A Mini Project Report on

PARKING MANAGEMENT SYSTEM

Submitted to

Jawaharlal Nehru Technological University, Hyderabad

in partial fulfillment of requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

BUGGAVEETI MITRA (16BD1A052L)
BANDARU SANDEEP (16BD1A0522)
CHAITANYA POHNERKAR (16BD1A052P)
ASHOK KUMAR BANOTH (16BD1A0524)

Under the esteemed guidance of
Mrs. Ch. Sita Kameswari
Assistant Professor
Department of CSE

Department of Computer Science and Engineering
KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
Approved by AICTE, Affiliated to JNTUH
3-5-1206, Narayanaguda, Hyderabad - 500029

2019 – 2020

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
Approved by AICTE, Affiliated to JNTU, Hyderabad
3-5-1206, Narayanaguda, Hyderabad - 500029.

CERTIFICATE

This is to certify that the project entitled "**PARKING MANAGEMENT SYSTEM"** being submitted by **Mr. BUGGAVEETI MITRA (16BD1A052L), Mr. BANDARU SANDEEP (16BD1A0522), Mr. CHAITANYA POHNERKAR (16BD1A052P), Mr. ASHOK KUMAR BANOTH (16BD1A0524)** students of **Keshav Memorial Institute of Technology, JNTUH** in partial fulfillment of the requirements of the award of the Degree of **Bachelor of Technology in Computer Science and Engineering** as a specialization is a record of bonafide work carried out by them under my guidance and supervision in the academic year 2019 – 2020

GUIDE
Mrs. Ch. Sita Kameswari
Assistant Professor

Dr. S. Padmaja
HoD-CSE

Submitted for the Project Viva Voce examination held on …….

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project report entitled **"PARKING MANAGEMENT SYSTEM"** is done in the partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering affiliated to Jawaharlal Nehru Technological University, Hyderabad. This project has not been submitted anywhere else.

BUGGAVEETI MITRA (16BD1A052L)

BANDARU SANDEEP (16BD1A0522)

CHAITANYA POHNERKAR (16BD1A052P)

ASHOK KUMAR BANOTH (16BD1A0524)

ACKNOWLEDGEMENT

We take this opportunity to thank all the people who have rendered their full support to our project work.

We render our thanks to **Dr. Maheshwar Dutta**, B.E., M Tech., Ph.D., Principal who encouraged us to do the Project.

We are grateful to **Mr. Neil Gogte**, Director for facilitating all the amenities required for carrying out this project.

We express our sincere gratitude to **Mr. S. Nitin**, Director and **Mrs. Deepa Ganu**, Dean Academics for providing an excellent environment in the college.

We are also thankful to **Dr. S. Padmaja**, Head of the Department for providing us with both time and amenities to make this project a success within the given schedule.

We are also thankful to our guide **Mrs. Ch. Sita Kameswari**, for her valuable guidance and encouragement given to us throughout the project work.

We would like to thank the entire CSE Department faculty, who helped us directly and indirectly in the completion of the project.

We sincerely thank our friends and family for their constant motivation during the project work.

BUGGAVEETI MITRA **16BD1A052L**

BANDARU SANDEEP **16BD1A0522**

CHAITANYA POHNERKAR **16BD1A052P**

ASHOK KUMAR BANOTH **16BD1A0524**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Parking Management System is implemented to manage occupancy of parking slots and allow customers to find and reserve available parking place and to inquire on the vacant slots before vehicle arriving at parking. Parking management system is implemented so that user can book the parking slots on the internet before arriving to the parking place. The parking management system will encourage customers to book parking slots in online and make the parking process a hassle-free experience.Manually the customer has to check the available slot and wait until the slot is free or they search for any alternative like any side of roads.

CHAPTER 1

1. INTRODUCTION

Parking Management system is a web-based Application to the events of customers. In this system, customer can register in the application and get user id and password. Administrator will manage all the events adding, deleting and updating the slots. Existing system is very difficult to manage and it is a time-consuming process. More manual work is needed. We have to face high-level risk in maintain the existed system. In order to overcome all these problems, the project is developed from manual procedure to automate.

This project has been developed in order to overcome the difficulties encountered while using the existing system. Parking Management System is developed to support the customers and also the management of parking with a flexible and Hassle-free experience. The system enables the customer to make registrations and reservations in online. Manually the customer has to check the available slot and wait until the slot is free or they search for any alternative like any side of roads. This wills a problem for Traffic. To avoid this problem Parking Management System is useful for both the customers and the management.

In this system all the events are managed by the Admin only. Admin adds, delete and update the information in the application. The user or customer login with his login-id and password then checks for availability of slots. If the customer has selected a particular slot in a particular slot then he/she must register by giving the required details. Upon successful registration the customer can simply go to that parking place and park the vehicle thereby creating a hassle-free experience.

CHAPTER 2

2. BUSINESS PROBLEMS AND SOLUTIONS

2.1 Existing System

The traditional parking system is highly dependent on manual work. The search for a parking space is also a tedious task. When a person needs to park his vehicle, he needs to first visit a parking space and search for an empty spot. Failing which will force the person to move on to the next parking space with no guarantee of finding a free space either. This process can be highly simplified by providing an automated system.

2.2 Client Expectations

To make parking a hassle-free process, the customers need a system which is accessible at all the times and is convenient to use. This automated system will provide these two features and will not consume an individual's time to an extent that his/her day-to-day activities are deviated.

2.3 Proposed Solution

With an online system, people in need of parking can login to the platform and parking spaces area wise are displayed. A person can select a slot at any desired parking space and choose the time of parking too. After doing so, the chosen parking slot is booked and will no longer be available for booking. Once the duration of slot is expired only then the slot is made available again. This ensures consistency and a hassle-free procedure.

2.4 Functional Requirements

For documenting the functional requirements, the set of functionalities supported by the system are to be specified. A function can be specified by identifying the state at which data is to be input to the system, its input data domain, the output domain, and the type of processing to be carried on the input data to obtain the output.

Functional requirements define specific behaviour or function of the application. Following are the functional requirements:

i. Should enter the details like Name, License plate number, Duration.

ii. Should press the BOOK button to confirm booking.

2.5 Non Functional Requirements

A non-functional requirement is the one that specifies criteria that can be used to judge the operation of a system, rather than specific behaviour. Especially these are the constraints that the system must work within. Following are the non functional requirements:

i. Should be available for installation on computer.

ii. Crashing of application should not occur.

iii. Response time for each query sent by the user should be minimum.

iv. Application should be updated properly.

2.6 Software and Hardware Requirements

Operating System	: Windows 7/8/10
Front end	: HTML, CSS, React
Back end	: Node JS, Mongo DB
Framework	: Express
Processor	: Intel i3
Hard disk	: 500 GB or more
RAM	: 4 GB or more

CHAPTER 3

3. LITERATURE SURVEY

3.1 Managing a Parking System

Parking management system is a system that is used to help managing vehicles in parking area in other to avoid congestion and arrange vehicles in an allocated position. The system also helps to track how many vehicles pass through the gate and the duration taken by each, and then it will calculate the amount of money a vehicle should pay when exiting. Vehicle parking system is being used in many congested area or location where there are many meeting point of people like where there are more than one shopping complex near to each other or where there is mega mall or stadium.

3.2 Various available systems

From the research conducted, there is not much direct interaction between the system and the user. Though some parking systems have different procedures, but for a parking system like in Sunway pyramid, the author got to understand that the process is generally as follows. The customer presses the button on the machine; ticket will come out from it and the customer take his ticket and then the gate will open. The customer will now proceed to the available space. The available spaces are known by the green light bulb on top of each available lot. The red light on top of parking lot indicate either the space is being reserved or there is car parked at it. For the process of exiting, if the customer is using “touch and go”, there is a sensor provided which will read the card and if there is enough money the system will deduct the amount of the charges. But if the customer is going to pay cash there are available paying machines which the customer will insert his ticket and the system will read the ticket, estimate the hours spent and calculate the amount to be paid by the customer. Customer should insert the money stated by the system into machine. The system will validate the ticket and assign 15minutes on it which is enough for a person to his vehicle to the nearest exit gate.

3.3 An updated online system

As the involvement of technology in everyday life has been increasing day by day, providing online solutions to parking is no surprise. User can pre-book his parking slot in any desired parking just by using an online parking system. This allows user to choose a parking space and parking slot of his choice without physically being present at the parking location. Online systems are not just limited to web but also have branches in application development. The Online Vehicle Parking Reservation System (OVPRS) is a system that enables customers/drivers to reserve a parking space. It also allows the customers/drivers to view the parking status at kyebando people's park. It was developed because the congestion and collision of the vehicle, the system was developed for Kyebando People's Park located in Kyebando Therefore the project aimed at solving such problems by designing a web based system that will enable the customers/drivers to make a reservation of available parking space at people's park.

CHAPTER 4

4. DESIGN

4.1 Introduction to UML

The UML (Unified Modelling Language) is a way of visualising a software program using a collection of diagrams. Today, UML is accepted by the Object Management Group (OMG) as the standard for modelling software development. UML stands for Unified Modelling Language. UML helped extend the original UML specification to cover a wider portion of software development efforts including agile practices.

- Improved integration between structural models like class diagrams and behaviour models like activity diagrams.

- Added the ability to define a hierarchy and decompose a software system into components and sub-components

4.2 UML Diagrams

4.2.1 Use case Diagram

A cornerstone part of the system is the functional requirements that the system fulfils. Use Case diagrams are used to analyze the system's high-level requirements. These requirements are expressed through different use cases. A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system. They provide a good high level analysis from outside the system. Use case diagrams specify how the system interacts with actors without worrying about the details of how that functionality is implemented. We notice three main components of this UML diagram:

Functional requirements – represented as use cases; a verb describing an action

Actors – they interact with the system; an actor can be a human being, an organisation or an internal or external application

Relationships - between actors and use cases – represented using straight arrows

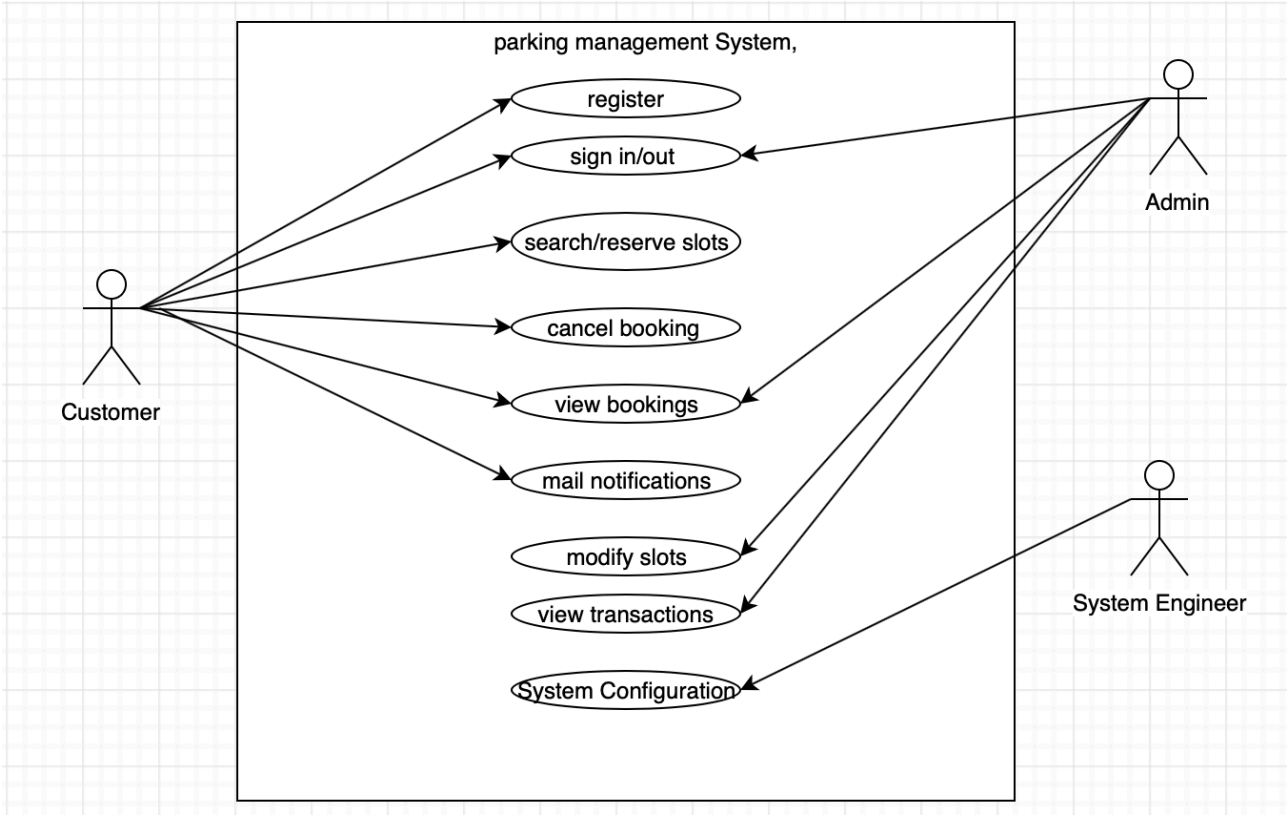


Figure 1: Use case Diagram

4.2.2 Class Diagram

Class UML diagram is the most common diagram type for software documentation. Since most software being created nowadays is still based on the Object-Oriented Programming paradigm, using class diagrams to document the software turns out to be a common-sense solution. This happens because OOP is based on classes and the relations between them.

Class diagrams contain classes, alongside with their attributes (also referred to as data fields) and their behavior (also referred to as member functions). More specifically, each class has 3 fields: the class name at the top, the class attributes right below the name, the class operations/ behavior at the bottom. The relation between different classes (represented by a connecting line), makes up a class diagram.

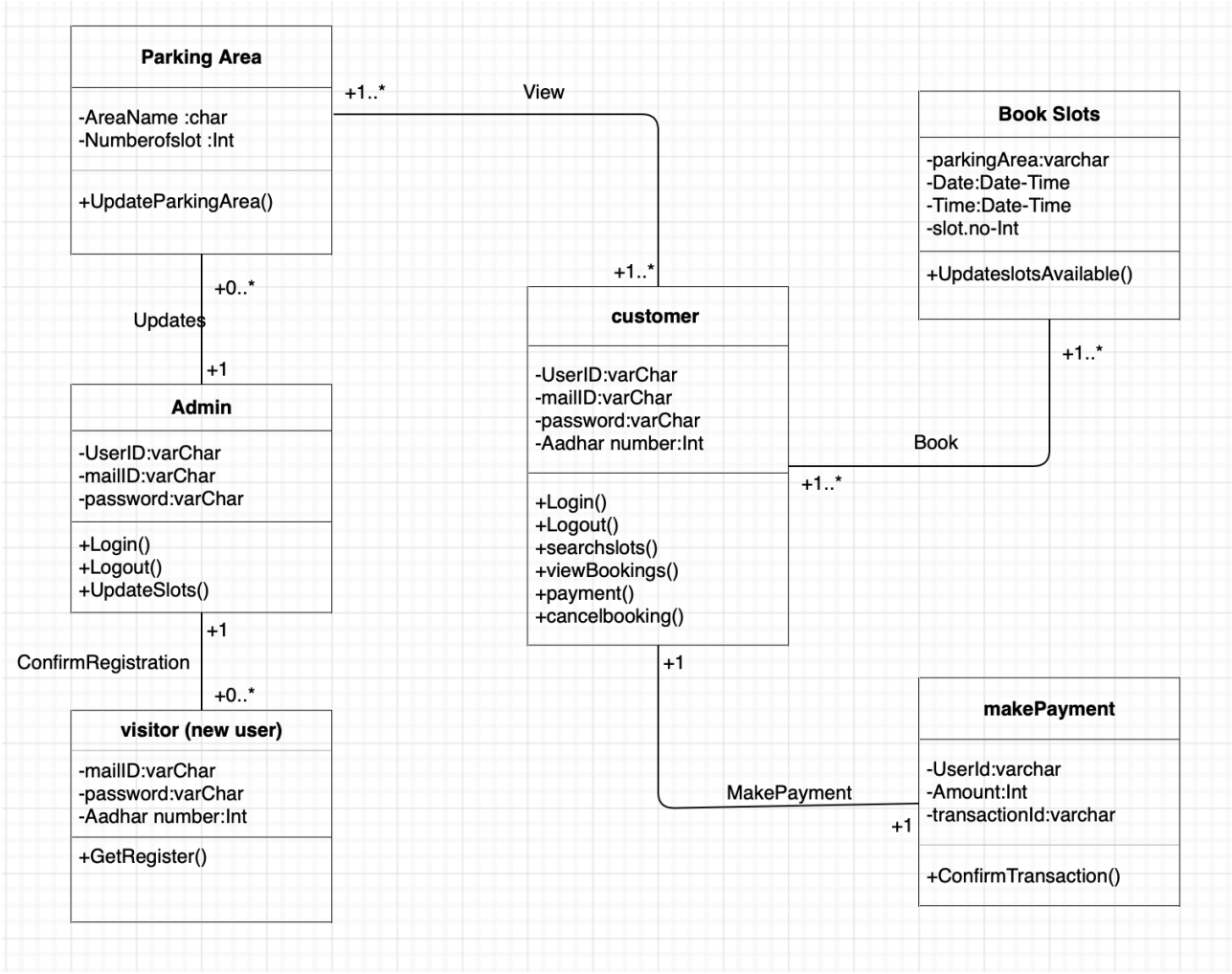


Figure 2: Class Diagram

4.2.3 Architecture Diagram

An architecture diagram is a graphical representation of a set of concepts, that are part of an architecture, including their principles, elements and components.

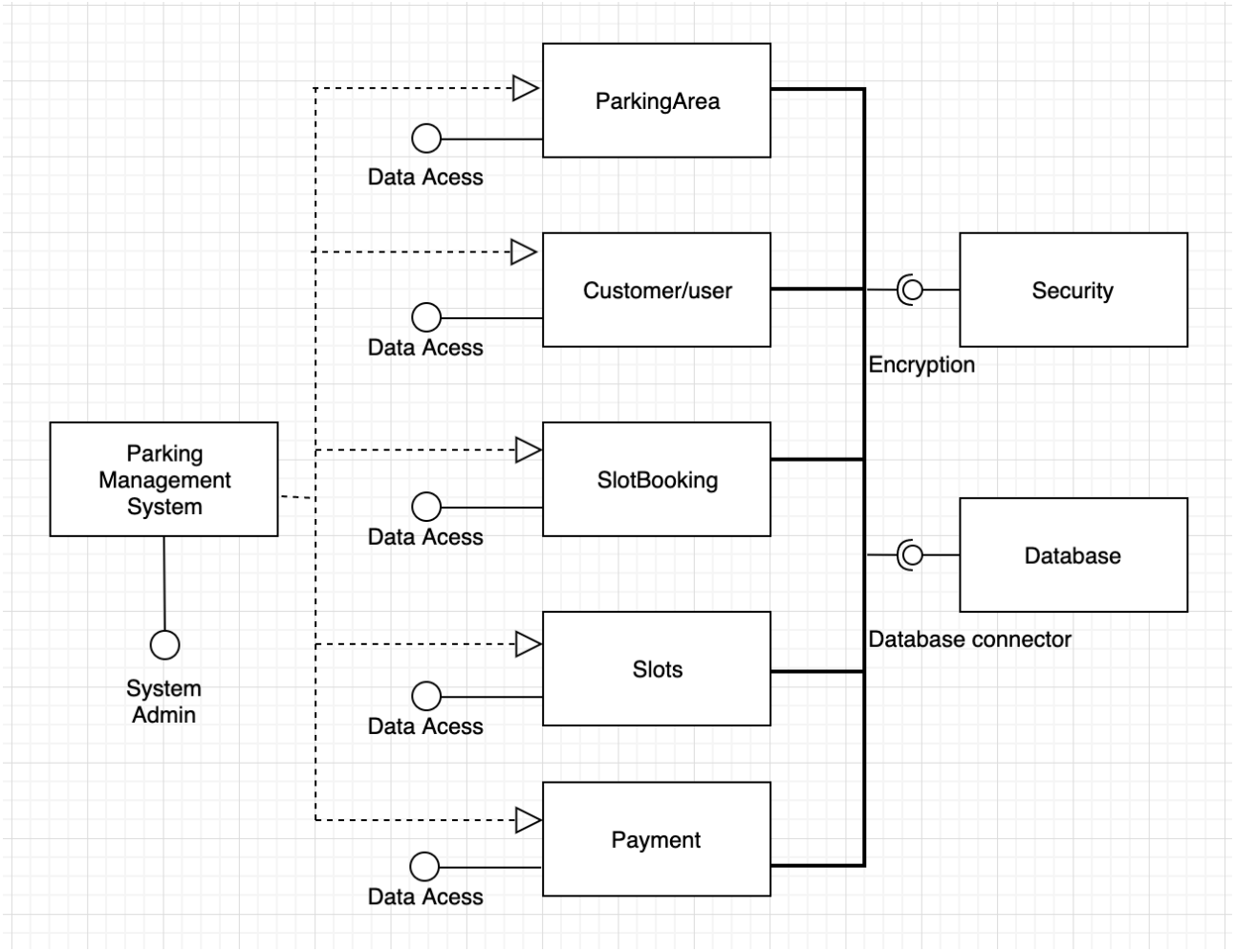


Figure 3: Architecture Diagram

4.2.4 Sequence Diagram

Sequence diagrams are probably the most important UML diagrams among not only the computer science community but also as design-level models for business application development. Lately, they have become popular in depicting business processes, because of their visually self-explanatory nature.

As the name suggests, sequence diagrams describe the sequence of messages and interactions that happen between actors and objects. Actors or objects can be active only when needed or when another object wants to communicate with them. All communication is represented in a chronological manner. To get a better idea, check the example of a UML sequence diagram below.

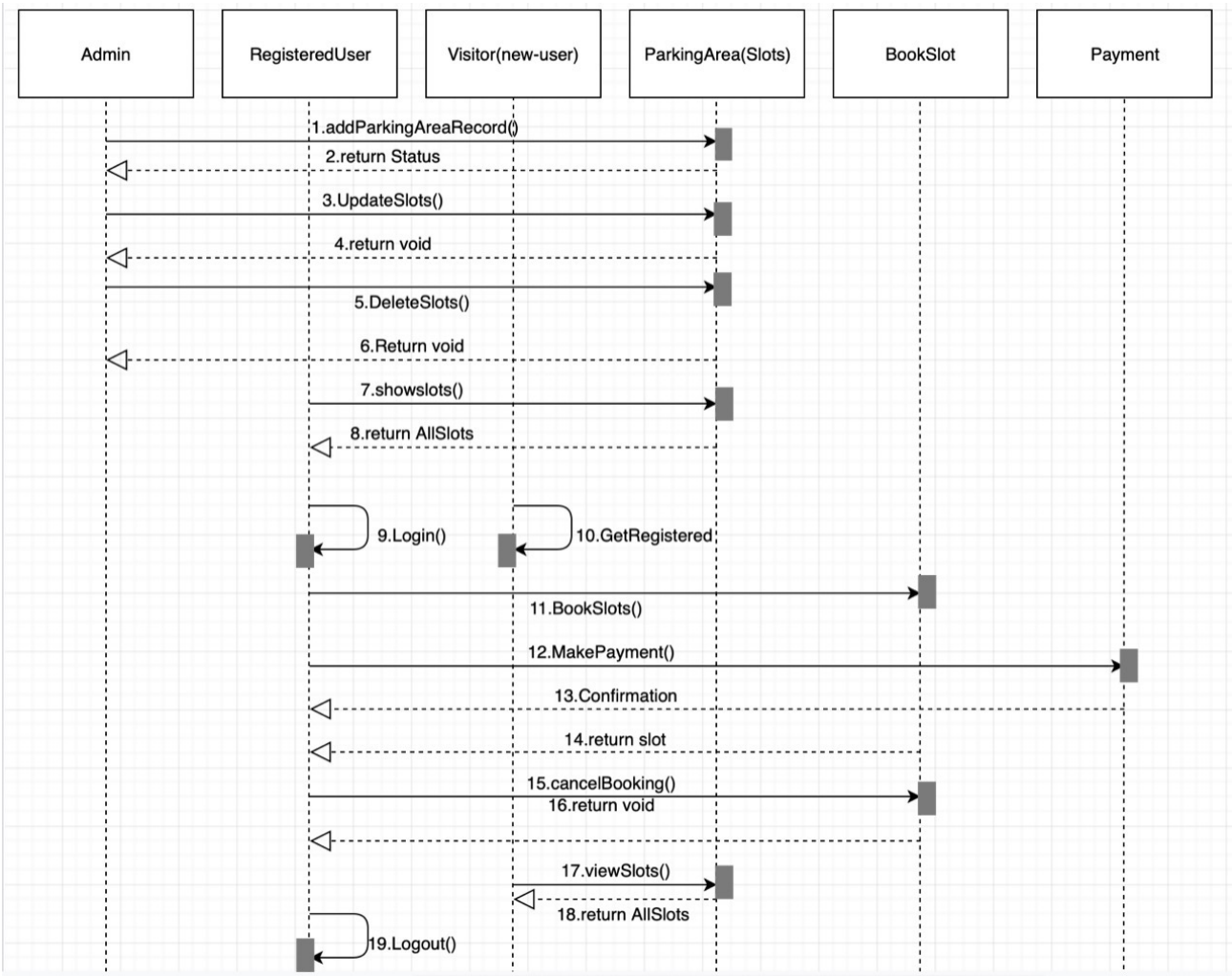


Figure 4: Sequence Diagram

4.2.5 Activity Diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc

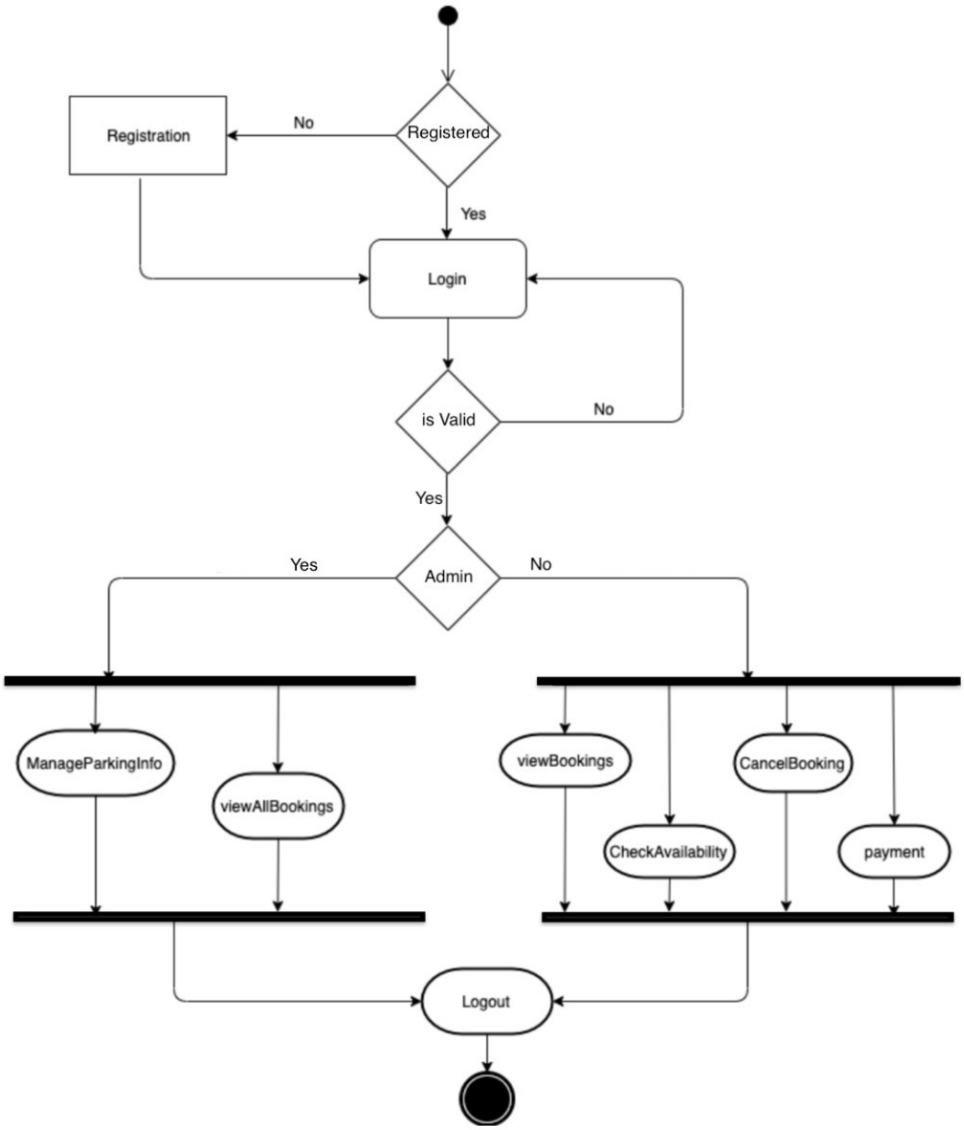


Figure 5: Activity Diagram

4.2.5.1 Purpose of Activity Diagrams

The basic purposes of activity diagrams is similar to other diagrams. It captures the dynamic behaviour of the system. Other diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another. Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part. It shows different flows such as parallel, branched, concurrent, and single.

The purpose of an activity diagram can be described as −

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

4.2.6 Modules

A module is a separate unit of software or hardware. Typical characteristics of modular components include portability, which allows them to be used in a variety of systems, and interoperability, which allows them to function with the components of other systems. The term was first used in architecture.

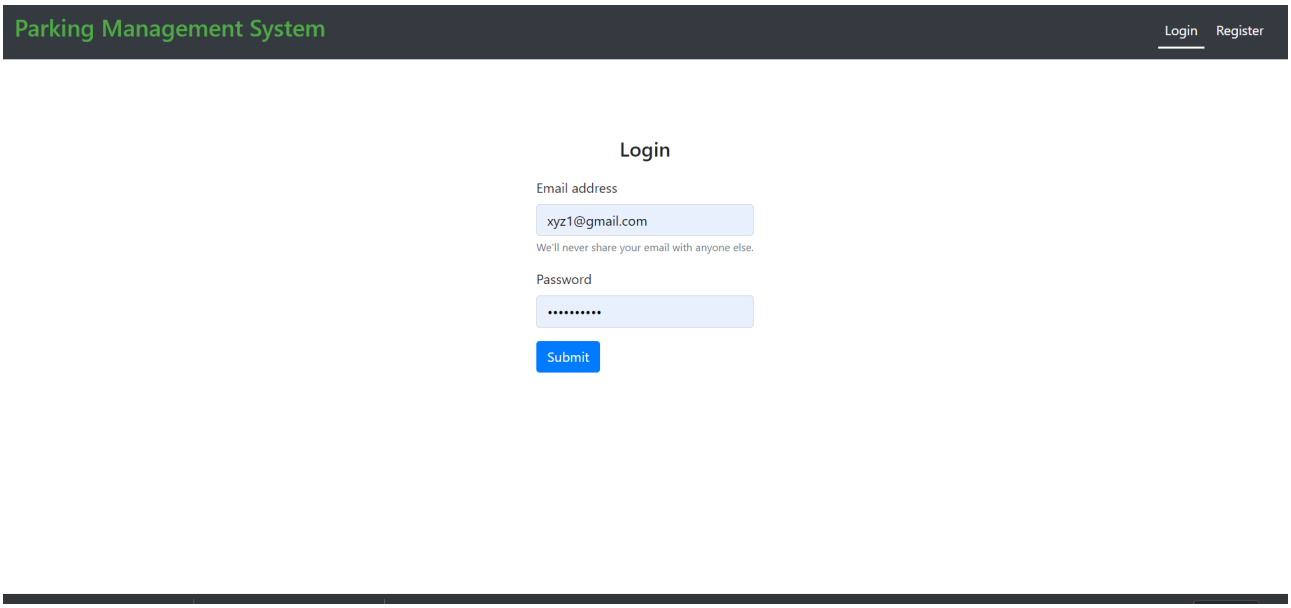
In computer programming, especially in older languages such as PL/1, the output of the language compiler was known as an object module to distinguish it from the set of source language statements, sometimes known as the source module. In mainframe systems such as IBM's OS/360, the object module was then linked together with other object modules to form a load module. The load module was the executable code that you ran in the computer.

Modular programming is the concept that similar functions should be contained within the same unit of programming code and that separate functions should be developed as separate units

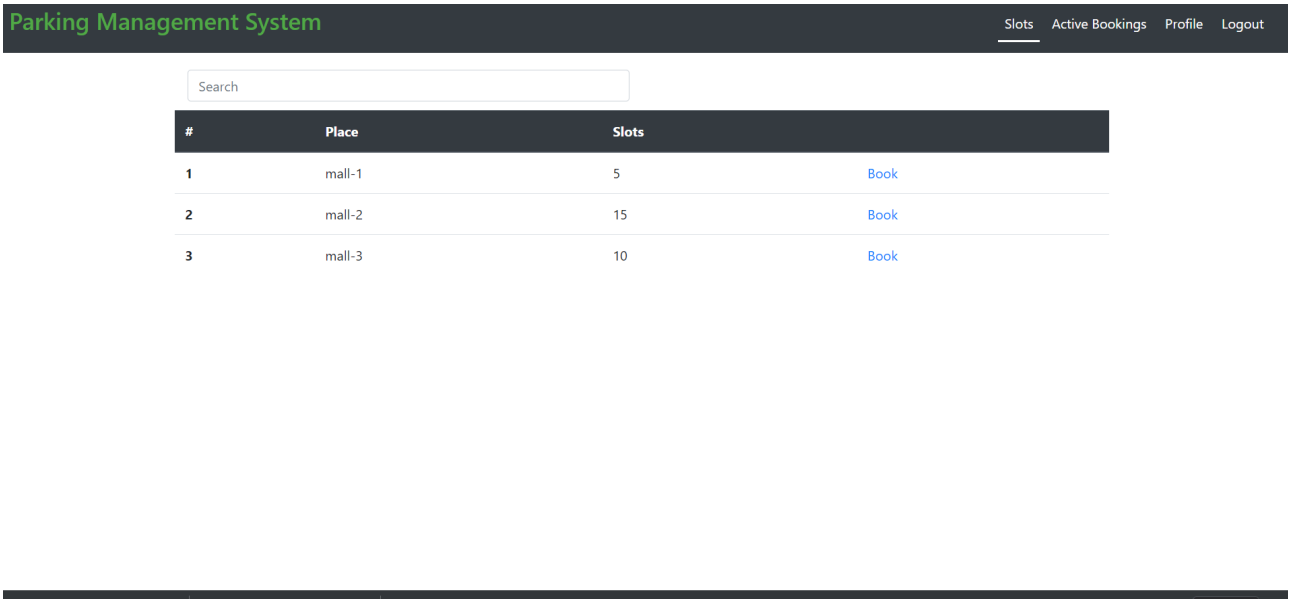
of code so that the code can easily be maintained and reused by different programs. Object-oriented programming is a newer idea that inherently encompasses modular programming.

4.2.7 User Interface

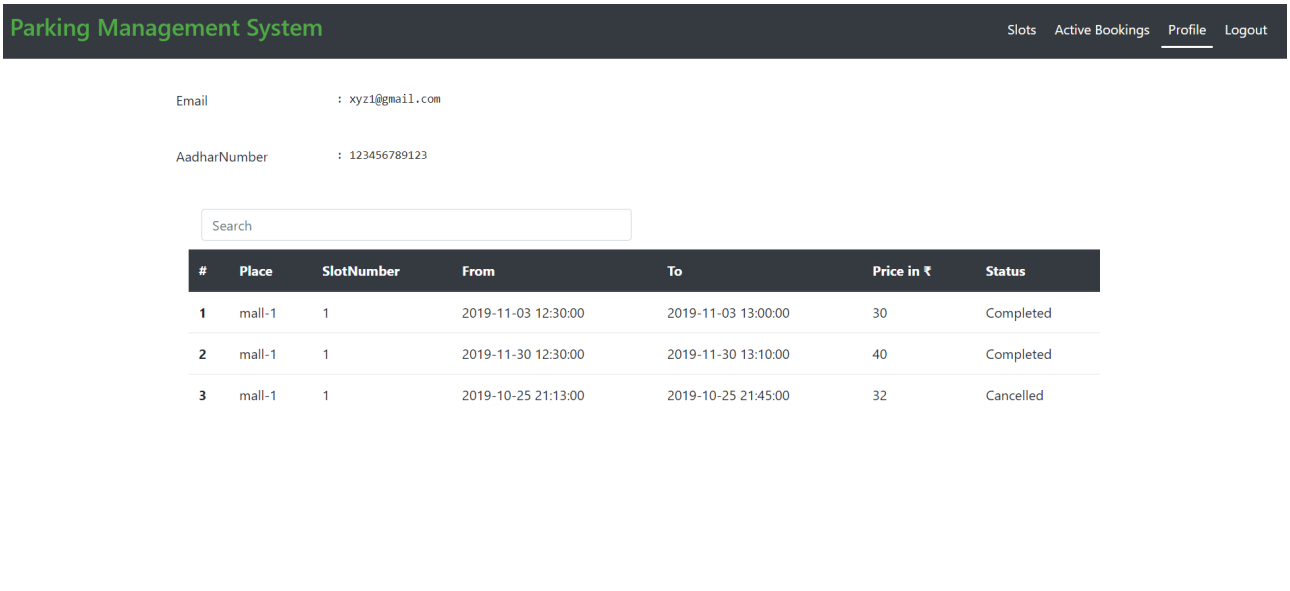
In information technology, the user interface (UI) is everything designed into an information device with which a person may interact. This can include display screens, keyboards, a mouse and the appearance of a desktop. It is also the way through which a user interacts with an application or a website. The growing dependence of many companies on web applications and mobile applications has led many companies to place increased priority on UI in an effort to improve the user's overall experience.



(Login Page)



(parking Slots search page)



(profile and previous booking history page)

CHAPTER 5

5. IMPLEMENTATION

5.1 INTRODUCTION:

For making things less confusing we could divide the implementation into three parts. Front end which provide UI for the users , and this is build with JAVASCRIPT framework REACT , backend which process all the requests is written in nodeJs . Front-end and backend programs runs on node server. For storing all the information about transactions & users a non-relational database is used called MongoDb. The database runs on mongo server locally.

For state management system a react library called redux is used.User can book his parking slots by choosing the data and time with the help of user interface , and request directly go to database with the help of nodeJs.Payments are handled by third party

5.2 Node JS:

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server- and client-side scripts.

Though .js is the standard filename extension for JavaScript code, the name "Node.js" does not refer to a particular file in this context and is merely the name of the product. Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

The Node.js distributed development project, governed by the Node.js Foundation, is facilitated by the Linux Foundation's Collaborative Projects program.

5.3 React:

A react application is made of multiple components, each responsible for rendering a small, reusable piece of HTML. Components can be nested within other components to allow complex applications to be built out of simple building blocks.React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding

5.4 Redux:

Redux is a predictable state container for JavaScript apps. As the application grows, it becomes difficult to keep it organized and maintain data flow. Redux solves this problem by managing application's state with a single global object called Store. Redux fundamental principles help in maintaining consistency throughout your application, which makes debugging and testing easier.The state of your whole application is stored in an object tree within a single store. As whole application state is stored in a single tree, it makes debugging easy, and development faster.To specify how the state tree is transformed by actions, you write pure reducers. A reducer is a central place where state modification takes place. Reducer is a function which takes state and action as arguments, and returns a newly updated state.

5.5 MongoDb:

MongoDB is a cross-platform, document oriented database that provides high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Any relational database has a typical schema design that shows the number of tables and the relationships between these tables. While in MongoDB, there is no concept of relationship.MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.

5.5 Code snippets:

For registration :

```
export const postRegisterUser = (req, res) => {
    const { email, password,adharNumber } = req.body;
    usersModel.findOne({ email },async (err, user) => {
        if (err) return serverError(res)
        if (user) return badRequest(res,
'user already exists with this email');
        const adhar = await usersModel.findOne({adharNumber
:parseInt(adharNumber)});
        if(adhar) return badRequest(res,
'user already exists with this adhar number');
        bcryptjs.genSalt(10, (err, salt) => {
            if (err) return serverError(res);
            bcryptjs.hash(password, salt, (err, hash) => {
                if (err) return serverError(res);
                new usersModel({
                    email,
                    password: hash,
                    role: 'user',
                    adharNumber
                }).save().then(() => sendResponse(res)).catch(err =>
 {
                    console.log('error while registering user', err);
                    return serverError(res);
                })
            })
        })
    })
}
```

For Login:

```
export const postUserLogin = async (req, res) => {
    try {
        const { email = '', password = '' } = req.body;
        const user = await usersModel.findOne({ email });
        if (!user) return badRequest(res, 'User not found');
        bcryptjs.compare(password, user.password, (err, sucess) => {
            if (err) return serverError(res, err);
            if (!sucess) return badRequest(res, 'Invalid credentials');
            const token = jwt.sign({ id: user._id }, SECRET_KEY);
            return sendResponse(res, token);
        });
    } catch (error) {
        serverError(res, error);
    }
}
```

To get slot details :

```
export const getSlotDetails = async (req, res) => {
    try {
        const { params: { slotId } } = req;
        const slot = await parkingSlotsModel.findById(slotId);
        sendResponse(res, slot.toObject());
    } catch (error) {
        serverError(res, error)
    }
}
```

To get all parking slots :

```
export const getParkingSlots =async (req, res) => {
    try {
        const parkings = await parkingSlotsModel.find({})
        return sendResponse(parkings);
    } catch (error) {
        serverError(res, error)
    }
}
```

For checking slot status :

```
export const checkSlot = async (req, res) => {
    try {
        let { body: { parkingId, slotNumber, from, to }, user: { id, email } } = req;
        from = new Date(from);
        to = new Date(to);
        const parkingDeatil = await parkingSlotsModel.findOne({ _id: parkingId, slots: { $gte
: slotNumber } })
        if (!parkingDeatil) return badRequest(res, 'No slot found');
        const bookings = await activeBookingsModel.aggregate([
            {
                $match: {
                    parkingId,
                    slotNumber: parseInt(slotNumber),
                }
            },
            {
                $match: {
                    $expr: {
                        $or: [
                            {
                                $and: [{ $gte: [from, '$from'] }, { $lte: [from, '$to'] }]
                            },
                            {
                                $and: [{ $gte: [to, '$from'] }, { $lte: [to, '$to'] }]
                            },
                            {
                                $and: [{ $lte: [from, '$from'] }, { $gte: [to, '$to'] }]
                            }
                        ]
                    }
                }
            },
            {
                $limit: 1
            }
        ]);
        if (bookings[0]) return badRequest(res, 'Slot has already booked');
        sendResponse(res);
    } catch (error) {
        serverError(res, error)
    }
}
```

Slot booking:

```
export const postBookSlot = async (req, res) => {
    try {
        let { body: { parkingId, slotNumber, from, to, tokenId }, user: { id, email } } = req;
        from = new Date(from);
        to = new Date(to);
        const parkingDeatil = await parkingSlotsModel.findOne({ _id: parkingId, slots: { $gte
: slotNumber } })
        if (!parkingDeatil) return badRequest(res, 'No slot found');
        const bookings = await activeBookingsModel.aggregate([
            {
                $match: {
                    parkingId,
                    slotNumber: parseInt(slotNumber),
                }
            },
            {
                $match: {
                    $expr: {
                        $or: [
                            {
                                $and: [{ $gte: [from, '$from'] }, { $lte: [from, '$to'] }]
                            },
                            {
                                $and: [{ $gte: [to, '$from'] }, { $lte: [to, '$to'] }]
                            },
                            {
                                $and: [{ $lte: [from, '$from'] }, { $gte: [to, '$to'] }]
                            }
                        ]
                    }
                }
            },
            {
                $limit: 1
            }
        ]);
        if (bookings[0]) return badRequest(res, 'Slot has already booked');
        const payment = await stripe.charges.create({
            amount: parseInt(Math.abs(new Date(from) - new Date(to)) / 60000) * 100,
            currency: "inr",
            description: "An example charge",
            source: tokenId
        });
        const newBooking = await new activeBookingsModel({
            userId: id,
            parkingId,
            from,
            to,
            slotNumber,
            name: parkingDeatil.name,
            payment: payment.id
        }).save();
        setTimer(newBooking._id, to, from, email);
        sendResponse(res, newBooking);
    } catch (error) {
        serverError(res, error);
    }
}
```

Delete current booking:

```
export const deleteBooking = async (req, res) => {
    try {
        const { params: { bookingId } } = req;
        let booking = await activeBookingsModel.findByIdAndRemove(bookingId);
        if (!booking) return badRequest(res, 'No record found');
        deleteTimer(bookingId);
        booking = new bookingsModel({ ...booking.toObject(), status: false }).save();
        sendResponse(res, booking);
    } catch (error) {
        serverError(res, error);
    }
}
```

All bookings :

```
export const getUserBookings = async (req, res) => {
    try {
        const { user: { id } } = req;
        const bookings = await bookingsModel.find({ userId: id }).sort({ _id: -1 })
        return sendResponse(res, bookings);
    } catch (error) {
        serverError(res, error)
    }
}
```

All active bookings:

```
export const getUserActiveBookings = async (req, res) => {
    try {
        const { user: { id } } = req;
        const bookings = await activeBookingsModel.find({ userId: id });
        sendResponse(res, bookings)
    } catch (error) {
        serverError(res, error);
    }
}
```

All active bookings (ADMIN access) :

```
export const getAllCompletedBookings = async (req, res) => {
    try {
        const bookings = await bookingsModel.aggregate([{
            $match: {}
        },
        {
            $lookup: {
                from: 'users',
                let: { userId: '$userId' },
                pipeline: [{
                    $match: {
                        $expr: {
                            $eq: ['$_id', '$$userId']
                        }
                    }
                }, {
                    $project: {
                        password: 0
                    }
                }],
                as: 'user'
            }
        }]);
        sendResponse(res, bookings);
    } catch (error) {
        serverError(res, error);
    }
}
```

All users bookings (ADMIN access) :

```
export const getAllCompletedBookings = async (req, res) => {
    try {
        const bookings = await bookingsModel.aggregate([{
            $match: {}
        },
        {
            $lookup: {
                from: 'users',
                let: { userId: '$userId' },
                pipeline: [{
                    $match: {
                        $expr: {
                            $eq: ['$_id', '$$userId']
                        }
                    }
                }, {
                    $project: {
                        password: 0
                    }
                }],
                as: 'user'
            }
        }]);
        sendResponse(res, bookings);
    } catch (error) {
        serverError(res, error);
    }
}
```

CHAPTER 6

6. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.1 TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flaw should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concern.' with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system lusting is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software. or at least its purpose. It is Purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge cf the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing:

Unit testing is usually conducted as part of the combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test objectives

- All field entries must work properly.
- No two slots have the same booking at the same time interval.
- The entry screen, messages and responses must have least delay.

Features to be tested

- Server side program runs without errors
- System are able to run the application and connect to the server
- Systems information is shown in the front end web page
- User should be able to make payment with ease

Integration Testing

Software integration testing is the incremental integration testing of we or more , integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or — one step up—software applications at the company level — interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

6.2 TEST CASES

A TEST CASE is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application. There are two types of test cases as mentioned below: 1. Formal test cases: Formal test cases are those test cases which are authored as per the test case format. It has all the information like preconditions, input data, output data, post conditions, etc. It has a defined set of inputs which will provide the expected output. 2. Informal test cases: Informal test cases are authored for such requirements where the exact input and output are not known. In order to test them the formal test cases are not authored but the activities done and the outcomes are reported once the tests are run.

Test Cases Examples for Website:

Module: User Login:

Test Cases	Input	ExpOutput	ActOutput	Description
Valid Login	username, password	Success	Success	Login Successful
Invalid Login	username, password	Failed	Failed	Invalid Credentials

Table 1: Login Test Case

Module: User Registration:

Test Cases	Input	ExpOutput	ActOutput	Description
Register new user	email, password, aadhar no	Register successful	Register successful	User registered

Table 2: Registration Test Case

Module: Slot Availability:

Test Cases	Input	ExpOutput	ActOutput	Description
Slot Booking	Date, Time	Proceed	Proceed	Slot available
Invalid Booking	Date, Time	Enter valid date	Enter valid date	Booking failed

Table 3: Slot Booking Test Case

Module: Payment:

Test Cases	Input	ExpOutput	ActOutput	Description
Payment	Credit/Debit card credentials	Booking successful	Booking successful	Slot booked successfully
Invalid Payment	Credit/Debit card credentials	Enter valid credentials	Enter valid credentials	Booking failed

Table 4: Payment Test case

CHAPTER 7

7. CONCLUSION

7.1 Advantages

- It is highly feasible for extremely small sites that are unable to accommodate a conventional ramped parking structure.
- There is high parking efficiency (i.e. sf/space and cf/space).
- There is no need for driving while looking for an available space.
- Emissions are greatly brought down and reduced.
- The patrons wait for their car in a highly controlled environment.
- There are less chances for vehicle vandalism.
- There is a minimal staff requirement if it is used by known parkers.
- It is possible that the retrieval time is lower than the combined driving/parking/walking time in conventional ramped parking structures.

7.2 Limitations

- There is a greater construction cost per space (but this may be offset by the chance for lesser land costs per space and the system manufacturers say that the operating and maintenance cost will be lower as compared to a conventional ramped parking structure).
- Use of redundant systems will result in a greater cost.
- It may be a bit confusing for unfamiliar users.
- It is not recommended for high peak hour volume facilities.
- There is an uncertain building department review and approval process.
- It requires a maintenance contract with the supplier.

7.3 Future Work:

- A clean record of user's history is maintained along with credentials.

- Would associate with shopping malls or high scale platforms where efficient parking system is necessary
- Would make the current web-version of application into mobile application so wide spectrum of people can use the application.

7.4 Conclusion

In this project, From the implementation of the parking system being presented, its efficiency in alleviating the traffic problem that arises especially in the city area where traffic congestion and the insufficient parking spaces are undeniable. It does so by directing patrons and optimising the use of parking spaces.

the pros and cons of each sensor technologies can be analysed. Although, there are certain disadvantages in the implementation of system as described earlier, the advantages far outweighs its disadvantages.

REFERENCES

1.https://reactjs.org/

2.https://nodejs.org/en/

3.https://redux.js.org/

4.https://www.mongodb.com/