



NOVEMBER 24, 2023

REPORT 3

PARKEZ AUTOMATED PARKING GARAGE

GROUP 4



Report 3

CSCI 441 Software Engineering

Group # 4:

ParKEZ Automated Parking Garage

Project URL: <https://fhsu-park-ez.vercel.app/>

11/24/2023

Team Members:

Benjamin Bylsma

Adrian Elgin

Mikael Mikaelian

Phongsavanh Mongkhonvilay

Geoffrey Sarpong

Christopher Smith (Team Leader)

Table of Contents

Table of Contents	2
Summary of Changes	3
Contribution Breakdown	5
Section 1: Customer Problem Statement	6
Section 2: Glossary of Terms	11
Section 3: System Requirements	12
Enumerated Functional Requirements	12
Enumerated Nonfunctional Requirements	13
User Interface Requirements	19
Section 4: Functional Requirement Specification	22
Section 5: Effort Estimation Using Use Case Points	37
Section 6: Domain Analysis	40
Section 7: Interaction Diagrams	44
Section 8: Class Diagram and Interface Specification	57
Data Types and Operation Signatures	58
Design Patterns:	65
Object Constraint Language (OCL) Contracts:	65
Section 9: System Architecture	68
Architectural Styles	68
Identifying Subsystems	69
Mapping Subsystems To Hardware	69
Persistent Data Storage	70
Network Protocol	71
Global Control Flow	71
Hardware Requirements	72
Section 10: Algorithms and Data Structures	73
Algorithms:	73
Data Structures:	73
Section 11: User Interface Design and Implementation	74
Section 12: Design of Tests	80
Use Case 1- Sign-in:	80
Use Case 5- Walk-in:	80
Use Case 6 - Reservation Editing/Cancellation:	80
Use Case 07 - Payment:	83
Use Case 16 - Reservation:	83
Use Case 22 - Elevator Interaction:	84
Section 13: History of Work, Current Status and Future Work	87
History of Work:	87

Key Accomplishments:	87
Current Status:	87
Future Work:	87
Product Ownership:	89
Section 14: References	91

Summary of Changes

- Added Domain Concept / Class Integration Traceability Matrix.
- Added Domain Concept Responsibilities.
- Added Object Constraint Language Contracts.
- Updated Registration Image.

Contribution Breakdown

Task	Bylsma	Elgin	Mikaelian	Mongkhonvilay	Sarpong	Smith
Coversheet/Contribution Breakdown/Table of Contents	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Summary of Changes	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%
Customer Problem Statement	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Glossary of Terms	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
System Requirements	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Functional Requirements Specification	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Effort Estimation Using Use Case Points	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Domain Analysis	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Interaction Diagrams	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Class Diagram and Interface Specification	0.00%	50.00%	0.00%	0.00%	0.00%	50.00%
System Architecture and System Design	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Algorithms and Data Structures	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
User Interface Design and Specification	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Design of Tests	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
History of Work, Current Status, Future Work	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
References	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%

Section 1: Customer Problem Statement

Parking Garages are a vital part of large cities, however ever since their creation there has been little done to improve the design of parking garages. Thus, we would like to request a system that would allow us to provide a better parking garage experience for our customers. Allowing them an innovative and stress-free way to use our parking garage.

Another of the largest issues we have noticed with how parking garages currently operate is how long it takes to enter, exist, and pay. Seeing as our new parking garage has an elevator to the upper floors, we would like to request some method of reducing the amount of time it takes to perform these tasks, for clarification we would like our customers to not have to worry about finding a parking spot or having to spend much time on the entering and exiting processes. Personal experience let us know that sometimes entering and exiting a parking garage can take upwards of twenty minutes, something we would like to avoid with the help of a new integrated system. Some basic ideas that we have thought up to help with that goal include a reservation system allowing the customer to book in advance. This would keep the amount of time customers would have to spend entering or exiting the garage down while also alleviating fears of not getting a parking spot. Our automated parking garage system would track how much time each vehicle stays parked in the garage and send automatic updates to the customer when their reserved time is approaching its end, as well as when the reservation time has expired. By reminding customers in this manner, it would give them the opportunity to extend their reservation time through the ParkeZ site

remotely before their reservation time expires. Making reservations through the ParkEZ website requires customers to pay beforehand, which in turn allows them to leave immediately without having to pay manually at a slow ticket booth and reduces overall parking garage congestion.

Another issue we have noticed is that many parking garages lack visibility and are not easily navigable. Many parking garage customers, as we can personally attest, spend quite some time driving around the parking garage looking for a parking spot, leading to congestion during high traffic hours. This congestion and lack of visibility might encourage frustrated parking garage customers to begin using a different parking garage. While we have a vehicle elevator to help alleviate part of the congestion issue, customers still do not know the layout or the happenings of the garage until they get into the garage. One idea we had would be to have a map of the parking garage on the website, while our parking garage has a rather simple layout, we would still like our customers to have the ability to see a map showing the location of the exit, and the locations of the various parking spaces. Similarly, we also think it would be nice if our customers had a way to get real-time knowledge of parking space availability and book specific parking spots. This would allow customers who are disabled, elderly, or have luggage to reserve parking spots closer to the elevators than if they were to just drive around looking for a parking spot.

While lightly touched upon in the previous paragraphs, our parking garage has several interesting features and pieces of technology that other parking garages do not possess. Our hope is that with the help of the new system in conjunction with the new technology we will be able to provide an unmatched parking garage experience for our

customers. While we will not be going over the entire suite of new features we will go over some of the more important ones. The most obvious feature would be our vehicle elevator. Our hope is that with the vehicle elevator we will be able to limit the congestion that comes with finding a parking spot by having the floors be separate areas inaccessible without first going through the elevator. The second feature we have in our parking garage is license plate reading cameras, we have them set up throughout the garage. We would like the cameras in the vehicle elevator to automatically scan and register if the vehicle in question has a reservation, if they had a reservation the vehicle elevator would take the customer to the correct floor with no input needed, cutting down on user input time and the amount of time customers behind them in line might have to wait. Another feature in our Parking garage would be the console inside the vehicle elevator, the console is there in the event that a customer is driving a different vehicle than the one they have registered, or the camera does not read the license plate correctly. The user interface console in the vehicle elevator could also allow a registered individual to make a reservation immediately, if any spaces are available, in case they forgot to reserve a space online. This would take longer and possibly make lines longer, but we think this would be a great way to still offer our services to registered customers who perhaps forgot to reserve a parking spot. Tying in with previous technology, we have set up a camera at the exit ramps to read the license plates of departing vehicles. The exit cameras, in conjunction with parking space sensors, would allow the automated system to know which vehicle has left, charge the owner's account accordingly and update the available parking space map. Each parking space will have an occupancy sensor that will toggle on when a vehicle is

present and off when empty allowing the system to know whether or not a space is occupied.

We have briefly touched upon reservations in the previous paragraphs, and while we are not developers, we still have some ideas on how they might work. Reservations should be tied to a user account and be very easy for registered customers to make. They should request the amount of time the customer plans to have their vehicle parked along with the accompanying rate for that duration. If the customer goes over their planned time, they would be charged an overtime fee that would be calculated by their vehicle departure time, which the automated system will keep track of using the parking space sensors and exit camera license plate recognition system. If the customer leaves early the system can be configured to bill them for the full reserved time or discount their bill, and their parking spot will be entered back into the system to be used by another customer. Another reservation scenario our system could be made to deal with is when a customer arrives late or does not arrive within the reservation window. We know from experience that running late is something that can happen quite often. Thus, if a customer is running late the system can be made to hold a reserved parking space for a grace period of 30 minutes before adding their parking space back to the pool of available parking spaces. If the customer arrives after the 30-minute grace period or outside of the reserved time entirely they will simply be treated as a registered customer who has not made a reservation and they will have the option to park on the first floor of the garage as a walk-in or reserve a spot on other floors from the garage user interface. Either case is dependent on whether or not there are spaces available.

The aim of the ParkEZ automated parking garage system is to change every parking garage experience into a good one for both garage customers and garage owners. If customers are happy, they will come back. If parking space availability is maximized, garage owners will also be happy.

Section 2: Glossary of Terms

Cancellation: The act of a customer voiding their reservation to receive reimbursement (to be completed before half of the reservation time is passed).

Guaranteed Reservation: A monthly contract with the parking garage to reserve a parking space(s) for specified days and times.

No-Show: The act of a customer failing to check into the garage and filling their reservation (payment will still be collected for a no-show).

Overstay: The act of a customer failing to exit their reservation past their reservation's time ending (will receive an overstay charge).

Overstay Charge: A penalty fee added to the customer's bill for failing to exit their reservation before their reservation's time ends.

Park-Ez Customer: A user of the application who pays to reserve single use parking spots and fills said spot during the reservation time.

Reservations: Single use parking spots that have been paid for and assigned to a user of the application.

Walk-in Customer: An individual who enters the garage without a reservation and is directed to the first floor of the garage.

Understay: A parking space vacated early by a customer before their reserved time was completed.

Database: Hosted through MongoDB, used to store customer data and parking information

Section 3: System Requirements

Enumerated Functional Requirements

REQ-X	Requirement Description	Priority
REQ-1	Real-time parking space availability.	High
REQ-2	Automated payment processing.	High
REQ-3	Automated entry and exit for vehicles	High
REQ-4	Ground floor display for vacancies	High
REQ-5	Reservation system with confirmation	High
REQ-6	Integration with security and surveillance.	Medium
REQ-7	User-friendly interface.	High
REQ-8	Reporting and analytics for administrators.	Medium
REQ-9	Elevator displays	Medium
REQ-10	The system-to be shall determine whether the customer has the necessary authorization to park at ParkeZ via license plate.	Medium
REQ-11	The system-to be shall display space occupied or available (including disability parking space, etc.).	Low
REQ-12	The system-to be shall guide the customer what area to park using map display functions.	High
REQ-13	The system-to be shall keep track of customer's time from entrance to exit, and lot number.	Medium

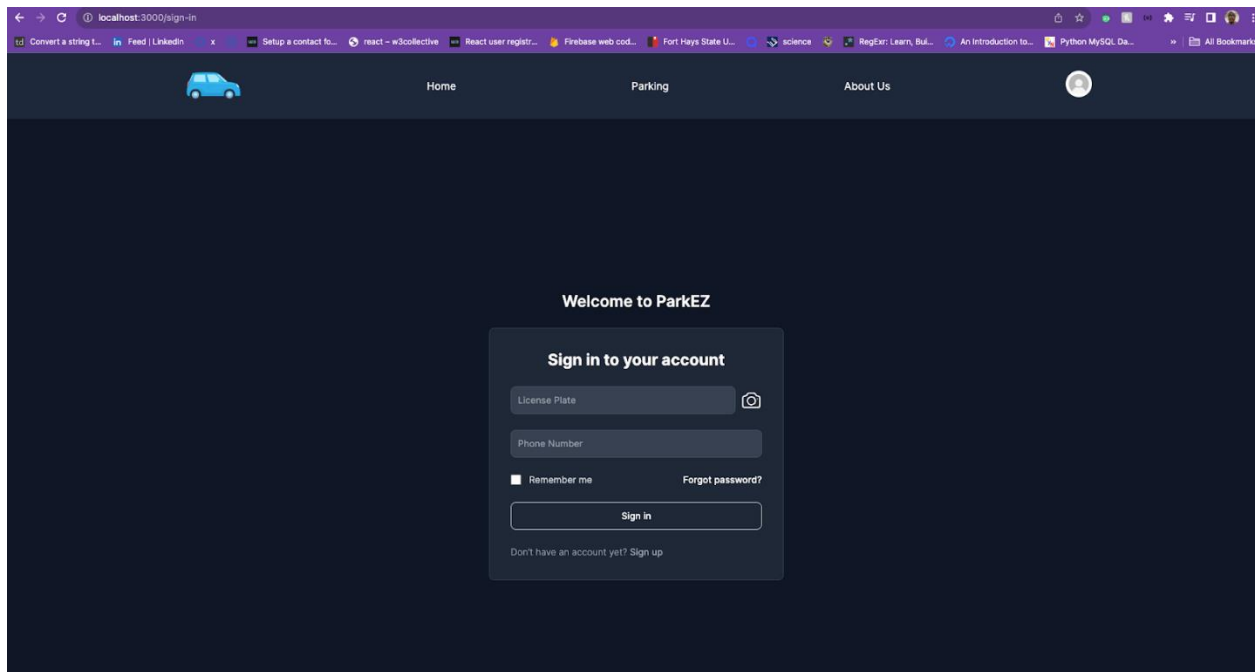
REQ-X	Requirement Description	Priority
REQ-14	The system-to-be shall update changes of the parking space occupied or available to inform other users.	Low
REQ-15	The system-to be shall alert the user upon leaving, maxing out of time allotted, time remaining, and any additional payment	Medium
REQ-16	The system-to be shall allow the user to make the necessary changes on the system including updating parking information, payment, cancel parking, etc.	Medium

Enumerated Nonfunctional Requirements

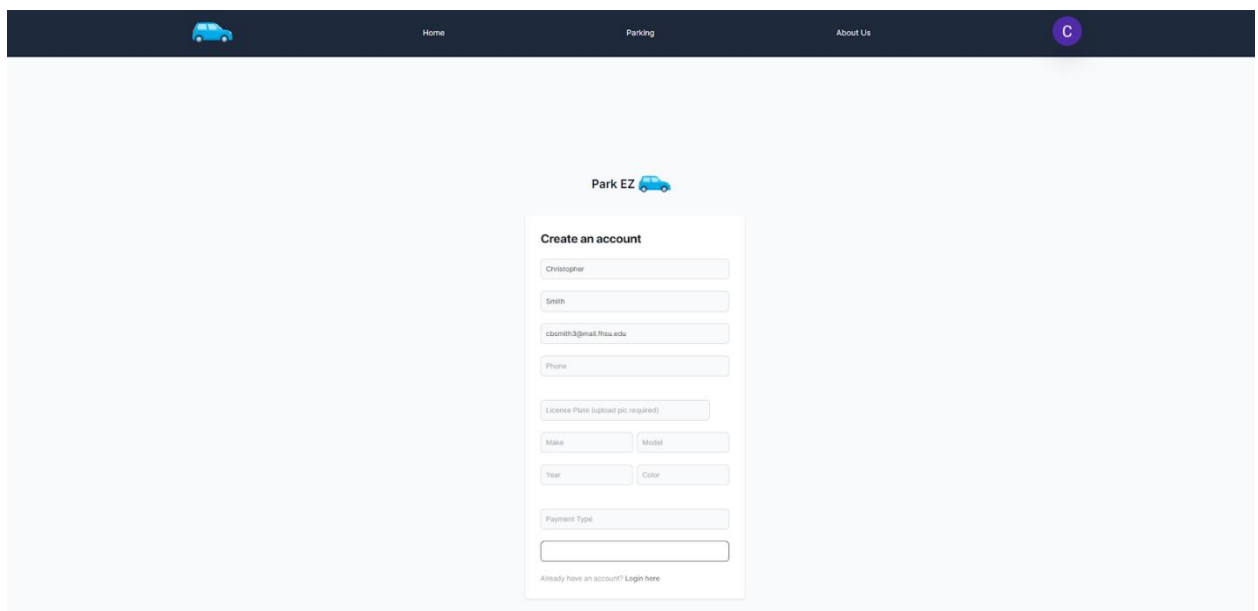
1. **Performance:** The system must provide real-time updates with minimal latency (High).
2. **Reliability:** The system should have 99.9% uptime (High).
3. **Security:** Data encryption and secure user authentication (High).
4. **Scalability:** The system must handle increased demand during peak hours (Medium).
5. **Usability:** The user interface should be intuitive and accessible (High).

UI-X	Description	Status
UI-1	Parking Space Locator/Map	High
UI-2	Payment Interface	High
UI-3	Registration/Login	High
UI-4	Store User Data for analytics	High
UI-5	Notifications	Medium

Login: /sign-in



Registration: **/sign-up**



Parking: **/parking-space**

Home

Parking

About

Menu

Available parking spaces

5 feet away

Level A - Space 12

Go

20 feet away

Level A - Space 22

Go

30 feet away

Level B - Space 11

Go

40 feet away

Level C - Space 13

Go

Parking Space Map

Level A

Level B

Level C

TextEdit

Payment

Payment Information

Card Number

0000 0000 0000 0000

Expiration Date

MM / YY

CVV

000

Card Holder

Full Name

Submit

Payment Confirm: /parking-space

Home

Parking

About

Menu

Available parking spaces

5 feet away

Level A - Space 12

Go

20 feet away

Level A - Space 22

Go

30 feet away

Level B - Space 11

Go

40 feet away

Level C - Space 13

Go

Parking Space Map

Level A

Level B

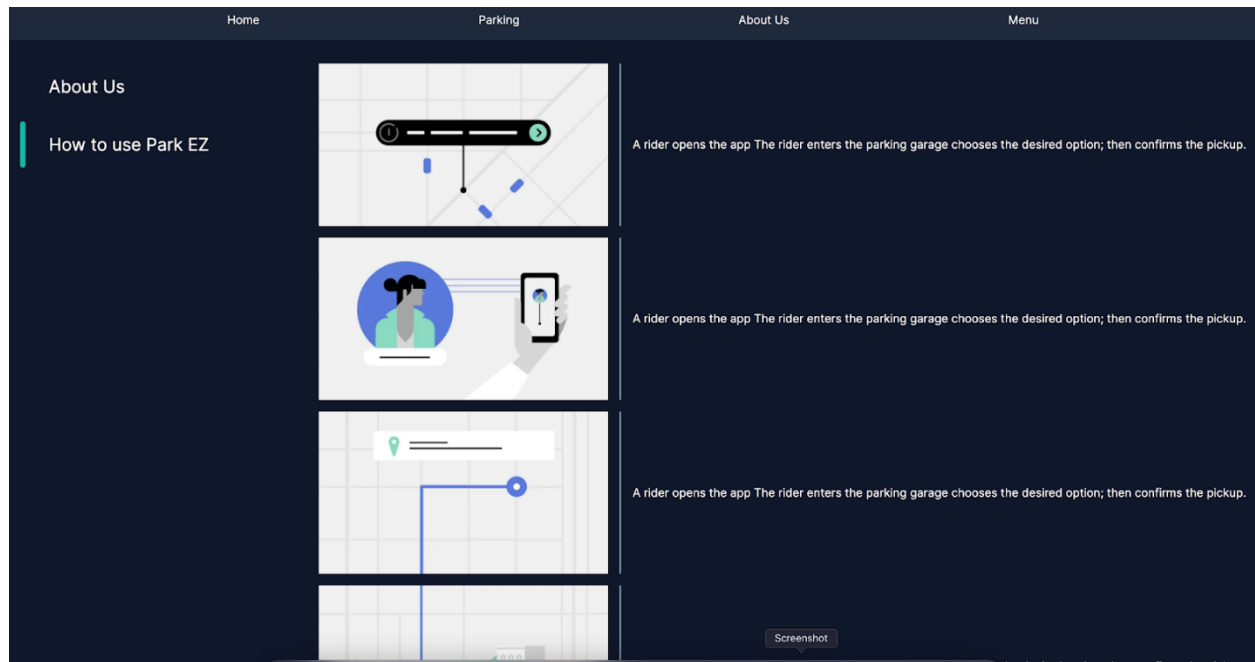
Level C

TextEdit

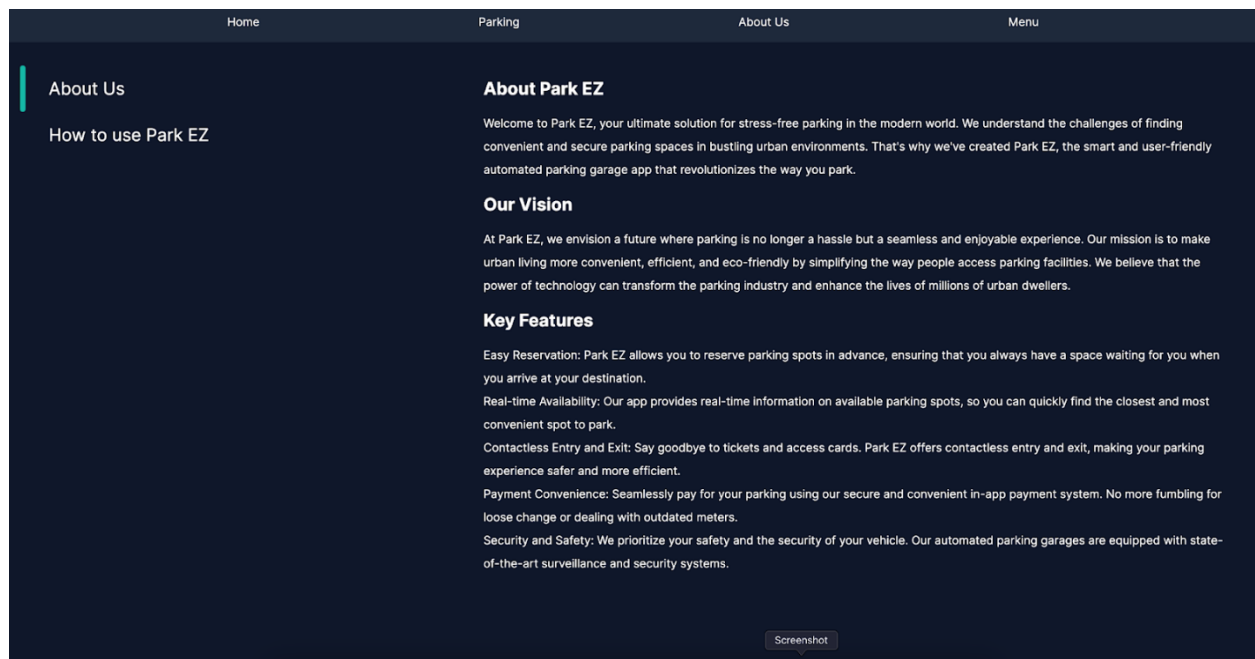
Payment

✓

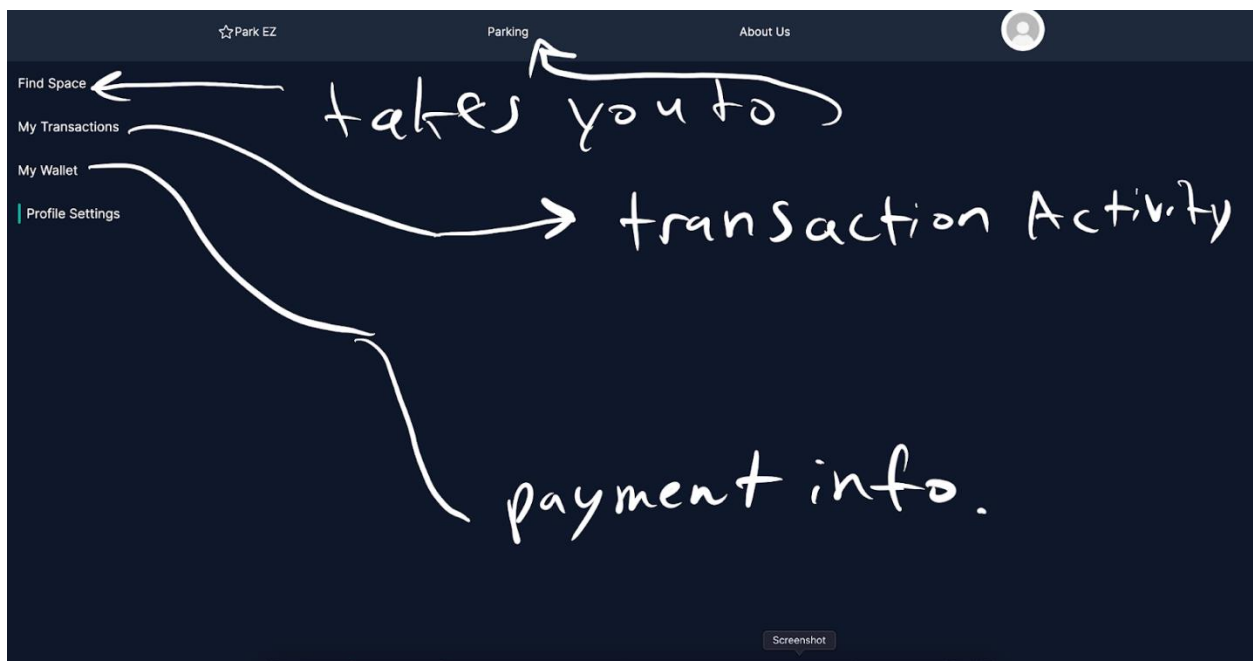
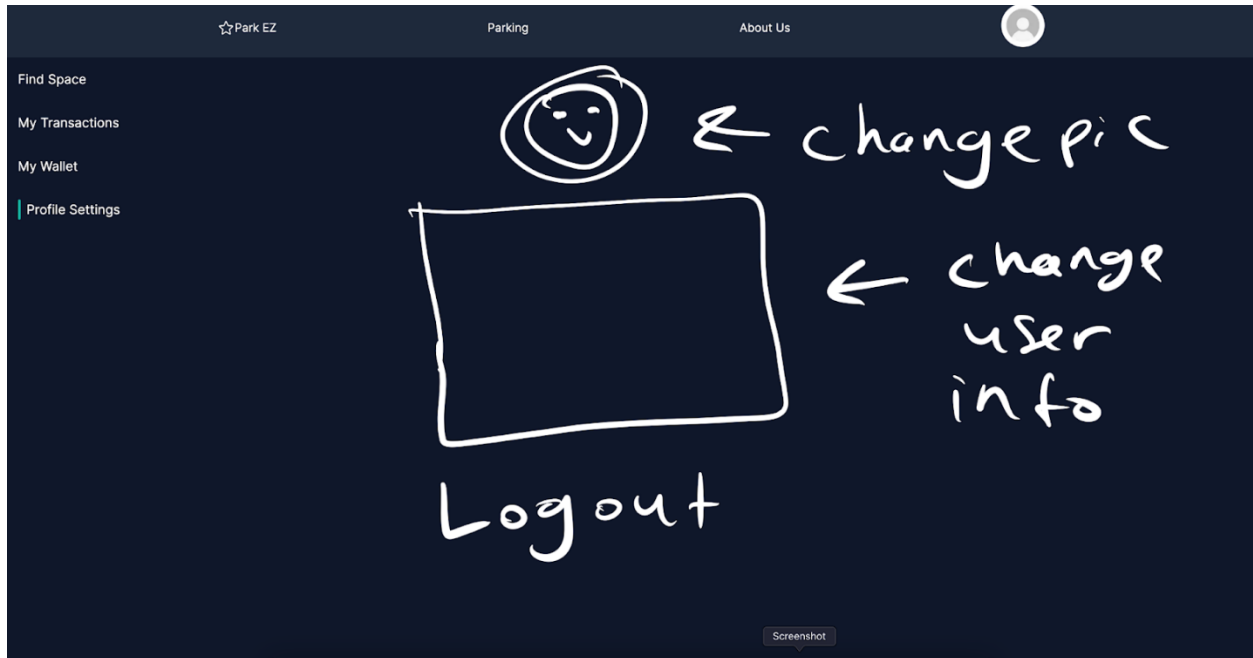
About : /about/how-to-use



/about/about-us



Menu: /user-menu



User Interface Requirements

REQ-X	Requirement Description	Priority
REQ-17	To create an account, customers require a mobile device with internet access. During registration, they must provide their name, email, mobile number, date of birth, and license plate number.	High
REQ-18	To log in to the customer's account, the customer needs to input their email or phone number and password.	High
REQ-19	The reset password feature needs to be available in the login case in case the customers forget the password	High
REQ-20	The sign-up button is shown in the login page for unregistered customers	High
REQ-21	The update and edit customer's info should be available after the customer logs in.	High
REQ-22	To top up the balance customers need to choose a preferred method on top-up account balance and make the payment.	High
REQ-23	To make an online reservation, customers need to log in to their account, choose the hours and slot to reserve and make a submission. The payment is done through account balance or directly through the credit card.	High
REQ-24	To edit or cancel the reservation, the customer needs to log in into their account, choose reservation and update or cancel it.	High
REQ-25	The system-to-be will allow users to click a button to see a map display of the garage location and features.	Medium
REQ-26	The system-to-be will allow users to click a button to access parking status (including available spots, and occupied spots).	Low
REQ-27	The system-to-be will display parking lot number, time remaining, additional fees on screen.	Medium
REQ-28	The system-to-be will display an edit button for the user to make changes to parking, payment, and other personal information.	Medium
REQ-29	The system-to-be will display a click button to allow users to continue, cancel or end parking.	High

REQ-X	Requirement Description	Priority
REQ-30	The system-to-be will display an alert notification when the customer enters or exits the parking garage via license plate readings.	High
REQ-31	The system-to-be will display parking history and fees.	Low
REQ-32	To enroll in a monthly guaranteed reservation online, customers need to log into their account, select the guaranteed reservation option, select the days of the week, select time slots and select the reserve button to make a submission. The payment is done through account balance or directly through the credit card.	Medium

Section 4: Functional Requirement Specification

a. Stakeholders

1. Customers: Primary users of the automated garage parking system, focused on quick parking, efficient entry/exit, and stress-free experience.
2. Garage Owners: Interested in maximizing parking space utilization, streamline garage operation for higher profitability.
3. System Administrators: Responsible for managing and maintaining the automated garage system.

b. Actors and Goals

Actors	Goals	Use Cases
Parking UI	Display available slots, enable online payment, reserve slots upon payment, and allow cancellations within a set time.	UC1, UC2, UC3, UC4, UC5, UC6, UC7, UC8, UC9, UC10, UC16, UC17, UC18
Parking Interface	Show open slots for walk-in customers and permit them to park upon entering the garage.	UC3, UC4, UC5, UC7, UC17
System	Update customer's information	UC18
System	Update parking information	UC8, UC9, UC10
Security System	Protect customer's info	UC13
Camera	Scan car plates	UC15
Customer	Register and log in on the website	UC1, UC2
Customer	Modify info or payment method	UC17, UC18
Customer	Check the available slots and reserve on website	UC3, UC4, UC6, UC7, UC17
Customer	Check the available slots and reserve for walk-in	UC3, UC4, UC5, UC7
Customer	Arrive at the garage	UC11, UC15
Customer	Exit and pay	UC12, UC15, UC7

Customer	Receive confirmation code / reservation details on email	UC8, UC9, UC14
Customer	Cancel/Edit the reservation on the website	UC6, UC10
System	Confirm customer on the garage elevator has a reservation, the license plate number is in the database and the customer is transported to the correct deck	UC3, UC4, UC5, UC15

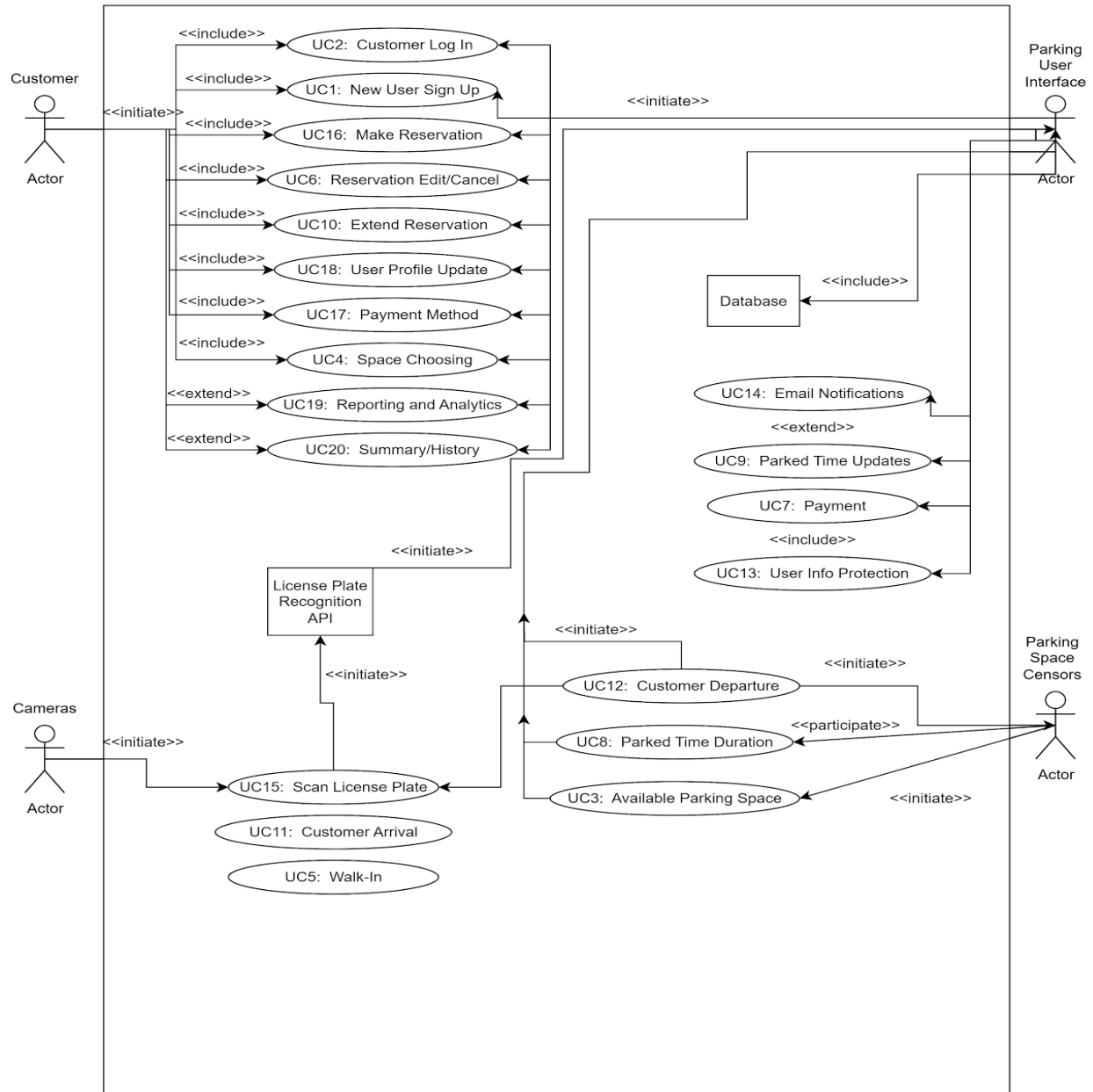
i. Casual Description

UC1	Sign Up - Customer create an account on the website
UC2	Log In - Customer use their phone number/email and password/one time code to access the website
UC3	Available Parking Space- Customer checks if free slots are available
UC4	Slot Choosing - Customer select desired slot from available ones
UC5	Walk-in - Customer with no advanced reservation
UC6	Reservation Editing/Cancelation: Customer edits or cancels reservation
UC7	Payment - Charging of the customers at the end of parking
UC8	Time duration - Keeping an account of time lasted since parking started
UC9	Time update - Keep customer updated about the time left before reservation expiry
UC10	Reservation Extension - Extend current reservation time
UC11	Arrival - customer enters the garage
UC12	Departure - customer exits the garage
UC13	Information Privacy - Encrypt customer information for security
UC14	Email/SMS Notifications - Send checks, security codes, reservation details
UC15	Scan plate - scan vehicle registration plate
UC16	Reservation - customer reserves a slot for selected time.
UC17	Payment method - Customer chooses payment method

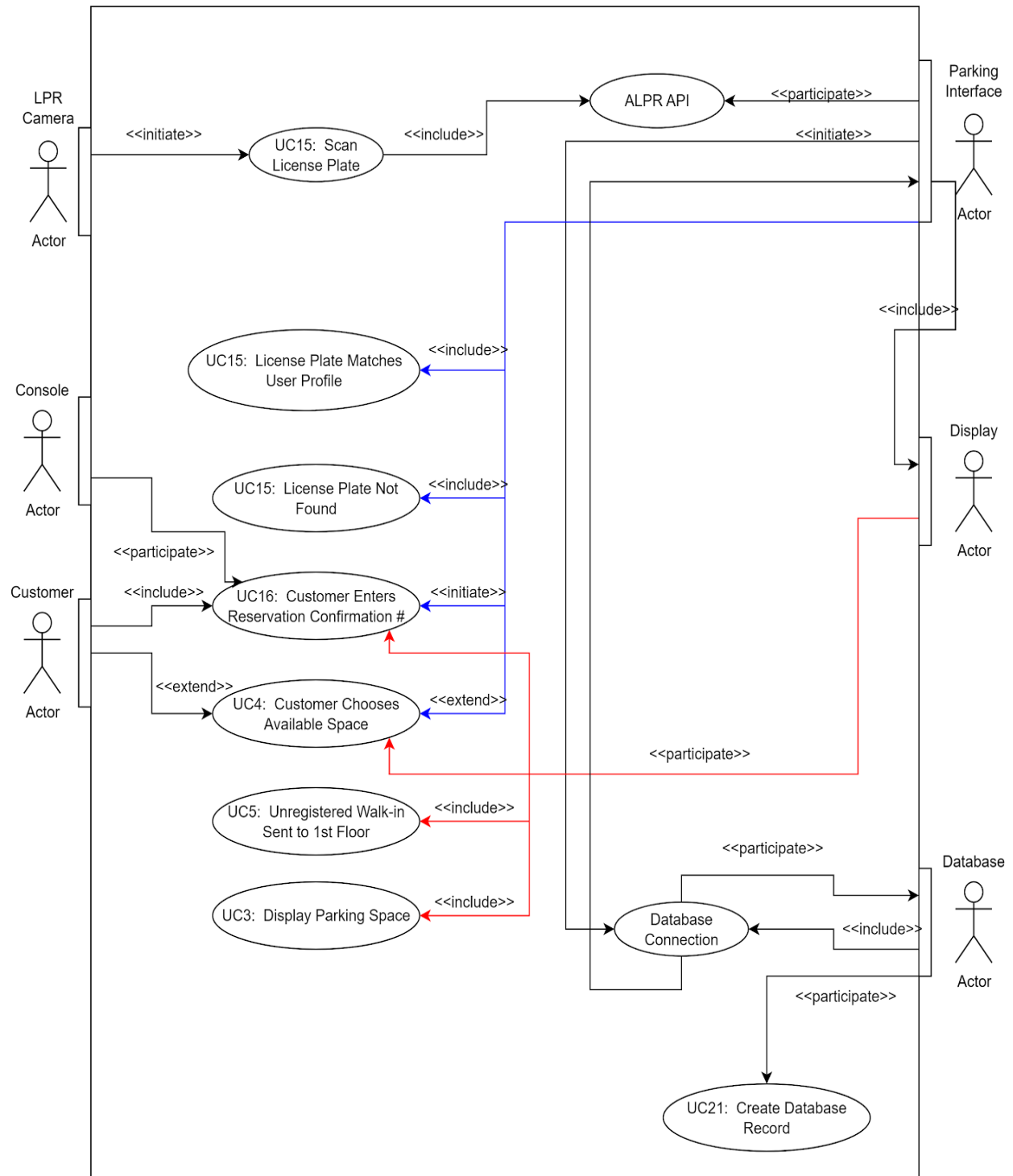
UC18	Info update - Customer updates their account information
UC19	Reporting and Analytics - Customer sends any issues and agree or disagree to administrator gathering and analyzing the customer usage
UC20	Summary/History- Customer sees their parking history and payments
UC21	Database Record Creation - User profile and parking activity saved as database records
UC22	Customer arrives in the garage elevator

ii. Use Cases

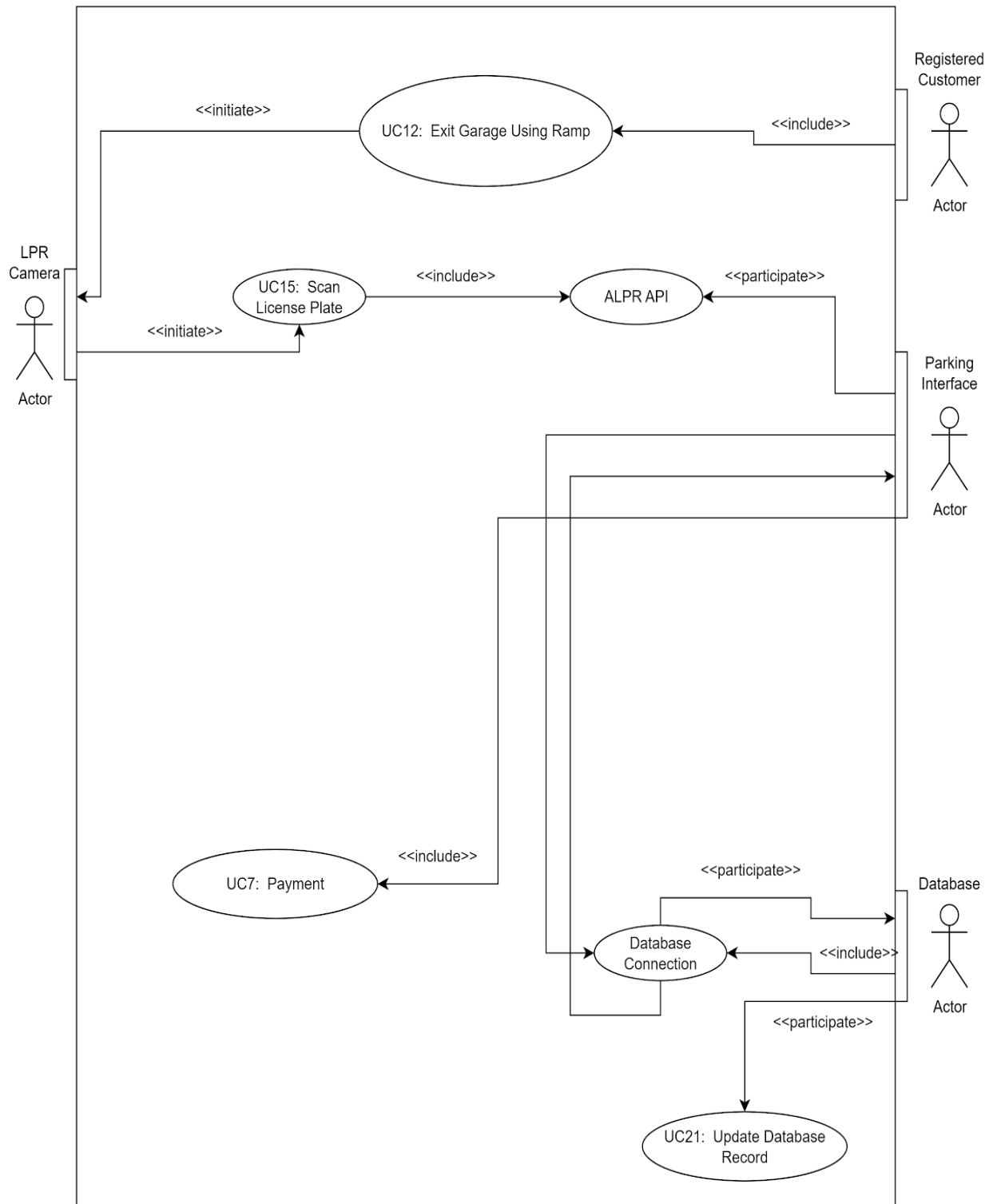
ParkeEZ Use Case Diagram



Garage Elevator Use Case Diagram



Registered Customer Exits Garage Use Case Diagram



iii. Traceability Matrix

Requirements	Priority Weights	U C1	U C2	U C3	U C4	U C5	U C6	U C7	U C8	U C9	UC 10	UC 11	UC 12	UC 13	UC 14	UC 15	UC 16	UC 17	UC 18	UC 19	UC 20	UC 21
REQ1	10			✓		✓																✓
REQ2	10							✓										✓				✓
REQ3	8										✓	✓										
REQ4	8			✓		✓																
REQ5	9																✓					
REQ6	7													✓	✓							
REQ7	8																					✓
REQ8	6																			✓		
REQ9	5			✓	✓	✓																
REQ10	6					✓										✓	✓					
REQ11	3				✓																	
REQ12	10			✓		✓																
REQ13	5							✓			✓	✓										
REQ14	2			✓		✓									✓							
REQ15	7								✓			✓										
REQ16	7						✓			✓		✓						✓	✓			
REQ17	8	✓																				
REQ18	8		✓																			
REQ19	8		✓																			
REQ20	8	✓	✓																			

REQ21	8	✓																	✓			
REQ22	9					✓											✓	✓				
REQ23	9	✓	✓	✓	✓	✓												✓				
REQ24	8	✓				✓					✓											
REQ25	5		✓		✓																	
REQ26	2		✓		✓								✓									
REQ27	6						✓	✓	✓										✓			
REQ28	7			✓		✓					✓						✓	✓				
REQ29	8					✓			✓		✓											
REQ30	8									✓	✓		✓	✓								
REQ31	4																		✓	✓		
REQ32	7	✓	✓	✓	✓	✓										✓						
Max Priority Weight		8	9	10	9	10	8	10	6	7	8	8	8	7	8	8	9	10	9	6	8	10
Total Priority Weight		16	49	51	31	64	31	35	11	13	21	21	58	7	19	21	15	42	31	6	14	32

iv. Fully Dressed Diagrams

Use Case (1):	Sign-in (need)
Related Requirements:	
Initiating Actor:	Customer
Actor's Goal:	To log into their account in ParkeZ
Participating Actors:	System (database)
Preconditions:	<ol style="list-style-type: none"> 1. The customer has an account 2. The customer does not have an account
Success End Condition(s):	Customer successfully logs into their account.
Failed End Condition(s):	Customer does not successfully log into their account.
Extension Points:	UC-2(log-in),

Flow of Events for Main Success Scenario:

Include::sign up UC-1

Include::login UC-2

1. **Customer** visits Park Ez application
2. **System** requests **Customer** login
3. **Customer** inputs login credentials
4. **System** verifies **Customer** credentials with **Database** credentials.
5. **System** allows **Customer** to access Park Ez's **Parking User Interface**.

Flow Events for Extensions:

Any Step: Customer leaves window or loses connection.

I.System deletes all imputed data and changes nothing.

3a. Customer requests a sign-up.

I.System sends **Customer** to sign-in form page.

II.System requests unique and valid credentials for new user.

III.Customer inputs custom credentials

IV.System verifies **Customer** sign-up credentials with **Database** credential.

A. System validates unique and valid credentials and proceeds to **5**.

B. System denies non-unique or invalid credentials and repeats from **3a:2**.

V. System allows **Customer** access to Park Ez's **Parking User Interface**.

3b. System does not find **Customer**'s login credentials in the **Database**.

I.System denies **Customer** access to the **Parking User Interface**

II.System repeats from **2**

Use Case (5):

(Walk-in)

Related Requirements:

REQ-1, REQ-4, REQ-7, REQ-8, REQ-14

Initiating Actor:

Customer

Actor's Goal:

To use parking garage.

Participating Actors:

System (database)

Preconditions:

The customer arrives without a reservation.

Success End Condition(s):

Customer successfully uses the parking garage.

Failed End Condition(s):

The Customer does not use the parking garage.

Extension Points:

Flow of Events for Main Success Scenario:

include::login UC-2

include::Available Parking Space UC-3

include::Choose parking spot UC-4

include::Arrival UC-11

1. **Customer** pulls up to the Parking garage without reservation.

2. **System** (a) recognizes that the Customer has no reservation, (b) checks to see if the customer is a registered user, (c) if the customer is not a registered user the system checks if there are any open parking spaces, (d) if there are parking spaces allow the customer to make a reservation (send to UC-16 step 1) on the ground floor.
3. **Customer** (a) makes a reservation, (b) enters the garage.

Flow Events for Extensions:

- 2b. The customer arrives without a reservation but is a registered user.
 1. **Customer** pulls up to the parking garage without a reservation.
 2. **System** (a) recognizes that the customer has no reservation, (b) checks to see if the customer is a registered user, (c) if the customer is a registered user the system checks if there are any available parking spaces, (d) if there are parking spaces allow the customer to make a reservation on any floor (send to UC-16 step 1).
 3. **Customer** (a) makes a reservation, (b) enters the garage, (c) if parking above the ground floor the customer enters the garage elevator.
 4. **System** (a) the elevator displays the customers parking space, (b) the elevator arrives at the desired floor.
 5. **Customer** (a) exits the elevator and drives to the reserved parking space.

Use Case (6):	Reservation Editing/Cancelation
Related Requirements:	REQ-1, REQ-5, REQ-7, REQ,11, REQ-16
Initiating Actor:	Customer
Actor's Goal:	To edit or cancel an existing reservation.
Participating Actors:	System (database)
Preconditions:	The customer has a reservation
Success End Condition(s):	The reservation is successfully modified or deleted
Failed End Condition(s):	The reservation is not successfully modified.

Extension Points: UC10 reservation extension

Flow of Events for Main Success Scenario:

include::login UC-2

include::Reservation Extension UC10

1. **Customer** requests to modify an existing reservation.
2. **System** (a) checks if **Customer** has an existing reservation, and whether the reservation is currently in progress, (b) **System** displays reservation information for **Customer** to modify.
3. **Customer** modifies or deletes reservation.

4. **System** (a) updates database, (b) shows **Customer** modified reservation or deleted reservation.
5. **Customer** confirms new information or exits the window.

Flow Events for Extensions:

Any Step: Customer leaves window or loses connection.

1. **System** deletes all imputed data and changes nothing.
- 2a. **System** does not find reservations.
 1. **System** (a) informs the **Customer** that they do not have a reservation, (b) **System** sends **Customer** to create a reservation page.
3. Customer attempts to delete an ongoing reservation.
 1. **System** (a) refuses to delete an ongoing reservation, (b) **System** informs **Customer** that they cannot delete an ongoing reservation, (c) **System** sends **Customer** to reservation page.
3. Customer requests to extend reservation.
 1. **System** (a) sends Customer to step 1 of UC-10, (b) afterwards sends customer to step 4.

Use Case (7):	Payment
Related Requirements:	REQ-5, REQ-7, REQ-13, REQ-11, REQ-13, REQ-15
Initiating Actor:	System
Actor's Goal:	To receive payment from Customer.
Participating Actors:	Customer, database
Preconditions:	The customer has completed their stay in the Parking garage.
Success End Condition(s):	System receives Customer payment.
Failed End Condition(s):	System does not receive Customer payment.

Extension Points:

Flow of Events for Main Success Scenario:

include::login UC-2

include::payment -method UC-17

include::Departure UC-12

1. **Customer** leaves the parking garage.
2. **System** (a) notices **Customer** leaving Garage, (b) **System** calculates how much the **Customer** owes, (c) **System** sends invoice to **Customer**.
3. **Customer** (a) receives the invoice, (b) pays.
4. **System** (a) updates database to mark invoice as paid, (b) **System** sends receipt to **Customer**.

Flow Events for Extensions:

Any Step: Customer leaves reservation window or loses connection.

1. **System** (a) deletes all input data and returns to the last completed step, if needed resending information.

3b.

1. **System** If the **Customer** (a) does not pay in time, (b) **System** resends invoice, (c) repeat x number of times.
2. **System** if the **Customer** (a) still does not pay in time, (b) alert **Garage Administrators**.
3. **Garage Administrators** will either send information to a collection agency or file a lawsuit against the **Customer**.

Use Case (16):	Reservation
Related Requirements:	REQ-1, REQ-5, REQ-7, REQ,11
Initiating Actor:	Customer
Actor's Goal:	To book a reservation at the parking garage.
Participating Actors:	System (database)
Preconditions:	Customer has an account, Customer has set up Payment options.
Success End Condition(s):	Customer successfully set up a reservation.
Failed End Condition(s):	Customer does not successfully set up a reservation.
Extension Points:	UC-6

Flow of Events for Main Success Scenario:

include::login UC-2

include::Available Parking Space UC-3

include::Chose parking spot UC-4

1. **System** (a) displays parking space availability data by floor, (b) **System** also displays the option of having a guaranteed parking spot.
2. **Customer** choses a parking space, and the amount of time the Customer wants to reserve the parking space for.
3. **System** (a) accepts the reservation, (b) **System** makes changes to the database, (c) **System** receives notice that the reservation was made successfully, (d) **System** shows Customer reservation success and reservation information.
4. **Customer** confirms that reservation is correct or exits system prompt.

Flow Events for Extensions:

Any Step: **Customer** leaves reservation window or loses connection.

1. **System** deletes all input data and confirms nothing.

- 1b. **Customer** chooses the guaranteed reservation option.
 1. **System** asks **Customer** which parking space they would like to reserve, and how long they would like to reserve it for (minimum of one month).
 2. **Customer** chooses a parking space and a duration.
 3. **System** sends **Customer** to UC-16 step 3.
2. **Customer** attempts to pick an already selected parking place.
 1. **System** (a) informs the **Customer** that that parking place is already reserved. (b) sends the **Customer** to step 2 (repeat if needed).

Use Case (22):	Elevator access.
Related Requirements:	REQ-9, REQ-10, REQ-12, REQ-13
Initiating Actor:	Customer
Actor's Goal:	To use the elevator in the parking garage.
Participating Actors:	System
Preconditions:	The customer is at the garage.
Success End Condition(s):	Customer successfully uses the garage.
Failed End Condition(s):	Customer does not successfully use the garage.

Extension Points:

Flow of Events for Main Success Scenario:

include::Registered Customer With Reservation Arrives UC-22

include::Arrival UC-11

include::Scan Plate UC-15

1. A registered customer arrives at the garage in a vehicle with the same license plate listed on the reservation.
 1. **Customer:** a) Pulls into the garage elevator.
 2. **System:** a) Scans the vehicle license plate, b) checks the license plate against reservations in the database, c) the database finds a match and returns true, d) displays a message on the elevator display to confirm the reservation.
 3. **Customer:** a) Confirms the reservation using the elevator console.
 4. **System:** a) The elevator display displays the parking space location, b) the elevator takes the customer to the floor the reserved parking space is located on.
 5. **Customer:** a) The customer exits the elevator on the desired floor.

Flow Events for Extensions:

2. A registered customer with a reservation arrives in a vehicle with a different license plate than the plate listed on the reservation.
 1. **Customer:** a) Pulls into the garage elevator.
 2. **System:** a) Scans the vehicle license plate, b) checks the license plate against reservations in the database, c) the database does not find a match and returns false, d) displays a message on the elevator display to confirm the reservation by entering the reservation confirmation number on the elevator console or press

“cancel” to quit if the customer is not a registered customer with a valid reservation..

3. **Customer:** a) Enters a reservation confirmation number using the elevator console.
 4. **System:** a) If the reservation confirmation number is valid the elevator display displays the parking space location, b) the elevator takes the customer to the floor the reserved parking space is located on.
 - a. **System:** If the reservation confirmation number is invalid the system displays a message on the elevator display to re-enter the confirmation number or press “cancel” to quit if the customer is not a registered customer with a valid reservation.
 - b. **Customer:** Enters a reservation confirmation number using the elevator console.
 - c. **System:** If the reservation confirmation number is valid the elevator display displays the parking space location.
 - d. **System:** The elevator takes the customer to the floor the reserved parking space is located on.
 5. **Customer:** a) The customer exits the elevator on the desired floor.
3. A Walk-in customer arrives at the parking garage.
1. **Customer:** a) Pulls into the garage elevator.
 2. **System:** a) Scans the vehicle license plate, b) checks the license plate against reservations in the database, c) the database does not find a match and returns false, d) displays a message on the elevator display to confirm the reservation by entering the reservation confirmation number on the elevator console or press “cancel” to quit if customer is not a registered customer with a valid reservation.
 3. **Customer:** a) Presses “cancel” on the elevator console.
 4. **System:** a) Displays a message on the elevator display to “Please exit the elevator and proceed to the 1st floor display for a list of available walk-in parking spaces.”
 5. **Customer:** a) Leaves the garage elevator.

Section 5: Effort Estimation Using Use Case Points

Unadjusted Actor Weight:

Parking UI: 3
Parking Interface: 3
Customer Information System: 1
Parking information System: 1
Security System: 1
Camera: 1
Customer registration/login: 1
Customer payment method/modification: 1
Customer reservation: 3
Customer availability slot: 2
Customer at garage: 1
Customer exit/pay: 1
Customer reservation confirmation: 1
Customer reservation edit/cancellation: 1
Elevator System: 2

UAW = 23

Unadjusted Use Case Weight

Sign Up: 5
Login: 5
Availability Parking Space: 5
Slot Choosing: 5
Walk-in: 5
Reservation Editing/Cancellation: 5
Payment: 5
Time Duration: 10
Time Update: 5
Reservation Extension: 5
Arrival: 5
Departure: 5
Information: 5
Email/SMS Notification: 15
Scan plate: 5
Reservation: 5
Payment Method: 5
Info update: 5
Reporting and Analytics: 15
Summary/History: 10
Database Record Creation: 10

Arrive at Garage: 5

UUCW = 145

Unadjusted Use Case Points

UCCP = 168

Technical Complexity Factors

T1: $3 * 2 = 6$

T2: $1 * 1 = 1$

T3: $1 * 1 = 1$

T4: $2 * .5 = 1$

T5: $2 * .5 = 1$

T6: $1 * 2 = 2$

T7: $2 * 0 = 0$

T8: $3 * 0 = 0$

T9: $1 * 0 = 0$

T10: $0 * 1 = 0$

Technical Factor Total = 12

TCF = $0.6 + 0.1 * 12 = .861$

Environment Complexity Factor

E1: $5 * 1.5 = 7.5$

E2: $3 * .5 = 1.5$

E3: $5 * .5 = 2.5$

E4: $5 * 1 = 5$

E5: $2 * 3 = 6$

E6: $3 * -1 = -3$

Environment Factor Total = 19.5

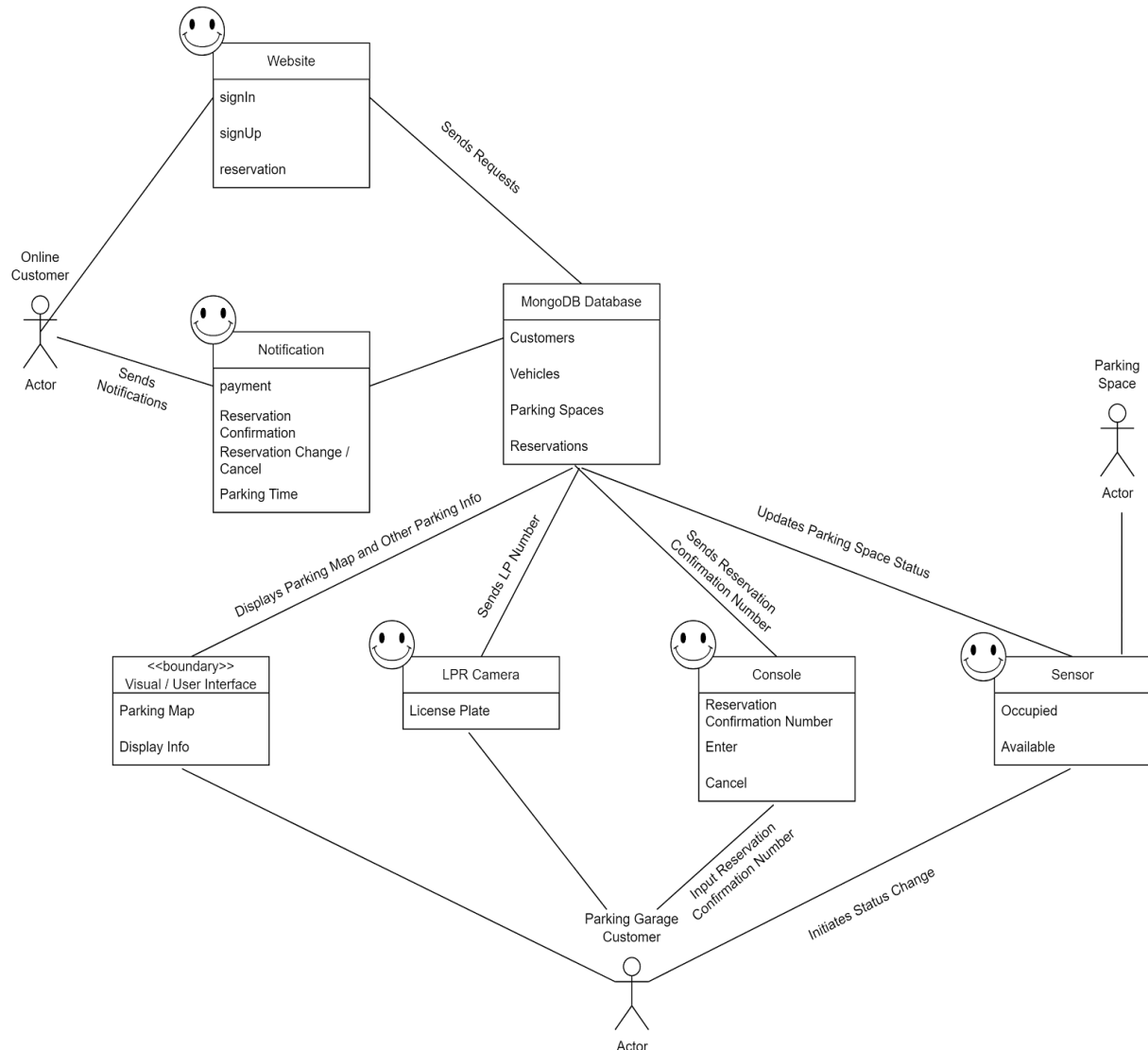
ECF = $1.4 * (-.03 * 19.5) = .815$

Use Case Point

UCP = $168 * .861 * .815 = 118$

Section 6: Domain Analysis

Domain Model



The Domain Model has 7 main concepts, which include the website, MongoDB database, notification, visual / user interface, LPR camera, console, and sensor. The website allows online customers to register / sign-up (UC1) and then sign-in (UC2) once they are registered. The website also gets the status of all parking spaces from the MongoDB database and updates the parking space map on the website. The website allows online customers to make reservations (UC16) and edit or cancel existing reservations (UC6). Customer information, vehicles, reservations, and parking spaces are stored as records in the database (UC21).

When customers arrive at the parking garage the LPR camera will take an image of the vehicle license plate (UC15). That image is translated into the string form of the

license plate number by the LPR API and stored in the database. If the customer is a walk-in customer (UC5) they will use the visual / user interface to reserve a parking space and enter payment information. If the customer is a registered member with a current reservation they will pull into the elevator (UC22). The elevator sensor will show a vehicle is present once the customer pulls into the elevator. A LPR camera will then take an image of the vehicle license plate (UC15), which will then be converted to a string and then checked against reservation records in the database. If a reservation match is confirmed the customer's parking space and level will be output on the elevator display and then the elevator proceeds to the desired floor. If a reservation match is not found, then the customer will be asked to input the reservation confirmation number using the elevator console. The database checks for a matching reservation confirmation number for the current time period. If a match is found the parking space and level is displayed on the elevator display and the elevator goes to the desired floor.

When customers pull into their reserved parking space the parking space sensor toggles from available to occupied and that information is sent to the database and the time parked counter starts (UC8). For this project we will assume that the customer always parks in the correct parking space.

When a customer leaves their parking space the parking space sensor toggles from occupied to available. That information is sent to the database and the time parked counter stops (UC8). When the customer leaves the parking garage (UC12) the LPR camera positioned at the exit takes an image of the rear license plate (UC15), the image is translated onto a string of text by the API and then is sent to the database. Once the database receives the exit license plate the payment is triggered (UC17). Notifications are sent to the customer after all major events such as, reservation confirmation, confirmation of reservation change or cancellation, profile information change and payment confirmation (UC14).

Domain Concept Definitions (D for Doing, K for Knowing)

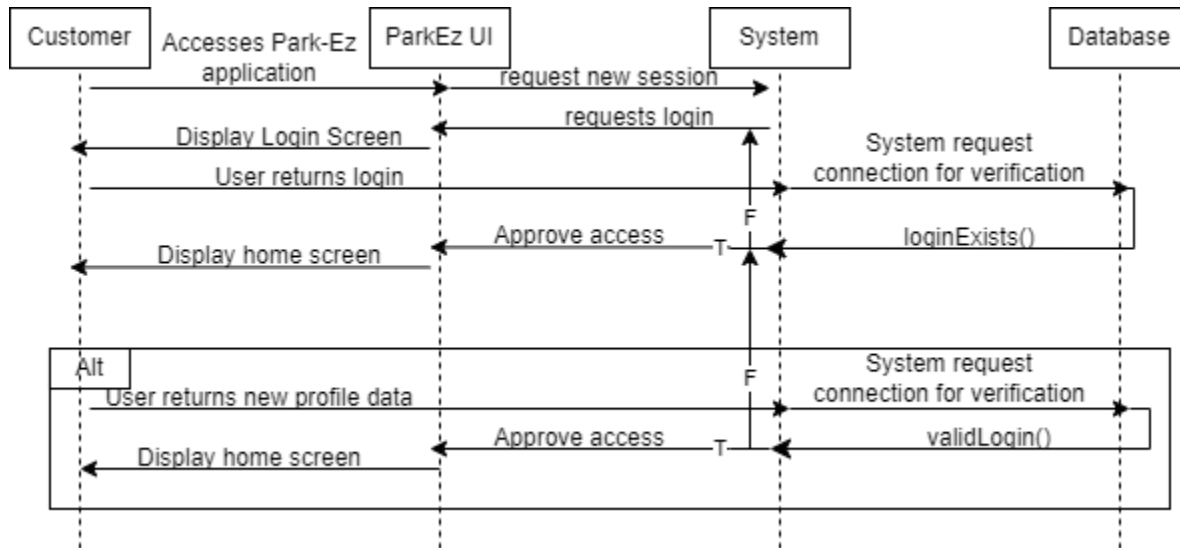
Responsibility Description	Type	Concept Name
To obtain customer sign-up information.	D	Website
To obtain customer sign-in information.	D	Website
To validate customer.	D	Database
To obtain parking space information.	K	Database
To obtain customer information.	K	Database
To obtain reservation information.	K	Database
To obtain vehicle information.	K	Database
To initiate customer reservation.	D	Website
To manage customer information and reservation status.	K	Website
To obtain vehicle license plate.	D	LPR Camera
To change or cancel reservation.	D	Website
To display available parking garage spaces.	K	Visual / User Interface
To get customer reservation ID number.	D	Console
To select customer payment method.	D	Website
To update parking space status.	D	Sensor
To initiate customer payment.	D	LPR Camera
To send reservation confirmation email.	D	Notification
To send payment confirmation email.	D	Notification
To send impending parking space reservation expiration notification.	D	Notification
To show remaining time of reservation.	D	Notification
To record successful / unsuccessful customer transactions.	D	Database
To validate license plate.	D	Database
To validate reservation confirmation number.	D	Database

Domain Concept to Use Case Traceability Matrix

Use Case	PW	Website	Database	Notification	Visual / User Interface	LPR Camera	Sensor	Console
UC1	8	X	X	X				
UC2	8	X	X					
UC3	7	X			X			
UC4	7	X	X	X	X			
UC5	8		X		X			X
UC6	4	X	X	X				
UC7	8		X	X				
UC8	8		X				X	
UC9	3		X	X			X	
UC10	3	X	X	X				
UC11	8		X		X	X		
UC12	8		X	X		X		
UC13	7	X						
UC14	5			X				
UC15	7					X		
UC16	8	X	X	X				
UC17	7	X						
UC18	5	X	X					
UC19	2	X		X				
UC20	3	X						
UC21	7		X					
UC22	8		X		X	X	X	X

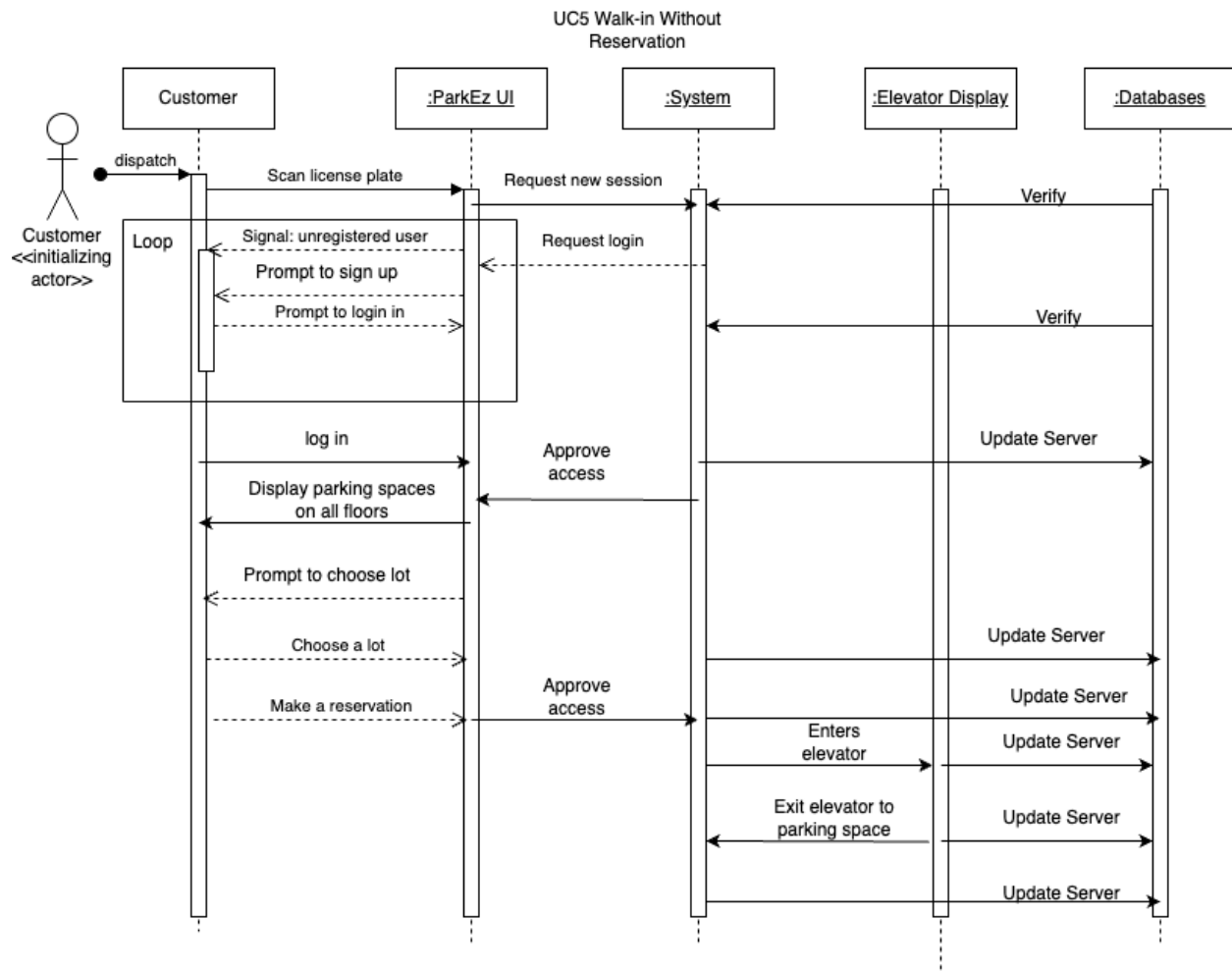
Section 7: Interaction Diagrams

Use Case 1: Sign-in



In the context of the MVC design pattern, interactions with the ParkEz application interface (the View) initiate with a login screen, which integrates Next.js 14's next-auth for authentication and uses the Google provider for sign-in. This screen facilitates both sign-in for existing users and account creation for new users. When the user opts to sign in via Google, the Controller processes this event and liaises with the Model, which now leverages next-auth to handle user authentication against Google's user accounts. Successful authentication by Google allows the user entry; if not, an error message is displayed. For new users choosing to register, next-auth streamlines the process, ensuring the uniqueness of user details before updating the Model, in this case, the application's user database. Post authentication or account creation, the user is seamlessly redirected to the main interface of the ParkEz application.

Use Case 5: Walk-in



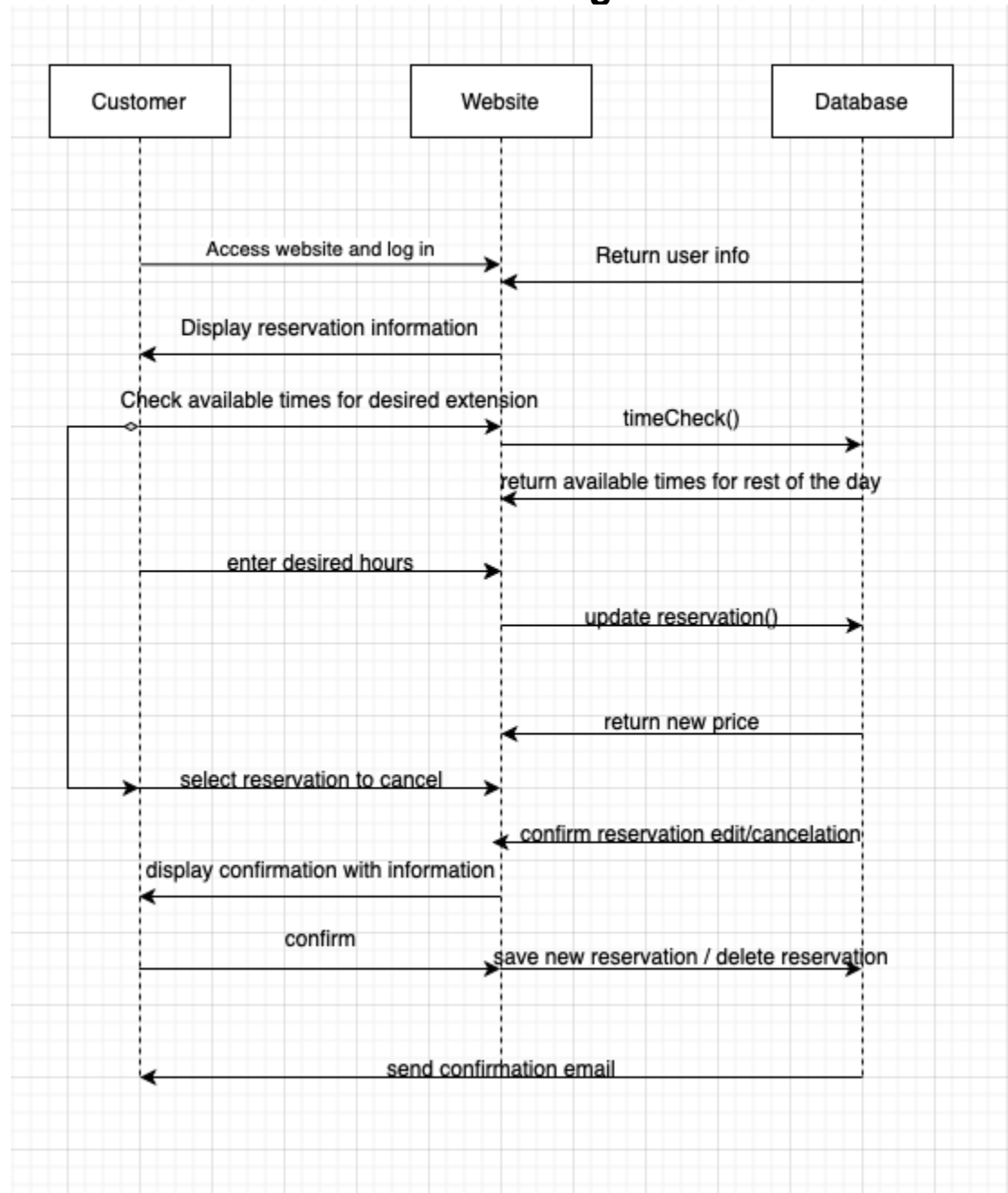
UML System sequence diagrams for walk-in use case using draw.io

In the MVC design pattern, as applied to the ParkEz application with Next.js 14, the flow begins when the user's license plate is scanned (View). Should the user not already have an account, the View prompts them to register via a Next.js 14-enhanced interface, utilizing next-auth for streamlined authentication processes, including options like Google's OAuth for a secure and user-friendly sign-in experience.

After registration or log-in, the Controller takes charge, processing the user's input and communicating with the Model, where the logic to verify the user's credentials and parking space availability is handled. The Model, backed by a database, confirms the user's status, and returns the relevant data to the Controller, which then updates the View to display available parking spaces.

When the user makes a selection, this choice is communicated back to the Controller, which updates the Model. The Model processes the reservation and confirms it by updating the database. The Controller may also interact with other system components, such as an elevator display system, to guide the user to their designated parking space.

Use Case 6: Reservation Editing/Cancellation



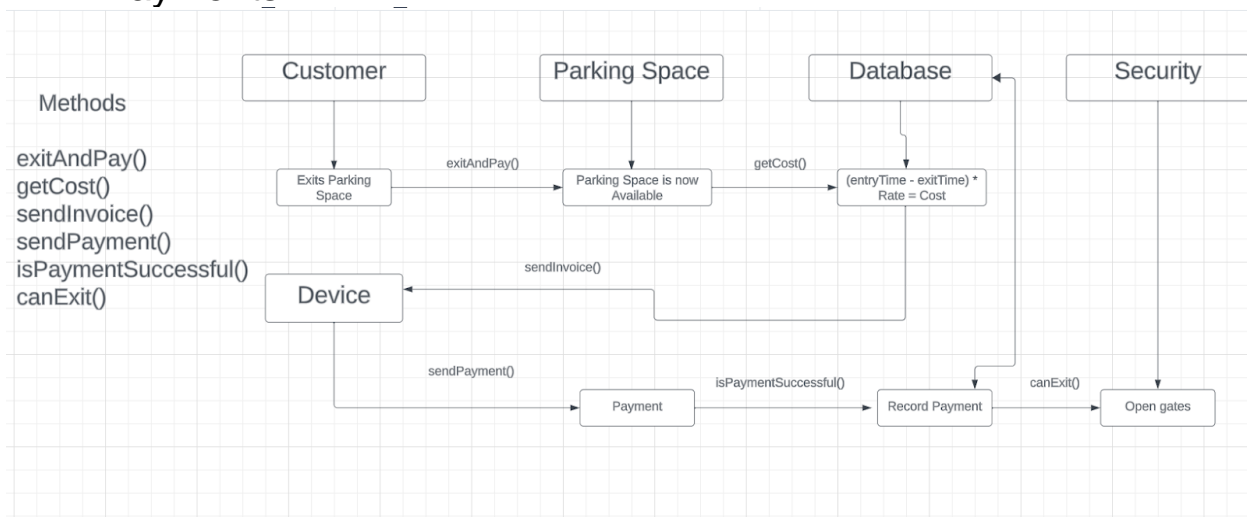
In the ParkeZ web application utilizing Next.js 14 and an MVC controller, the reservation editing, and cancellation process is user-centric and systematic. When a customer logs into the ParkeZ web interface, the request is managed by the controller

which communicates with the model to retrieve user information from the database. The model relays the user's data back to the controller, which then instructs the view to present the customer's reservation details and the available times for extension. If a customer wishes to extend their reservation, they select a new time slot through the interface. This selection is processed by the controller that updates the reservation in the model, which in turn, updates the database and calculates the new pricing.

Should the customer opt to cancel a reservation, this choice is also routed through the view to the controller. The model then confirms the cancellation, updates the database accordingly, and the controller signals the view to display a confirmation message. The customer is asked to confirm these changes; upon confirmation, the controller ensures that the model either saves the new reservation details or deletes the existing reservation. To finalize the process, the model initiates the sending of an email confirmation to the customer, providing a record of the updated reservation status and ensuring clarity and completion of the transaction. This architecture provides a seamless and interactive experience, ensuring that each step is intuitive for the customer and efficiently processed by the system.

Use Case 7: Payment

Payments



As the customer prepares to exit the parking space, a sensor registers the change from 'occupied' to 'available' and communicates this to the controller via a function called 'exitAndPay()'. This event triggers the controller to interact with the model, which queries the database for the customer's data associated with that parking space.

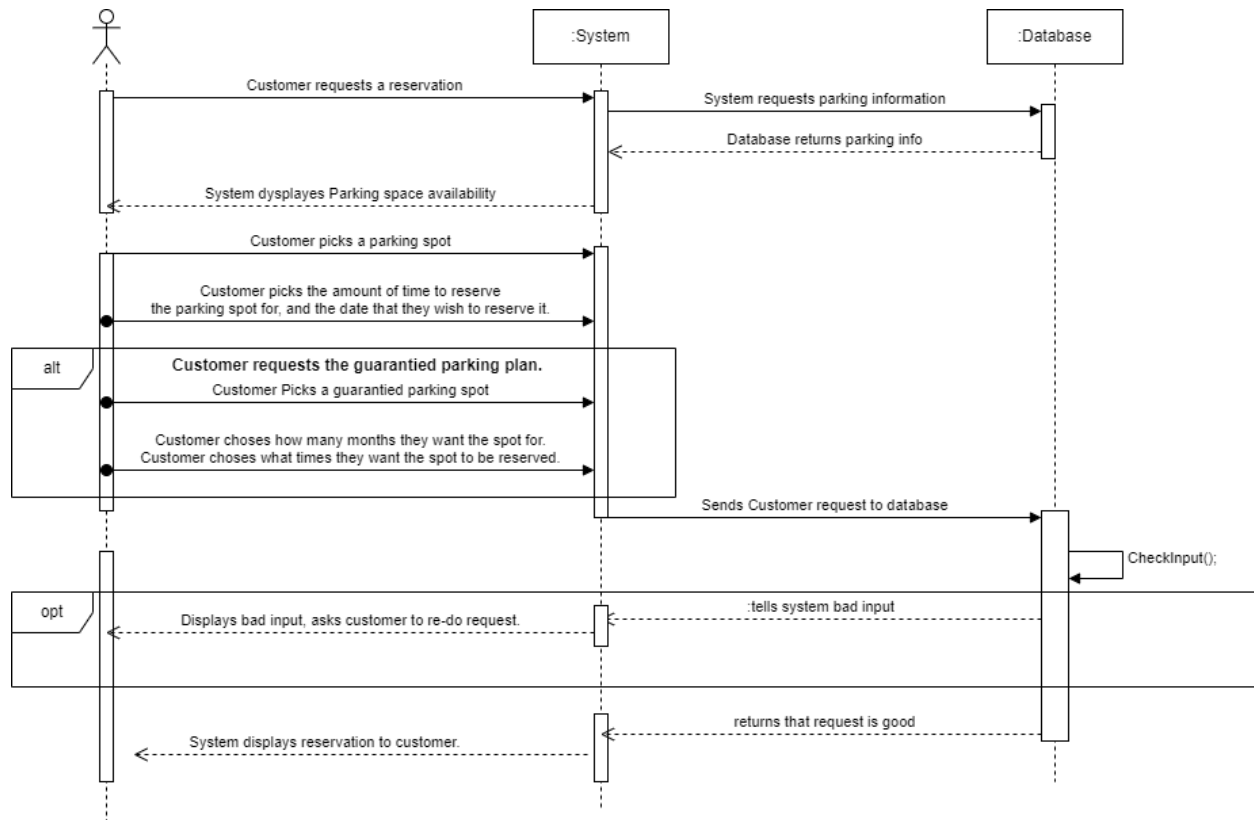
The data retrieved includes the customer's payment method and vehicle information, along with the parking space's time log that captures both entry and exit times. Using this data, the model calculates the cost of parking by applying the appropriate rate to the duration the space was occupied, which is the difference between the exit time and entry time.

With the cost determined, the model proceeds to create a transaction that includes the total cost, timestamps, and the identifiers for the customer, vehicle, and parking space. This transaction data is essential for the 'sendInvoice()' method, which the controller uses to instruct the Next.js 14 front end to generate and send a formatted invoice to the customer's device.

Upon receipt of the invoice, the customer's device issues a 'sendPayment()' command, which interacts with the controller. The controller, through the model, processes the payment using the customer's chosen method. If the payment is successful, as confirmed by the 'isPaymentSuccessful()' check, the model updates the transaction status and informs the controller. The controller then signals the security system to open the gates, allowing the customer to exit.

This process showcases how the MVC architecture facilitates the separation of concerns, with the model handling data and business logic, the view presenting data, and the controller managing user input and application flow.

Use Case 16: Reservation



In the context of a web application built with Next.js 14 and leveraging an MVC architecture, the process for a customer making a parking reservation on the ParkeZ webpage is both intuitive and structured.

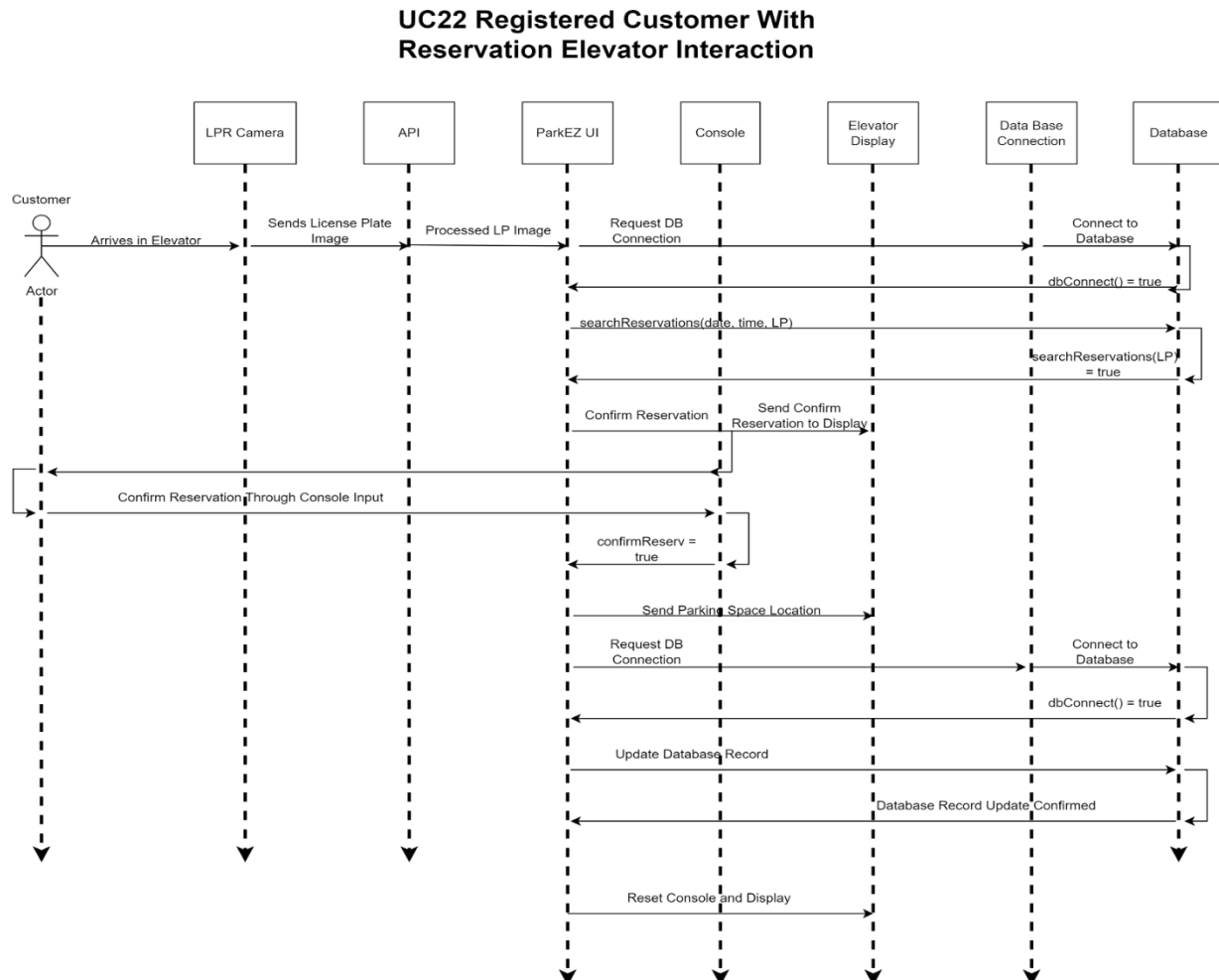
Upon logging into the ParkeZ webpage and navigating to the reservation tab, the customer is presented with an option by the view layer: making a reservation. The controller, which mediates between the view and the model, captures the customer's selection and conveys it to the model for further action.

If the customer opts for a single reservation, the view prompts them to input details such as the date, start time, and duration of the reservation. They are also given the opportunity to select a parking spot from a list of available options provided by the model, which has queried the database for current parking space availability.

After the customer inputs their reservation details, the controller instructs the model to perform an input check. If there is an issue with the input, the model informs the controller, which then prompts the view to notify the customer of the error and request that they revisit the reservation process. If the input is correct, the model confirms the reservation details and updates the database accordingly. Subsequently, the controller instructs the view to send a confirmation message to the customer, confirming the successful reservation.

Upon completion of the process, the customer can then exit the reservation tab, having been assisted by a seamless integration of Next.js 14's reactive user interface and the efficient processing of the MVC architecture. This system provides a clear path from customer input to database update, ensuring that reservation details are accurately captured and confirmed.

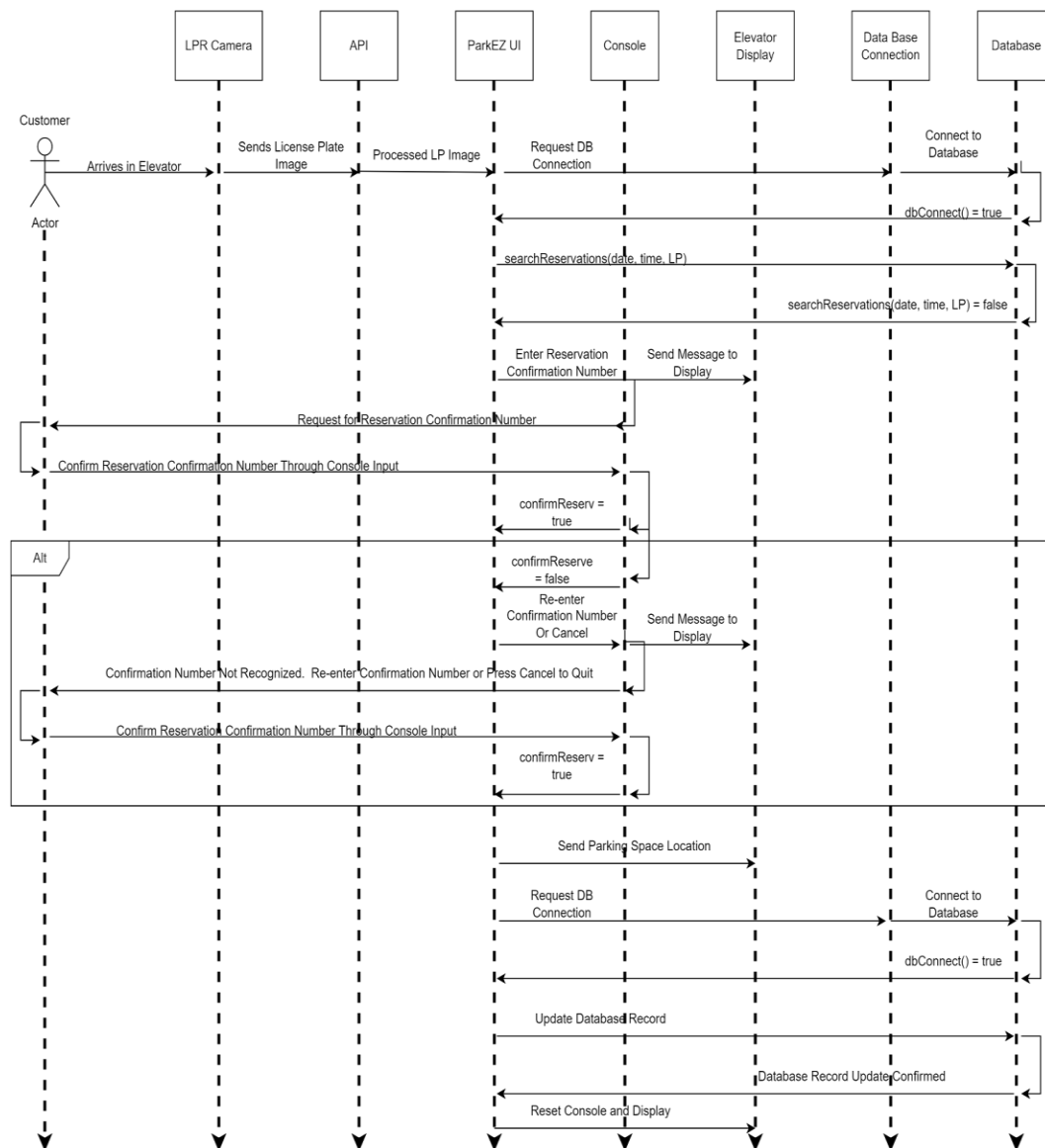
Use Case 22: Elevator Access



In a Next.js 14 application using MVC architecture, a registered user's parking garage entry is managed through coordinated interactions between hardware and software components. As the customer enters the garage elevator, an LPR camera reads their license plate and communicates with the API. The processed data is displayed on the ParkeZ UI, which confirms the reservation by querying the database for a match with the license plate and current date and time.

Once verified, the UI prompts the elevator console for user confirmation. The customer confirms, and the UI provides visual guidance to the allocated parking space. After parking, the UI records the transaction in the database, updates the reservation status, and resets the system for the next user.

UC22 Registered Customer With Reservation Alternate Interaction: License Plate Does Not Match Reservation

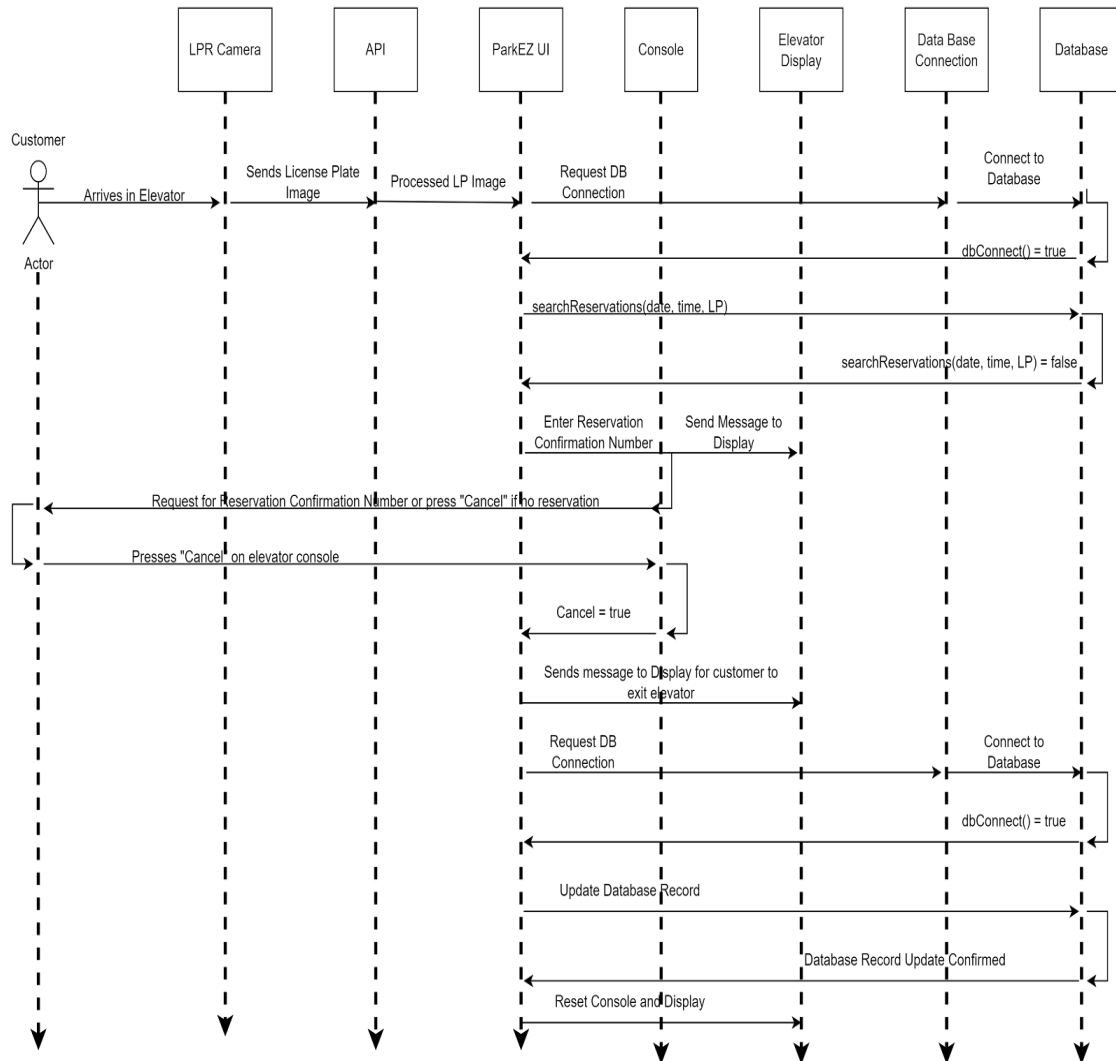


In the given scenario, leveraging an MVC architecture, the sequence of events unfolds as follows: A user, with a reservation but a non-matching license plate, enters the parking garage and the LPR camera scans their plate. This triggers the View component of the ParkeZ application to interact with the Controller, sending the image to an API for processing. Upon receipt of the processed license plate number, the Controller attempts to validate the user's reservation by establishing a connection with the Model, which represents the database. The initial search for a matching reservation based on the date, time, and license plate is unsuccessful.

In response, the View displays a prompt on the elevator console, asking the user to input their reservation confirmation number. This action by the customer is captured by the View and passed to the Controller, which conducts a second verification with the Model. This time, the input confirmation number matches the reservation details. The Controller then confirms the reservation status to the View, which displays a personalized greeting and directions to the reserved parking spot on the elevator console.

Following the user's confirmation, the Controller commands the Model to record the elevator interaction and update the reservation record in the database. Once the transaction is complete, the Controller ensures that the View resets all components, including the camera, API, console, and display, to their original states, ready for the next user. This process demonstrates the efficient problem-solving workflow enabled by the MVC architecture, with clear delineation of roles and responsibilities ensuring a seamless user experience despite initial discrepancies.

UC22 Alternate Interaction: Registered Customer Without a Reservation or a Walk-in Customer Arrives in the Parking Garage Elevator



In this interaction scenario within an MVC architecture framework, the user experience is managed through a series of system responses to a walk-in customer without a reservation. As the customer pulls into the parking garage elevator, the LPR camera scans their license plate. The View, part of the ParkeZ UI, communicates this data to the Controller, which coordinates with the Model to process the image through an API.

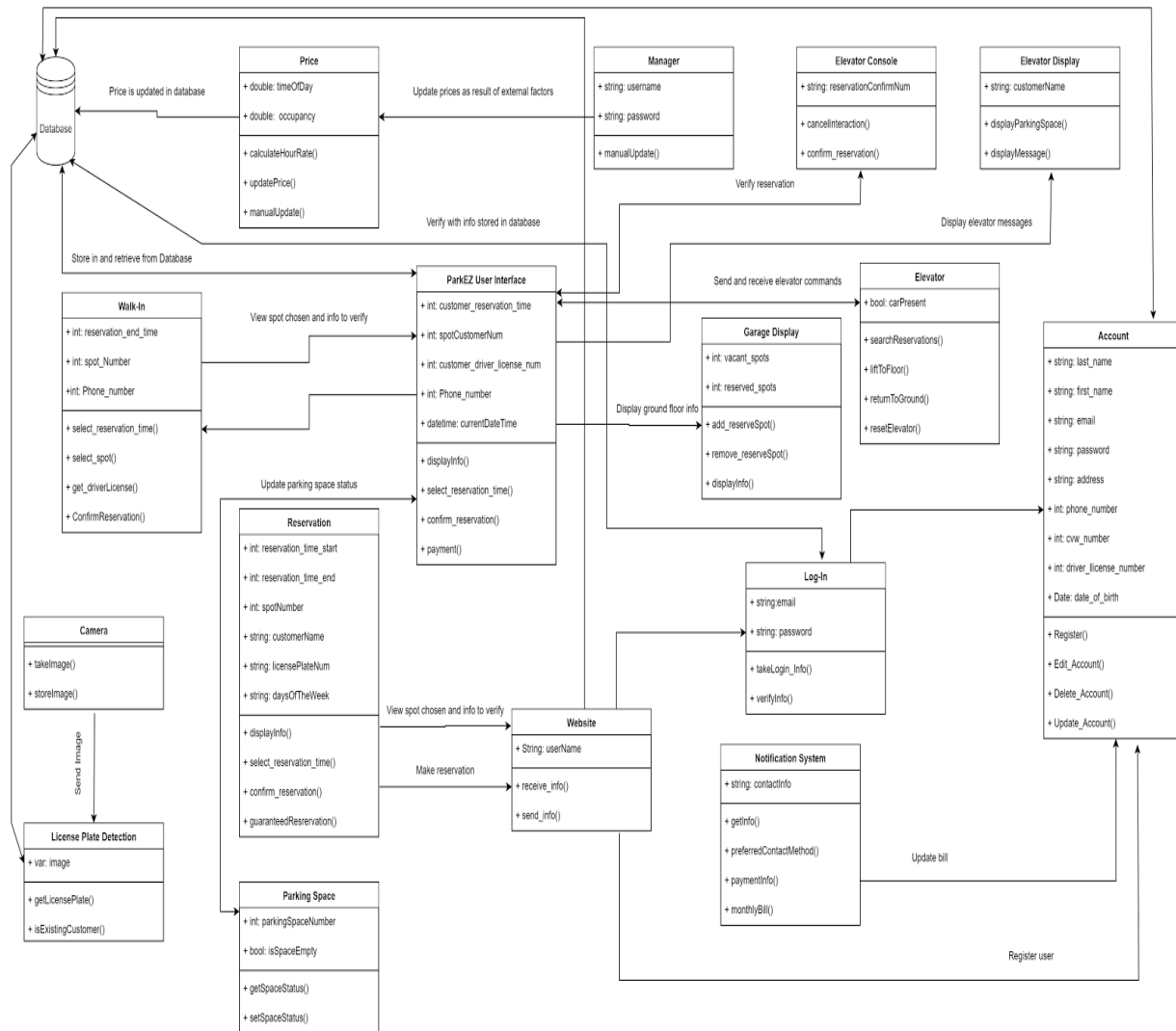
Once processed, the Controller instructs the Model to query the database for a reservation matching the current date, time, and license plate. When no reservation is

found, the View prompts the customer via the elevator display to enter a reservation number. The customer, having none, selects "Cancel" on the console.

The Controller receives the cancellation command and guides the View to instruct the customer to exit the elevator and check the first-floor display for available parking. Simultaneously, the Controller commands the Model to log the interaction in the database before closing the connection.

The process concludes with the Controller ensuring the system components, including the camera, API, console, display, and UI, reset to their initial state, ready for the next user. This interaction reflects the MVC structure's adaptability in handling scenarios that deviate from the typical reservation flow, providing clear instructions to the user while accurately logging system interactions.

Section 8: Class Diagram and Interface Specification



Data Types and Operation Signatures

Account: Includes registered customer information obtained from the ParkeEZ website and saved to the database.

Attributes:

- String: last_name
- String: first_name
- String: email
- String: password
- String: address

- Int: phone_number
- Int: cvw_number
- Int: driver_license_number
- Date: date_of_birth

Operations:

- Register(): Registers a customer through the website.
- Edit_Account(): Edits the customer account.
- Delete_Account(): Removes a customer Account from the database.
- Update_Account(): Updates the customer account in the database.

Camera: Class for the LPR camera that captures an image of a vehicle license plate and sends it to License Plate Detection class for processing.

Attributes:

none

Operations:

- takeImage(): Captures a license plate image.
- storeImage(): Stores the license plate image.

Elevator: Controls the parking garage elevator.

Attributes:

- Bool: carPresent

Operations:

- liftToFloor(): Send elevator to floor of reserved parking space.
- returnToGround(): Sends elevator back to the ground floor after the customer pulls onto the desired floor.
- resetElevator(): Once the elevator returns to the ground floor the elevator, console, camera and display are reset to a beginning state.
- searchReservations(): Queries the database for a reservation that matches the date, time, LP/reservationConfirmNum.

Elevator Console: Takes customer input from inside the elevator.

Attributes:

- String: reservationConfirmNumber

Operations:

- cancelInteraction(): Cancels or Exits the interaction with a customer, as in the case of a walk-in customer that is asked for a confirmation number, but does not have one.
- confirm_reservation(): After a database query for a matching reservation comes back true the customer is asked to confirm the reservation.

Elevator Display: Displays messages to the customer in the elevator.

Attributes:

- customerName

Operations:

- displayParkingSpace(): Displays the customer's parking space.
- displayMessage(): Displays messages/directions to the customer in the elevator.

Garage Display: This is the walk-in display on the first floor of the garage. It is used to show parking space availability.

Attributes:

- Int: vacant_spots
- Int: reserved_spots

Operations:

- add_reserveSpot(): Updates the display with the numbered spot showing as reserved.
- remove_reserveSpot(): Updates the display to show a once reserved spot changing to an available parking space.
- displayInfo(): Displays messages.

License Plate Detection: Receives a captured image of a license plate and processes it into a readable license plate.

Attributes:

- Var: image

Operations:

- getLicensePlate(): Gets the license plate number.
- isExistingCustomer(): Queries database for matching license plate to determine if the vehicle belongs to a registered customer.

Log-In: Website log-in.

Attributes:

- String: email
- String: password

Operations:

- takeLogin_info(): Receives user log-in information.
- verifyInfo(): Verifies user input log-ing information.

Manager: Garage manager has access to adjust the rates.

Attributes:

- String: username
- String: password

Operations:

- manualUpdate(): Updates the current rates.

Notification System: Used to notify garage customers about payments and billing.

Attributes:

- String: contactInfo

Operations:

- getInfo(): Gets user information.
- preferredContactMethod(): Uses the customer's preferred contact method for communications.
- paymentInfo(): Sends notifications for payment confirmations.
- monthlyBill(): Sends notification of customers monthly accumulated bill.

ParkeZ User Interface: The User Interface controls the parking garage automated and user-initiated operations.

Attributes:

- Int: customer_reservation_time
- Int: spotCustomerNum
- Int: customer_driver_license_num
- Int: Phone_number
- Datetime: currentDateTime

Operations:

- displayInfo(): Used to display information to the customer.
- select_reservation_time(): Registered customer can select reservation time.
- confirm_reservation(): Used to confirm customer reservation.
- payment(): Calculates cost based on time and rate and receives payment from the customer.

Parking Space: Instance for each parking space in the garage. The parking space sensors are included in this class.

Attributes:

- Int: parkingSpaceNumber
- Bool: isSpaceEmpty

Operations:

- getSpaceStatus(): Gets the current status of a parking space. Is it empty or is it occupied?
- setSpaceStatus(): Changing the state of the parking space.

Price: Used to change the rates based on time of day, overstay or manager override.

Attributes:

- Double: timeOfDay
- Double: occupancy

Operations:

- calculateHourRate(): Calculates the hourly rate based on peak hours or other circumstances.
- updatePrice(): Updates the price.
- manualUpdate(): manager can update price manually.

Reservation: Used to initiate a parking space reservation.

Attributes:

- Int: reservation_time_start
- Int: reservation_time_end
- Int: spotNumber
- String: customerName
- String: licensePlateNum
- String: daysOfTheWeek

Operations:

- displayInfo(): Used to display customer reservation information.
- select_reservation_time(): Customers can select the desired reservation date and time.
- confirm_reservation(): Once a reservation is selected the customer has the option to confirm it.
- guaranteedReservation(): The customer has the option to select a guaranteed reservation, which adds a reservation for recurring dates and times over an extended period of time.

Walk-In: Handles walk-in customers in the garage.

Attributes:

- Int: reservation_end_time
- Int: spot_Number
- Int: Phone_number

Operations:

- select_reservation_time(): Walk-in customer selects reservation time.
- select_spot(): The customer can select an unoccupied and unreserved parking space.
- get_driverLicense(): Gets the walk-in customers driver's license number.
- ConfirmReservation(): Confirms the walk-in customers parking space reservation.

Website: ParkEZ website allows customers to register and reserve parking spaces.

Attributes:

- String: userName

Operations:

- receive_info(): Receives customer information from the database.
- send_info(): Sends information to the database.

Domain Concept / Class Integration Traceability Matrix

Domain Concepts	Software Classes															
	Website	Account	Log-In	Reservation	Notification System	ParkEZ User Interface	Walk-In	Garage Display	Elevator	Camera	License Plate Detection	Parking Space	Price	Manager	Elevator Console	Elevator Display
Website	X	X	X	X									X	X		
Database		X	X	X		X						X	X			
Notification				X	X											
Visual / User Interface						X	X	X	X					X		X
LPR Camera									X	X	X					
Sensor									X			X				
Console									X						X	

1. Website

a. Responsibilities include:

1. Registration
2. Log-In
3. Reservations
4. Edit / Cancel Reservations
5. Update / Change customer profile

b. Classes:

2. Database

a. Responsibilities include:

1. Store customer information
2. Store reservations
3. Store parking space status
4. Store parking garage interactions
5. Store vehicle information

b. Classes:

3. Notification

a. Responsibilities include:

1. Send reservation confirmation
2. Send reservation cancel / edit confirmation

- 3. Send payment confirmation
 - 4. Send parking space timer information
- b. Classes:
- 4. Visual / User Interface
 - a. Responsibilities include:
 - 1. Display parking map with available parking spaces
 - 2. Take walk-in information
 - 3. Display number of available spaces or no vacancy when garage is full
 - b. Classes:
- 5. LPR Camera
 - a. Responsibilities include:
 - 1. Take image of vehicle license plates on entrance and exit of the garage
 - b. Classes:
- 6. Sensor
 - a. Responsibilities include:
 - 1. Change parking space status to occupied when a vehicle is present
 - 2. Change parking space status to available when no vehicle is present
 - 3. Update database of parking space status change
 - b. Classes:
- 7. Console
 - a. Responsibilities include:
 - 1. Accept required input from user, such as reservation confirmation ID
 - b. Classes:

Design Patterns:

Object Constraint Language (OCL) Contracts:

CLASS: account

Invariants

Context: Account

inv:self.userSign-in;

Preconditions

pre:self.userExists=true

Postconditions

post:self.userAccount=self.parkEZ

CLASS reservation

Invariants

Context:reservation

Inv:self.reservation

Context:reservation

inv:self.spotNumber

Preconditions

Context reservation

Pre:self.spotNumber=true

Postconditions

If self.spotNumber=true

then self.spotNumber.setSpaceStatus(false)

CLASS License Plate Detection

Invariants

Context:License-Plate-Detection

Inv:self.image

Context:License-Plate-Detection

Inv:getLicensePlate

Context:License-Plate-Detection

Inv:isExistingCustomer

Preconditions

Context:License-Plate-Detection

pre:self.isExistingCustomer

Postconditions

If carLeaving

then reservation.reservationEnd(getLicensePlate)

Else

Then reservation.reservationStart(getLicensePlate)

CLASS Parking Space

Invariants

context:parkingSpace

inv:self.parkingSpaceNumber

context:parkingSpace

inv:self.isSpaceEmpty

Preconditions

Context ParkingSpace

Pre:self.isSpaceEmpty(parkingSpaceNum)=true

Postconditions

Context: parkingSpace

post: self.setSpaceStatus(ParkingSpaceNum)=false

Section 9: System Architecture

Architectural Styles

In the ParKEZ Automated Garage system, utilizing the MVC architecture alongside Next.js 14, we adopted a hybrid approach that combines Event-Driven and Component-Based architectural styles. This approach is ideal for managing the web interface and automated garage functionalities.

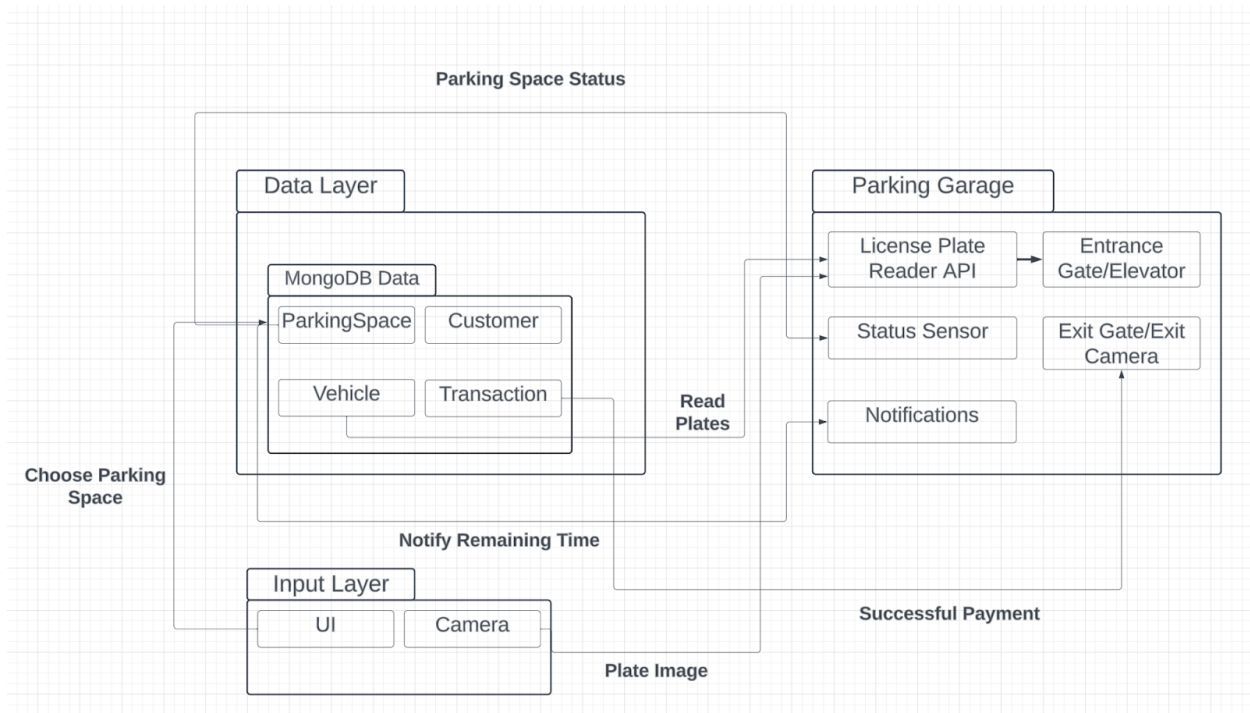
The Next.js 14 framework enriches the View component of the MVC architecture. It efficiently handles user interactions on the web interface, where actions such as mouse clicks for registration or making a reservation are events in the Event-Driven style. These user interactions trigger responses in the system, managed by the Controller, which coordinates the flow of data between the View and the Model. For instance, when a customer exits a parking space, this event leads the Controller to update the Model, reflecting the space's new availability in the system.

In the garage, sensor-related events like the LPR camera capturing a car's departure activate the Controller to instruct the Model to update the database, recording the exit time and modifying the customer's bill.

The system's design allows for the modular development of components, such as different displays in the garage. These components are managed within the MVC framework, demonstrating the flexibility and adaptability of the system.

Central to the architecture is the database, a pivotal element of the Model in the MVC framework. It is accessed and updated through interactions initiated by the View and managed by the Controller. This includes managing customer accounts, processing reservations, and monitoring parking space usage by both reserved and walk-in customers.

Identifying Subsystems



Mapping Subsystems to Hardware

Various components of the ParKEZ system work cohesively, with Next.js 14 and MVC architecture at the core, supported by MongoDB for data management and IoT devices for real-time data capture and processing.

1. Next.js 14 Application:

Hardware Component: Hosted on cloud or dedicated servers.

Description: The Next.js 14 application, acting as the View in the MVC architecture, runs on servers capable of handling web traffic and serving content efficiently. These servers should have the necessary computing power and memory to manage user requests, and a load balancer might be employed to distribute traffic for optimal performance.

2. MVC Architecture:

Hardware Component: Utilizes the same servers as the Next.js 14 application.

Description: The MVC architecture splits the application into three interconnected components - Model, View, and Controller. The Controller and Model, which handle business logic and data management, respectively, operate on the same servers as the Next.js application, ensuring seamless data processing and business logic implementation.

3. MongoDB Database:

Hardware Component: Hosted on dedicated database servers or cloud-based services.
Description: MongoDB, a NoSQL database, is used for storing and retrieving data. It requires robust servers with high storage capacity and redundancy to ensure data integrity and availability. Servers can be configured with RAID for data protection and to improve performance.

4. Visual User Interface:

Hardware Component: Runs on client devices (computers, smartphones, tablets).
Description: The user interface, built with Next.js 14, is rendered on client devices. These devices require adequate processing power and graphical capabilities to deliver a smooth user experience. The hardware choice depends on the end-users' devices, ensuring compatibility and responsiveness.

5. Camera and License Plate Detector:

Hardware Component: IoT devices installed in the garage.
Description: These specialized IoT devices are essential for capturing and processing vehicle data. They need to be robust and capable of transmitting data efficiently to the system. It's important these devices are connected to a local network or IoT gateway, allowing for seamless data transmission to the MongoDB database.

Persistent Data Storage

Most of the important data collected by ParkeZ will be stored inside a MongoDB database. Since the data is being stored on the MongoDB servers it will be persistent and mutable, allowing the users and administrators to update the data while keeping the data as incorruptible and persistent as possible. Adding to this as MongoDB is a cloud-based server system meaning that little to no data is kept on site, meaning that unless something happens to the MongoDB main servers, the data should exist indefinitely regardless of any corruption on the user end.

Network Protocol

To date, the current network protocol to be used in ParkEz's system is planned to be the widely used, mature TCP/IP protocol.

Global Control Flow

Execution Orderliness

The linear execution of the system-to-be event-driven designed purposely for the customer. The customer creates an account via the website, signs into the system using their username and password, clicks a button to make a reservation by selecting day, time, and location, pays for the reservation, submits the reservation, verifies the reservation via camera picks up the license plate, follows a guide to the elevator (if needed) and then to a parking lot, and exit the parking lot at the end of the reservation time. This linear execution is the same for every customer, including walk-in customers.

Time Dependency

The periodic system is dependent on real-time tracking. The customer selects the time of reservation, and the system-to-be calculates the hourly rate based on the reservation start time and end time using a real-time clock. Once a customer reserves a parking lot, the time block of the reservation for the lot is inaccessible to other customers until the end of the first user's reservation period in real-time. After the end-time reservation of the lot, the lot becomes unreserved and accessible to other customers. The system-to-be sends email notifications to customers in real-time and updates the customer on time remaining, overtime, and charges based on additional time on the reservation.

Concurrency

All the customers using ParKEZ have independent threads including system notifications, logins, signups, parking lot reservations, time reservations, and elevator guides despite sharing the same database/server with one another. MongoDB server handles the multiple threads that customers create while making sure customers' parking lot reservations are not in conflict time with one another and preventing a customer with one license plate from booking two slots thereby ensuring efficiency of resources.

Hardware Requirements

1. License Plate Camera Reader: 10 MB minimum for reading the license plates entering and exiting the garage.
2. LED/LCD Display: LED display in the elevator and inside the lot for guidance, and LCD outside the elevator for display.
3. Network Bandwidth: The system-to-be uses network bandwidth to connect with the garage visual user interface and other garage systems needing a network.
4. Server: The system-to-be uses a server to manage ParKEZ reservations' database and website hosting.
5. Touch Screen Tablet: The system-to-be uses a tablet as a visual user interface to interact with the user.
6. Hard-drive Disk and RAM: 10 GB minimum to store customers' cached information and data.

7. Sensors: The system-to-be uses sensors to track the status of the parking lots whether occupied or unoccupied.
8. Camera: The system-to-be uses camera functionalities to keep track of cars and security surveillance.

Section 10: Algorithms and Data Structures

Algorithms:

At this stage in the project no algorithms have been identified as necessary.

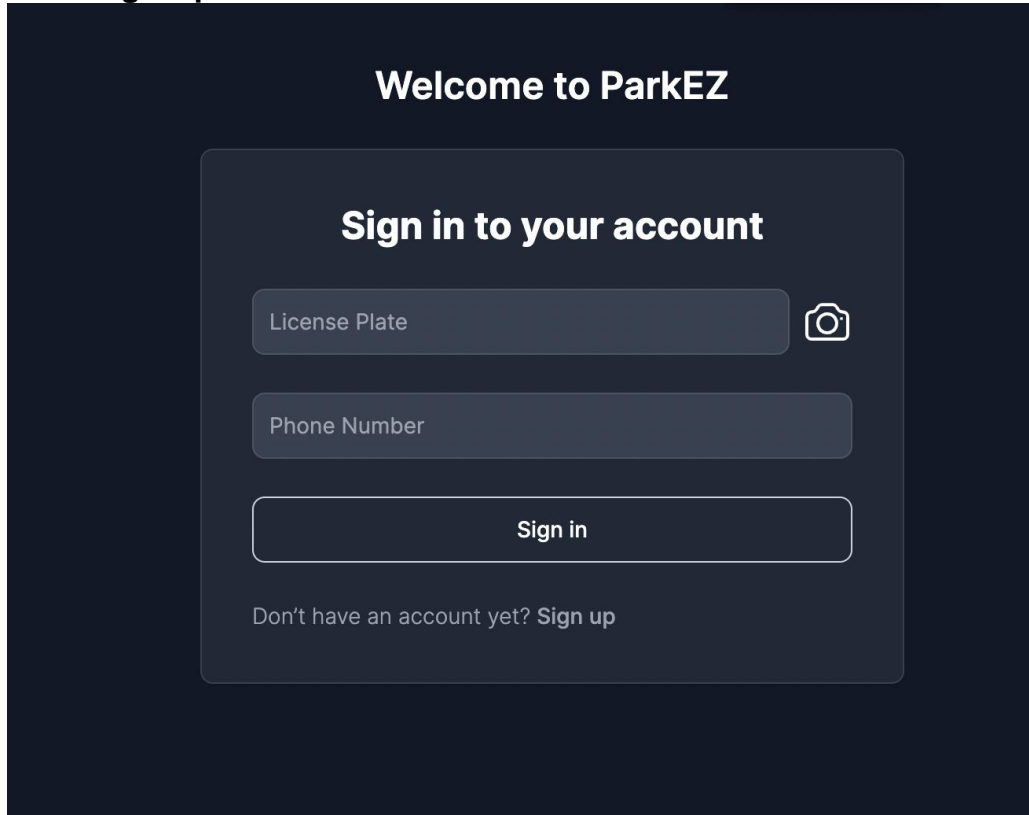
Data Structures:

The ParkEZ project does not use data structures. All necessary data is to be stored on a MongoDB database.

Section 11: User Interface Design and Implementation

There have been no changes to the original screen mock-ups that were developed for Report 1, up to this point. The original mock-ups have been included in this report below:

UC-1 Sign-Up

The image shows a dark-themed user interface for a service called 'ParkeZ'. At the top, the text 'Welcome to ParkeZ' is displayed in a white, sans-serif font. Below this, there is a central white box with rounded corners containing the heading 'Sign in to your account'. Inside this box, there are two input fields: the first is labeled 'License Plate' and has a camera icon to its right; the second is labeled 'Phone Number'. Below these fields is a white button with the text 'Sign in'. At the bottom of the white box, there is a link that says 'Don't have an account yet? Sign up'.

The user can log in with License plate and phone number, if there is no phone number connected with that license plate, the page will be redirected to /sign-up route.

Estimation: 1 text input + 2 clicks.

Create an account

First Name

Last Name

Email

or

Continue with Google

Continue with Facebook

Continue with Apple

Make

Model

Year

Color

Payment information

PayPal

Apple Pay

Pay with Am Ex

Pay with Visa

Pay with MasterCard

Create an account

Sign up route will be passed the parameters from /sign-in which are **license plate** and **phone number**.

Additional information needed from the end-user for registering in the /sign-up route will be **first name**, **last name**, **email**.

Alternatively, they can use O-Auth sign with providers such as **google**, **Facebook** or **apple**, where first name, last name, email, and profile image will be attached to the form data of the /sign-up route.

Next, the end-user will enter in vehicle **make**, **model**, **year**, and **color**. Finally, a **payment option** to keep on file, that includes credit card number and billing address, will be filled out by the user.

After completing these inputs, the user will be allowed to click the Create an Account button, which redirects them to the /sign-in route.

Estimation: 7 text inputs and 2 clicks or 3 text inputs + 3 clicks.

UC-2 Sign-In

If there exists a user with matching credentials of license plate and phone number, there will be a one- time code sent via SMS, where standard messaging rates will apply.

After authentication is granted, a link will be sent to the user that redirects them to the /parking-space route.

Estimation: 1 click if license plate is registered into system.

UC-3 Available Parking Space

In the /parking-space route there will be a map of available and occupied parking spaces

Estimation: 1 click to find parking space.

UC-4 Slot Choosing

In the /parking-space route there will be a display of available slots, a parking space map, and a date and time picker.

A time stamp for a parking space will start once the status sensor for a parking space changes from available to occupied.

Estimation: 1 click to find parking space.

The interface is divided into three main sections:

- Date and Time:** Includes a date input field (mm/dd/yyyy) and a time input field (--:-- --).
- Available parking spaces:** A list of four options:
 - 5 feet away (Level A - Space 12) with a green 'Go' button.
 - 20 feet away (Level A - Space 22) with a green 'Go' button.
 - 30 feet away (Level B - Space 11) with a green 'Go' button.
 - 40 feet away (Level C - Space 13) with a green 'Go' button.
- Parking Space Map:** Three maps labeled Level A, Level B, and Level C, each showing a grid of 12 green squares representing parking spaces.

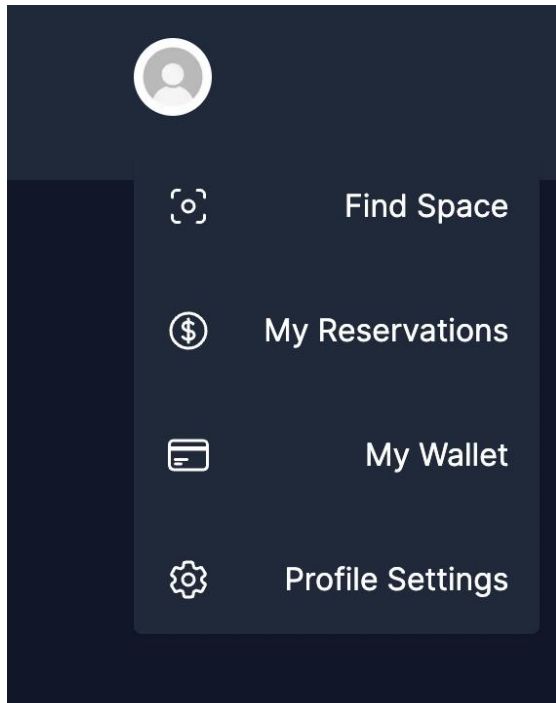
UC-5 Walk-in

Customers with no advanced reservation will stop at the user interface at the garage entrance. The user interface will ask for a registered user ID or reservation number. If a valid ID or reservation number is provided the user will be allowed to enter and be directed to the garage elevator. If no valid ID or reservation number is provided the display will show any available parking spaces or show no occupancies available if the ground floor is full. If parking spaces are available, the user will be prompted for name and payment/billing information. The license plate camera takes a picture of the customer's license plate and displays the license plate number on the screen. Once payment/billing information has been entered the user interface display will show the customer the selected parking space and they will be allowed to enter.

UC-6 Reservation Editing/Cancellations

After choosing a spot, the user can edit user parking space slot data, including time, date, parking space and be able to delete/cancel the reservation in the My Reservations, menu option.

Estimation: 2 clicks to get to reservations.



UC-7 Payments

Once a user leaves the parking space and the status for the sensor changes from occupied to available, payment is initiated from the user payment method. Upon leaving.

Estimation: 0 clicks for payment, just a confirmation message?

UC-8 Time Duration

Parking space time duration will depend on parking space status sensors.

Estimation: 0 clicks

UC-9 Time Update

Time updates will be sent through SMS updates.

Estimation: 0 clicks

UC-10 Reservation Extension

A link will be sent to the user for quick time extension capabilities.

Estimation: 1 click

UC-11 Arrival

When a user enters the garage, a camera will take a picture of the license plate and if the user is registered an SMS will be sent to the user for phone authentication and link to access Park EZ web app with user credentials.

Estimation: 1 click, one time code input, and another click.

UC-12 Departure

When a customer exits the garage, a picture will be taken of the license plate and vehicle and a thank you message will be sent.

UC-13 Information Privacy

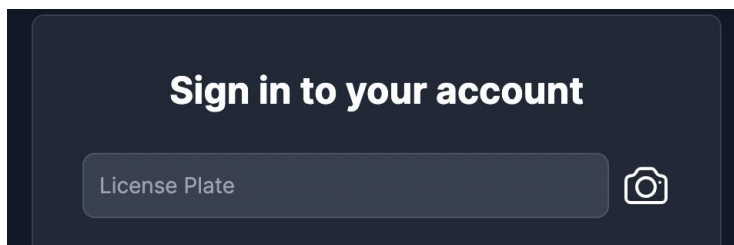
A password less, phone authenticated sign in will ensure all users are authorized and data will be better secured. Transactions will be encrypted and secured using Stripe API.

UC-14 Email/Phone Notifications

Send notifications by phone or email, which includes purchase and reservation confirmations, security codes, and reservation details.

UC-15 Scan Plate

Cameras will scan plate numbers and automatically display them in UI, or the user can upload a picture of the license plate, implemented through License Plate Scanner API.



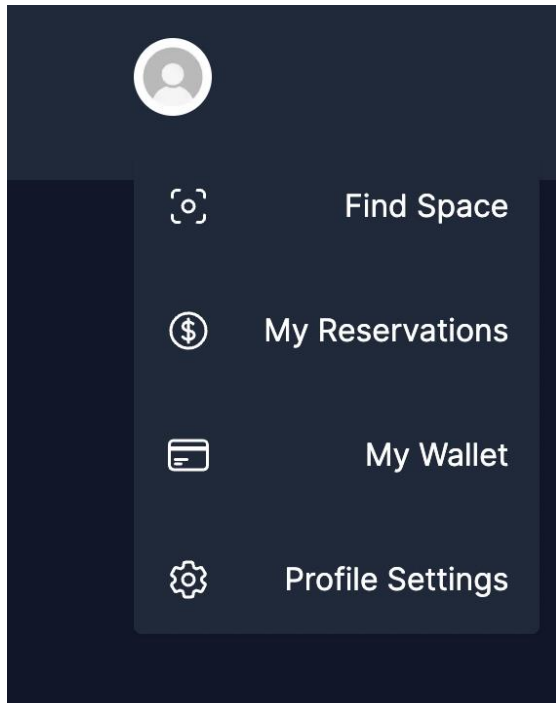
UC-16 Reservation

Customers can reserve a “one time” parking space or a monthly guaranteed reservation through /parking-space route.

Estimation: To reserve takes 3 clicks and 2 inputs for date and time.

UC-17 Payment Method

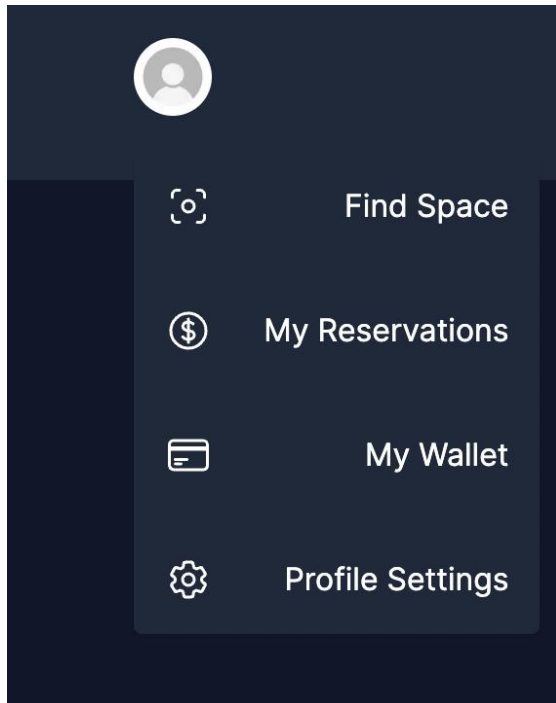
The user payment method should be saved in the My Wallet section of user profile for faster transaction implementation.



Estimation: To update payment method takes 2 clicks and text inputs.

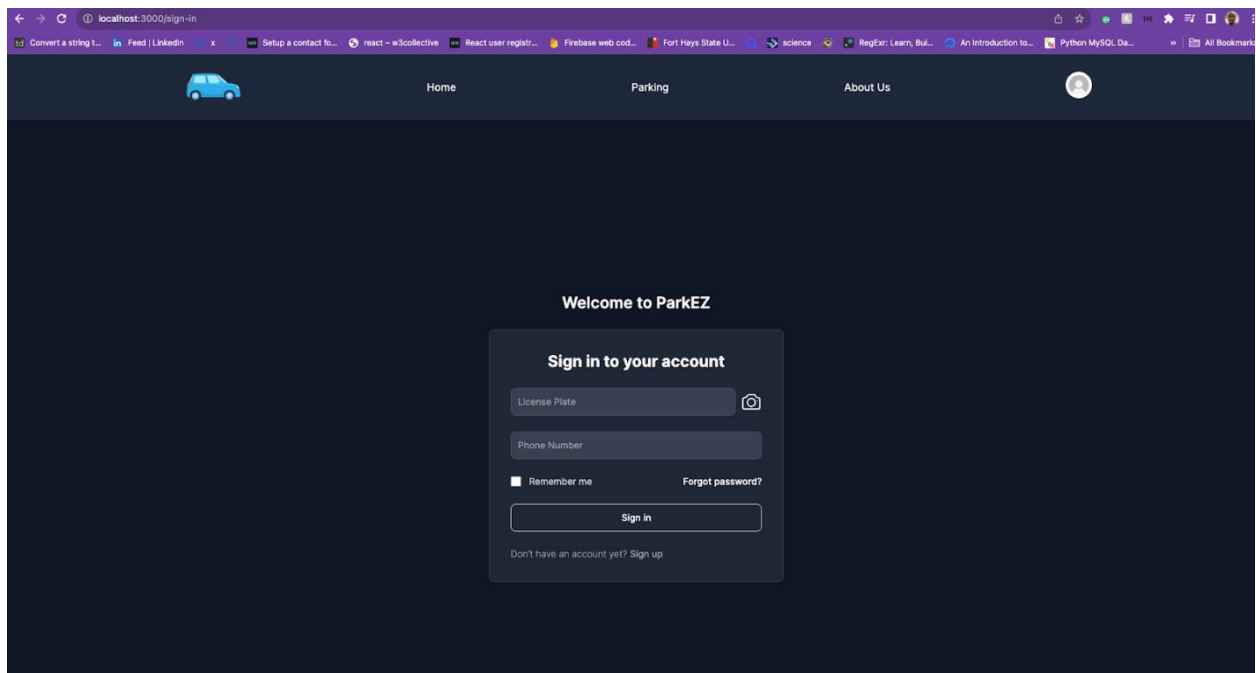
UC-18 Info Update

The user can update user details by clicking the profile picture icon, then clicking Profile Settings. The user changes whichever field needs to be updated, such as address, phone number, etc., and then clicks the Save Profile button.

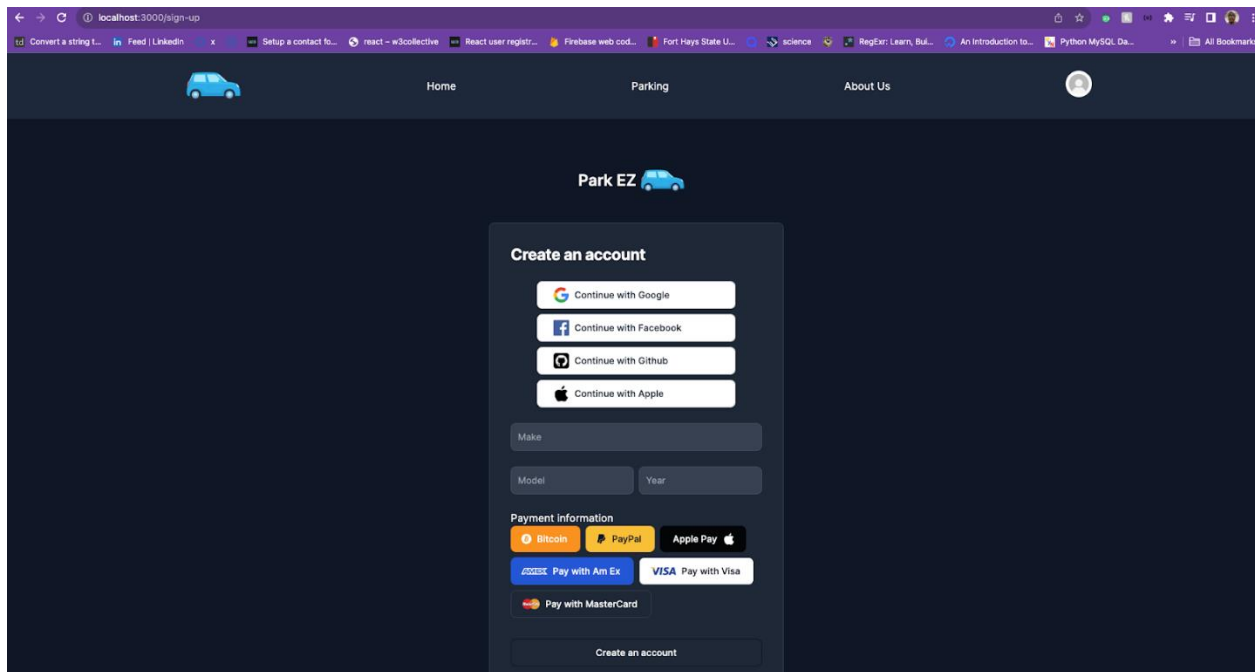


Estimation: 2 clicks to get to the Profile Settings section, however many input fields that need updating and 1 click to save changes.

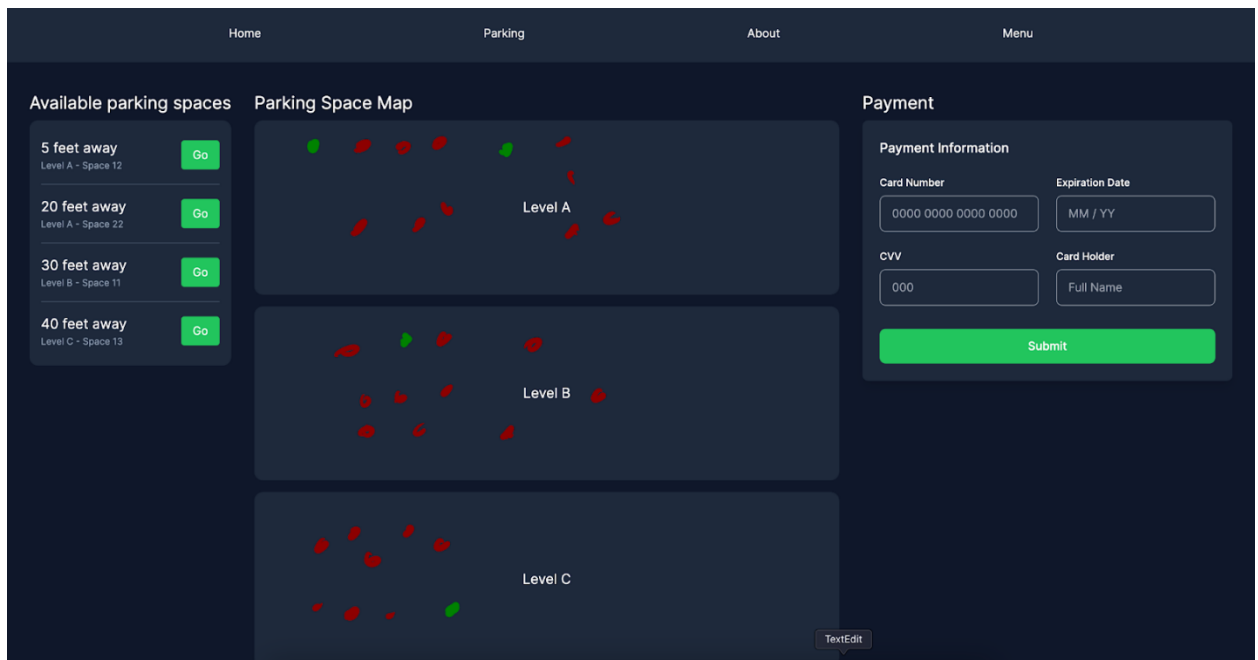
Login: /sign-in



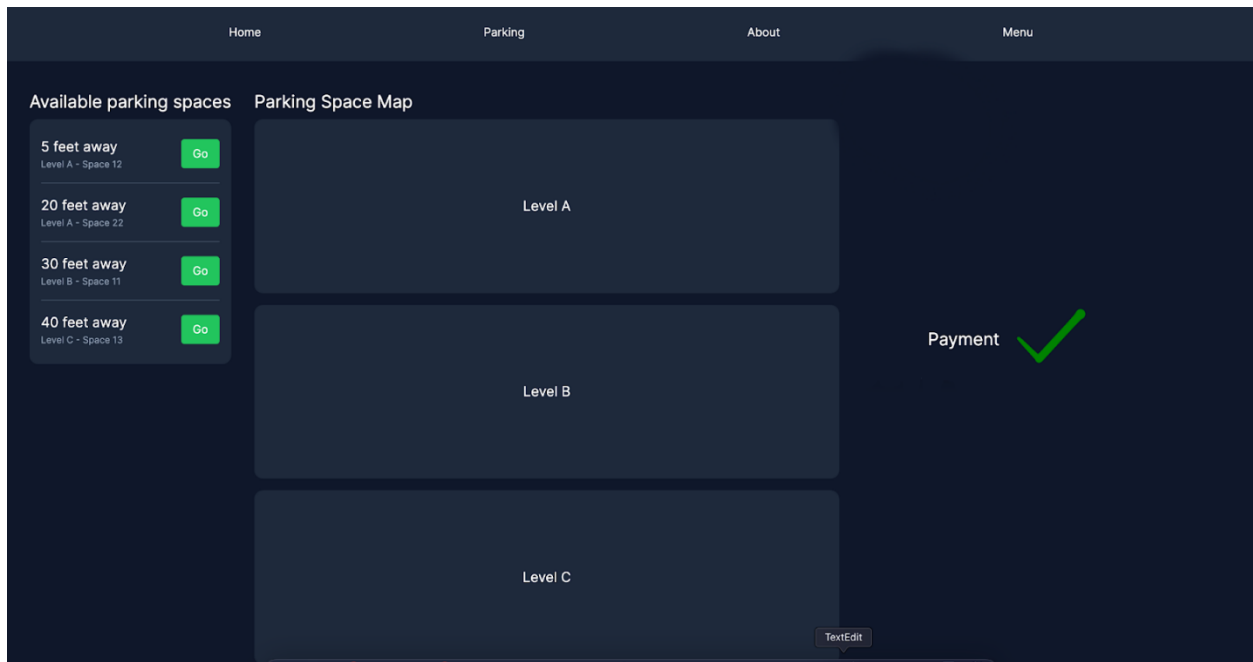
Registration: /sign-up



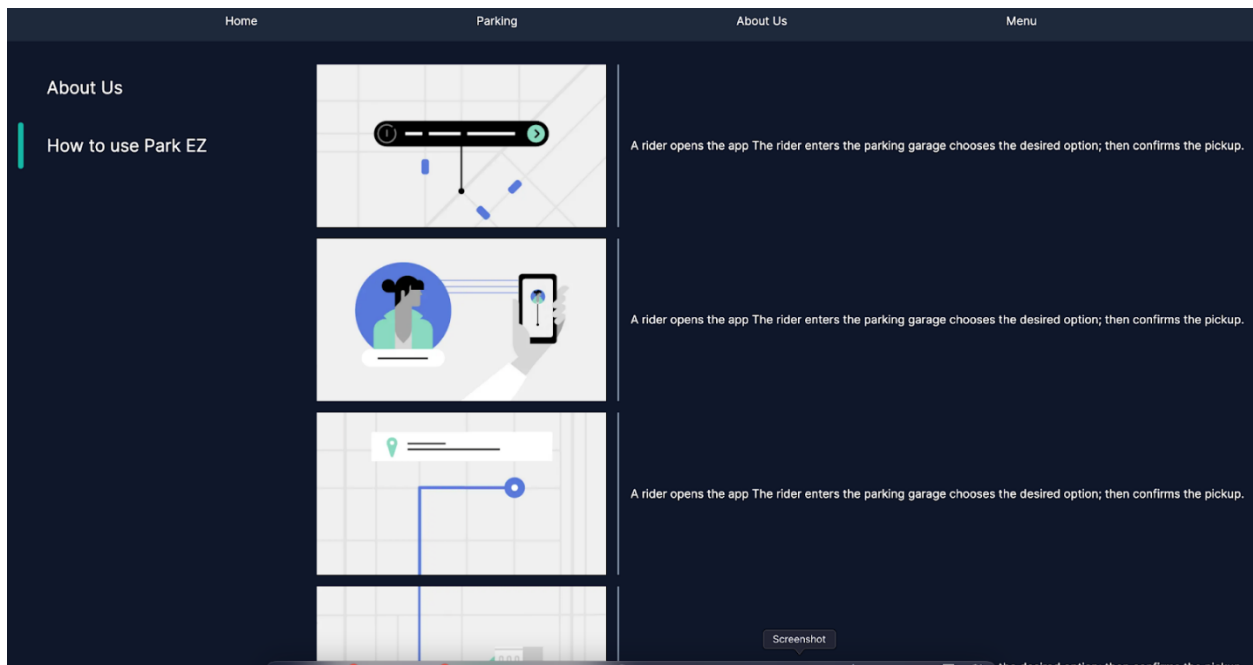
Parking: `/parking-space`



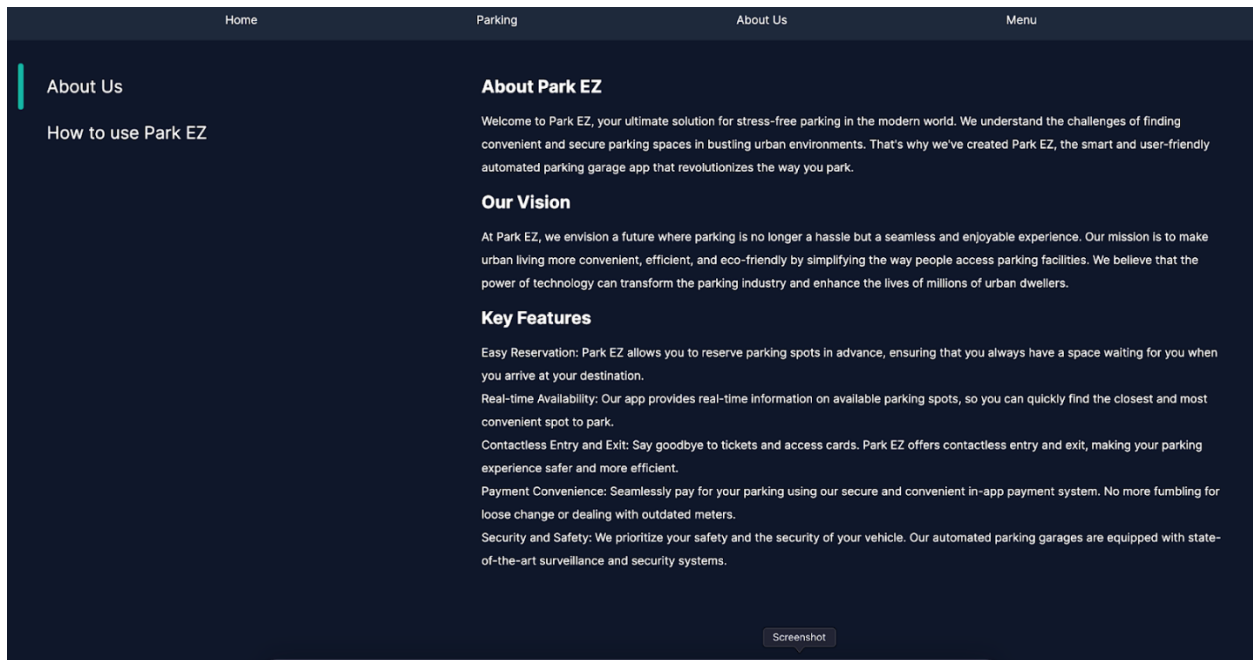
Payment Confirm: `/parking-space`



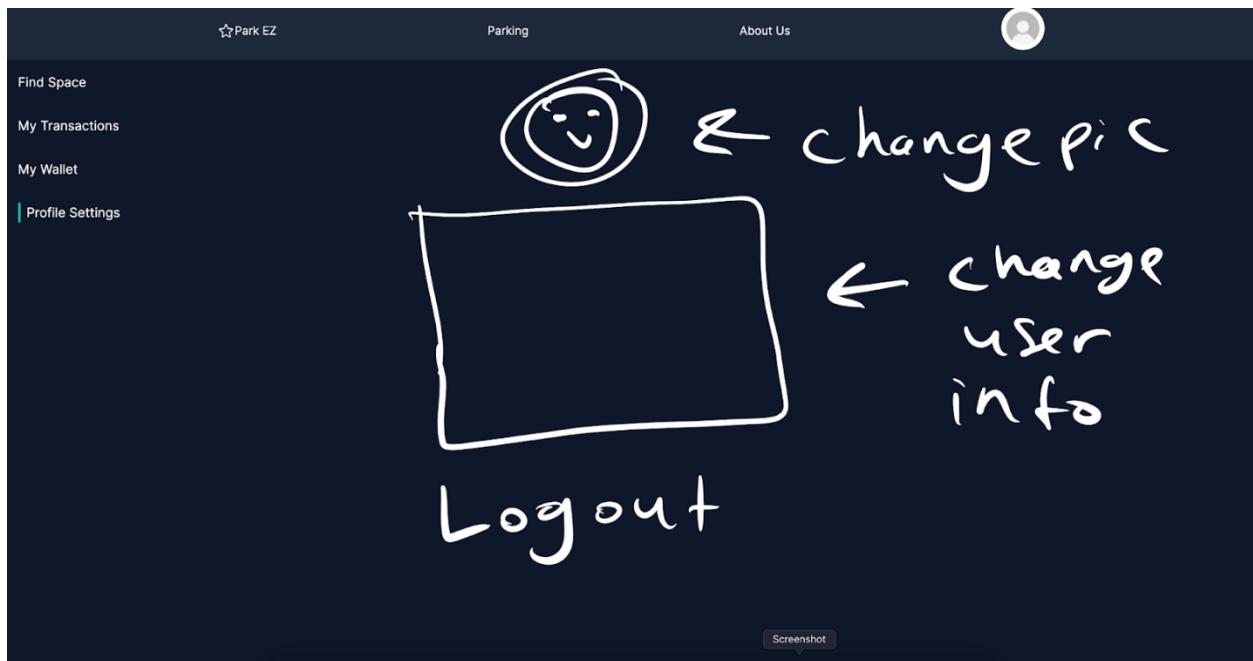
About: </about/how-to-use>

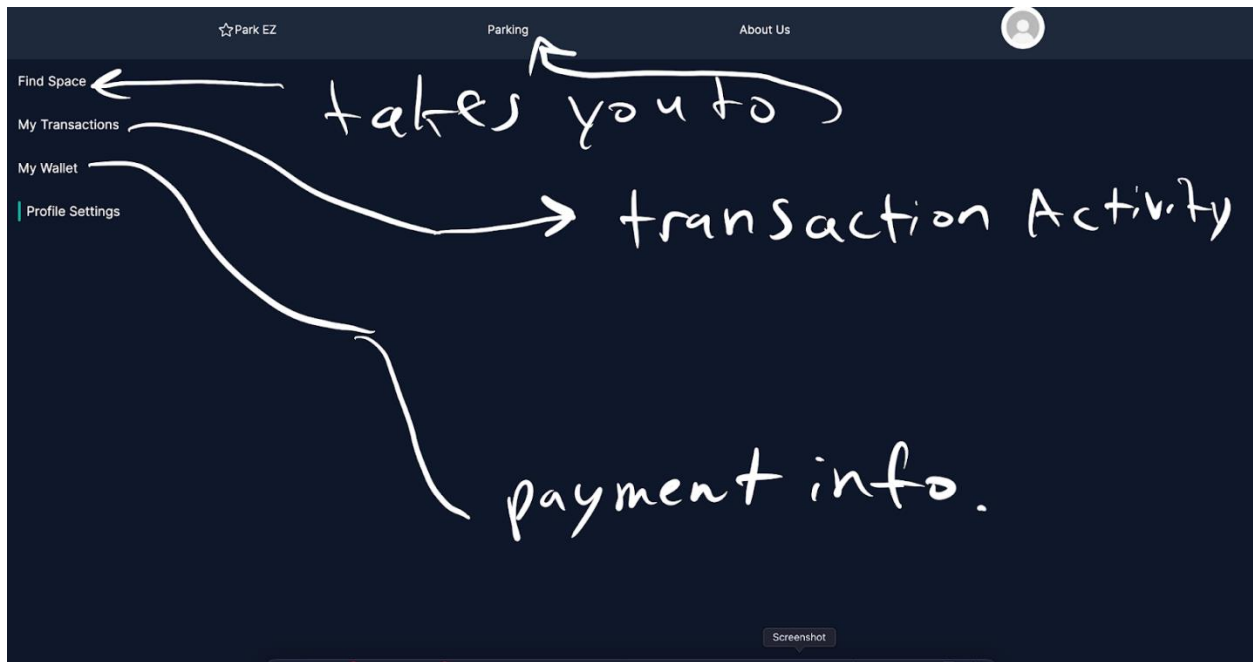


</about/about-us>



Menu: /user-menu





Section 12: Design of Tests

Use Case 1- Sign-in:

For testing the sign-in, we will begin by verifying that the database will return a correct truthy or falsy when the user attempts to login by inserting test users into the database and submitting sign-in entries that are both true and false. If the credentials do not match the database's entry, the use case should return an error to the user that the credentials do not match. To ensure that unique attributes are reliably stored and tracked, when the user enters the sign-up page, we will enter multiple test data entries that are both unique and already existing. In the case that the credentials already exist the use case should return an error alerting the user that the credentials are already in use.

Use Case 5- Walk-in:

Unregistered users should be able to make reservations with the help of managers using lot number, license plate, personal information, payment information, and phone number when making a reservation as a walk-in. Pictures of the license plate would be taken with cameras and sent over to the database to confirm if the user plate matches any plate in the database to avoid duplicates. The testing also involves checking to see if unregistered users can select the available slots they want, however, if there are no available slots, the walk-in would not be able to enter the garage. The testing for unavailability of slots would not be considered at this demo stage. After one manager helps the user sign up/log in, another manager guides the user through the lot layout to the reserved spot. Another test would be designed to test if the users could exit the elevator and log out after the end of the reservation.

Use Case 6 - Reservation Editing/Cancellation:

To start, we will assess the system's ability to handle various input scenarios, including incorrect dates, invalid data, and incomplete required information, to ensure robust error handling. Additionally, we will run tests to confirm that customers can successfully edit and cancel their reservations. This involves checking if the website accurately displays reservation details, allows for extensions with updated pricing, and enables smooth cancellations. We will also examine the system's double booking prevention measures to guarantee customers cannot accidentally reserve the same parking spot twice. Throughout the testing process, we will assess the database's ability to reflect reservation changes and ensure that customers receive email confirmations for their actions.

Use Case 07 - Payment:

Test Case 1: Verify Payment Options

Test Case Description:

Verify that the Park EZ app provides multiple payment options.

Test Steps:

- Open the Park EZ app.
- Navigate to the payment section.
- Check if there are multiple payment options available, such as credit card, debit card, PayPal etc.
- Expected Result:
- The app should offer multiple payment options for user convenience.

Test Case 2: Payment Validation

Test Case Description:

Verify that the Park EZ app validates payment information correctly.

Test Steps:

- Enter valid payment details (e.g., credit card number, expiration date, CVV).
- Attempt to make a payment.
- Enter invalid payment details (e.g., incorrect card number, expired card, incorrect CVV).
- Attempt to make a payment.
- Verify the response in each case.

Expected Result:

Valid payment details should result in a successful payment.

Invalid payment details should result in an error message.

Test Case 3: Payment Confirmation

Test Case Description:

Verify that the Park EZ app displays a payment confirmation message.

Test Steps:

- Make a successful payment.
- Check for a confirmation message after the payment is completed.

Expected Result:

The app should display a confirmation message, including the transaction details.

Test Case 4: Refund Process

Test Case Description:

Verify that the Park EZ app allows users to request refunds.

Test Steps:

- Attempt to request a refund for a previously made payment.
- Provide a reason for the refund request.
- Check the refund status.

Expected Result:

The app should allow users to request refunds and provide information on the status of the refund request.

Test Case 5: Currency and Amount Display

Test Case Description:

Verify that the Park EZ app displays the currency and payment amount accurately.

Test Steps:

- Make a payment.
- Check that the currency symbol is displayed correctly.
- Verify that the payment amount matches the fee for the parking duration.

Expected Result:

The app should display the correct currency symbol and payment amount for the selected parking duration.

Test Case 6: Payment History

Test Case Description:

Verify that the Park EZ app maintains a payment history.

Test Steps:

- Make multiple payments for parking.
- Access the payment history section.
- Verify that all payments are listed with details.

Expected Result:

The app should maintain a payment history with accurate transaction details.

Test Case 7: Security Testing

Test Case Description:

Verify the security of the payment process.

Test Steps:

- Attempt to make a payment with incorrect payment details.
- Attempt to manipulate the payment request (e.g., intercepting requests with a proxy tool).
- Attempt to access payment data without authorization.

Expected Result:

The app should have robust security measures to prevent unauthorized payments or access to sensitive payment data.

Test Case 8: Network Interruption Handling

Test Case Description:

Verify how the Park EZ app handles network interruptions during payment.

Test Steps:

- Make a payment.
- Disable the internet connection during the payment process.
- Attempt to complete the payment.

Expected Result:

The app should handle network interruptions gracefully and provide an appropriate error message or option to retry.

Use Case 16 - Reservation:

A user should be able to enter the reservation web page, make either a standard or ongoing reservation, and have the reservation successfully made and logged to the database. The Customer should receive confirmation that the reservation has been made in the form of a receipt. The User should then be able to go to the parking garage at the time of the reservation and be able to use the reservation.

If the user entered data is incorrect the system should return a message informing the Customer that they entered incorrect information. If the reservation for some reason does not go through the customer should receive a notice that the reservation did not happen because of some system error, such as a time out, parking spot conflict, or the customer entered clashing data.

Test Cases:

In order to test the system, we will run the reservation system through a series of tests to ensure that the System runs as intended and will not give faulty reservations.

1. First we will perform a test checking the input acceptance values with data values that are outside the norm or not acceptable such as incorrect dates, faulty data, or not entering all required data.
2. Next, we will also run tests to ensure that all correct reservations are correctly registered with the database and create successful usable reservations.
3. Another area we will also test is the double-booking prevention measures making sure that customers cannot accidentally double book a parking spot.
4. We will also test the repeated nature of the ongoing reservations to ensure that the parking spaces are reserved at the requested times, and that outside of those times the parking space is available for reservation.

Use Case 22 - Elevator Interaction:

A user should be able to enter the elevator, have the license plate scanned and, if the license plate matches a reservation for the current date and time, the elevator display will show the reservation along with the reserved parking space and parking garage floor where the space is located. The elevator will move to the desired floor, the display will notify the user to exit the elevator and once the user has exited the elevator will return to the ground floor and a waiting condition.

If the user's license plate is scanned and does not match a reservation the elevator display will prompt the user to enter the reservation confirmation number using the elevator console or to press cancel to exit. The user will then have the opportunity to enter the number using the console. If the correct number is entered, then the elevator display will show the reservation along with the reserved parking space and parking garage floor where the space is located. The elevator will move to the desired floor, the display will notify the user to exit the elevator and once the user has exited the elevator will return to the ground floor and a waiting condition.

If the user's license plate number does not match a reservation and the user cannot supply a valid reservation confirmation number or the user selects cancel on the elevator console, the elevator display will prompt the user to exit the garage and obtain a walk-in parking space on the ground floor.

Test Coverage: We will have to simulate a sensor in the elevator to sense when a car is pulled in. The sensor will be toggled true for on (a car present) and false for off (no car present). Also, we will use license plate images and an LPR API to simulate an LPR camera system. All tests will require use of the database. Once the sensor is activated and stays activated for a few seconds there are 3 scenarios that need to be tested. 1) The first scenario is that of a registered customer with a reservation and the vehicle license plate matches the user's reservation. 2) The second scenario is that of

a registered customer with a reservation, but the vehicle license plate number does not match the reservation. 3) The third scenario is that of a registered customer or walk-in customer, that does not have a license plate number matching a reservation and enters an invalid reservation confirmation number max times. 4) The fourth scenario represents a walk-in customer that cancels the elevator interaction.

1. To test scenario one, an image of a license plate number that matches a reservation will be sent to the LPR API. The image will be processed, and a query will be sent to the MongoDB database. The query will have to be successful for this test. A successful query returns the user's name, reservation confirmation number, reserved parking space number, reserved parking space floor and the duration of the reservation. That information will be displayed on the screen to simulate the elevator display. The elevator position will be represented by an integer and will be changed to the floor number that the reserved parking space is located on. Once the elevator shows the correct location a message to exit the elevator will be printed on the screen. The elevator sensor will be toggled to false to show the vehicle has exited. Once the sensor shows false the elevator will be returned the value of 1 to represent a return to the ground floor. The screen buffer will be cleared.
2. To test scenario two, an image of a license plate number that does not match a reservation will be sent to the LPR API. The image will be processed, and a query will be sent to the MongoDB database. The query will have to be unsuccessful for this test. An unsuccessful query will prompt a message to the screen asking for user input in the form of a reservation confirmation number or to press the "cancel" button to exit. A matching reservation number will be entered. A new query will be sent to the database using the confirmation number. For this test to be successful it will need to match a reservation database record for the current date and time period. If successful the query returns the user's name, reservation confirmation number, reserved parking space number, reserved parking space floor and duration of the reservation. That information will be displayed on the screen to simulate the elevator display. The elevator position will be represented by an integer and will be changed to the floor number that the reserved parking space is located on. Once the elevator shows the correct location a message to exit the elevator will be printed on the screen. The elevator sensor will be toggled to false to show the vehicle has exited. Once the sensor shows false the elevator will be returned the value of 1 to represent a return to the ground floor. The screen buffer will be cleared.
3. To test scenario three, an image of a license plate number that does not match a reservation will be sent to the LPR API. The image will be processed, and a query will be sent to the MongoDB database. The query will have to be unsuccessful for this test. An unsuccessful query will prompt a message to the screen asking for user input in the form of a reservation confirmation number or to press the "cancel" button to exit. An invalid reservation confirmation number will be entered. A new query will

be sent to the database using the invalid confirmation number. For this test to be successful the database query will need to fail, meaning a matching reservation, date/time and reservation confirmation number is not found. A message will be printed on the screen to re-enter the reservation confirmation number. This will be a loop with a maximum of three tries to enter a valid confirmation number. The same invalid reservation confirmation will be entered until the max tries equals 3 and then a message will be printed to the screen prompting the user to exit the garage and obtain a walk-in parking space on the ground floor. The elevator sensor will be toggled to false to show the vehicle has exited. Once the sensor shows false the elevator will be returned the value of 1 to represent a return to the ground floor. The screen buffer will be cleared.

4. To test scenario four, an image of a license plate number that does not match a reservation will be sent to the LPR API. The image will be processed, and a query will be sent to the MongoDB database. The query will have to be unsuccessful for this test. An unsuccessful query will prompt a message to the screen asking for user input in the form of a reservation confirmation number or to press the “cancel” button to exit. A cancel command is executed. A message will be printed to the screen prompting the user to exit the garage and obtain a walk-in parking space on the ground floor. The elevator sensor will be toggled to false to show the vehicle has exited. Once the sensor shows false the elevator will be returned the value of 1 to represent a return to the ground floor. The screen buffer will be cleared.

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

History of Work:

The deadlines for the different parts of this project revolved around the due dates on blackboard and the CSCI441 syllabus. These deadlines were much easier to hit at the beginning of the project during the information gathering and analysis stage of the project. The deadlines got increasingly more difficult to achieve as we got into the design and coding parts of this project. The same was true of our milestones. We were consistently on target for early milestones such as Reports 1 and 2, but got behind schedule for Demo 1.

Key Accomplishments:

- Report 1
- Report 2
- ParkeEZ website structure implementation
- MongoDB database setup
- License plate reader implementation
- Customer sign-up on the website
- Customer sign-in on the website
- Successful customer reservation on the website
- Successful entry of payment information
- Demo 1

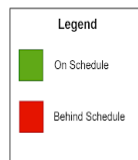
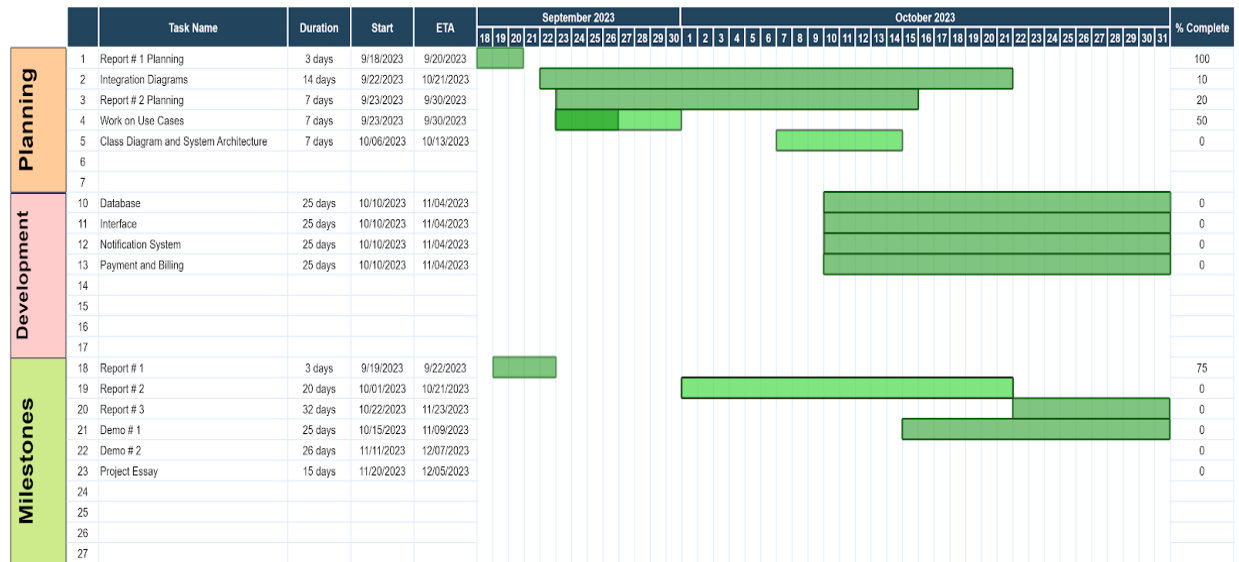
Current Status:

At this time, we have a ParkeEZ website that allows customer registration, sign-in and reservation. Customer and reservation information is saved in the MongoDB database. When a parking space is reserved, it shows red on the website parking space map and green otherwise. When logging in on the website we have to provide an image of a license plate to simulate the LPR camera taking an image of the license plate. We also have a working form for payment information using STRIPE.

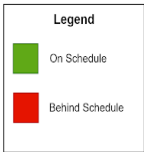
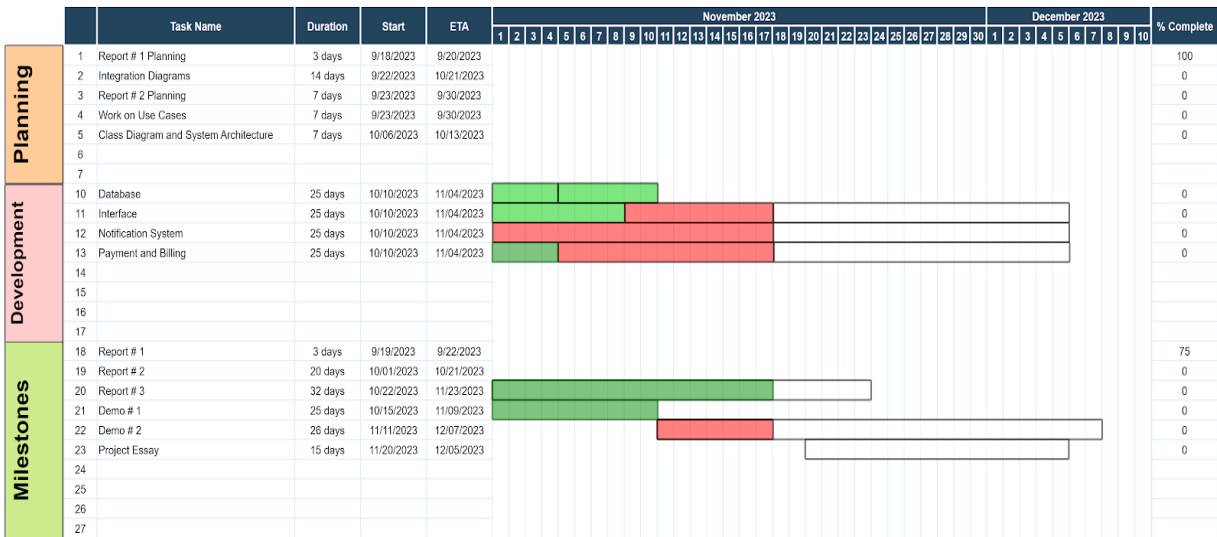
Future Work:

We need to continue to fine tune the website in anticipation of Demo 2. We also need to figure out the notification system to send email/SMS notifications for reservation confirmations, reservation changes / cancellations, parking space reservation time running out / expired, billing and payment confirmation. We also need to add a representation of the elevator and the interactions a customer can have with it on our website. Lastly, we need to streamline our website code / files for easy readability and optimization.

ParKEZ Gantt Chart



ParkEZ Gantt Chart



Product Ownership:

Breakdown of Teams	Team Member Assignment	Coordination Activities
Team Pair #1: Benjamin Bylsma Mikael Mikaelian	Benjamin Bylsma <ul style="list-style-type: none"> Customer registration and profile management. Mikael Mikaelian <ul style="list-style-type: none"> Payment processing and billing. 	Functionalities: Customer registration and profile management Qualitative Property: Develop an intuitive and user-friendly online registration process that securely stores customer data and reservation data in a database. Treatment: Address the problem of user registration complexity and ensure a seamless onboarding experience.

<p>Team Pair # 2: Christopher Smith Adrian Elgin</p>	<p>Christopher Smith</p> <ul style="list-style-type: none"> Real-time parking space availability updates. <p>Adrian Elgin</p> <ul style="list-style-type: none"> Reservation system with confirmation. 	<p>Functionalities: Real-time parking space availability updates by integrating parking space sensors, license plate cameras and customer check-in software.</p> <p>Qualitative Property: Ensure system performance to display availability within Seconds.</p> <p>Treatment: Alleviate the problem of customer frustration due to inefficient parking space searches.</p>
<p>Team Pair # 3: Phongsavanh Mongkhonvilay Geoffrey Sarpong</p>	<p>Phongsavanh Mongkhonvilay</p> <ul style="list-style-type: none"> Integration with security and surveillance. <p>Geoffrey Sarpong</p> <ul style="list-style-type: none"> Automated entry and exit for vehicles. 	<p>Functionalities: Automated entry and exit for vehicles using license plate recognition camera system.</p> <p>Qualitative Property: Develop and evaluate a user-friendly interface for the entry and exit process.</p> <p>Treatment: Resolve the issue of congestion and delays at entry and exit points.</p> <p>Additionally, we will collectively work on payment processing, billing, and the reservation system to enhance overall system efficiency.</p>

Section 14: References

Marsic, I. (2009). *Software Engineering*. Ivan Marsic.

https://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf.

Marsic, I. (2009). *Software Engineering Course Project Parking Garage/Lot*. Ivan Marsic.

<https://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/ParkingLot.pdf>

Gantt Chart and Use Case Diagram Design: <https://app.diagrams.net/>

Tran, L., Nguyen, K., Choudhury, S., Ngo, T., Nguyen, D., Xiao, Z., Patel, N. (2019).

Blockchain And Docker Assisted Secure Automated Parking Garage System. Ivan Marsic.

<https://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2019f-g4-report3.pdf>