

ZippyPark

An Optimized Automated Parking Garage

14:332:452 Software Engineering

Project Report 3

Team 2

Samantha Moy

Atmika Ponnusamy

Samantha Cheng

Kylie Chow

Andrew Ko

Parth Patel

Shreya Patel

Nandita Shenoy

Piotr Zakrevski

Submitted May 7, 2020

CONTRIBUTION BREAKDOWN

All team members contributed equally.

Contents

SUMMARY OF CHANGES	6
Section 1: CUSTOMER STATEMENT OF REQUIREMENTS	8
1a) Problem Statement	8
1b) Proposed Solution	9
Section 2: GLOSSARY OF TERMS	13
Section 3: SYSTEM REQUIREMENTS	14
3a) Enumerated Functional Requirements	14
3b) Enumerated Nonfunctional Requirements	16
3c) User Interface Requirements	16
Section 4: FUNCTIONAL REQUIREMENTS SPECIFICATION	18
4a) Stakeholders	18
4b) Actors and Goals	19
4c) Use Cases	20
i) Casual Description	20
ii) Use Case Diagram	22
iii) Traceability Matrix	23
iv) Fully-Dressed Descriptions	24
4d) System Sequence Diagrams	30
Section 5: EFFORT ESTIMATION USING USE CASE POINTS	36
5a) Unadjusted Actor Weight (UAW)	36
5b) Unadjusted Use Case Weight (UUCW)	37
5c) Technical Complexity Factor (TCF)	37
Section 6: DOMAIN ANALYSIS	40
6a) Domain Model	40

i) Concept Definition	41
ii) Association Definitions	42
iii) Attribute Definitions	43
iv) Traceability Matrix	45
6b) System Operation Contracts	45
Section 7: INTERACTION DIAGRAMS	48
Section 8: CLASS DIAGRAM AND INTERFACE SPECIFICATION	54
8a) Class Diagram	54
8b) Data Types and Operation Signatures	54
8c) Traceability Matrix	59
8d) Design Patterns	61
8e) Object Constraint Language Contracts	62
Section 9: SYSTEM ARCHITECTURE AND SYSTEM DESIGN	63
9a) Architectural Styles	63
9b) Identifying Subsystems	63
9c) Mapping Subsystems to Hardware	64
9d) Persistent Data Storage	65
9e) Network Protocol	66
9f) Global Control Flow	66
9g) Hardware Requirements	67
Section 10: ALGORITHMS AND DATA STRUCTURES	69
10a) Algorithms	69
10b) Data Structures	72
Section 11: USER INTERFACE DESIGN & IMPLEMENTATION	74
Section 12: DESIGN OF TESTS	89

Section 13: HISTORY OF WORK, CURRENT STATUS, & FUTURE WORK	104
13a) History of Work	104
13b) Current Status	110
13c) Future Work	112
Section 14: REFERENCES	113

SUMMARY OF CHANGES

All sections:

The report language of each section was updated to reflect the following policy changes:

- QR code → barcode
- Penalty-reward system → loyalty program
- License plate scanned/identified → barcode scanned/identified
- Confirmation email → confirmation pop-up in app
- Payment deducted from balance → notification of charge
- Spot assigned when reservation made → spot assigned when customer scans in
- Describes automatic pricing tool found on manager website

--

Section 1: Customer Statement of Requirements

- Reworded to make more concise

Section 2: Glossary of Terms

- Changed “penalty-reward system” to “loyalty program” (same definition)

Section 3: System Requirements

- Removed irrelevant requirements (REQ 6, 7, 10, 16, 29, 36)
- Renumbered requirements

Section 4: Functional Requirements

- Added missing actors
- Emphasized manager and scanner interface (*Report 1 feedback*)
- Added fully-dressed descriptions for final demo UCs
- Added system sequence diagrams for final demo UCs
- Updated system sequence diagrams with alternate scenarios (*Report 1 feedback*)

Section 5: Effort Estimation Using Use Case Points

- Calculated duration with $PF = 28$

Section 6: Domain Analysis

- Edit domain model so that methods/functions are not shown as elements of concepts (*Report 1 feedback*)
- Reformat Attribute Definition table

- Add system operation contracts for final demo UCs

Section 7: Interaction Diagrams

- Updated interaction diagrams with alternate scenarios (*Report 2 feedback*)
- Emphasized design principle terms in diagram descriptions (*Report 2 feedback*)

Section 8: Class Diagram and Interface Specification

- Updated class diagram and descriptions
- Clarified class evolution from domain concepts (*Report 2 feedback*)
- Added Section 2d (Design Patterns)
- Added Section 2e (OCL Contracts)

Section 9: System Architecture and System Design

- Updated list of database tables
- Emphasized that special hardware is simulated by GUIs (*Report 1 feedback*)

Section 10: Algorithms and Data Structures

- Added content from Report 1 Mathematical Model section
- Clarified differentiation between manually and automatic pricing tools

Section 11: User Interface Design and Implementation

- Add additional Customer UI images, descriptions, and worst-case user effort
- Add Scanner UI images, descriptions, and worst-case user effort
- Add Manager UI images, descriptions, and worst-case user effort
- Ensure all user effort scenarios are worst case (*Report 1 feedback*)
- Updated all screenshots

Section 12: Design of Tests

- Updated to reflect tests actually executed (UC 1-3, 6, 7, 11, 12; REQ-49)

Section 13: History of Work, Current Status, and Future Work

- Added updated management procedures
- Updated technologies used
- Updated timeline and calendar
- Added new future work items

Section 14: References

- Alphabetized references

Section 1: CUSTOMER STATEMENT OF REQUIREMENTS

1a) Problem Statement

Parking garages are a modern development created to make parking easier and more convenient for drivers who do not want to circle the streets for open parking. However, the current system of parking garages has not been optimized to mitigate the full frustrations of parking for customers.

Not many of today's parking garages operate with the aid of computerized systems to manage garage occupancy. Instead they utilize a system where employees walk around to inspect the occupancy of individual spots. Those that do monitor occupancy with a sensor system simply keep count of vehicles that enter and exit. Thus, there exists the issue of searching for a vacant spot once a customer has entered the garage. For many customers, locating an unoccupied parking spot is a troublesome and tiring process. According to a report published in 2017 by INRIX Research, an analytics company that provides parking and traffic data, Americans spend on average 17 hours per year searching for parking [5]. Taking into account fuel and emissions, the resulting cost of this wasted time is calculated to be \$345 per driver [5]. The grueling process of circling floors upon floors of a parking garage causes frustration for customers. Furthermore, garages nationwide currently utilize a parking system where customers pay after parking based on the length of their stay. This payment occurs during the process of leaving the garage, which leads to the issue of slowdown at the exit of the garage. Overall, under current parking garage systems, the process of entering and exiting the garage is slow and tiresome for customers. This is counterproductive of the purpose of parking garages which is to provide faster and more convenient parking.

These issues can be alleviated through a more streamlined process which can be provided by an automated parking garage system. By reducing the hassle of locating empty parking spots and eliminating the need to pay at departure, parking at garages will be made quicker and easier, lessening the burden on customers and increasing customer satisfaction.

1b) Proposed Solution

Description

ZippyPark aims to alleviate the frustrations of old parking garages with an enhanced automated parking system that can be implemented in any garage. The following product was designed to provide customers with a simple and streamlined experience, while also helping a owner to better manage the garage.

The process of parking at a garage can first be revolutionized by two basic features. An online reservation system gives customers the ability to secure a guaranteed parking spot. A system to serve customers without a reservation, also gives customers the flexibility of being a “walk-in.” Both features lower traffic congestion and the customer’s parking spot search time, because the elevator directs them to the exact spot systematically selected for them by the system. The system contains sensors at each parking spot to track the garage occupancy. This information is used by the system to choose the closest spot within the customer’s parking section (general reservation, VIP reservation, handicap reservation, walk-in).

To begin, all customers (reservation or walk-in) must download the mobile application and create an account to use our product, similar to the business model of famous companies such as Costco Wholesale Corporation and Uber Technologies, Inc. [3][11]. A valid credit card must be entered during account creation to streamline the payment process. The account will give a customer the ability to create, edit, delete, and view current or past reservations; view, edit, and delete profile information; and display the barcode associated with their account.

The reservation process begins with a customer reserving a spot in the parking garage for an allotted time. The fee will be charged immediately after creating the reservation. Upon arrival, a scanner will scan the customer’s barcode, the system will assign a parking spot, and the elevator will take the customer to the correct location. The system will never allow the number of reservations from a time exceed the number of spots available in the reservation parking section, to ensure all customers who reserve a spot will have a spot on their arrival. The customer must also scan their barcode when exiting so that the system can calculate extra fees if they stay past their reservation time.

The walk-in process begins when a customer arrives and scans in with their barcode. The process of assigning a parking spot is the same. However, the entire price calculation is

determined when the customer scans out. The customer will be charged for time between scan-in and scan-out; no extra fee calculations are needed.

This system design will create an easy and flexible process for parking in a garage. However, there are several novel features ZippyPark includes in the hope of further enhancing the parking garage experience.

Novelty

Existing projects have aimed to solve the issues of modern parking garage systems, through features such as dynamic pricing [1] and barcode implementation [2]. However, to create a new automated garaging system, these ideas have been improved and built upon with the goal of assisting both the customer and parking garage management in novel ways. These features, some of which have been mentioned in the product description, include: (1) a loyalty program, (2) handicap accessible parking, (3) dynamic pricing, and (4) barcode usage. The key novel item of this project is (5) the manager portal.

(1) A new functional feature is a loyalty program that will encourage customers to obey the regulations of the parking garage and reward preferred behavior. Good behavior will include general use of the parking garage, showing up to reservations on time, and making reservations rather than walking in. Conversely, bad behavior will include staying past a reservation. Customers will not be reprimanded for arriving late to or missing a reservation since they have already paid for the reservation and may use the spot as they would like. Each act of good or bad behavior results in points being given/taken away from a user's score. Customers may spend their points on awards, such as access to VIP parking spots. Managers may also use the balance to determine if a customer deserves penalties for extensive bad behavior (i.e. a negative point balance indicates repeated offenses). Repeated offenses may result in more severe penalties such as a restriction of walk-in privileges or loss of the ability to use the garage. This system aims to establish a larger number of frequent, loyal customers who consistently exhibit ideal behavior.

(2) A qualitative feature of our garage is accommodations for handicapped patrons. Due to the size of the garage, customers with physical disabilities may struggle to reach nearby exits. Our aim is to provide an extra option to help handicapped individuals without sacrificing user-interface simplicity. During registration, new customers may request handicap arrangements so that their reserved spot will be closer to the entrance/exit. Their profiles will reflect this

information, and future reservations will automatically reserve handicap parking spots. This feature guarantees the customer will always have accessible parking without the need for further action. Ideally, the addition of handicap parking spots will improve customer loyalty without ruining the user experience. The handicap component is also a feature of the 2019 group's project, but is improved upon by ensuring handicapped spots are automatically reserved/selected for disabled patrons [4].

(3) An additional functional feature is a dynamic pricing model which determines parking-rates based on the time of day and the demand for walk-in reservations. A dynamic pricing system allows owners of the parking garage to maximize their profits by charging a premium on the regular parking-rate when conditions permit. The factors this system specifically uses to determine price multipliers are the time of day—to raise prices during rush hour and other peak driving hours—and the volume of customers requesting parking spots. This pricing system would affect the rates for confirmed reservations, guaranteed reservations, and walk-in reservations uniquely. This component is a feature of the 2018 group that we will be enhancing by improving the manager's ability to regulate the pricing scheme [1].

To determine the dynamic parking rate, the minimum parking rate will be multiplied by a fee multiplier which will be predetermined by the manager. The multiplier will be particularly high during rush hours and weekends. Local traffic data may also be used to determine these rates, as well as statistical data regarding the use of the parking garage itself. Customers who then elect to make confirmed reservations would pay the sum of the dynamic parking fees for their time slot. To prevent unfair manipulation of the pricing system, there will be a set length of time for which reservations can be made in advance. For example, customers may not be allowed to make reservations more than one year in advance since managers do not know how the pricing fees may need to change to account for changes like inflation.

This pricing policy will maximize profits for the parking spot because it will pick up on sporadic surges in demand for parking and charge drivers accordingly. The system will also reduce the overall parking fees for particularly low frequencies of walk-in requests to encourage more drivers to use the parking garage. By lowering fees, the number of customers at the parking garage is more likely to be maximized which in turn raises profits.

(4) Another qualitative feature we propose is a new barcode tool associated with every profile. When the customer creates an account, they will receive a barcode associated with their

profile which they can find in the mobile app at any time. Even in the instance where a registered customer has a rental car, the customer may show the scanner their barcode. The garage will then be able to identify the customer and let them enter. The time that the customer saves, especially for those who frequently rent cars, can lead to increased customer satisfaction. A similar barcode feature was utilized by the 2019 group. However, their feature had customers scan a QR code presented by the garage for registration, while ours has the garage scan a user's code for entry [4]. We improve upon this feature through a more user-friendly experience where the customer does not have an extra step of scanning a code in order to park.

(5) Past projects have had components that give managers some administrative features, such as the ability to view all reservations [1]. However, we improve upon this idea with the manager portal, a website that compiles many of these features into one centralized hub. From this website, managers have basic administrative capabilities, such as the ability to create, edit, delete, and view current or past reservations for all customers; as well as create, edit, delete, and view all customer accounts. There is also a page where they can adjust the pricing scheme and loyalty program point scheme either manually or using a scheme that the system suggests based on past usage trends. On the statistics page, managers can view various data and graphs about the garage's customers and usage, which will help to decide changes in the pricing and points scheme. Finally, on the dashboard, managers can see quick information about the garage. The manager portal brings together many helpful tools for managers or owners, which will allow for more informed decisions and increase management efficiency.

Overall, a fully automated garaging system with these enhancements can eliminate the need for parking attendants and optimize the management of occupancy. Furthermore, it provides a more reliable toll system and user-friendly search system for parking spots. Such a parking system will surely increase customer satisfaction and manager productivity.

Section 2: GLOSSARY OF TERMS

- **Account:** Customers can access the garage app using their username and password. Each customer has their own account, where they can manage information and reservations.
- **Customer:** A person who interacts with the garage app to purchase its services.
- **Database:** A part of the overall server, where all of the customer account data and information is recorded and stored.
- **Elevator:** A lift for cars that will transport them to the appropriate floor. An elevator is located at both the entrance and exit.
- **Manager:** A person who interacts with the garage app to manage its services.
- **Manager Portal:** Another name for the manager website.
- **Owner:** A person who the garage belongs to.
- **Point(s):** A reward for customers for using the parking garage services. Points may be rewarded, deducted, and spent by customers for rewards.
- **Profile:** A part of the account. Here, customer information is stored.
- **Reservation:** An arrangement to park in a spot during a fixed, future time-frame. It is guaranteed that customers will be able to park in a spot if a reservation is made.
- **Loyalty program:** A system where customers receive points for good behavior, such as making reservations, and lose points for bad behavior, such as leaving a reservation late. Points may be redeemed for parking in the VIP section. Managers may use the balance to determine if a customer deserves penalties for extensive bad behavior.
- **Scanner:** A camera located at the entrance of the garage that can recognize and read barcodes. Simulated by GUIs and text entry for the purposes of this project.
- **Spot:** An individual parking space where one car can park at a time.
- **System:** The overall parking garage project; includes the physical garage, the app, the different sensors, and the software that binds them all together.
- **User:** Any person who interacts with the software.
- **Walk-in:** Customers who have not reserved a parking spot. Unlike reservations, walk-in spot availability is not guaranteed.
- **Weight Sensor:** Located at every parking spot, this sensor determines if a car is currently located at the given spot.

Section 3: SYSTEM REQUIREMENTS

3a) Enumerated Functional Requirements

Identifier	Priority	Requirement
REQ-1	5	The system shall allow the customer to create an account on the app.
REQ-2	4	The system shall allow the customer to edit their own information in their accounts.
REQ-3	5	The system shall allow the customer to make a reservation or cancel a reservation.
REQ-4	2	The system shall send the customer a confirmation of their reservation.
REQ-5	4	The system shall display the vacant spot that the customer will park at when they arrive at the entrance.
REQ-6	3	The system shall be able to scan the barcode.
REQ-7	4	The system shall mark the time the customer enters and exits the lot.
REQ-8	2	The system shall automatically charge a user for making a reservation.
REQ-9	2	The system shall automatically charge a walk-in user when they exit the garage.
REQ-10	4	The system shall keep track of the occupancy of the walk-in section to determine if there are still available walk-in spots.
REQ-11	4	The system shall automatically update points when the customer makes a reservation or arrives on time.
REQ-12	4	The system shall automatically deduct points when the customer overstays their reservation or exchanges them for a VIP spot reward.
REQ-13	4	The system shall allow the garage manager to view all the reservations.

REQ-14	3	The system shall allow the garage manager to edit or cancel any reservations.
REQ-15	2	The system shall allow the garage manager to be able to adjust the pricing system.
REQ-16	2	The system shall allow the garage manager to be able to adjust the point system.
REQ-17	2	The system shall allow the garage manager to adjust the point thresholds for rewards.
REQ-18	5	The system shall keep statistics of the number of reservations made, reservations cancelled, and number of cars entering the lot.
REQ-19	1	The system shall allow the garage manager to access and view the statistics of the parking lot.
REQ-20	5	The system shall save the customer's information in the database.
REQ-21	4	The system shall allow the garage manager to view the information in the database.
REQ-22	3	The system should allow the garage manager to edit the information in the database.
REQ-23	1	The system shall allow the garage manager to add, edit or delete a customer account

Table 1. Enumerated Functional Requirements.

3b) Enumerated Nonfunctional Requirements

Identifier	Priority	Requirement
REQ-24	3	The system will have access to the internet in order to connect with customers.
REQ-25	2	The system will have weight sensors at each parking spot to determine a vacancy.
REQ-26	3	The system will allow for disability to be taken into account while assigning parking spots.
REQ-27	5	The system will store database information in a file system.
REQ-28	5	The system will use an app that customers can download to create accounts, make reservations, and access any information about their account.
REQ-29	1	The system will use an elevator system to guide cars towards open spots. An elevator will also be available for only leaving cars.
REQ-30	3	The system will use barcodes to handle assigning customers spots.

Table 2. Enumerated Nonfunctional Requirements.

3c) User Interface Requirements

Identifier	Priority	Requirement
REQ-31	5	After first downloading the app, the customer will be required to create an account.
REQ-32	5	Upon opening the app, the customer will be required to login with their username and password.

REQ-33	4	In the app, the customer will be able to click a button to go to reservation(s).
REQ-34	4	In the app, the customer will be able to click a button to view their profile.
REQ-35	3	In the app, the customer will be able to redeem points for a reward when making a reservation.
REQ-36	1	In the app, the customer will be able to sign out of their account.
REQ-37	2	In the app, the customer will be able to click a button to edit account information on the profile page.
REQ-38	3	In the app, the customer will be able to view their barcode on the profile page.
REQ-39	3	In the app, the customer will be able to indicate their handicap status on the profile page.
REQ-40	4	In the app, the customer will be able to make a reservation on the reservation page.
REQ-41	3	In the app, the customer will be able to edit their reservations on the reservation page.
REQ-42	3	In the app, the customer will be able to cancel reservations on the reservation page.
REQ-43	2	In the app, the customer will be able to save a credit card to their account on the profile page.
REQ-44	4	On the manager portal, managers will be able to view reservations.
REQ-45	3	On the manager portal, managers will be able to see the number of spots deemed available by the scanner.

REQ-46	1	On the manager portal, managers will be able to check garage statistics over time.
REQ-47	3	On the manager portal, managers will be able to view all registered customers.
REQ-48	2	On the manager portal, managers will be able to view and edit the parking price for a specified time period.
REQ-49	3	On the manager portal, managers will be able to view and edit the loyalty program point scheme.
REQ-50	2	The customer will be able to move to an assigned spot deemed vacant by the scanner.
REQ-51	1	The scanner will sense time-elapsd for a walk-in or reservation overtime and customers will be billed accordingly.

Table 3. User Interface Requirements.

Section 4: FUNCTIONAL REQUIREMENTS SPECIFICATION

4a) Stakeholders

ZippyPark is intended to increase the efficiency of operation of parking garages, and therefore, maximize the profits. This system will be of interest to people and organization who own, operate, or maintain parking garages, as well as individuals who purchase the services of parking garages. Possible stakeholders include:

- Parking garage owners
- Parking garage managers
- Customers – reserved or walk-in

4b) Actors and Goals

Actors are denoted as the following types:

I = initiating

S = supporting

O = offstage

Actor	Type	Goal	Use Case(s)
Customer, Garage Manager	I	To create, edit, or delete customer account by clicking the applicable buttons on the parking garage app	UC-1, UC-2, UC-3
Mobile App	S	To collect and display customer information	
Manager Portal	S	To display existing customer information and collect new or updated information	
Database	O	Stores updated customer information	
Customer, Garage Manager	I	To create, edit, or delete customer reservation by clicking on the applicable buttons on the parking garage app	UC-4, UC-5, UC-6
Mobile App	S	To collect and display reservation information (time, date)	
Manager Portal	S	To display existing customer reservations and collect new or updated reservation information	
Database	O	Stores updated reservation information	
Customer	I	To enter or exit the garage	UC-7, UC-8
Scanner	S	To scan and read the customer's account barcode	
Database	O	Searches for the customer's barcode to identify a reservation; finds what floor to send the car to and sends this information to elevator	
Elevator	S	To bring customer to correct floor	
Customer	I	To park or pull out of parking spot	UC-9
Database	O	Changes parking spot status depending on sensor output	

Garage Manager	I	To view parking garage statistics on the manager portal	UC-10
Manager Portal	S	To retrieve parking garage statistics from database and display on manager portal	
Database	O	Stores parking garage statistics	
Customer	I	To view points	UC-11
Garage Manager	I	To view and update customers' points	
Mobile App	S	To display customer points as requested	
Manager Portal	S	To display existing customer points and set point scheme.	
Database	O	Stores customer points	UC-12
Customer	I	To pay for parking reservation	
Mobile App	S	To collect customer payment information including credit card number and billing address and display reservation price.	
Scanner	S	To display final payment upon exit of garage.	
Manager Portal	S	To set payment scheme.	
Database	O	Stores customer payment information and pricing scheme.	

Table 4. Actor roles, types, goals, and associated use cases.

4c) Use Cases

i) Casual Description

In this section, use cases marked with an asterisk will be highlighted in the final demo.

	Use Case	Description
UC-#1	Create Account	Utilizes REQ-1, REQ-18, REQ-19, REQ-22, and REQ-23. Allow new customers to register for the first time through the mobile application. Garage managers can also add customers

		by filling out a form on the customer information tab on the manager website.
UC-#2	Edit Account	Utilizes REQ-2, REQ-21, REQ-22, and REQ-23. Allow current customers to update account information. Garage managers can update account information through the customer information tab on the manager website.
UC-#3	Delete Account	Utilizes REQ-22 and REQ-23. If needed, garage managers can delete a customer account through the customer information tab on the manager website.
UC-#4	Create Reservation	Utilizes REQ-3, REQ-4, REQ-13, and REQ-17. Customers can create reservations for the parking garage. Customers receive confirmation pop-up in the mobile application upon creation. Managers are able to view customer reservations, as well as create a new reservation, by filling out a form on the reservation tab of the manager website.
UC-#5	Edit Reservation	Utilizes REQ-3 REQ-14, REQ-17, and REQ-21. Customers can edit an existing reservation through the mobile application, and managers through the reservation tab of the manager website.
UC-#6	Cancel Reservation	Utilizes REQ-3, REQ-14, REQ-15, and REQ-17. Customers can cancel an existing reservation through the mobile application, and managers through the reservation tab of the manager website.
UC-#7	Enter Garage	Utilizes REQ-5, REQ-6, REQ-7, and REQ-17. As the customer enters the garage, the scanner identifies the barcode and determines if they have a reservation or are a walk-in. The customer is assigned a spot and enters the elevator, which takes the customer to the floor where the assigned spot is located.
UC-#8	Exit Garage	Utilizes REQ-7. Elevator brings the car to the main floor for exit. As the customer exits the garage it records the time and adds it to their reservation history
UC-#9	Update Spot Status	Utilizes REQ-9 and REQ-17. Records in the database whether a parking spot is occupied or not.
*UC-#10	Display Stats	Utilizes REQ-10, REQ-13, REQ-17, REQ-18, REQ-20 and REQ-21. Retrieves information from the database and displays updated trends to managers on the statistics tab of the manager website.

*UC-#11	Points Management	Utilizes REQ-11, REQ-12, REQ-15, REQ-16, and REQ-17. Customers can view their points through the mobile application and exchange points for VIP parking spots when making a reservation. Managers can adjust the points reward scheme through the manager website.
*UC-#12	Payment	Utilizes REQ-8, REQ-9, REQ-15. If a customer makes a reservation, the fee should be charged immediately as it is submitted. They receive a confirmation pop-up in the mobile application that shows that they have been charged. If the customer is a walk-in then the fee should be calculated and displayed on the scanner after they leave the garage. Also, if a reserved customer overstays the parking spot a late fee is charged and displayed on the scanner. Managers may alter the pricing scheme manually or using the price suggestion tool.

Table 5. Casual descriptions of use cases.

ii) Use Case Diagram

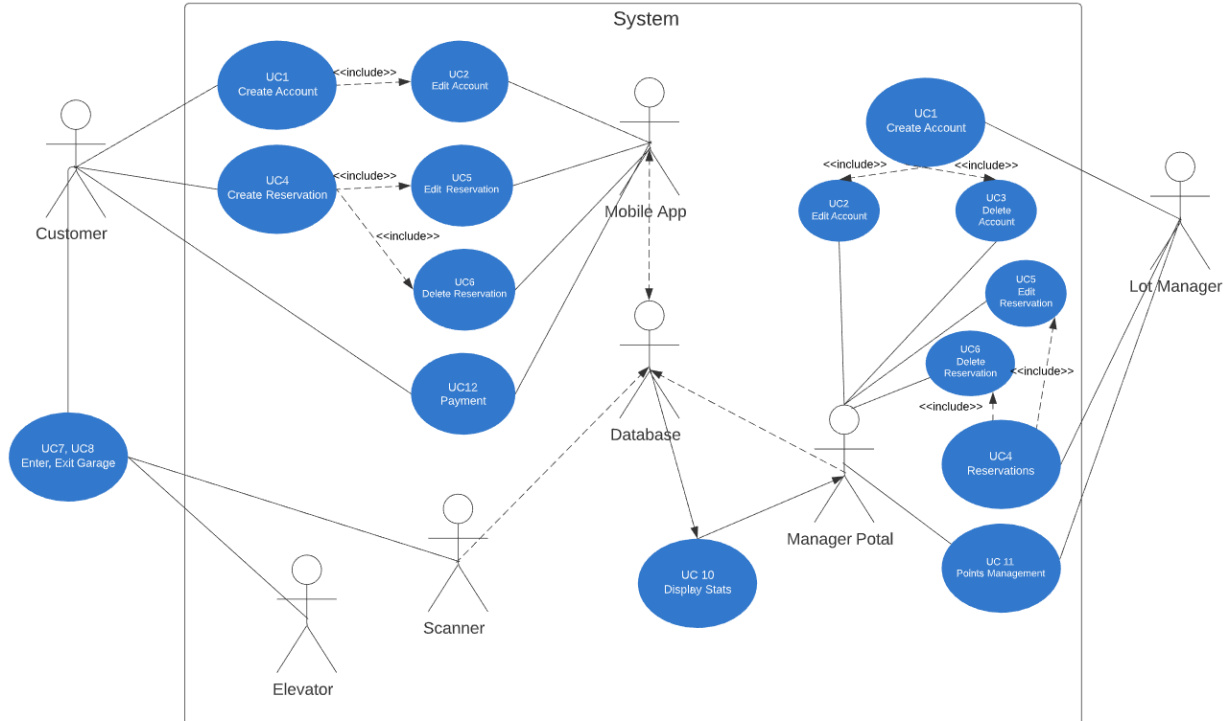


Figure 1. Diagram with all use cases.

iii) Traceability Matrix

This matrix maps the system requirements to the use cases. The priority weight (PW) of each requirement is given on a scale from 1 to 5, with 5 being the highest priority.

Requirement	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12
REQ-1	5	X											
REQ-2	4		X										
REQ-3	5				X	X	X						
REQ-4	2				X	X							
REQ-5	4							X					
REQ-6	3							X					
REQ-7	3							X	X				
REQ-8	3									X			X
REQ-9	4									X			X
REQ-10	4										X		
REQ-11	2											X	
REQ-12	2											X	
REQ-13	4				X						X		
REQ-14	4						X						
REQ-15	4						X					X	X
REQ-16	2											X	
REQ-17	4				X	X	X	X		X	X	X	
REQ-18	3	X									X		
REQ-19	2	X											
REQ-20	2										X		
REQ-21	5		X			X					X		
REQ-22	1	X	X	X									
REQ-23	1	X	X	X									
Max PW		5	5	1	5	5	5	4	3	4	5	4	4
Total PW		11	10	1	15	16	17	14	3	11	22	14	11

Table 6. Traceability matrix mapping requirements to use cases.

iv) Fully-Dressed Descriptions

According to the traceability matrix, the use cases with the highest priority weight are UC-4, 5, 7, 10. The fully-dressed descriptions of each are given below. The use cases highlighted in the final demo are UC-10, 11, and 12, which are marked by an asterisk in this section.

Use Case: UC-4 Create Reservation
<p>Related Requirements: REQ-3, REQ-14, REQ-13, REQ-17</p> <p>Initiating Actors: Garage Customer</p> <p>Actor's Goals: To reserve a parking spot to use for a set amount of time</p> <p>Participating Actors: ZippyParkApp, Database</p> <p>Preconditions:</p> <ul style="list-style-type: none">• The customer has an existing profile stored in the database• The customer has logged in to the ZippyParkApp• The ZippyParkApp displays a calendar for the current month as well as available time slots <p>Postconditions:</p> <ul style="list-style-type: none">• The system reduced the number of available spots for the reserved time• The new reservation is tied to the user's account reservations <p>Flow of events for main success scenario:</p> <p>→ 1.) Customer selects the "Reservations" item on the account page</p> <p>← 2.) System displays the Reservations page with a calendar for the current month</p> <p>→ 3.) Customer selects a date with start and end times</p> <p>← 4.) System checks with the Database to confirm availability</p> <p>← 5.) System:</p> <ul style="list-style-type: none">(a) saves the reservation to the customer's account,(b) reduces availability for the selected time, and(c) signals to the customer that reservation was successful <p>Flow of events for Extensions (Alternate Scenarios):</p> <p>4a. System finds time slot is invalid/filled</p> <p>← 1.) System informs customer that the requested time is unavailable</p> <p>→ 2.) Customer selects a valid date and time</p> <p>← 3.) Same as Step 4 above</p>

Table 7. Fully-dressed description of Use Case 4.

Use Case: UC-5 Edit Reservation

Related Requirements: REQ-3, REQ-18, REQ-21, REQ-25

Initiating Actors: Garage Customer, Garage Manager

Actor's Goals: To edit a reservation previously made with the parking garage

Participating Actors: ZippyParkApp/Website, Database

- The manager uses ParkingGarageWebsite and the customer ZippyParkApp
 - In this use case, both have the same capabilities with different people accessing them

Preconditions:

- At least one reservation for the customer of interest must exist in the system

Postconditions:

- An edited version of the reservation will take the place of the original in the database
- The original reservation is wiped from the database

Flow of events for main success scenario:

- 1.) User chooses option to edit a reservation of the ZippyParkApp/Website
- ← 2.) ZippyParkApp/Website communicates with Database and displays all upcoming reservations under customer of interest in chronological order
- 3.) User chooses the reservation they would like to edit and change the time of day, duration, or date of reservation
- 4.) ZippyParkApp/Website asks user if they would like to proceed as changes can't be undone
- ← 5.) User confirms and agrees to increased/decreased charges on the account of the customer of interest if the duration of the reservation was lengthened/shortened.
- 6.) ZippyParkApp/Website communicates with system to edit the reservation in the Database
- ← 7.) ZippyParkApp/Website sends a confirmation of completed editing

Flow of events for Extensions (Alternate Scenarios):

- ← 5a. The system determines that the request information cannot be edited
 - for some conflicting reason (e.g. conflicts with another reservation)
- ← 1.) ZippyParkApp /Website notifies user that the change cannot be made
- 2.) User chooses to cancel editing the reservation

Table 8. Fully-Dressed description of Use Case 5.

Use Case: UC-7 Enter Garage

Related Requirements: REQ-5, REQ-6, REQ-7, REQ-17

Initiating Actors: Garage Customer

Actor's Goals: To enter the garage and be directed to a parking spot

Participating Actors: Scanner, Elevator, Database

Preconditions:

- The customer has yet to enter the garage

Postconditions:

- The customer is directed either to the appropriate parking spot, or is asked to back out of the elevator

Flow of events for main success scenario:

- 1.) Customer navigates onto the elevator
- ← 2.) Scanner scans the customer's barcode
- ← 3.) Barcode ID is sent to and compared to existing reservations in database
- ← 4.) Database confirms whether the customer has a reservation or not
 - a) If customer has a reservation continue to step 6
 - b) If customer does not have a reservation continue to step 5
- ← 5.) The system determines there are available walk-in parking spots
- ← 6.) System assigns customer a spot and they drive to it

Flow of events for Extensions (Alternate Scenarios):

- ← 5a. The system determines if there are no more available walk-in parking spots
- 1.) Customer backs out of elevator and exits the garage

Table 9. Fully-Dressed description of Use Case 7.

Use Case: UC-10 Display Statistics*

Related Requirements: REQ-13, REQ-17, REQ-18, REQ-19, REQ-21, REQ-24, REQ-25

Initiating Actors: Garage Manager

Actor's Goals: To pull and display updated parking trend data to parking managers.

Participating Actors: Website, Database

Preconditions:

- Garage Manager has access to an authorized account
- Database has existing records to display

Postconditions:

- The Website will display graphs and tables constructed from customer parking data

Flow of events for main success scenario:

- 1.) Manager selects "Statistics" from website sidebar navigation
- ← 2.) Database processes query and provides garage records
- ← 3.) System compiles and sorts data into organized format
- ← 4.) Website displays stats on-screen

Flow of events for Extensions (Alternate Scenarios):

- ← 2a. The system determines there are no statistics to display
- ← 1.) Website displays on-screen message notifying lack of data

Table 10. Fully-Dressed description of Use Case 10.

Use Case: UC-11 Points Management*

Related Requirements: REQ-11, REQ-12, REQ-15, REQ-16

Initiating Actors: Garage Customer, Garage Manager

Actor's Goals: To view a customer's points (app) or edit points scheme (website).

Participating Actors: ZippyParkApp/Website, Database

- The manager uses Website and the customer ZippyParkApp
 - In this use case, the manager has the same capabilities as the customer and more.

Preconditions:

- The customer has an existing profile stored in the database
- The customer/manager has logged in to the ZippyParkApp/Website

Postconditions:

- An edited version of the customer's point value will take the place of the original in the database (if the manager is to edit the customer's points)
- The ZippyParkApp/Website will display the current point value

Flow of events for main success scenario:

- 1.) User chooses option to view points in the ZippyParkApp/Website
 - If user is a manager, the customers page will allow viewing of every customer's points
- ← 2.) ZippyParkApp/Website communicates with Database and displays the points under customer of interest

Flow of events for Extensions (Alternate Scenarios):

- 1a. Manager chooses option to edit the pricing scheme
- 1.) Manager increases/decreases points for a customer action
- ← 2.) ZippyParkApp /Website communicates with system to edit the points on the database table
- ← 3.) ZippyParkApp /Website sends a confirmation of completed editing

Table 11. Fully-Dressed description of Use Case 11.

Use Case: UC-12 Payment*

Related Requirements: REQ-9

Initiating Actors: Garage Customer

Actor's Goals: To charge payment for reservation.

Participating Actors: ZippyParkApp, Scanner Database

Preconditions:

- The customer has an existing profile stored in the database
- The customer has logged in to the ZippyParkApp
- At least one reservation for the customer of interest must exist in the system

Postconditions:

- The ZippyParkApp will display that the payment has been made

Flow of events for main success scenario:

- 1.) Customer enters payment information into ZippyParkApp
- ← 2.) System collects information and inputs into database under corresponding account
- 3.) Customer makes reservation in ZippyParkApp.
- ← 4.) ZippyParkApp displays confirmation of payment on screen

Flow of events for Extensions (Alternate Scenarios):

- 3a. Customer is walk-in and exits garage.
- 1.) Customer scans barcode.
- ← 2.) Scanner displays confirmation of payment on screen.

Table 12. Fully-Dressed description of Use Case 12.

4d) System Sequence Diagrams

The system sequence diagrams for the fully-dressed descriptions are given below.

UC-4: Create Reservation

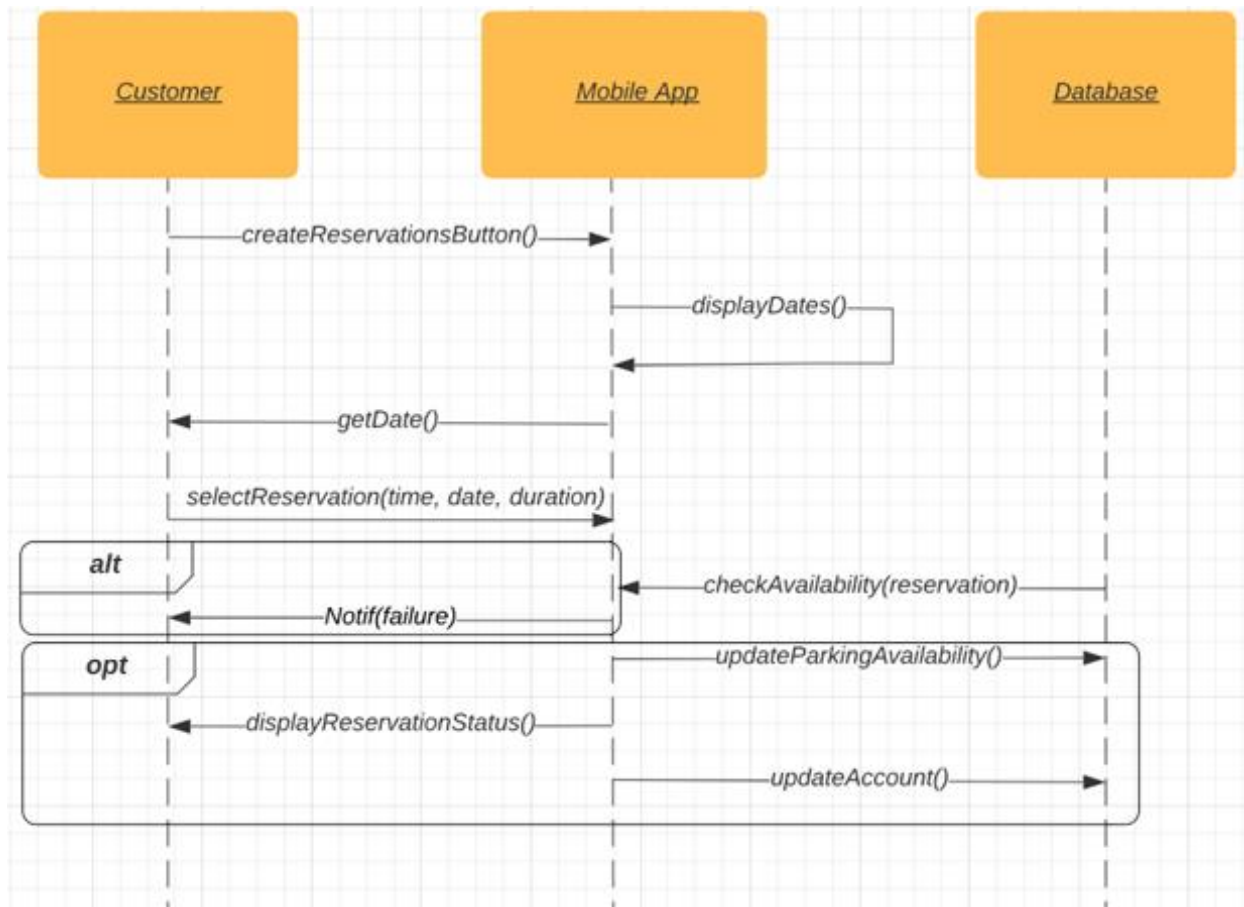


Figure 2. UC-4 Create Reservation.

UC-5: Edit Reservation

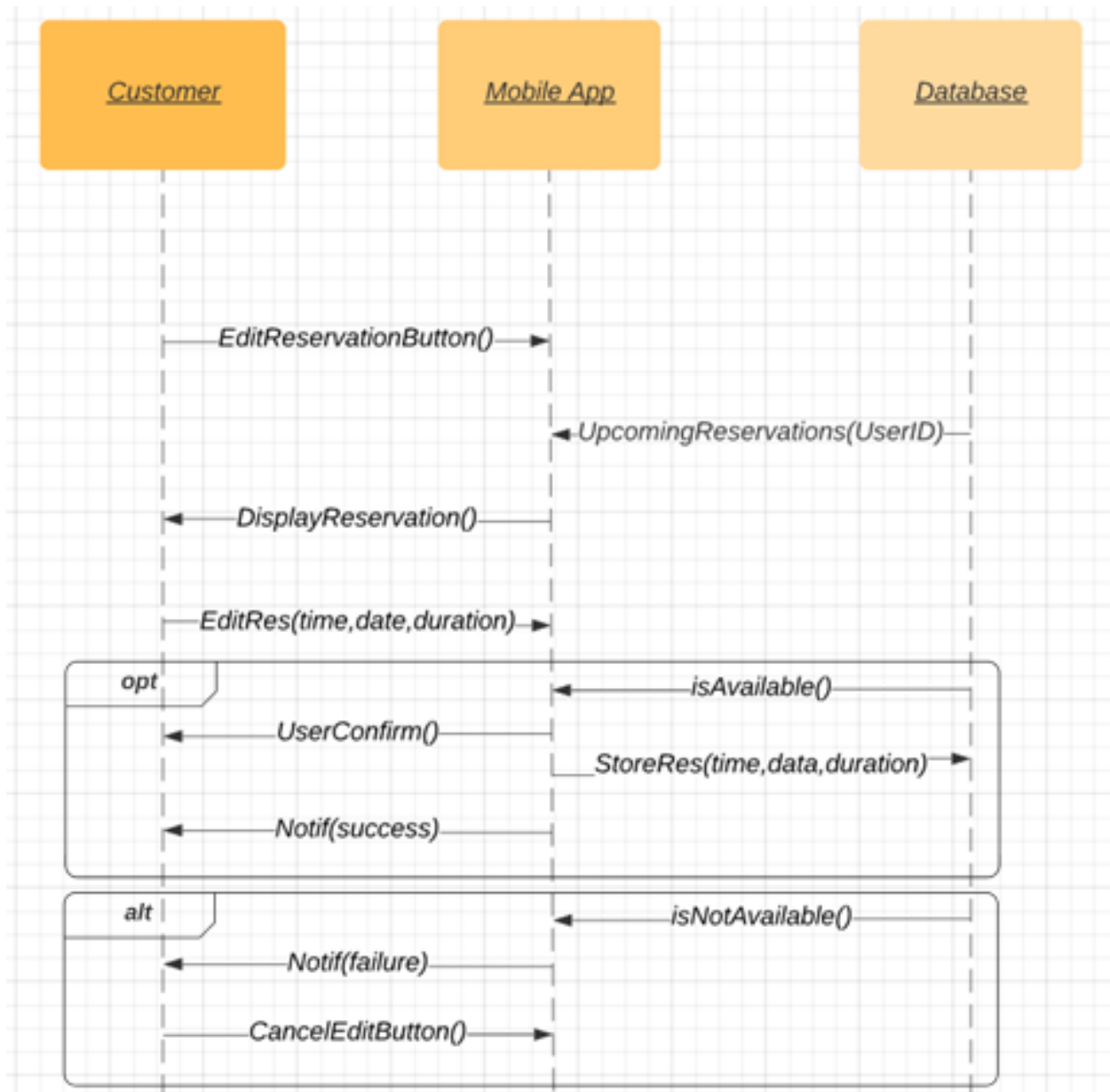


Figure 3. UC-5. Edit Reservation.

UC-7: Enter Garage

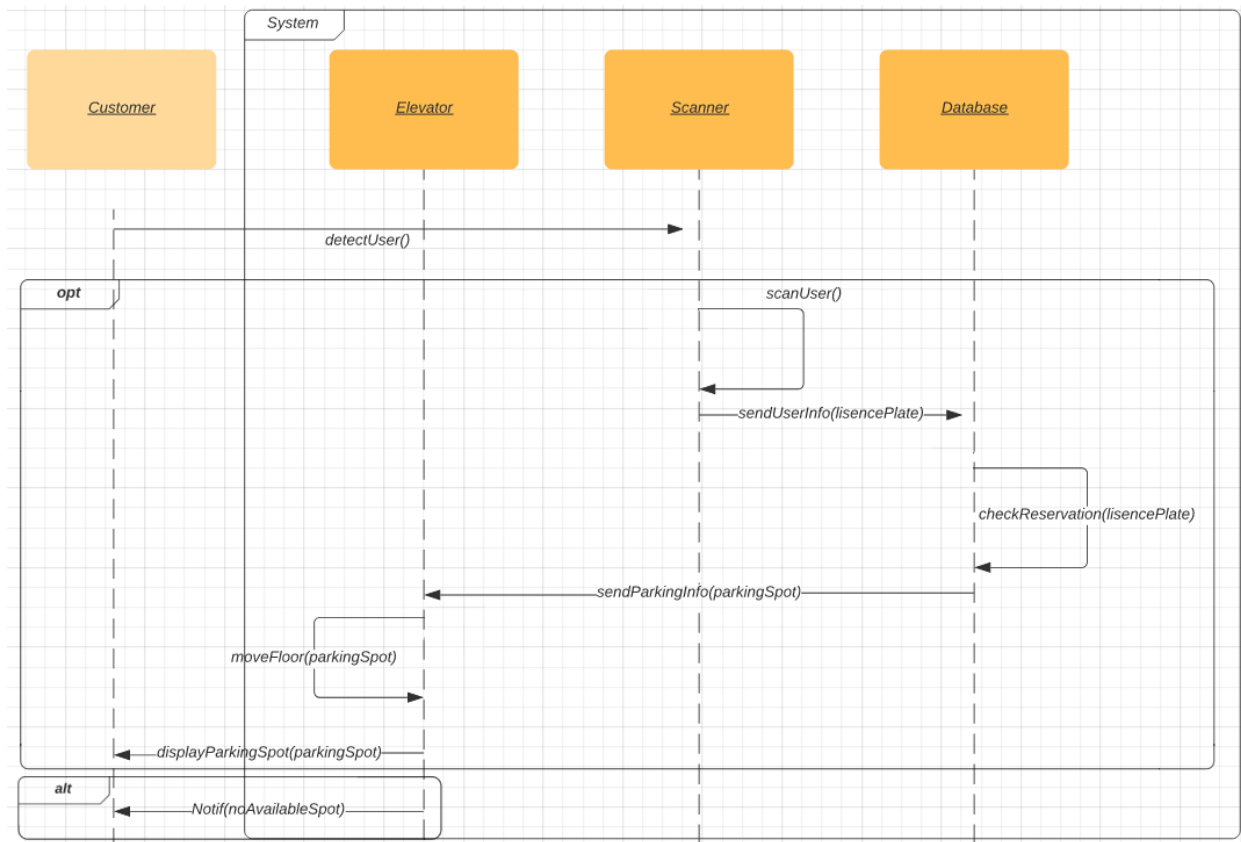


Figure 4. UC-7 Enter Garage.

UC: 10 Display Statistics

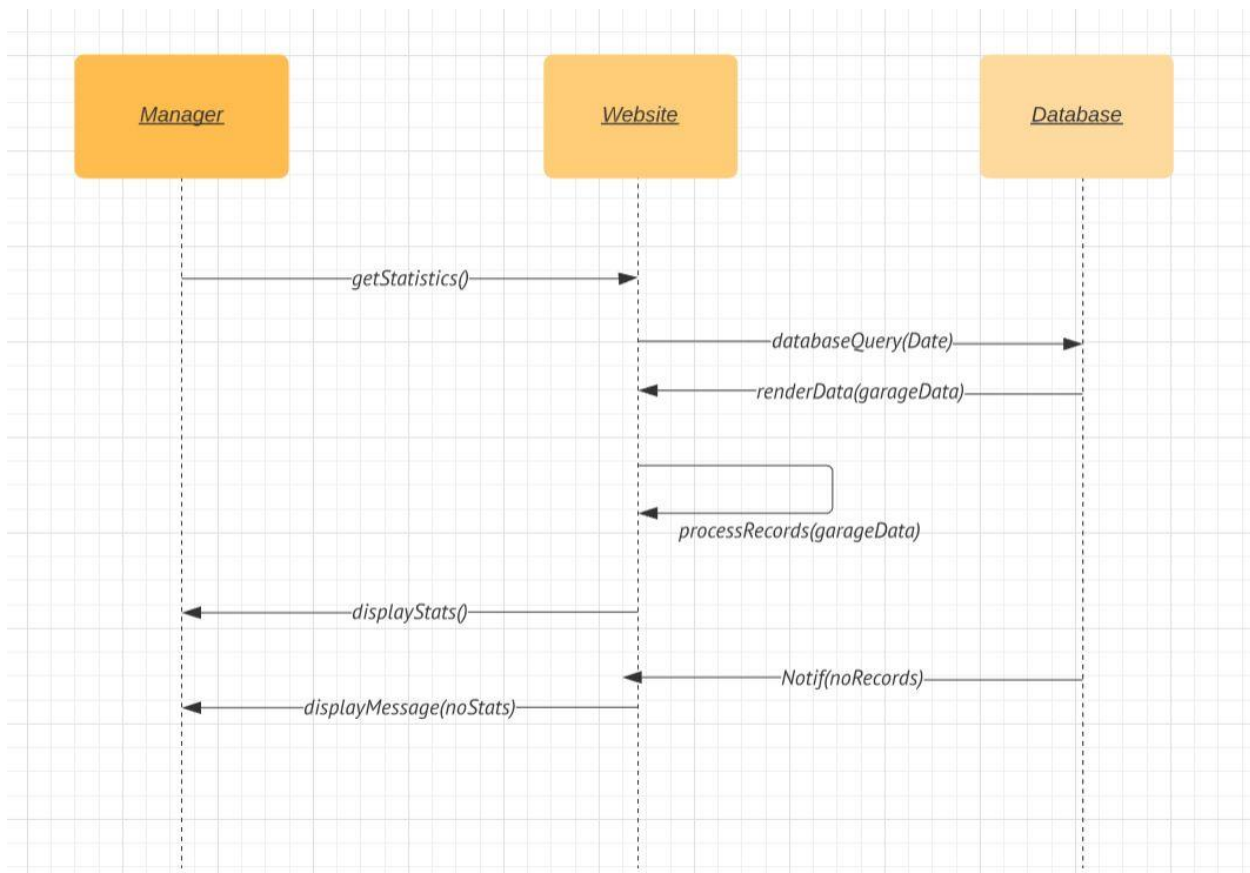


Figure 5. UC-10 Display Statistics. UC-10 is a final demo use case.

UC-11: Points Management

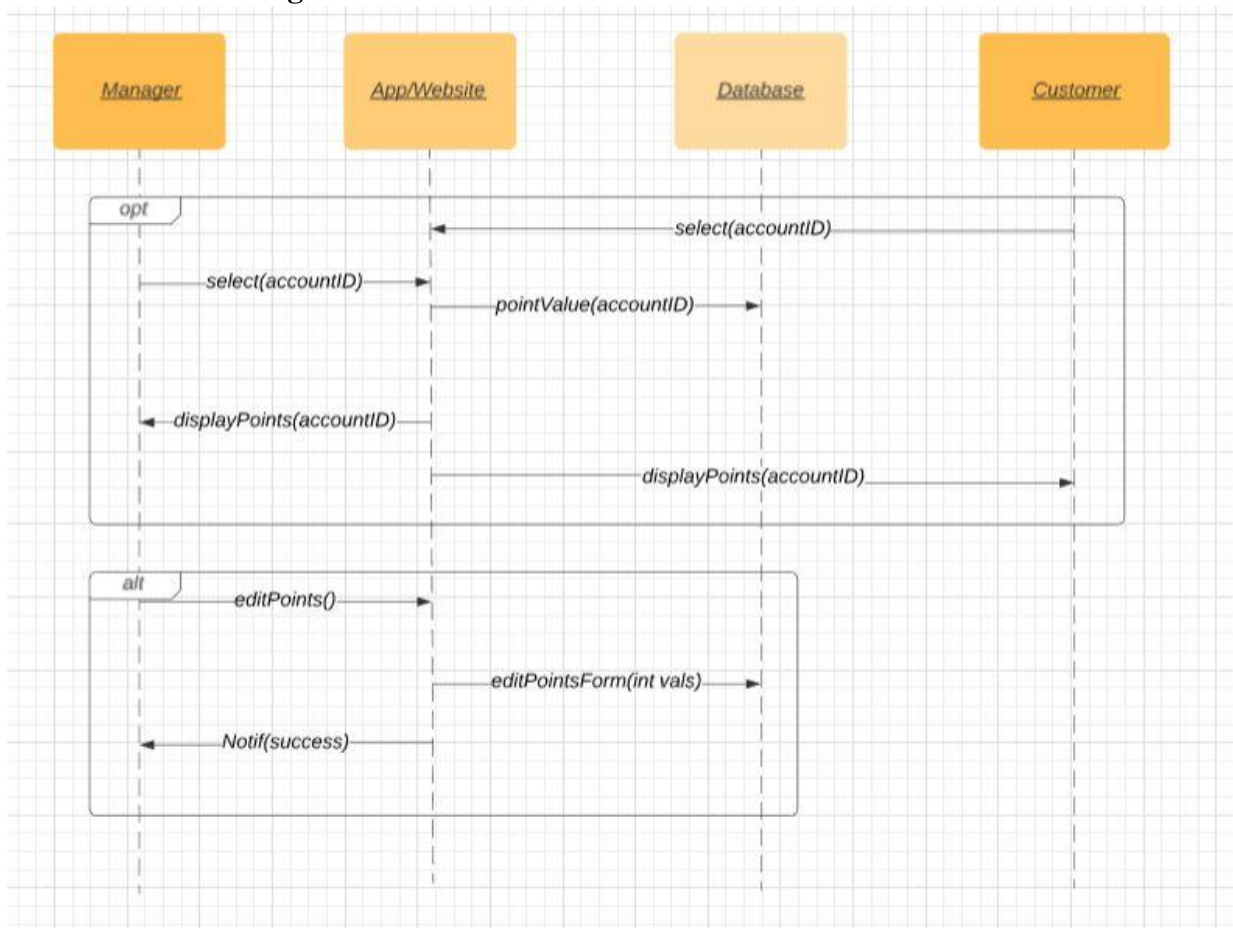


Figure 8. UC-11 Points Management. UC-11 is a final demo use case.

UC-12: Payment Management

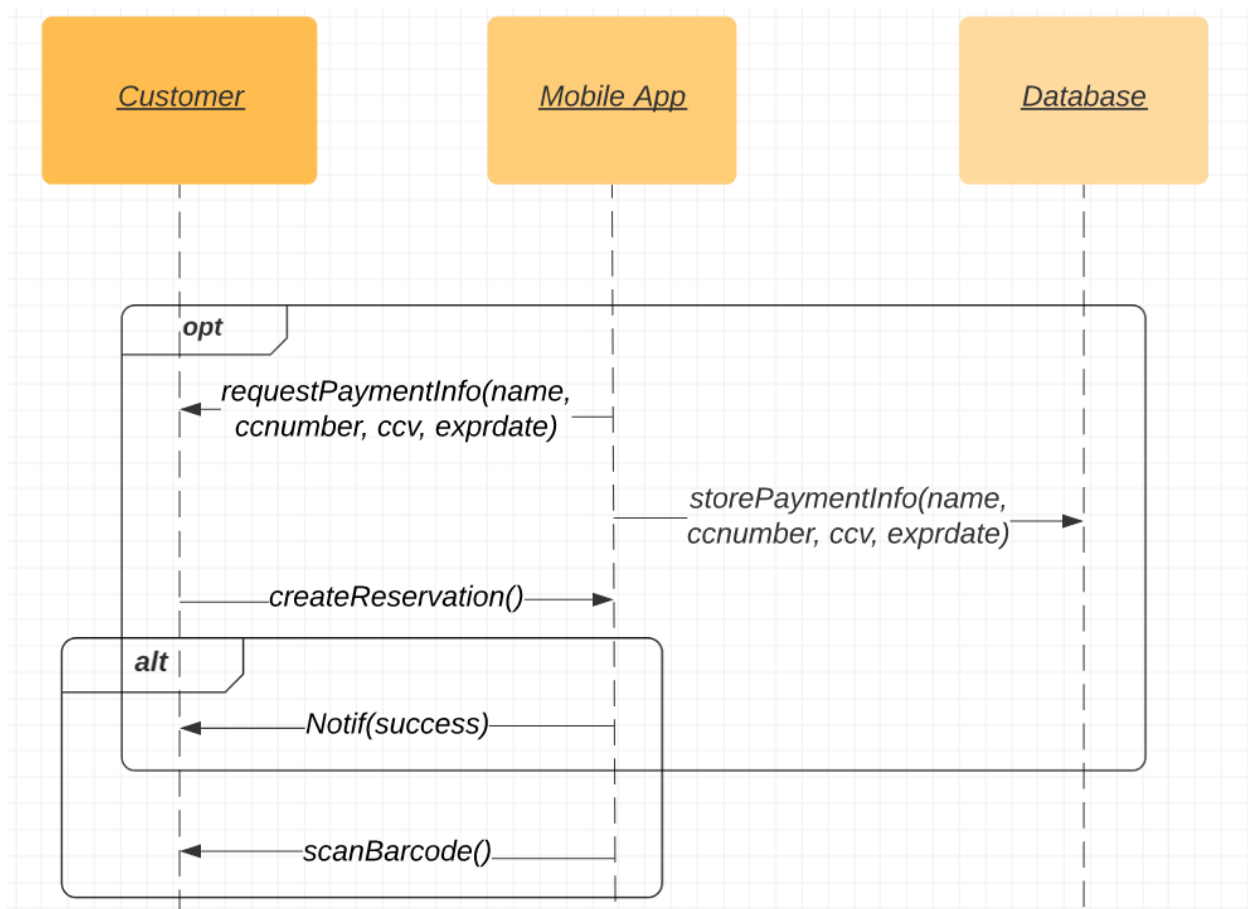


Figure 9. UC-12 Payment Management. UC-12 is a final demo use case.

Section 5: EFFORT ESTIMATION USING USE CASE POINTS

5a) Unadjusted Actor Weight (UAW)

The complexity level was assigned based on the intricacy of the associated interface. The weight level was determined by the complexity, with higher values assigned to more complex levels.

Actor Name	Description of Relevant Characteristics	Complexity	Weight
Customer	The customer can interact with the system through a graphical user interface to manage their account and reserve parking spots.	Complex	3
Garage Manager	The garage manager can interact with the system through a graphical user interface to manage the garage as an authorized user.	Complex	3
Mobile App	Customers interact with the mobile app through a graphical user interface on an application downloaded to their mobile devices	Complex	3
Manager Portal	The garage manager can interact with the manager portal to edit customer accounts and adjust garage settings through text-based inputs and automatic tools.	Complex	3
Database	Contains and updates customer information. Interacts with mobile app, manager portal, and scanner through text inputs.	Average	2
Scanner	Customers can interact with the scanner by having their barcode scanned upon entry. The scanner also records data in the information database.	Average	2
Elevator	The elevator interacts with the information database to lift customers to their designated parking level.	Simple	1

Table 13. Actors with associated weights used in calculations.

$$\text{UAW} = (1 * \text{SIMPLE}) + (2 * \text{AVERAGE}) + (3 * \text{COMPLEX})$$

$$= (1 * 1) + (2 * 2) + (3 * 4)$$

$$\text{UAW} = 17$$

5b) Unadjusted Use Case Weight (UUCW)

The UUCW was found for the highest-priority cases. The category level is based on the number of steps in the main success scenario. A higher weight was assigned to a higher category level.

Use Case	Description of Relevant Characteristics	Category	Weight
Create Reservation	Average user interface (graphical). Three participating actors (Customer, Mobile App, Database). 5 steps for main success scenario.	Average	10
Edit Reservation	Complex user interface (graphical). Three participating actors (Customer, Mobile App, Database). 7 steps for main success scenario.	Complex	15
Enter Garage	Average user interface (graphical). Four participating actors (Customer, Scanner, Elevator, Database). 6 steps for main success scenario.	Average	10
Display Stats	Simple user interface (graphical). Three participating actors (Customer, Manager Portal, Database). 4 steps for main success scenario.	Simple	5

Table 14. Use cases with associated weights used in calculations.

$$\begin{aligned}\text{UUCW} &= (5 * \text{SIMPLE}) + (10 * \text{AVERAGE}) + (15 * \text{COMPLEX}) \\ &= (5 * 1) + (10 * 2) + (15 * 1)\end{aligned}$$

$$\text{UUCW} = 40$$

5c) Technical Complexity Factor (TCF)

Technical factors were identified to estimate the impact on productivity by each requirement of the project. Each factor is assigned a complexity rating based on the perception of effort required to satisfy the requirement. Greater values correlate to higher perceived complexity levels. The weight is determined by the relative impact, with higher values assigned to more impactful items. The calculated factor is produced by multiplying the perceived complexity and weight.

Technical Factor	Description	Perceived Complexity	Weight	Calculated Factor
T1	Distributed system. Website and mobile app need to communicate with the database	3	2	6
T2	Customers expect a fast-performing parking system with minimal interaction.	4	1	4
T3	Customers and managers expect low response times and system stability.	4	1	4
T4	Dynamic pricing model will require moderate calculation. The system will use a simple formula.	2	1	2
T5	Design is reusable and can be implemented in other garages with similar structure.	3	1	3
T6	Installation is complex and would require installation of various software and hardware modules such as cameras and scanners at parking areas.	4	0.5	2
T7	Ease of use is extremely important. The reservation process should be simple and can be completed with minimal interaction.	5	0.5	2.5
T8	Not portable. Hardware systems and databases would be difficult to migrate.	0	2	0
T9	Dynamic pricing model will be easy to modify by the garage managers.	2	1	2
T10	The system should support many concurrent users both using the app and reserving spots	4	1	4
T11	The system will encrypt personal and transaction information.	2	1	2

T12	Third-party companies will not be permitted to use the system.	0	1	0
T13	No special training needed, system is automated and simple to use	0	1	0

Table 15. Technical factors with associated weights used in calculations.

$$\begin{aligned}\mathbf{TCF} &= 0.6 + 0.01 * \mathbf{TOTAL_CALCULATED_FACTOR} \\ &= 0.6 + 0.01 * 31.5\end{aligned}$$

$$\mathbf{TCF = 0.915}$$

Unadjusted Use Case Points (UUCP)

$$\begin{aligned}\mathbf{UUCP} &= \mathbf{UAW} + \mathbf{UUCW} \\ &= 17 + 40\end{aligned}$$

$$\mathbf{UUCP = 57}$$

Environmental Complexity Factors (ECF)

We have been instructed to use **ECF = 1**.

Use Case Points (UCP)

$$\begin{aligned}\mathbf{UCP} &= \mathbf{UUCP} * \mathbf{TCF} * \mathbf{ECF} \\ &= 57 * 0.915 * 1\end{aligned}$$

$$\mathbf{UCP = 52.155}$$

Duration

We have been instructed to assume the productivity factor PF = 28.

$$\begin{aligned}\mathbf{Duration} &= \mathbf{UCP} * \mathbf{PF} \\ &= 52.155 * 28\end{aligned}$$

$$\mathbf{Duration = 1460.34}$$

Section 6: DOMAIN ANALYSIS

6a) Domain Model

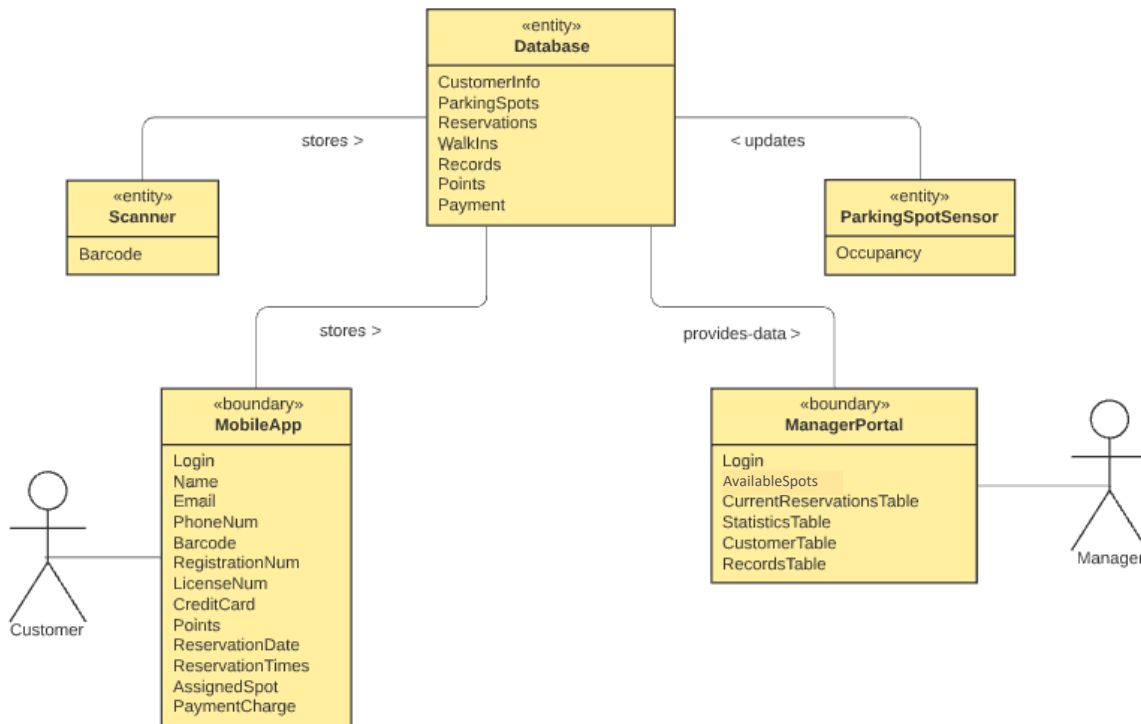


Figure 6. Domain Model.

The domain model shown above was derived directly from the list of use cases and requirements for this project. This model displays how this complex project has been divided into a few concepts, based on the number of actors involved, to make it much simpler to understand. Information regarding the connections between each concept and certain use cases, in addition to their priority weights, can be found in the traceability matrix below.

We chose to split up the inner workings of this project into five main concepts, which will all work together to simulate the procedure of our parking garage:

1. Mobile App: the first interaction our customers will have with our garage is creating a new account via the mobile app. The app will serve as their main interface for managing their reservations and viewing their profile.

2. Scanner: a system that will record which customer has entered and exited the garage, identifying the user by their barcode. Upon entry, it will assign the customer a parking spot, and all information received by the scanner is sent to the database.
3. Parking Spot Sensor: one sensor mounted at each parking spot will determine whether that spot is currently occupied or not, and communicate this information with the database. This helps to ensure that the customers are parked in the correct spot, and whether they have arrived and departed from their spot on time.
4. Manager Portal: displays important statistics to managers, so they can view analytics on the performance of the garage. The portal serves as the main interface for managers to interact with the garage.
5. Database: stores all information regarding reservations, available spots, and customer data, and provides centralized access to all entities.

The Mobile App and Manager Portal are labeled as boundary concepts, because they will serve as the main interface for user interaction. All other concepts are labeled as entities, since information will be communicated between them with no need for user interaction.

i) Concept Definition

Responsibilities are denoted as the following types:

K = knowing

D = doing

N = neither

Responsibility Description	Type	Concept Name
To receive information on a new customer	K	Mobile App
To store information relating to customer and reservations	K	Database
To create/edit/delete reservations	D	Mobile App
To determine vacant spots	D	Parking Spot Sensor
To view all information regarding a customer	K	Manager Portal
To view statistics regarding garage	K	Manager Portal
To compute data involving the garage/customers	D	Database

To determine how long walk-in customers have been in a spot	D	Parking Spot Sensor
To scan in customers by barcode	D	Scanner
To determine if the customer has a valid reservation	D	Scanner
To edit reservation information	D	Mobile App
To edit pricing and reservations	D	Manager Portal
To send customer notifications	D	Mobile App
To ensure customers are in the right spots	D	Parking Spot Sensors

Table 16. Definitions of domain concepts.

ii) Association Definitions

Concept Pair	Association Description	Association Name
Mobile App ↔ database	App receives information from the customer regarding personal details, billing information, or reservations and sends this to the database to keep in records.	Store customer information
Manager portal ↔ database	Manager portal allows for the manager to view and edit information in the database regarding pricing, reservations, user statistics, etc.	Exchange or change information
Scanner ↔ database	Scans in user and sends information regarding entry over to the database.	Scanning in
Parking spot sensors ↔ database	Updates database entries on which spots are available or occupied. Allows for the assigning of spots on user entry.	Assigning spots

Table 17. Definitions of concept associations.

iii) Attribute Definitions

Concept	Attributes	Attribute Description
Database	Customer information table	A data structure that stores the customer's information such as name, license plate number, demographics, etc.
	Parking spots table	A data structure that stores the current information on the parking spots, to determine their occupancy status.
	Reservations table	A data structure that stores the information on upcoming and current reservations, including the customer's license plate number, date and time of reservation.
	Walk-Ins table	A data structure that stores information on walk-in customers, such as the time the customer enters and leaves the garage.
	Records table	A data structure that stores the information on all reservations, existing and past, including whether the reservation was deleted or not.
	Points table	A data structure that stores information on different actions and how many points are gained or lost in association.
	Payment table	A data structure that stores information regarding the dynamic pricing scheme.
Manager Portal	Login	The manager's username and password to verify their identity and access the portal.
	Available spots	The number of available parking spots in the garage, used to determine availability for walk-in customers
	Current reservations table view	A table that displays current reservations. This includes start and end times of each individual reservation and the customer they are under.
	Statistics table view	A table that displays garage statistics, including average reservation times and average reservation lengths.
	Customer table view	A table that displays the number of customers daily as well as individual customer information such as time of arrival, time of departure, and license plate numbers.
	Records table view	A table that displays past records. This includes past reservation dates and times, payment information (credit card information, name).

Mobile App	Login	The customer's username and password to verify their identity and access the profile.
	Name	The customer's first and last name for contact information.
	Email	The customer's email address for contact information.
	Phone number	The customer's phone number for contact information.
	Barcode	The customer's assigned barcode to identify the account.
	Registration number	The customer's assigned registration number to identify the account.
	License number	The customer's registered car's license number.
	Credit card	The customer's credit card information, including name, billing address, and CVV.
	Number of points	The customer's number of points collected.
	Date	The date of reservation.
	Start time, end time	The start and end times of reservation, also used to determine payment charge by dynamic pricing model.
	Assigned spot	The parking spot number assigned to reservation.
	Payment charge	The dollar amount of cost of reservation.
Parking Spot Sensor	Occupancy	This detects if the parking spot is empty or filled (Boolean).
Scanner	Barcode	The barcode that the scanner reads and recognizes.

Table 18. Definitions of concept attributes.

iv) Traceability Matrix

This matrix maps the use cases to the domain concepts. The total priority weight (PW) of each requirement was calculated using the requirements-to-use-case traceability matrix. A higher number indicates a higher priority.

PW	Use Cases	Mobile App	Manager Portal	Scanner	Parking Spot Sensor	Database
12	UC-1	x	x			x
8	UC-2	x				
1	UC-3		x			x
18	UC-4	x				x
16	UC-5	x	x			
15	UC-6	x	x			x
22	UC-7			x	x	x
4	UC-8			x	x	x
11	UC-9				x	x
18	UC-10		x			
4	UC-11		x			x
2	UC-12	x				

Table 19. Traceability matrix mapping use cases to domain concepts.

6b) System Operation Contracts

In this section, use cases marked with an asterisk will be highlighted in the final demo*. All unmarked cases have the highest priority weight. UC-10 is both a high priority case and a final demo case**.

Operation	UC-4 Create Reservation
Preconditions	<ul style="list-style-type: none"> • The customer has an existing profile stored in the database • The customer has logged into the ZippyPark app
Postconditions	<ul style="list-style-type: none"> • The system reduced the number of available spots for the reserved time • The new reservation is tied to the user's account reservations

Table 20. System operation contract for use case 4.

Operation	UC-5 Edit Reservation
Preconditions	<ul style="list-style-type: none"> • At least one reservation for the customer of interest must exist in the system
Postconditions	<ul style="list-style-type: none"> • An edited version of the reservation will take the place of the original in the database • The original reservation is wiped from the database

Table 21. System operation contract for use case 5.

Operation	UC-7 Enter Garage
Preconditions	<ul style="list-style-type: none"> • The customer has yet to enter the garage
Postconditions	<ul style="list-style-type: none"> • The customer is directed either to the appropriate parking spot or is asked to back out of the elevator

Table 22. System operation contract for use case 7.

Operation	UC-10 Display Statistics**
Preconditions	<ul style="list-style-type: none"> • The Garage Manager has access to an authorized account • The Database has existing records to display
Postconditions	<ul style="list-style-type: none"> • The ZippyPark Manager Portal will display graphs and tables constructed from customer parking data

Table 23. System operation contract for use case 10.

Operation	UC-11 Points Management*
Preconditions	<ul style="list-style-type: none"> • The customer has an existing profile stored in the database • The Garage Manager has access to an authorized account • The Garage Manager sets a point rewards scheme
Postconditions	<ul style="list-style-type: none"> • The customer can view their points via their account • The customer is able to redeem their points for VIP parking spots if they have points available

Table 24. System operation contract for use case 11.

Operation	UC-12 Payment*
Preconditions	<ul style="list-style-type: none"> • The customer has an existing reservation • The customer has provided accurate payment information in their account profile • The Garage Manager has access to an authorized account • The Garage Manager sets a point rewards scheme
Postconditions	<ul style="list-style-type: none"> • The pricing scheme has been manually changed by the Garage Manager or automatically altered by the pricing tool. • The amount is charged to the credit card on the customer's profile as soon as the reservation is made

Table 25. System operation contract for use case 12.

Section 7: INTERACTION DIAGRAMS

UC-4: Create Reservation

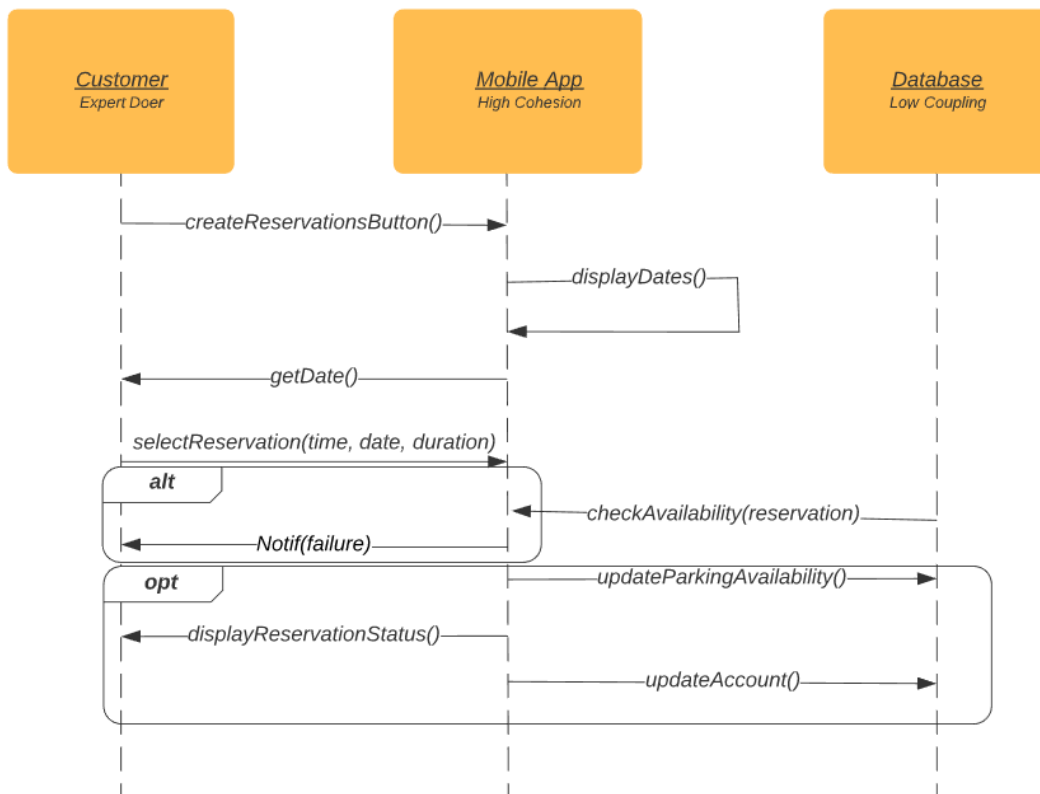


Figure 7. Interaction Diagram of UC-4.

Description of Design Principles -

UC-4 describes the process of creating a reservation. The interaction diagram of UC-4 was derived from the system sequence diagram shown in Figure 2. Concepts from the domain model were used, with no new domain concepts or objects introduced.

The mobile app is responsible for communication between the database and the customer. The app provides a centralized hub to accept input and display output on an organized user interface. Because the app focuses more on delivering information between the user and the database and less on computational processing, it follows the high cohesion principle. The app readily provides information to the customer and pulls/pushes to the database. However, the majority of changes are up to the users, who know how to interact with the app and what they want to accomplish. They are designated as expert-doers within the system. Through the app, the users can add a new reservation to the database. By leaving the user interactions to the app, the

database can focus on updating itself with incoming requests. Minimizing communication to a single incoming and outgoing link follows the low coupling principle. The idea for this design is to streamline communication and limit latency, promoting a smoother customer experience. And by having fewer objects, the system is less likely to fail due to any single component. The final design allows customers to create their new reservations by typing in their desired date, start time, and end time, and then receive confirmation quickly. While that occurs, the mobile app accepts the incoming information, checks availability with the database, and updates the customer's account information.

UC-5: Edit Reservation

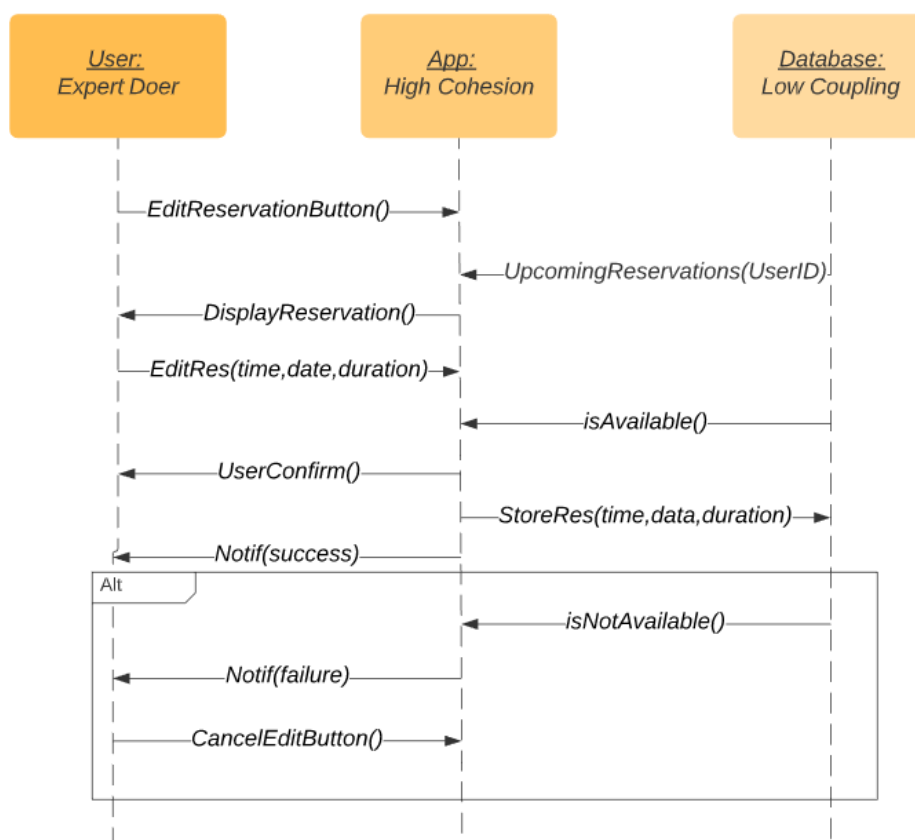


Figure 8. Interaction Diagram of UC-5.

Description of Design Principles -

UC-5 describes the process of editing a reservation. The interaction diagram of UC-5 was derived from the system sequence diagram shown in Figure 3. Concepts from the domain model were used, with no new domain concepts or objects introduced.

For UC-5, customers may make reservations through the app. The app allows whichever user to interact with the database that holds all the information regarding reservations. We denote the user to be the expert-doer as the user is the one with the knowledge of what changes need to be made. The app is designed with the high cohesion principle in mind-that is, the app should do very little computational work. Instead, the app is used more as a medium of communication. Making the app to be the mode of communication allows for the database to focus on doing the work to modify the reservations so that it follows the low coupling design principle. The database will therefore have to do less communication on its part. Allowing for different parts of the system to take on different roles ensures that no one actor takes on too much work.

A Publisher-Subscriber pattern is used here with the app being the publisher and the customer being the subscriber. The subscriber inputs information, such as their user identification and the edits to a reservation. The publisher makes changes given the information from the subscriber; it communicates with the database to change the reservation as desired.

UC-7: Enter Garage

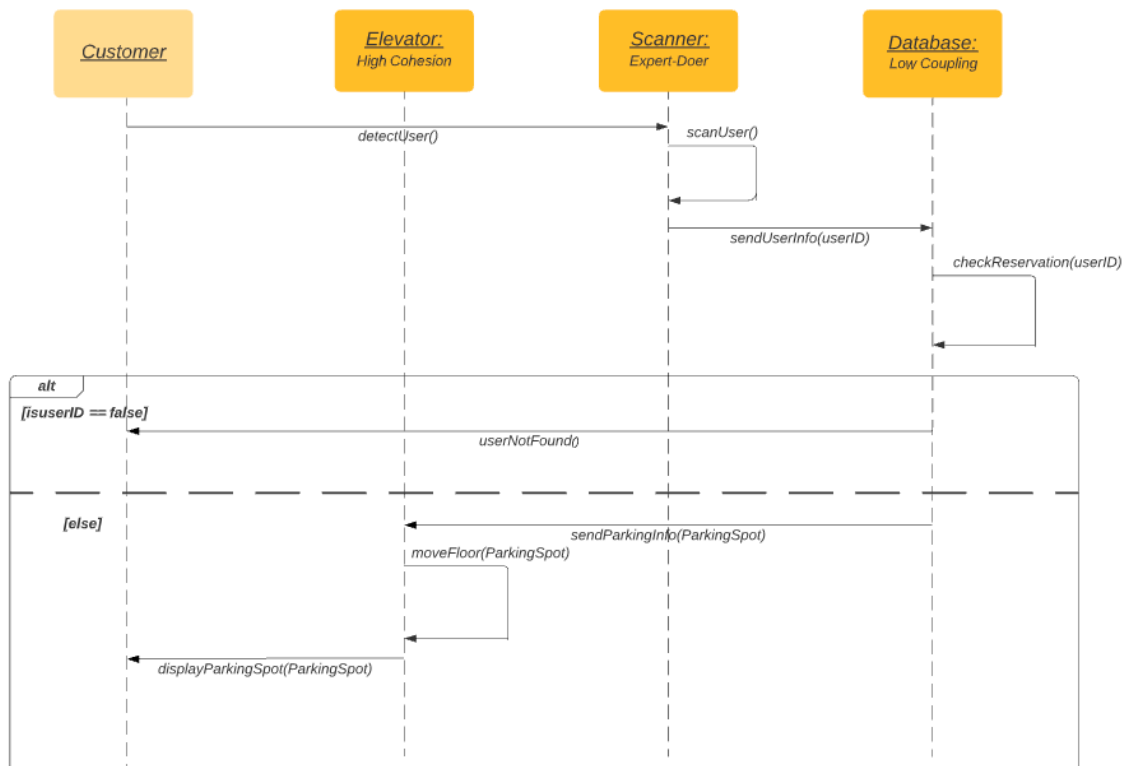


Figure 9. Interaction Diagram of UC-7.

Description of Design Principles -

UC-7 details the process of a customer entering the garage. The interaction diagram of the fully dressed UC-7 was derived from the system sequence diagram shown in Figure 4.

Concepts from the domain model were used, with no new domain concepts or objects introduced.

This process starts with the customer driving onto the elevator, which is detected by the scanner. The scanner is then responsible for obtaining and sending the customer's data (barcode ID) to the database. The scanner would be classified as the Expert-Doer since it knows to scan the customer's ID, which is the most important data that is to be passed throughout the process. We decided on this structure as this method requires minimal interaction from the customer's part. Since the only job of the customer is to drive onto the garage, they do not have to enter in any information as the scanner will automatically do it for them. The database is then responsible for finding and confirming the customer's existing reservation based on the data sent from the scanner. The database follows a low coupling principle because it helps reduce unnecessary associations between objects by providing messages to the scanner if the customer account exists and acts as a medium of communication for the actions of other objects. If the ID isn't matched to one existing in the database, the customer doesn't have an account so they are not allowed to enter. If the ID is found, the database reports to the elevator an available parking spot. The elevator moves to the appropriate floor and displays what parking spot the customer should navigate to. The customer is then informed exactly where they should park and how to arrive at that location, making the process easy and efficient for the customer and the system. The elevator follows a high cohesion principle since it takes care of the workload once the user enters the garage by assigning, displaying and guiding the user to their designated parking spot.

UC-10: Display Statistics

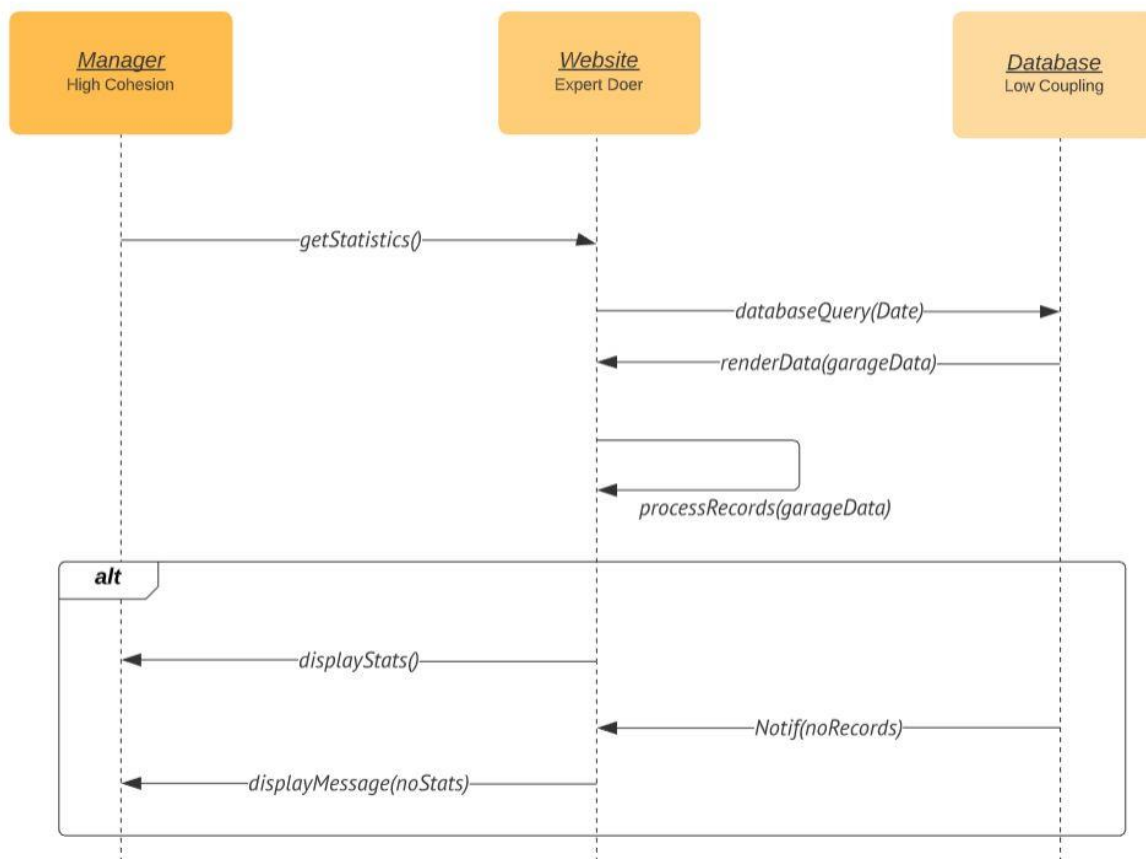


Figure 10. Interaction Diagram of UC-10.

Description of Design Principles -

UC-10 enables managers to view a variety of parking statistics for their parking garage. The interaction diagram of UC-10 was derived from the system sequence diagram shown in Figure 5. Concepts from the domain model were used, with no new domain concepts or objects introduced.

The manager initiates the action by visiting the statistics page of the website. The website retrieves relevant data from the database, then displays graphs and tables for the manager to view. In an alternate scenario, the website does not find any data and reports to the manager that there are no statistics available.

The website is responsible for pulling all parking data from the database and processing it to create a clean and easy way for the manager to view parking statistics and metrics in a

dedicated “Statistics” page of the website. We have decided on this design so that the parking manager can quickly view relevant information about their garage on one page, then navigate to other pages (i.e. the dynamic pricing model editor) and make changes with relative ease. The manager’s only responsibility is to request to view statistics; this decision is based on the high cohesion principle whereby the manager does not take on computation responsibilities. This way, the end-user can manage and diagnose their garage with minimal effort. The website behaves according to the expert-doer principle because it is the information expert, and so has the most responsibilities in this use case. The website makes requests to the database, then handles all data processing so it is readily available for display to the manager. This minimizes latency between the initiating command and final display. The database’s responsibilities were decided based on the low coupling principle, so that it does not need to do much communication; its only job is to return records when requested by the website. Once the website compiles and performs computations on the data, the garage manager can view graphs such as the number of reservations per day, number of reservations per hour, revenue over time, and other relevant metrics. The accompanying tables for these graphs are displayed below them. The parking manager will be able to view all of this information on a single site which will help them make proper business decisions for their garage.

Section 8: CLASS DIAGRAM AND INTERFACE SPECIFICATION

8a) Class Diagram

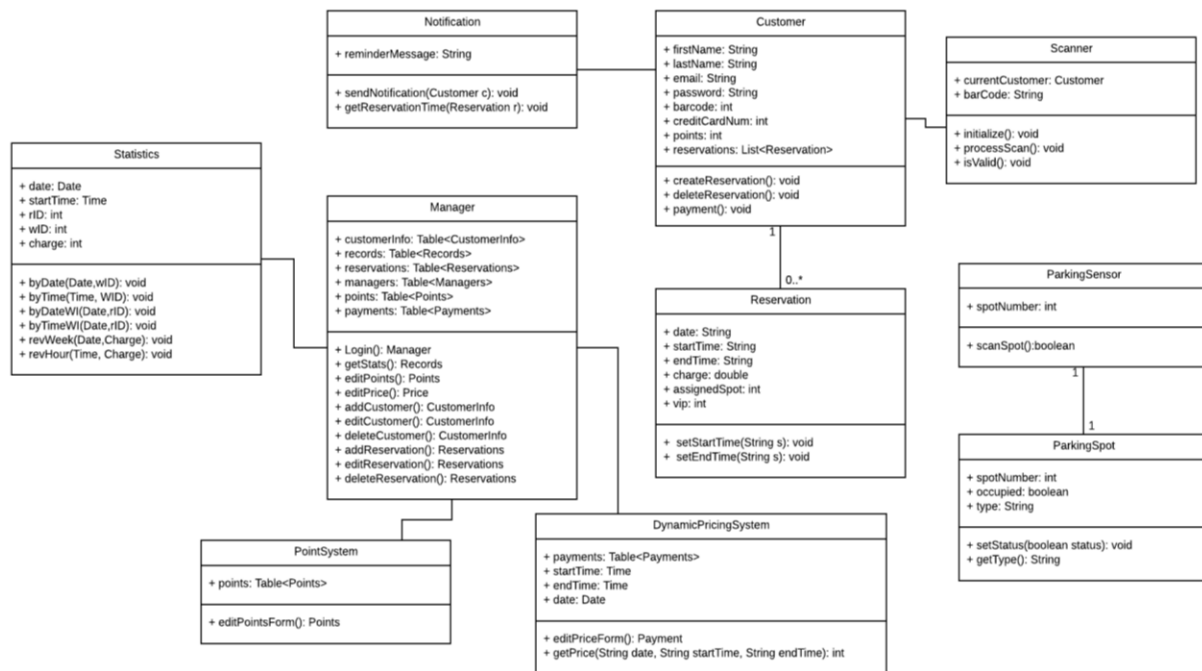


Figure 11. Full class diagram.

Each class in the Class Diagram originated from an abstract concept of the Report #1 domain model. An explicit outline of which classes/objects came from which domain model concept is further detailed beneath the table in “Section 8c: Traceability Matrix.”

8b) Data Types and Operation Signatures

The classes shown in the class diagram are enumerated below with descriptions of the class itself, its attributes, and its operations.

1. Customer

Controls customer account information

a. Attributes

- i. String firstName: customer’s first name
- ii. String lastName: customer’s last name
- iii. String email: customer’s email address

- iv. String password: customer's account password
- v. String phoneNum: customer's phone number
- vi. int barcode: customer's unique barcode
- vii. String licenseNum: customer's registered license plate number
- viii. String creditCardType: customer's credit card type
- ix. int creditCardNum: customer's credit card number for payment
- x. String expDate: customer's credit card expiration date
- xi. String CVV: customer's credit card CVV
- xii. int numPoints: customer's number of reward points
- xiii. List<Reservation> reservations: list of customer's reservations

b. Operations

- i. createReservation(): creates a new reservation for the customer
- ii. deleteReservation(): deletes an existing reservation for the customer
- iii. payment(): uses credit card information to pay for reservation

2. Parking Sensor

Senses parking spot occupancy

a. Attributes

- i. int spotNumber: number associated with a parking spot
- ii. String type: type of parking spot

b. Operations

- i. scanSpot(): scans parking spot to determine occupancy

3. Parking Spot

Tracks parking spot occupancy

a. Attributes

- i. int spotNumber: number associated with a parking spot
- ii. String type: type of parking spot
- iii. boolean occupied: tracks whether spot is occupied or not

b. Operations

- i. setStatus(): sets parking spot status to occupied or not occupied

4. Scanner

Tracks customer entrances and exits

a. Attributes

- i. Customer currentCustomer: customer that the scanner is tracking
- ii. String barCode: barcode that the scanner scans

b. Operations

- i. initialize(): sets up the interface for the app
- ii. processScan(): processes customer entry/exit for reservations/walk-ins
- iii. isValid(): verifies the managers credentials when starting the app.

5. Manager

Creates, edits, deletes reservations and customers. Edits payment and point schemes.

a. Attributes

- i. Table<CustomerInfo> customerInfo: gets a table of all customer info
- ii. Table<Records> records: gets a table of all records
- iii. Table<Reservations> reservations: gets a table of all reservations
- iv. Table<Managers> managers: gets a table of all managers
- v. Table<Points> points: gets a table of all the point values/activities
- vi. Table<Payments> payments: gets a table of all payment values/activities

b. Operations

- i. Login(): allows manager to access portal
- ii. getStats(): retrieves statistics based on records
- iii. editPoints(): edits point values
- iv. editPrice(): edits price values
- v. addCustomer(): manually adds a customer
- vi. editCustomer(): manually edits customer info
- vii. deleteCustomer(): manually deletes a customer
- viii. addReservation(): manually creates a reservation
- ix. editReservation(): manually edits an existing reservation
- x. deleteReservation(): manually deletes an existing reservation

6. Reservation

Creates a reservation

a. Attributes

- i. String date: date of reservation

- ii. String startTime: start time of reservation
- iii. String endTime: end time of reservation
- iv. int barcode: scanned barcode associated with reservation
- v. double charge: price of reservation
- vi. int vip: vip status flag
- vii. boolean deleted: tracks whether reservation has been deleted or not
- viii. int assignedSpot: customer's assigned spot upon entering parking garage

b. Operations

- i. setStartTime(): sets the start time of a reservation
- ii. setEndTime(): sets the end time of a reservation

7. Point System

Tracks and changes customer points

a. Attributes

- i. Table<Points> points: gets a table of all point values/activities

b. Operations

- i. editPointForm(): edits the point form and allows the manager to change the point value for different actions, such as create reservation

8. Dynamic Price System

Changes the parking price rate

a. Attributes

- i. Table<Payments> payments: gets a table of all payment values/activities
- ii. Time startTime: value of the start time of interest
- iii. Time endTime: value of the end time of interest
- iv. Date date: value of the date of the interest

b. Operations

- i. editPriceForm(): allows the manager to edit the price data for different reservation times and dates
- ii. getPrice(): gets price of parking reservation given the date and start and end times

9. Notifications

Sends notifications to the customer

a. Attributes

- i. String reminderMessage: reminder message for customer to notify about the upcoming reservation

b. Operations

- i. sendNotification(): sends notification to customer with reminder message
- ii. getReservationTime(): gets reservation date and time to notify customer with specific time

10. Statistics

Tracks different statistics based on date and time

a. Attributes

- i. Date date: value of the date of interest
- ii. Time startTime: value of the start time of interest
- iii. int rID: reservation ID of a specific reservation
- iv. int wID: walkin ID of a specific walkin
- v. int charge: amount charged for a specific parking

b. Operations

- i. byDate(): gets statistics for reservations based on a certain date
- ii. byTime(): gets statistics for reservations based on a certain time
- iii. byDateWI(): gets statistics for walkins based on a certain date
- iv. byTimeWI(): gets statistics for walking based on a certain time
- v. revWeek(): get statistics on the revenue for a specific week
- vi. revHour(): get statistics on the revenue for a specific hour

8c) Traceability Matrix

This matrix maps the software classes to the domain concepts. Further explanation of the mappings is given below for each enumerated domain concept.

		Domain Concepts				
		Database	Mobile App	Manager Portal	Scanner	Parking Spot Sensor
Software Classes	Customer	✓	✓	✓		
	Parking Spot	✓			✓	✓
	Manager	✓		✓		
	Reservation	✓	✓	✓	✓	
	Point System	✓	✓	✓		
	Dynamic Pricing System	✓	✓	✓		
	Notifications	✓	✓			

Table 26. Traceability matrix of software classes to domain concepts.

All of the classes originated from the Report #2 domain concepts. See below for further explanation. Each domain concept is enumerated and its relation to a certain class is listed beneath it (a, b, c...).

1. Database

- a. Customer – the database will store customer information in a table in the database.
- b. Parking Spot – the database will modify parking spot vacancy information as customers enter/exit the garage.
- c. Manager – the database will save all parking data for the website to display and calculate statistics for.
- d. Reservation – the database will keep a record of all current and past reservations
- e. Point System – the database will keep an up-to-date point allocation model based on actions committed by customers.
- f. Dynamic Pricing System – the database will store the current pricing model for each hour that the garage is open.

- g. Notifications – the database will provide updated reservation and customer data to display pertinent notifications to the customer.

2. Mobile App

- a. Customer – the mobile app will allow the customer to view and modify their reservations and account information.
- b. Reservation – the mobile app will be able to retrieve/create reservations for each customer.
- c. Point System – the mobile app allows customers to earn and trade points in for a VIP parking spot reward under the point system.
- d. Dynamic Pricing System – the mobile app utilizes the dynamic pricing system to determine the most up to date cost for each parking reservation.
- e. Notifications – the mobile app has the ability to display pertinent notifications.

3. Manager Portal

- a. Customer – the manager can view and interact with individual customers as well as their data.
- b. Manager –the manager can view statistics, set prices, and edit customer information as well as reservations through the manager portal.
- c. Reservation – the manager portal has the capability to create, edit, and delete reservations.
- d. Point System – the manager portal can modify the point allocation model for all customers.
- e. Dynamic Pricing System – the manager portal sets the current pricing model for the parking garage.

4. Scanner

- a. Parking Spot – the scanner assigns a parking spot to the customer based on their reservation/walk-in information
- b. Reservation – the scanner reads the list of reservations to determine when a customer may enter/exit a garage

5. Parking Spot Sensor

- a. Parking Spot – the parking spot sensor indicates whether a parking spot is occupied or vacant.

8d) Design Patterns

Multiple design patterns were incorporated into the coding of the project, with the purpose of making the design easy to organize and use for various needs. Since the three major aspects of the project were the app, the website, and the scanner there were three distinct design patterns used.

The major design pattern for the app was the command pattern which is a behavioral design pattern. Behavioral design patterns deal with the class's object communication and communication between objects. The command pattern is a design where command requests between objects happen as objects [7]. This is due to the fact that the app passes information through different screens. The requests coming from one object on the app exist as objects that can be interacted with by the user on the app.

The website incorporates the publish-subscribe pattern (pub-sub pattern). This pattern is similar to the visitor behavioral design pattern as it allows one object to be accessed and interacted with by many other unknown objects. This follows the pattern of how a publisher is interacted with by many unknown subscribers [7]. This pattern is extremely useful in the website portion of the project as a lot of the HTML coding used between different pages is the same and can be reused for other pages/objects. By following the pub-sub pattern, the complexity of the overall program decreases and the same code was not written over and over again.

Finally, the scanner utilizes the iterator pattern. This is a behavioral design pattern which organizes the way objects interact with other objects. The iterator design pattern allows objects to sequentially access the elements of a collection [8]. This provides a way for the scanner objects to access the elements of an aggregate object sequentially without exposing its underlying representation. This is used when retrieving the list of reservations associated with a specific barcode and it allows the scanner to store and sort the list of reservations by star time to determine the most relevant listing. Finally, this design pattern also allows the scanner to pull the hourly pricing information from the database into an array. The array is then utilized to calculate the total cost of walk-ins and reservations.

8e) Object Constraint Language Contracts

Class	Invariant	Precondition	Postcondition
Notification	context notifications inv: self.hasReservations > 0	context notifications::notify(currentTime: Integer, reservation: Integer):String pre: reservation.isValid = 1	post: result = notificationOfReservation
Scanner	context scanner inv: self.barcodeIsValid	pre:self.barcodeIsValid = 0	post:self.barcodeIsValid = 1
Manager	context manager inv: self.managerAccess = 1	pre:self.managerOptions = 0	post:self.managerOptions > 0
Reservation	context reservation inv: self.reservationList	context reservation::newRes(startTime: Integer, endTime: Integer): Integer pre: startTime < endTime	post:self.reservationList.add(result)
Sensor	context sensor inv: self.isActive = 1	pre:self.detectsCar = 0	post:self.detectsCar = 1
Point System	context pointSys inv: self.hasPoints = 1	context pointSys::calcPoints(points: Integer): Integer pre: points > 0	post: result = points + pointReward
Pricing System	context pricingSys inv: self.hourlyRate.isEmpty = 0	context pricingSys::getCost(startTime: Integer, endTime: Integer): double pre: startTime < endTime	post: result = cost
Parking Spot	context ParkingSpots inv: self.numSpots > 0	context ParkingSpots::getSpot(spots: Integer[]):Integer pre: spots.isEmpty = 0	post: result = spotNum, numSpots = numSpots@pre-1

Section 9: SYSTEM ARCHITECTURE AND SYSTEM DESIGN

9a) Architectural Styles

Our project will follow an “event-driven” architectural style. This is a style that focuses on how the system works as it responds to the events that occur. In our case, events are described as any significant change that is made in relation to the parking garage. For example, a change may be editing a reservation time or changing the occupancy status of a parking spot. These events result in the changing of the data held in the database. In the instance of editing a reservation, we would see the records change to hold the values of the new reservation as a result of the communication between the website/app and the database. With the parking spot more communication is required as the weight sensor must determine the spot to be “occupied” and not “vacant” and then send this information to the database.

The event-driven style is based on event emitters and event consumers. Additionally, event channels may be involved to connect the two. Emitters may also be referred to as agents and the consumers are also known as sinks. An agent is essentially the component that detects the change or the fact that an event occurred. In our example of parking the car, the emitter would be the weight sensor that determines that a car has taken up a certain spot. The sink is what reacts to the event that occurred. Following our example, the database is the collector that receives a notification of this change and reacts by storing “occupied” in a spot that was formerly “vacant”. In general, our project’s specific architectural style will always refer to the database as the sink since it is the actor that always ends up taking note of the changes. In most cases, we would see the app/website as the event channel that works to communicate between the event emitter and the event consumer.

9b) Identifying Subsystems

Our website runs on the typical client server technology where the server handles requests from the client through an Android app. The server or database is able to collect data through either one or multiple devices, depending on the usage. The server interacts with the user interface which is the app, the manager portal, and the hardware devices such as our scanners and parking sensors. The server receives information from the app which is then updated and

verified using the hardware. The history of that information is updated to the manager portal. Using the app, manager portal, and the hardware devices the database is able to store a full history of every customer's profile information and reservation. An outline of our subsystems is shown in Figure 6.

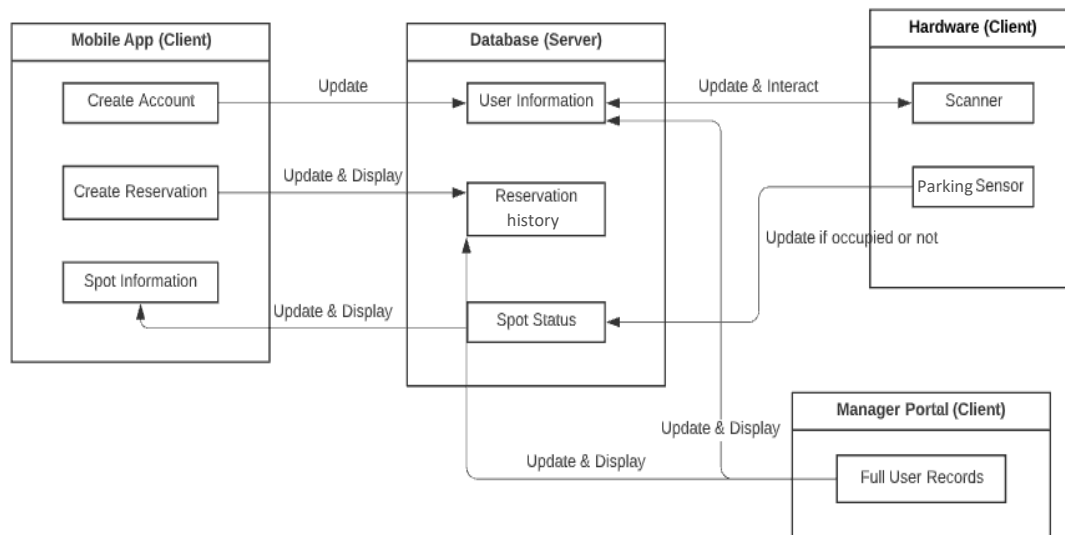


Figure 12. UML Package Diagram of Subsystems.

9c) Mapping Subsystems to Hardware

There are multiple hardware components required for the system to function smoothly. The main computer will host the server/database and can receive requests and interact with clients. Such requests can come from customers who are using either the app or the web browser from a personal device. Any customer can request an action at any time, meaning many hardware devices are being used by the system at once, or very few. Of course, the computer hosting the server/database must always be functioning for the system to work.

Another notable client would be a manager, who can also interact with the server/database from a personal device. Just like the customer, the manager can link his hardware device to the system, request certain actions, and then unlink his device from the system, making the hardware non-essential to the continuous uptime of the system.

The last notable client is the garage itself, where the hardware consists of a scanner and a sensor. The scanner both sends information and requests from the server, while the sensor only sends information to the server. Unlike the other clients in this system, the garage hardware is essential to the system and needs to be running at all times for the system to continuously function. Figure 12 has a visual representation of these systems.

9d) Persistent Data Storage

A MySQL database is used to store all information that is used by the components of this project. All information stored in the tables will be modified and/or accessed by the mobile app, manager portal, and scanner systems. This database is comprised of eight separate tables:

1. CustomerInfo: stores basic customer data required to construct an account (i.e. first name, last name, email, phone number, license plate number, vehicle registration number, credit card information, as well as an auto-generated unique user barcode which serves as the user's identification number).
2. Managers: stores credentials of all users considered managers and allowed access to the manager website (i.e. username and password).
3. ParkingSpots: a map of the current status of each parking spot in the garage (i.e. spot number, type [handicap, walk-in, VIP, reserved], and status [occupied or vacant]).
4. Reservations: contains a record of currently active reservations (i.e. date, start time, end time, user barcode, assigned spot number, and total charge amount).
5. Walk-ins: contains a record of current walk-in customers who are parked at the garage (i.e. date, start time, user barcode, assigned spot number, and total charge amount). The end time is not stored in this table, since we don't know exactly when the walk-in customer will leave, so this information is stored in the Records table.
6. Records: contains a history of all reservations that have ever been placed at the garage (i.e. date, start time, end time, user barcode, assigned spot, charge, cancelled/not).
7. Payment: contains the pricing scheme used to calculate a customer's fee (i.e. base price and multiplier for each hour of the day).

8. Points: contains the loyalty program point scheme to calculate a customer's new point balance with each action (i.e. action taken and associated number of points added/detracted).

9e) Network Protocol

The system uses the common network protocol HTTP, or hypertext transfer protocol, for data communication on the manager portal website. Communication between clients and servers will be done specifically using the REST, or representational state transfer, architectural style. The most common HTTP methods used for this system are GET, POST, PUT, and DELETE for system users to obtain, add, and delete information from the MySQL database.

9f) Global Control Flow

Execution Orderliness:

The parking garage system is both procedure-driven and event-driven. The entire process of staying at the garage includes several steps that customers will generally follow. They will log in to the system, or register if they do not have an account, and then they will make a reservation by selecting a date, start time, and end time. Once the reservation date arrives, customers are scanned in, stay for a predetermined period of time, and leave once their reservation expires. These steps make for a procedure-driven system. However, multiple scenarios can trigger events throughout the use of the system. For example, the parking process is initiated once the camera detects an approaching vehicle. The system is waiting for this event to occur before checking for reservations in the database. An unsuccessful license plate scan can signal to a separate barcode code scanner to turn on as an alternate source for verification. Finally, the system can trigger point deductions once a timer detects that a customer has overstayed their booked reservation.

Time Dependency:

The system operates in real-time, regularly comparing the current time to the reservations of incoming customers. This includes timers that track how long individuals stay during their

reservations. The timers are used to ensure that customers arrive and leave on time. If a customer overstays their reservation, then the timer triggers a point penalty to their account.

Concurrency:

The parking garage system does not directly implement multiple threads to handle simultaneous inputs and outputs. Because the system is centralized around the mobile app and the website communicating with the MySQL database, concurrent requests will be handled by the server. Multiple reservations that begin or end at the same time are processed by the database such that they are properly linked with the appropriate accounts.

9g) Hardware Requirements

1. Application/Website Hosting Server
 - a. 1 mb/s connection speed
 - b. 100 mb disk space
2. Database Host
 - a. 10 gb disk space
 - b. 4 gb memory
 - c. 100 mb/s transfer speed
3. Garage Hardware*
 - a. Camera/Scanner
 - b. Parking IR Motion Sensors
 - c. Raspberry Pi Model 3A
 - d. 32'' Displays for Parking Instruction

At the input layer, customers and garage managers will interact directly with the mobile application or website. Both of these will be hosted on local servers with the design specifications listed above.

All of the data entered by the customers and managers will then be stored on the database server hosted by the Rutgers Engineering Computing Services (ECS). System administrators will be given permission to read and write directly to the database specified above.

Inside the garage, a scanner will read in the customer's license plate or QR code. Following a successful scan, the system utilizes a Raspberry Pi with a display to instruct the customer of their specific parking location. Finally, the system monitors occupied and vacant parking spots with infrared motion sensors.

**Note – the garage hardware will be simulated by GUIs for the purpose of this course.*

Section 10: ALGORITHMS AND DATA STRUCTURES

10a) Algorithms

The system has simple algorithms that control parking spot selection, the pricing scheme, and the loyalty program point structure.

Parking Spot Selection

Ideally, a system should limit the number of steps a user must take to interact with the system. Therefore, to achieve a minimalist design, the system selects a parking spot when a customer makes a reservation or enters the garage as a walk-in. If the customer requires parking in one of the special spots (handicap, VIP, etc.), the system will select a spot within the designated area. The system selects a spot for the customer by using a MySQL “Update” statement. The first row with a “Status” of “Vacant” is selected for the customer, and the status is changed to “Occupied” in the “Parking Spots” and “Reservations” tables. Parking spots are represented in a linear list fashion so that the software is not restricted to a certain shaped garage.

Dynamic Pricing Model

This system allows managers to adjust the pricing scheme manually or automatically. Managers may choose to adjust prices manually given their own knowledge about the current economic situation or customer habits. They instead may choose to see a suggested pricing scheme that the system automatically develops using past parking garage data. Both methods are detailed below.

Manual Pricing

Research and calculations done by Fall 2018 Group 3 have determined that parking garages reach peak usage from 7:00 a.m. to 1:00 p.m [1]. This conclusion is made under the assumption that a single customer reserves only one spot during a span of time, disregarding group reservations [1]. This mathematics of this model is based off algorithms from the paper, “Dynamic Pricing for Reservation-Based Parking System [10],” and are modeled to work in a simple case. The model also accounts for customers that act with conditions described in the

paper, “Optimal Dynamic Pricing of Inventories with Stochastic Demand Over Finite Horizons [4].” Their purpose is to find available parking rather than find the cheapest parking. However, this model considers only a single demand function, and would not be valid for special cases such as events that bring more drivers into the area.

Thus, we divide the parking into three intervals: regular-priced parking, surge-priced parking, and discounted parking. During hours where there is no peak or lull in parking garage usage, the price will be set at an arbitrary dollar amount per hour x , to be set by the manager of the garage. During peak hours when the demand for the parking garage spots is greater than its supply, then the price for this interval will be x multiplied by a factor of 1.6. This is based off of cost analysis from a previous study which states that an increase of less than a factor of 0.5 would not decrease the demand enough while a factor of 0.8 would cause there to be too much decreased demand [1]. A multiplier of 0.7 will be placed on x to calculate the price for discounted parking, which is applied during times when the parking garage does not expect as many customers in order to attract more customers. This factor was determined based on calculations that state a decrease by a factor of 0.5 would increase demand too greatly [2].

Considering data collected from the previous study [2], we set surge-priced parking to be from 7:00 a.m. to 1:00 p.m. Regular-priced parking will be from 12:00 a.m. to 3:00 a.m. and 4:00 p.m. to 12:00 a.m. Parking is charged at a discounted rate from 3:00 a.m. to 7:00 a.m. and 1:00 p.m. to 4:00 p.m. On the manager website, the parking garage manager can set a base price per every 15 minutes and multiplier values for different times of the day. A customer’s total fee is determined by the following equation:

$$\text{Customer Fee} = \text{Base Price} * \text{Number of 15 minute intervals} * \text{Hourly Multiplier}$$

Given these factors, a manager may choose to set the following sample parking price schedule:

Start Time	End Time	Price (\$/hr)
12:00 a.m.	3:00 a.m.	x
3:00 a.m.	7:00 a.m.	$0.7x$
7:00 a.m.	1:00 p.m.	$1.6x$
1:00 p.m.	4:00 p.m.	$0.7x$
4:00 p.m.	12:00 a.m.	x

Table 27. Sample pricing scheme.

Note, the above values are simply suggestions for a strong pricing model based on research. The value of x is a fixed, base price set by the managers. Additionally, the multipliers of 0.7 and 1.6 may be adjusted as a manager sees fit as well. For example, these multipliers may be greater if we factor in the cost of database maintenance.

Automatic Pricing Suggestion

Alternatively, a manager may choose to use pricing suggested by the system. The system automatically develops a pricing scheme using parking garage data. The hourly multiplier is determined by the following equation, which was developed through research of successful dynamic pricing algorithms [6]:

$$\text{Multiplier} = \frac{\text{Average capacity per hour for hour } X}{\text{Average capacity per hour for all hours}}$$

The multiplier value has a minimum of 0.5 and a maximum of 3. The manager may click the button “Suggest prices” to view the system-suggested scheme. Then manager may decide to save this scheme as the new pricing or ignore it.

Reward/Penalty System Model

This project utilizes a simple model for distributing and deducting points as a part of our reward and penalty system. The following is a table that dictates guidelines for rewarding, deducting, and exchanging points. Outlined are the action, points exchanged, and description of the action. The action depicts whether the points received were for a reward, penalty, rectification, or exchange. An addition symbol in front of the point value signifies that points are added to the registered account while the subtraction symbol signifies that points are deducted. The point values were arbitrarily decided based on the weight of each action; hence, showing up on time to a reservation gains a customer five points but having a reoccurring reservation would yield more points (15). The description of each action describes the conditions for the exchange of points.

Action	Points	Description
Reward	+10	Customer makes a reservation

Reward	+15	Customer makes a reoccurring reservation (for every 30 reservations)
Reward	+5	Customer shows up to reservation within 5 minutes of their start time
Penalty	-5	Customer stays longer than their reserved time
Rectification	-10	Customer cancels a reservation (to reconcile for the 10 points gained from making a reservation)
Exchange	-100	Credit to receive a preferred parking spot (closer to the entrance/exit)

Table 28. Loyalty program point scheme.

As a customer fulfills a task, a MySQL “Update” statement will adjust the number of points that customer has in the “Customer Info” table accordingly. For instance, a manager may assign ten points to the action of making a reservation. Therefore, when a hypothetical customer with barcode ID, 1234, makes a reservation, the value in the customer’s “Points” column will increase by ten. The example MySQL statement is shown:

```
UPDATE CustomerInfo
SET points = points + 10
WHERE barcode = 1234
```

10b) Data Structures

None of the subsystems use data structures. However, all subsystems interact with a MySQL database on the server provided by Rutgers ECS. This database has multiple tables that store all of the information that our subsystems access and modify using MySQL statements such as “Select,” “Update,” “Insert,” and “Delete.” See the section “Persistent Data Storage” for more detailed descriptions of each of the database tables.

Note that contrary to past projects, the parking garage is not simulated as a two-dimensional array. Instead, there is a database table named “ParkingSpots” with three columns, “SpotNum,” “Type,” and “Status.” For the sake of this explanation, let us pretend that the garage has 30 spots in total. “ParkingSpots” is populated with 30 entries numbered one to 30. The first 5 entries have Type “Handicap,” the next are “Walk-In,” the next are “VIP,” and the rest are “Reserved.” The “Status” field is changed to “Vacant” or “Occupied” based on the occupancy

status of that spot at the present time. Therefore, this table serves as a “real-time view” of the garage. Each parking spot in the physical garage is assigned a number that corresponds to a row with that “SpotNum” value. We chose to represent the garage in a fashion that does not mirror the physical building so that our software can easily adapt to different garages. Parking garages can be square, rectangle, or L-shaped, as well as differ in overall capacity. It would have been difficult to adapt a two-dimensional array to these various scenarios, while it is simple to add more parking spot entries to our table. Parking spot representation in a linear list fashion does not restrict our software to a certain shaped garage.

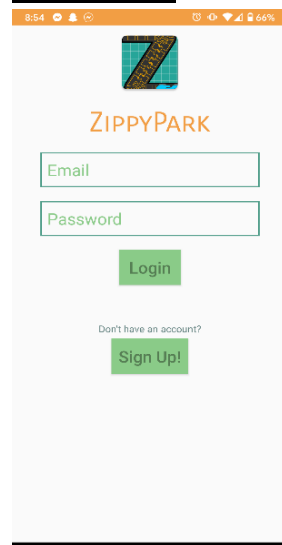
Section 11: USER INTERFACE DESIGN & IMPLEMENTATION

Customer Services (Mobile App)

Description: The mobile application allows the customer to interact with the system. The app focuses on the customer experience with a minimalistic design that requires the least number of actions from the customer. The goal is to provide a straightforward interface that does not overwhelm the customer with unnecessary features. An Android rather than an iOS application was designed because while iPhones are popular in the U.S., Android dominates in terms of market share worldwide. Since as of March 2020, 72.26% of the sector is Android, it seemed appropriate to begin with an Android app for the first iteration of the product [9].

Modifications: The mobile app interface design we had initially displayed in the project proposal consisted of several screens depicting a web application. However, we have made several modifications to our project since then and these design choices are shown below in images of the final application. The significant change from the proposal is that customers will use a mobile phone application instead of a web application. All other changes are mainly stylistic alterations, such as choosing a new color palette. The original mockups were already designed to reduce the user effort, so our updated interface was created with the old setup in mind. These new mockups depict the activities which will make up the customer's mobile application.

Use Case #1



Login Page

Upon initial installation of the app, the user will be asked to login with the email and password associated with their existing account. In the case that they do not have an account, they are directed to the “Sign Up!” button at the bottom of the screen which leads to the registration form activity.

Worst case: 1 button click, variable number of keystrokes

Figure 13. Customer UI - Login Page.

Figure 14. Customer UI - Create Account Page (part 1).

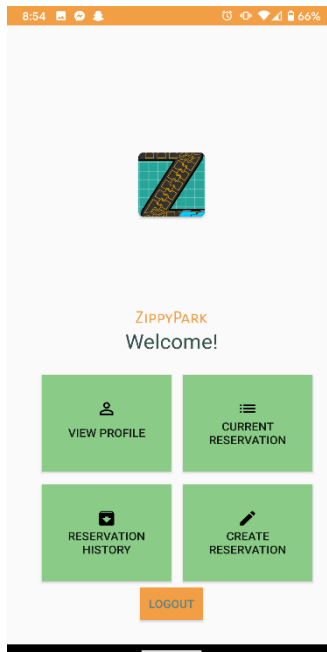
Figure 15. Customer UI - Create Account Page (part 2).

Registration Form

This activity contains a scrolling list, consisting of text fields for the user to fill out with the information necessary in order to create an account. Each text field contains a hint, indicating to the user what information is required to be inputted into that field. Once the form has been completed, the user can press the “NEXT” button, which will lead to the Home Screen. All fields will be checked for properly formatted responses. In the case that the “NEXT” button is pressed but certain fields are found empty or improperly formatted, then the user will be prompted to correct these errors and the new account will not be created until all errors have been resolved.

Worst Case: 1 click, variable number of keystrokes

Use Case #2, 3, 4, 5, 6



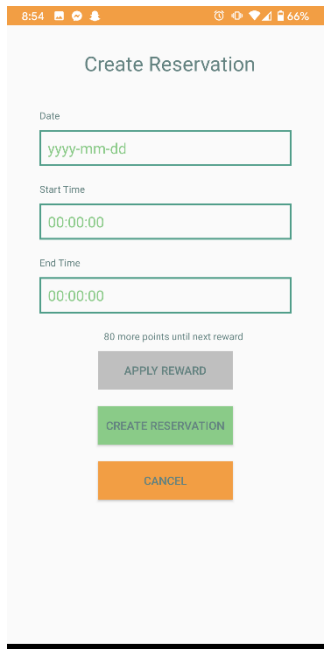
Home Screen

The Home Screen of the ZippyPark application serves as the user's main dashboard for all key actions. At the top of the screen, the app name is displayed along with a text that will be customized to display the customer's first name. Below that they can select to perform a specific action. Each button will lead to a separate activity where the user can complete that action.

Worst Case: 1 button click

Figure 16. Customer UI - Home Screen.

Use Case #4, 12



Create Reservation

If a user chooses to create a new reservation, they will be directed to this screen. They can input the date, starting time, and end time in the format prompted through the text fields. If they have greater than 100 points in their account, the APPLY REWARD button turns green prompting the user to apply a reward to their reservation. This means they will be assigned a VIP spot once they arrive at the garage. After they are satisfied with the details they have provided, pressing the CREATE RESERVATION button will take them back to the Home Screen.

Worst Case: 2 button clicks, variable number of keystrokes

Figure 17. Customer UI - Create Reservation Screen.

Use Case #5, 6

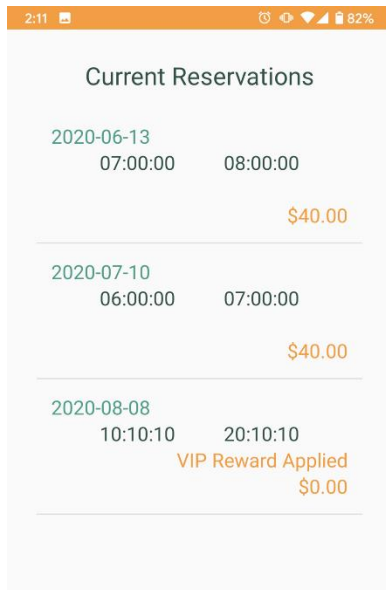


Figure 18. Customer UI - Current Reservations Screen.

Current Reservation

If a user chooses to view their current reservations, they will be directed to this screen. Each list item represents an individual reservation, which displays the data, start time, end time, reward status, and price of the reservation. Upon clicking a reservation in the list, the user can choose to edit or delete that reservation.

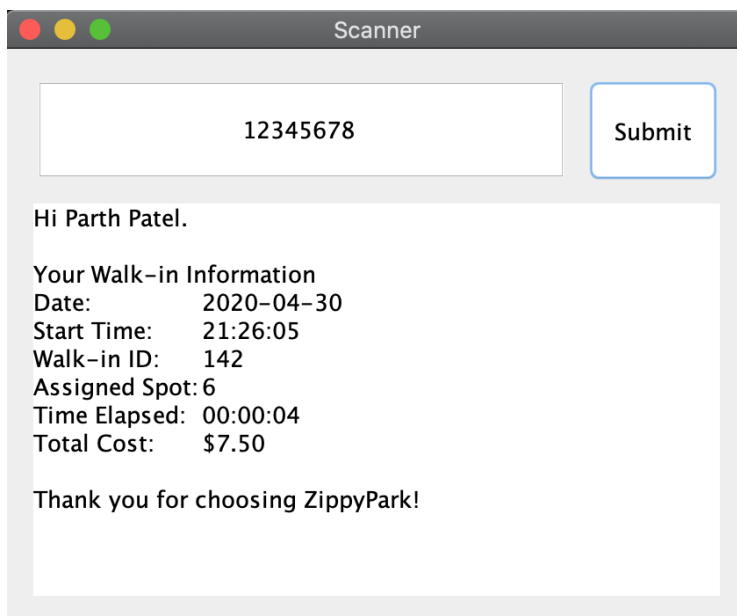
Worst Case: 1 button click

Arrival & Departure Operations (Simulate Scanners/Sensors)

Description: The scanner application simulates the machine that would be present in the parking garage. It was designed to require the minimum number of actions from the customer, since requiring too many inputs would inhibit the flow of customers entering the garage. The displayed interface will promote the smoothest experience for customers as they use the parking garage.

Modifications: None. A scanner interface was not planned at the time in the project proposal.

Use Case #7, 8, 9, 12

A screenshot of a web application window titled "Scanner". At the top, there are three colored circles (red, yellow, green) and the title "Scanner". Below this is a text input field containing the number "12345678". To the right of the input field is a blue button labeled "Submit". Below the input field and button is a large white box containing the following text: "Hi Parth Patel.", "Your Walk-in Information", "Date: 2020-04-30", "Start Time: 21:26:05", "Walk-in ID: 142", "Assigned Spot: 6", "Time Elapsed: 00:00:04", "Total Cost: \$7.50", and "Thank you for choosing ZippyPark!".

Scanner

12345678

Submit

Hi Parth Patel.

Your Walk-in Information

Date: 2020-04-30

Start Time: 21:26:05

Walk-in ID: 142

Assigned Spot: 6

Time Elapsed: 00:00:04

Total Cost: \$7.50

Thank you for choosing ZippyPark!

Figure 19. Scanner UI.

Scanner Screen

The scanner interface is displayed for each incoming customer, prompting them to display their unique barcode to the camera so they can enter or exit the garage. For the purposes of this course, the camera is simulated by the text field at the top of the screen. The numerical code is typed in, and the blue “Submit” button at the top right is pressed to send the code to the system. After a brief delay for processing, the white box at the bottom of the screen will output information corresponding to the entered barcode.

Worst Case: 1 button click, variable number of keystrokes

Managerial Tasks (Website)

Description: The design of the website provides access to every managerial action from the dashboard so that owners can get to the desired page quickly. The dashboard serves as the central hub for the managers. From here, they can access the statistics, a list of current reservations, and a list of registered customers by clicking the corresponding tab along the left of the screen.

Modifications: None. A scanner interface was not planned at the time in the project proposal.

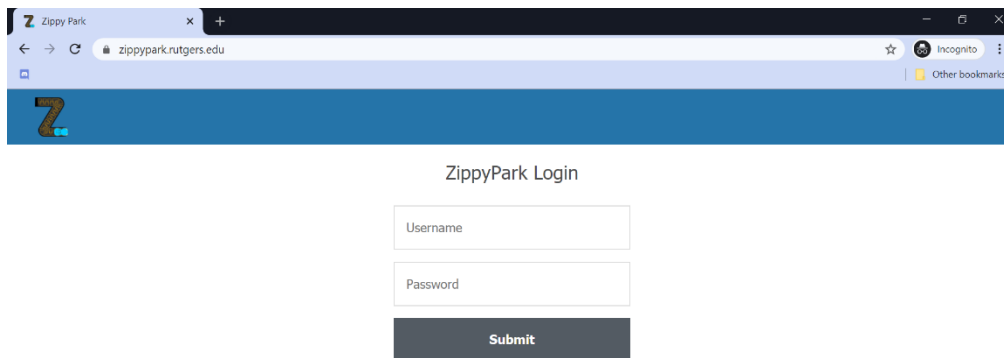


Figure 20. Manager UI - Login Page to Manager Website.

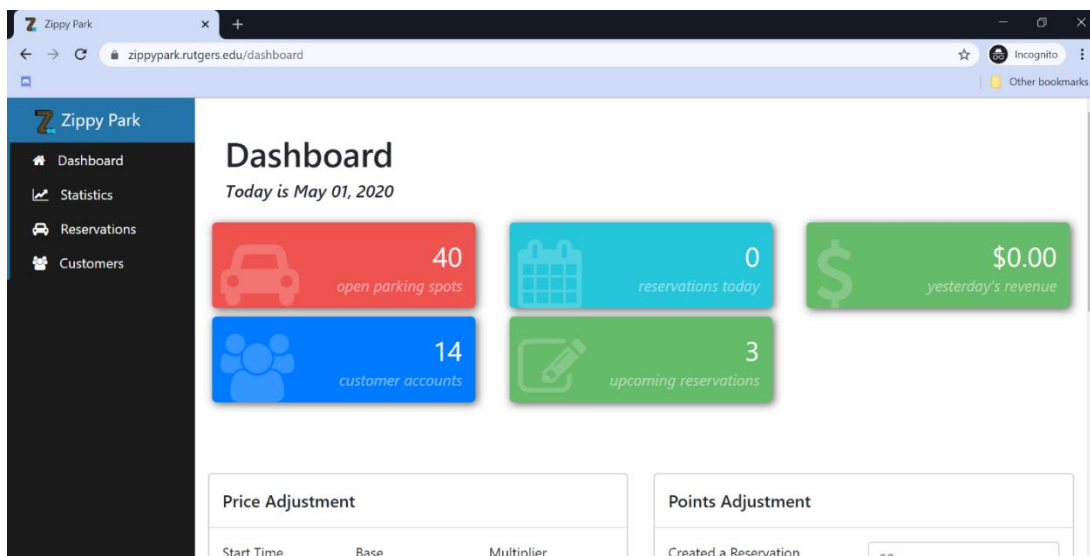


Figure 21. Manager UI - Dashboard Page.

In addition to basic garage information, the dashboard also contains the modals for points and price adjustment.

Worst Case: 1 button click

Use Case #1, 2, 3

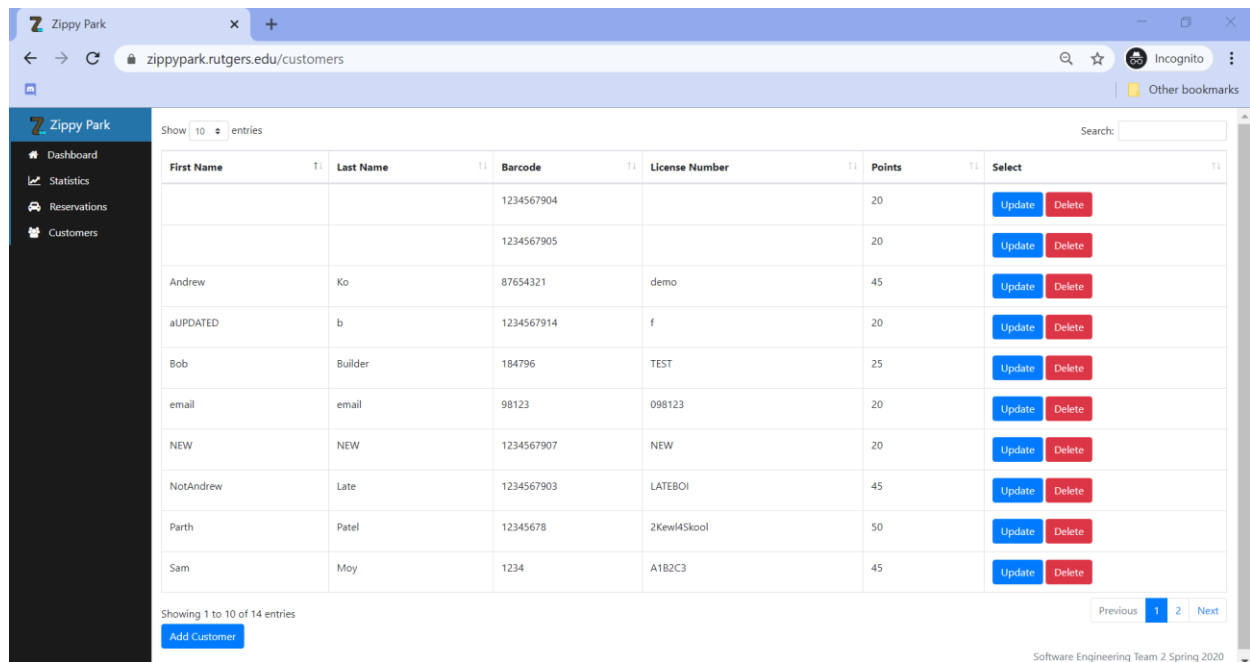


Figure 22 shows the Manager UI - Customers Page. The page displays a table of customers with the following columns: First Name, Last Name, Barcode, License Number, Points, and a Select column. The Select column contains 'Update' and 'Delete' buttons for each customer. The table is sorted by Points in descending order. A sidebar on the left contains navigation links for Dashboard, Statistics, Reservations, and Customers. A search bar is located at the top right, and a pagination bar is at the bottom right.

First Name	Last Name	Barcode	License Number	Points	Select
		1234567904		20	Update Delete
		1234567905		20	Update Delete
Andrew	Ko	87654321	demo	45	Update Delete
aUPDATED	b	1234567914	f	20	Update Delete
Bob	Buider	184796	TEST	25	Update Delete
email	email	98123	098123	20	Update Delete
NEW	NEW	1234567907	NEW	20	Update Delete
NotAndrew	Late	1234567903	LATEBOI	45	Update Delete
Parth	Patel	12345678	2Kewl4Skool	50	Update Delete
Sam	Moy	1234	A1B2C3	45	Update Delete

Figure 22. Manager UI - Customers Page.

Customers Page

The customer tab displays a sorted list of all customers registered with ZippyPark. The drop-down box at the top left lets managers view more customers on a single page. The text field at the top right acts as a search bar, pulling up customers who match the entered key. Along the first row of the table, the manager may click the arrows to change the sorting algorithm and to choose whether the information is displayed in ascending or descending order. Below this, managers may opt to edit or delete a chosen customer's account.

Create Account

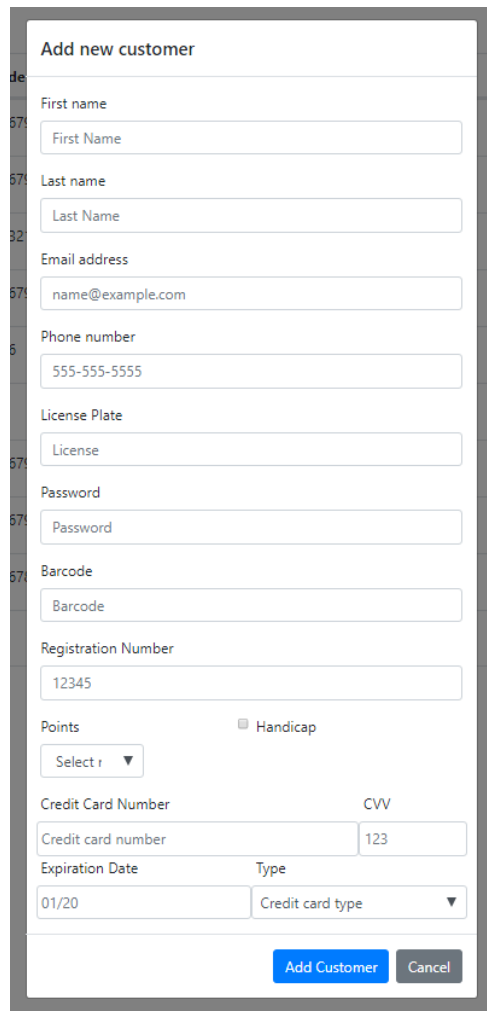
Clicking the "Add Customer" button at the bottom left will bring up a modal where the manager can populate the fields with a new customer's information. Once the manager is finished, they may click the "Add Customer" button at the bottom of the modal which will add the given information to the database, or they may click the "Cancel" button to close the modal without changing the database.

Update Account

Clicking the “Update” button will bring up a modal where the manager can populate the fields with updated information. The fields are pre-populated with the customer’s current information. Once the manager is finished, they may click the “Update” button which will update the database with the given information, or they may click the “Cancel” button to close the modal without changing the database.

Delete Account

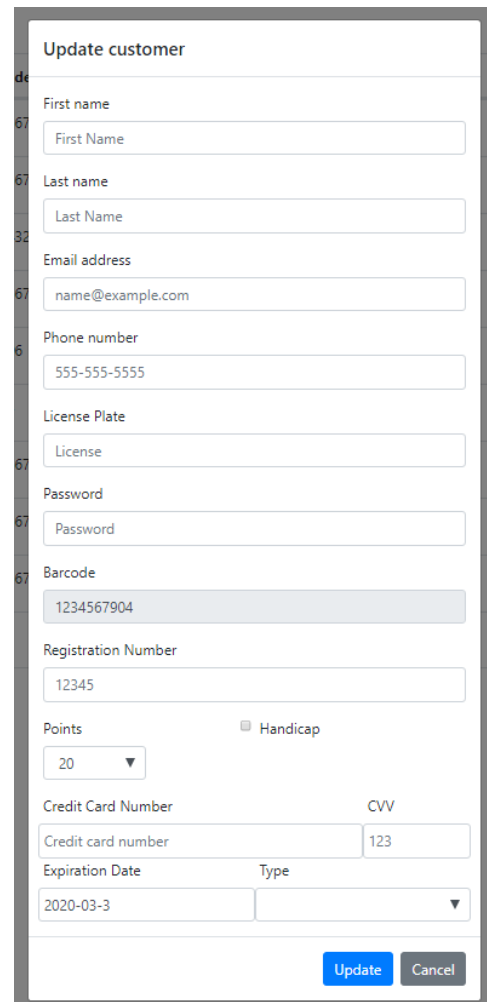
Clicking the “Delete” button will bring up a modal that asks for confirmation of the deletion. The manager must then click the “Delete” button to delete the entry from the database or the “Cancel” button to close the modal without deletion.



The "Add new customer" modal form contains the following fields and controls:

- First name: Text input with placeholder "First Name"
- Last name: Text input with placeholder "Last Name"
- Email address: Text input with placeholder "name@example.com"
- Phone number: Text input with placeholder "555-555-5555"
- License Plate: Text input with placeholder "License"
- Password: Text input with placeholder "Password"
- Barcode: Text input with placeholder "Barcode"
- Registration Number: Text input with placeholder "12345"
- Points: Dropdown menu with "Select r" and a downward arrow
- Handicap: Checkbox
- Credit Card Number: Text input with placeholder "Credit card number"
- CVV: Text input with placeholder "123"
- Expiration Date: Text input with placeholder "01/20"
- Type: Dropdown menu with "Credit card type" and a downward arrow
- Buttons: "Add Customer" (blue) and "Cancel" (gray)

Figure 23. Manager UI - Create Account Modal.



The "Update customer" modal form contains the following fields and controls:

- First name: Text input with placeholder "First Name"
- Last name: Text input with placeholder "Last Name"
- Email address: Text input with placeholder "name@example.com"
- Phone number: Text input with placeholder "555-555-5555"
- License Plate: Text input with placeholder "License"
- Password: Text input with placeholder "Password"
- Barcode: Text input with placeholder "1234567904"
- Registration Number: Text input with placeholder "12345"
- Points: Dropdown menu with "20" and a downward arrow
- Handicap: Checkbox
- Credit Card Number: Text input with placeholder "Credit card number"
- CVV: Text input with placeholder "123"
- Expiration Date: Text input with placeholder "2020-03-3"
- Type: Dropdown menu with a downward arrow
- Buttons: "Update" (blue) and "Cancel" (gray)

Figure 24. Manager UI - Edit Account Modal.

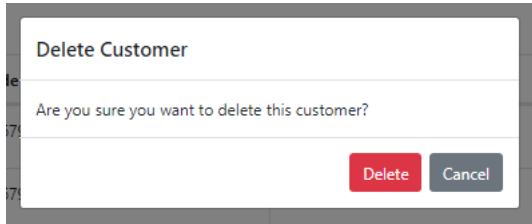


Figure 25. Manager UI - Delete Account Modal.

Updating an account results in the worst-case number of inputs, since it pulls up a scrolling list with text fields and a confirmation button at the bottom. Clicking “Delete” will permanently remove the customer account directly to the left, and the webpage will refresh to show the change.

Worst Case: 2 button clicks, variable number of keystrokes

Use Case #4, 5, 6

 A screenshot of the Zippy Park Manager UI Reservations Page. The page has a dark sidebar with navigation links: Dashboard, Statistics, Reservations, and Customers. The main content area shows a table of reservations with columns: Date, Start Time, End Time, Barcode, Assigned Spot, Charge, rID, and Select. There are three rows of data. Below the table, it says "Showing 1 to 3 of 3 entries" and has an "Add Reservation" button. At the bottom right, it says "Software Engineering Team 2 Spring 2020".

Date	Start Time	End Time	Barcode	Assigned Spot	Charge	rID	Select
Fri Oct 09 2020	13:10:00	14:15:00	2345678	-1	25	265358995	Update Delete
Tue Jul 14 2020	09:00:00	10:00:00	2345678	-1	40	134636	Update Delete
Tue Sep 15 2020	11:00:00	12:00:00	2345678	-1	20	134642	Update Delete

Figure 26. Manager UI - Reservations Page.

Reservations Page

The reservations page lists all pending reservations from registered accounts. The drop-down box at the top left changes the number of reservations shown on a single page, while the text field at the top right hides all reservations that do not match the entered key. The top row of the table has arrows in each cell, allowing managers to change the category the table is sorted by, as well as whether it is in increasing or decreasing order. In the right-most column, the manager may click the blue “Update” button to edit a reservation’s details or the red “Delete” button to permanently remove the reservation.

Create Reservation

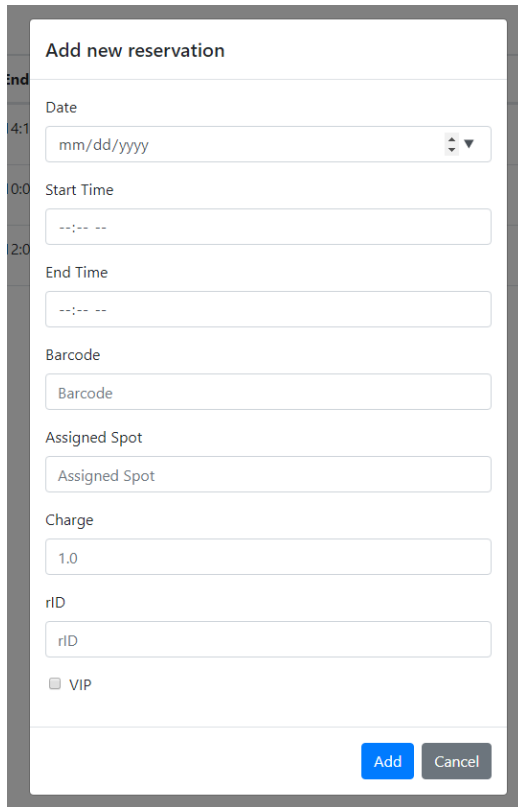
Clicking the “Add Reservation” button at the bottom left will bring up a modal where the manager can populate the fields for a new reservation. Once the manager is finished, they may click the “Add” button at the bottom of the modal which will add the given information to the database, or they may click the “Cancel” button to close the modal without changing the database.

Update Reservation

Clicking the “Update” button will bring up a modal where the manager can populate the fields with updated information. The fields are pre-populated with the current reservation information. Once the manager is finished, they may click the “Update” button which will update the database with the given information, or they may click the “Cancel” button to close the modal without changing the database.

Delete Reservation

Clicking the “Delete” button will bring up a modal that asks for confirmation of the deletion. The manager must then click the “Delete” button to delete the entry from the database or the “Cancel” button to close the modal without deletion.



Add new reservation

Date
mm/dd/yyyy

Start Time
--:-- --

End Time
--:-- --

Barcode
Barcode

Assigned Spot
Assigned Spot

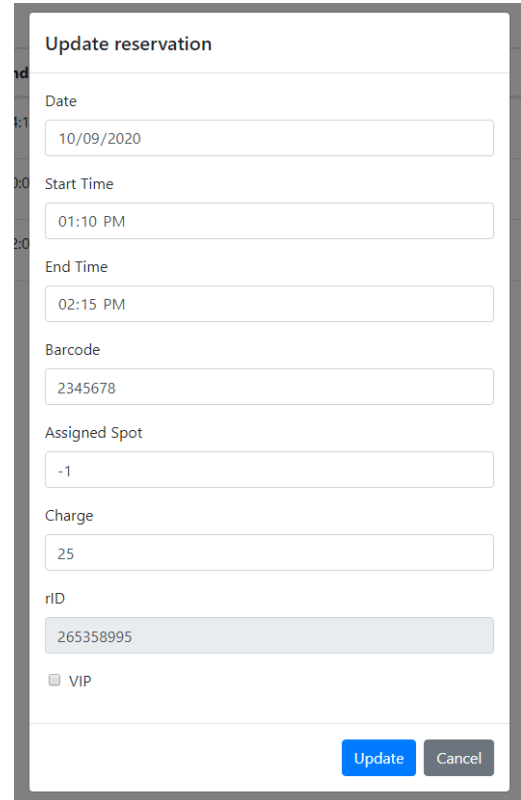
Charge
1.0

rID
rID

☐ VIP

Add **Cancel**

Figure 27. Manager UI - Create Reservation Modal.



Update reservation

Date
10/09/2020

Start Time
01:10 PM

End Time
02:15 PM

Barcode
2345678

Assigned Spot
-1

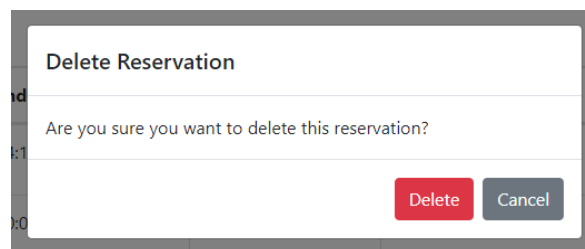
Charge
25

rID
265358995

☐ VIP

Update **Cancel**

Figure 28. Manager UI - Edit Reservation Modal.



Delete Reservation

Are you sure you want to delete this reservation?

Delete **Cancel**

Figure 29. Manager UI - Delete Reservation Modal.

The worst-case arises from updating reservations, since clicking “Update” pulls up a list of text fields. These have been auto filled with the selected reservations information, but they can be changed. The manager will then need to click “Update” to confirm their changes.

Worst Case: 2 button clicks, variable number of keystrokes

Use Case #10

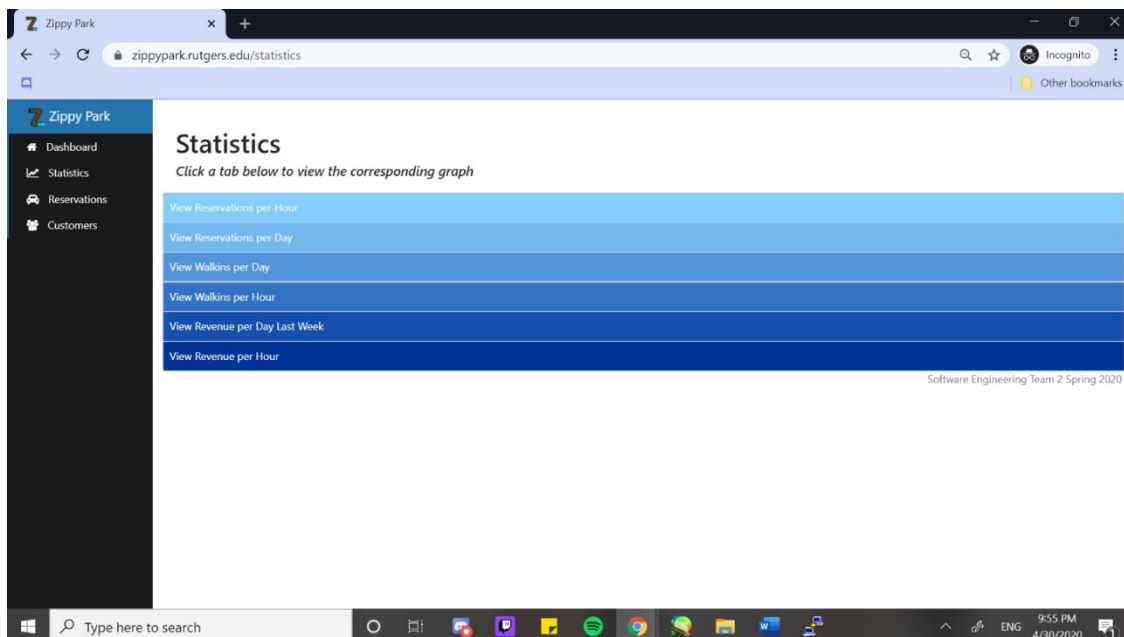


Figure 31. Manager UI - Statistics Page (collapsed charts).

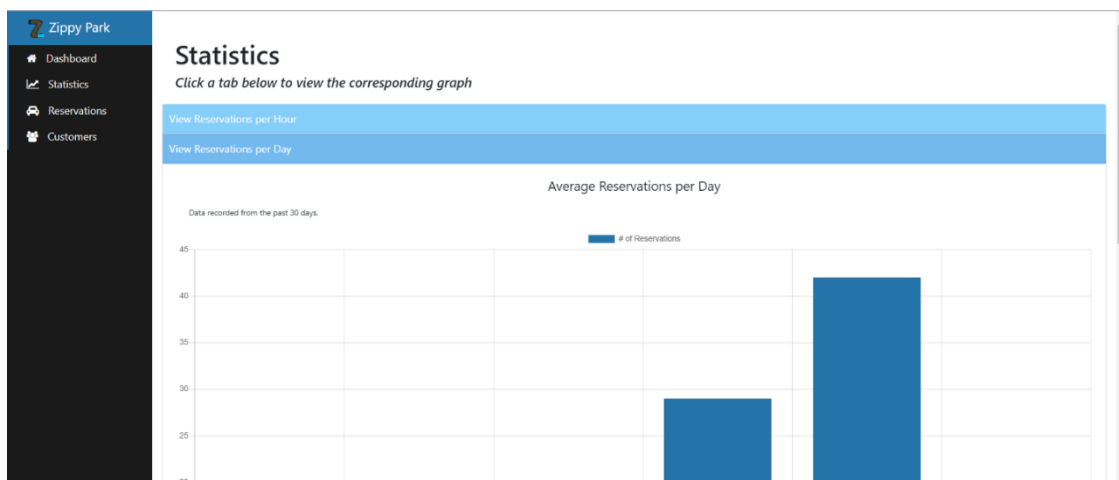


Figure 30. Manager UI - Statistics Page (expanded charts).

Statistics Page

The statistics tab displays various charts to aid the manager in analyzing garage data. These charts include the following: reservations per hour, reservation per day, walk-ins per day, walk-ins per hour, revenue per day, and revenue per hour. The charts displaying reservations and walk-ins by day and hour compile data from the past 30 days. The revenue charts display revenue over the last week per day and per hour. Below each chart lists the data for each chart in

the form of a table. If the entries exceed the maximum to be visible, the manager can navigate through them using “Previous” and “Next” buttons.

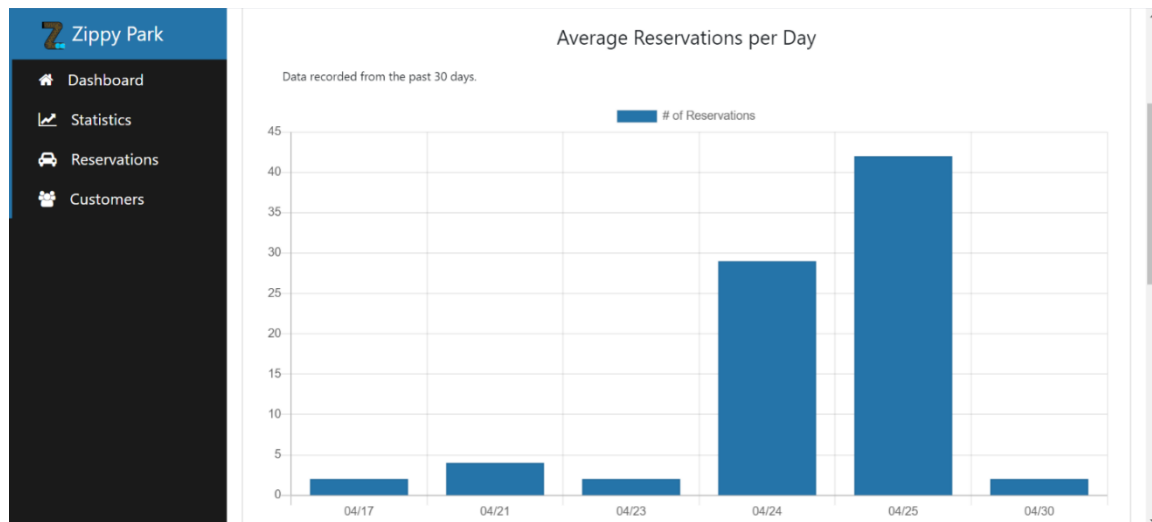


Figure 32. Manager UI - ex. Average Reservations per Day Chart.

04/17 04/21 04/23 04/24 04/25 04/30

Show 10 entries Search:

Date	Number of Reservations
04/17	2
04/21	4
04/23	2
04/24	29
04/25	42
04/30	2

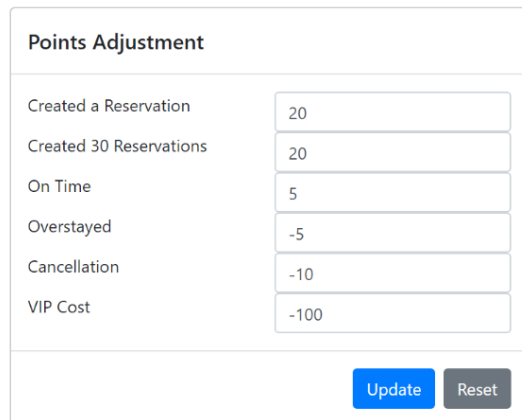
Showing 1 to 6 of 6 entries Previous 1 Next

Figure 33. Manager UI - ex. Average Reservations per Day Table.

Expanding all the graphs would results in the worst-case number of inputs.

Worst Case: 6 button clicks

Use Case #11



The 'Points Adjustment' modal is a form with a title bar. It contains six rows, each with a label on the left and a text input field on the right. The labels are 'Created a Reservation', 'Created 30 Reservations', 'On Time', 'Overstayed', 'Cancellation', and 'VIP Cost'. The input fields contain the values '20', '20', '5', '-5', '-10', and '-100' respectively. At the bottom right of the modal are two buttons: 'Update' (blue) and 'Reset' (grey).

Category	Value
Created a Reservation	20
Created 30 Reservations	20
On Time	5
Overstayed	-5
Cancellation	-10
VIP Cost	-100

Figure 34. Manager UI - Points Adjustment Modal.

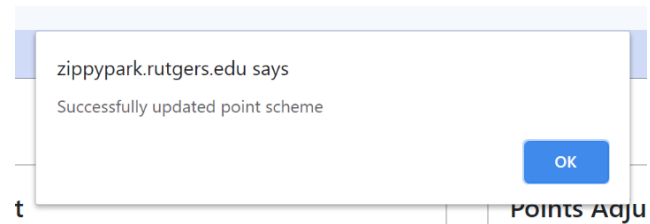


Figure 35. Manager UI - Points Adjustment Confirmation.

Points Adjustment Modal

The points adjustment modal can be used to change the value of each loyalty reward function. The current point values are pre-populated in the corresponding fields. The manager can enter the new point value for each deduction or reward and click the “Update” button to make changes to the database or click the “Reset” button if they wish to clear the changes. There is a pop-up notification once the database has been updated.

The worst-case arises from updating point values, since clicking “Update” would require one click and another click to close the pop-up notification.

Worst Case: 2 button clicks, variable number of keystrokes

Use Case 12:

Start Time	Base	Multiplier
6:00	5	0.5
7:00	5	0.5
8:00	5	0.5
9:00	5	0.5
10:00	5	0.5
11:00	5	0.5
12:00	5	0.9
13:00	5	0.8
14:00	5	0.8
15:00	5	1.7
16:00	5	1.5
17:00	5	2.2
18:00	5	2.3
19:00	5	1.7
20:00	5	1.6
21:00	5	1.9
22:00	5	1.8

Figure 36. Manager UI - Price Adjustment Modal.

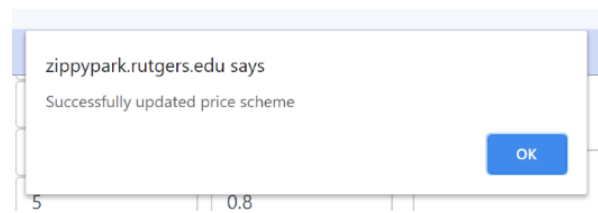


Figure 37. Manager UI - Price Adjustment Confirmation.

Price Adjustment Modal

The price adjustment modal can be used to change the price point for each hour. The fields are pre-populated with the current prices. The manager may enter the new price for any hour they choose or click the “Suggest Price” button which will populate all the fields with suggested values based on a pricing algorithm that uses historical data of the garage’s use to maximize profit. To make changes to the database, the manager must click the “Update” button, or they can click the “Reset” button if they wish to clear the changes. There is a pop-up notification once the database has been updated.

The worst-case arises from updating price values using suggested prices, since clicking “Suggest Price,” clicking “Update,” and closing the pop-up notification would each require one click.

Worst Case: 3 button clicks, variable number of keystrokes

Section 12: DESIGN OF TESTS

Use Case Testing

The test plans for all of the use cases promised in the final product are listed.

UC #1 – Create Account

Both a user and manager have the ability to create an account. A user should be able to register for an account by supplying necessary personal, payment, and vehicle information. Their email address and password are needed every time they log in to their account. They would additionally be assigned a unique barcode attached to their account.

Test Coverage: Upon pressing the “Sign Up” button in the app, 3 different scenarios are attempted. First, every field is correctly filled, and when submitted, the home screen should appear. We should check the database to see if the user account is displayed. Second, we attempt to add incorrect information, such as a word in the phone number field, which should not be allowed by the system. Third, we should attempt to leave a field blank, which should not be allowed by the system either. In the manager website we check this test by creating a user account by correctly filling all the fields, and it will be reflected in the database.

UC #2 – Edit Account

This functionality is present for both the user and the manager. A user should be able to edit their account whenever they need. They should be able to update any field of their account.

Test Coverage: On the profile screen of the app, one field of the account is updated at a time (13 fields total) and the database is checked to ensure that the change was saved. The change should also be reflected on the profile screen of the app. For instance, if the email is changed, the new email should be seen in the database, as well as the updated profile screen. On the manager website, on the Customers webpage, we can select a user and change any information such on their account besides the account barcode and save

the changes. Once again, this change should be seen in the database as well as being updated on the website.

UC #3 – Delete Account

Both a customer and manager have the ability to delete an account. The customer may delete their account through the app. A manager may delete an account through the manager website, which may be done in the case of an inactive account.

Test Coverage: We attempt to delete an account as a customer and as a manager. In both cases, the user should receive a confirmation of deletion. The database will then no longer hold information on this account with the exception of the records table.

UC #4 – Create Reservation

A user should be able to create a reservation for a parking spot for a selected time. They can select a date and time by looking at the calendar which shows an updated version of the amount of reservations there are currently. The user receives a pop-up notification in the app for their confirmed reservation. They are also able to view the amount charged in their reservation list.

Test Coverage: On the mobile app, we test if after the reservation is made, the user gets a notification for the reservation and is able to view it in their current reservation list along with the price they were charged. On the manager website, we create a new reservation on the Reservations webpage by filling out the modal form that displays. For both, the new reservation made should be available in the database correctly as well as on the respective screens for the app and the website.

UC #5 – Edit Reservation

A manager or customer should be able to edit an upcoming reservation whenever they need. The user can move the date of their reservation or change the start time or end time of the reservation.

Test Coverage: On the mobile app, a test case for this scenario is to create a reservation and then attempt to change the time, date, or duration. If the desired time is available, the changes will be made and the database updated with the new information in place of the old. If the duration is changed, the user will see an increase/decrease in what they owe. These updated changes are also visible to the user in their Current Reservation screen. If not available, the database is not updated. On the website, the test case is to navigate to the Reservation webpage and click edit on an entry. This brings up a modal form with the current information, which can then be edited. The changes are reflected in the database.

UC #6 – Delete Reservation

The manager or customers should be able to delete a reservation and receive a full refund. They would receive a pop-up message for confirmation of the deleted reservation.

Test Coverage: For the reservation, we test deleting an upcoming reservation to see if the user receives notification of the cancellation and if it is omitted from the user's Current Reservation screen. We also check that the user is notified that they received a full refund. On the manager website, we test by deleting a reservation from the table on the Reservations webpage. The databases are updated to no longer hold information regarding these reservations with the exception of the records which show the reservations as cancelled.

UC #7 – Enter Garage

A user should be able to easily drive onto the elevator with the scanner being able to read the user's barcode. The elevator should then move to the correct floor and the user should be prompted to which parking spot to navigate to.

Test Coverage: There are multiple scenarios that need to be covered. First, the user may or may not have a reservation, which the scanner system must determine by looking for existing reservations connected to the barcode. Both scenarios will be tested. If the user has a reservation the elevator should move to an appropriate floor and direct the user to an empty spot. If the user has no reservation, the elevator should remain on the first floor and check if any spots are available. If a spot is available, the user should be directed to it otherwise, the user should be told to exit the garage. The test coverage takes in different barcodes as inputs and outputs the location the user should go to.

UC #8 – Exit Garage

A user should be able to easily drive out of the garage from any floor. When the user drives out, the system should be able to tell what user drove out, and then compile a list of statistics including total duration of the stay, type of reservation, and if the user was late in leaving or not. This information should then be visible to the user in their past reservations.

Test Coverage: To test whether the statistics are working correctly, a list of different users will enter and exit the garage. As each user exits the garage, the system should be able to scan their barcode, and based on this ID, tell which user left the garage, when they parked, when they left, and what their reservation status was (duration, not reserved, etc.). This information should match the entry in the database with the corresponding ID. The list of statistics shown should be specific to the user scanning out.

UC #9 – Update Spot Status

The system should be able to tell when a spot is currently occupied by a user and when it is not. Every spot in the garage should be monitored while the system is running. The sensors at each spot should also be able to tell at what time a user parks in a spot, and then transitions that spot to an occupied state. The sensors should be able to tell at what time a user leaves a spot, and then transitions that spot to a free state, as well. These times should then be stored in the database.

Test Coverage: To test whether the parking spot's availability is being updated correctly we will "park" a car at a certain spot and see if that spot is now marked as occupied in the database. If we try to park another car at this spot, it should not be possible. When the car "leaves" the spot, we will check if that spot is now marked as free in the database. If we try to now park another car at this post, it should succeed. One final test is to fill up the entire walk-in floor, and then try to park another car at that floor. In this scenario the system should report that the floor is entirely occupied.

UC #10 – Display Statistics

The parking manager should be able to view parking trend data. This feature is provided by the managerial website whose access is limited to parking administration. The managerial website has a single page dedicated solely to garage statistics. The parking garage records are taken and that data is compiled into statistics which can be viewed in the form of graphs and other data models for certain time frames determined by the manager.

Test Coverage: Testing the statistics display will require the database to be populated with parking garage records spanning at least the past month. This use case is tested by navigating to the Statistics webpage of the manager website, and clicking on each title in the accordion display to see whether the graphs and tables are displayed properly. Automated testing of the data routes, which is detailed in "Automated Testing," is used to ensure the data is correct.

UC #11 – Points Management

The parking manager should be able to adjust the reward scheme from the managerial website. This feature includes setting the number of points added/detracted for each action, as well as managing trading for an award. An optional feature is allowing the manager to view, add, or take away points from a customer manually. This feature would be in place in case the automatic point calculation fails.

Test Coverage: The manager sets the point scheme on the Dashboard webpage. To test this use case, we set a condition for ten points to be exchanged for a certain service from the Dashboard feature. A test customer account is created, from which an action is executed. Ten points should be automatically loaded on the test account. From the customer account, the points should be able to be exchanged for VIP access (at a point cost determined by the manager, say ten points as well). On the customer app or managerial website, a user should be able to see the adjusted point balance after the exchange occurs (zero). To test the manual points management feature, the manager may navigate to the Customer webpage of the manager website and select to edit this test account. The manager may add points, which would result in the recalculated, greater balance being shown on both the manager and customer's side. To test the manual steps of the user, when the user creates a reservation an "Apply Reward" button is available where if the user has enough points they can exchange those for a VIP parking spot or be informed how many points they are missing. The points can be shown to be updated for creating a new reservation and decremented for deleting a reservation in the database and also in the user's profile.

UC #12 – Payment

Customer fees should be calculated for the duration of their stay. Payments for reservations are made calculated to the reserved time. Payments for walk-ins are calculated after the customer has left the garage. Additional charges are incurred for staying past a reservation time. Customers are notified of these charges. The garage manager should be able to adjust the pricing scheme of the garage, specifically the base

fee and multipliers which are reflective of the dynamic pricing model utilized in this system.

Test Coverage: To test whether payments are being deducted correctly, a test account must first be created. On the managerial website, we must set price multipliers for each hour. Then, multiple reservations during various time periods can be made to test that the dynamic pricing algorithm is working using the multipliers set. Reservations in hours with different multipliers will notify users of charge values with varying costs per hour. Customers will be charged immediately after making a reservation and their charge will be displayed in their account's Current Reservation screen. To test the additional fee payment, we will check the account into its reservation and wait until it is after the reserved exit time to check the account out. This should show another payment occurring after this action. For the walk-in scenario, we can simulate a walk-in appointment by checking an account into an open spot with an existing customer. We should see a payment as soon as the simulated walk-in leaves. This charge will also be reflected in the records table found in the database.

Integration Testing

Integration testing is performed through big-bang integration testing, where all modules are combined and functionality is verified after testing is completed. This is practical because our system is small and the convenience of this testing is a large benefit. Because smaller modular testing will be completed on each use case, errors within a use case will already have been debugged before integration testing.

Integration testing is split up into two main sections: testing involving data, and testing not involving data.

- Tests with data:
 - Garage Scanners/Sensors → Manager Website
 - Upon simulating a car entering and exiting a parking spot, test that the manager sees the correct number of open parking spots on the website.

- Upon a car exiting the parking garage, test that the appropriate reservation is moved from “Active Reservations” to “Records” on the manager website.
- Customer App ↔ Manager Website
 - Upon creating, updating, or deleting a customer profile or reservation, test that the manager sees this change in the appropriate webpage.
 - Upon creating reservations for multiple users for multiple dates and times, test that the reservations are displayed as charts on the manager webpage
 - Upon updating a user’s profile, reservation, or points from the manager end, test that the customer sees this change on their app.
 - Upon changing the price scheme by the managers, base price and multipliers, test that the user is charged different prices for the same time block depending on the changed pricing values.
- *Data Validation*: Data is stored in a MySQL database. This can be checked during integration testing. We can also validate data in the form of XML files. Thus, whenever a change involving data is made, we check whether the XML files are generated correctly, if they hold the correct data, and if they are being correctly transferred between modules.
- Tests without data:
 - *Cross-browser compatibility*: Though the customer app will be done through Android only, the manager website is viewable on different browsers. We will test that the website remains consistent and viewable on popular browsers such as Google Chrome, Safari, Firefox, and Internet Explorer.
 - *Verifying interface links*: In both the customer app and the manager website, interface links will be tested for all tabs and buttons that require redirection to a new tab. For example, clicking on the statistics tab on the homepage of the manager website should open the statistics page.

Additional Testing

Manager Website:

Instructions for running automated tests are detailed in the repository readme files, and an example is displayed below. The manager test file was written using Mocha, which is a JavaScript test framework that runs on Node.js, and Chai, an assertion library. We test that data is properly transferred and changed between the front-end and back-end, and that the pages are properly rendered by checking that the result returns an HTTP status of 200, which is the status code for success. This is done by testing every route that is used on the project. Routing is using Express.js “app” objects methods that correspond to HTTP methods in order to complete requests.

There is a total of 20 tests, split into three sets, that correspond to different routes. The first set of tests is to initialize the website by checking that the routes to the webpages, so that the Login, Dashboard, Statistics, Reservations, and Customers webpages are properly rendered. The second set of tests is for the Customer Account, which tests routes specific to the Customers page. A customer account is added, updated, then deleted to test these routes. The third set of tests is for Reservations, which tests routes specific to the Reservations page. Because a reservation can only be made using the barcode of an existing customer, a temporary customer is first entered into the database. A reservation is added, updated, then deleted to test the reservation routes, and then the temporary customer is deleted from the database. The last set of tests is for the Dashboard, which tests routes specific to the Dashboard page. Most of these routes are for retrieving data. The route to update the point scheme is tested by saving the original point scheme, testing an update, then reverting to the original point scheme. There is no set of tests specifically for the Statistics page, because all database interaction for statistics is tested upon initialization, since all data is received and rendered upon initialization and not in separate routes.

After running ‘npm test’ in the terminal as detailed in the instructions, the results for each test will be displayed in the terminal, indicating whether each test passed or failed. A failing test will explain the error that occurred. Each test also shows how much time was spent on each test, with a total time displayed at the end. All 20 tests will take an average of 3-4 seconds total to run.

```

$ npm test

Initializing...
  ✓ check LOGIN (43ms)
  ✓ check DASHBOARD
  ✓ check STATISTICS and send DB information (780ms)
  ✓ check RESERVATIONS (142ms)
  ✓ check CUSTOMERS (117ms)

Customer Account
  ✓ ADD a customer account (258ms)
  ✓ UPDATE a customer with an existing account (103ms)
  ✓ GET an existing customer (92ms)
(node:1528) [DEP0066] DeprecationWarning: OutgoingMessage.prototype._headers is deprecated
  ✓ DELETE a customer given the barcode (217ms)

Reservations
  ✓ ADD a reservation for an existing account (360ms)
  ✓ GET a reservation given the reservation id (89ms)
  ✓ UPDATE a reservation given the reservation id (99ms)
  ✓ DELETE a reservation given the reservation id (195ms)

Dashboard
  ✓ GET number of open parking spots (107ms)
  ✓ GET number of reservations by date (90ms)
  ✓ GET yesterday's revenue (89ms)
  ✓ GET number of customer accounts (99ms)
  ✓ GET price scheme (137ms)
  ✓ GET points scheme (87ms)
  ✓ UPDATE points scheme (117ms)

20 passing (4s)

```

Figure 38. Output of Automated Website Testing.

Garage Scanner:

To automatically test the Scanner application's reservation system which includes pricing and point calculations one simply has to type "unitTest" into the barcode field within the Scanner app's GUI. The test will display the current point total for a customer then proceed to simulate a variety of parking garage actions. On-time reservations are simulated and variations in point totals are displayed. Then a late-exit reservation is simulated displaying changes in point totals and price.

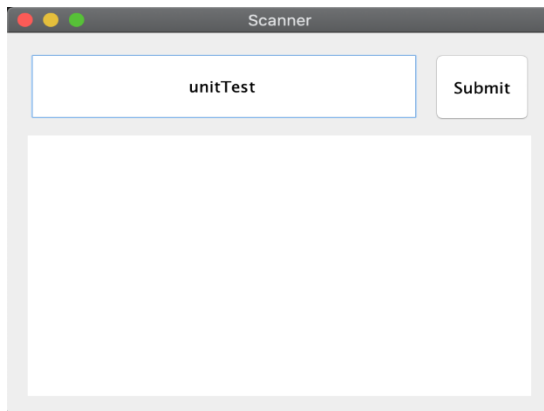


Figure 39. Scanner Test Input.

```

Testing Points and Pricing System.
-----

Parth Patel currently has 50 points.

Points Table:
OnTime: +5
Overstay: -5

-----

Entering Reservation

Hi Parth Patel.

Your Reservation Information
Date: 2020-04-30
Start Time: 20:15:54
End Time: 22:15:54
Charge: $0.00
Reservation ID: 265359054
Assigned Spot: 11

-----

Exiting Reservation On Time

Hi Parth Patel.

Your Reservation Information
Date: 2020-04-30
Start Time: 20:15:54
End Time: 22:15:54
Reservation ID: 265359054
Assigned Spot: 11
Time Elapsed: 01:00:14
Total Cost: $0.00

Thank you for choosing Zippypark!

-----

Displaying New Point Total

Parth Patel currently has 55 points.

Points Table:
OnTime: +5
Overstay: -5

```

Figure 40. Output of Automated Scanner Testing.

Algorithms:

Parking Spot Selection

A user should automatically be assigned a parking space by the app/website upon being scanned when entering the garage based on their parking preference. The system should communicate with the database to find a free spot in either the regular, handicap, or VIP parking sections.

Test Coverage: The functionality of this algorithm will be tested by requesting spots in all three sections while they are empty. Then, parking spots will be requested when all sections are all partially filled. Finally, parking spots will be requested when all sections are filled which should result in an error.

Dynamic Pricing

Parking spot prices should automatically change based on the time of day according to the dynamic pricing model. The model should be directly affected by changes made in the base price and multipliers set by the garage manager.

Test Coverage: The dynamic pricing scheme is set by the garage manager, so we test this by manually editing the price scheme from the Dashboard webpage or by clicking “Suggest Prices.” Ensuring that the correct data is coming from the database and that the algorithm for suggesting multipliers is correct is done through “Automated Testing” listed below. After updating the price scheme, the algorithm to calculate payment based on different base and multiplier rates is tested by making reservations during rush hours, noon, and night-time. The parking quotes given by the app will be monitored to confirm that they match the pricing equation. In-app parking prices will also be monitored to see if they change with changes in base price and multipliers. Changes to the pricing table will also be made and new reservations/walk-ins will be made. The prices of these actions will be monitored to see if they reflect the changes in the pricing table.

Point Rewards

A user should gain points for preferred behavior and lose points for infractions. The system and database should communicate to record changes in a user's points.

Test Coverage: The functionality of this algorithm will be tested by creating a test customer account and executing positive and negative actions, as well as reward exchanges. The customer account will be monitored to ensure that the point balance reflects the action performed. Further testing is conducted by modifying the points table in the database to change the values associated with positive and negative actions. Customer accounts will be monitored again to ensure new actions result in point changes associated with the changes made in the points table.

Non-Functional Requirements:

REQ #26– Accounting for Disability

The system should recognize if the account making a reservation has opted for disability features. During registration, customers have the option to request additional accommodations for disability and then verify their status using their unique ID code. For these users, the system will automatically reserve parking spots designated for disabled persons.

Test Coverage: We will be testing this by creating a reservation with two different types of accounts. The first type will be a normal account without the extra disability features. The second type of account will be an account that has requested for disability accommodations. During these tests, we will be checking to ensure that accounts with disability features are provided with handicapped parking spots. However, only these accounts should be given handicapped parking spots for their reservations. For all other accounts, the reserved parking spot should not be handicapped.

REQ #27 – Storing into the Database

When the system interacts with the database, it should read from and write to the appropriate tables. To stay organized, the database is divided into seven tables ranging from account information to garage records. When a new account is created, a new reservation is made, or an existing reservation is edited or cancelled, the system should update the correct database table to accurately reflect the changes.

Test Coverage: We will observe the mobile app and website's interactions with the database during any actions the customers can take. These tests include creating and editing account information, making or cancelling reservations, and arriving or leaving a parking spot. We will test that all of the actions result in an entry in the appropriate table of the database, and that a copy is also logged in the records section of the database.

REQ #30 – Accepting Barcodes

When a customer arrives, they will have to provide a barcode to verify their identity. This code is available on the profile page of the app and is linked to their account information.

Test Coverage: We will create a set of barcodes and connect them to profiles at a one-to-one ratio. When these codes are scanned into the system, they should be recognized and the system should display the respective account's pending reservations. However, we will also test sets of codes that are not linked to the system. In these tests, the system should respond by displaying that the code is unrecognized and instruct the customer to either try again or exit. Since we do not have access to a camera, the unique customer ID is entered into a text field of the scanner GUI to simulate the hardware.

User Interface Requirements

REQ #47 – Viewing Registered Customers (Manager)

When a manager logs into the system using an authorized account, they have access to a list of registered customers. This should pull all of the entries from the Customer Info table of the database and display them in an organized format.

Test Coverage: To test this, we will manually enter ten example customers into the database. Then, logged in as a manager, we will press the button to pull up registered customers. The app should access the database and output the list onscreen. We will repeat this multiple times with different customer information to represent unusual scenarios, such as customers with the same name. In each test, the database should pull and display the list of customers for the manager. For similar entries, the accounts will be differentiated by their unique user IDs.

REQ #49 – Edit Loyalty Program (Manager)

When a manager is logged in with an authorized account, they have access to the settings for the loyalty program. On this page, the manager can modify the point value associated with each action and the number of points required to exchange for an award. The app refreshes to reflect these changes so that customers do not continue to use outdated information.

Test Coverage: To test this requirement, a customer account will first take actions that result in the addition or subtraction of points, and the point balance should reflect the correct new value. Next, the manager account will edit the loyalty program point values and save the changes. When the same actions are taken, the point balance should reflect these new values. A similar test is done to ensure the cost of a reward may be altered. The customer account exchanges points for a reward before and after the manager changes the cost. The point balance should change by a different number of points each time.

Section 13: HISTORY OF WORK, CURRENT STATUS, & FUTURE WORK

13a) History of Work

General Setup

After the first Software Engineering lecture, our roster was finalized and sent to Professor Marsic and Kartik Rattan. We agreed to appoint Samantha Moy as the project leader and Atmika Ponnusamy as the configuration manager. Mondays at 10:00am were reserved for full team meetings, scheduled as necessary. Sub-group meeting schedules were decided among members of each group. On Sundays before full-team meetings, Samantha M. sent a meeting agenda to the team. She recorded meeting notes on the agenda while leading the meeting. Members gave updates on progress, discussed future work, and assigned responsibilities. Our main form of communication was via Facebook Messenger. Samantha M. sent full-team reminders about meetings and deadlines through Messenger. We also used Slack to simplify communication as separate work on sub-projects began.

Throughout the semester, our team aimed to collaborate efficiently by utilizing file sharing resources that were available to us. These resources, which included Google Drive and GitHub, enabled us to operate as one unit, while still allowing each member to make their own individual contributions to the project materials.

A Google Drive folder of resources was updated throughout the semester. The folder included contact information, calendars with deadlines, project resources, meeting agendas, and meeting minutes. This folder served as the primary tool for project management and was overseen by Samantha M.

Team members who possessed expertise regarding GitHub served as configuration managers. Each sub-group had a configuration manager for their sub-project, and Atmika served as the overall configuration manager. These individuals were responsible for ensuring that all code submitted to the project's central repository was cohesive and well-written, both syntactically and semantically. Atmika oversaw the merging of all group work. See the section, "Key Accomplishments," for specific information about the division of work.

Merging Contributions

Our team took several steps to ensure seamless report integration at the end of each week. After a report section was submitted, Samantha M. immediately began preparing for the next

submission. For example, for Report 2 – part 2, she added a document on the team Google Drive called “Report #2 – part 2 (draft)” to the folder labeled “Report #2.” There was a folder for Report 1, 2, and 3. Every folder held relevant writing pieces and report submissions. Samantha M. researched the upcoming report section and, in the draft document, outlined the requirements and possible ideas to consider. She divided the section into evenly-sized parts, and each teammate was asked to sign up for one part. If the week’s report section required diagrams, Atmika shared a template with the team so that everyone’s diagram style was the same. Team members finished writing their pieces on the document by 11:59pm of the Thursday before the deadline. Samantha M. spent Friday and Saturday compiling everyone’s work. She checked that the content was correct, while also ensuring consistency and uniformity in the terminology and formatting.

Project Milestones

Initially, our team was apprehensive given our overall lack of software engineering experience. Therefore, our team leaders met with professors, advisors, and the Rutgers Engineer Computing Services (ECS) to gain insight into a project of this magnitude within days of being assigned the project. The team then worked hard to use this advice to create a clear business plan, basic software infrastructure, and knowledge of the technologies required very early in the semester.

Consequently, many of our milestones were not reach until later in the semester, but this was a negligible consequence of creating a strong foundation and preparing all teammates with the necessary knowledge. See the following table for key project milestones. Report and demo documentation are not included in this table; they can be found in the Gantt chart and timeline.

Date Milestone Reached	Milestone
February 27, 2020	Created project server. Implemented shared database.
March 14, 2020	Established external connections to database.
March 20, 2020	Implemented UC 7, 8, 9.
March 22, 2020	Implemented UC 1, 2, 3, 4, 5, 6.
March 31, 2020	Finished minimum viable product.

April 23, 2020	Implemented UC 10, 11, 12.
April 28, 2020	Finished product.

Table 29. Project Milestones.

In the foundation and planning stages, full-team meetings were held weekly. As we transitioned into sub-team work, these regular meetings were no longer necessary. Sub-teams could regulate work as they pleased as long as the above milestones were reached by the corresponding date. However, we still met occasionally on pre-scheduled days for full-team check-ins, such as in preparation for demos. We also were in contact weekly via Slack when assembling the report portion that was due each week.

Located in our shared group resources folder, we have a Gantt Chart that shows the high-level overview of past work on this project. This spreadsheet is maintained by Samantha M.

TEAM 2 OVERALL TIMELINE

PROJECT TITLE	ZippyPark
PROJECT MANAGER	Samantha Moy

PHASE	DETAILS		
		JAN	
	PROJECT WEEK: Sunday of each week -->	26	2
1	Project Definition and Planning	- Project Outlining	Project Outlining
		- Infrastructure Setup	
2	Project Building	- Customer Interface (app)	
		- Scanner/Sensor System	
		- Managerial Website	
		- Overall Integration	
3	Course Deliverables	- Project Proposal	Project Proposal
		- Report 1	
		- Report 2	
		- Demo 1	
		- Report 3	
		- Demo 2	
		- Electronic Project Archive	

Figure 41. Preview of Gantt Char (part 1).

Overall phases of work are shown in this chart. See Calendar of deadlines for specific assignments.

Team 2
Last updated - 4/9/20

Iteration #1						Iteration #2							
FEB			MAR					APR				MAY	
9	16	23	1	8	15	22	29	5	12	19	26	3	
Infrastructure Setup													
		Customer Interface (app)											
		Scanner/sensor system											
		Managerial Website											
			Integration 1					Integration 2					
Report 1													
		Report 2											
						Demo 1							
								Report 3					
											Demo2		
												Electronic Archive	

Figure 42. Preview of Gantt Chart (part 2).

The same spreadsheet may also be viewed here: <https://tinyurl.com/team2Gantt>

Additionally, we have a calendar of deadlines that is maintained by Samantha M. This spreadsheet serves as a simple way to view more specific project deadlines and milestones.

LEGEND
Course deadlines
Meetings
Writing deadlines
Suggested sub-project deadlines

Figure 43. Legend for calendar of deadlines.

FEBRUARY						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
26 Email team roster	27 Full Team Meeting	28	29	30 Finish writing your piece	31 Proposal assembled & submitted Prof office hours about sub-projects	1
2 Proposal due	3 Full Team Meeting	4	5	6 Finish writing your piece	7 Report assembled & submitted	8
9 First report - p1 due	10 Full Team Meeting Establish project infrastructure Git setup	11 Meeting with Sabian - ECS Server	12	13 Finish writing your piece	14 Report assembled & submitted	15
16 First report - p2 due	17 Full Team Meeting All groups display environment	18	19	20 Finish writing your piece	21 Report assembled & submitted	22
23 First report - full due	24 Group 1 screen setups Group 2 setup Group 3 setup	25	26	27 Finish writing your piece	28 Report assembled & submitted	29

Figure 44. Preview of February calendar of deadlines.

MARCH						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1 Second report - p1 due	2	3	4	5 Full Team Meeting - checkpoint Finish writing your piece Prep demo pieces (Group 1 & 2)	6 Report assembled & submitted	7
8 Second report - p2 due	9	10 ~MIDTERM~	11	12	13	14 Finish writing your piece External DB connection setup
15 Report assembled & submitted	16	17	18	19	20	21 All groups finish demo code
22 Second report - full due	23	24 Prep Q&A doc.	25 Full Team Meeting - demo prep	26	27 All groups finish demo docs	28
29 Subgroup demo videos.	30 Finish overall video Send demo docs to TA	31	1	2	3	4

Figure 45. Preview of March calendar of deadlines.

APRIL						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29	30	31	1	2	3	4
			WebEx Demo 1			Finish report feedback updates
5	6	7	8	9	10	11
				Finish writing your piece	Report assembled & submitted	
12	13	14	15	16	17	18
19	20	21	22	23	24	25
Third report - p1 due		Prep Q&A doc.		Finish demo code.	Finish demo docs.	Subgroup demo videos.
26	27	28	29	30	1	2
Finish overall video Send demo docs to TA		Full Team Meeting - demo prep	WebEx Demo 2	Finish writing your piece		

Figure 46. Preview of April calendar of deadlines.

MAY						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
26	27	28	29	30	1	2
					Report assembled & submitted	
3	4	5	6	7	8	9
	E-archive edits.	E-archive submission.		Electronic project archive due		
				Third report - full due		
				Reflective Essay due		
10	11	12	13	14	15	16

Figure 47. Preview of May calendar of deadlines.

The same spreadsheet may also be viewed here: <https://tinyurl.com/team2calendar>.

13b) Current Status

Together, our team agreed on the overall project business plan and shared infrastructure. Samantha M. and Atmika researched and setup the shared infrastructure to lay a strong foundation for the team's sub-projects. Our team of nine members divided into three groups of three. Each group was responsible for a single mini project with specific functional features, qualitative properties, and use cases listed below. There was inter-group collaboration if a member's specific expertise was ever required. Some members also took on additional tasks.

**Note – past work served as the inspiration for our project's ideas and policies. However, no code was taken, repurposed, or referenced from past course projects. Every part of this project was developed by this team for educational purposes.*

Key Accomplishments

All 12 use cases were implemented. The project is composed of three sub-projects that interact with a shared database on a server. Product ownership of each accomplishment is listed below.

Shared Infrastructure – Samantha Moy, Atmika Ponnusamy

- Rutgers ECS server
- MySQL database with 8 tables
 - Customer Info
 - Managers
 - Parking Spots
 - Reservations
 - Walk-Ins
 - Records
 - Payment
 - Points
- 3 sub-projects interact with the database (see the following)

Sub-Projects

1. **Customer Services (Mobile App)**

Members: Samantha Moy, Atmika Ponnusamy, Shreya Patel

Technologies Used: Java, XML, JSch, JDBC, ADB, AVD, Android SDK 28, MySQL

- UC-1: Create Account
- UC-2: Edit Account
- UC-3: Delete Account
- UC-4: Create Reservation
- UC-5: Edit Reservation
- UC-6: Cancel Reservation
- UC-11: Points Management
- UC-12: Payment

2. **Arrival & Departure Operations (Simulated Scanners/Sensors)**

Members: Andrew Ko, Parth Patel, Piotr Zakrevski

Technologies Used: Java, Swing GUIs, JSch, JDBC, MySQL

- UC-7: Enter Garage
- UC-8: Exit Garage
- UC-9: Update Spot Status
- UC-11: Points Management
- UC-12: Payment

3. **Manager Tasks (Website)**

Members: Kylie Chow, Samantha Cheng, Nandita Shenoy

Technologies Used: Node.js, jQuery, EJS/CSS, MySQL;

Frameworks: Express, Bootstrap

- a. UC-1: Create Account
- b. UC-2: Edit Account
- c. UC-3: Delete Account
- d. UC-4: Create Reservation
- e. UC-5: Edit Reservation
- f. UC-6: Cancel Reservation
- g. UC-10: Display Statistics

- h. UC-11: Points Management
- i. UC-12: Payment

Additional Tasks

1. Project Management – Samantha Moy
 - Manage group deadlines and weekly tasks (set by professor and by the group).
 - Update group calendars and documents.
 - Run team meetings and write agendas.
 - Assemble and submit weekly reports/deliverables.
 - Coordinate integration of sub-group work.
2. Configuration Management – Atmika Ponnusamy
 - Manage GitHub account.
 - Ensure the team maintains good project practices when integrating deliverables.
 - Educate team on best coding practices and project infrastructure.
 - Troubleshoot any issues with group infrastructure (server, database, GitHub).
 - Perform integration testing

13c) Future Work

The current project is a completed parking garage software product that contains all the features promised. However, with any project, there is always the possibility of future enhancements.

Below is a list of features that we excluded from the current project for specific design reasons.

Yet, we believe they may serve as good additions in future iterations.

- Family account feature
- Develop an iOS app
- Integrate a camera that scans actual barcodes and/or license plates (instead of simulating).
- Automatic point scheme tool (similar to automatic pricing tool)

Section 14: REFERENCES

- [1] Chen, C., Deng, C., Feng, X., Liao, S., Musale, S., Sakhuja., R. (2018, December 9). *Auto Park Parking Garage Automation*. Software Engineering Project: Parking Garage/Lot Automation.
<https://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2018f-g3-report3.pdf>
- [2] Choudhury, S., Ngo, T., Nguyen, D., Nguyen, K., Patel, N., Tran, L. (2019, December). *Blockchain and Docker Assisted Secure Automated Parking Garage System*. Software Engineering Project: Parking Garage/Lot Automation.
<https://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2019f-g4-report3.pdf>
- [3] Costco. (2020) *Why Become a Member*. Costco Wholesale Corporation.
<https://www.costco.com/membership-information.html>
- [4] Gallego, G., Van Ryzin, G. (1994) "Optimal dynamic pricing of inventories with stochastic demand over finite horizons." *Management science* 40.8 (1994): 999-1020.
- [5] Inrix. (2017, July 12). *Searching for Parking Costs Americans \$73 Billion a Year*. Inrix Research. <https://inrix.com/press-releases/parking-pain-us/>
- [6] Katsov, Ilya. (2019, March 5). *A Guide to Dynamic Pricing Algorithms*. Grid Dynamics.
<https://blog.griddynamics.com/dynamic-pricing-algorithms/>
- [7] Marsic, I. (2012). *Software Engineering* (pp. 246-270). Rutgers University
- [8] Shvets, A. (2018). *Dive Into Design Patterns*. Source Making.
<https://sourcemaking.com/design-patterns-ebook>
- [9] Statcounter. (2020, March). *Mobile Operating System market Share Worldwide*. GlobalStats. <https://gs.statcounter.com/os-market-share/mobile/worldwide>
- [10] Tian, Q., et al. "Dynamic pricing for reservation-based parking system: A revenue management method." *Transport Policy* 71 (2018): 36-44.
- [11] Uber Help. (2020). *How does Uber work?*. Uber Technologies, Inc.
<https://help.uber.com/riders/article/how-does-uber-work?nodeId=738d1ff7-5fe0-4383-b34c-4a2480efd71e>