) is the concept of database transactions which use to handle errors and unexpected events such as power failures, etc.

A transaction is the database operation which fulfills the ACID concept. The ACID characteristic will be explained below.

1. Atomicity → Every transaction represents one unit. The entire transactions will fail and the change will not be made, if any part of the transaction fails.
2. Consistency → The data after a transaction completes is guaranteed to be internally consistent. That is, no partial changes will be allowed.
3. Isolation → Transactions are independent and an ongoing transaction will not be visible to other transactions.
4. Durability → If a failure occurs before the data are committed, the system will return to the data to its previous correct state.

2.2.8 Library vs. Framework

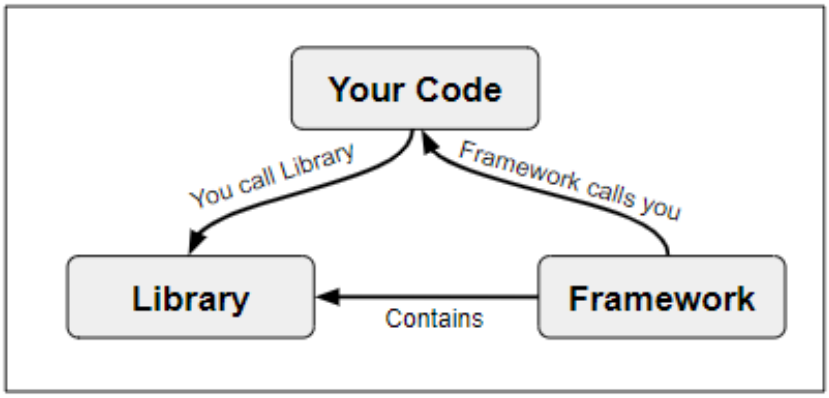

Figure 2.3 Diagram of relationship between Library, code, and framework [16]

A library is just a collection of functions or class definitions. The functions or classes and methods normally define specific operations in a domain specific area. For example, there are some libraries of mathematics which let developers execute complex calculations, without needing to see or code the implementation of how an algorithm works. Libraries save time and facilitate code reuse.

Frameworks are also designed to save development time and effort. However, a framework is more structured than a library. All the control flow is already provided, with predefined "white spots" or hooks where you should fill in your code. A framework defines a skeleton where the application defines its own features to fill out the skeleton. Then, your code will be called by the framework. The benefit is that developers do not need to worry whether if a design is good or not, but about implementing domain specific functions.

2.2.9 Progressive Web App

Progressive Web App (PWA) is a set of best practices to make a web application function similar to a desktop or mobile application. PWA uses modern web capabilities to deliver an app-like user experience. PWA consists of four concepts:

1. Reliable – Should launch and give users meaningful content even it offline.
2. Responsive– Should be able to adapt to different screen sizes, operating system and orientations to ensure that the user experience is great for all users and all devices.
3. Engaging– Should provide engagement like as native application by having icons, notification and so on.
4. Secure– To use PWA ability, the web-application have to serve over HTTPS.

PWA depends on two important files, manifest.json and serviceworker.js. Manifest is a file that gives meta information about the web app. It contains information like the icon of the app (which a user sees after installing it in their app drawer), background color of the app, name of the app, short name, and so on. Service worker is a client-side proxy that tells the browser which element in web apps should be cached into the device. That makes it possible for a PWA to work offline in some functions like native apps.

2.3 Software technologies

In this section, we will talk about software that we are using for implementing this project including frameworks and libraries.

2.3.1 NodeJS

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. As we know, JavaScript was designed for client-side scripts. NodeJS allows you to write JavaScript code that runs outside browser, often on the server side. Node.js has an event-driven architecture capable of asynchronous I/O. This design aims to optimize throughput and scalability with many I/O operation that help developers provide real-time web applications.[7]

2.3.2 ReactJS

React is a JavaScript library for building user interfaces. It is not a framework. A React-based application can be divided into many components. Each component file contains business logic and HTML markup. You can use JS library or Flux which is a new kind of architecture that complements React to communicate between components.

In React, elements are separated into the primary building blocks called "Components". React will compose the all components into "Component-tree." Each component provides a render() method that is used for rendering a view. When the render() method is called, it creates the intermediate-DOM (Virtual-DOM) on the root component and recursively going down the Component-tree and build-up the Virtual-DOM before converting to the real HTML-DOM.

React works by using the concept of state and props. State objects can be used to pass the data from component to layout that is the root component of React and are used only inside the component while props pass the data from parent component to child component. This is illustrated in Figure 2.4.

There are many community frameworks that have been based on React such as NextJS and GatsbyJS. Moreover, React also has several Command-line Interface (CLI) created by the React developer community in order to speed the development of a React project likes create-react-app (Command: "create-react-app <name>". Create repository for developing React), react-static (Command: "react-static <create,start,build> <Arguments>". Use to create or manage react static dependency to one or many project), etc.

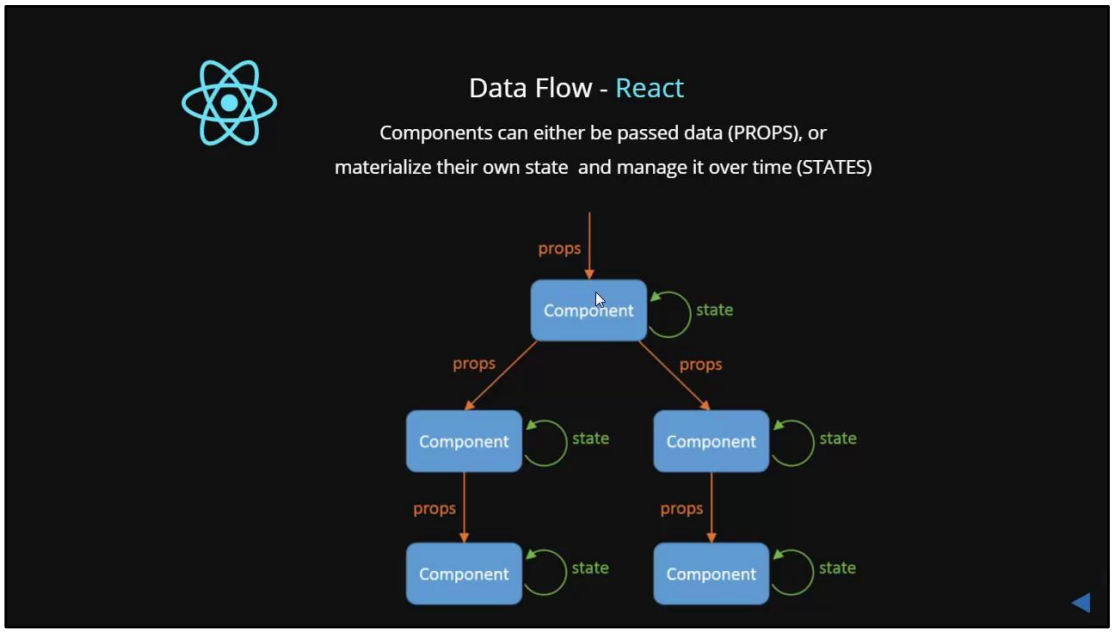

Figure 2.4 React data flow diagram [4]

2.3.3 MariaDB

MariaDB is open source relational database software developed by MariaDB Corporation. MariaDB is a forked version of MySQL due to the fact that MySQL was acquired by Oracle Corporation. MariaDB supports Unix, Windows, Solaris, Mac OS, BSD and Linux. It is an enhanced version of MySQL. MariaDB is fast, scalable and robust. Also, it can be installed with plugins and many tools which makes it very easy to use and can be used comprehensively in any use cases. Currently, MariaDB also supports GIS and JSON features. It has come to replace MySQL due to the better performance and functionality.

2.3.4 FastifyJS

Fastify is a NodeJS framework for making a web server. It is an open-source project under MIT License. It first developed by lead engineer of LetzDoIt, Tomas Della Vedova and the Node.js Foundation Technical Steering Committee (Node.js TSC) member, Matteo Collina. This framework is focused on providing the best developer experience with the least overhead and a powerful plugin architecture. Using Fastify will help a developer create a web server more easily. The core features of FastifyJS include:

1. **High performance:** The framework designed to handle 30,000 requests per sec. That is the highest speed for NodeJS framework.
2. **Extensible:** Fully extensible with hooks, plugins and decorators.
3. **Schema based:** Fastify recommends using a JSON Schema to verify routes and sort your outputs. Internally Fastify compiles the schema with very high-performance function.
4. **Logging:** Fastify chooses to use the best logger to reduce the cost of logs.
5. **Developer friendly:** the framework is designed to be expressive and to help developers in their everyday use, without losing efficiency and safety.

2.3.5 MomentJS

Moment JS is a library used for validating, manipulating, parsing, etc. the time in JavaScript. MomentJS provides the ability to read the time format from MySQL and reform the Date/Time to be common format.

2.3.6 SweetAlert2

SweetAlert2 is the library for JavaScript which will come to replace the JavaScript alerts. SweetAlert2 can do everythings that JavaScript alerts can do, but in addition it is responsive, and more flexible compare to the JavaScript alerts.

2.3.7 Amazon Relational Database Service

Amazon Relational Database Service (Amazon RDS) is an online relational database cloud which provides the feature to configure the database, with easy access. It provides the ability to auto back-up data, auto prepare data, auto update, etc. which can help the user to save time and effort.

Amazon RDS allows the user to control the amount of storage required, performance needed, and number of I/O operations. It provided many tools to manage databases including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database and SQL Server. You can always move your local database into Amazon RDS at any time. Amazon RDS is available from free to costly, depend on the feature and amount of data that user chooses. Each database engine will cost differently.

2.3.8 Dotenv

Dotenv is a zero-dependency module that provides the ability to read the variables from .env files which is separate from the code and will not affect the code.

2.3.9 Postman

Postman is software that provides API development environment. It has a user interface which makes it easy to create request to test API services. You will able to create the request to test it manually or create automate test.

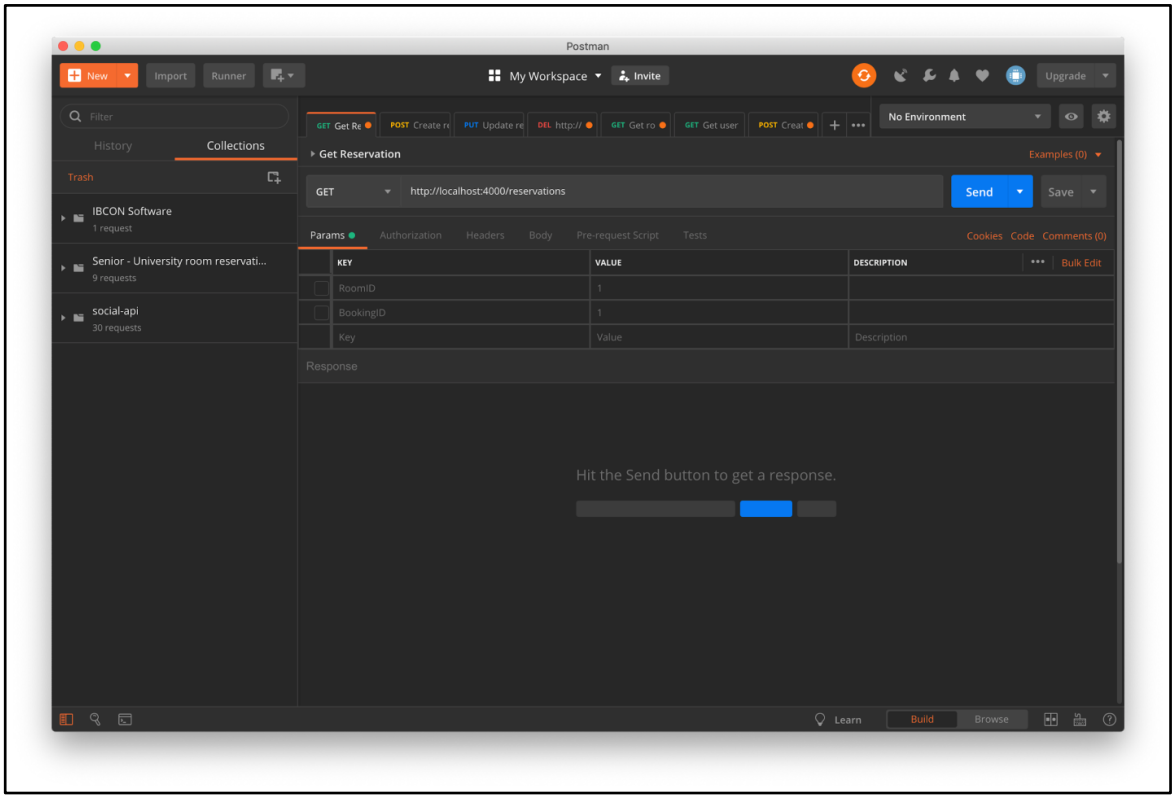

Figure 2.5 Screenshot for postman

2.3.10 SendGrid

SendGrid is a cloud-based email service from United State. It provided the feature to create custom template with custom data. It offers two types of email which are marketing email and transaction email. It can send many types of email by different trigger such as receipt, shipping notifications, sign-up confirmations, newsletters, etc.

2.3.11 Knex

Knex is a SQL query builder which has ability to manage relational database from MySQL, PostgresQL, SQLite3, WebSQL and Oracle. It is able to run on Node.js and browser. It can handle transactions, polling, steaming queries, promise, etc. without the need to write normal SQL code

2.3.12 Bookshelf

Bookshelf is a JavaScript Object-Relational Mapping (ORM)for Node.js, built on the Knex SQL query builder. Bookshelf aims to provide a simple library for common tasks when querying databases in JavaScript, and forming relations between these objects.

2.3.13 Axios

Axios is a library that provide the ability to send HTTP request to API endpoints. Axios can be used by writing a JavaScript code or used with frameworks. It can be used in front-end and backend.

2.3.14 Software Usage and Cost

The usage of each software on each side of the system, also the cost of its will be shown in the table 2.1

Table 2.1 Software Usage and Cost

Software	**Usage**		**Type**	**Cost**
	Server Side	**Client Side**		
NodeJS			Open Source	Free
ReactJS			Open Source	Free
MariaDB			Open Source	Free
FastifyJS			Open Source	Free
MomentJS			Open Source	Free
SweetAlert2			Open Source	Free
Amazon RDS			Proprietary	Depend on traffic (Free tier about 20 USD)
Dotenv			Open Source	Free
Postman			Proprietary	Free, 18$/Month for more features.
SendGrid			Proprietary	Free (100 email/day), Price depend on each plan.
Knex			Open Source	Free
Bookshelf			Open Source	Free
Axios			Open Source	Free

2.4 Hardware technologies

The Kiosk Key Box is the hardware that will be created to keep the key in front of each room. The system will be controlled by Raspberry Pi Zero W with the 3x4 keypad, RFID scanner and the monitor display the status of the key box.

In hardware technology, we will talk about related hardware that will be used in the project including sensor, microcontroller, motor, etc.

2.4.1 RFID

Radio-Frequeny Identification (RFID) is an approach that uses radio waves to detect the information on a tag which is usually attached to an object. The tag can be detected from several feet away from the detector. RFID systems have two parts which are tag and reader.

1. The tag contains two parts which are microchip that stores information and antenna to send and receive a signal. Each tag has unique serial number which is not duplicated by any others.
2. Reader to interrogate the tag which can detect and read the information from the tag.

As shown in Figure 2.6, to read the information encoded on a tag, the reader sends a signal to the tag by using antenna. The tag responds to the signal with the information in the microchip. Then the reader sends the results to RFID computer program.

There are two types of RFID tags:
1. Passive: using interrogator's radio wave energy to relay its stored information.
2. Battery powered: contains a small battery to power the relay of information.

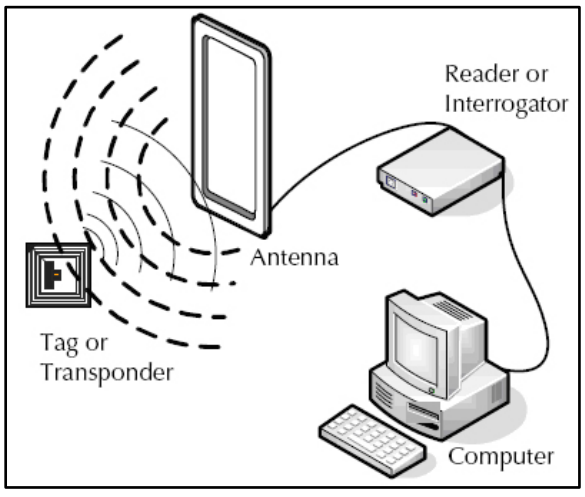

Figure 2.6 Diagram of how RFID works [5]

There are numerous benefits to RFID technology, although it has some limitations and drawbacks as well. An RFID reader can scan a tag as long as it is within frequency range. It does not have any line-of-sight limitations. For instance, bar coding requires the reader to be close to the barcode to scan it while RFID systems can automatically pick up tag IDs from a distance.

RFID systems can also scan multiple items simultaneously when other ID systems typically have a single or limited identifier for each object. Some RFID tags contain read-write memory which allows you to add or change data. You can implant tags into objects or use plastic coverings to protect them. This makes them more robust than some other ID tags.

RFID systems can still have problems. Although readers can scan through most non-metallic materials, they have problems with metal and water. The fact that you can scan multiple objects in a range is a benefit but sometimes tag collision may occur if a reader picks up signals from multiple tags at the same time. Reader collision may be an issue if two readers interfere with each other's signals.

2.4.2 Raspberry Pi Zero W

The Raspberry Pi is a series of small single-board computers. The Raspberry Pi Zero is half the size of a Model A+. It provides wireless and Bluetooth connectivity. It has a 1GHz, single-core CPU with 512MB Ram which is enough for our project. Also, 40 pin headers are provided on the board. The price is around 350 THB. We choose the Raspberry Pi Zero because the price is much cheaper than normal Raspberry Pi and the size is smaller. The power usage by the different Raspberry Pi devices is shown in Figure 2.7 below.

	Zero	Zero W	A+	A	B+	B	Pi2B	Pi3B
	/mA	/mA	/mA	/mA	/mA	/mA	/mA	/mA
Idling	100	120	100	140	200	360	230	230
Loading LXDE	140	160	130	190	230	400	310	310
Watch 1080p Video	140	170	140	200	240	420	290	290
Shoot 1080p Video	240	230	230	320	330	480	350	350

Figure 2.7 Pi Power Usage table for each model [17]

2.4.3 3x4 Matrix Keypad

The Keypad is a part created by Arduino which allows your board like Arduino, Raspberry Pi, etc. to read the data input by keypad. The keypad contains numbers, * and #.

2.4.4 L298N Motor Driver

The L298N is the motor driver which can control the speed and direction of a motor. The motor should have voltages between 3V-35V to work probably.

2.5 Software Internationalization

We design software internationalization to be compatible with most of the platform, language, etc. For the languages, we design to separate all text from the code. Each language will be kept in different file as UTF-8 which is dominant character encoding for the World Wide Web (WWW). Also, the font size will be different depending on the language.

For the date and time format, we design date to display as DATE/MONTH/YEAR format as default format in Thailand. Date will be represented as DD/MM/YYYY. However, date format can be changed in customization option. Time will be based on Coordinated Universal Time (UTC) which is world regular time standard. The system time will be UTC+7 which mean 7 hours ahead of UTC time due to the University location. Also, the time will be shown as HOUR/MINUTE/SECOND. Represent as HH:MM:SS

For user interface (UI), there are many platforms with different screen size, resolution, operating system, etc. So, the UI will be developed using responsive web design which is the technique that makes the user interface always show the good result on every device by sharing the same code and URL.

Chapter 3
Design and Methodology

3.1 Methodology

University Room Reservation is a project for addressing the needs of a specific group of stakeholders. Our stakeholders are staff, professors and CPE students. These stakeholders may have several problems. For example, they cannot plan for a meeting due to the fact that they are not sure that they can get a room or not. Also, the current room reservation process has many steps and requires filling in many documents to get a room. Also, there is the need to contact the staff who take care of the current room reservation process. To be confirmed, the problem may be applied on these steps to find the root of problem and to initialize our project

1. Survey the needs of people all over university, but especially CPE students.
2. Gather the information about the current KMUTT booking system.
3. Plan the algorithm for the booking application
4. Design System architecture - System architecture is the most important part, because it is an overview of the project.
5. Design Database - Database need to have good design. Bad database design will affect the performance of the application.
6. Design User Interface - User interface is important because it is the first thing that attract the user.
7. Get access to LDAP student account information so that we do not have create a separate, burdensome authentication component.

3.2 User roles

The system has two user roles: normal user and admin user. Originally, we planned to separate faculty users from student users, but the LDAP system does not provide sufficient information for us to distinguish these two sub-categories of normal user.

The functional capabilities available to each user role are as follows

- **Normal User**:
  - Book / Edit / Cancel the room
  - See the time available for booking in each room.
  - Look at booking history
  - Access to normal classroom, teacher meeting room, etc.
- **Admin**:
  - Book / Edit / Cancel the room for student and teacher.
  - Book the room in schedule for entire semester
  - Cancel the booking that reserve by others user.
  - Add a new room to the system.
  - See the time available of the room in a semester
  - Edit / Delete the room from the system.
  - Look at room booking history.
  - Ban the user
  - Generate reports.

3.3 Feature

In this section, we will talk about features that our university room reservation system will be provided.

Table 3.1 Main Features

Features	***Detail***
Reserve the room	*This feature is the main component of this system. In order to reserve a room, user has to specify their needs like room size, equipment, etc. User who has admin role can assign booking to another user.*
Cancel the reserved room	*If the user has reserved a room, they can cancel the reservation for that room. To cancel reservation, tab on current reservation menu and select the reservation that user want to cancel.*
Room management (Admin)	*Admin role has permission to modify the room time usage. To do this, select on the room that you want to edit information. Click on edit button, it will show the form that let you edit the room information such as operating time, equipment in the room, room size, etc.*

3.4 Use case diagram

3.4.1 Normal User Use Case Diagram

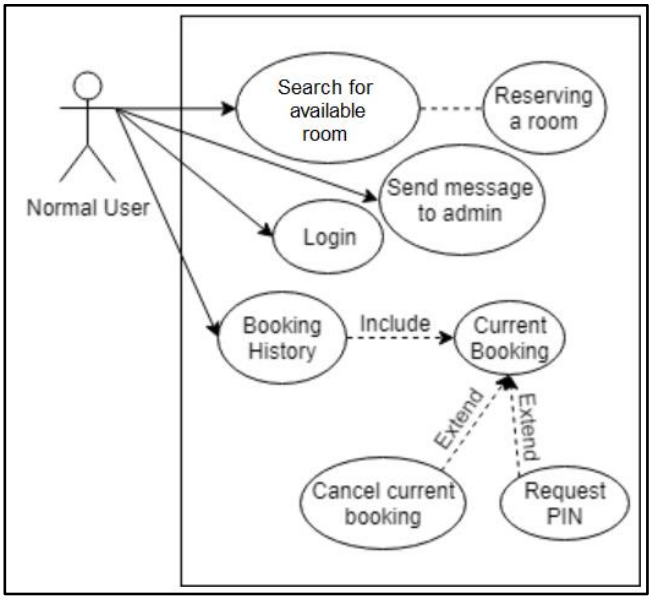

Figure 3.1 Normal User Use Case Diagram

The use case diagram of the normal user is shown in figure 3.1. Each function will be described below.

1. **Search for available room**
   The user is able to search the requirement of the room to filter the room they need.
2. **Reserving a room**
   The user can reserve a room that they need.
3. **Send message to admin**
   The user able to send message to admin
4. **Booking History**
   The user able to look at the room that they reserved at any time.

5. **Current Booking**
   The user able to look at their current booking at any time.
6. **Cancel current Booking**
   The user able to cancel their current booking at any time.
7. **Request PIN**
   The user able to request new PIN via phone or email at any time.
8. **Login**
   The login is needed to use the system.

3.4.2 Admin Use Case Diagram

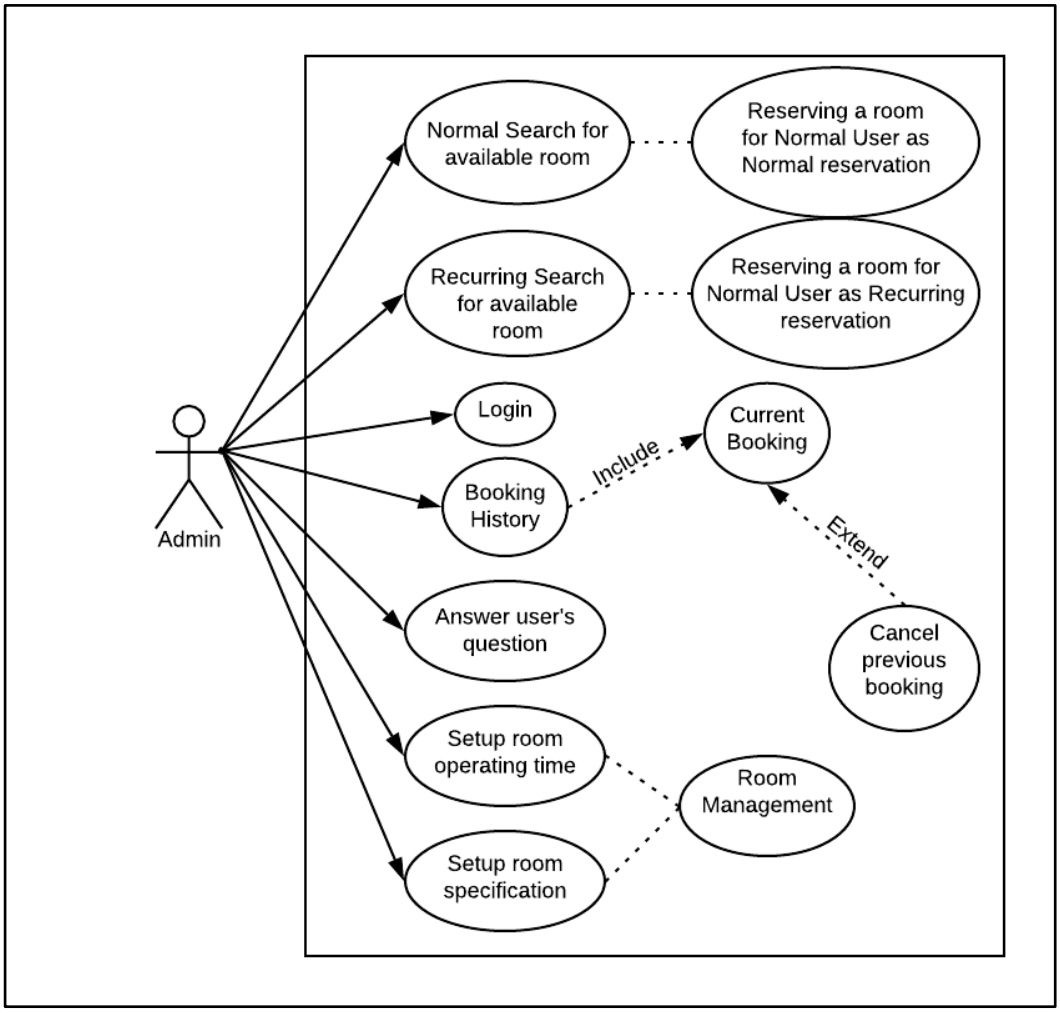

Figure 3.2 Admin Use Case Diagram

The use case diagram of the Admin is shown in figure 3.2. Each function will be described below.

1. **Perform all function as Normal user**
   Admin is able to perform all functions of normal user except "Send message to admin and request PIN"
2. **Normal Search for available room**
   Admin is able to insert the requirement of the room to filter the room they need for normal reservation
3. **Reserving a room for normal user as normal reservation**
   Admin is able to reserve the room for another user as normal reservation.

4. **Recurring Search for available room**
   Admin is able to insert the requirement of the room to filter the room they need for recurring reservation
5. **Reserving a room for normal user as recurring reservation**
   Admin is able to reserve the room for another user as recurring reservation.
6. **Answer user's question**
   Admin is able to answer the question by the user
7. **Setup room operating time**
   Admin can set the operating day and time of each room.
8. **Setup room specification**
   Admin can add the information about the room and equipments that room provided.
9. **Room Management**
   The room can be added, deleted, or changing the information from the system by admin.

3.4.3 Key Box Use Case Diagram

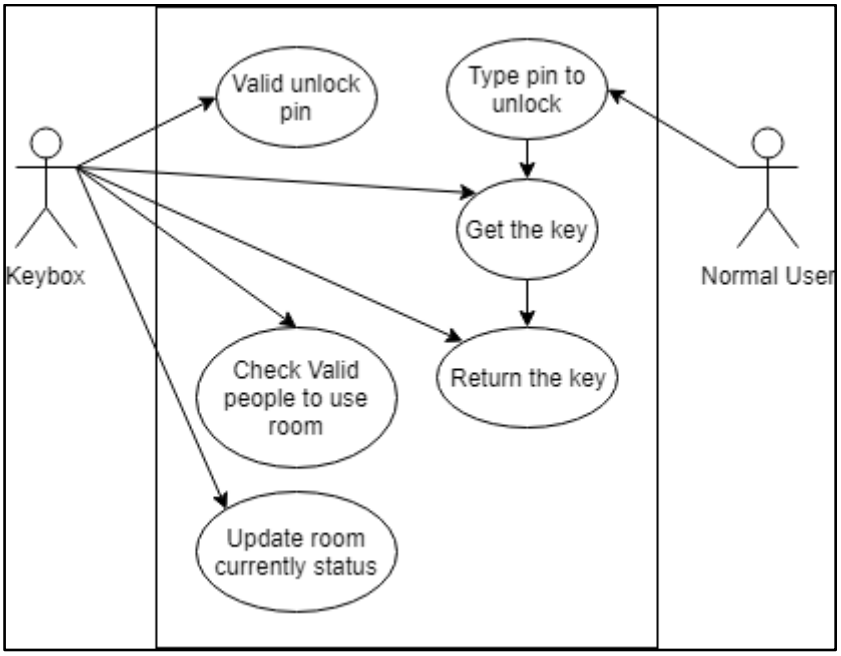

Figure 3.3 Kiosk Key Box Use Case Diagram

The use case diagram of the keybox is shown in figure 3.3. Each function will be described below.

1. **Accept the valid unlock pin**
   Get the pin to compared when the user input pin.
2. **Check valid people to use room**
   Check if the pin is true to open the key box
3. **Update current room status**
   Update the status of the room
4. **Type pin to unlock**
   User Insert pin to unlock the Kiosk Key box
5. **Get the key**
   Get the key from the keybox
6. **Return the key**
   Return the key into the keybox

3.5 Journey map

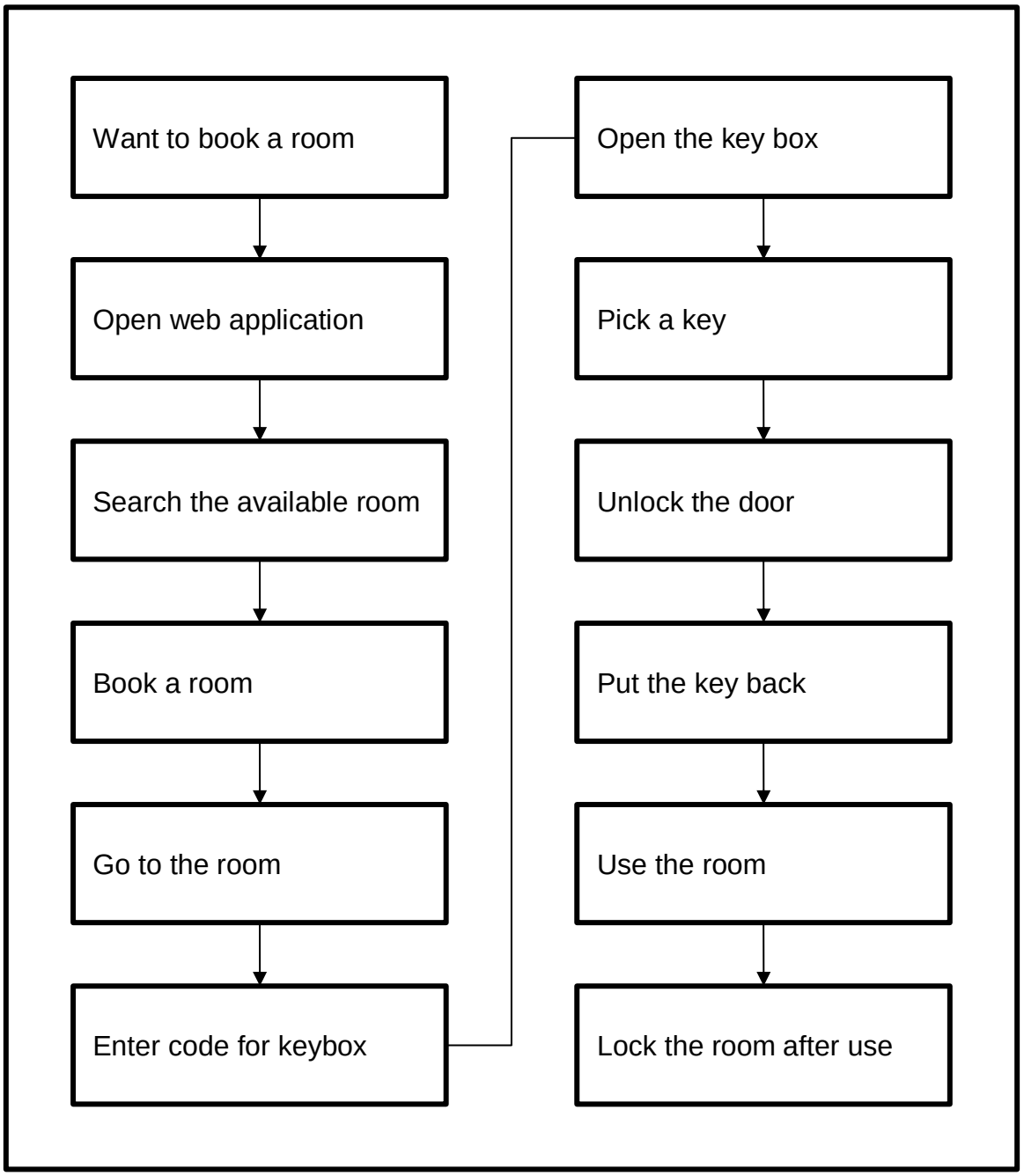

Figure 3.4 User Journey Map

The user journey map shown the process of the booking and using the room in step by step. The process will be started when a user prefers to book a room, he/she need to open web application and search for available rooms. If the user found available room, he/she can pick a room to book. When the time has arrived, the user needs to get to the room and insert the PIN into the keybox in front of the room. Then, get the key from the keybox and open the room. After unlocked the room, user need to return the key back into the keybox. Finally, user can use the room and lock the room after use.

3.6 Database

This section will show how we design the database and how we store the data. The ER Diagram shown the relationship between each entity in the database. Each entity will be described below.

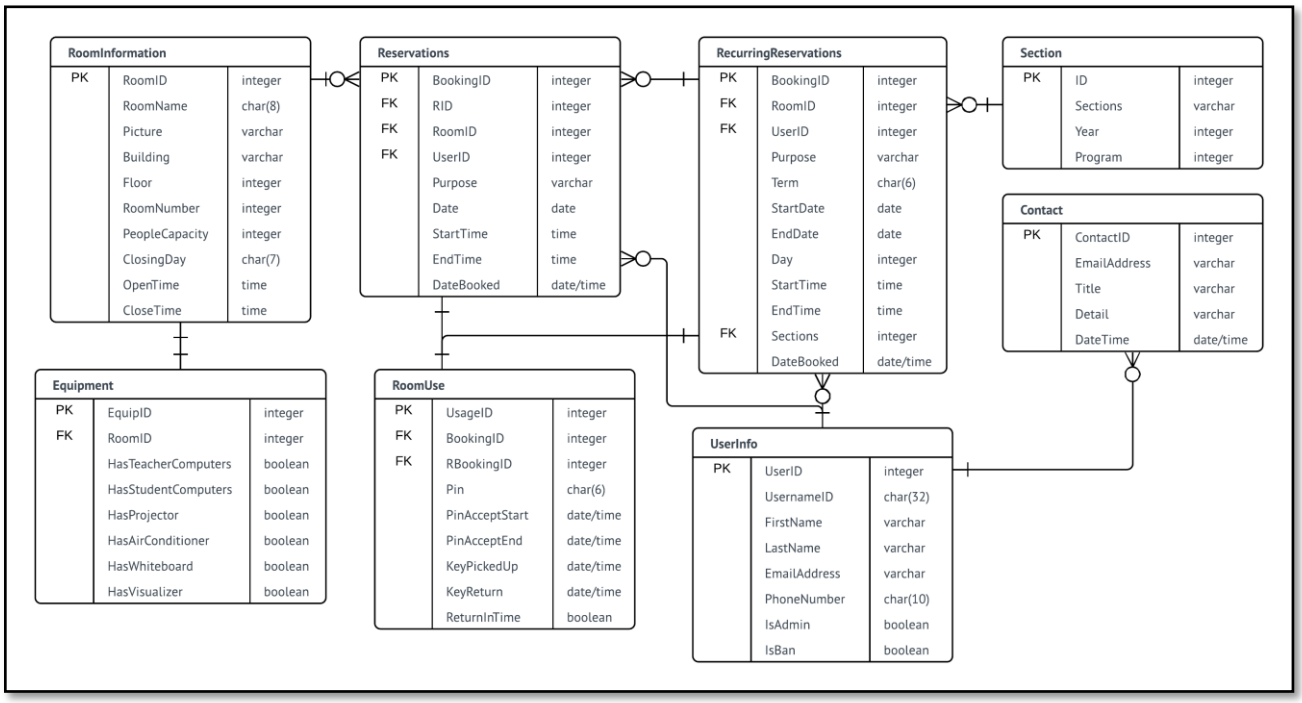

Figure 3.5 Database ER Diagram

1. **RoomInformation**

RoomInformation entity is the table that contains the information for each room. It contains the information that can define the room picture, where the rooms are, how many people the room can handle and the operating time.

2. **Equipment**

This entity contains the data of equipment in each room, such as projector, computers, etc. It can define which equipment is valid or not valid in each room.

3. **User**

The User entity stores the information about each user in the system. The information of the user will be stored if the user has used the system at least once.

4. **RoomUse**

RoomUse is the entity that stores the data for a user when they book the room. When the room is reserved, the system will store the BookingID, Pin and the time when user available to insert the pin to get and return the key. And the time when user pick up the key and return the key will be store when the user act with it. When the time arrived at KeyPickedUp time the keybox that related to the RoomID will be activate and wait to accept PIN.

5. **Reservations**

This entity stores all information of a reservation. It can define which room that the user reserved, purpose of reservation, date and time of reservation.

6. **RecurringReservations**

This entity stores all information of a reservation with recurring option. It is similar to

Reservations entity, but it also contains the Semester, start reservation date, end reservation date, the purpose or the class name. Also, the sections and year of students' study in that class.

7. **Section**

The Section entity contain the information about the section and year of the students and which program they are studying. This is use for bookings that the admin creates for classroom use by regular courses.

3.7 Use Case Narrative

Table 3.2 Reserving a Room Use Case

Use case	**Reserving a room**
Iteration:	
Primary actor:	Normal User
Goal in context:	The system will allow the actor to book the room
Precondition:	User have to login to the system.
Trigger:	When the actor wants to book a room.
Scenario:	User pass login page and get the log in access from system User see the welcome page with search menu User search the room that they want by select time, room size, equipment that they want System will look for free room that match the requirement User see the search result User select the room that want to book System will create booking User see the successful message
Exceptions:	1) No rooms available that fulfill the user's requirements Show the message "There are no matching room" 2) User decided to not make a booking and click cancel. Redirect to main page.
Priority:	1
When available:	
Frequency of use:	Usually
Channel to actor:	Web application
Secondary actor:	Admin
Channel to secondary actors:	
Open issues:	

Table 3.3 Cancel Booking Use Case

Use case:	**Cancel booking**
Iteration:	
Primary actor:	Normal User
Goal in context:	User able to cancel their booking
Precondition:	User have to login to the system. User has previously booked a room
Trigger:	User want to cancel their booking
Scenario:	From welcome page, select history Click on current booking tab System will query all current booking that user have User see list of booking User select booking that you want to cancel System process request and display result to screen.

Exceptions:	1) User have cancelled the room after start time. Show error message.
Priority:	1
When available:	
Frequency of use:	Sometimes
Channel to actor:	Web application
Secondary actor:	Admin
Channel to secondary actors:	
Open issues:	

Table 3.4 View Booking History Use Case

Use case	**View booking history**
Iteration:	
Primary actor:	Normal User
Goal in context:	User views their booking history
Precondition:	User has to login to the system. User has reservation in the past
Trigger:	User want to view their booking history
Scenario:	From welcome page, select history System will query all booking that user have System will show only booking which passed View booking history
Exceptions:	1) User never book the room Show empty page
Priority:	2
When available:	
Frequency of use:	Sometime
Channel to actor:	Web application
Secondary actor:	
Channel to secondary actors:	
Open issues:	

Table 3.5 Assign Booking to User Use Case

Use case	**Assign booking to user**
Iteration:	
Primary actor:	Admin
Goal in context:	User is able to make a reservation to the other users
Precondition:	User must be admin role User has to login
Trigger:	Some users ask admin user to make a reservation
Scenario:	User passes login page and gets the login access from system User approaches the welcome page with search menu User search the room that they want by select time, room size, equipment that they want System will look for free room that match the requirement User see the search result

	User selects the room that wants to book User inserts the username, email, and phone number for people who will own the booking System will create booking User see the successful message System sends the PIN to user email and phone number
Exceptions:	1) No rooms available that fulfill the user's requirements Show the message "There are no matching room"
Priority:	3
When available:	
Frequency of use:	Sometime
Channel to actor:	Web application
Secondary actor:	
Channel to secondary actors:	
Open issues:	

Table 3.6 Recurring Reservation Use Case

Use case	**Recurring Reservation for a room**
Iteration:	
Primary actor:	Admin
Goal in context:	User is able to make a recurring reservation to the other users
Precondition:	User must be admin role User has to login
Trigger:	Some users ask admin user to make a reservation
Scenario:	User passes login page and get the log in access from system User see the welcome page with search menu User search the room that they want System will look for the room that match the requirement User see the search result User selects the room that want to book User fills in the period of booking and time slot User inserts the username, email, and phone number for people who will own the booking System will create booking User see the success message System sends the PIN to user email and phone number
Exceptions:	1) No rooms available that fulfill the user's requirements Show the message "There are no matching room"
Priority:	3
When available:	
Frequency of use:	Sometime
Channel to actor:	Web application
Secondary actor:	
Channel to secondary actors:	
Open issues:	

Table 3.7 Add Room Use Case

Use case	**Room management (Add)**
Iteration:	
Primary actor:	Admin
Goal in context:	User able to create new room
Precondition:	User must be admin role User has to login
Trigger:	There is new room to add in the system
Scenario:	User logins into the system Select manage room System will query all room in the system and show on screen Click add new room Fill up an information System will create new room and write it in database
Exceptions:	1) Information filled in wrong type or don't met the conditions Show error message and let the user fill the information again. 2) RoomID already exist – NOT POSSIBLE, though room number might already be in the system. Show error message and ask for new RoomID.
Priority:	3
When available:	
Frequency of use:	sometime
Channel to actor:	Web application
Secondary actor:	
Channel to secondary actors:	
Open issues:	

Table 3.8 Edit/Delete Room Use Case

Use case	**Room management (Edit/Delete)**
Iteration:	
Primary actor:	Admin
Goal in context:	User able to create new room
Precondition:	User must be admin role User has to login
Trigger:	The room is no longer available to use
Scenario:	Login into the system Select manage room System will query all room in the system and show on screen Click edit button on room that you want to edit System query room information Fill up an information System will edit room information and write it in database or delete room on database
Exceptions:	1) [EDIT]Information filled in wrong type or don't met the conditions Show error message and let the user fill the information again. 2) User delete the room they have reserved by others

	Show warning message. Ask to confirm the action. If confirmed all the booking in that room will be cancel for all user
Priority:	3
When available:	
Frequency of use:	sometime
Channel to actor:	Web application
Secondary actor:	
Channel to secondary actors:	
Open issues:	

Table 3.9 Get the PIN Use Case

Use case	**Get the Pin**
Iteration:	
Primary actor:	Normal User
Goal in context:	User able to see PIN anytime
Precondition:	User have to login
Trigger:	User forgot the PIN
Scenario:	Login into the system Go to history page Go to current reservation System will query all reserved room for the user and show on the screen Click show PIN on the room that user need PIN shown as pop-up
Exceptions:	1) Time has passed the Key Pickup Time. Show error message “Key cannot be pick up anymore”.
Priority:	3
When available:	
Frequency of use:	sometime
Channel to actor:	Web application
Secondary actor:	
Channel to secondary actors:	
Open issues:	

Table 3.10 Validate PIN to Unlock Keybox Use Case

Use case	**Validate pin to unlock keybox**
Iteration:	
Primary actor:	Normal user
Goal in context:	User able to make a reservation to the other user
Precondition:	User have to login
Trigger:	Some users ask admin user to make a reservation
Scenario:	User type pin on key box panel Keybox will send the pin to check in database Keybox will open box door to be able to get the key and timer start User use the key to open the door and replace back into the box User close the box Keybox validate the key inside.

	Keybox show success message
Exceptions:	1) User insert the wrong pin. Show error message and ask for a new pin 2) User doesn't return the key in time Key box plays an alarm sound and send message to user. If the key doesn't return in a while the key box will send message to admin.
Priority:	3
When available:	
Frequency of use:	sometime
Channel to actor:	Keybox
Secondary actor:	
Channel to secondary actors:	
Open issues:	

3.8 System Architecture Design

There are six type of components in the system, which are Frontend Server, Application Server, Database/File Server, Kiosk Key Box, LDAP Server and Email service

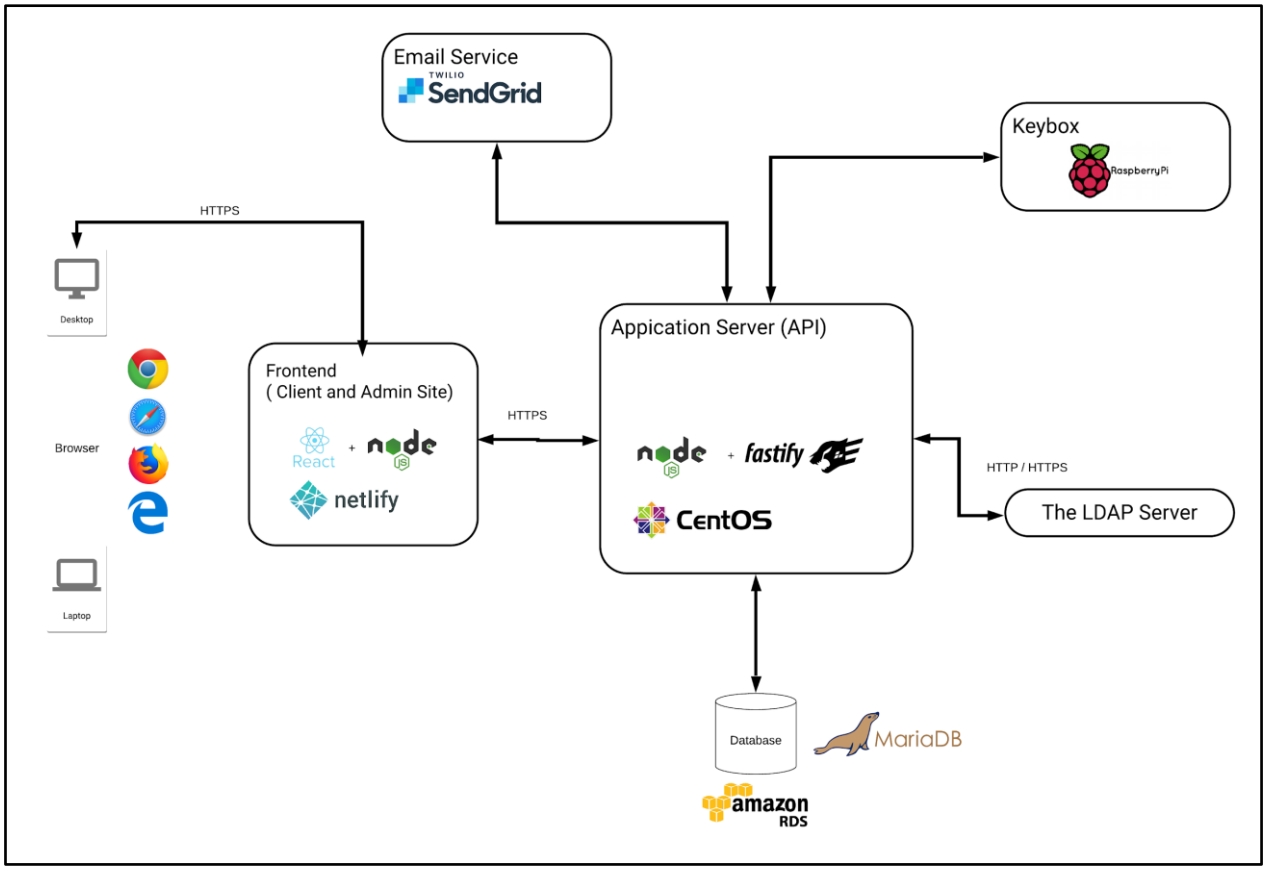

Figure 3.6 System Architecture design diagram

3.8.1 Frontend Server (Admin Side, Client Side)

The front-end code is separated into two parts by checking user permission. Only admin user can be able to access admin functions. The frontend service is served via Netlify. In our project, we choosing to use React as a frontend library.

3.8.2 Application Server (API)

The main purpose of API is to manage data in the system and provide some business logic, also connect all component in the system together. The software on application server is Fastify (NodeJs).

3.8.3 Database / File Server

Using Amazon Relational Database Service (RSD) to store information of the system that connect to application server. We choose MariaDB as the database engine.

3.8.4 Keybox

The Keybox is using to store the key. The Keybox will get the PIN insert by user and send the request to validate with API by using HTTP/HTTPS. The board on Keybox is Raspberry Pi Zero W. Additionally, any number of key boxes can be added to the system for each room.

3.8.5 LDAP

The LDAP is using for authentication. The system will send the request to validate user is along to KMUTT member.

3.8.6 Email Service

The email service is using for sending the code to unlock the room and reply user problem. In our system, we choose SendGrid to be the email server since it provides good document, easy to set up, and has free tier.

Chapter 4
Results and Discussion

4.1 Introduction

The user interface and Kiosk key box has been designed to be user friendly. The user interface is developed using ReactJS and also developed to be responsive on any device. Also, the kiosk key box is designed to be a small size so it can be installed at every room. The key box is designed with high security but not complexity of use.

The web application has been implemented. The system language is in bilingual which are Thai and English. But there is some feature that we have not finished yet. The feature to ban the user and generate reports for each user is not implemented yet.

The Kiosk key box has been printed by 3D printer. The components inside is connected together inside the Kiosk Key Box. Therefore, database has successfully created and deploy to store the data.

The API is implemented and connected to overall system. For API, we chose to develop it based on NodeJS by using FastfyJS plug with knexjs and bookshelf as an ORM connecting with Mariadb. Moreover, API is connected with web application, Kiosk Key Box, database and able to work properly.

4.2 User Interface

Every pages of user interface are designed and implemented. Actual screenshot of user interface for Normal User and Admin was shown as pictures below.

4.2.1 User Interface for Normal User

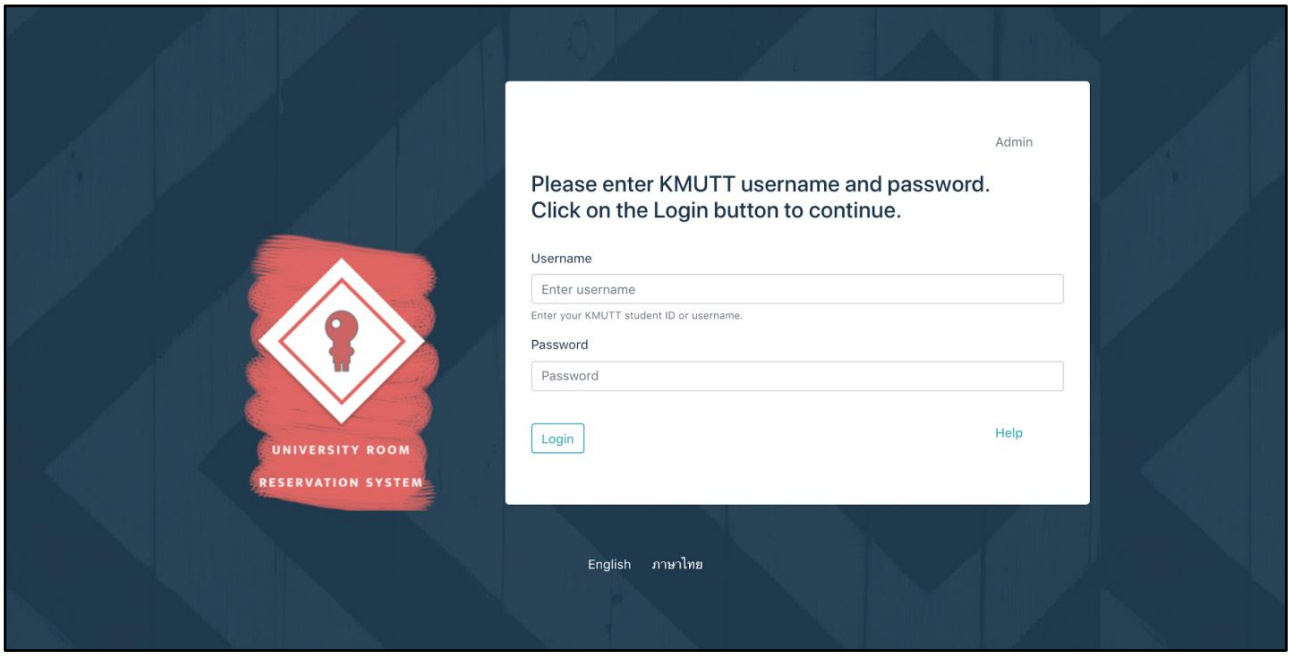

Figure 4.1 Login page for normal user

The page shown in figure 4.1 is the login page. Users need to login by filled KMUTT intranet login credentials into the form. After the form has been filled, the user clicks on “Login” button to get into the system.

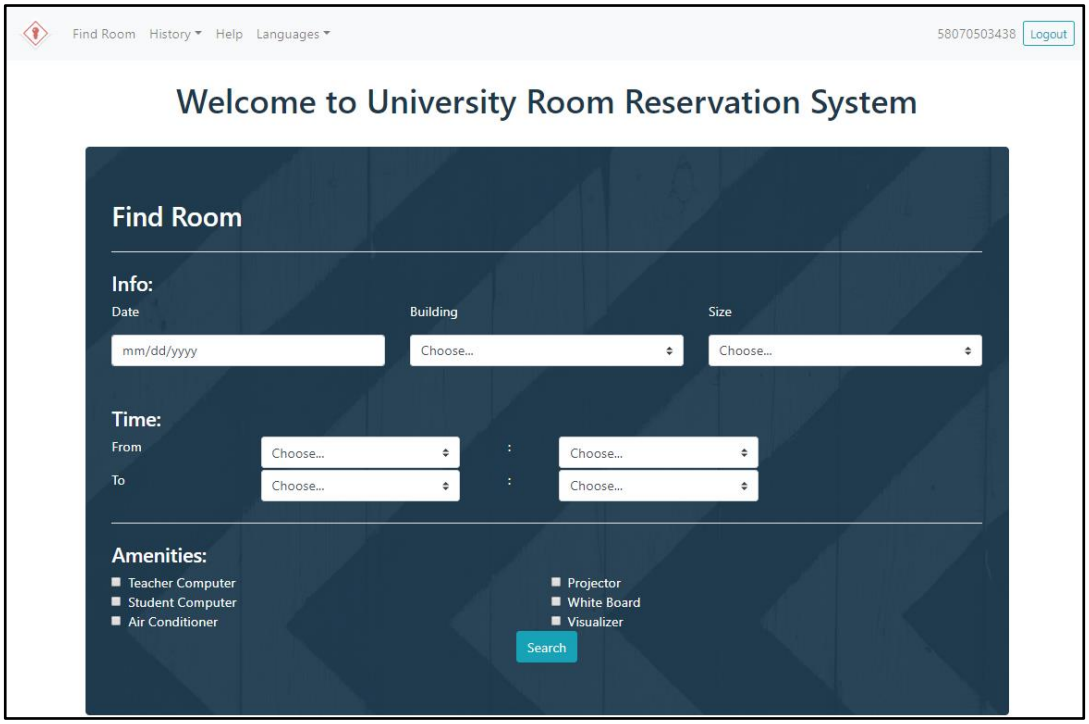

Figure 4.2 Searching page for normal user

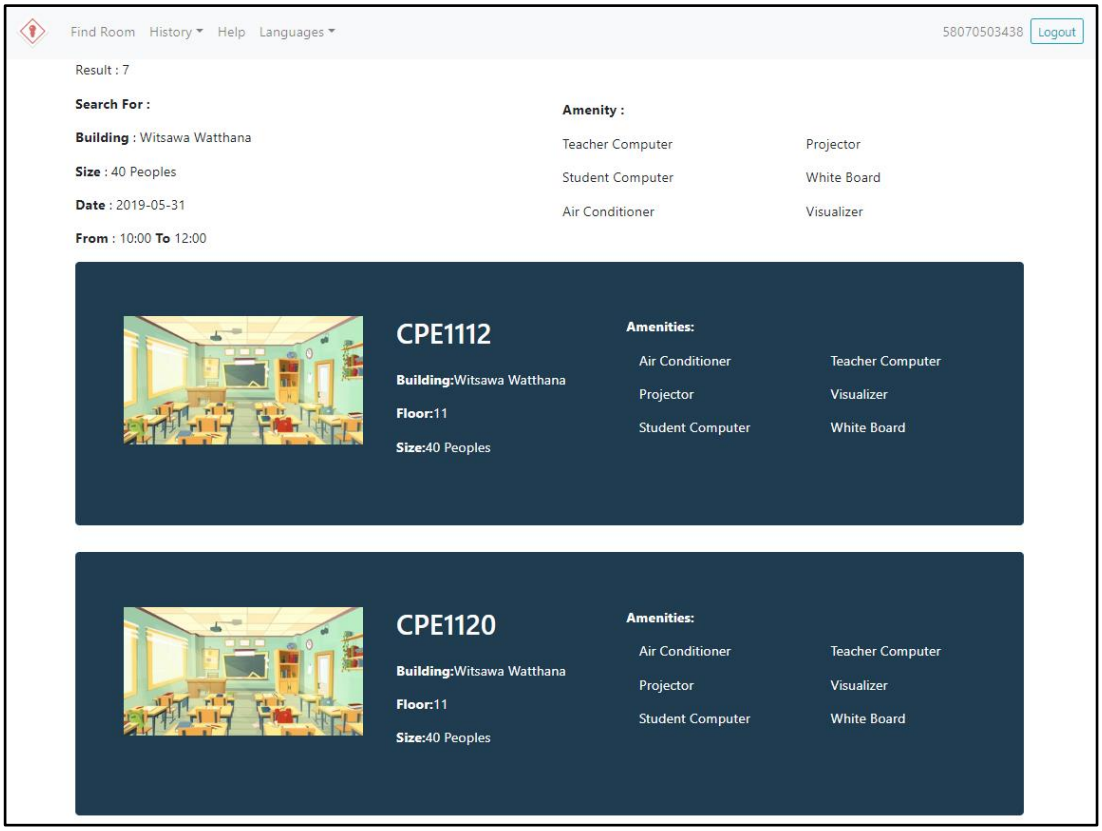

Figure 4.3 Room listing page for normal user

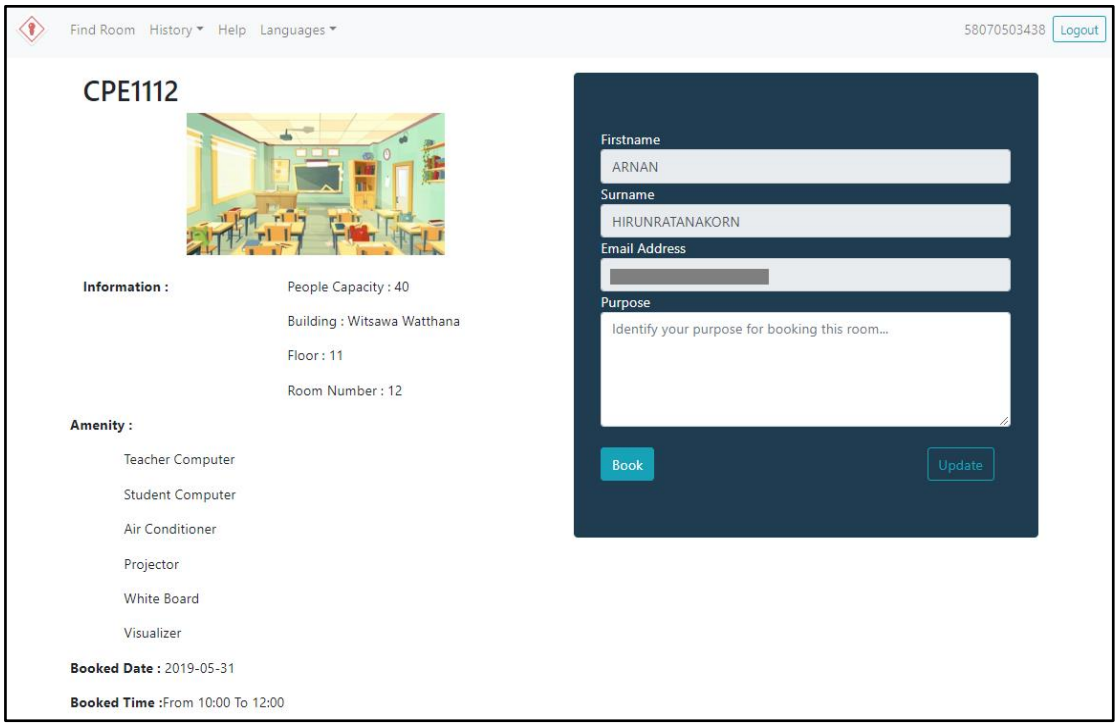

Figure 4.4 Booking page for normal user

The page shown in figure 4.2 allows the user to find their desired room. There are two ways to reach figure 4.2. First, the system will automatically display this page after logging in. Second, users can click on 'Find Room' on the menu bar to enter this page. To find a room, the user must specify the required information which is date and time that they want to reserve, building where the room is, size of the room. They can also filter the room. When user finishes filling in info, the user clicks on 'Search' button to continue searching.

After clicking the 'Search' button, users will reach the page shown in figure 4.3, which shows the list of all rooms that match their searching. Click on the blue tab to see more information of each room.

The page shown in figure 4.4 allow the user to book the room. When they click on the blue tab in the room list. The room's information will be provided on the left side. On the right side, users will see another form to fill but this is optional. They do not need to fill in the form because there is information in database since they first login to the system. They only need to fill in the purpose of using this room

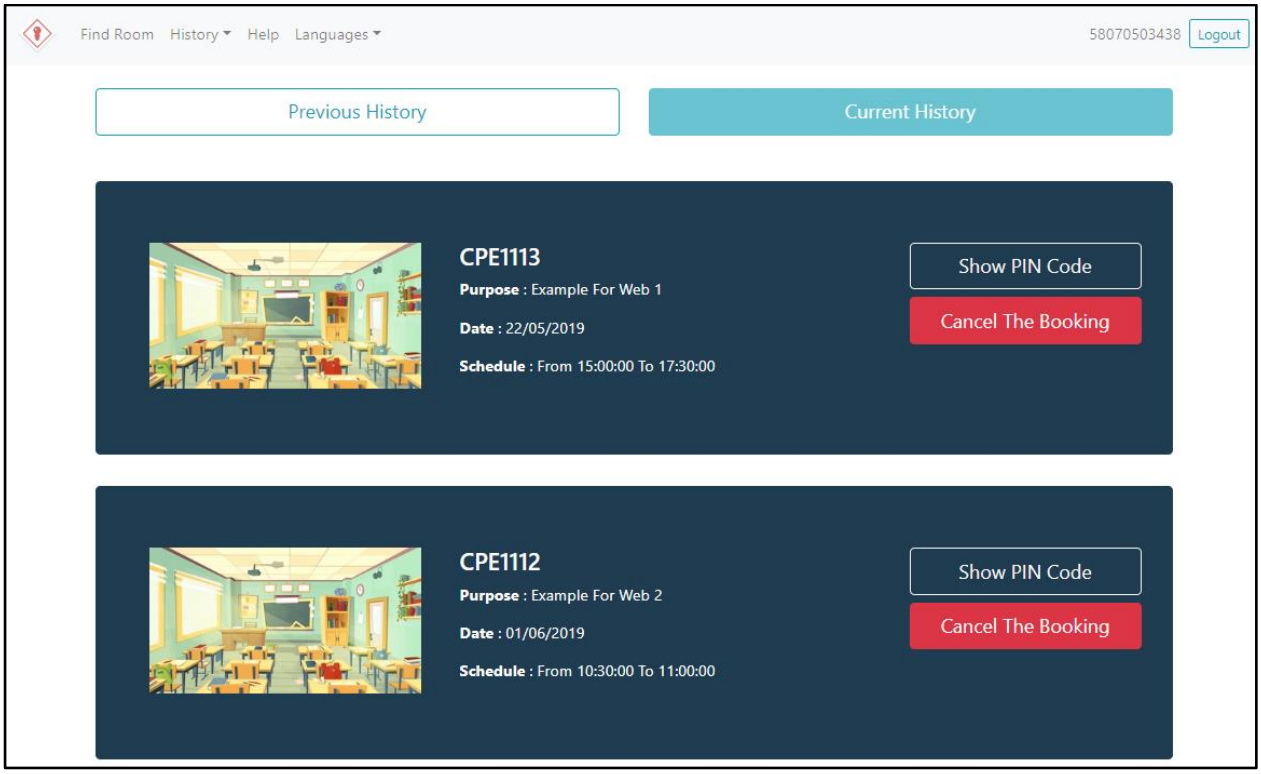

Figure 4.5 *Current booking page for normal user*

The page shown in figure 4.5 allow the user to see the booking history, request PIN and cancel the booking. Users can get PIN in order to use with the kiosk keybox to get the key. Users can visit this page at any time by clicking on 'History' on menu bar and then click on 'Current Booking'.

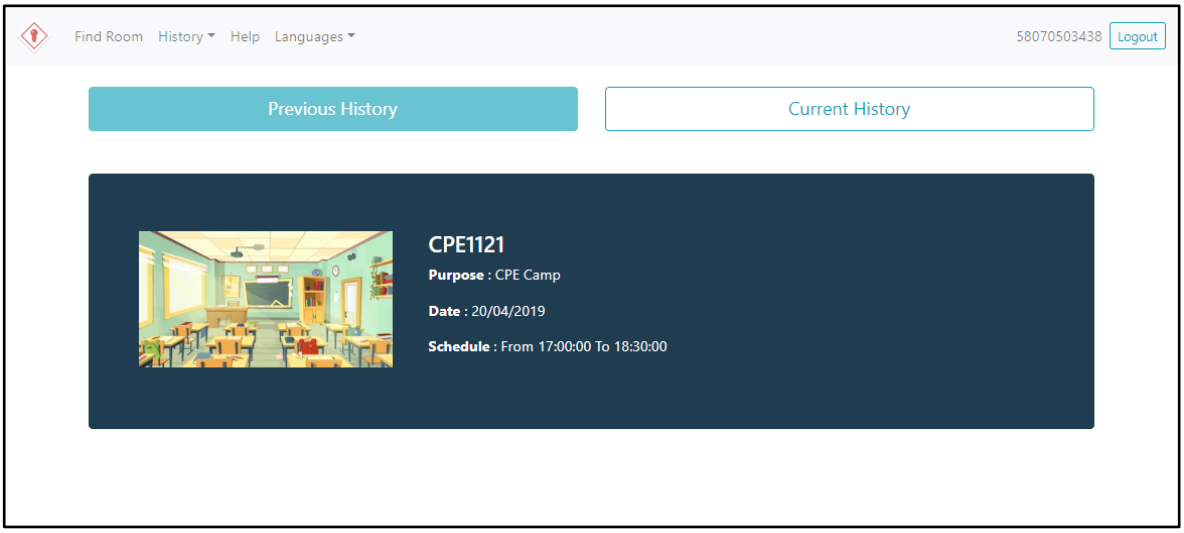

Figure 4.6 Previous booking page for normal user

The page shown in figure 4.6 allow the user to see the booking history that has passed. User can get into this page by clicks on 'History' button or click on 'Previous Booking' button if they are on the current booking page. This page will provide their booking history. Each history will show the status of the booking.

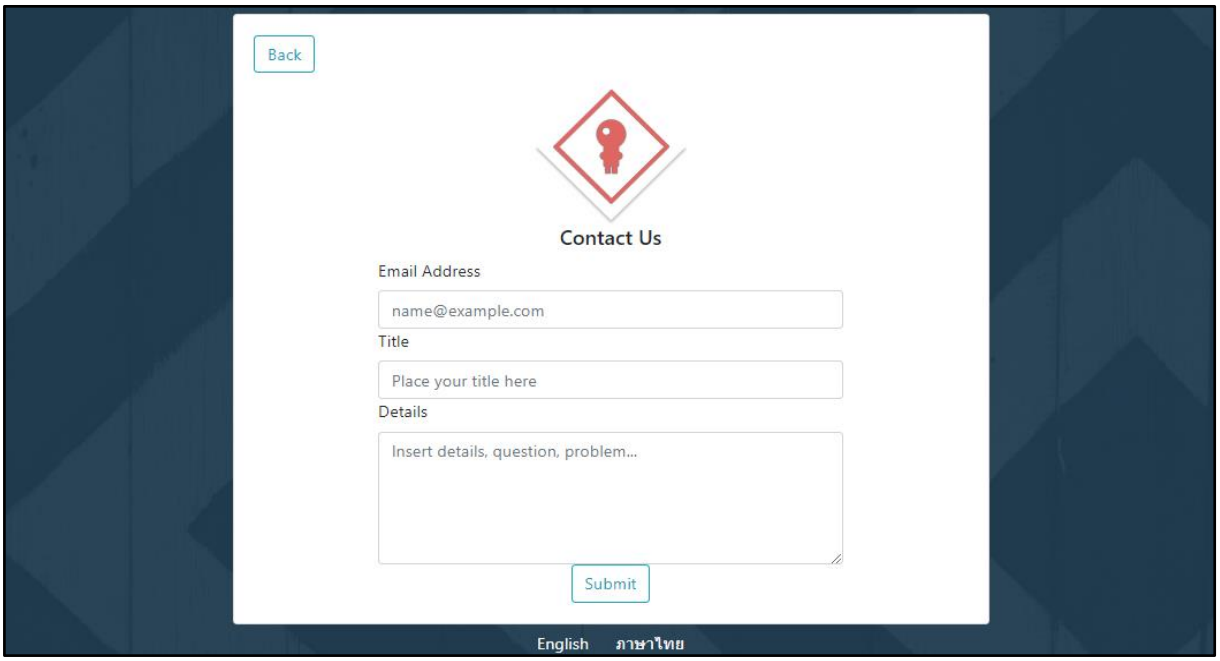

Figure 4.7 Contact page for normal user

The page shown in figure 4.7 allow the user to contact admin. User can clicks on “Help” button from wherever they are at that moment (any page) including the Login page (figure 4.1). Users will directly contact the administrator on this page. They just need to type their email, title of their story along with the detail.

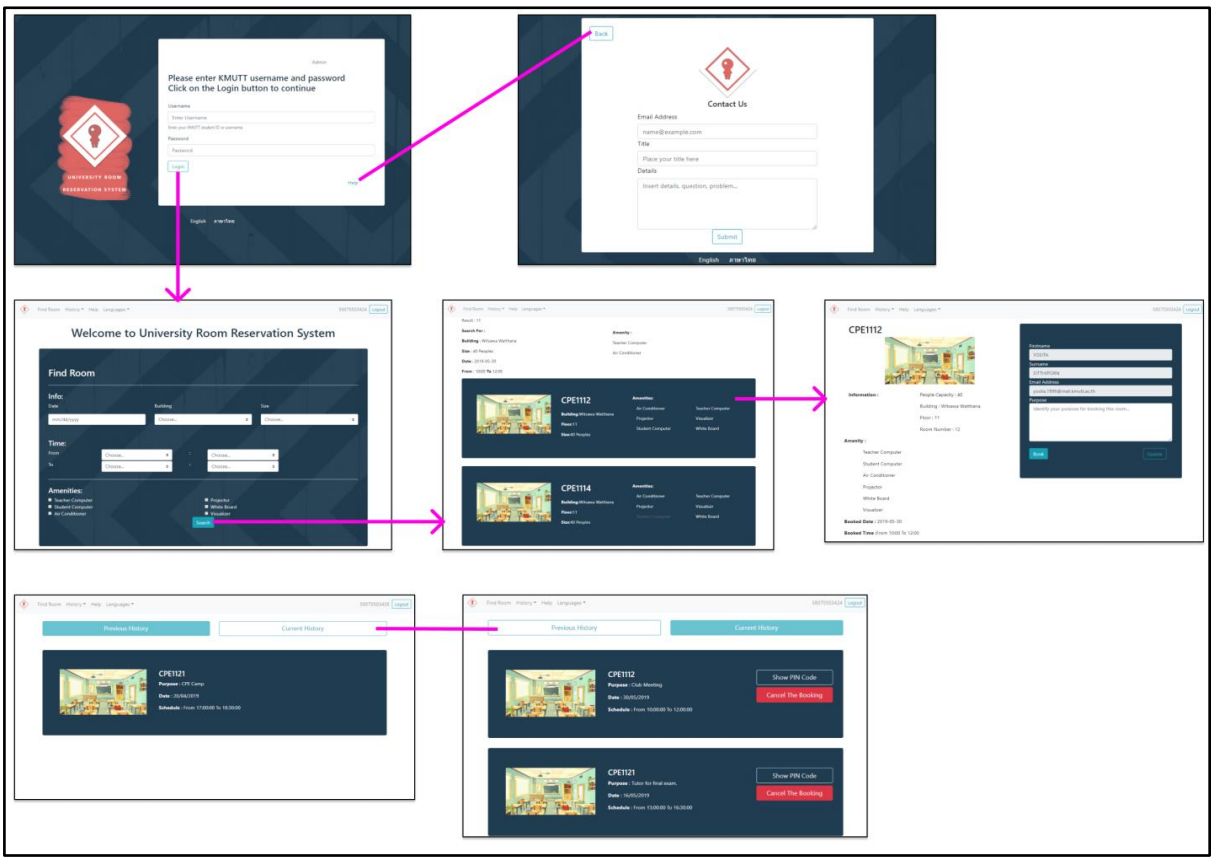

Figure 4.8 The use flow of the user interface for normal user

4.2.2 User Interface for Admin

The user interfaces for Admin is a little more complex than the normal user due to additional features. The user interface of Admin is shown below.

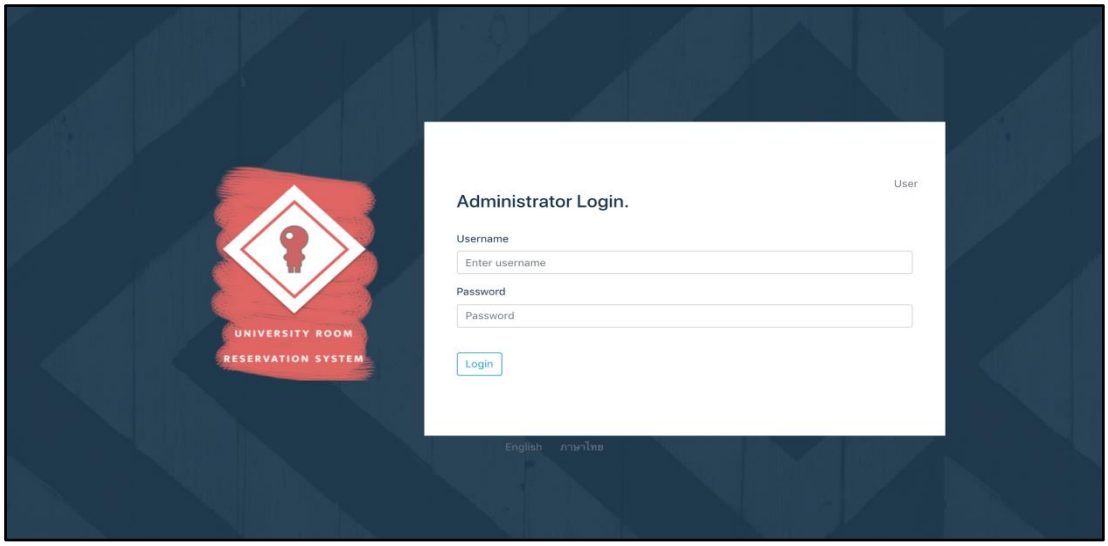

Figure 4.9 Login page for admin

The page shown in figure 4.9 allow the user to login to the system as admin. Admin needs to login by typing their unique ID and password into the system and click on the Login button to continue. The usernames and password of all the admins will be stored in the database as part of the system configuration.

Figure 4.10 Searching page for admin

The page shown in figure 4.10 allow the user to see all room list. This page will show immediately after logging in. Admin is allowed to search any room in the system from the ‘Search’ bar above by type the name of the room in the ‘Search’ bar.

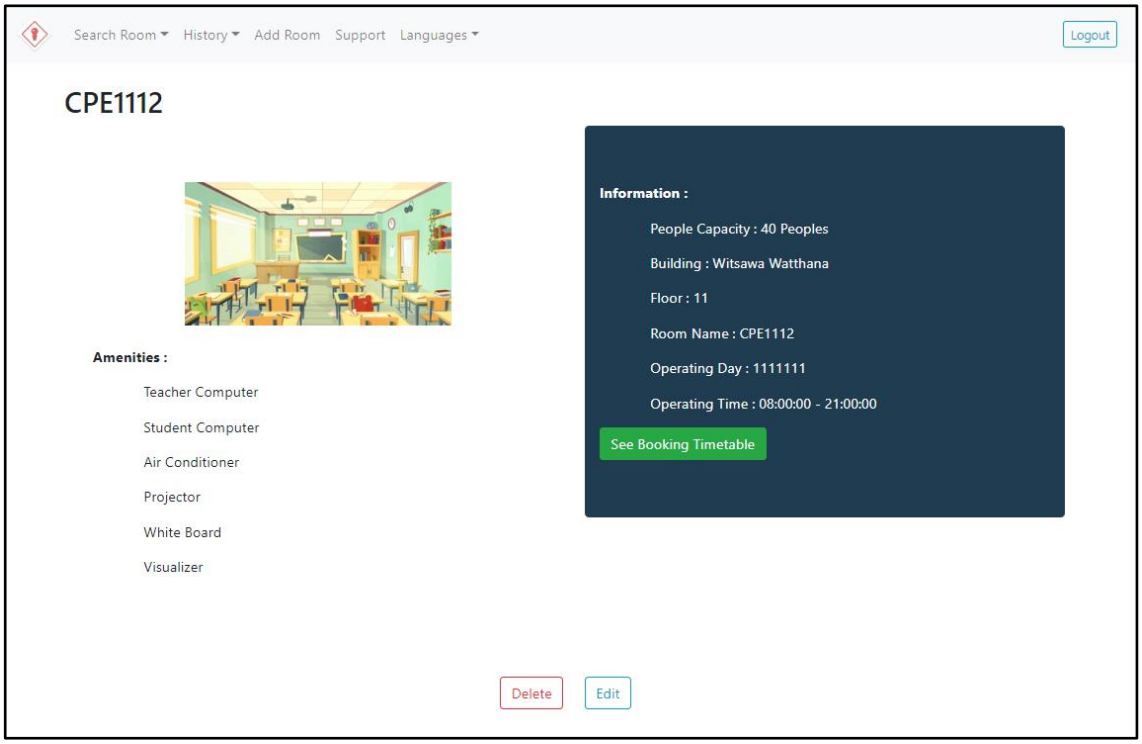

Figure 4.11 Room information page for admin

The page shown in figure 4.11 allow the user see more information about each room. Admin can delete, edit and view room information on this page.

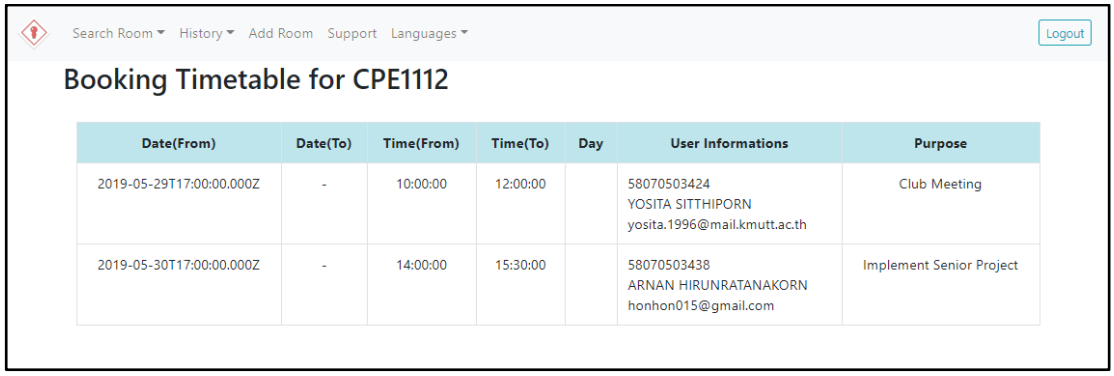

Figure 4.12 Booking table page for admin

The page shown in figure 4.12 allow the user to see booking timetable of each room. Admin will enter figure 4.12 when they click on the ‘See booking timetable’ link to see more information about each booking. On this page, it will show all the reservations of this room. For the reserved room, it also shows that who has booked this room and what was the specified purpose for using this room.

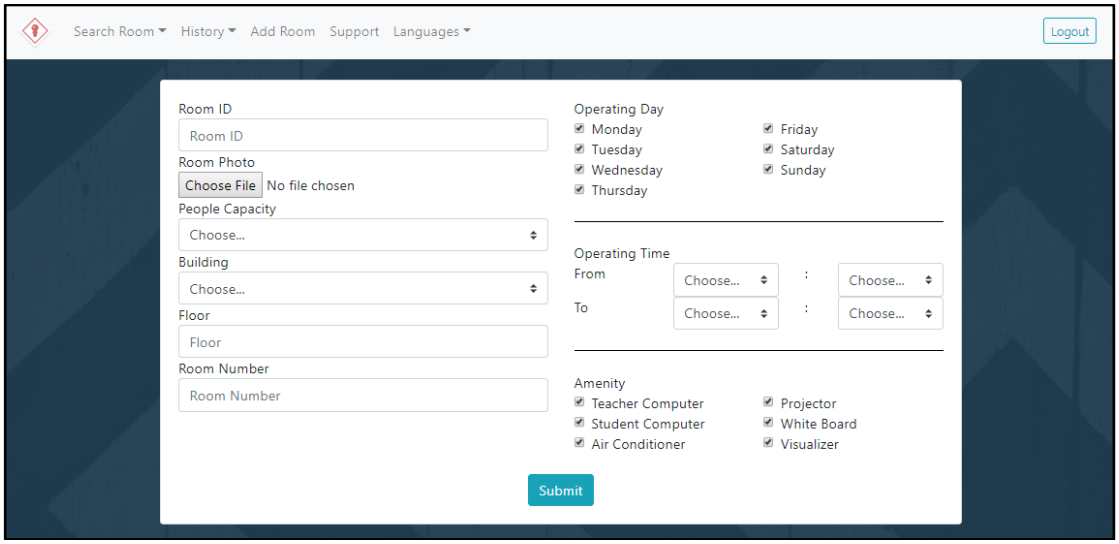

Figure 4.13 Add and edit room page for admin

The page shown in figure 4.13 allow the user to add more room or edit the existing room. If admin wants to add new room, he/she has to click on 'Add Room' button on the navbar. But if he/she wants to edit room, he/she has to click on 'edit' button on the room information page because when he/she click edit, the old information is still shown in the blank but can be modified.

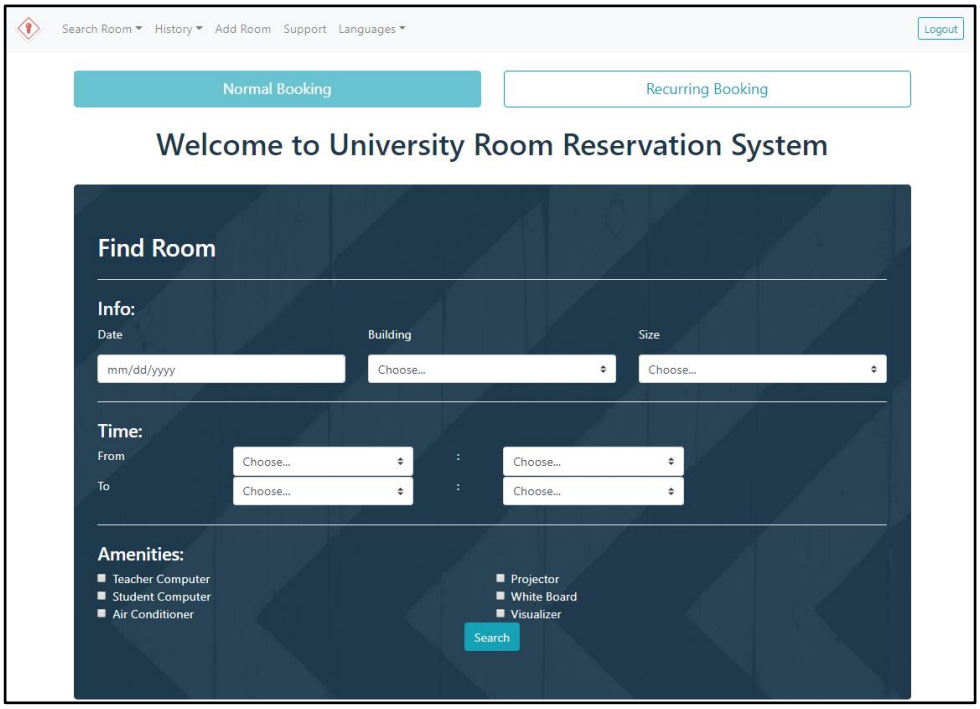

Figure 4.14 Searching page for normal booking

There are two types of booking for admin, normal (one date) and recurring (multiple dates, usually the same day and time each week). The page shown in figure 4.14 allow the user to search for normal room booking. This page allows admin to find their desired room. They just have to specify the required information which is date and time that they want to reserve, building where the room is, size of the room. They can also filter the room. When user finishes filling in info, User can click on 'Search' button to continue searching.

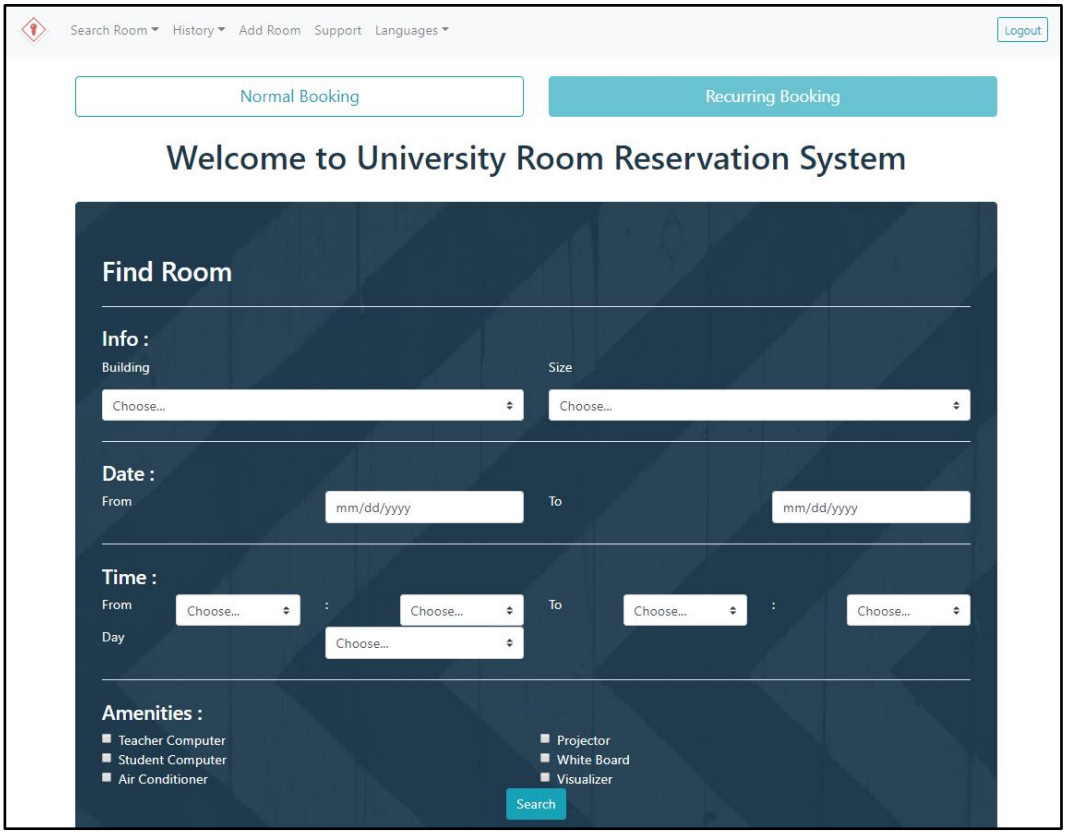

Figure 4.15 Searching page for recurring booking

The page shown in figure 4.15 allow the user to search for recurring room booking, which is normally used for a regular class, lecture or lab. This page allows admin to find their desired room. They just have to specify the required information which is date and time that they want to reserve, building where the room is, size of the room.

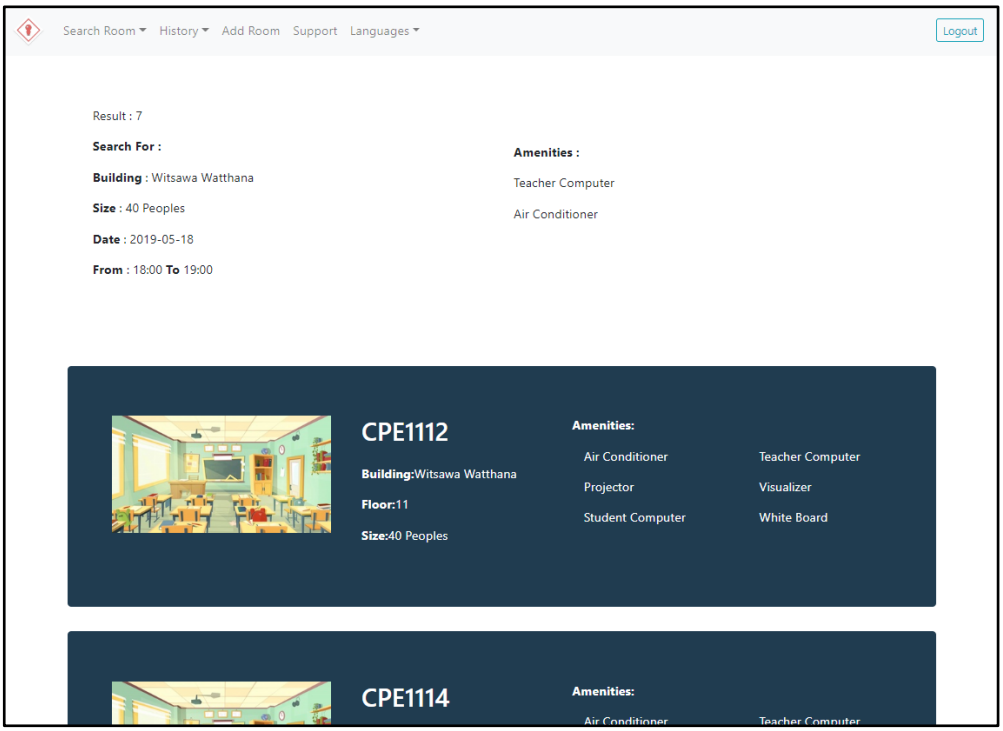

Figure 4.16 Room listing page for admin

The page shown in figure 4.16 allow the user to see the room available for booking. After clicking the ‘Search’ button, admin will reach figure 4.16, which shows the list of all rooms that match their searching. Click on the blue tab to see more information of each room.

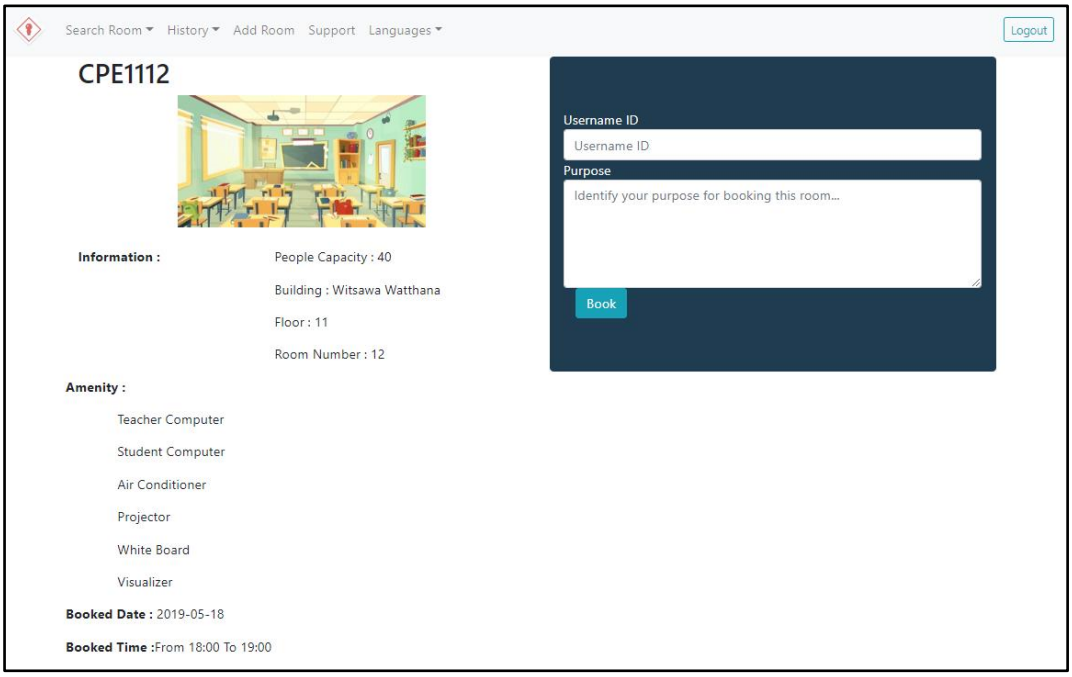

Figure 4.17 Normal Booking page for admin

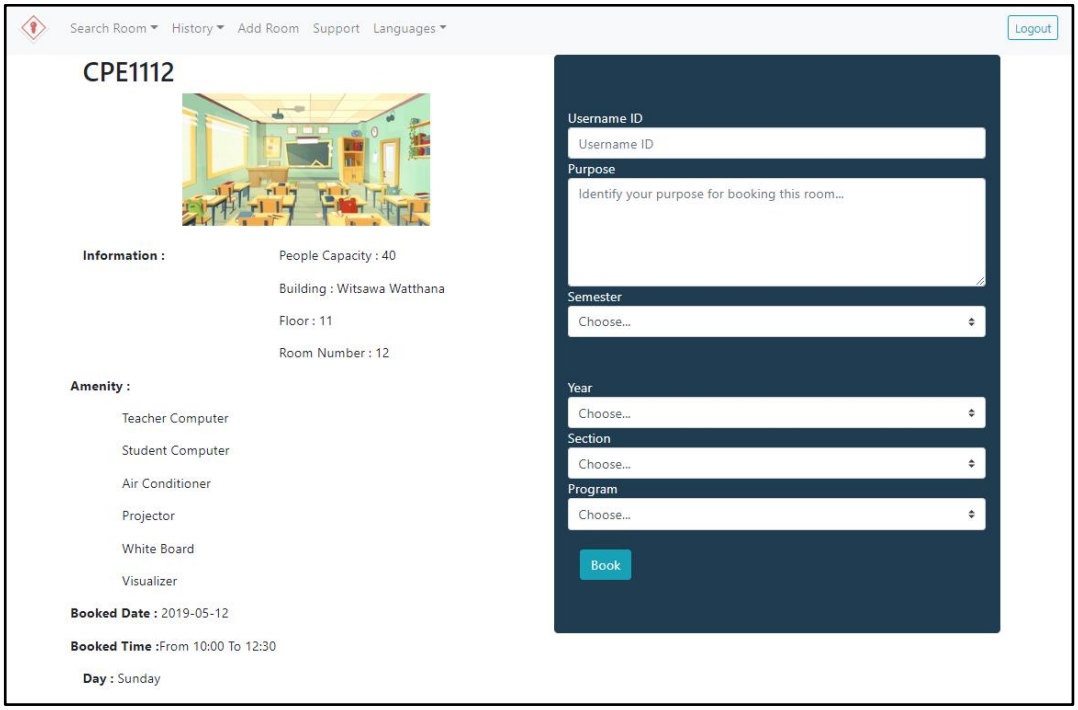

Figure 4.18 Recurring Booking page for admin

The page shown in figure 4.17 allow the user to book for normal booking while figure 4.18 allow the user to book for recurring booking, when they click on the blue tab in the room list. The room’s information will be provided on the left side. On the right side, admin will see another form to fill but this is optional. He/she needs to fill in the form only if he/she wants to book this room. Thus, if he/she wants to book this room, he/she needs to fill all these blanks and clicks ‘Book’ button.

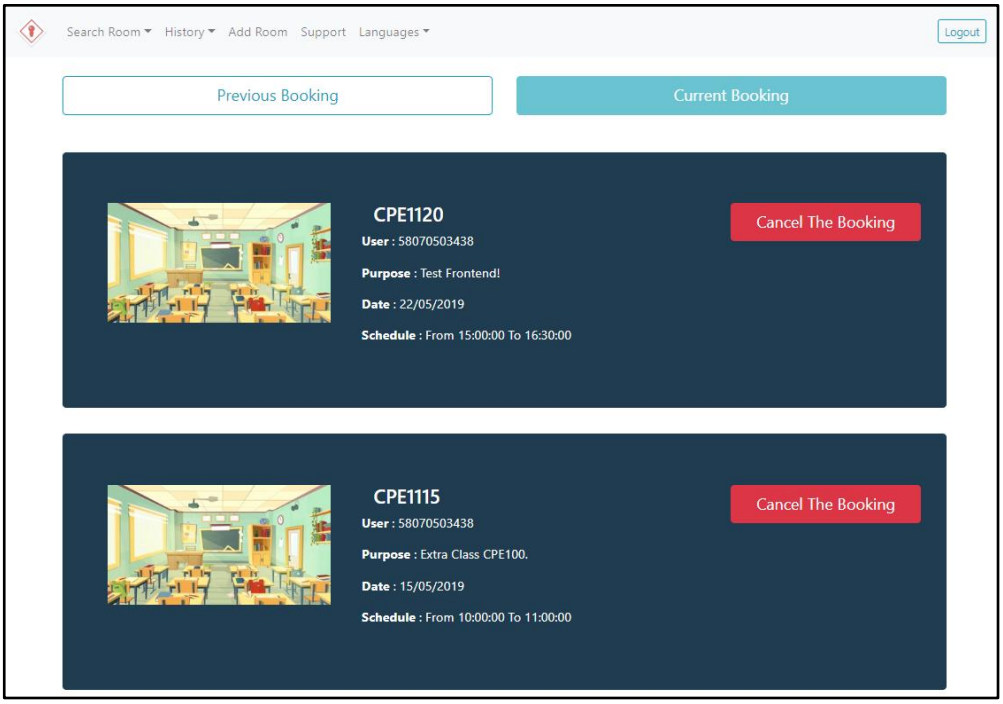

Figure 4.19 Current booking page for admin

The page shown in figure 4.19 allow the user to see booking history for all user and admin is allowed to cancel it. Admin can enter figure 4.19 by click on ‘History’ button. This page will provide all booking history and the button to cancel each booking.

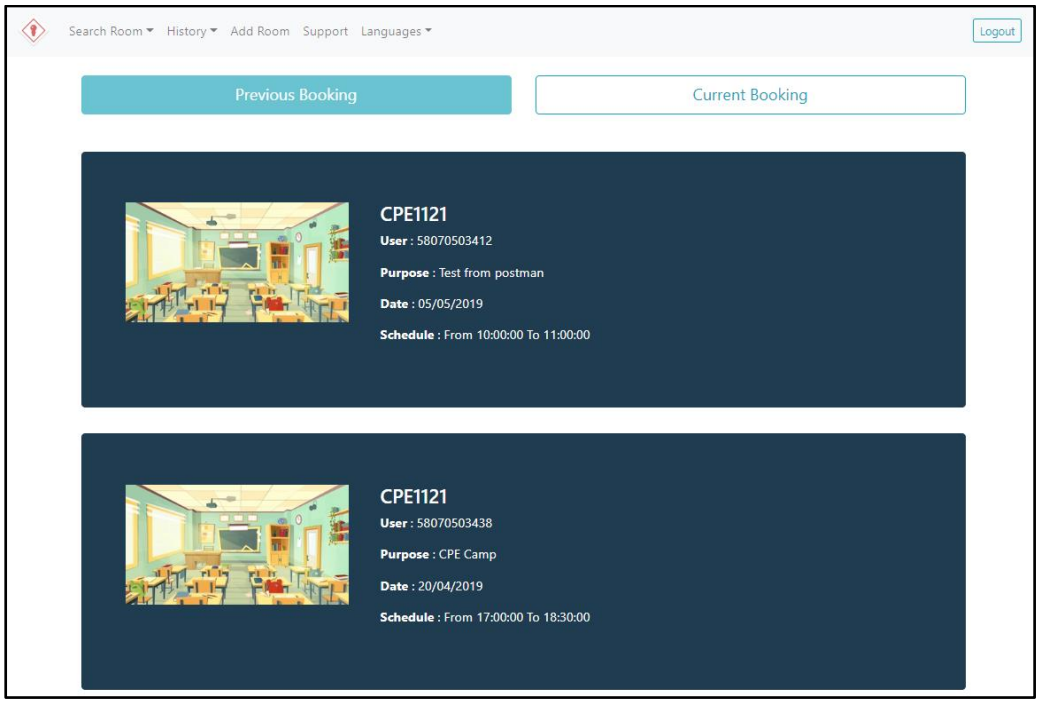

Figure 4.20 Previous booking page for admin

The page shown in figure 4.20 allow the user to see booking history for all user that has passed. Admin can enter figure 4.20 by click on ‘History’ button or click. This page will provide all booking history.

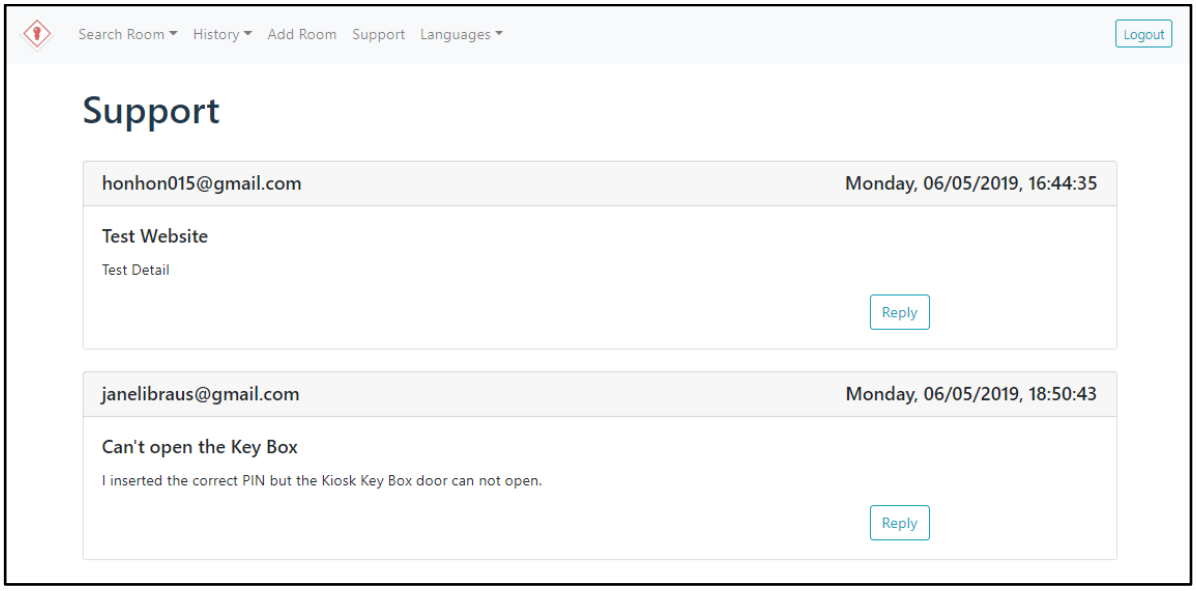

Figure 4.21 Support page for admin

The page shown in figure 4.21 allow the user to see the message from normal user. Admin will enter figure 4.21 by click on 'Support' on the navbar. This page will show all the message from each user. Admin can also reply to users by click on 'Reply' button.

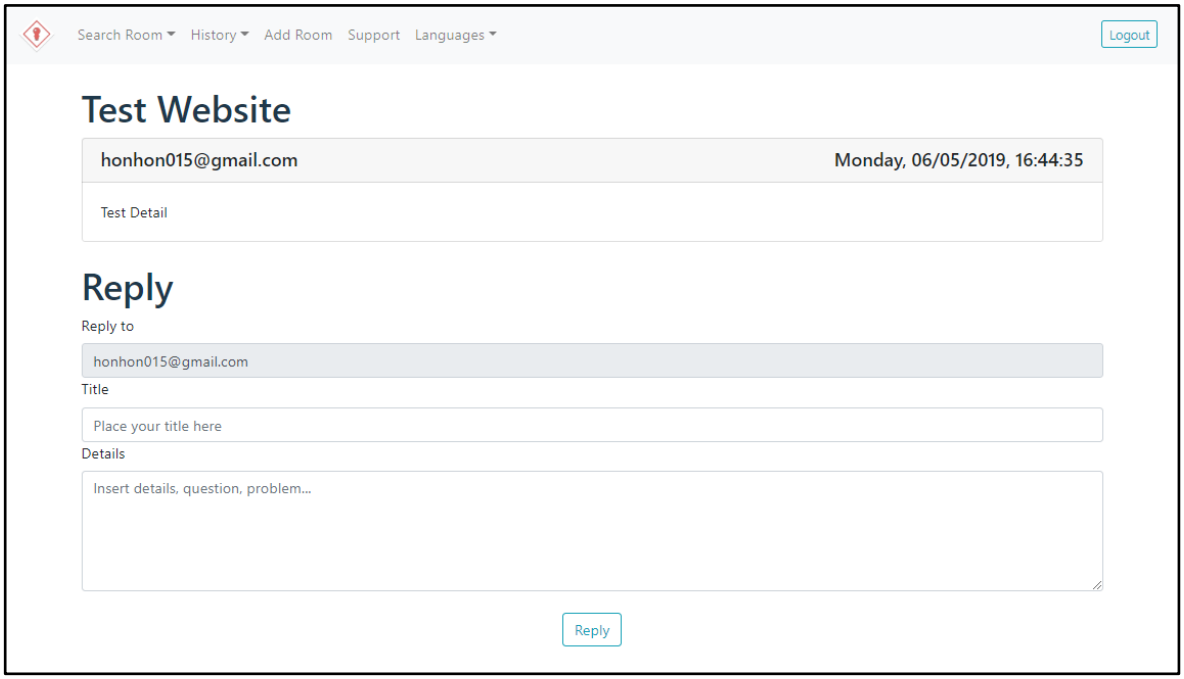

Figure 4.22 Reply page for admin

The page shown in figure 4.22 allow the user reply user message. This page is where admin can read the full version of the comment and reply them. The website will convert the data and send to user email.

Figure 4.23 The use flow of the user interface for admin

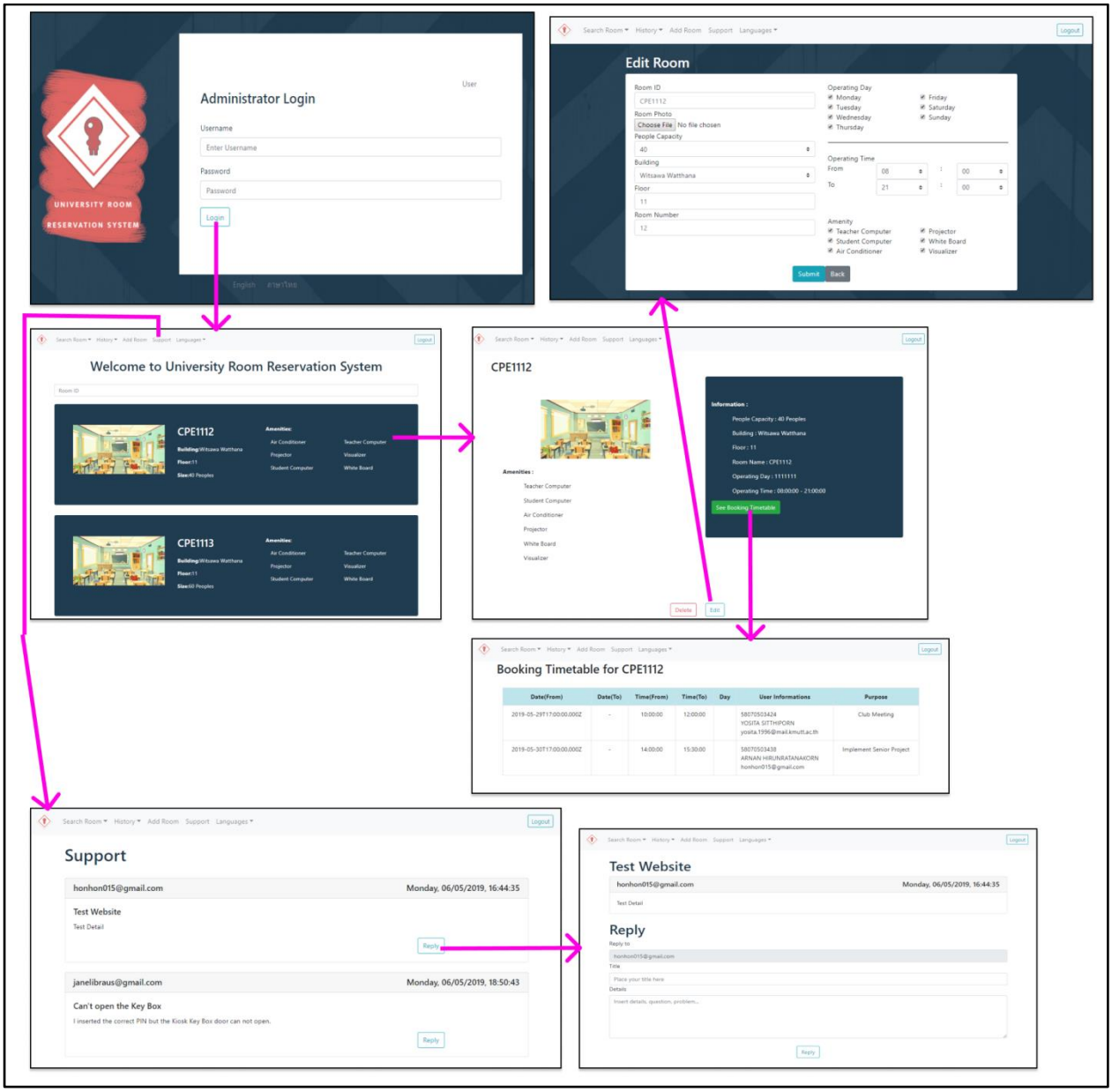

Figure 4.24 The continue of use flow of the user interface for admin

Figure 4.23 and figure 4.24 summarizes the navigation from screen to screen for an admin user.

4.2.3 User Use Flow for each use case

The use flow for the web application is very basic. The use flow for each use case is shown below.

4.2.3.1 Login flow

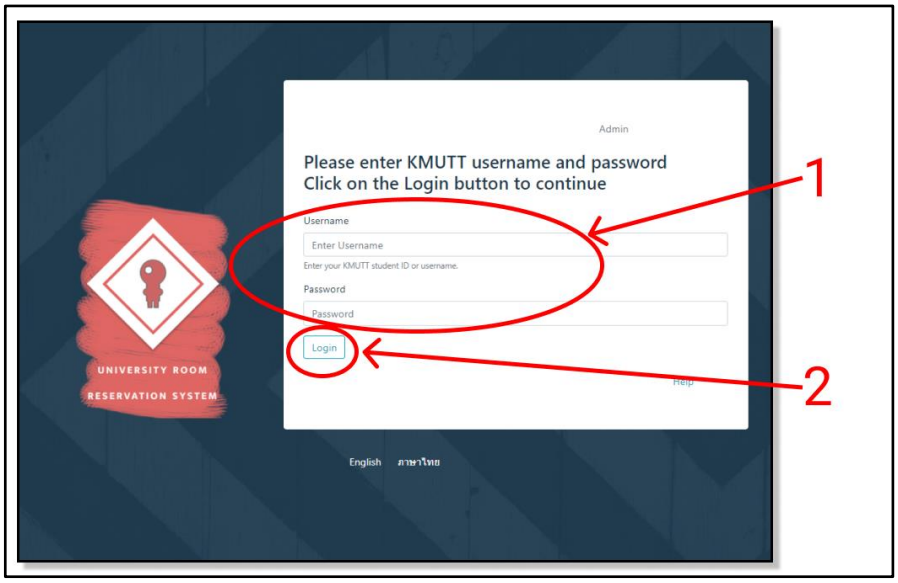

Figure 4.25 Use Flow for Login

Figure 4.25 shown the use flow for login step-by-step.

1. User need to fill KMUTT username and password into the form
2. Click Login Button

4.2.3.2 Logout flow

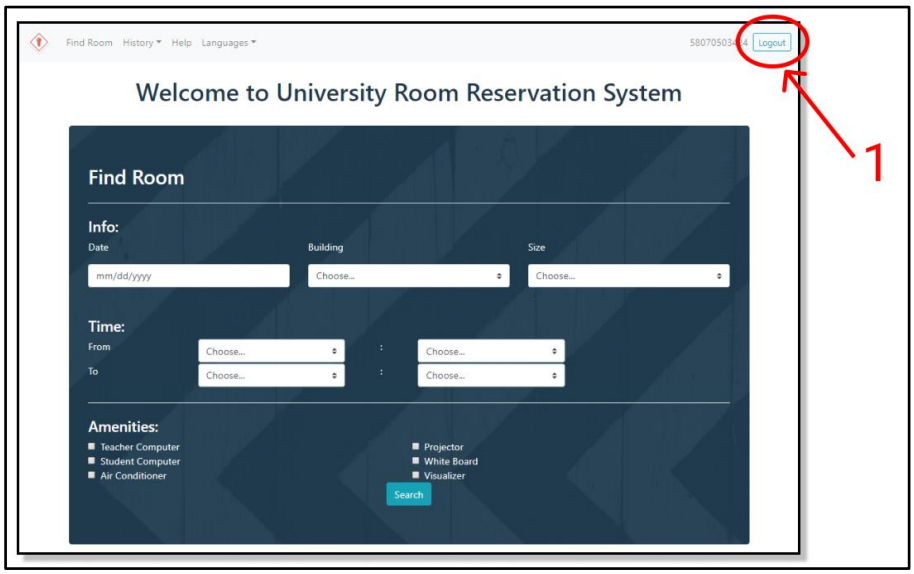

Figure 4.26 Use Flow for Logout

Figure 4.32 shown the use flow to logout from the system.

1. Click on the logout button which is on the top-right of the page.

4.2.3.3 Booking flow

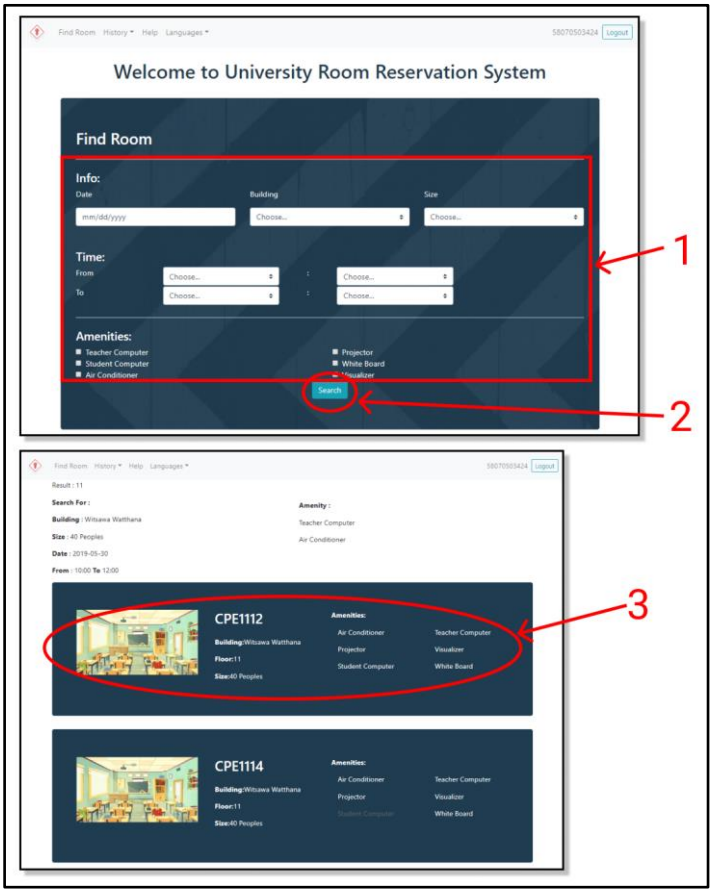

Figure 4.27 Use Flow for Booking 1

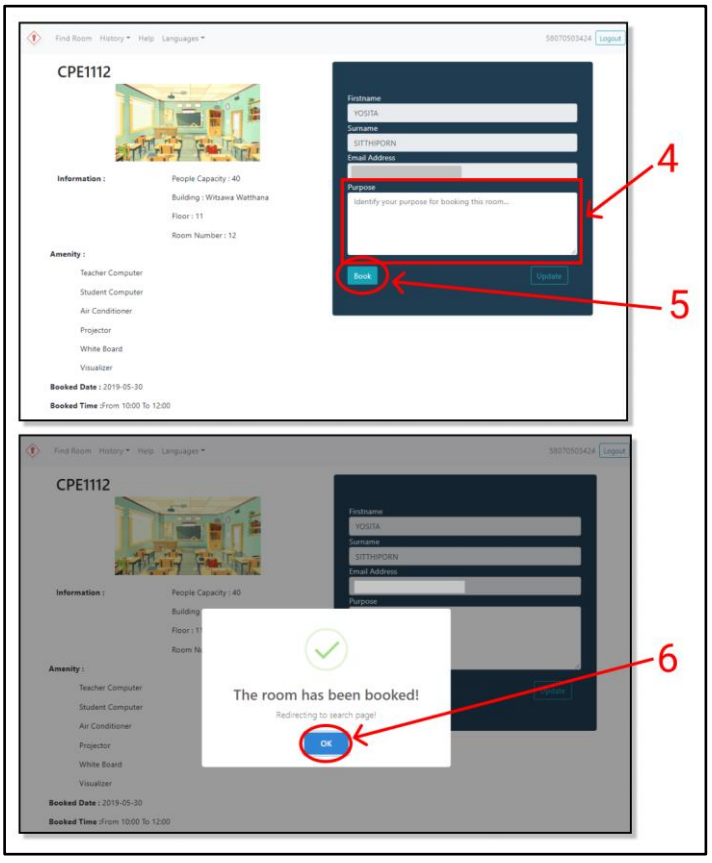

Figure 4.28 Use Flow for Booking 2

Figures 4.27 and 4.28 shown the use flow for booking step-by-step.

1. User need to fill in the form to search for available room
2. Click search button. The website will redirect to available room page.
3. Click on the room that you want to book. The website will redirect to booking page.
4. Fill in the purpose for booking the room
5. Click the book button to book the room.
6. The success message will be shown, click on the button to redirect back to search page.

4.2.3.4 Delete Booking flow

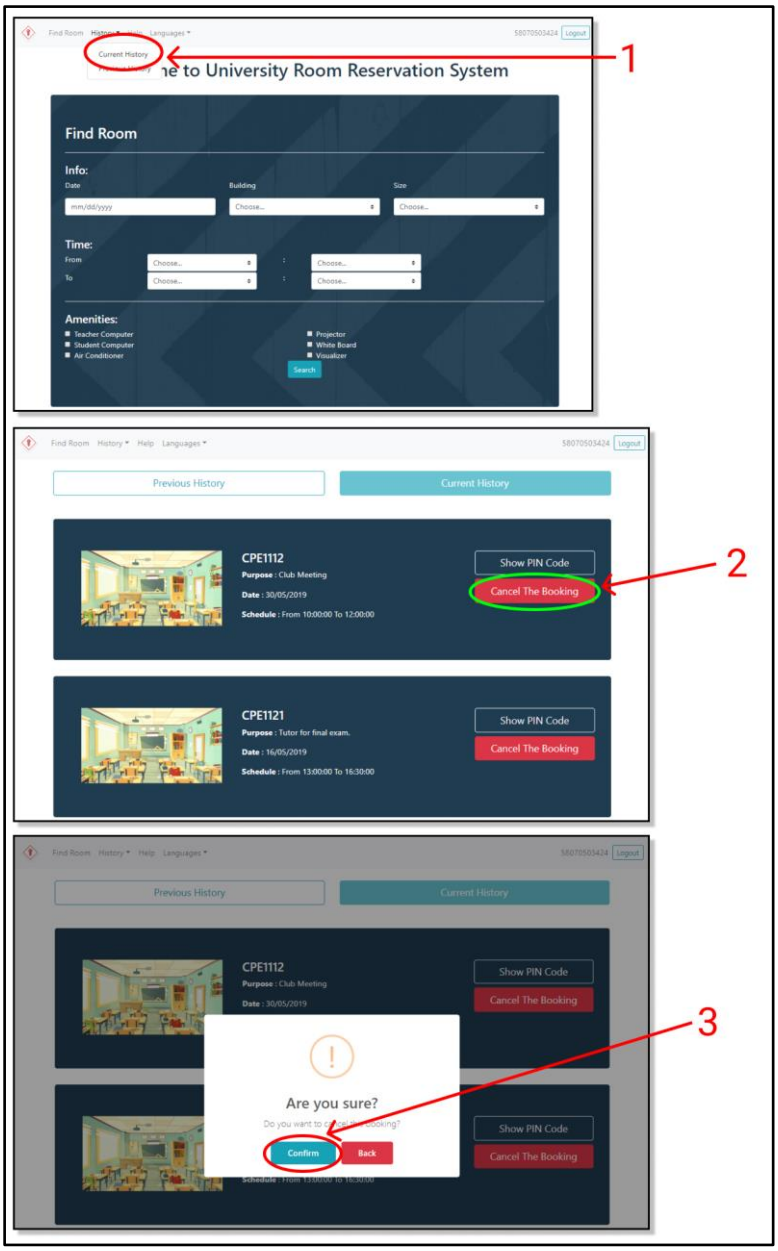

Figure 4.29 Use Flow for Cancel Booking

Figure 4.29 shows the use flow to cancel the booking step-by-step.

1. User need to click the link on the navbar to redirect to Current History Page.
2. Click on the “Cancel the Booking” button on the booking that want to cancel.
3. Confirm to delete the booking.

4.2.3.5 Get PIN flow

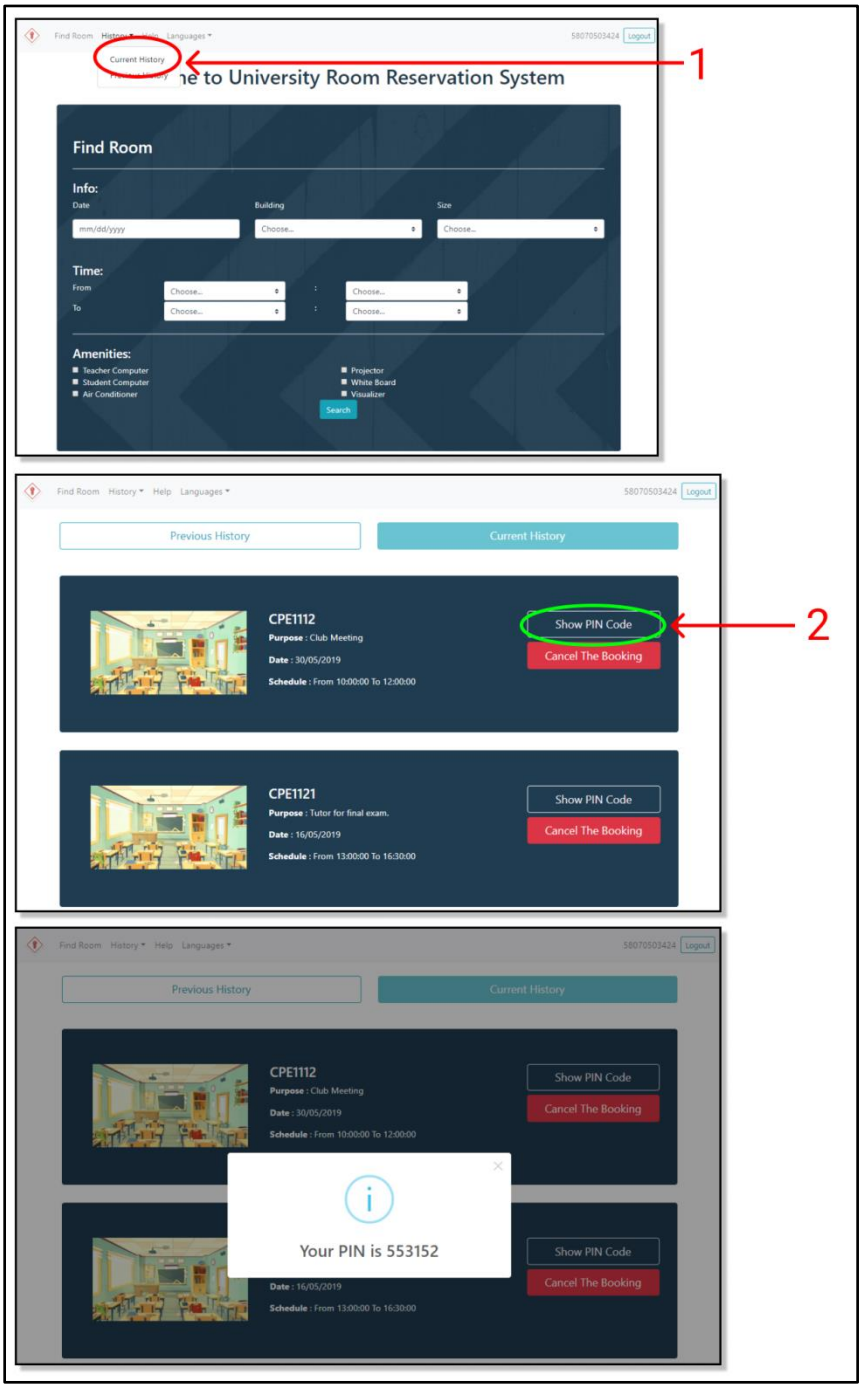

Figure 4.30 Use Flow for get PIN

Figure 4.30 shown the use flow to get the PIN for each booking step-by-step.

1. User need to click the link on the navbar to redirect to Current History Page.
2. Click on the “Show PIN Code” button on the booking that want to get the PIN.

4.2.3.6 Change Language flow

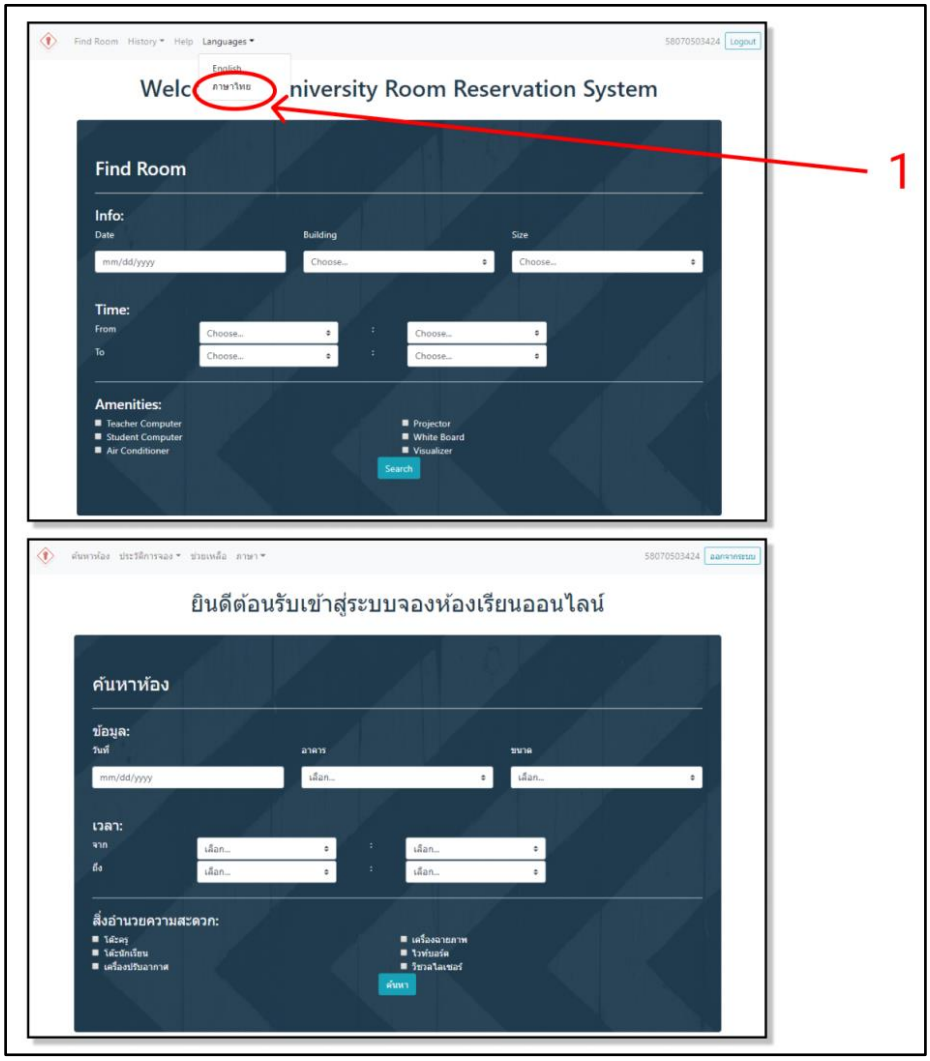

Figure 4.31 Use Flow for Change Language

Figure 4.31 shown the use flow to change the web application language.

1. User need to click the language link on the navbar to change the language. The page will be refreshed and show the selected language.
2. This language will be used on all displayed pages from that point.

4.2.3.7 Contact Admin flow

Figure 4.32 shown the use flow to contact the admin (No authentication needed).

1. Click on the “Help” button that existed on every page to redirect to Contact Us page.
2. Fill in the form the report problem to admin.
3. Click on submit button to send the message to Admin.

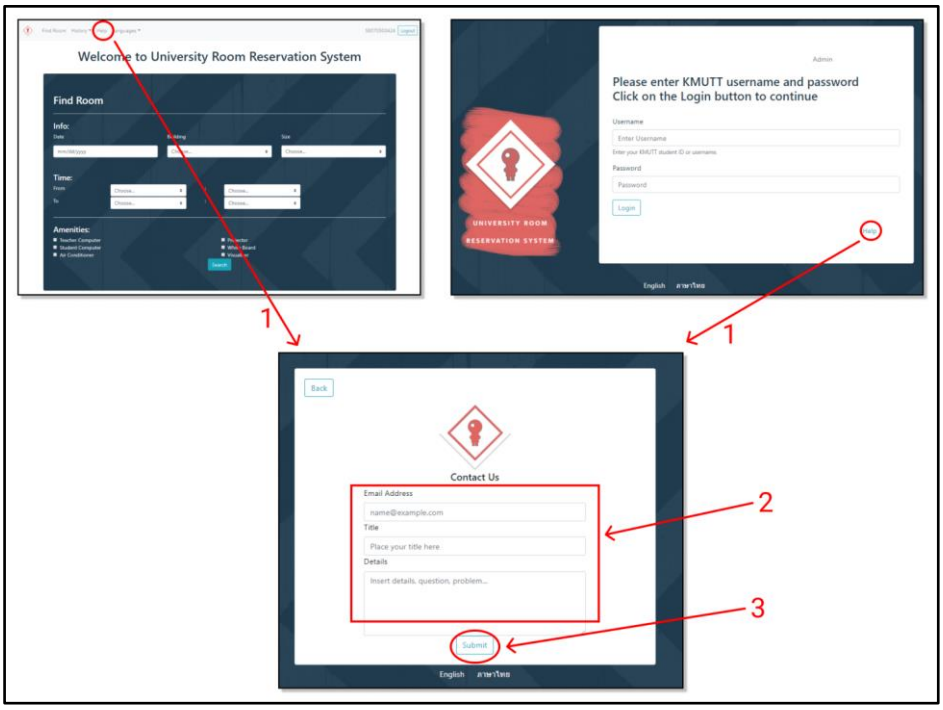

Figure 4.32 Use Flow for Contact Admin

4.2.4 Admin Use Flow for each use case

The use flow for the system admin will be a little more complex than normal user. Admin will have some feature that normal user doesn't. The use flow for each use case is shown below.

4.2.4.1 Admin Search Room flow

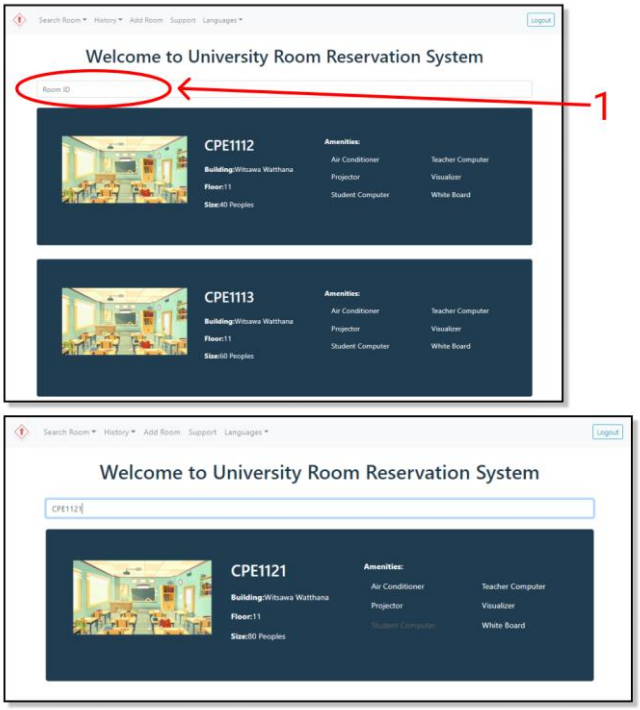

Figure 4.33 Admin Use Flow for Search Room by Name

Figure 4.33 shown the admin use flow for search room by name.

1. On the main page, fill in the Room name that want to search.

4.2.4.2 Admin Normal Booking flow

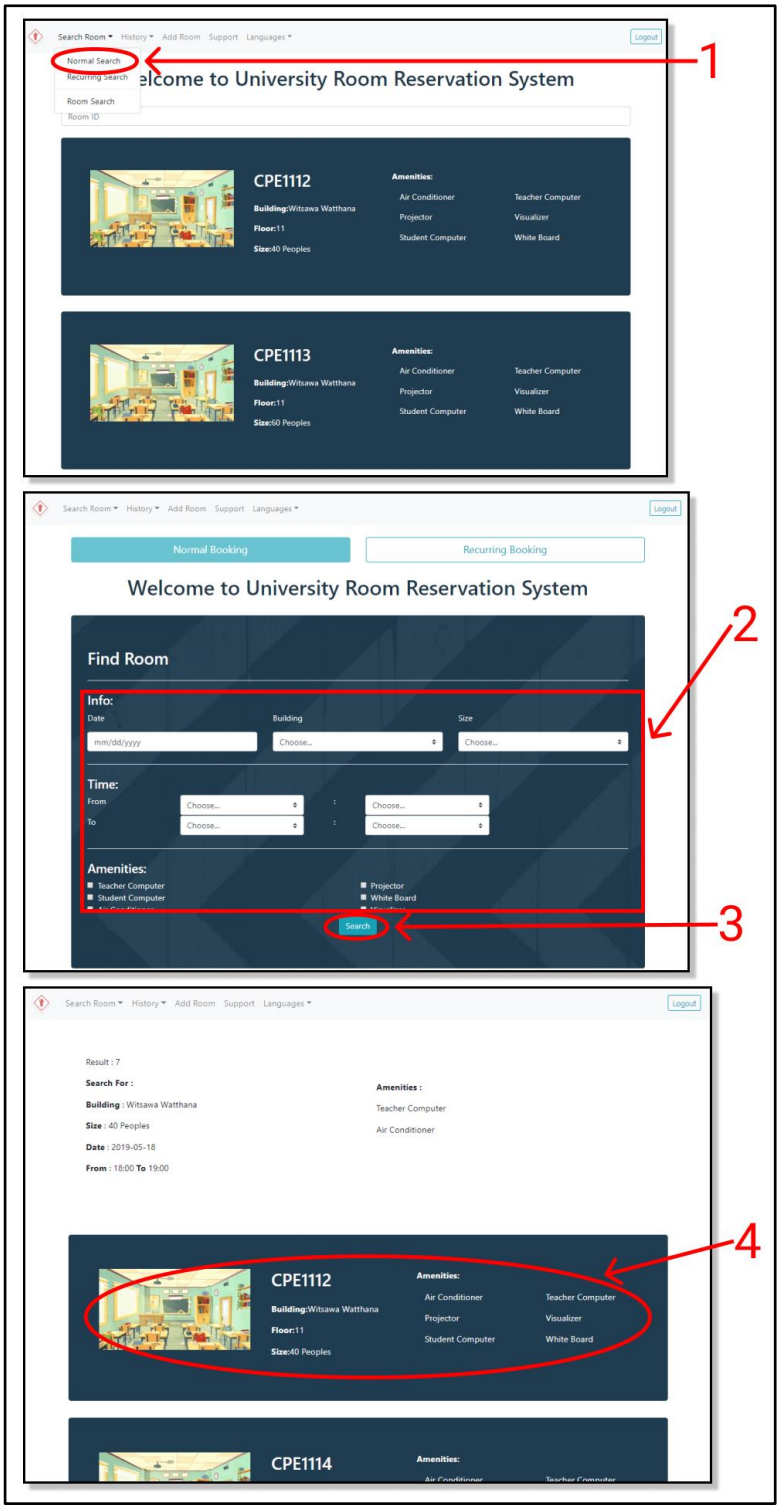

Figure 4.34 Admin Use Flow for Normal Booking 1

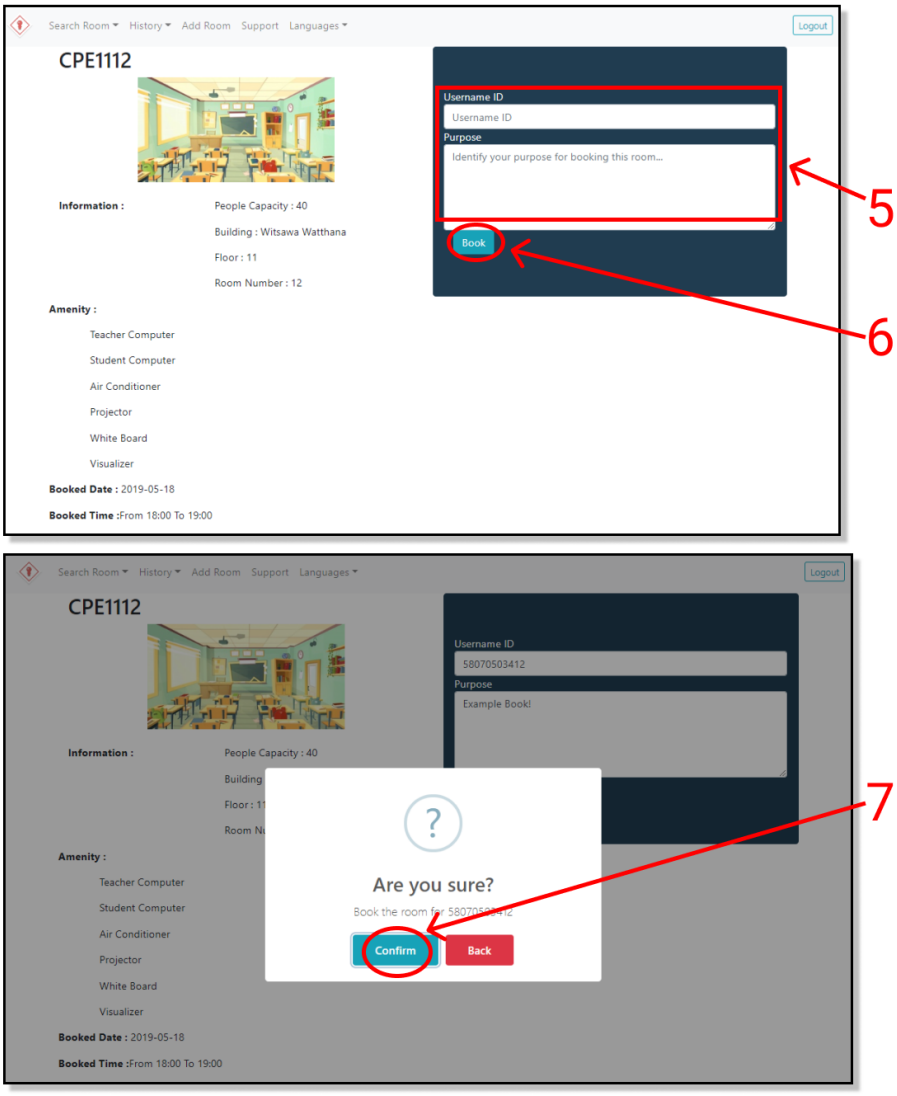

Figure 4.35 Admin Use Flow for Normal Booking 2

Figure 4.34 and 4.35 shown the admin use flow for normal booking step-by-step.

1. Click on “Normal Search” link on the navbar to redirect to “Normal Search” Page.
2. Fill in the form to search for available room.
3. Click search button. The website will redirect to available room page.
4. Click on the room that you want to book. The website will redirect to admin booking page.
5. Fill in the purpose for booking the room and the username who you want to book the room for.
6. Click the book button to book the room.
7. The confirmation message will be shown, click on the confirm button to book the room.

4.2.4.3 Admin Recurring Booking flow

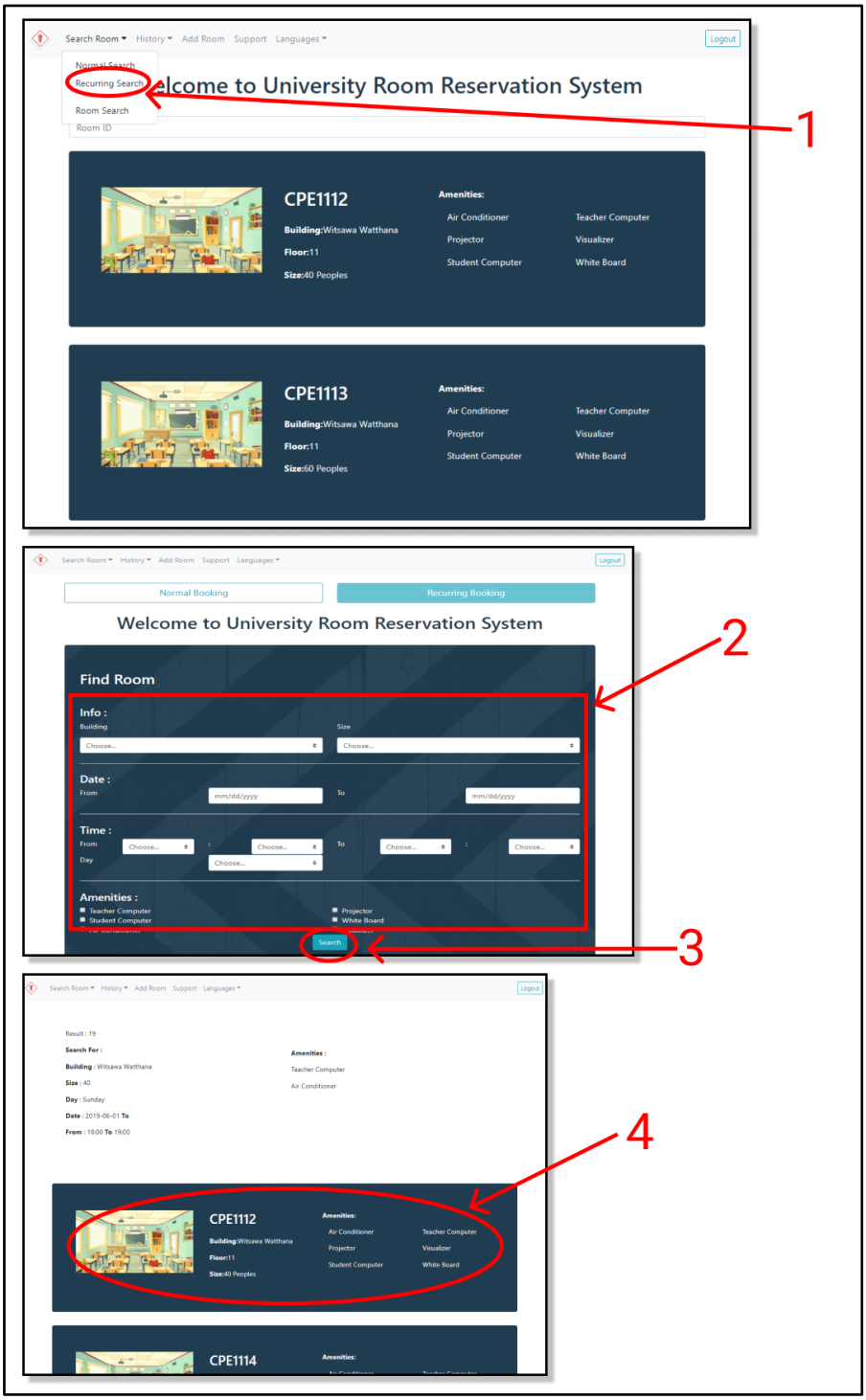

Figure 4.36 Admin Use Flow for Recurring Booking 1

Figure 4.37 Admin Use Flow for Recurring Booking 2

Figure 4.36 and 4.37 shown the admin use flow for recurring booking step-by-step.

1. Click on “Recurring Search” link on the navbar to redirect to “Recurring Search” Page.
2. Fill in the form to search for available room.
3. Click search button. The website will redirect to available room page.
4. Click on the room that you want to book. The website will redirect to admin recurring booking page.
5. Fill in the purpose for booking the room and the username who you want to book the room for. The semester is optional.
6. Choose year, sections, study program. This is optional, can be leave blank.
7. Click the book button to book the room.
8. The confirmation message will be shown, click on the confirm button to book the room.

4.2.4.4 Admin Delete Booking flow

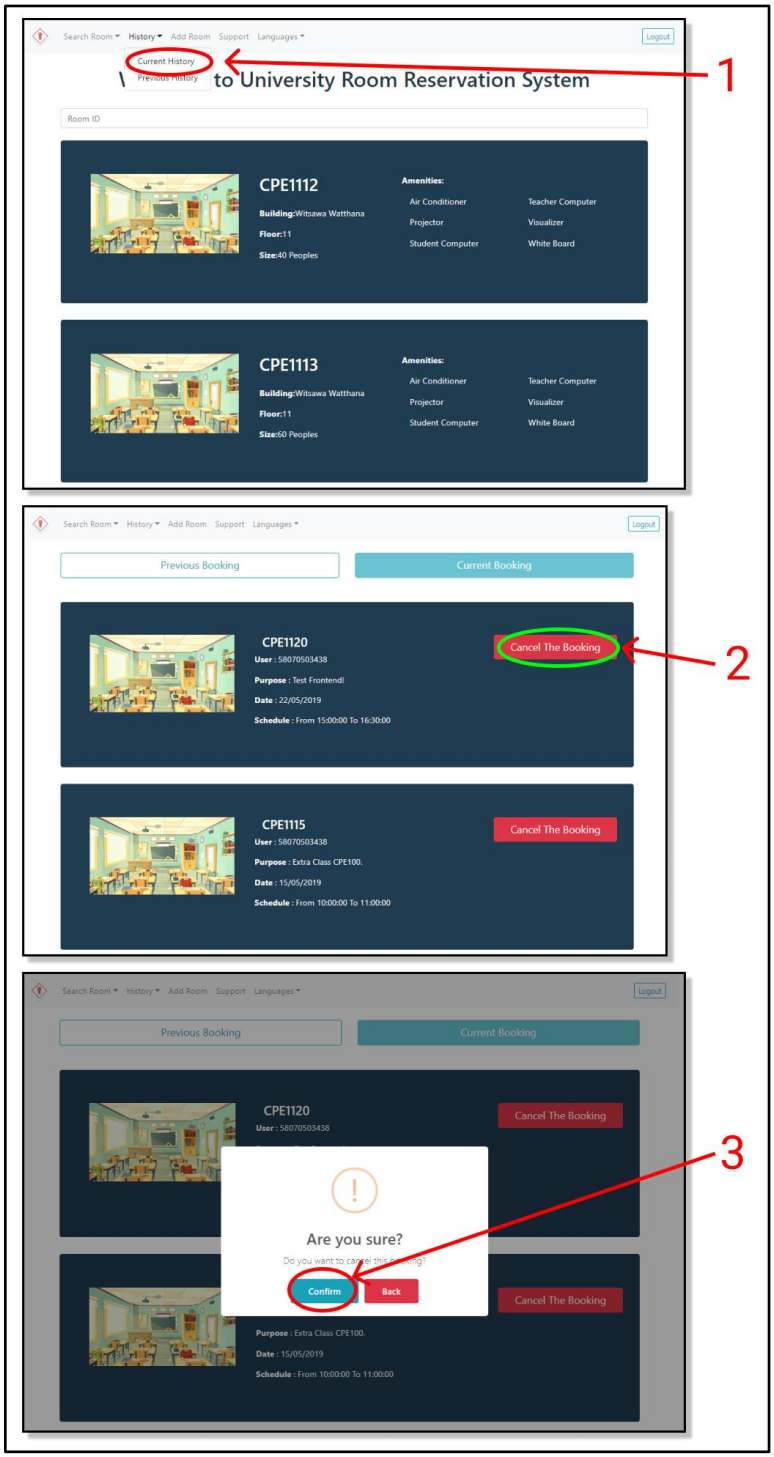

Figure 4.38 Admin Use Flow for Cancel the Booking

Figure 4.38 shown the admin use flow for Cancel the booking step-by-step.

1. Click on “Current History” link on the navbar to redirect to “Current History” Page.
2. Click on the “Cancel the Booking” button.
3. Click confirm to cancel the reservation.

4.2.4.5 Admin Look Timetable flow

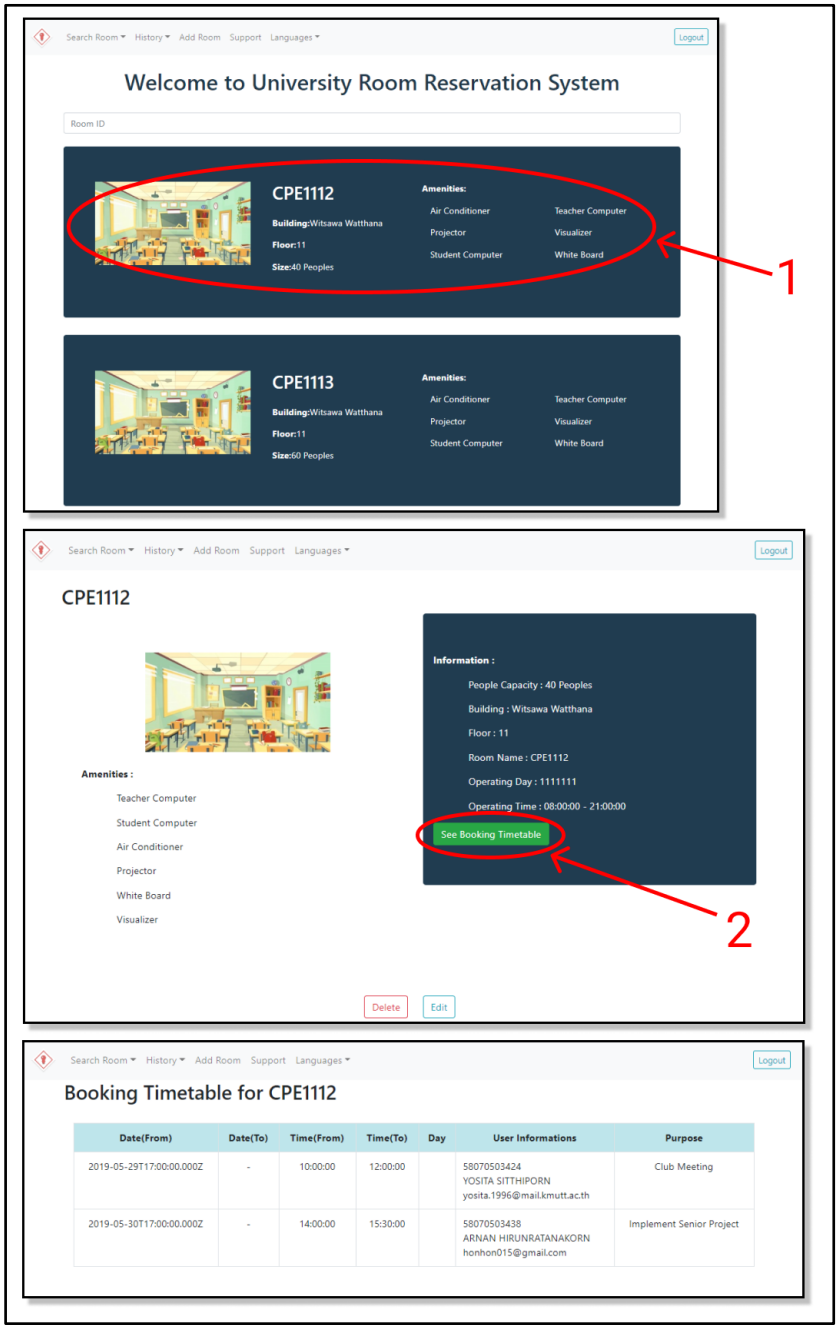

Figure 4.39 Admin Use Flow for Looking timetable

Figure 4.39 shown the admin use flow for looking time table of each room.

1. On the main page click on the room, the page will be redirect to room information page.
2. Click on the “See Booking Timetable” button to redirect to Timetable page.

4.2.4.6 Admin Delete Room flow

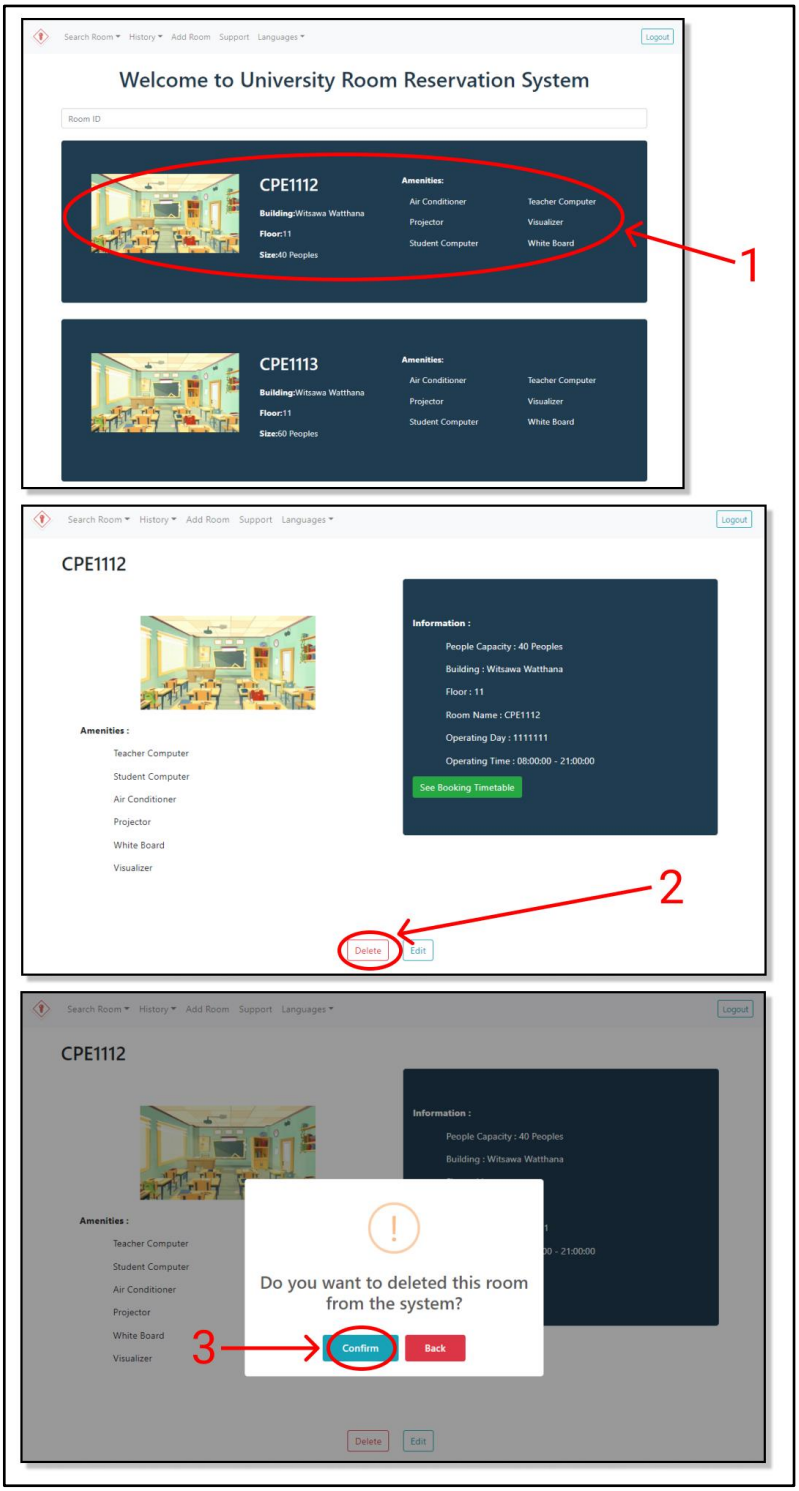

Figure 4.40 Admin Use Flow for Delete Room

Figure 4.40 shown the admin use flow for looking time table of each room.

1. On the main page click on the room, the page will be redirect to room information page.
2. Click on the “Delete” button.
3. Click confirm the delete the room.

4.2.4.7 Admin Edit Room flow

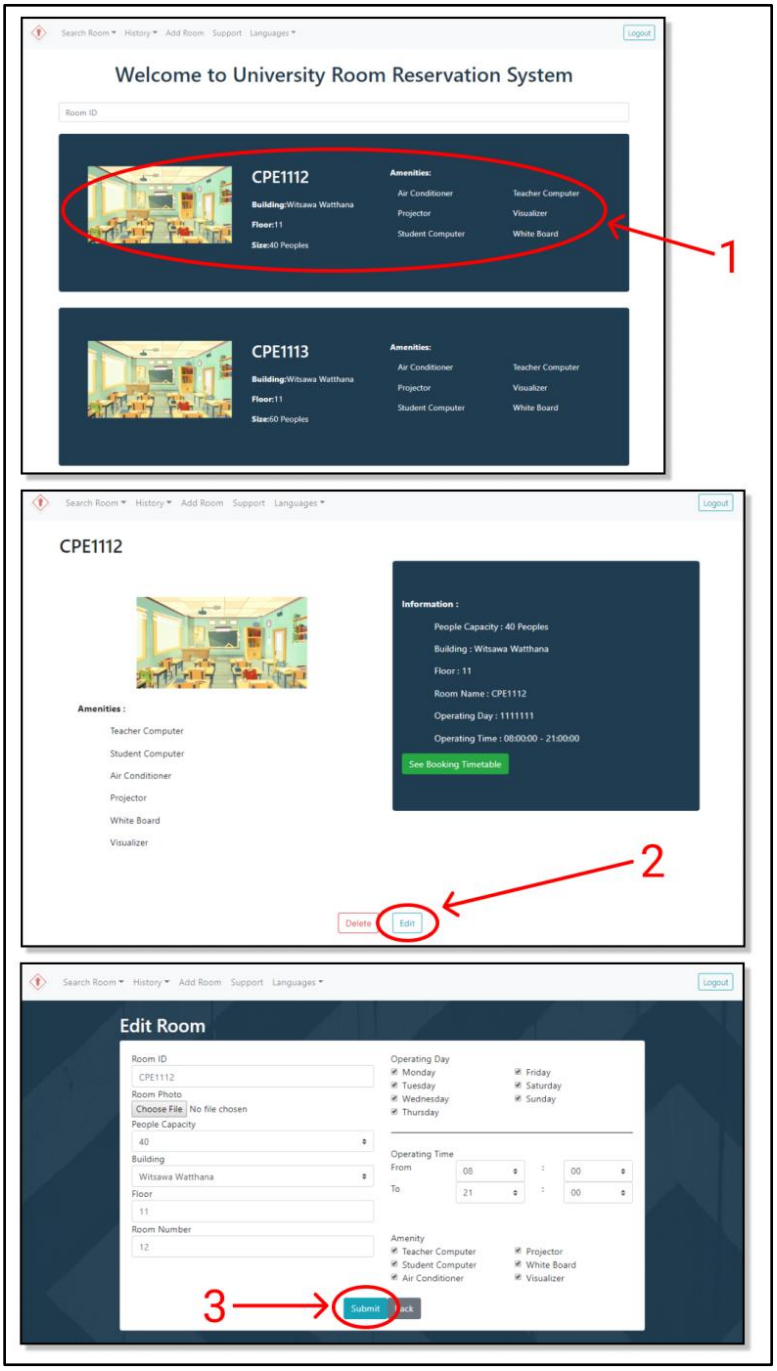

Figure 4.41 Admin Use Flow for Edit Room

Figure 4.41 shown the admin use flow for edit room.

1. On the main page click on the room, the page will be redirect to room information page.
2. Click on the “Edit” button. The page will be redirect to edit room page.
3. Modify the information of the room and click submit to save the data.

4.2.4.8 Admin Reply Contact flow

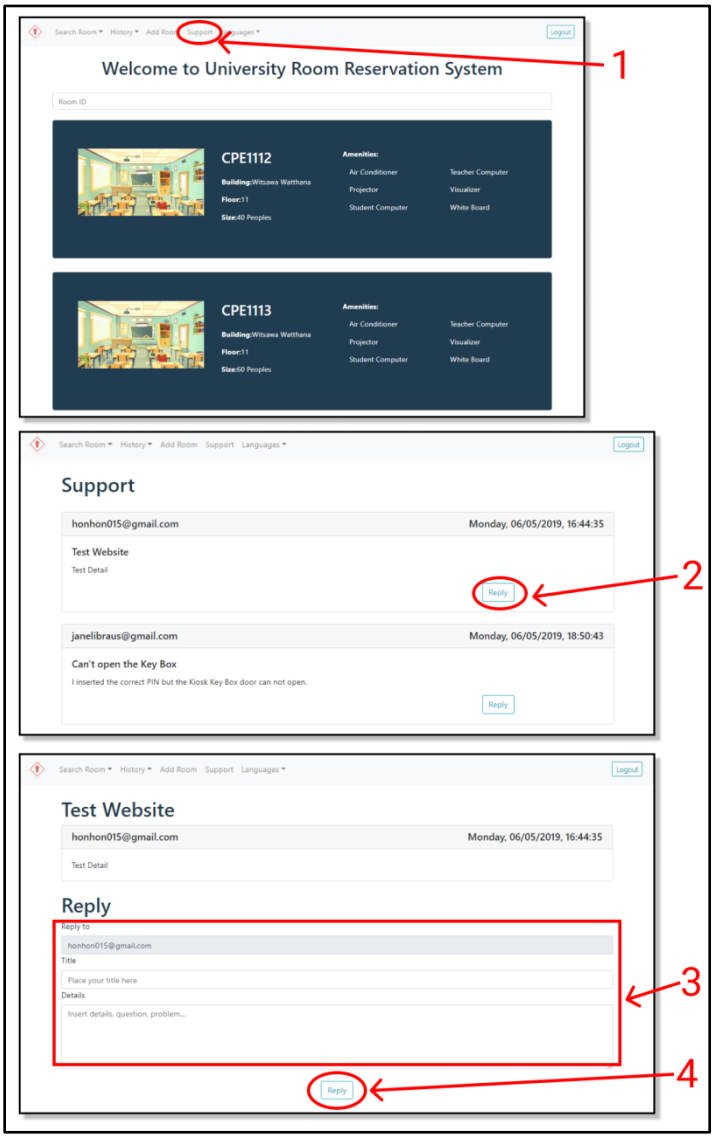

Figure 4.42 Admin Use Flow for Reply Contact

Figure 4.42 shown the admin use flow for add room.

1. Click on “Support” link on the navbar to redirect to “Support” Page.
2. Choose the message to answer.
3. Fill in the form.
4. Click reply to reply to user email.

4.2.4.9 Admin Add Room flow

Figure 4.43 shown the admin use flow for add room.

1. Click on “Add Room” link on the navbar to redirect to “AddRoom” Page.
2. Fill in the information of the room.
3. Click submit button to save the data.

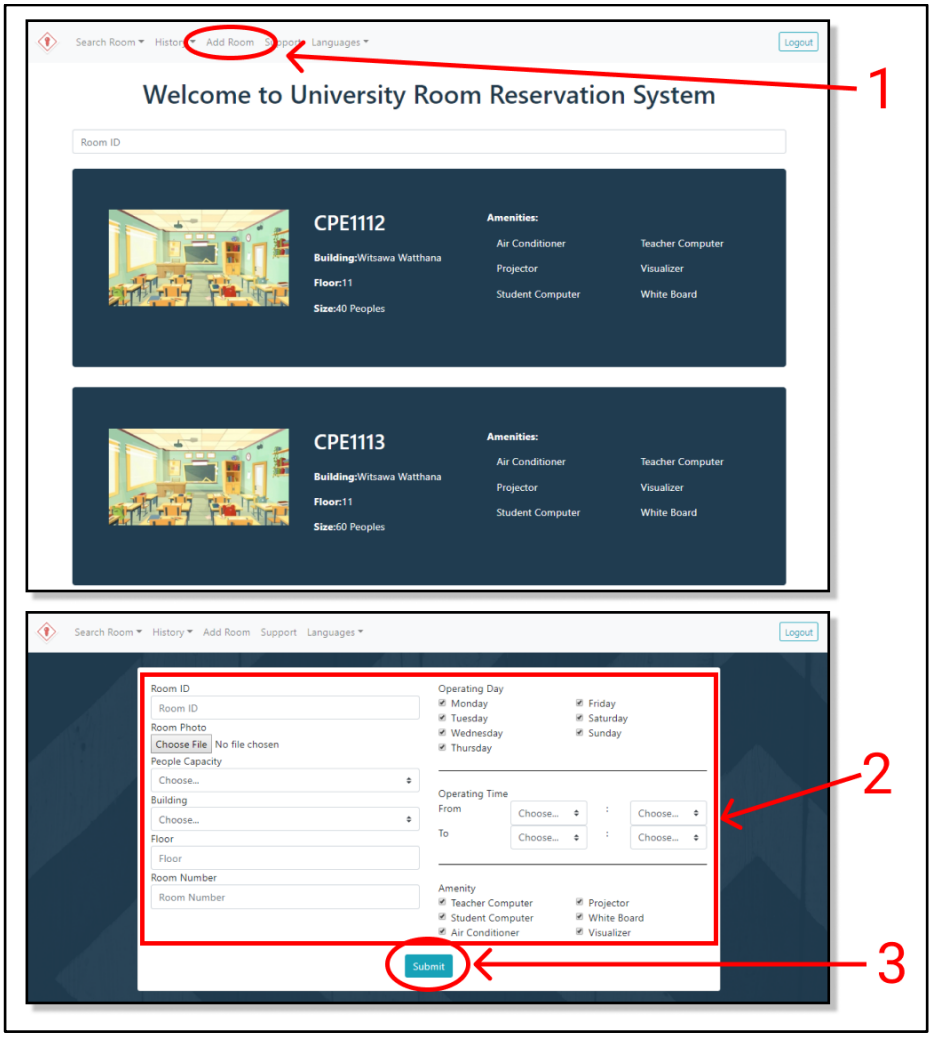

Figure 4.43 Admin Use Flow for Add Room

4.3 Kiosk Key Box

4.3.1 Kiosk Key Box Design

The Kiosk Key Box was designed to use 3D printed components for its main structure. Inside it will include Raspberry Pi Zero W, L298N Motor Driver, Mini DC Motor, 3*4 Keypad, LCD1602 Monitor, RFID reader and RFID tag attached to the key. The Kiosk Key Box will be separated into two parts which are top side that contains electronic components and bottom size for storing the key.

4.3.1.1 Main Box Design

Each part of the Kiosk Key Box that has been designed without the component inside is shown below.

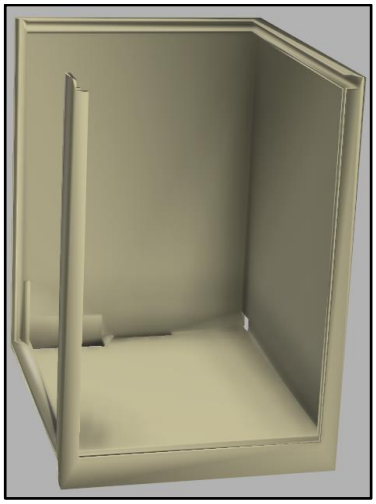

Figure 4.44 Main Box Part

The Main Box is the main part used in Kiosk Key Box to cover most of components inside. It has a slot to connect to the bottom box below. It provides a small hole at the bottom for wiring with components in the bottom box and another at the right bottom corner for electricity.

Figure 4.45 Left Part

The left part is the part that use to install on left side of main box to be the left wall of the Kiosk Key Box.

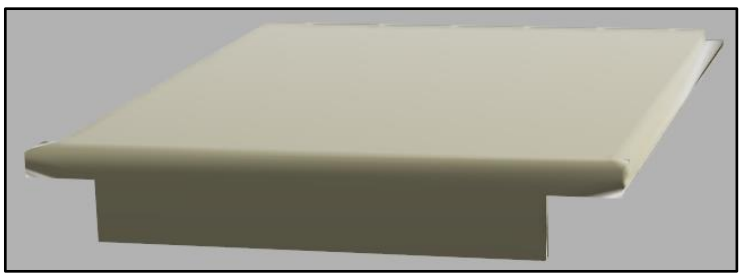

Figure 4.46 Top Part

The top part is the part that use to install on the top of the main box to cover upper side of Kiosk Key Box.

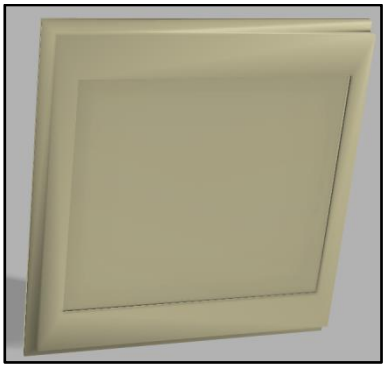

Figure 4.47 Keypad Box Part

Keypad Box Part is the part that use to contain the keypad and install in front of the main box. It provided one square holes for 3*4 Keypad.

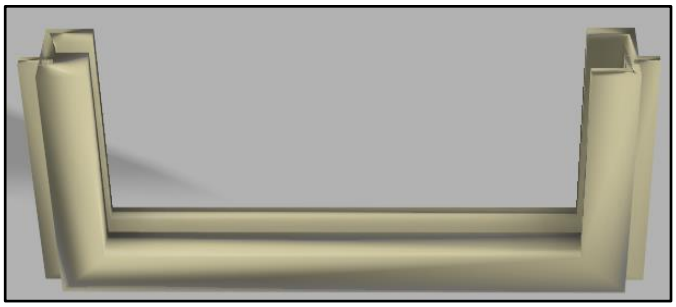

Figure 4.48 Monitor Box Part

Monitor Box Part is the part that use to install in front of the main box on the Keypad Box Part. LCD1602 monitor can be insert from the top of the box.

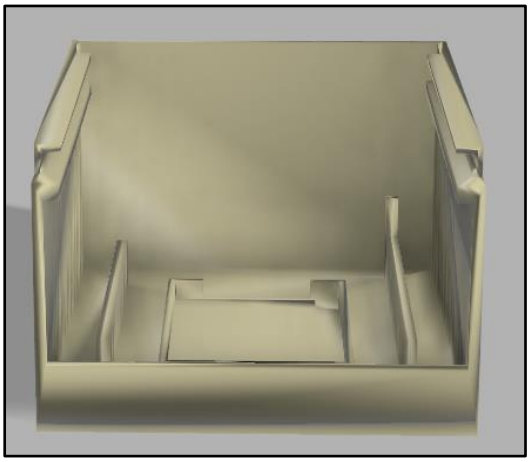

Figure 4.49 Bottom Part

The bottom part is the part use to install under the main box. It provides the space for placing RFID Reader under the space for key slot to be able to read through the key slot and see the RFID tag on the upper side.

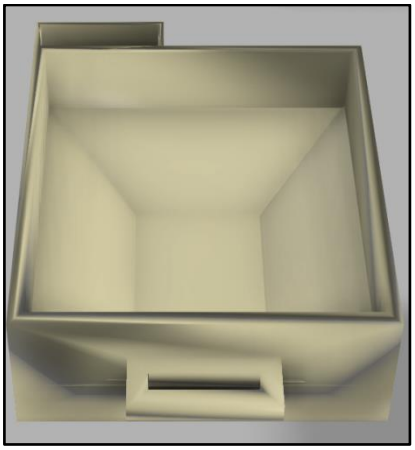

Figure 4.50 Key Slot Part

The key slot can be inserted into the bottom part of the key box. It provides the space for placing the key with RFID on it. On the back of the key slot, there is a square hole for locking purpose.

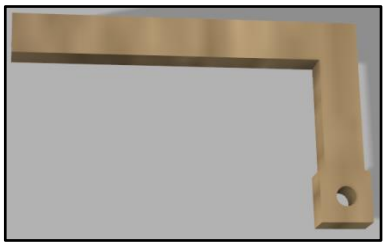

Figure 4.51 Lock part

The lock part is designed to with DC Motor and use to lock the key slot with the main box.

Figure 4.52 Merged Key Box

Figure 4.52 above shows the overall appearance of 3D model of kiosk key box after merging the parts together.

4.3.1.2 Components Inside Kiosk Key Box

Each part of the components inside the Kiosk Key Box is shown below.

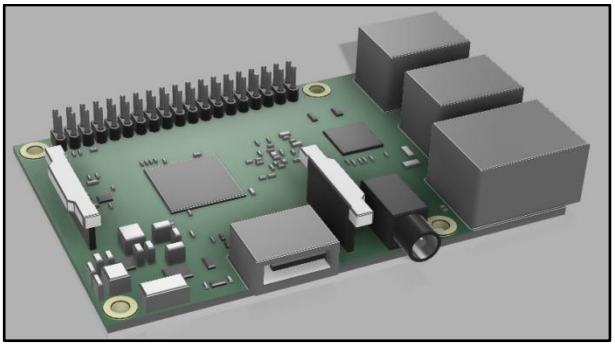

Figure 4.53 Raspberry Pi

The Raspberry PI is used as the main board for the Kiosk Key Box. It is designed to be installed on the floor in main box near the corner with hole to connect the electricity wire from outside.

Figure 4.54 3*4 Keypad

The 3*4 Keypad is designed to be installed in the front of the keybox to get the user input.

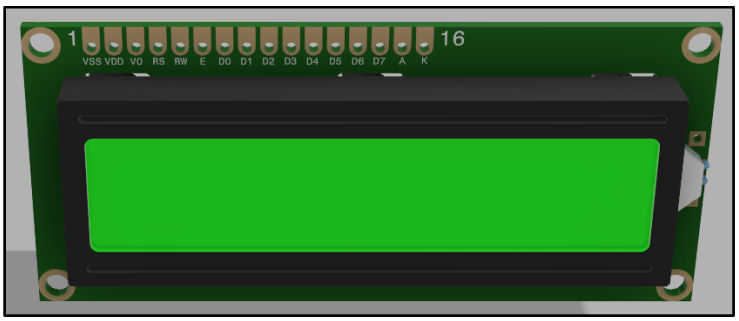

Figure 4.55 LCD1602 Monitor

The LCD1602 Monitor is designed to be installed in the front of the keybox above the 3*4 Keypad. It shows the PIN as the user entering it, as well as any messages from the box, such as “Incorrect PIN” or “Please Open”.

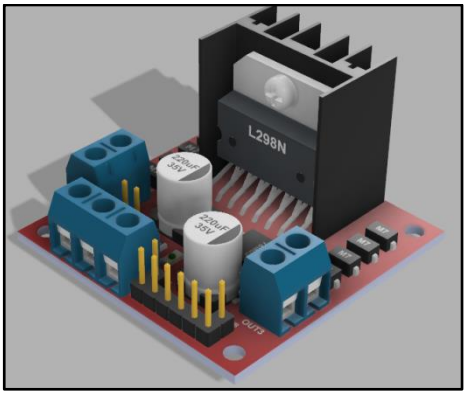

Figure 4.56 L298N Motor Driver

The L298N Motor Driver is designed to be installed behind the Keypad in the main box. It is used to control the Mini DC Motor

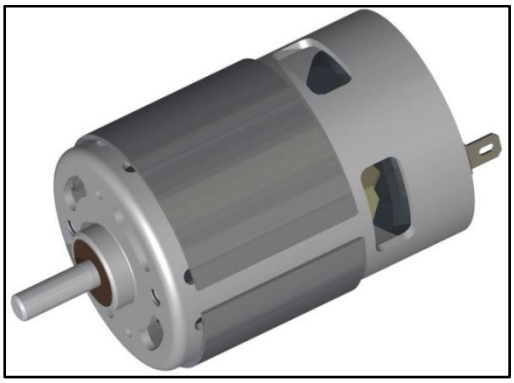

Figure 4.57 Mini Dc Motor

The Mini Dc Motor is placed in the main box near the back. It will be attached to the lock part.

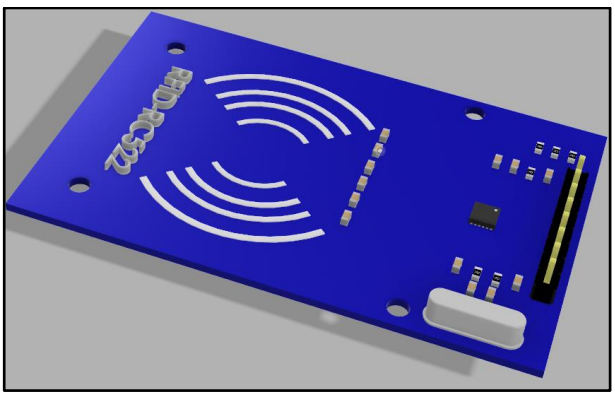

Figure 4.58 MFRC522 RFID Reader

The RFID Reader is designed to be inserted under the Key Slot part on the bottom part floor to be able to read the data from RFID inside the Key Slot.

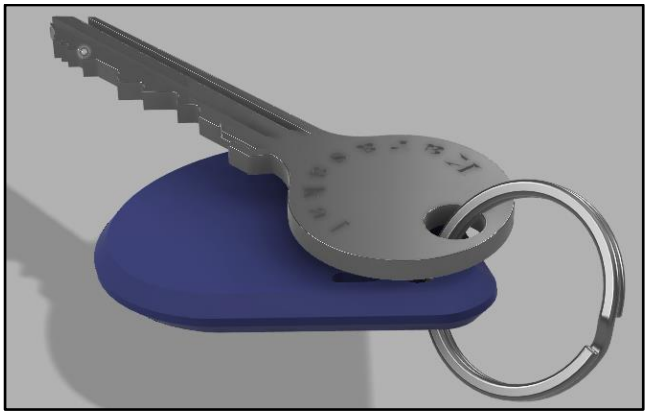

Figure 4.59 RFID Tag

The RFID tag comes with the key. It will be placed in the Key Slot Part. The RFID Reader will be able to detect the RFID tag through the floor. The key can be removed from the Kiosk Key Box to unlock the room.

4.3.1.3 Kiosk Key Box Circuit Design

The circuit design of the components inside the kiosk key box are shown below Figure 4.60 show how to connect 3*4 Keypad and LCD 1602 to Raspberry Pi.

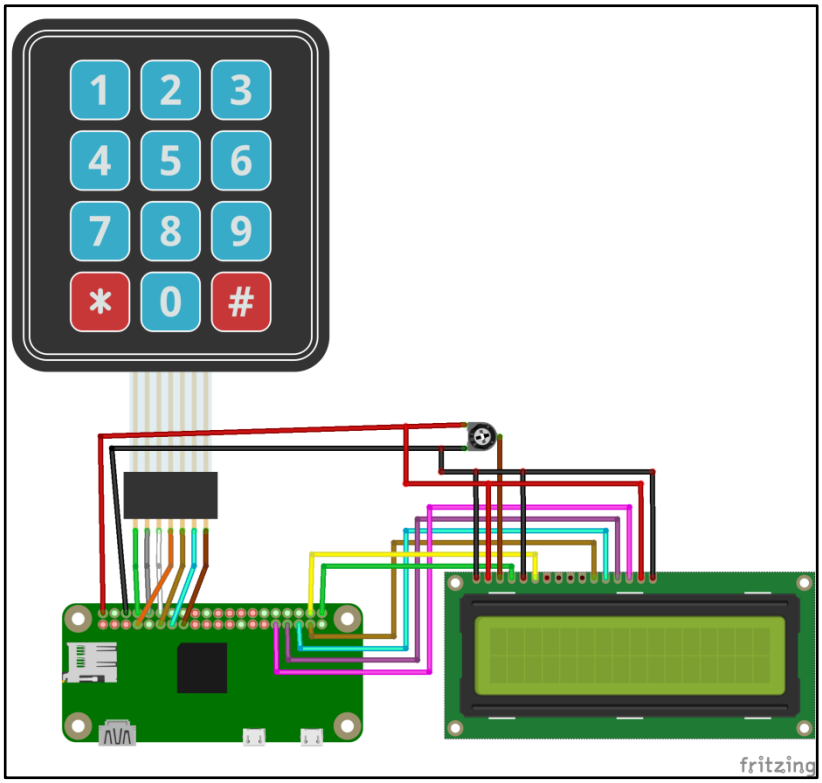

Figure 4.60 3*4 Keypad and LCD 1602 Monitor Circuit Design

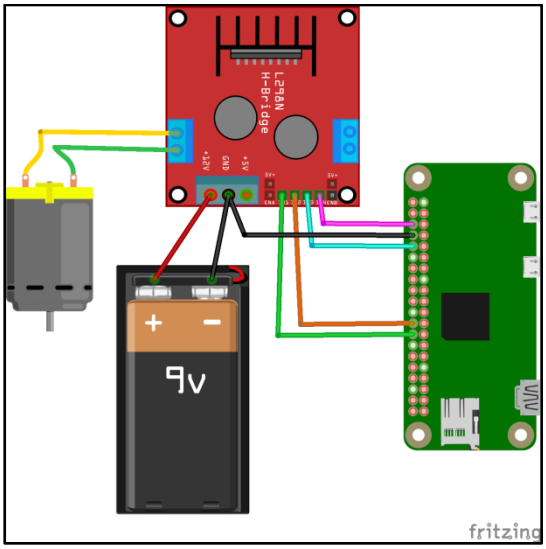

Figure 4.61 L298N and DC Motor Circuit Design

Figure 4.61 show how to connect L298N, DC Motor, 9V battery and Raspberry Pi to make able it works together. L298N will connect directly to Raspberry Pi while 9V battery and DC motor is connect to L298N. Figure 4.62 below show how to connect MFRC522 RFID Scanner to Raspberry Pi.

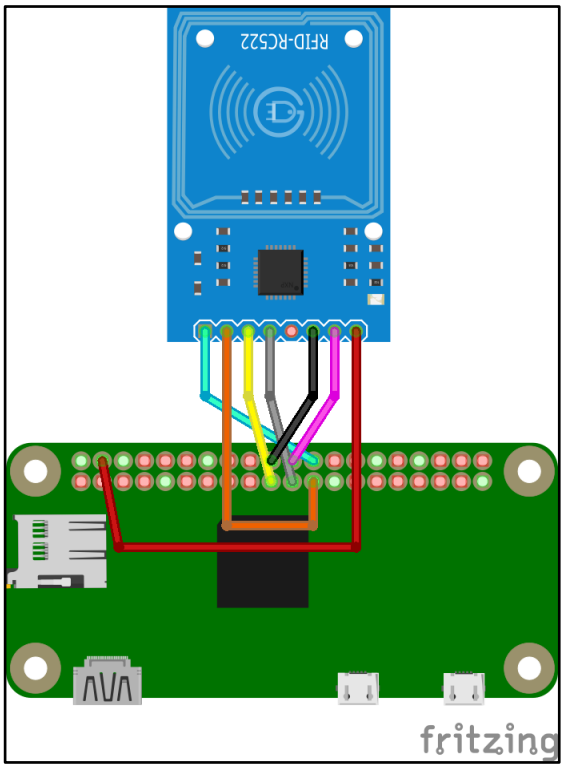

Figure 4.62 MFRC522 RFID Reader Circuit Design

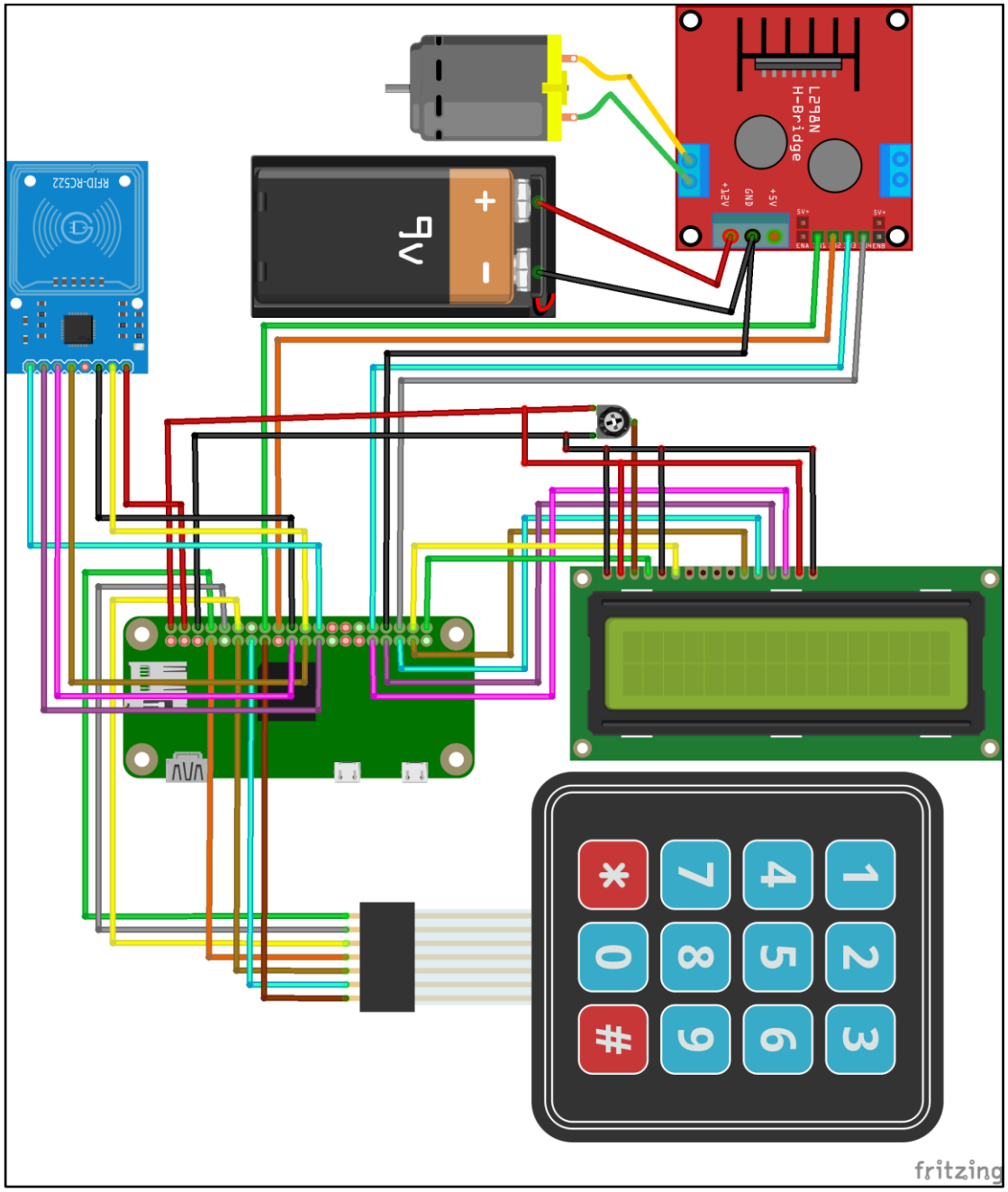

Figure 4.63 Circuit Design of Every Components Connected to Raspberry Pi

The design in figure 4.63 show how to connect MFRC522 RFID Scanner, L298N Motor Driver, 3*4 Keypad, LCD1602 Monitor to Raspberry Pi. In this circuit, LCD1602 monitor, 3*4 Keypad, MFRC522 RFID Scanner, L298N motor is connected directly to Raspberry Pi Zero W. While 9V battery and DC motor is connect to L298N because DC motor need 9V battery to power it. The program will get input data from keypad and send the data to display on the monitor. MFRC522 RFID Scanner will detect the existence of RFID tag (Key). The L298N motor driver will control the motor to move forward or reverse.

4.3.1.4 Mechanism of the Kiosk Key Box

The Kiosk Key Box will be able to work when all the component connects together, and Raspberry PI connected to power. The completed Kiosk Key Box is shown below.

Figure 4.64 Front Kiosk Key Box

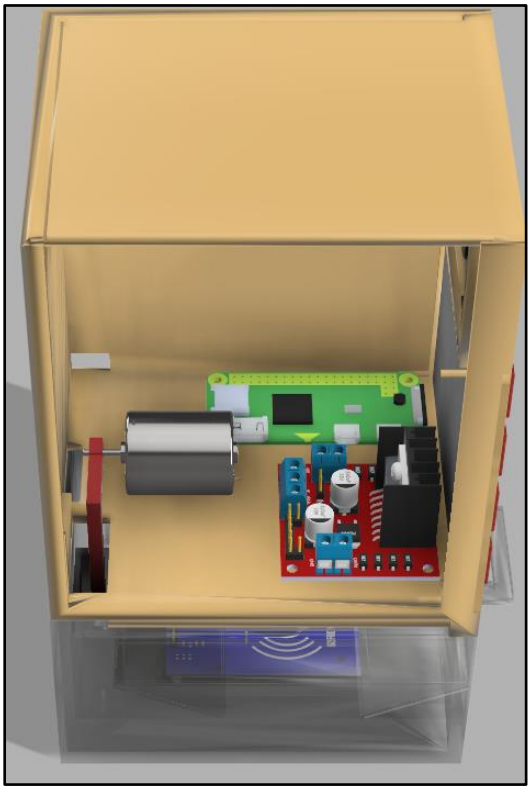

Figure 4.65 Inside Kiosk Key Box

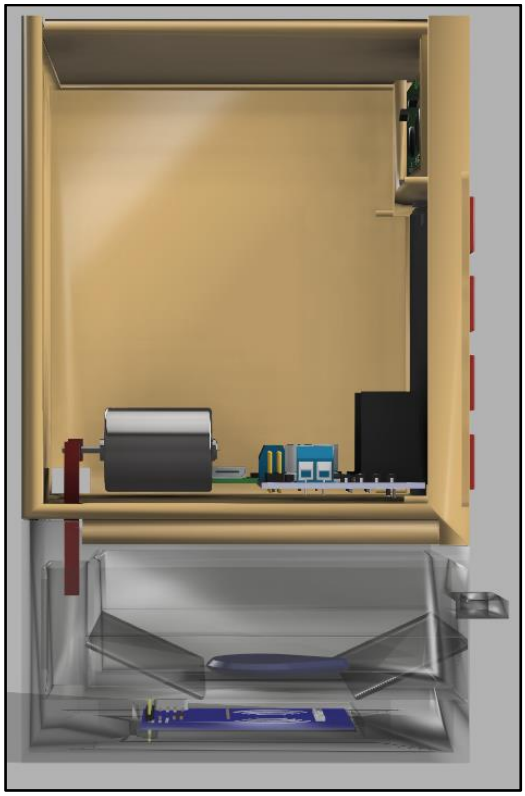

Figure 4.66 Inside Kiosk Key Box (Side)

The 3D Model of Kiosk Key Box mechanism will be shown step by step below. The use case begins when the user come to enter the PIN to get the key and ends when the user returns the key.

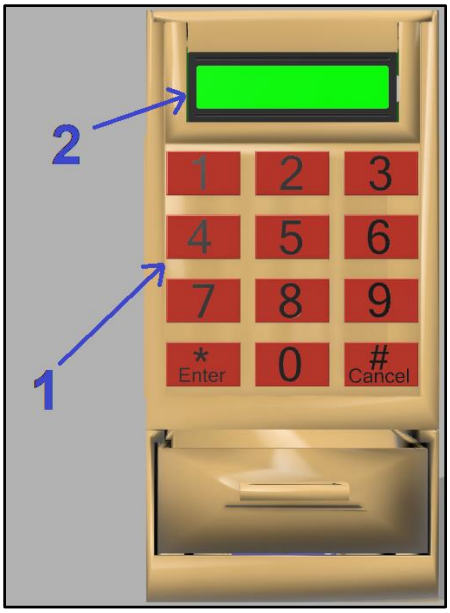

Figure 4.67 Insert PIN

1. User enters the PIN on the 3*4 Keypad
2. Raspberry Pi validates the PIN and shows the result on the LCD Monitor (If success go to step 3)

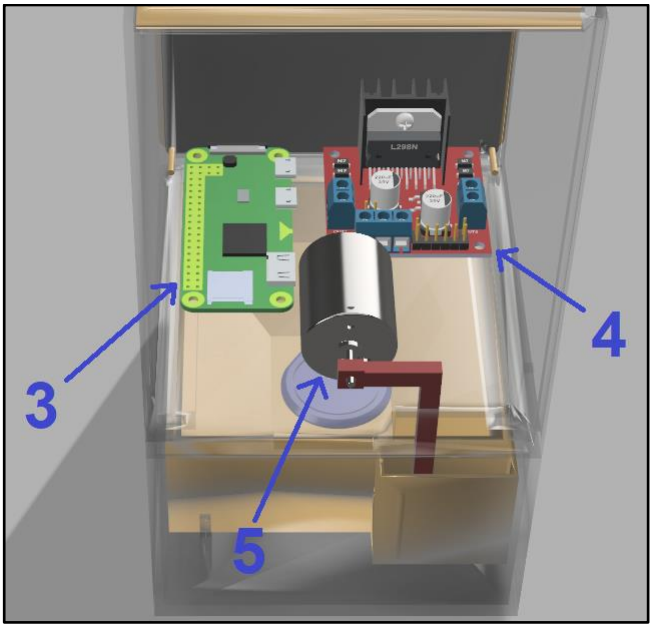

Figure 4.68 Mechanism

3. Raspberry Pi sends signal to L298N Motor Driver to control motor
4. L298N controls the Motor to move.
5. Motor moves the lock part to unlock the key slot.

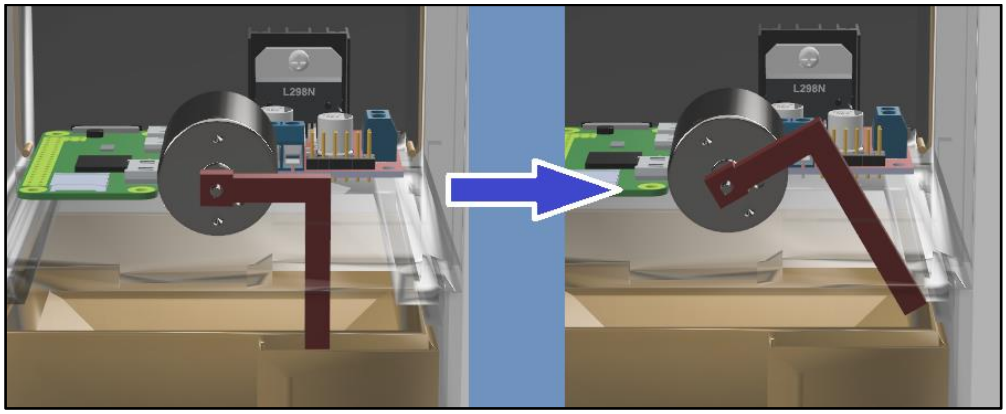

Figure 4.69 Unlocking

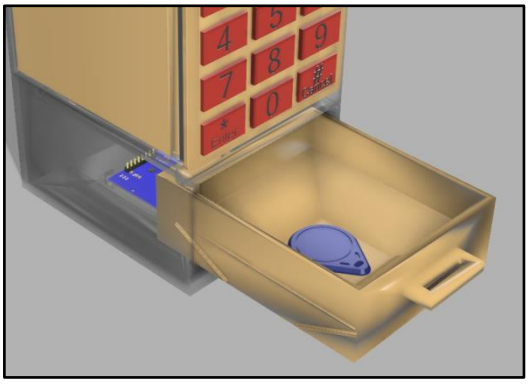

Figure 4.70 Get the key

6. User pulls out the slot and removes the key

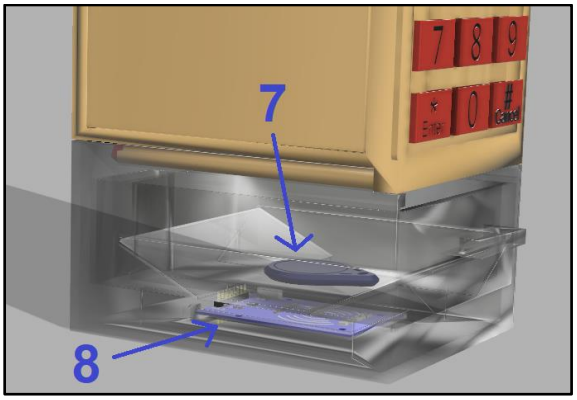

Figure 4.71 Return the key

7. After unlocking the room, the user returns the key and pushes the key slot back into place.
8. RFID Reader detects the RFID tag
9. Motor move the lock part to lock the Kiosk Key Box

4.3.2 Kiosk Key Box Implementation

4.3.2.1 Circuit Implementation

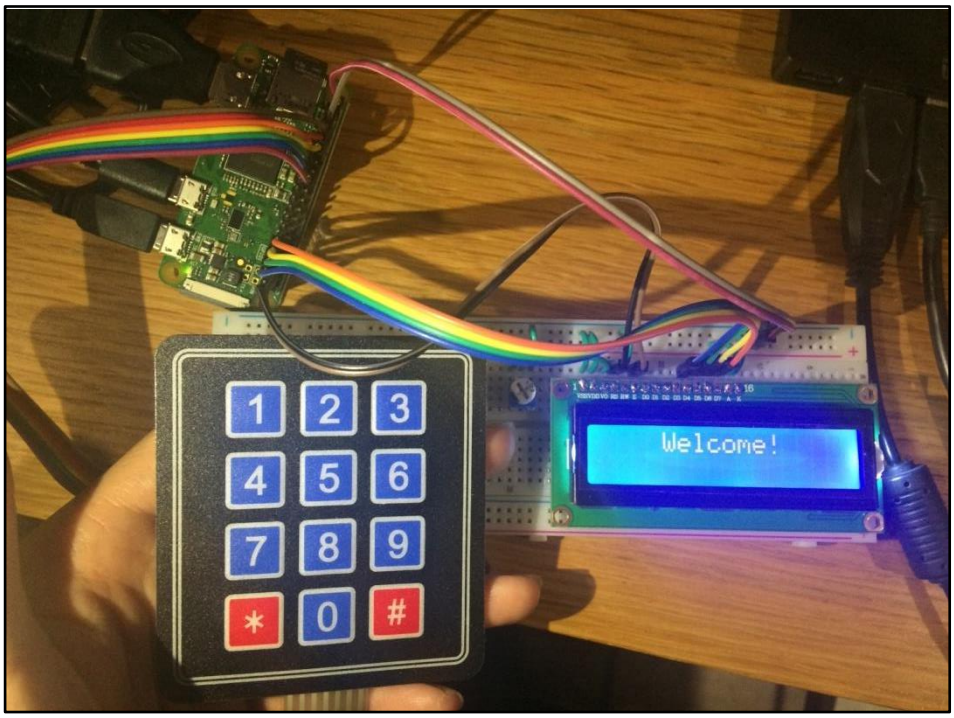

Figure 4.72 3*4 Keypad and LCD 1602 Monitor Implementation

Figure 4.72 shows the implementation of 3*4 keypad and 1602 LCD Monitor with Raspberry Pi Zero W. The keypad can send the six-digit code to the monitor and check if the code is correct or not. The user can press "*" key to reenter the code.

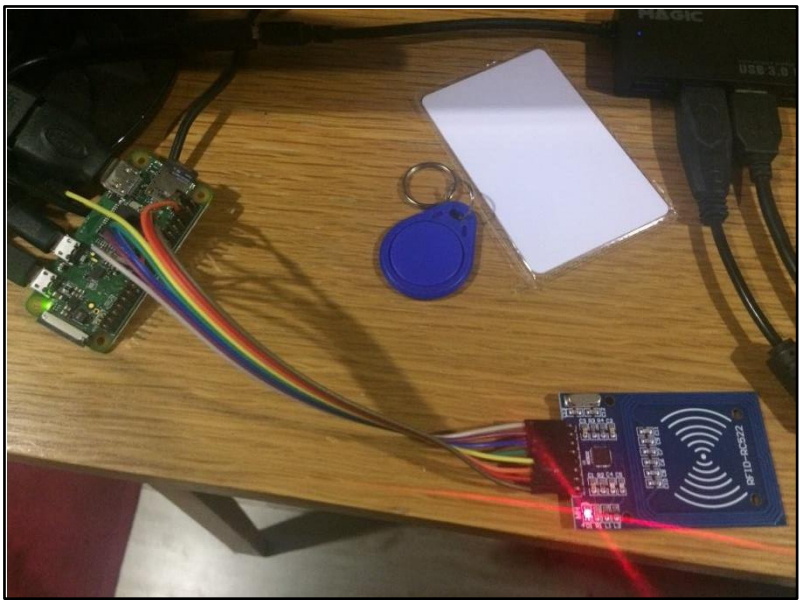

Figure 4.73 MFRC522 RFID Reader Implementation

Figure 4.73 shows the implementation of MFRC522 RFID Reader with Raspberry Pi Zero W. For the first time we need to write the data into new RFID tag. After that, each RFID tag will be specific for each room. The MFRC522 RFID Reader is allowed to read the data in each RFID tag and get the RoomID from that tag.

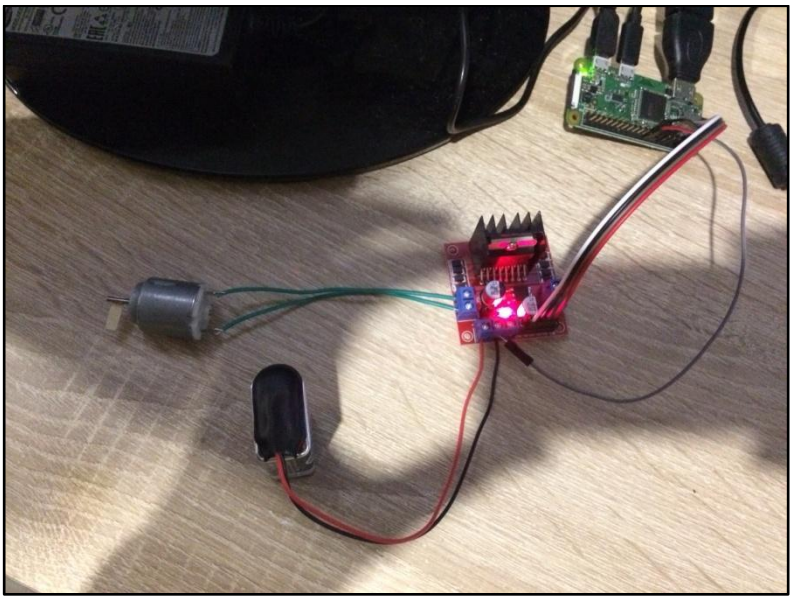

Figure 4.74 L298N and DC Motor Implementation

Figure 4.74 shows the implementation of L298N Motor Driver with Raspberry Pi Zero W. 9V battery and DC Motor is connected to L298N. DC Motor is able to spin forward and backward.

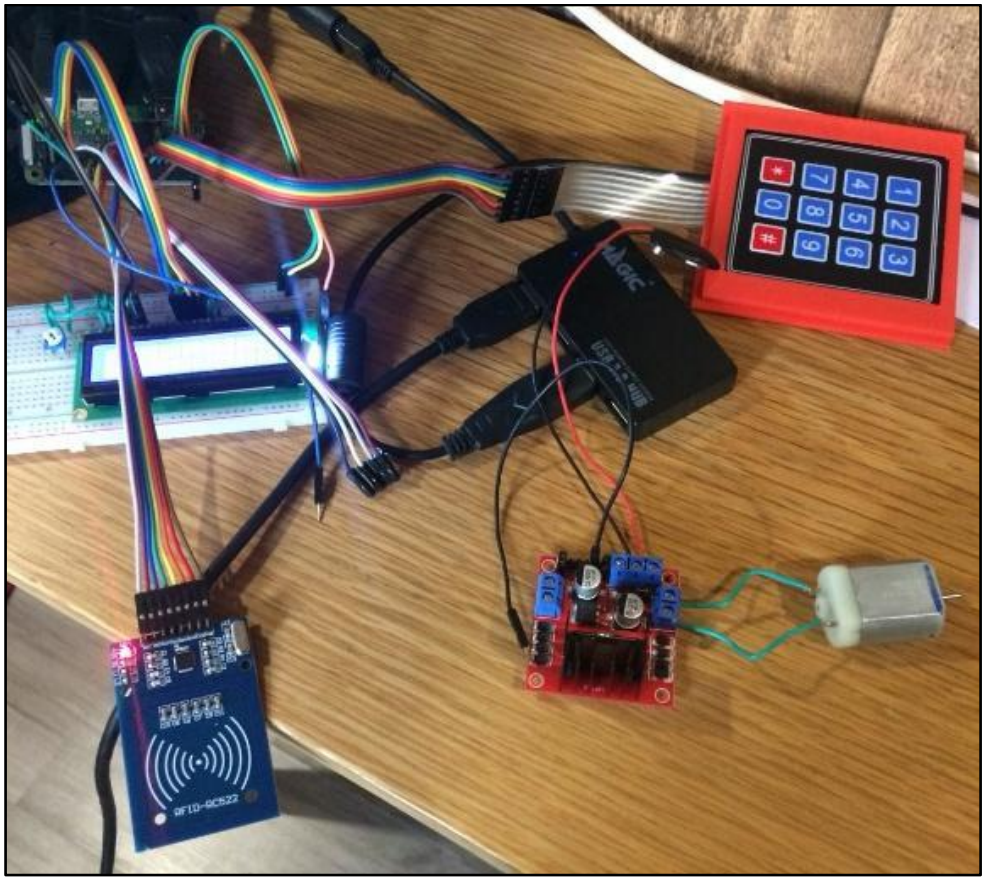

Figure 4.75 Overall Circuit Implementation

The overall implementation of all components with Raspberry Pi Zero W is shown in the picture above.

4.3.2.2 Key Box Container Implementation

The Kiosk Key Box Prototype is created by using a 3D printer to print each part of it. The implemented Key Box Prototype is shown below.

Figure 4.76 Implemented Kiosk Key Box

Figure 4.76 show the appearance of the Kiosk Key Box that is implemented with components and able to work properly.

Figure 4.77 Inside Kiosk Key Box Implemented

Figure 4.78 Inside Kiosk Key Box Implemented (Side)

Figure 4.78 show the components inside Kiosk Key Box that is connected together.

4.3.2.3 Use Flow of Kiosk Key Box

Figure 4.79 Kiosk Key Box Start Running

Figure 4.79 above shows the situation when we start running the program. The key box will wait for user to insert the pin as on the monitor shown “Welcome! Insert PIN”.

Figure 4.80 Kiosk Key Box Insert PIN

The monitor on Figure 4.80 show the value “32596_”, this is the insertion of the PIN which is contain of 6 digits. When the PIN is inserted, the Kiosk Key Box will push the data to the API and get response that the PIN is correct or not. If the PIN is wrong, the monitor will show error message and ask the user to insert PIN again. If the users insert wrong PIN more than 3 time, the Keybox will be locked for five minutes. If the PIN is correct, the motor will open the lock.

Figure 4.81 Kiosk Key Box Getting the key

When the lock is opened, the monitor shows “Unlocked!”. The key slot be pulled out so the user can get the Key. If the key is removed from the box, the Key Box will continue counting the time until the Key is returned.

Figure 4.82 Kiosk Key Box Return the key

When the sensor finds that the key is returned, the motor will show the message “Key Returned” and lock the Key Slot. After the lock is successful, the Kiosk Key Box will push the data into API about the usage statistics.

Figure 4.83 Kiosk Key Box Finished

Once the data has been pushed to the API. The Kiosk Key Box will stop and will be run again on the next key pickup time which is triggered by the API.

4.4 API Function Design

The API implements each use-case in the domain via RESTful approach. Each module will provide http request method which are mostly based on the CRUD (create, read, update, delete) concept. Overall, the API will be provided four main functions which are user, room, reservation, recurring, and pin.

4.4.1 File structure

The API is divided into three component which are controller, service and model. Each component has their own job as described below:

- **Controller**: Control the data which are going to be used inside the service.
- **Service**: The interface for calling model which is the interface of database.
- **Model**: The interface of Object-relational mapping (ORM) configuration

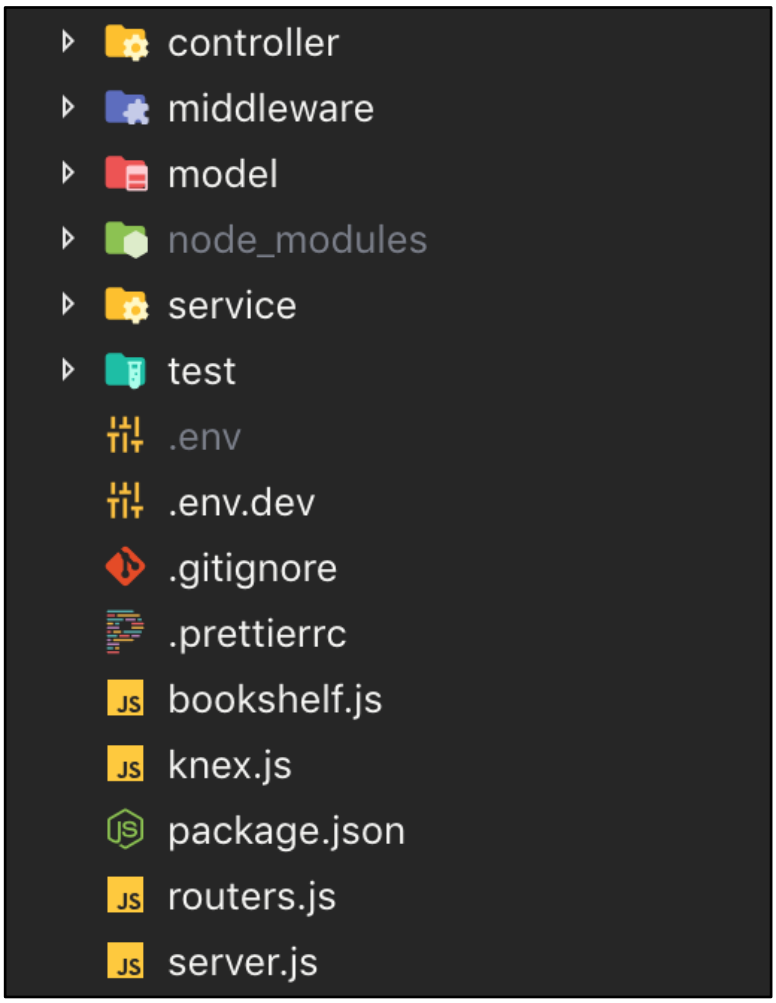

Figure 4.84 File Structure of System

As shown in figure 4.84, the API as installed on the server has several files and folders. The table 4.1 the information about each file and, folder and its function.

Table 4.1 Summary of Each File and Folder

File / Folder name	File inside	Function
controller	ContactController.js EquipmentController.js RecurringController.js ReservationController.js RoomController.js RoomUseController.js SectionController.js UserController.js	Control the data which are going to be used inside the service.
middleware	auth.js royterAuthen.js	For authentication and integration with LDAP.
model	index.js	The interface and ORM configuration.
node_modules	modules of each nodejs library	Module of node library.
service	ContactService.js EquipmentService.js RecurringService.js ReservationServicejs RoomService.js RoomUseService.js SectionServicejs UserService.js	The interface for calling model which is the interface of database ORM
test	Reservation.test.js	Test script for testing controller / service
.env	-	Store environment variable for this project
.env.dev	-	Environment variable for this project (development environment)
.gitignore		The git config file for ignore upload file
.prettierrc		The prettier config for formatting code
bookshelf.js		Import bookshelf library wrapper
knex.js		Import knex library wrapper
package.js		The json file using for npm or yarn to install library into node_modules.
router.js		The router of each domain
server.js		The root file which using for startup server.
yarn.lock		The lock version file of yarn package.

4.4.2 Summary of API endpoints

This section shows the summary of all API collections and endpoints. The information is shown in table 4.2

Table 4.2 API Function Summary

Collection	**Method**	**EndPoint**	**Name**
User	GET	/user	Get users (Get all / get by specific field)
	POST	/user	Create user
	PUT	/user	Update user
	DELETE	/user	Delete user
Room	GET	/room	Get rooms (Get all / get by specific field)
	POST	/room	Create room
	PUT	/room	Update room
	DELETE	/room	Delete room
Reservation	GET	/reservation	Get reservation (Get all / get by specific field)
	PUT	/reservation	Get available reservation
	POST	/reservation/avaiable	Create reservation
	PUT	/reservation	Update reservation
	DELETE	/reservation	Delete reservation
Recurring	GET	/recurring	Get recurring (Get all / get by specific field)
	POST	/recurring	Create recurring
	DELETE	/recurring	Delete recurring
Room use	GET	/pin/validate	Validate pin for keybox

(Keybox pin)	PUT	/pin/status	Update usage of pin
Contact (Mail issue)	GET	/contact	Get issue (Get all / get by specific field)
	POST	/contact	Create issue
	POST	/contact/reply	Reply issue
	DELETE	/contact	Delete user
Equipment	GET	/equipment	Get Equipment (Get all / get by specific field)
Section	GET	/section	Get section (Get all / get by specific field)
Authentication	POST	/auth	Get authentication

4.5 Discussing the Various Alternatives and Considerations

The system architecture is the first thing that we are thinking about. We decided to make all of component isolate from each other. The main reason is we want to make the system that will able to scale. We separated the component to

- Frontend component
- Backend API component
- Database component
- Keybox component

Along the way, we have tried to design three difference type architecture. The section below will describe the difference of them.

4.5.1 The First Design: Docker Container

At first, we design to use container software instead of virtual machine. The container separates each service by each container and share the same operating system and infrastructure. This make us can avoid of infrastructure configuration by each machine. Moreover, it reduces IT management resource, size of snapshot, code to transfer, migrate, upload workloads, also increase spinning up apps.

However, we decided to not use this decided because it need a little high spec of server. From docker specification, it requires 2.00 GB of RAM, 3.00 GB of available disk space and Linux kernel version 3.10 or higher, which is higher than the server that we have from department.

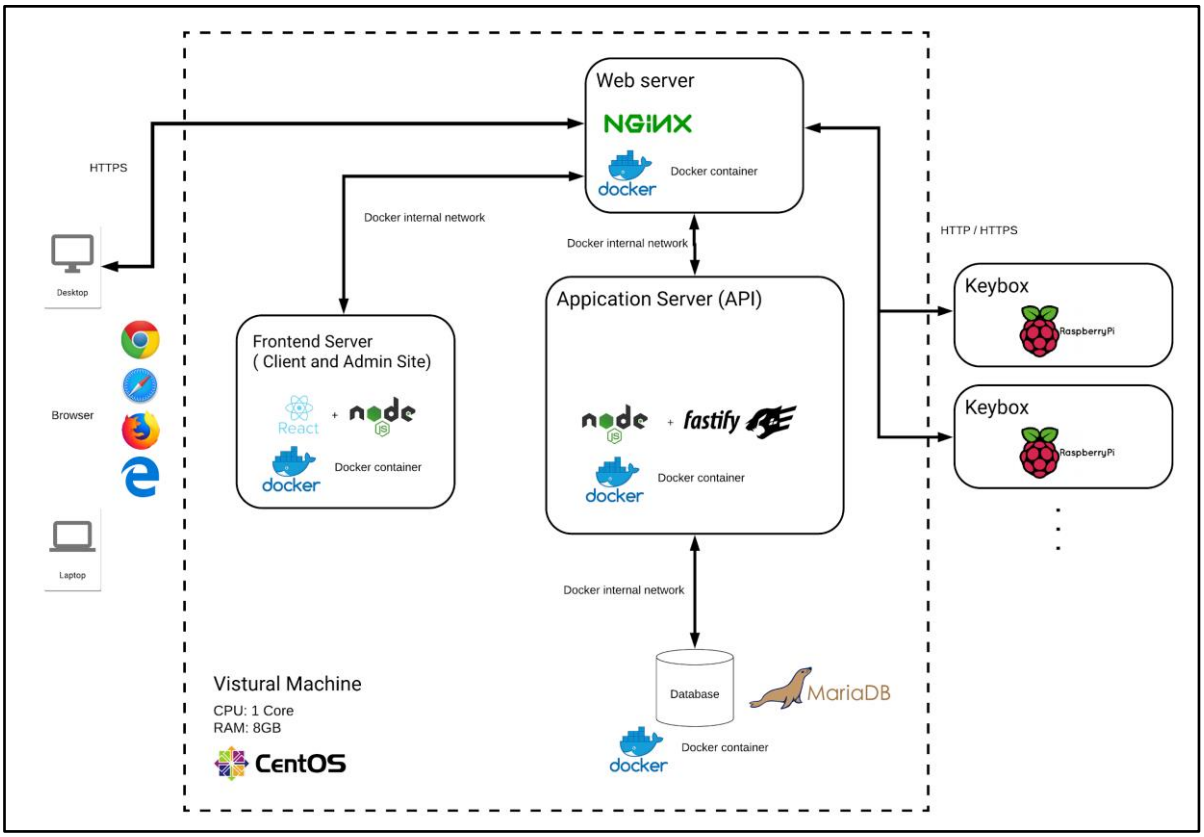

Figure 4.85 System Architecture Design Diagram of First Design

4.5.2 The Second Design: Single Server

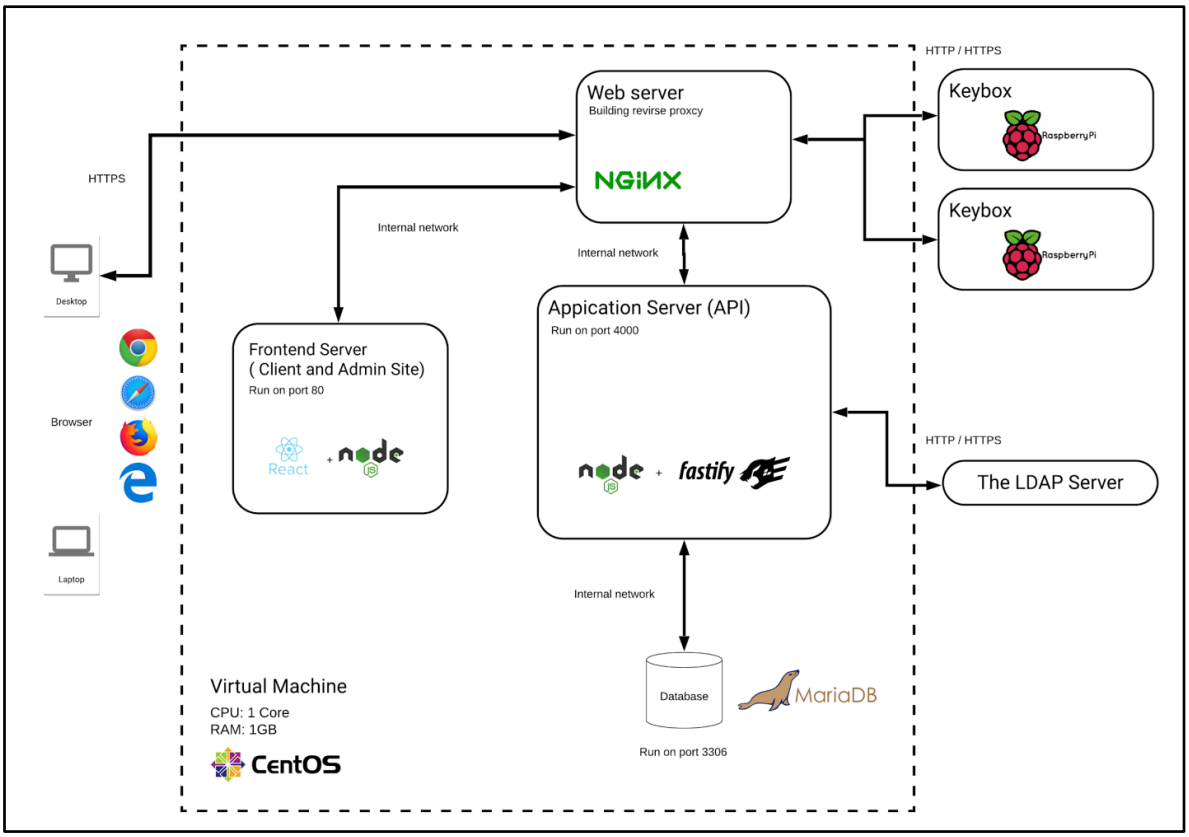

Figure 4.86 System Architecture Design Diagram for Second Design

In this design, we decided to store all of service in single server. Each service will serve and communicate by localhost:ports. It seems to solve our limitation of server spec; however, our server can pass only minimum requirement of each service which may risk to be down due to high demand even we choosing to use fastest Nodejs library.

4.5.3 The Current Design: Cloud Service

From limitation of our server, we decided to use cloud service to serve component except the API because the API need the KMUTT private network in order to authenticate with LDAP server. Therefore, the frontend and database component we choose to use cloud service which are:

- Frontend: Netlify
- Database: Amazon RDS

This make us avoid about the performance of all service. Also, this architect is easy to move or migrate. For example, we can move frontend from netlify to own server without change to api. Moreover, we decided to use Sendgrid to be the email service for our system.

Chapter 5
Discussion and Conclusions

5.1 Task and Progression

The progress of each task is listed below in Table 5.1.

Table 5.1 Task Progression

Task No.	Task name	Status		
		Done	**In Progress**	**Not Started**
1	Web Application Implementation			
1.1	User Interface Implementation			
1.2	Use Flow Implementation			
1.3	Connect to API			
2	Kiosk Key Box Prototype Implementation			
2.1	Equipment Preparation			
2.2	Circuit Implementation			
2.3	Create 3D Printed Model			
2.4	Connect to API			
3	Database Implementation			
3.1	Create Database			
3.2	Connect to API			

4	API Implementation			
4.1	Write API Test script (Unit test)			
4.2	Write API function			
4.3	Implement with database			
4.4	Refactor and Validation			
4.5	Functional test			
4.6	Integration test			
5	Test and Fix bug			
5.1	Test the system			
5.2	Fix Bug			
6	Login and Authentication Integration			
6.1	Building Login function			
6.2	Build up authentication			
6.3	Integration and test			
7	Deployment			
7.1	Production deploy			

5.2 Problems, Issues and Solutions

We had some problems in the progress of our project. which will be discussed below along with the solutions that we used.

We were stuck on some problems in our code to implement the API integrating with the other systems. Most of the software infrastructure we used was new knowledge that we want to try and that caused many problems to happen. So, we've had to spend some time to learn how to use each software component.

The Kiosk Key Box is the hardest part for us, because we only know about basic circuits and sensors. So, we needed to put more time and effort into it to implement it to work properly.

Also, this is the first time we have ever used 3D printing. So, it took us some time to learn how to design a 3D model in software. So, we have to take courses on the internet to learn about designing a 3D model. Moreover, when we got the 3D model from the software, we have been to CPE Department to use 3D printer which is very hard for beginner. So, we have asked the staff in the hardware room to teach us how to use 3D printer. Finally, we can use the printer by ourselves, but not all the printers work well. So, it took us several attempts to finish the 3D printing.

The last problem is about the deployment. It seems like the spec is a little bit low. It makes us worry that the software will down while it has to handle many requests. Therefore, we decided to change our architect again by split out some component, frontend and database, into cloud service. This decision makes our project able to handle more request without any problem.

5.3 Changes from Original Design

First of all, we remove Docker from our project with two reasons. We concluded that this was over engineering, and also the server we had available would not support Docker. We also change some part of user interface to make it more friendly. Also, for the Kiosk Key Box prototype, we plan to change some part of the model make it smaller and easier to use.

Therefore, the system architecture has change. In the original design, all of services were installed in one server. We decided to separate the database to use cloud database. This change make the API will stand alone in the server.

5.4 Outcome Summarizing

For web Application, the web application is implemented with all features as planned. Web application can get the value insert by user and communicate with API. The task that may need optimizing is the query time for searching a room for booking due to the number of reservations will increase overtime. Features to ban user and generate reports is not implemented yet.

Kiosk Key Box is implemented as a prototype. The prototype is made of plastic from 3D printing. The components inside Kiosk Key Box is also implemented. The keybox can send the data and get response from API. The necessary thing to improve is the strength of DC Motor. If the user pulls the Key Slot while it is locked, the DC motor may be broken.

API is all implemented and deploy on carto server which located inside department. The API provide all of used function for all system component. All of business logic are work, but it also has many areas to implement such as algorithm, error handling and so on.

The database is implemented and deploy on Amazon Relational Database Service. It can communicate with API to read, add, edit or delete data. Therefore, it may have cost to pay for database in the future if the number of users is going up. However, it is able to move to any private server and run by MariaDB.

5.5 Future Suggestion

The feature to ban user and generate report for admin should be implemented to optimize the usage of the room and prevent the user from making too much mistake.

The server of an API and database should be improved in specification or move to another server which provide better performance due to the database and system usage when deploy will need a lot of resource to run the system.

From asking the person who will be the system administrator we have got many suggestions to improve in our project.

Firstly, she suggests that we should add the feature to calculate and summary the usage of each room in a period. For example, admin can go to the website choose the room and filter it with period. Then, the system should show number of hours use in that period.

Secondly, we should add the feature to book a room by insert start time with the period of the class without needing to fill the end time. For example, the standard is fifty minutes per period. Class start at 8.30 and have three period. So, the system will auto calculate the time and book on 8.30 to 11.00.

Thirdly, we should improve the user interface to be user friendlier. The booking timetable should change to calendar to easily see when the room has been booked. The room list should show less information to make it smaller and able to see more rooms in one page.

Fourthly, the type of the room should be added to room information. It will be easier for room management by admin and easier for the user to get the room that they prefer.

Finally, to get feedback and suggestion, the user should be able to review and comment about each room usage. For example, they can comment that the microphone can’t be use, the air conditioner is not working. Also, with the ability for admin to ban a user if the user doing inappropriate action like make room dirty, leave without turnoff air conditioner, etc.

References

1. Reserving Study Rooms Online: Room Reservation, http://researchguides.gonzaga.edu/studyrooms, Nov 3, 2017
2. Ecobook, Room & Resource Booking, http://www.facilitiesbooking.com/Products/ecobook-Room-Resource-Booking
3. Addy Osmani, Getting started with Progressive Web Apps, https://addyosmani.com/blog/getting-started-with-progressive-web-apps/, Dec 23, 2015
4. React Data Flow Diagram, https://www.thitiblog.com/wp-content/uploads/2018/02/maxresdefault.jpg
5. What is RFID? https://www.epc-rfid.info/rfid
6. Skella and Mcollina, Fastify, https://github.com/fastify/fastify
7. NodeJS, https://en.wikipedia.org/wiki/Node.js
8. Pete Brey, Containers vs. Virtual Machines (VMs): What's the Difference, https://blog.netapp.com/blogs/containers-vs-vms/, March 16, 2018
9. Representational state transfer, https://en.wikipedia.org/wiki/Representational_state_transfer#Architectural_properties
10. JavaScript, https://developer.mozilla.org/bm/docs/Web/JavaScript
11. JavaScript, https://en.wikipedia.org/wiki/JavaScript
12. JavaScript technologies overview, https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview, Mar 12, 2018
13. Introduction to the DOM, https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
14. JavaScript HTML DOM, https://www.w3schools.com/js/js_htmldom.asp
15. Jonathan Robie, Texcel Research, What is the Document Object Model?, https://www.w3.org/TR/WD-DOM/introduction.html
16. Library vs. Framework, https://www.programcreek.com/2011/09/what-is-the-difference-between-a-java-library-and-a-framework/
17. Alex, How much power does Pi Zero W use?, https://raspi.tv/2017/how-much-power-does-pi-zero-w-use, Mar 1, 2017
18. Advantages and Disadvantages of RDBMS, http://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-RDBMS.html
19. Advantages & Disadvantages of RFID, https://www.techwalla.com/articles/advantages-disadvantages-of-rfid
20. Web Component, https://developer.mozilla.org/en-US/docs/Web/Web_Components, Sep 6, 2018
21. Pavan Podila, Intro to the React Framework, https://code.tutsplus.com/tutorials/intro-to-the-react-framework--net-35660, Nov 15, 2013
22. Sequelize, http://docs.sequelizejs.com/
23. API Management, What are the advantages of a REST API, https://www.chakray.com/en/advantages-of-rest-api, Feb 9, 2013
24. Keypad Tutorial, https://playground.arduino.cc/Main/KeypadTutorial, Sep 4, 2013
25. Interface 16x2 Alphanumeric LCD And4x4 Matrix Keypad with Raspberry Pi3, https://www.instructables.com/id/Interface-16x2-Alphanumeric-LCD-And4x4-Matrix-

Keyp/
26. How to setup a Raspberry Pi RFID RC522 Chip, https://pimylifeup.com/raspberry-pi-rfid-rc522/, Oct 3, 2019

Appendix A

API Endpoint Detailed Description

API Endpoint Detailed Description

This section will acknowledge you about using API including passing parameter and usage. All of response coming in json format

1. Users collection: [/users]

This collection will provide all of user business logic which has some several function which contain all CRUD (create, read, update, delete) method. The information below show the method and request parameter.

GET [/users]: Get all users in system

- Request:
  - Parameter:
  - Header:
    - Content-type: application/json
- Response:
  - Code: 200
  - Header:
    - Content-type: application/json
  - Body
    - <Json object array>

GET [/users/? <params>]: find user by search parameter (select attribuild as show in Parameter)

- Request:
  - Parameter:
    - UserID: String
    - FirstName: String
    - LastName: String
    - isBan: Boolean
  - Header:
    - Content-type: application/json
- Response:
  - Code: 200
  - Header:
    - Content-type: application/json
  - Body
    - <Json object array>

POST [/users]: Create new user

- Request:
  - Parameter:
  - Header:
    - Content-type: application/json
  - Body:
    - UserID: String
    - FirstName: String
    - LastName: String
- Response

        - Header:
            - Content-type: application/json
        - Body
            - <Json object (Same as create information)>

PUT [/users]: Update existing user

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
    - Body:
        - UserID: String
        - FirstName: String
        - LastName: String
- Response
    - Header:
        - Content-type: application/json
    - Body
        - <Json object >, true mean success

DELETE [/users]: Delete existing user (select only one body field)

- Request:
    - Parameter:
        - UserID: String
    - Header:
        - Content-type: application/json
- Response
    - Header:
        - Content-type: application/json
    - Body
        - {delete: boolean}, true for success

2. Rooms collection: [/rooms]

This collection will provide all of user business logic which has some several function which contain all CRUD (create, read, update, delete) method. The information below show the method and request parameter.

GET [/rooms]: Get all rooms in system

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

GET [/rooms? <params>]: Get rooms which specific parameter

- Request:
    - Parameter:
        - RoomID: String
        - Building: String
        - Floor: Int
        - PeopleCapacity: Int
        - StartTime: Time
        - EndTime: Time
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

POST [/rooms]: Create new room

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
    - Body:
        - RoomID: String
        - Building: String
        - Floor: String
        - RoomNumber: Int
        - PeopleCapacity: Int
        - StartTime: Time
        - EndTime: Time
        - HasTeacherComputers: Boolean
        - HasStudentsComputers: Boolean
        - HasProjector: Boolean
        - HasWriteboard: Boolean
        - HasVisualizer: Boolean
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object>

PUT [/rooms]: Update room information

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
    - Body:
        - RoomID: String
        - Building: String
        - Floor: String

            - RoomNumber: Int
            - PeopleCapacity: Int
            - StartTime: Time
            - EndTime: Time
            - HasTeacherComputers: Boolean
            - HasStudentsComputers: Boolean
            - HasProjector: Boolean
            - HasWriteboard: Boolean
            - HasVisualizer: Boolean
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object>, true mean sucess

Delete [/rooms]: Delete room out from database

- Request:
    - Parameter:
        - RoomID: String
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object>, true mean success

3. Recurring collection: [/recurring]

This collection will provide all of user business logic which has some several function which contain all CRD (create, read, delete) method. The information below show the method and request parameter.

GET [/recurring]: Get all recurring reservations in system

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

GET [/recurring? <params>]: Get recurring reservation which specific parameter

- Request:
    - Parameter:
        - BookingID: String

        - Header:
            - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

POST [/recurring]: Create new recurring reservation

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
    - Body:
        - RoomID: String
        - UserID: String
        - Title: String
        - Term: String
        - StartDate: Date
        - EndDate: Date
        - Day: Int
        - StartTime: Time
        - EndTime: Time
        - Sections: String
        - Year: Int
        - DateBooked: DateTime
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

4. Reservation collection: [/reservations]

This collection will provide all of user business logic which has some several function which contain all CRUD (create, read, update, delete) method. The information below show the method and request parameter.

GET [/reservations]: Get all reservations in system

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

GET [/reservations? <params>]: Get reservations which specific parameter

- Request:
  - Parameter:
    - UserID: String
  - Header:
    - Content-type: application/json
- Response:
  - Code: 200
  - Header:
    - Content-type: application/json
  - Body
    - <Json object array>

POST [/reservations/avaiable]: Find available reservations

- Request:
  - Parameter:
  - Header:
    - Content-type: application/json
  - Body:
    - room: < JSON of room filed >
    - equipment: <JSON of equipment>
    - reservation:<JSON of reservation>
- Response:
  - Code: 200
  - Header:
    - Content-type: application/json
  - Body
    - <Json object array>

POST [/reservations]: Create new reservation

- Request:
  - Parameter:
  - Header:
    - Content-type: application/json
  - Body:
    - RoomID: String
    - UserID: String
    - Title: String
    - Day: Int
    - Date: Date
    - StartTime: Time
    - EndTime: Time
- Response:
  - Code: 200
  - Header:
    - Content-type: application/json
  - Body
    - <Json object array>

PUT [/reservations]: Update exist reservation

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
    - Body:
        - RoomID: String
        - UserID: String
        - Title: String
        - Day: Int
        - Date: Date
        - StartTime: Time
        - EndTime: Time
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object>

Delete [/reservations]: Delete exist reservation

- Request:
    - Parameter:
        - BookingID: String
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object>

5. Keybox pin collection: [/pin]

This collection will provide all of user business logic which has some several function which necessary for validate room booking. The information below show the method and request parameter.

GET [/pin/check? <params>]: Validate pin for keybox

- Request:
    - Parameter:
        - Pin: String
        - RoomID: Int
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - {Open: Boolean}, true for valid, vise visa

PUT[/pin/validate]: Send status to server for making room usage such as successful used, ban status that user not return the key

- Request:
  - Parameter:
  - Header:
    - Content-type: application/json
  - Body
    - Pin: String
    - KeyReturn: Boolean
- Response:
  - Code: 200
  - Header:
    - Content-type: application/json
  - Body
    - {Status: Boolean, message: String}, true for complete (not have any problem), vise visa

6. Contact collection: [/contact]

This collection will provide all of contact business logic which has some several function which contain all CRUD (create, read, update, delete) method. The information below show the method and request parameter.

GET [/contact]: Get all issue system

- Request:
  - Parameter:
  - Header:
    - Content-type: application/json
- Response:
  - Code: 200
  - Header:
    - Content-type: application/json
  - Body
    - <Json object array>

GET [/contact? <params>]: Get issue which specific parameter

- Request:
  - Parameter:
    - ContactID: String
  - Header:
    - Content-type: application/json
- Response:
  - Code: 200
  - Header:
    - Content-type: application/json
  - Body
    - <Json object array>

POST [/contact]: Create new issue

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
    - Body:
        - UserID: String
        - EmailAddress: String
        - Title: String
        - Detail: String
        - DateTime: Date
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object>

POST [/contact/reply]: Reply issue

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
    - Body:
        - EmailAddress: String
        - Title: String
        - Detail: String
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object>, true mean sucess

Delete [/contact]: Delete issue

- Request:
    - Parameter:
        - ContactID: Int
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object>, true mean success

7. Equipment collection: [/equipment]

This collection will provide all of equipment business logic. The information below show the method and request parameter.

GET [/equipment]: Get all equipment in each room

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

GET [/equipment? <params>]: Get equipment which specific parameter

- Request:
    - Parameter:
        - HasTeacherComputers: Boolean
        - HasStudentComputers: Boolean
        - HasProjector: Boolean
        - HasAirConditioner: Boolean
        - HasWhiteboard: Boolean
        - HasVisualizer: Boolean
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

8. Section collection: [/section]

This collection will provide all of section business logic. The information below show the method and request parameter.

GET [/section]: Get all section in recurring

- Request:
    - Parameter:
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>

GET [/section? <params>]: Get section which specific parameter

- Request:
    - Parameter:
        - Section: String

        - Year: Int
        - Program: Int
    - Header:
        - Content-type: application/json
- Response:
    - Code: 200
    - Header:
        - Content-type: application/json
    - Body
        - <Json object array>