SOFTWARE ENGINEERING

J COMPONENT REVIEW – 3

Parking Garage Automation System

Submitted to: Prof Ramanathan L

Submitted by:

Charizma Gupta 17BCI0157

Rohith Rajeev Pillai 17BCI0181

Course Code: CSE3001

Slot: A1

Software Requirements Specification

for

Parking Garage Automation System

Version 1.0 approved

Prepared by Charizma Gupta & Rohith Pillai

VIT, Vellore

08/05/2020

Table of Contents

Contents

Revision History

Name	Date	Reason For Changes	Version
Parking Garage Automation System	10/05/2020	First Version	1.0

1. Introduction

1.1 Purpose

This software requirements specification is made for **Parking** version 1.0. The product is a computerized parking system, implemented in an Android app. The purpose of this document is to convey both the functional and non-functional requirements of this project to the reader. This document will contain:

- A description of how the app functions
- A detailed description of all the functions the app is capable of
- A specification of the app's functional and non-functional re
- The app will allow a company to manage parking spaces in a parking garage of their choice. The app will allow users to login and then view which spaces are vacant and which are taken. The user can then pick a free space and then reserve it for themselves. This makes the space locked to every other user. The user must then un-reserve the spot when they are done using it and that returns its state to vacant. The company is able to set different kinds of users and lock specific parking spaces to specific users.

1.2 Document Conventions

This document uses the following abbreviations:

SRS	Software Requirements Specification
App	(The Android) Application

1.3 Intended Audience and Reading Suggestions

This document is intended to be read by various different groups of audiences. The first is the developers of the application, they will use this documentation as a guide to create the fully fleshed out product. The second audience is the client who uses this product, they can use the document to have a detailed description of all that the app is capable of and point out any discrepancies with the developers. The final group is the maintenance staff, who will use this document to repair, upgrade and maintain the product.

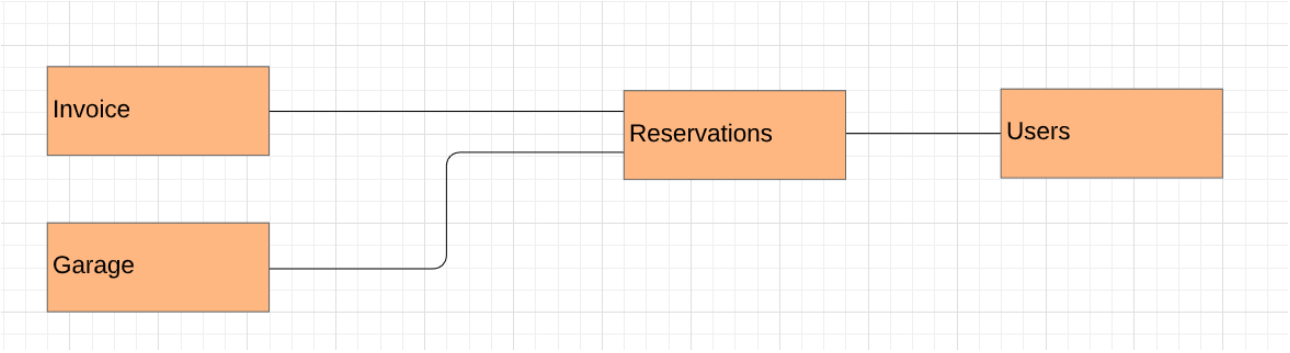
1.4 Product Scope

The purpose of this project is to create a mobile application which can be used to efficiently manage a parking garage. The users will need to input their desired free parking space for it to become locked. The output will be a successful locking of the space. Later the user will also need to unlock their space when they leave. The software has to confirm that no other reservation request has taken place in that space at the same time or a few seconds prior, ergo synchronization has to be present. An SQLite-based database will be used to store all the booking data and Android Studio will be used to develop the product.

2. Overall Description

2.1 Product Perspective

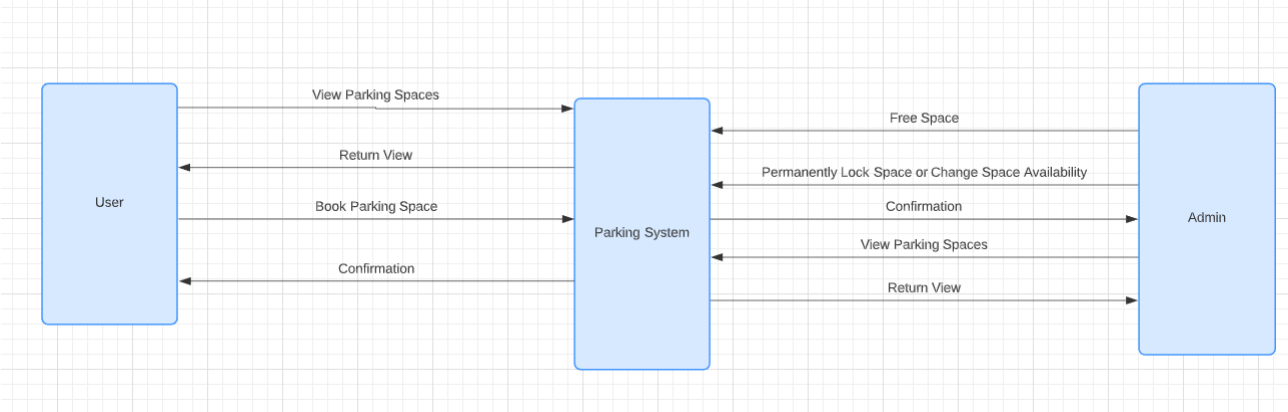
This is a new and self-contained product. The entity “User” should be able to interact with the system to create a “Reservation” for a specific parking space. There will be two types of users, the normal user and an administrator who will have more functions available to them. The diagram below shows how the tables are set up in the SQLite database.



2.2 Product Functions

The product should be capable of allowing for any unoccupied parking space to be booked provided it isn’t reserved for specific types of users. In the case that it is a (for example) “staff only” parking space, then only an admin or users chosen by the admin should be able to book that space. Users should be capable of viewing and booking/unbooking parking spaces. Admins should be capable of everything users are capable of and also should be able to lock spaces permanently (in the case that the parking space is unusable) or reserve spaces for specific kinds of users (changing the space’s availability).

The level 0 dataflow diagram below shows how the product functions in a simplified manner.



2.3 User Classes and Characteristics

There's two main types of user classes for this product, "Normal Users" and "Admins". The vast majority of users for this product will be from the "Normal Users" class and so they have influenced most of the design choices in the app. The table below demonstrates the differences in functionalities offered to both types of classes, their requirements and characteristics.

Type of User	Normal User	Admin
Functions available	-Normal users can use the app to reserve a space for their personal need -Normal users can also un-reserve their space -An invoice will be generated depending on the time duration for which the spot was reserved	-Admins will use the app to reserve one or many spaces for more than just themselves (example, employee parking spaces) -Admins can also un-reserve any spaces, including the spaces other users have reserved -Admins can also lock specific spaces from being used (in case of damage or changes to the garage)
Technical Expertise	-Varied, can be low level or high level according to experience with other Android Applications	-High level
Design choices made due to user type	-Application should be simple, favor usage of self-explanatory buttons to large amounts of text -Application should prefer being visual (pictures over text) -Application should be intuitive for non-experienced users, no hidden buttons, simple aesthetic	-Admin should have a different menu with more buttons to allow for the usage of their extra functions -This menu does not have to be tailored towards lower skill level users as admins have to be at a high skill level

2.4 Operating Environment

Since the application is based on the Android system, it will require an android smartphone. The hardware platform is therefore Android-based mobiles and the software will be all Android versions between 8.1 (Oreo) and 10 (Android Q). This application should peacefully co-exist with all other google play store based android applications as it's completely detached from them, and exists as its own independent application.

2.5 Design and Implementation Constraints

The application will be limited to the Android OS only as there aren't enough people working on this project to also create an IOS port. The application won't be CPU or RAM heavy; it should be able to run on all ranges of smartphones that meet the hardware and software platforms described earlier. The product also has to be designed in such a way as to allow the client to be able to maintain and upgrade it after the handover, this means it has to be as simple as possible and use efficient code to execute all of its functions.

Security is also an important consideration; a normal user's booking shouldn't be non-consensually cancelled as this could lead to revenue loss for the client. The databases the app uses should be

secured as a breach could lead to the exposure of admin details, and with the privileges admins have, the entire garage can be shut down.

2.6 User Documentation

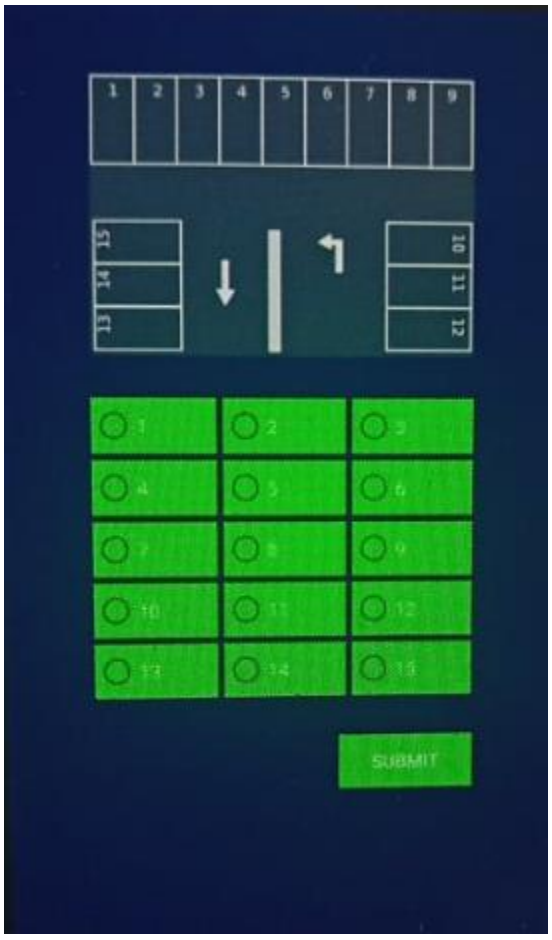
This Software Requirements Specification alongside the Software Design Documentation shall be delivered alongside the product to ensure the smooth functioning of the app.

2.7 Assumptions and Dependencies

The project assumes that the average user at least has the slightest amount of knowledge over how modern smartphone applications operate. The app is also designed to be used with Android and Android only; no considerations were made during the development phases to make the code easily portable to other operating systems. The final dependency is the skill level of the technical staff that will manage, maintain and upgrade the app, it is assumed that they are highly technical trained developers.

3. External Interface Requirements

3.1 User Interfaces



The Graphical User Interface used in the app can be interacted with using touch. There are 12 different screens in the app. The space reservation screen is shown beside, each button corresponds to each parking space. Buttons that are green indicate that space is free whereas if a space is booked, its button turns red. This simplistic design is to be able to appeal to users of even extremely low smartphone experience. Touch-based inputs are the most easy and intuitive type of inputting for the average human.

The design of the app was heavily influenced by industry standards for UI/UX design. Here are the conditions that the application design meets:

- **Consistency** is met by the consistent color scheme throughout all the menus, all (usable) buttons being the same shape and colour.
- **Error Handling** is managed using toasts to indicate when an unusable button was pressed on (example, trying to reserve an already reserved space)
- **Easy reversal of actions** is possible for the reservation step as un-reserving is only 2 button presses away.
- **Reduced short term memory load** is fulfilled by allowing a user who's reserved a spot to be able to view that immediately upon logging in (it's present in the reservation tab).
- Finally, **Minimalist** design is used to not overwhelm any less experienced users.

3.2 Hardware Interfaces

All phones that use Android 8.1 and later versions (till Android 10) support this application. The hardware requirements are very low as this application doesn't take much CPU power, GPU power or RAM to sustain. Theoretically, all phones that run those android versions should be capable of meeting the minimum hardware requirements to run this app.

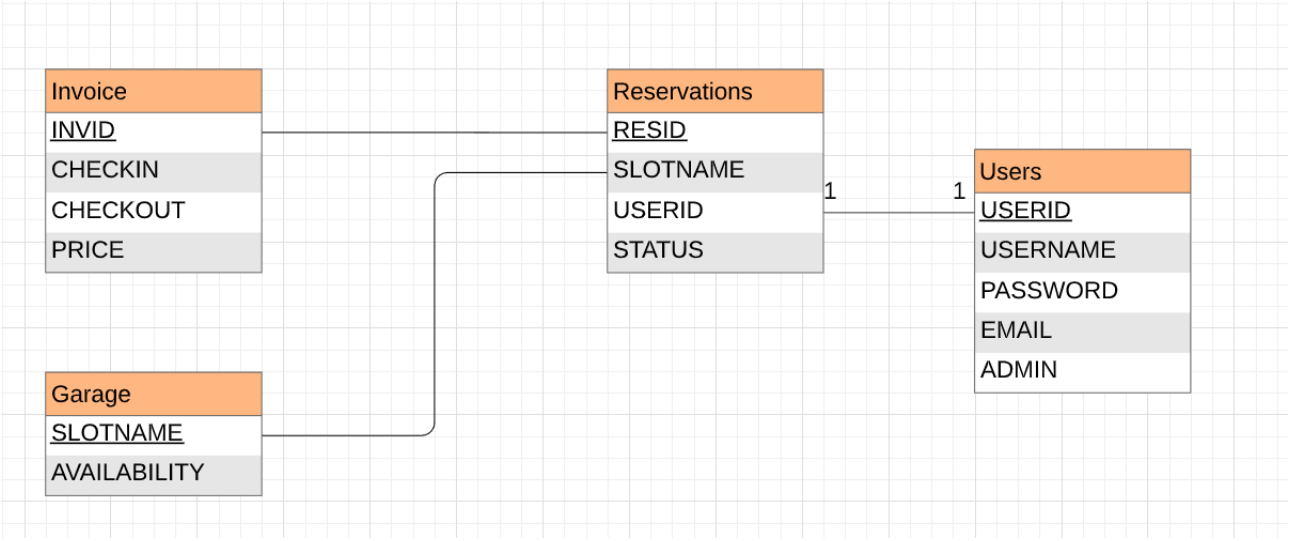
3.3 Software Interfaces

This application does use a built in SQLite database to store all of its data, which it stores locally and on the cloud. Communications between the SQLite database and the main application is facilitated through a Database helper thread that's always running provided the application is open.

The application is also built in Android Studio using Java code.

4. System Features

The ER diagram below shows the main modules of the product. All these modules interact with each other to produce the required output.



4.1 System Feature 1

Administrators are able to unreserve spaces from other users

4.1.1 Description and Priority

Administrators are capable of booking one or more spaces for different users, the main use of this feature is to allow for admins to forcefully free up spaces. This is a high priority function.

4.1.2 Stimulus/Response Sequences

First the Admin has to log in. Once logged in, they have to choose a parking lot and press on it. They can then forcefully unlock any space to unreserve it.

4.1.3 Functional Requirements

The app must open to the login screen. The admin login should take the admin to an admin homepage. The admin must be capable of unlocked slots in the parking lot menu.

4.2 System Feature 2

Administrators should be able to lock spaces permanently to all normal users.

4.2.1 Description and Priority

A lot of changes can happen over time to a parking garage, to ensure that the application is viable if there were changes to the parking space layout (in the case of damages), this function will allow the admin to ensure no user books a damaged space. This is a high priority function.

4.2.2 Stimulus/Response Sequences

First the Admin has to log in. Once logged in, they have to choose a parking lot and then press on it. They can then forcefully lock any parking space permanently.

4.2.3 Functional Requirements

The app must open to the login screen. The admin login should take the admin to an admin homepage. The admin must be capable of locking slots in the parking lot menu.

4.3 System Feature 3

Normal users should be able to reserve and unreserve available spaces, invoices should be created upon check-out.

4.3.1 Description and Priority

This function is the base functionality of the app. The application should allow normal users to reserve available spaces and cancel their reservations

4.3.2 Stimulus/Response Sequences

First the user has to log in. Once logged in, they have to pick which parking lot they want to reserve at. A map of the parking lot is then shown, a slot must be picked by pressing on the radio button of the slot they desire and then tapping on the submit button. This reserves the space and confirms it through a toast. When the user wishes to use the spot, they must go to the reservation page once again and choose check-in. Once done using the spot, they must check out. Cancellation is a button available in the reservation page after a spot has been reserved provided the check-in button hasn’t been pressed yet.

4.3.3 Functional Requirements

The app must open to the login screen. A successful login must take the user to the main page. The reservation of a slot must instantly make the slot reserved in the database and prevent further reservations from other users. A failure to reserve must toast the user with a message telling them the reservation failed. The invoice generation must produce a bill according to the difference in check-in and check-out times.

4.4 System Feature 4

Users should be able to register an account

4.4.1 Description and Priority

The registration process allows for the creation of accounts, which then allow users to access other functions depending on their account type. Low priority.

4.4.2 Stimulus/Response Sequences

First the user has to open the app. On the home screen the user has to press on the “registration” button, this moves them to the registration page. Once the user fills in their details and hits register, they get redirected to the login page and the process is complete. If the password and confirm password don’t match, an error is also thrown asking the user to rewrite them.

4.4.3 Functional Requirements

The app must open to the login screen. The “register” button must move the user to the registration screen. The register button must then add the user’s details to the database and redirect to the login screen. In case of packet overload, the database must not be updated and the user should be informed of the failure to register.

4.5 System Feature 5

Users should be able to login to their account

4.5.1 Description and Priority

The login process allows users to access functionalities the app provides. This involves reserving, cancelling and viewing reservations.

4.5.2 Stimulus/Response Sequences

First the user has to open the app. On the home screen the user has to type in their details and press the login button.

4.5.3 Functional Requirements

The app must open to the login screen. The login button must initiate a database query which confirms the details given (username and password). In case the system or database is overloaded, the login should fail and ask the user to attempt again. If the details are wrong the login should fail.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The application should not crash under normal operation, and in the case of a crash, should recover and choose whether the operation it was trying to conduct goes through or not based on the circumstance. This is to prevent interruption to other users as this application will be used by multiple users at the same time and be updating/querying the same database.

5.2 Safety Requirements

The reservation process on the database side should take a minimal amount of time as to avoid synchronization issues, spaces that are reserved should update the reservation page in real time for other users. Invoice generation should also accurately reflect the fee per hour rate according to the client's wishes, users must not be overcharged.

5.3 Security Requirements

The "Users" database holds email addresses and passwords of all users. A breach into the database could lead to massive privacy infringement. To counter this, all passwords are hashed before they're stored in the database. This doesn't' affect the login module as during the login procedure, the password the user inputs is hashed and compared to the password stored in the database. This effectively makes it so that only the email of the user and reservation they've made can be viewed, which greatly decreases the loss of privacy.

5.4 Software Quality Attributes

As the product is an application that runs on Android smartphones, it has to be portable to be able to be downloaded and spread. This is done by storing it in its APK (Android Package) format and handing that over to the client to do with as they please. The product also needs to be reliable as users who reserve their spot must have their spot actually reserved and an appropriate fee must be created by the invoice module. The application must also have high maintainability as this is a product that the client will most definitely make alterations to as it's only built for a specific parking architecture currently, other ones can be easily added with how the developers have coded the application. The software is also easy to learn to use, appealing to the widest audience of users as anyone planning on parking isn't guaranteed to be tech savvy.

5.5 Business Rules

Admins are capable of executing any function normal users can at any point in time. They are also capable of restricting the functions of normal users in the case of an emergency, by locking all spaces. The space locking feature also allows for a quick solution in a situation where a parking spot is damaged or has been put out of service temporarily. Normal users can reserve and un-reserve at all times the admins deem fit.

6. Other Requirements

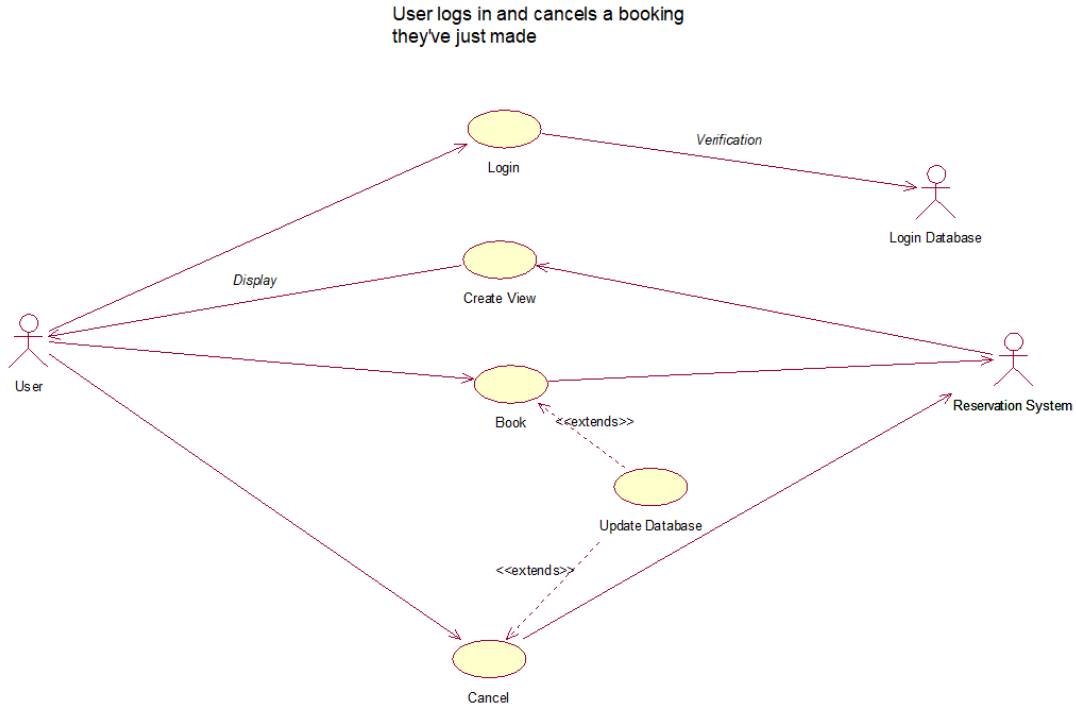
The database is implemented using SQLite, which does not support certain standard SQL conventions. This should be put into consideration if future developers decide to change the database architecture to a more standard one.

Appendix A: Glossary

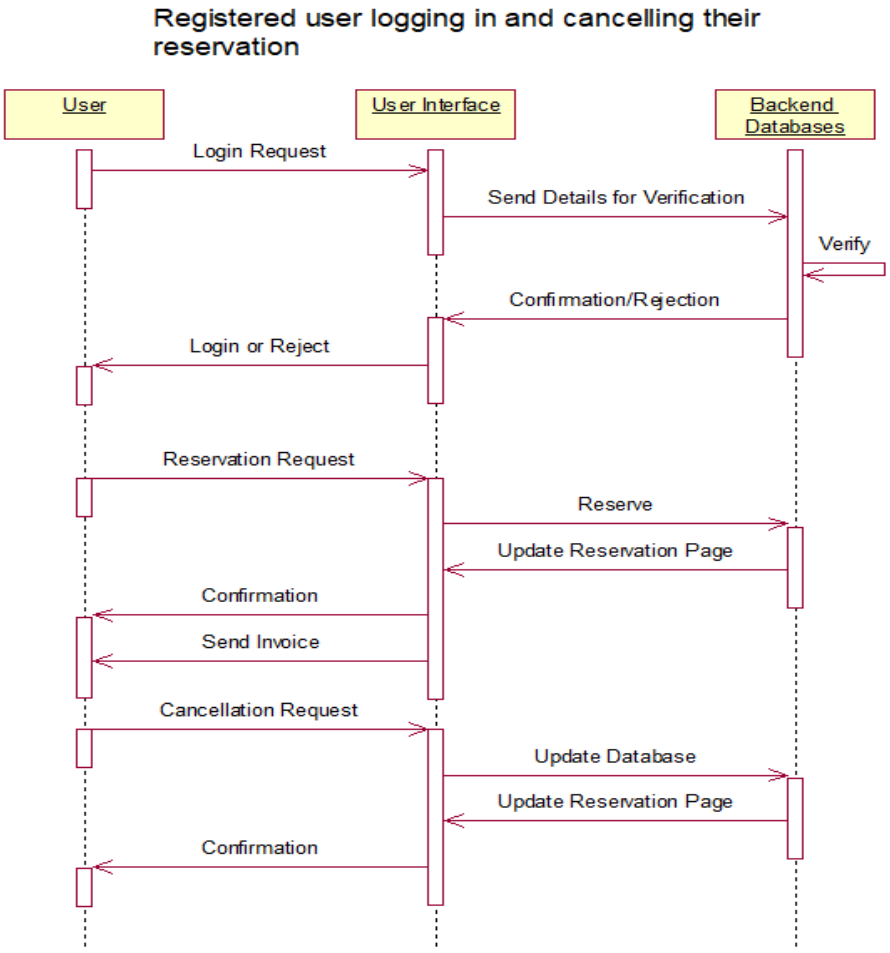
Refer back to section 1.2 to see document conventions and abbreviations used.

Appendix B: Analysis Models

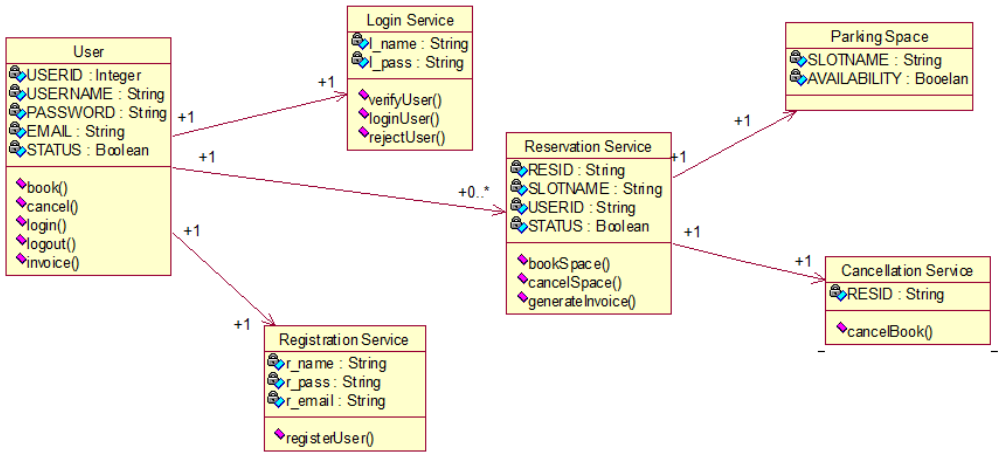
Use Case Diagram for a Normal User Booking and then cancelling



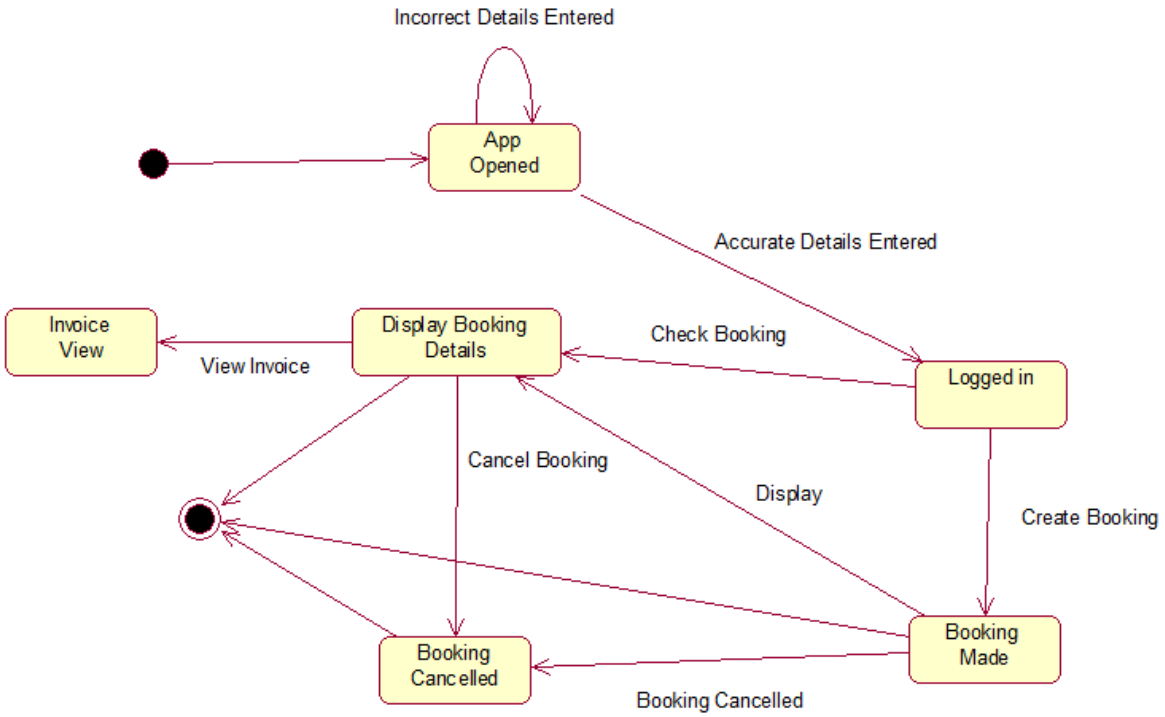
Sequence diagram for Reservation Cancellation after reservation is made



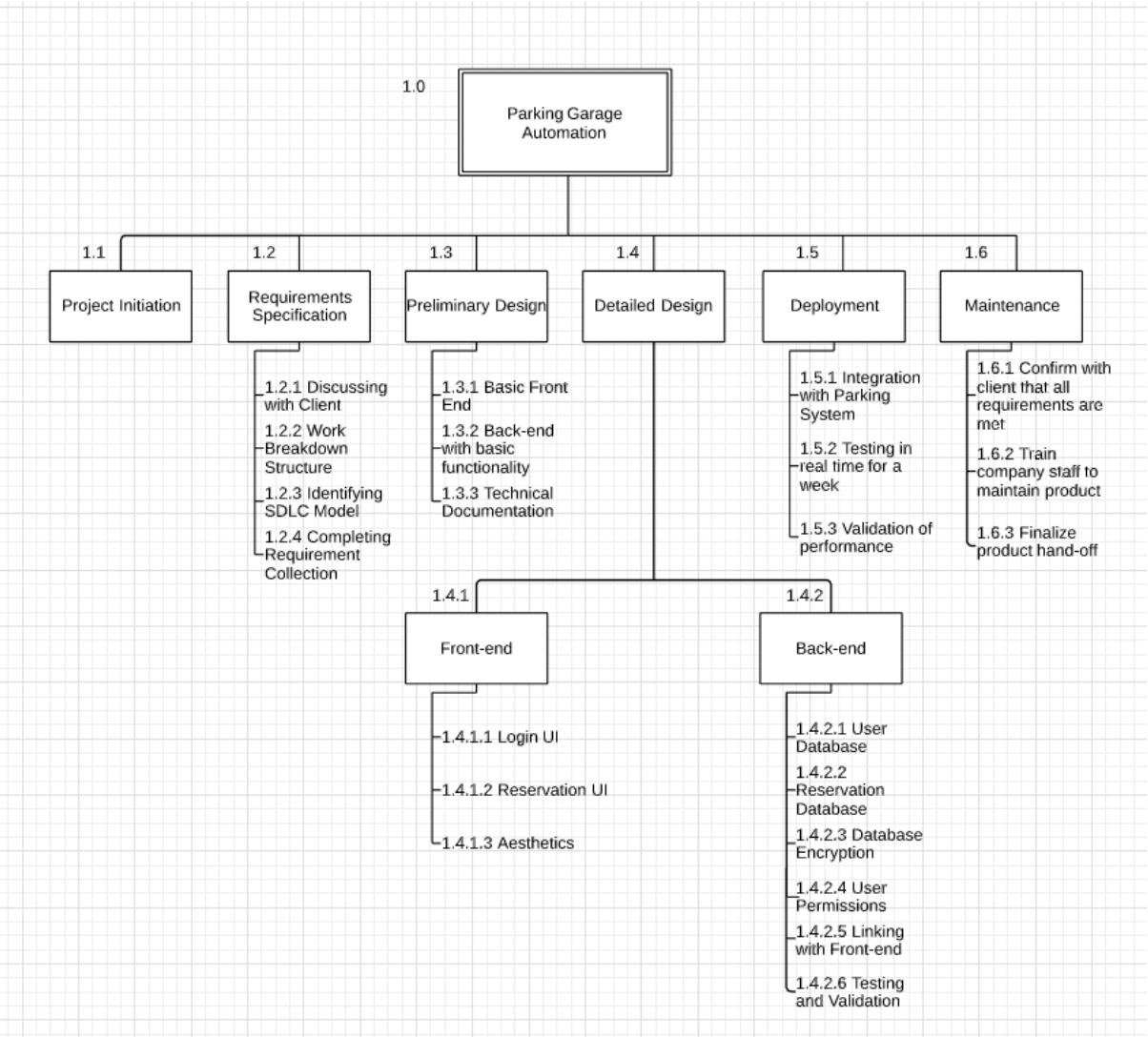
Class Diagram for the entire system



State Chart diagram for the entire system



Work Breakdown Structure



Software Design Specification for Parking Garage Automation System

Charizma Gupta and Rohith Pillai

.

Table of Contents

1. Introduction

The purpose of this Software Design Document is to provide a description of the design of our system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to build. The system described in this document is the Parking Garage Automation System. The product is an Android app that allows for efficient management of a parking lot. Users log in, reserve spaces, check in when they're using the space and check out when they're done. An invoice is generated and they pay for the duration that the car is parked. This document is intended to be read by the maintenance staff of the client, as it documents the entire software development cycle of this

product and all the design decisions made during that time. This document is made for Parking version 1.0 for Android versions 8.1 and above.

This document will go through:

- System Overview
- Design Considerations during development
- Goals and Guidelines
- Architectural Strategies
- System Architecture
- Policies and Tactics
- Detailed System Design

This document uses the following abbreviations:

SRS	Software Requirements Specification
App	(The Android) Application

This document uses *italics* for table names and **bold text** the first time a module is mentioned. The first use of this is in the "System Architecture" section.

1. System Overview

Parking is an Android app that allows users to reserve parking spots in up to 2 different parking lots. There are two types of users, "Admins" and "Normal Users", admins have more features available to them.

The functions available to normal users are:

- Register an account. The registration page is available to for anyone who has the app to use, allows for the creation of an account
- Login with their account. This allows for each type of user to access the features available to them.
- Reserve a parking space. This can be done provided the space isn't already reserved, or isn't locked by the admin
- Cancel a reservation. This can be done provided the space was reserved by that user. Admins can cancel reservations made by other users.
- Check-in to their parking space. This stores the time the user checked in, later used for the invoice generation
- Check-out from their parking space. This function initiates the invoice generation function
- View reservation. This takes the user to a page where they can check-in, check-out or cancel their reservation
- Generate invoice. This allows the user to view their invoice.

Admins can also use all those features. The extra functions available to admins are:

- Permanently locking parking spaces. This makes a parking space not available for reservations until the admin unlocks it.

2. Design Considerations

2.1. Assumptions and Dependencies

The project assumes that the average user at least has the slightest amount of knowledge over how modern smartphone applications operate. The app is also designed to be used with Android and Android only; no considerations were made during the development phases to make the code easily portable to other operating systems. The final dependency is the skill level of the technical staff that will manage, maintain and upgrade the app, it is assumed that they are highly technical trained developers.

2.2. General Constraints

The application will be limited to the Android OS only as there aren't enough people working on this project to also create an IOS port. The application won't be CPU or RAM heavy; it should be able to run on all ranges of smartphones that meet the hardware and software platforms described earlier. The product also has to be designed in such a way as to allow the client to be able to maintain and upgrade it after the handover, this means it has to be as simple as possible and use efficient code to execute all of its functions.

Security is also an important consideration; a normal user's booking shouldn't be non-consensually cancelled as this could lead to revenue loss for the client. The databases the app uses should be secured as a breach could lead to the exposure of admin details, and with the privileges admins have, the entire garage can be shut down

2.3. Goals and Guidelines

The main goals of the product are to be:

- Efficient
- Easy to use
- Consistent
- Minimalist

2.3.1. Efficient

The app should be efficient as it has to support multiple users using it at the same time. The app should also rarely crash. The reason for this is to not have the parking lot be completely locked up due to slow speeds or crashes.

2.3.2. Easy to use

There's no guarantee that everyone using the app is going to have a lot of experience with android apps. The product was built in such a way as to be intuitive and use visual cues (pictures) rather than large amounts of text.

2.3.3. Consistent

Since this app is solely for the reservation of parking spaces, the procedure for all the different features should be identical from update to update.

2.3.4. Minimalist

A minimalist aesthetic is chosen to appeal to the widest possible audience of consumers as anyone could be using the app. For example, if the parking lot in question was one from a shopping center, a myriad of different types of users would need to use the app, all with a different level of technical skill and experience with android apps.

2.4. Development Methods

This development cycles this product has gone through is the incremental software development cycle. In the incremental model, the iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed. An incremental life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

The first build of the product had the registration, login, reservation and cancellation systems implemented. They were tested and then the second build was conceived. The second build added the Admin's slot locking feature alongside invoice generation. After thorough testing of those new modules, the product is in the state it is in now. A potential next build could implement in-app payment options.

The incremental model was used to fit the deadlines assigned by the client, and also because it creates a tangible product that is complete (a build) every week or so. This allows for frequent reviews with the client as they desired.

3. Architectural Strategies

3.1. Software and Development Kit

The main programming language used is Java. Android Studio is the development platform used as it's the one the developers had most experience with. It also provided a virtual machine which allows for easy testing of any changes made to the code. SQLite-based database is used to keep the performance as high as possible to meet the goal of trying to make an efficient app.

3.2. User Interface Paradigms

As previously specified in the software requirements specification, the app's aesthetics were designed in such a way as to appeal to the broadest audience of users. Components of the Ben Shneiderman's 8 golden rules for UI design and Nielsen's 10 heuristics were considered in the creation of this aim, most important ones being consistency and the ability to easily reverse actions (such as with making a reservation).

3.3. Database

The app uses a SQLite database that's built into the app itself. It stores data like the booking id, usernames, and many more attributes discussed later and processes them through queries to allow for the successful updating of the app's state. The database changes affect the app's user interface in real time, one user reserving a slot makes that unavailable to all other users.

3.4. Error Detection and Recovery

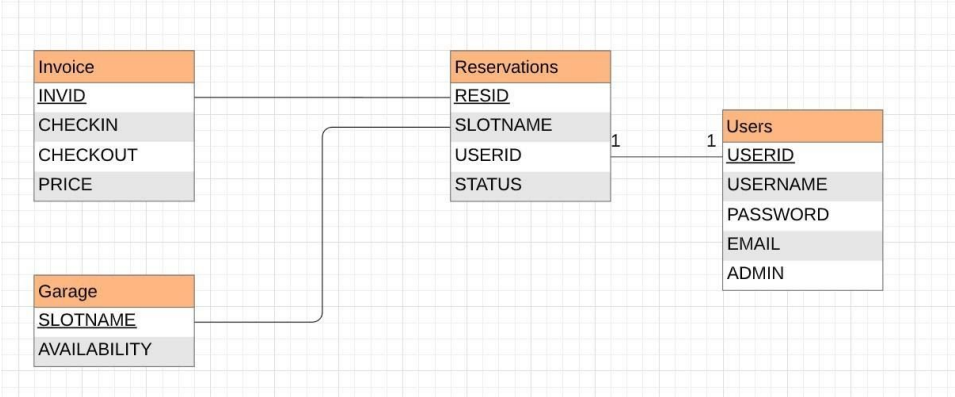
The app mainly uses the "toast" feature of android systems to inform the user about errors. During the registration process, the user is asked to confirm the password they want to set to their account, if the password is written incorrectly, a toast informs them it's incorrect and asks them to attempt it again. During the login process a user giving the incorrect password is also given a toast to be informed of a failed login. If a reservation goes wrong due to server overload or database overload, the action isn't completed and a toast is popped up to inform the user.

3.5. Future Plans

The next build of the software already has been conceived. The most significant addition would be the option to pay inside the app. Currently the app only generates an invoice and the payment must still be handled either external in the physical world or through a redirect to another payment gateway. The addition of a customized in-app payment gateway can streamline the app's invoice module. Another addition is the capability of adding more parking lots to the app, this should be installed as an admin feature.

4. System Architecture

The overall system is perfectly described by the ER diagram shown below.



The system was split into 12 different main modules/classes. This segmentation is based on how on a page by page basis for the application. The first module any user will interact with is the **Login module**. It is directly linked to the **Registration module** as both of them use the *Users* table shown in the ER diagram above. The login module queries the *Users* table to find matching credentials to the ones input by the user. The registration module is capable of updating the *Users* table to add new entries for users, allowing those new users to pass through the login screen.

Depending on whether an admin or a normal user logged in, either the **Normal User** module or the **Admin module** is initiated. Normal users are capable of viewing the parking lot, reserving, cancelling reservations, viewing their invoice/receipt, checking in and checking out. During the reservation procedure, the **Parking Lot module** is called upon to handle which slot is being taken from which parking lot. The **Reservations** screen is a module as it does also have to query the *Reservations* table to be able to display what reservation the user has made. The admin user also sees a different page entirely for the parking lot view and reserving slots, which uses another module, the **Admin Parking Lot** module.

There is a **Receipt module** that normal users can interact with through the check-in and check-out buttons. This module is responsible for generating accurate invoices by calculating the time difference between the check-in and check-out times.

The admin has exclusive access to a few modules. They can use the **Reservation Logs module** to view the history of all reservations and cancellations. The **Receipt Logs module** can be used in a similar fashion to view all past and current receipts. Finally, there is the **User List module** which allows the admin to view all currently existing accounts, the passwords stored in the *Users* table are hashed so privacy is minimally infringed on.

4.1. Subsystem Architecture

The only module not mentioned is the **Database Helper module**, this module is different to all the others as it doesn't represent a type of user or a page. This module is deeply interlinked with all the other modules as any query, update or interaction with the database initiated by any other module is actually conducted by the database helper module. It contains all the code for the SQL queries and the code for all the internal SQL tables.

5. Policies and Tactics

The integrated development environment used for this project has been Android studio as it is the development kit the developers had the most experience with and also provided a highly useful virtual machine. Almost all android-based apps are developed using Android studio because of how incorporated it is with the OS itself.

5.1. Coding Guidelines and Conventions

As Java and Android Studio were used, a lot of coding guidelines and conventions were followed. Java is an extremely robust language and forces its coders to follow most standard conventions.

The code is safe, it doesn't hold any malicious intent towards users that install it. It functions as a parking reservation system and only a parking reservation system. It is quite secure as the database is encoded. But even if the database was breached into, the passwords stored in the database are in hashed in ___. This means that even in the worst-case scenario the only thing a hacker will attain are email addresses. The code is also reliable, every time the program is tested on the virtual machine It runs perfectly. Any feature added is thoroughly tested on the virtual machine. Portability is a given when it comes to android apps as the APK format ensures the app is greatly compatible with most android versions above a certain version. In the case of this product, Android version 8.1 is the minimum and all versions after support it.

The alternative platform for this product was to be an online desktop-based application with mobile browser support. However, most people who want to reserve a parking space would already be away from their desktop PC and so the product was more geared towards mobile. After deciding the app was going to be a mobile one, the main remaining choice was whether to make it an Android app or an IOS app. Fewer people own IOS devices and since the product has to appeal to the largest audience, Android was chosen.

7. Detailed System Design

7.1 Classification

Parking	Package
Login	Module
Registration	Module
Parking Lot	Module
Reservation	Module
Receipt	Module

Admin Parking Lot	Module
Reservation Logs	Module
Receipt Logs	Module
User List	Module
Database Helper	SubSystem
Home	Class
Admin Homepage	Class
Parking.db	Database File

7.2 Definition

Parking	The application project's code package
Login	This module verifies the user as registered, admin or unrecognised attempt.
Registration	The module registers regular users.
Parking Lot	A module that allows slot reservation.
Reservation	The module that shows the active reservation and allows for cancellations.
Receipt	The module to display the Invoice
Admin Parking Lot	Module to allow admin control of the Garage
Reservation Logs	Module to log every Reservation Entry
Receipt Logs	Module to log every Invoice Generation
User List	Module to display the list of registered users.
Database Helper	Module to access and query database.
Parking.db	Database File to store all the tables and data.

7.3 Responsibilities

Parking	Contains all the class files.
Login	Verifies login attempts, directs users to admin or user homepages accordingly.
Registration	Registers a user, hashes and stores their password
Parking Lot	Displays the garage layout, allows for slot selection, informs if slots are unavailable.
Reservation	Shows active reservations, lets users check in and checkout for accurate invoice generation, allows for booking cancellation.
Receipt	Generates an invoice on checkout.
Admin Parking Lot	Allows the admin to lock a slot so it can't be reserved by regular users and also to unlock a slot even if it was previously reserved or engaged by another user.
Reservation Logs	Displays all the records of the reservation table.
Receipt Logs	Displays all the records of the invoice table.
User List	Displays all the records of the users table.
Database Helper	Allows access to the SQLite database and contains all the functions to insert, delete, retrieve and query the tables of Parking.db.
Parking.db	Contains and stores all the data necessary.

7.4 Constraints

Login	Only a fixed number of sessions possible at a time.
Registration	Does not apply strong password policies.
Parking Lot	Attempts at Reserving have to be made before Slot Availability is revealed.
Receipt	Uses current device timestamps that can be fooled

	by changing system date and time settings.
Admin Parking Lot	Can't see what slots are reserved and not reserved directly. Must use a Reservation module for that.

7.5 Composition

Parking	Contains all the classes. Database Helper, Homepage, Parking Lot, Reservation, Receipt, Admin Homepage, Admin Parking Lot, Reservation Logs, Receipt Logs, User List.
Login	Links to the Admin or Regular homepage as well as the registration page.
Parking Lot	Has every slot as a radio button, an image layout and a submit button.
Reservation	Contains the reservation information as text, and the buttons for checkIn, checkOut and cancel.
Admin Parking Lot	Has every slot as a radio button, an image layout and lock and unlock buttons.
Database Helper	Has functions for every record addition, update and query.
Parking.db	Contains Users, Garage, Reservations and Invoice tables.

7.6 Uses/Interactions

Login	Interacts with Registration, Homepage, Admin Homepage.
Registration	Interacts with Login.

Parking Lot	Interacts with Homepage and Database.
Reservation	Interacts with Homepage and Database.
Receipt	Interacts with Homepage and Database.
Admin Parking Lot	Interacts with Admin Homepage and Database.
Reservation Logs	Interacts with Admin Homepage and Database.
Receipt Logs	Interacts with Admin Homepage and Database.
User List	Interacts with Admin Homepage and Database.
Database Helper	Interacts with Parking Lot – Reservation – Receipt, Admin Parking Lot – Reservation Logs – Receipt Logs – User List and the Database.

7.7 Resources

Parking.db	The race conditions and deadlock situations are handled by S

7.8 Detailed Subsystem Design

DatabaseHelper()

addUser	Add user to Users Table.
checkUser	Verifies username and password with Users table.
checkSlot	Checks for Slot availability.
makeRes	Adds relevant data and creates a Reservation record.
checkUserID	Confirms user ID to insert into

	Reservation record.
genInv	Generates an Invoice record with Check In time.
updInv	Updates the Invoice record with Check Out time.
calcInv	Uses check in and check out times from the invoice table to calculate total charges.
lock	Turns the availability status of a slot to Not Available in the Garage table.
unlock	Turns the availability status of a slot to Available in the Garage table.
printUL	Gives the entire list of users with attribute data.

8. Glossary

Android Studio - Android Studio is the official integrated development environment for Google's Android operating system

Development Kit - A software development kit (SDK) is a collection of software development tools in one installable package.

Dependency - This is when a component relies on another component or event to occur for its functioning

Module - A module is a software component or part of a program that contains one or more routines

Minimalist - Graphic Design which aims to be simple, minimal

Package - DescriptionAndroid Package is the package file format used by the Android operating system for distribution and installation of mobile apps

9. Bibliography

Documents and Libraries referenced:

https://developer.android.com/docs

https://www.sqlite.org/docs.html

https://github.com/jgilfelt/android-sqlite-asset-helper

Implementation Screenshots

for

Parking Garage Automation System

Version 1.0 approved

Prepared by Charizma Gupta & Rohith Pillai

VIT, Vellore

08/05/2020

User table

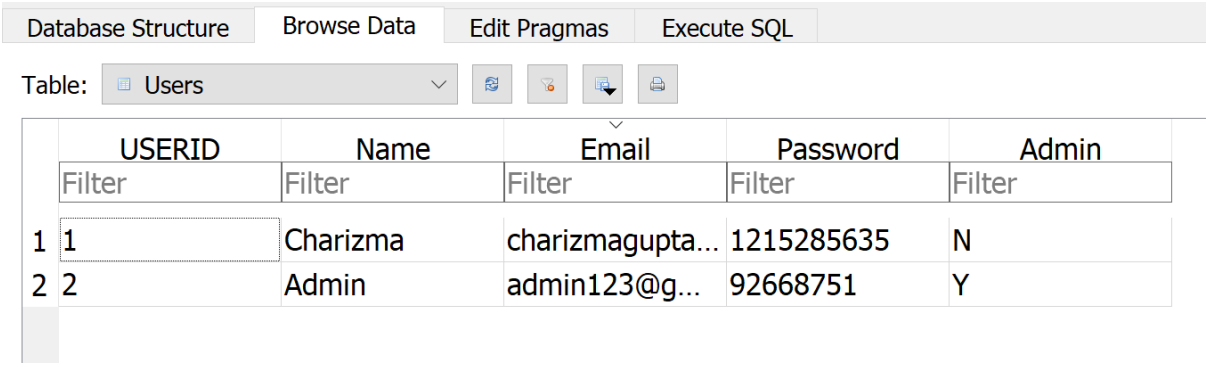

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Users

	USERID	Name	Email	Password	Admin
	Filter	Filter	Filter	Filter	Filter
1	1	Charizma	charizmagupta...	1215285635	N
2	2	Admin	admin123@g...	92668751	Y

Reservations Table

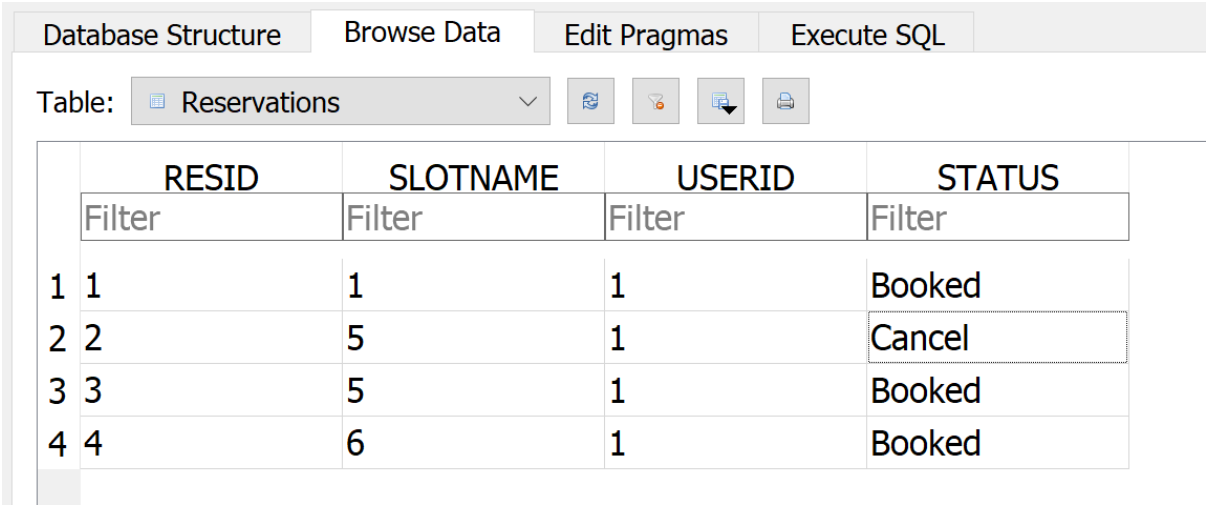

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Reservations

	RESID	SLOTNAME	USERID	STATUS
	Filter	Filter	Filter	Filter
1	1	1	1	Booked
2	2	5	1	Cancel
3	3	5	1	Booked
4	4	6	1	Booked

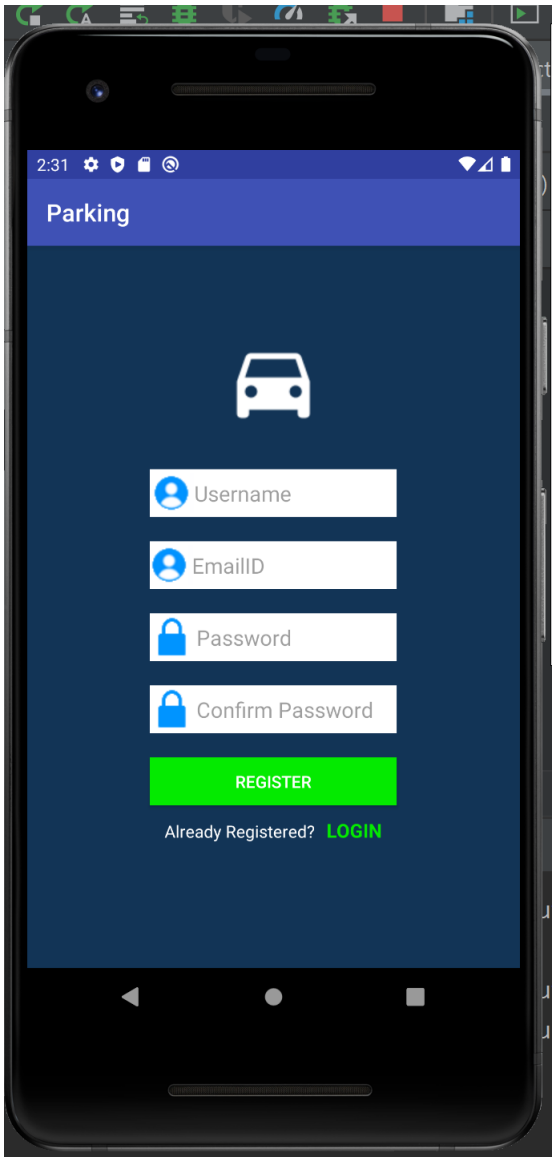
Garage Table

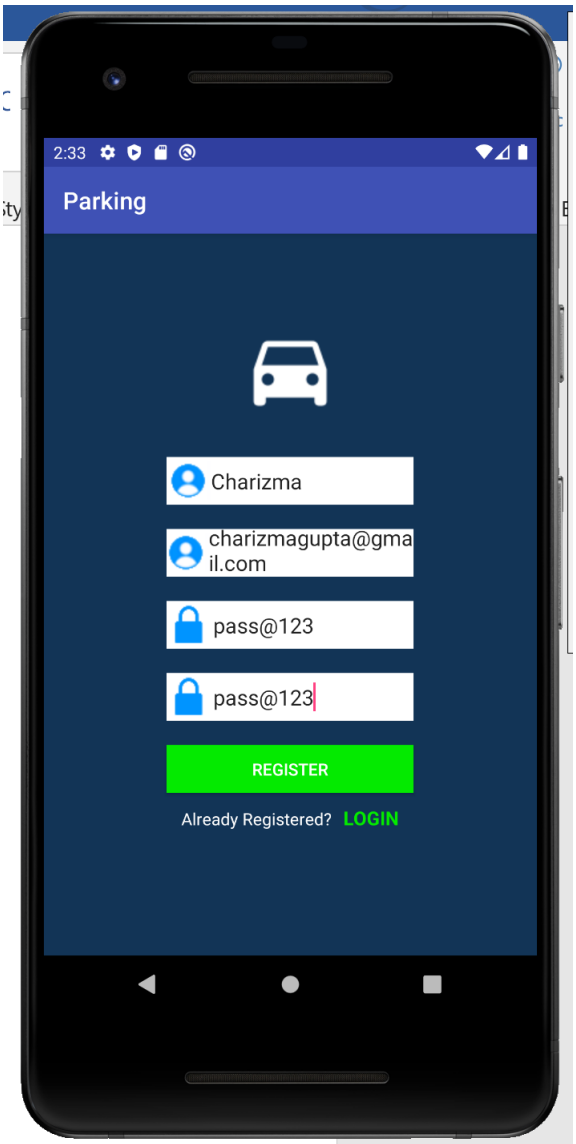
Database Structure | Browse Data | Edit Pragn

Table: Gara | New Rec

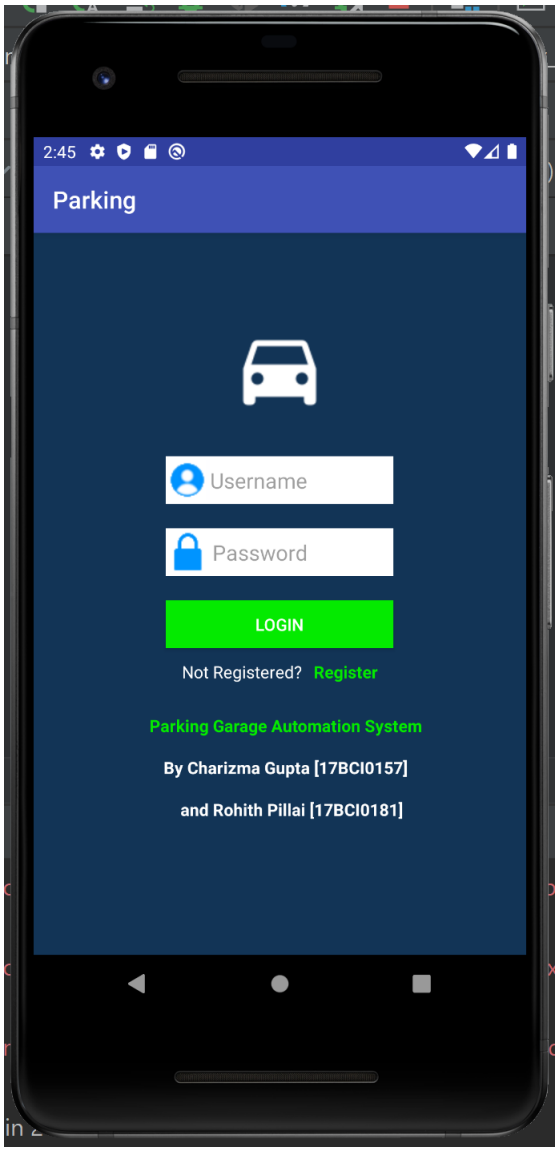
	SLOTNAME	AVAILABILITY
	Filter	Filter
1	1	0
2	2	1
3	3	1
4	4	1
5	5	0
6	6	0
7	7	1
8	8	1
9	9	1
10	10	1
11	11	1
12	12	1
13	13	1
14	14	1
15	15	1

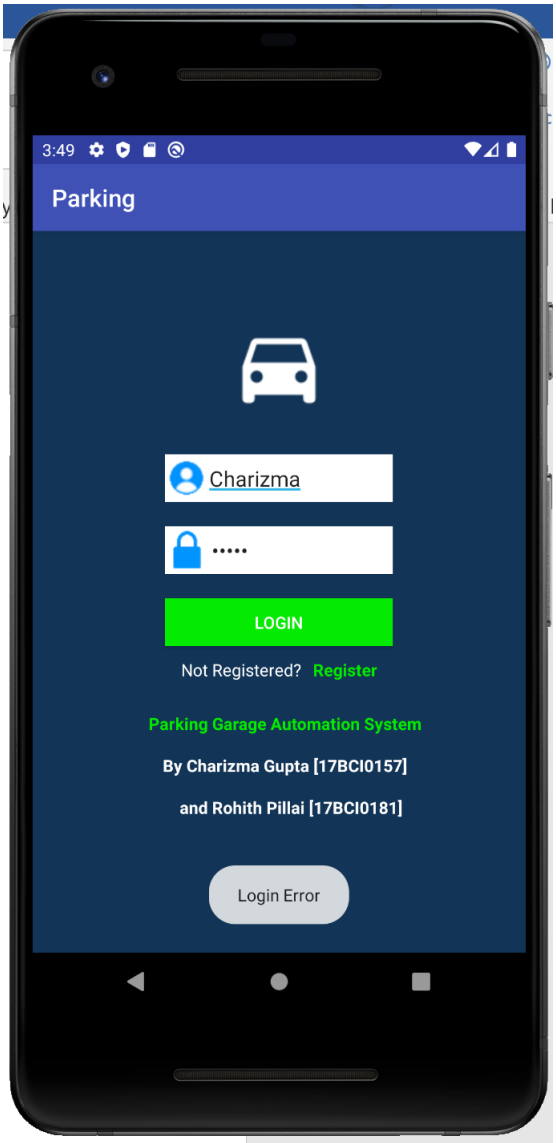
Registration

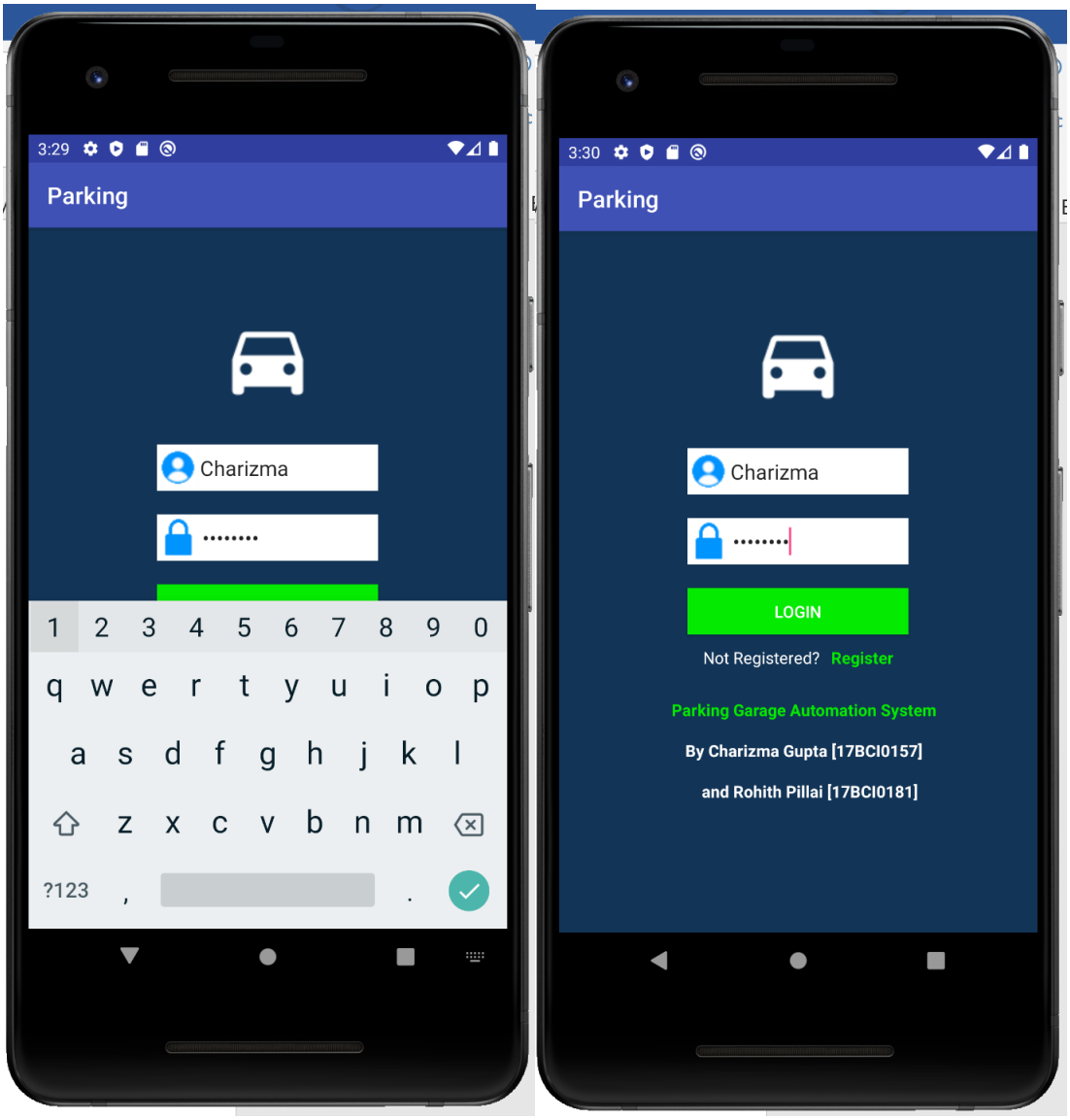




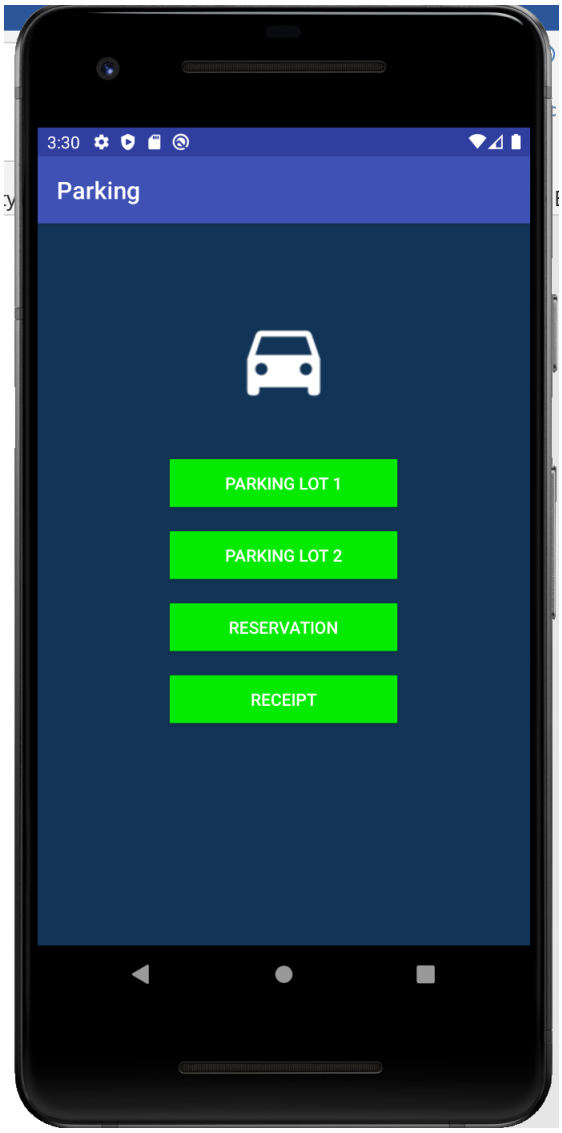
Login


2:45
Parking
Username
Password
LOGIN
Not Registered? Register
Parking Garage Automation System
By Charizma Gupta [17BCI0157]
and Rohith Pillai [17BCI0181]

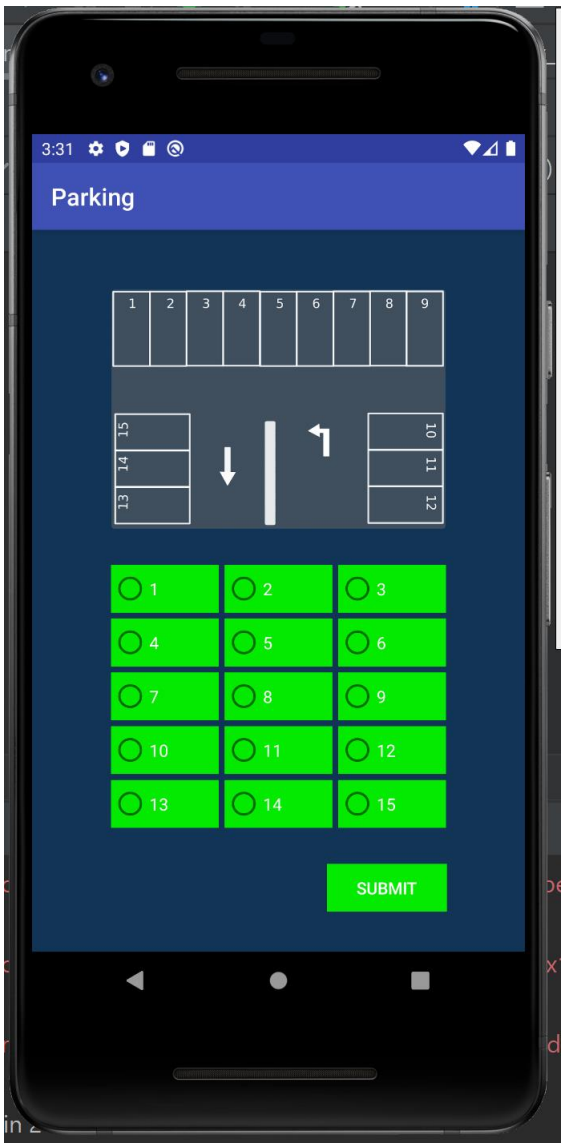

3:49
Parking
Charizma
LOGIN
Not Registered? Register
Parking Garage Automation System
By Charizma Gupta [17BCI0157]
and Rohith Pillai [17BCI0181]
Login Error

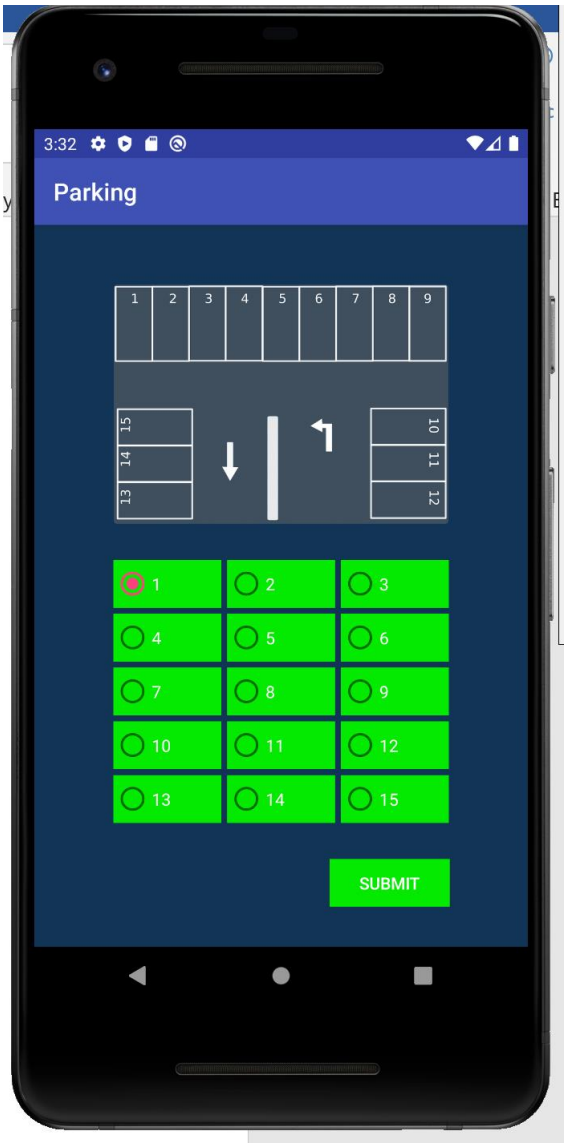

3:29
Parking
Charizma
1 2 3 4 5 6 7 8 9 0
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
3:30
Parking
Charizma
LOGIN
Not Registered? Register
Parking Garage Automation System
By Charizma Gupta [17BCI0157]
and Rohith Pillai [17BCI0181]

User Homepage

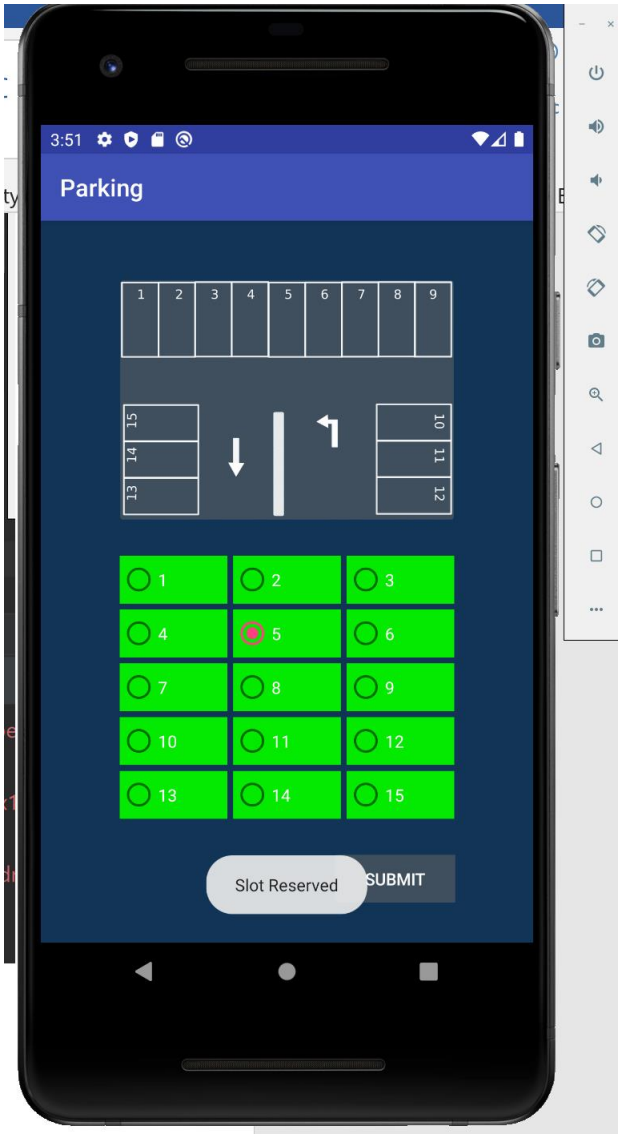


Select Parking Spot

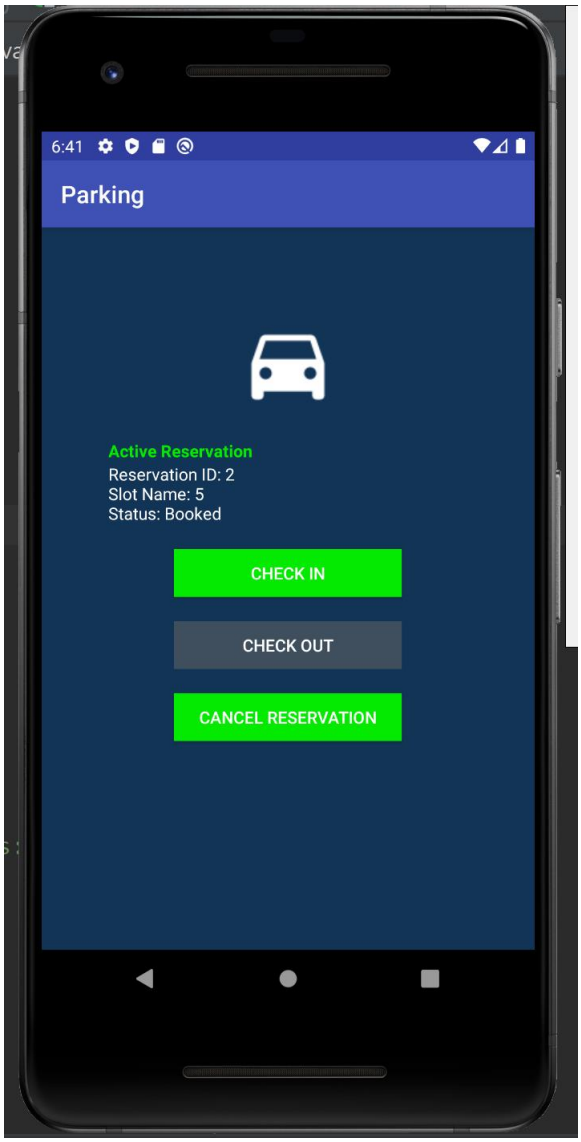




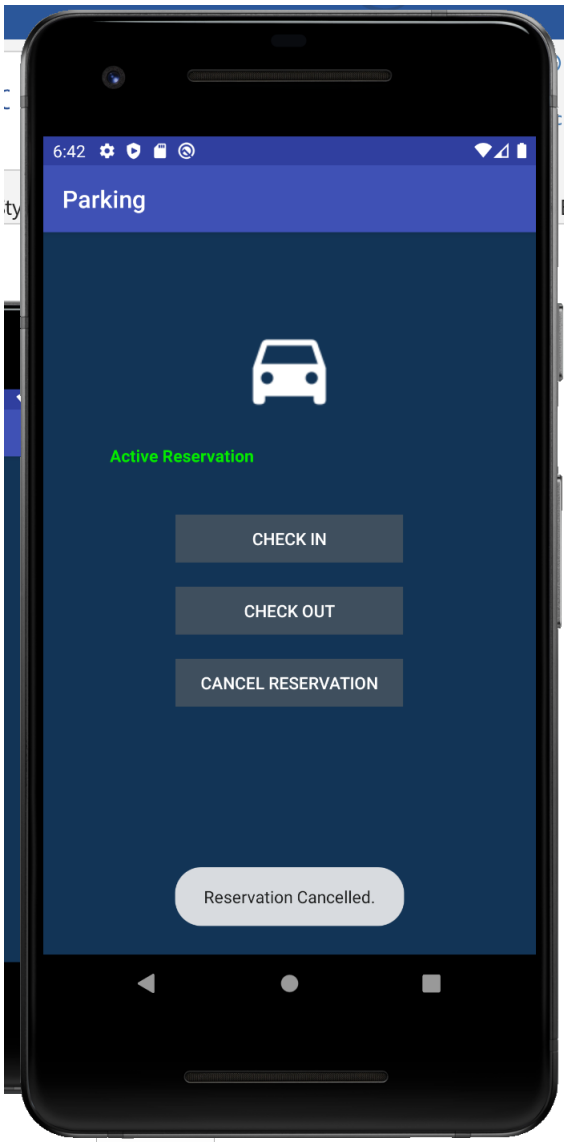

3:32
Parking
1 2 3 4 5 6 7 8 9
15 14 13
10 11 12
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
Slot Unavailable
UBMIT
3:34
Parking
1 2 3 4 5 6 7 8 9
15 14 13
10 11 12
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
SUBMIT


3:51
Parking
1 2 3 4 5 6 7 8 9
15 14 13
10 11 12
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
Slot Reserved
SUBMIT

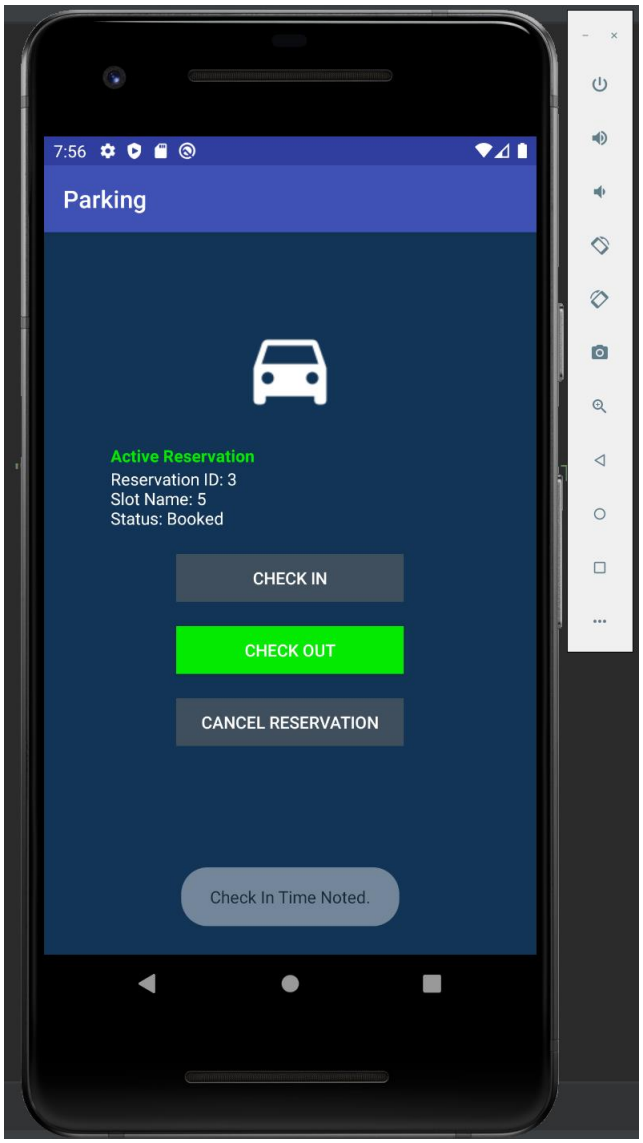
Reservation Module



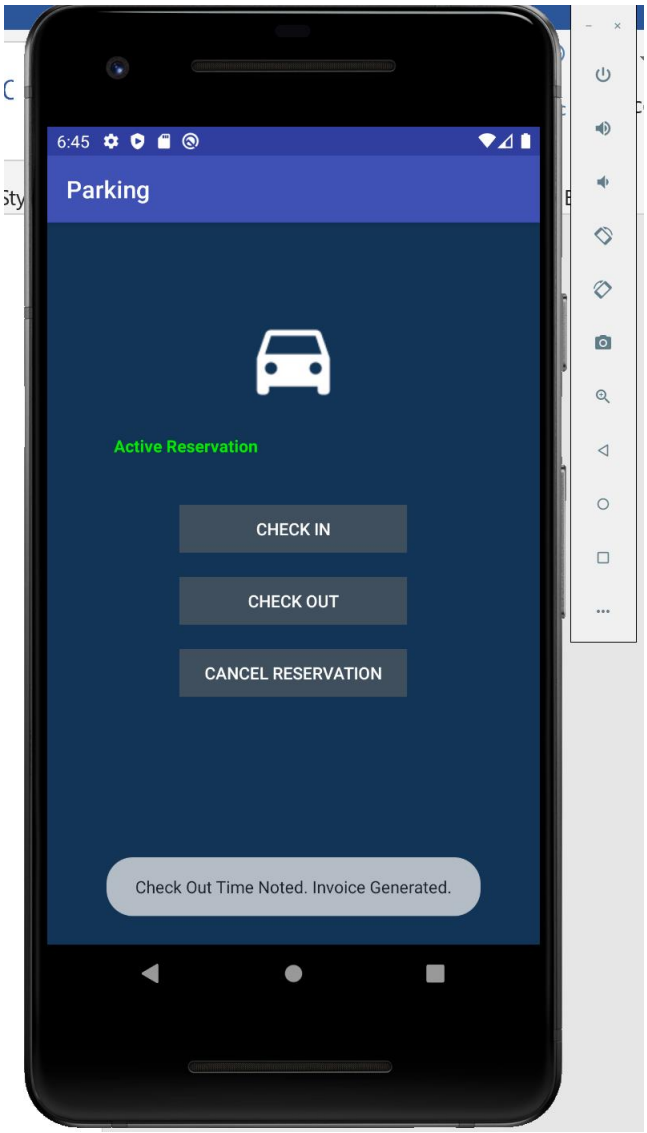
Cancelling Reservation



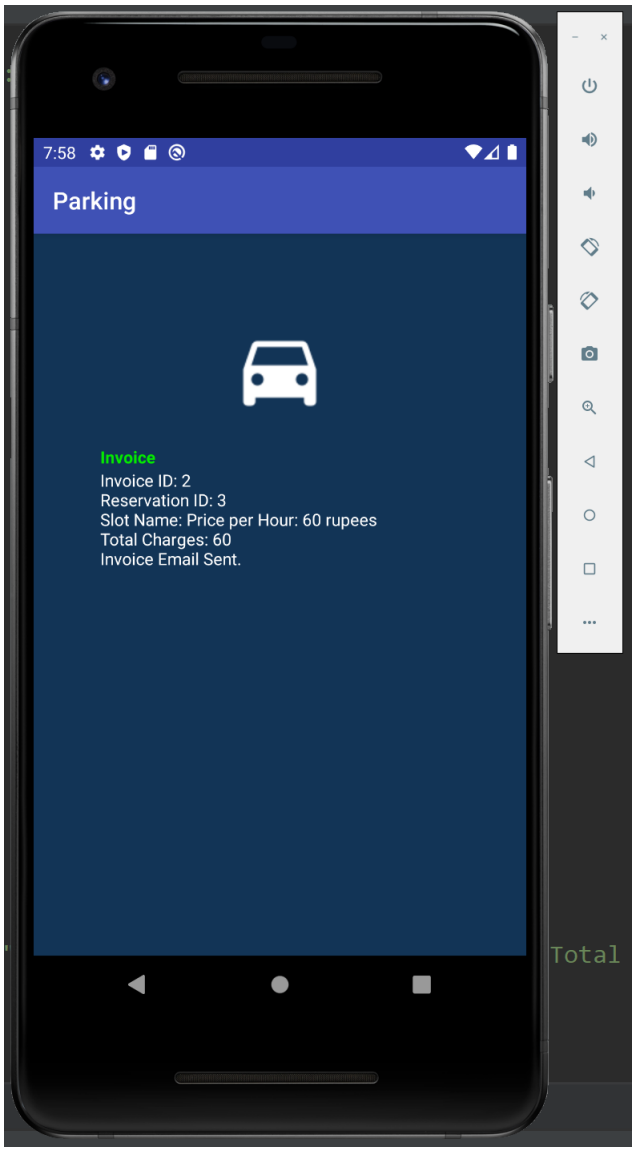
Check-In


7:56
Parking
Active Reservation
Reservation ID: 3
Slot Name: 5
Status: Booked
CHECK IN
CHECK OUT
CANCEL RESERVATION
Check In Time Noted.

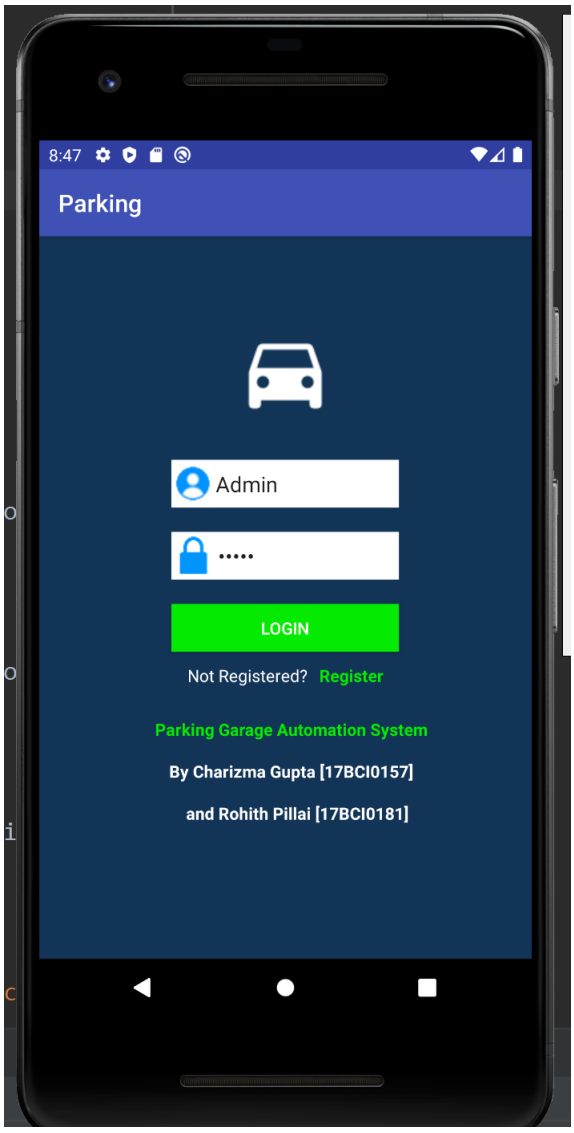
Check-Out


6:45
Parking
Active Reservation
CHECK IN
CHECK OUT
CANCEL RESERVATION
Check Out Time Noted. Invoice Generated.

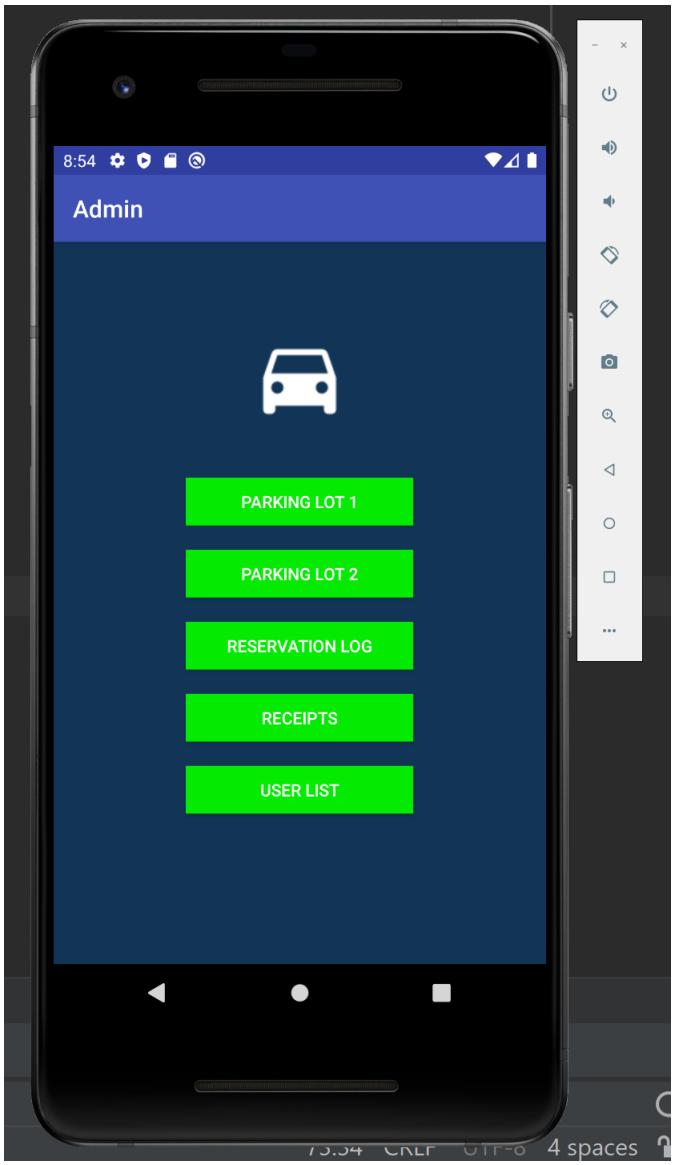
Invoice Generation



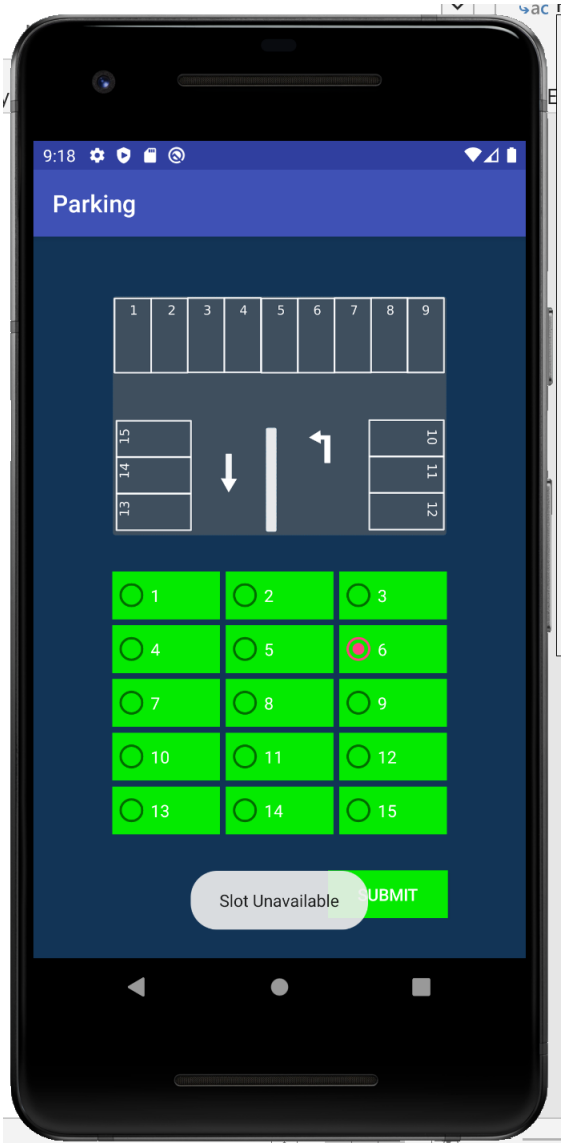
Admin

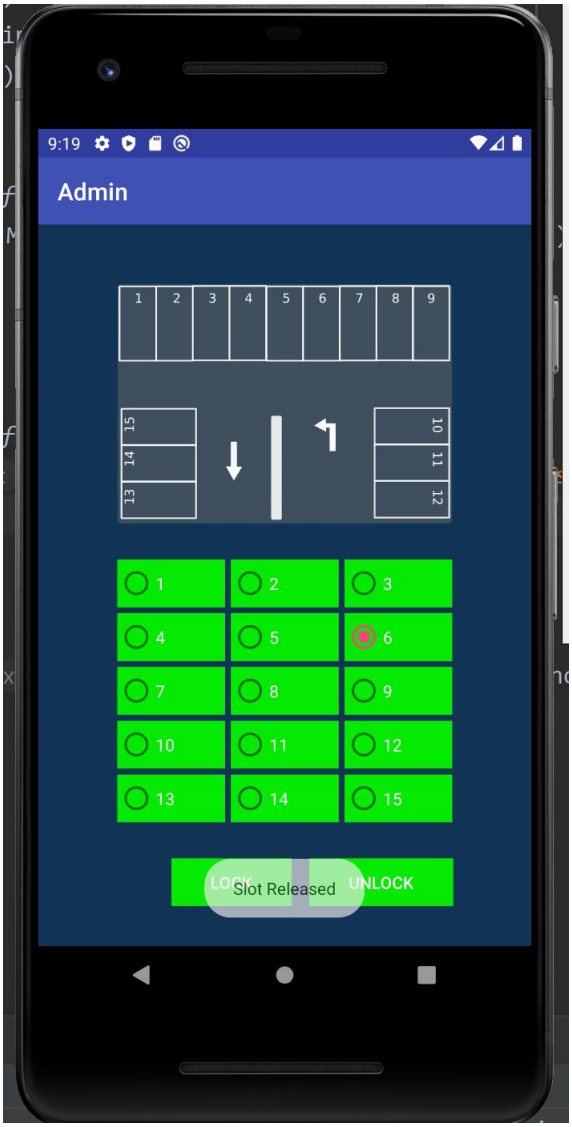

8:47
Parking
Admin
LOGIN
Not Registered? Register
Parking Garage Automation System
By Charizma Gupta [17BCI0157]
and Rohith Pillai [17BCI0181]

Admin Homepage



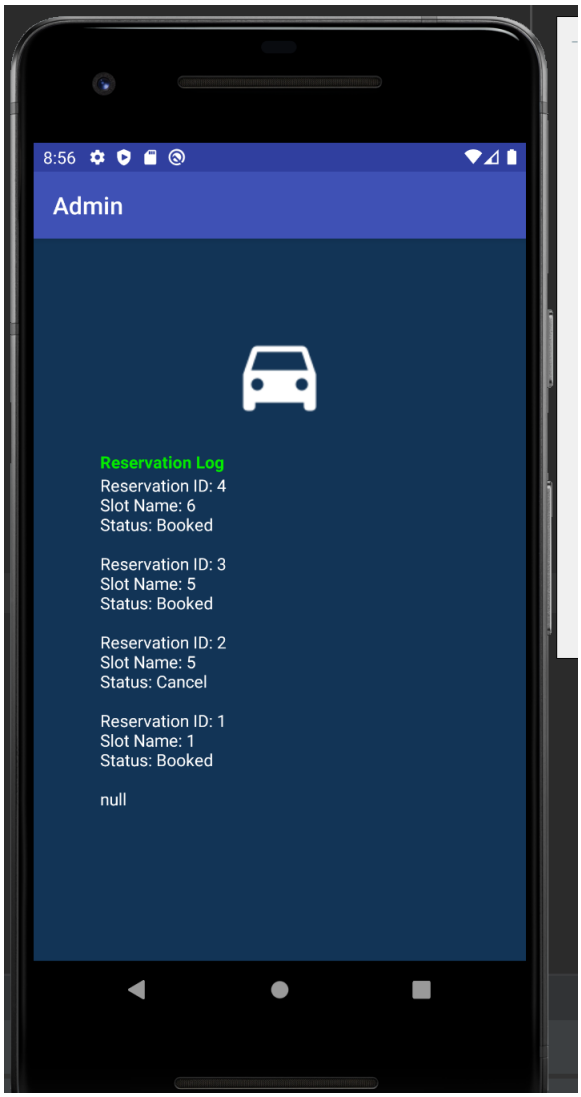
Lock/Unlock Slots


9:18
Parking
1 2 3 4 5 6 7 8 9
15 14 13
10 11 12
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
Slot Unavailable
UBMIT

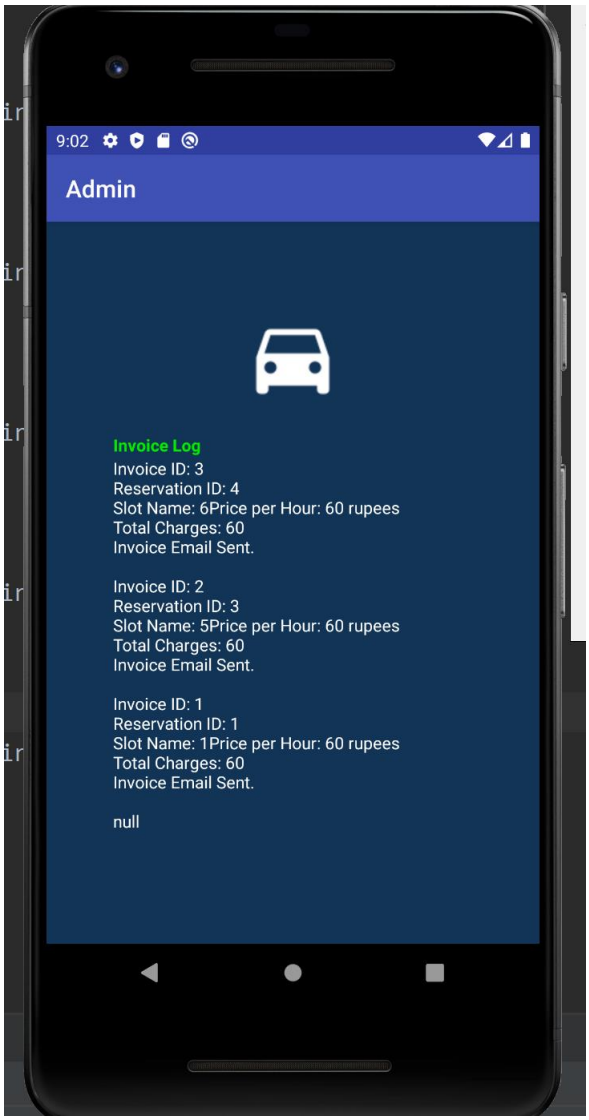

9:19
Admin
1 2 3 4 5 6 7 8 9
15 14 13
10 11 12
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
Slot Released
UNLOCK


9:21
Parking
1 2 3 4 5 6 7 8 9
15 14 13
10 11 12
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Slot Reserved
SUBMIT

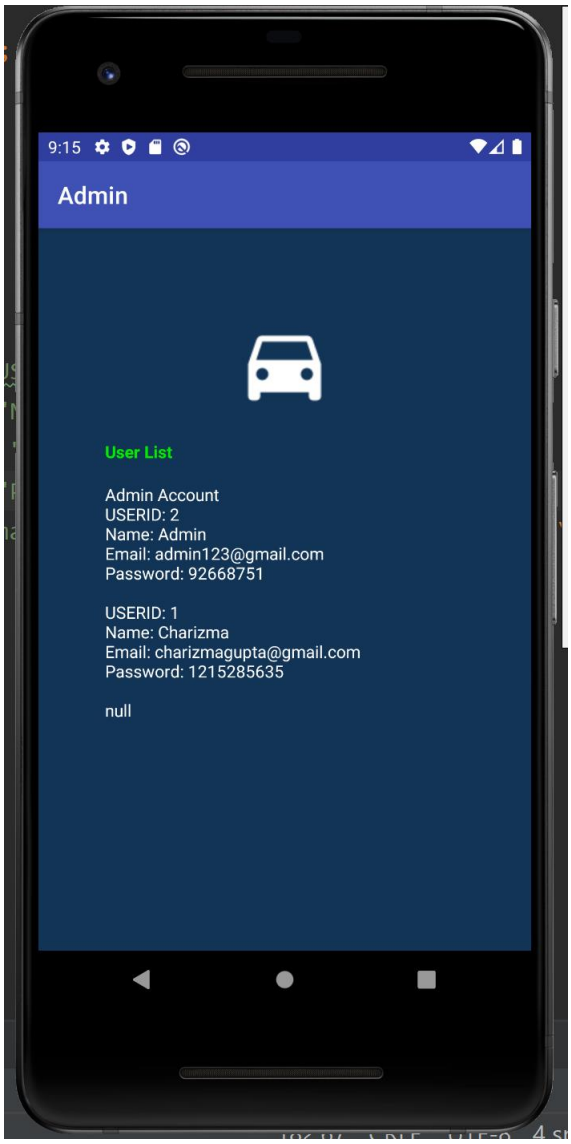
Reservation Log



Receipts


9:02
Admin
Invoice Log
Invoice ID: 3
Reservation ID: 4
Slot Name: 6Price per Hour: 60 rupees
Total Charges: 60
Invoice Email Sent.
Invoice ID: 2
Reservation ID: 3
Slot Name: 5Price per Hour: 60 rupees
Total Charges: 60
Invoice Email Sent.
Invoice ID: 1
Reservation ID: 1
Slot Name: 1Price per Hour: 60 rupees
Total Charges: 60
Invoice Email Sent.
null

User List


9:15
Admin
User List
Admin Account
USERID: 2
Name: Admin
Email: admin123@gmail.com
Password: 92668751
USERID: 1
Name: Charizma
Email: charizmagupta@gmail.com
Password: 1215285635
null