



OCTOBER 22, 2023

## REPORT 2

PARKEZ AUTOMATED PARKING GARAGE

GROUP 4



## **Report 2**

CSCI 441 Software Engineering

Group # 4:

### **ParKEZ Automated Parking Garage**

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

10/22/2023

Team Members:

Benjamin Bylsma

Adrian Elgin

Mikael Mikaelian

Phongsavanh Mongkhonvilay

Geoffrey Sarpong

Christopher Smith (Team Leader)

# Table of Contents

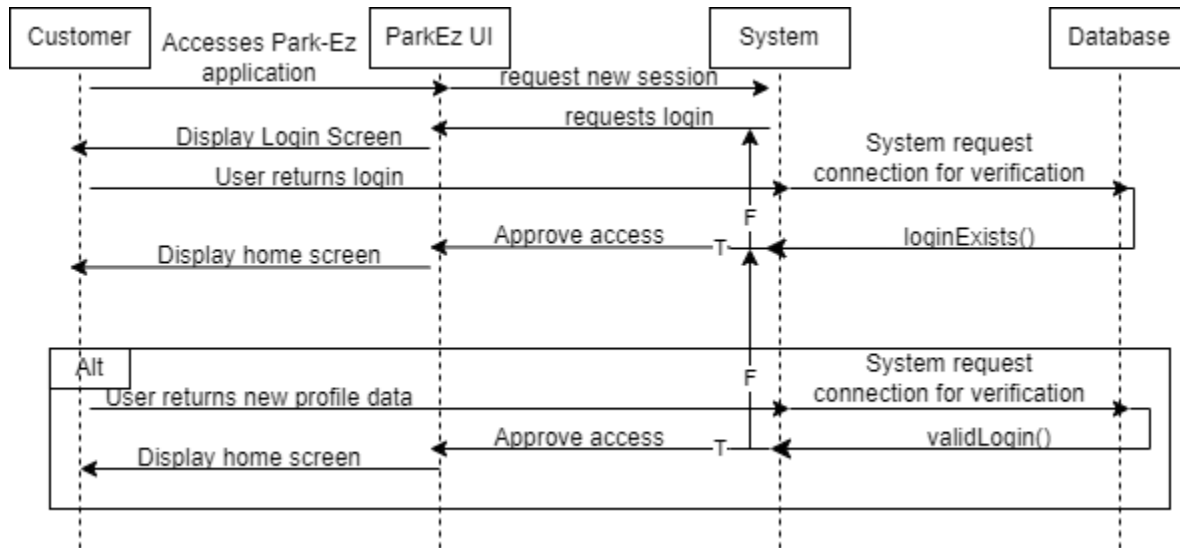
Table of Contents	2
Contribution Breakdown	3
Section 1: Interaction Diagrams	4
Section 2: Class Diagram	15
Data Types and Operation Signatures	15
Section 3: System Architecture	21
Architectural Styles	21
Identifying Subsystems	22
Mapping Subsystems to Hardware	22
Persistent Data Storage	23
Network Protocol	23
Global Control Flow	23
Hardware Requirements	24
Section 4: Algorithms and Data Structures	25
Algorithms:	25
Data Structures:	25
Section 5: User Interface and Design Implementation	26
Section 6: Design of Tests	37
Use Case 1- Sign-in:	37
Use Case 5- Walk-in:	37
Use Case 6 - Reservation Editing/Cancellation:	37
Use Case 07 – Payment:	38
Use Case 16 – Reservation:	40
Use Case 22 - Elevator Interaction:	41
Project Management	44
Merging the Contributions from Individual Team Members:	44
Progress Coordination and Progress Report:	44
Plan to Work:	44
Product Ownership:	46
References	48

## 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%
Interaction Diagrams	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Class Diagram	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
System Architecture	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 and Design Implementation	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%
Project Management	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: Interaction Diagrams

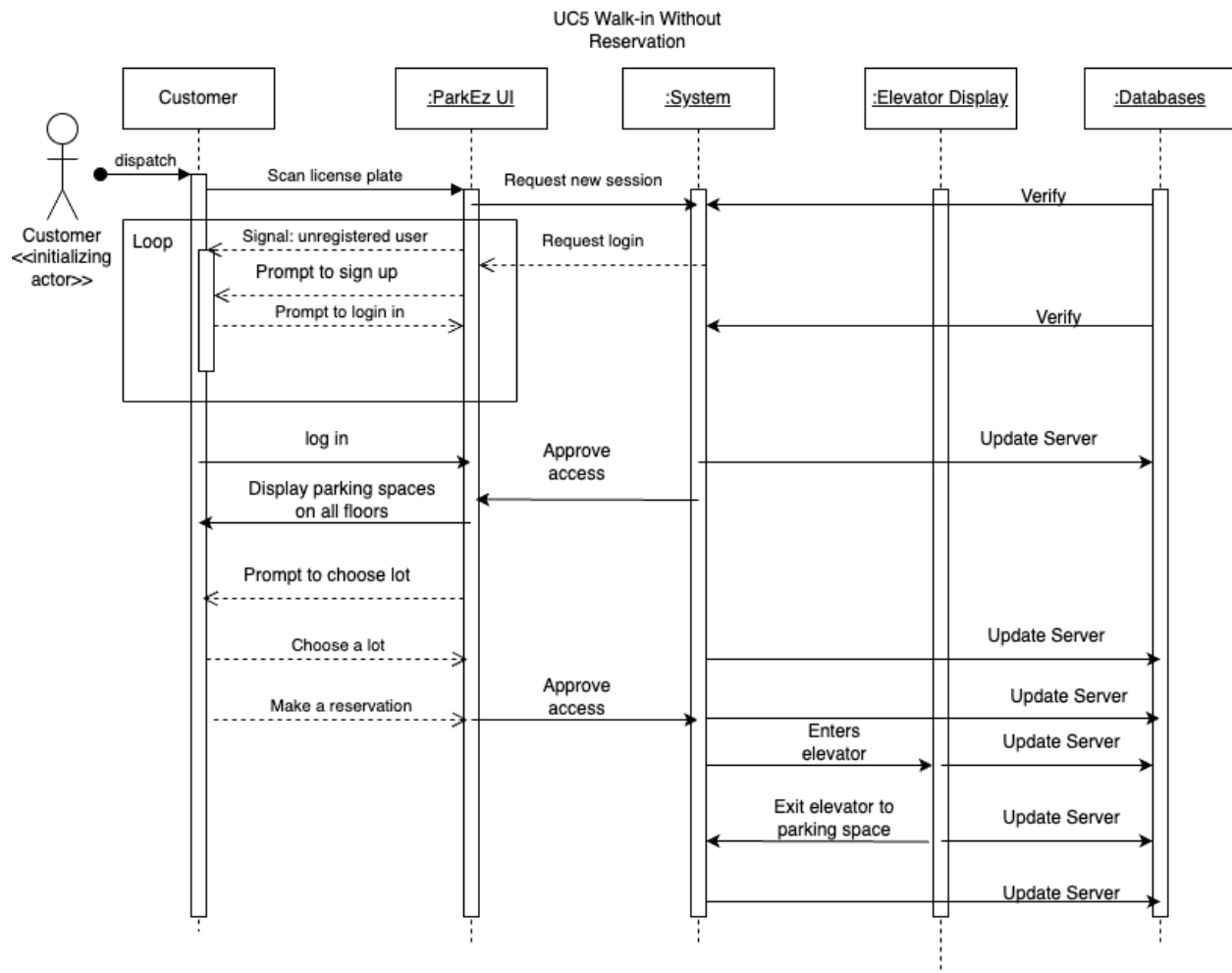
### Use Case 1: Sign-in



When the customer accesses the ParkEz application, they are greeted with a login screen. This reflects a typical login screen with both an existing customer login or an option to sign up for a new account. The customer will enter their existing credentials (with the new user option being an alternate route) and the system will check to see if the credentials exist. If the password does not match, the system will prompt the customer with an error message stating that the password does not match with the username they entered. If the username does not exist, the system will prompt the customer with an error message stating that no account under that username exists.

If the customer chooses to sign up for a new account, they will be directed to another page similar to the login page that will ask them for a username and password. The system will then check the database to ensure that the username is unique. If there is already a username registered to the name they entered, the system will prompt the customer with an error message stating that the username is already in use and to enter a new one. (the usernames, for simplicity, will be the customers email address) If the credentials pass the uniqueness check, the user will then be taken to the homepage with prompts to add their license plate and driver's license before they can sign up for a registration.

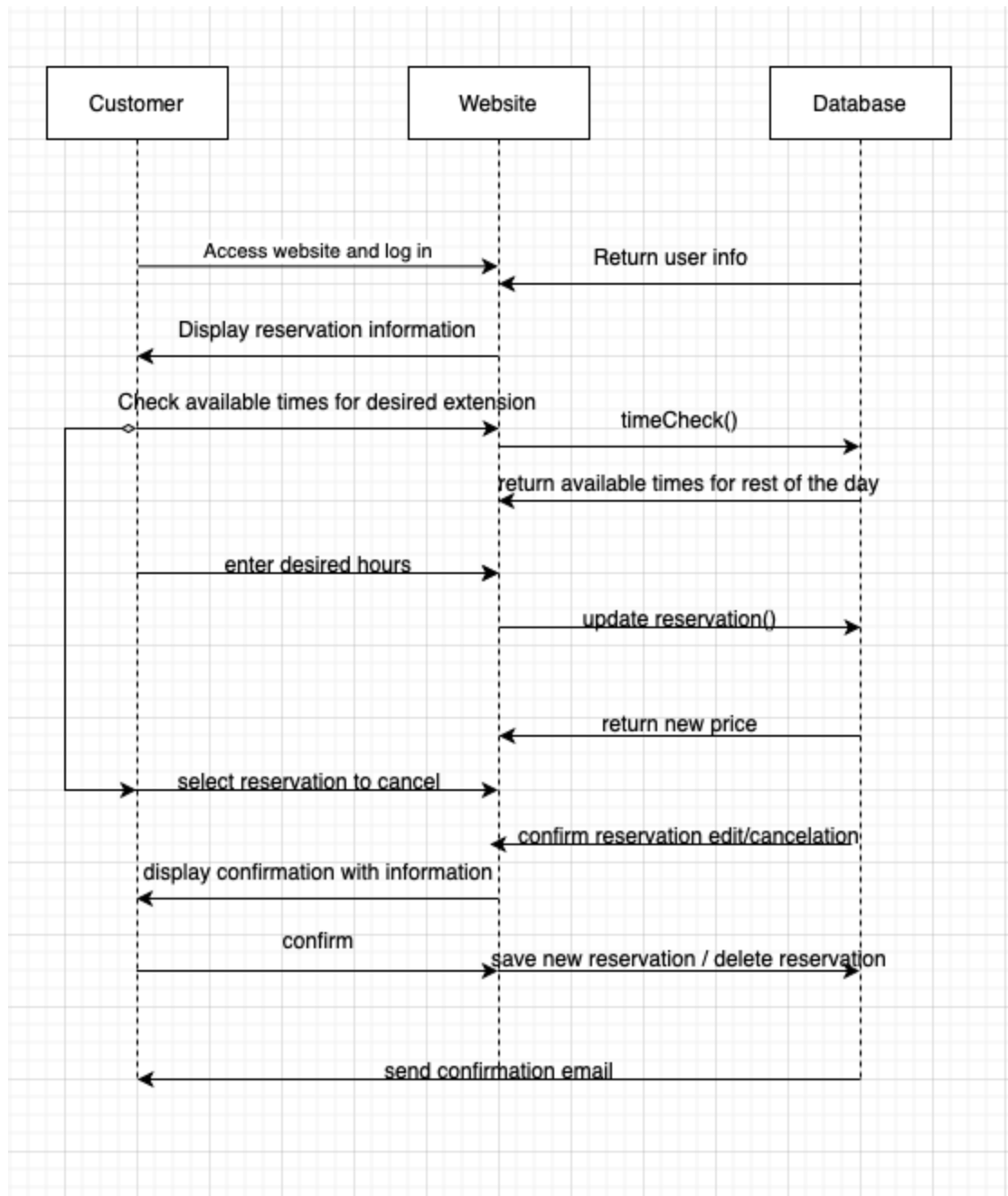
### Use Case 5: Walk-in



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

The user accesses the ParkeZ garage via their scanned license plate. A new session is requested, and the user is prompted to sign up on the website if he/she does not have a login account already. After logging in, the system-to-be approves access to visual interface display parking spaces available on all floors, updates the server with the user information, and prompts the user to choose a lot to make a reservation via visual user interface. The system-to-be approves the reservation and guides the user to the elevator while updating the server on the user's thread. The user exits the elevator and follows guiding information to his parking lot to park at his/her reserved parking lot. At the end of parking lot reservation, the user leaves his/her lot and follows guiding information to exit the elevator and the parking garage. The system-to-be updates the server on the user thread.

## Use Case 6: Reservation Editing/Cancelation

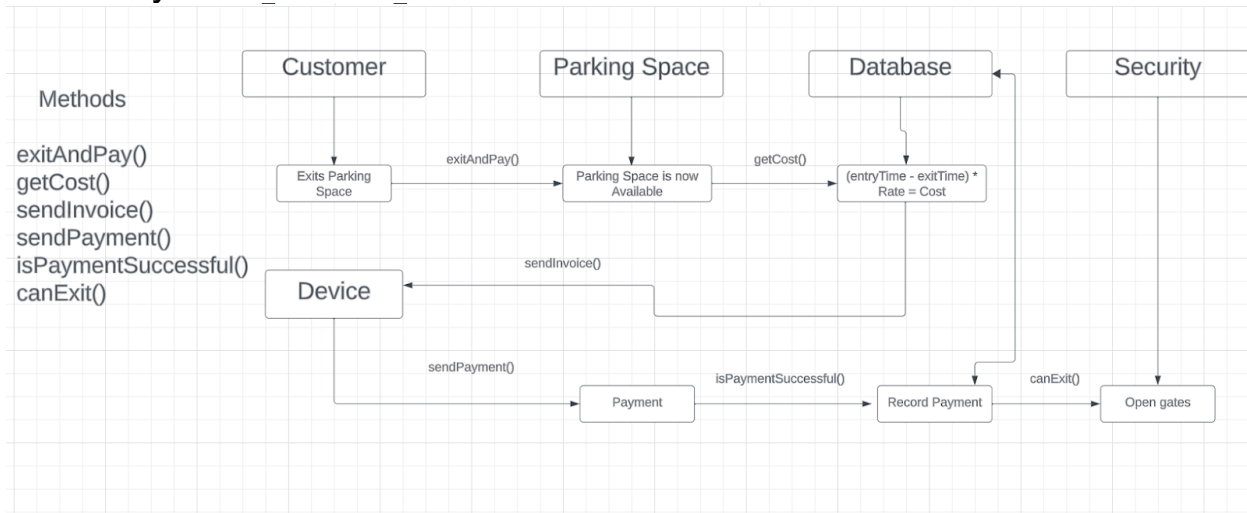


The customer accesses login functions via the ParKEZ website, and the database returns the user information, displaying reservation information and available time for the desired extension. The time check on the databases returns the available time for the rest of the day, and the customer selects the desired time. The website updates the reservation and returns a new price for which the customer can select a reservation to cancel, and the database will confirm the reservation edit and cancellation. Then, the website will display confirmation with information for the customer to confirm, and the website will save the new reservation or delete the

reservation on the database. The database will finally send email confirmation to the customer.

## Use Case 7: Payment

### Payments



As Customer exits parking space, the parking space sensor will change from occupied to available, This will trigger a function called `exitAndPay()`. The initialization of `exitAndPay()`, will query the database for the current customers data occupying that parking space.

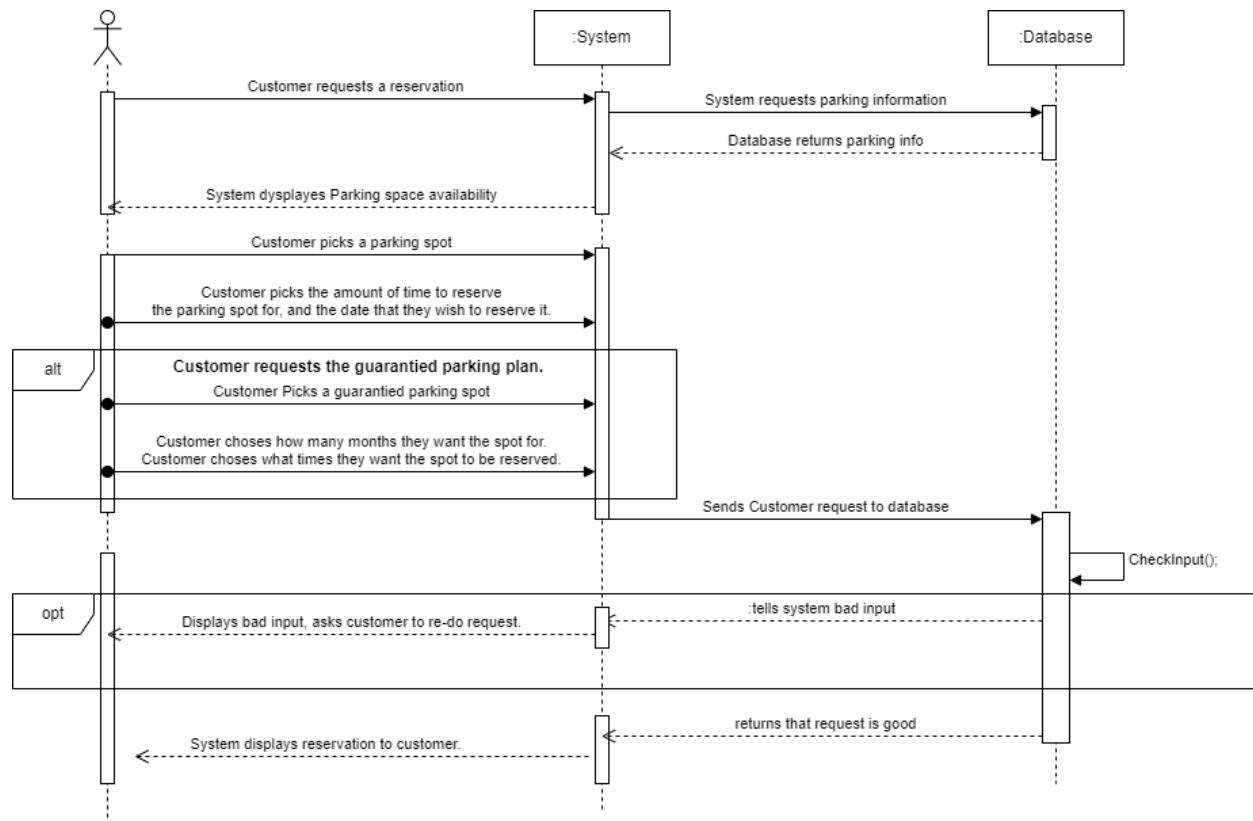
This data will include Customer data which includes `paymentMethod`, Vehicle information, and Parking space information which uses `exitTime` and `entryTime`. Using this data, we can use  $\text{exitTime} - \text{entryTime} \times \text{rate}$  to calculate the cost of parking.

Once the cost is calculated then we can create a Transaction, which will include `totalCost`, timestamps, and the foreign keys of customer, vehicle, and parking space. This Transaction data will then be used for the `sendInvoice()` method which will send a formatted display of the transaction for the user to read easily.

Once an invoice is received from the device there will be a `sendPayment()` method which will use the `paymentMethod` of the customer to pay for the Transaction. There will be a response if the payment is successful, if the payment is successful then the Transaction can be updated to paid. Once the Transaction is checked to be successful with `isPaymentSuccessful()` then the security gate will open for the customer to leave.



## Use Case 16: Reservation



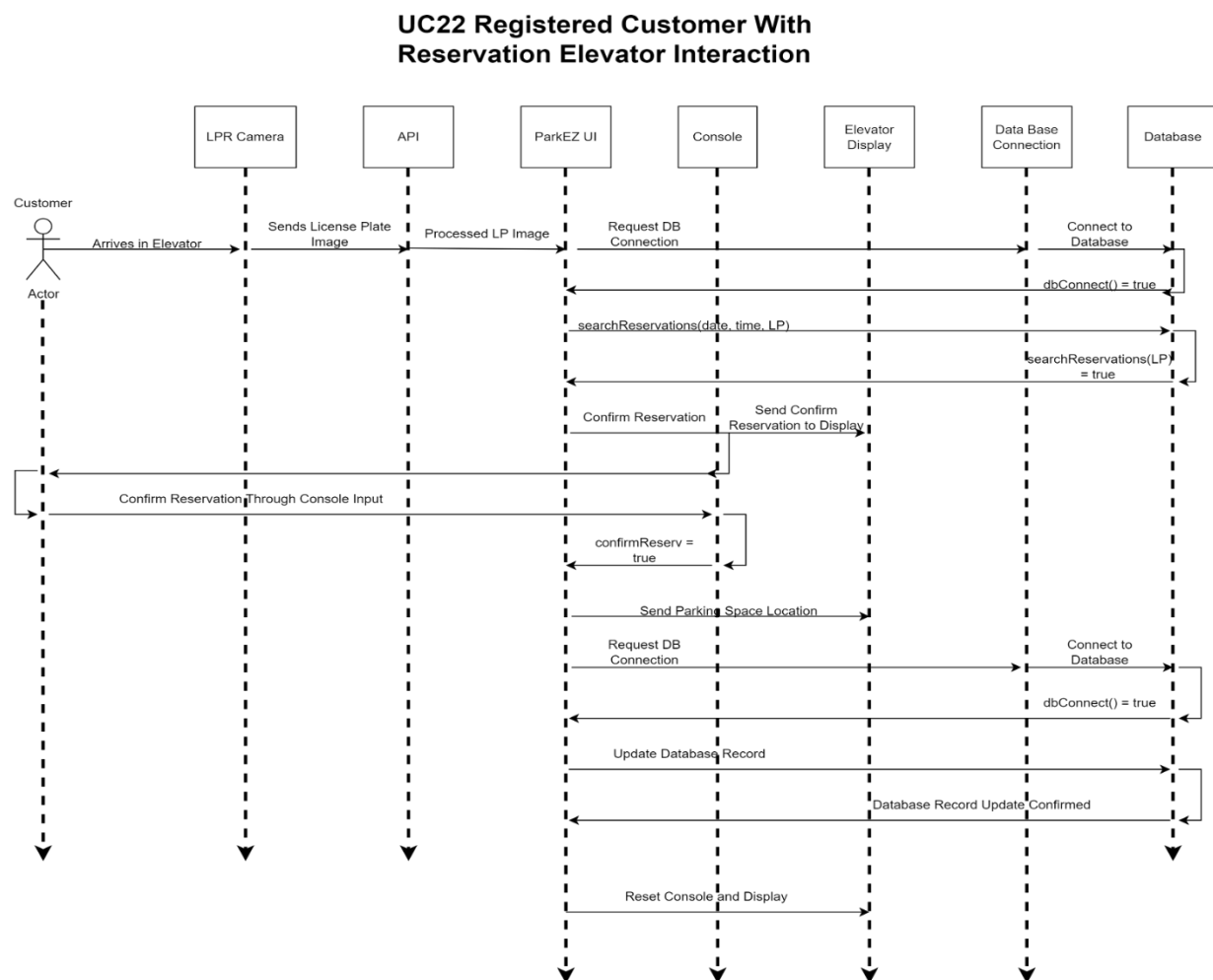
After the Customer logs into the ParkeZ webpage and enters the reservation tab they will be greeted with two options. First the customer will decide whether they want to make a single reservation, or if they want to make a guaranteed reservation for a month or longer.

If the Customer chooses to make a single reservation, they will be asked to specify the date, start time and duration of the reservation. The Customer will also be asked to choose a parking spot from the available options.

If the Customer decides to select the guaranteed reservation option, they will be asked to specify how many months they wish to reserve a parking space, the days of the week they will need it, the reservation start time for each day and the reservation duration for each day. The Customer will also be asked to choose a parking space out of the available options. For example, a customer may request to have a spot reserved for them Monday, Thursday, and Saturday from 8:00am to 6:00pm for two months. Outside of these hours the spot can be made available for use by other customers, but during the specified dates and times the parking space belongs to the Customer who made the guaranteed reservation.

After the Customer has chosen the specifics of their reservation the system will check the Customer's input. Depending on the results of the input check the system will either inform the customer that something has gone wrong and prompt the Customer to redo the reservation process, or the system will confirm the input and send a confirmation message to the Customer stating that the reservation request has been successfully made. The Customer will then exit the tab.

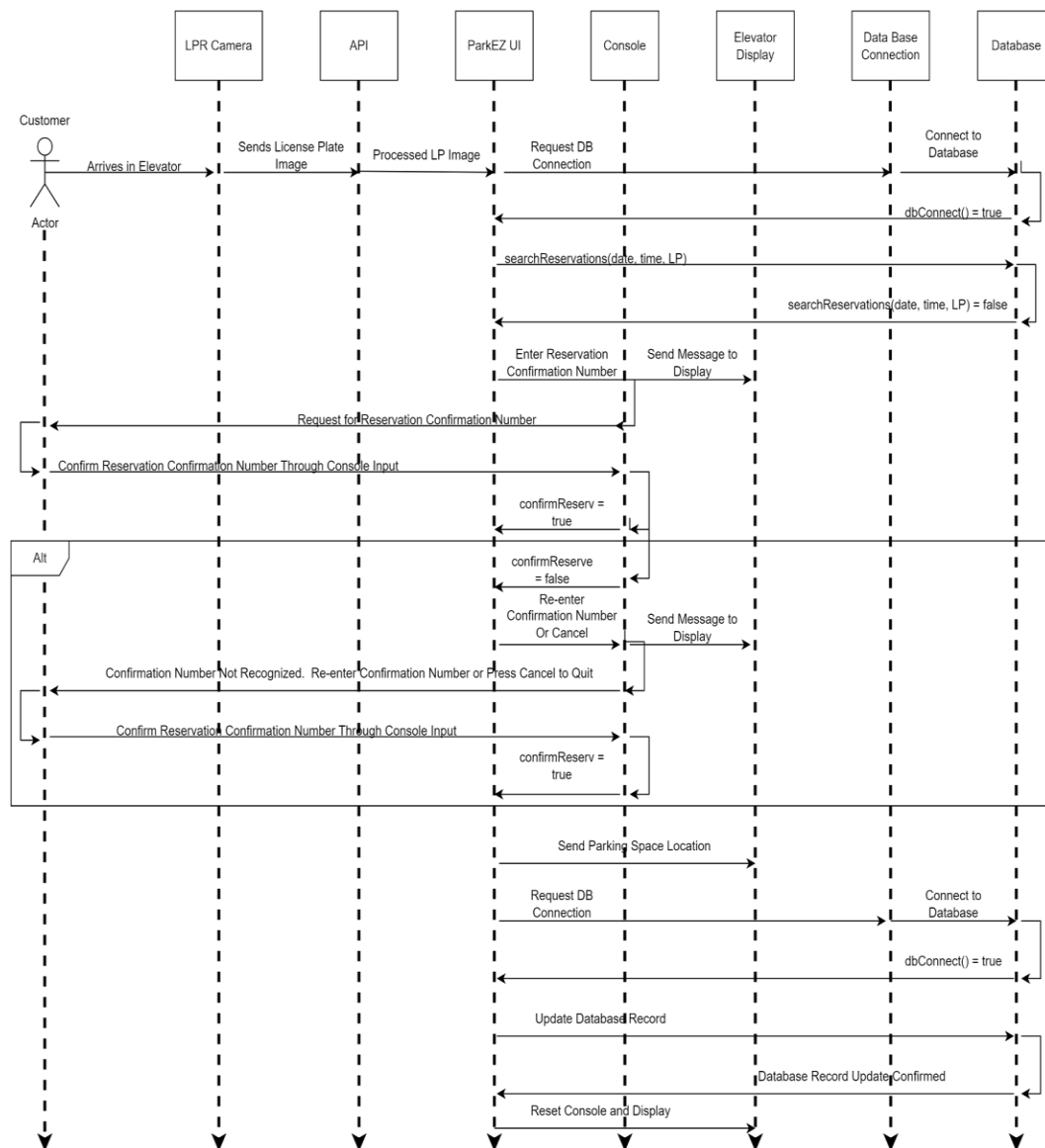
## Use Case 22: Elevator Access



The first Interaction Diagram of Use Case 22 represents the scenario of a registered user, with a reservation and a vehicle license plate number that matches the reservation, pulling onto the parking garage elevator. The customer arrives at the garage and pulls into the elevator. The LPR camera scans the customer's license plate, which sends the image to the API for processing. The processed license plate number is sent to the ParkeZ UI, which then requests a database connection. Once the

database connection is established (dbConnection() = true) the ParKEZ UI queries the database for outstanding reservations that satisfy the current date, current time and license plate number criteria (searchReservations(date, time, LP)). In this scenario the customer is registered, the license plate in the database matches the license plate scanned by the LPR camera and searchReservations(date, time, LP) returns true to the ParKEZ UI. The database connection is closed. The UI sends a request to the elevator console for user input to confirm the reservation. The UI also sends a message to the elevator display for the customer that states "Hello [customerName]! Please press "enter" on the console to confirm your reservation or "cancel" to quit and exit the garage." The customer uses the keypad on the console to confirm the reservation. The console sends the confirmation to the UI, which sends a graphical representation of the customer's floor, parking space and directions on how to get to the parking space from the garage elevator to the elevator display. The elevator moves to the floor where the customer's parking space is located. The customer pulls out of the elevator on the desired floor. The ParKEZ UI sends a request to connect to the database. Once the database connection is established the UI creates a record of the elevator interaction and saves it to the database. The reservation record is also updated. The database connection is then closed. The elevator returns to the ground floor and the camera, API, console, display and UI are returned to the original state.

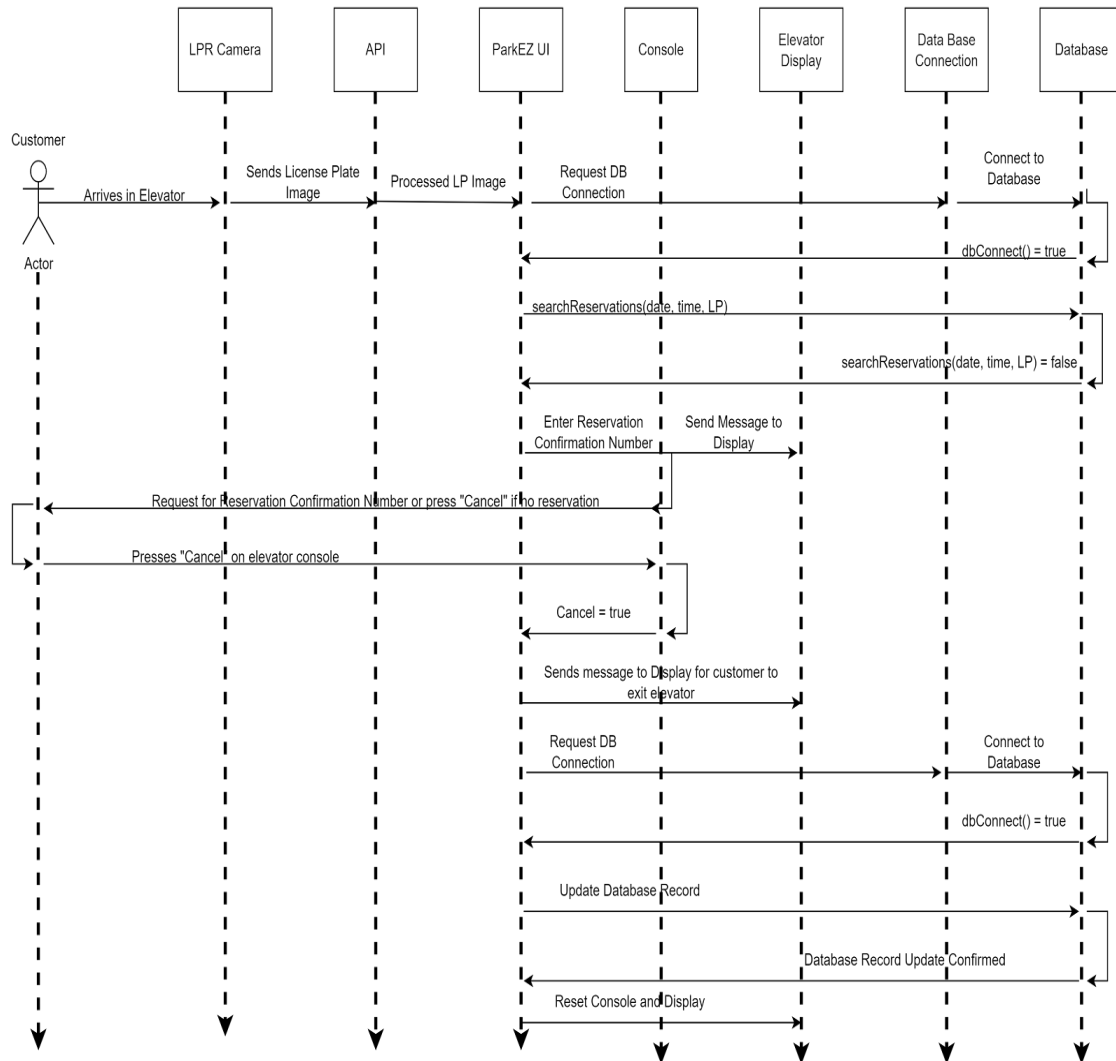
## UC22 Registered Customer With Reservation Alternate Interaction: License Plate Does Not Match Reservation



The second Interaction Diagram of Use Case 22 represents the scenario of a registered user that has a reservation, but whose vehicle license plate does not match the reservation, pulling onto the parking garage elevator. The customer arrives at the garage and pulls into the elevator. The LPR camera scans the customer's license plate, which sends the image to the API for processing. The processed license plate number is sent to the ParkeZ UI, which then requests a database connection. Once the database connection is established (`dbConnection() = true`) the ParkeZ UI queries the database for outstanding reservations that satisfy the current date, current time and license plate number criteria (`searchReservations(date, time, LP)`). In this scenario `searchReservations(date, time, LP)` returns false to the ParkeZ UI because the license

plate number does not match any of the current reservations for the current date and time. The database connection is closed. The UI sends a request to the elevator console for the user to enter a reservation confirmation number to confirm the reservation. The UI also sends a message to the elevator display for the customer that states, "The vehicle license plate number does not match any current reservations. Please enter your reservation confirmation number on the console and press "Enter" to confirm your reservation or "Cancel" to quit and exit the elevator." The customer then enters a valid reservation confirmation number using the console's keypad and presses "Enter." The console sends the confirmation number to the ParkEZ UI. The UI requests a database connection. Once the database connection is established (`dbConnection() = true`) the ParkEZ UI queries the database using date, time and confirmation number as valid criteria (`searchReservations(date, time, confirmationNumber)`). In this case the current date, time and reservation confirmation number is valid, so the search returns true. The database connection is severed. The UI sends a request to the elevator console for user input to confirm the reservation. The UI also sends a message to the elevator display for the customer that states "Hello [customerName]! Please press "enter" on the console to confirm your reservation or "cancel" to quit and exit the garage." The customer uses the keypad on the console to confirm the reservation. The console sends the confirmation to the UI, which sends a graphical representation of the customer's floor, parking space and directions on how to get to the parking space from the garage elevator to the elevator display. The elevator moves to the floor where the customer's parking space is located. The customer pulls out of the elevator on the desired floor. The ParkEZ UI sends a request to connect to the database. Once the database connection is established the UI creates a record of the elevator interaction and saves it to the database. The reservation record is also updated. The database connection is then closed. The elevator returns to the ground floor and the camera, API, console, display, and UI are returned to the original state.

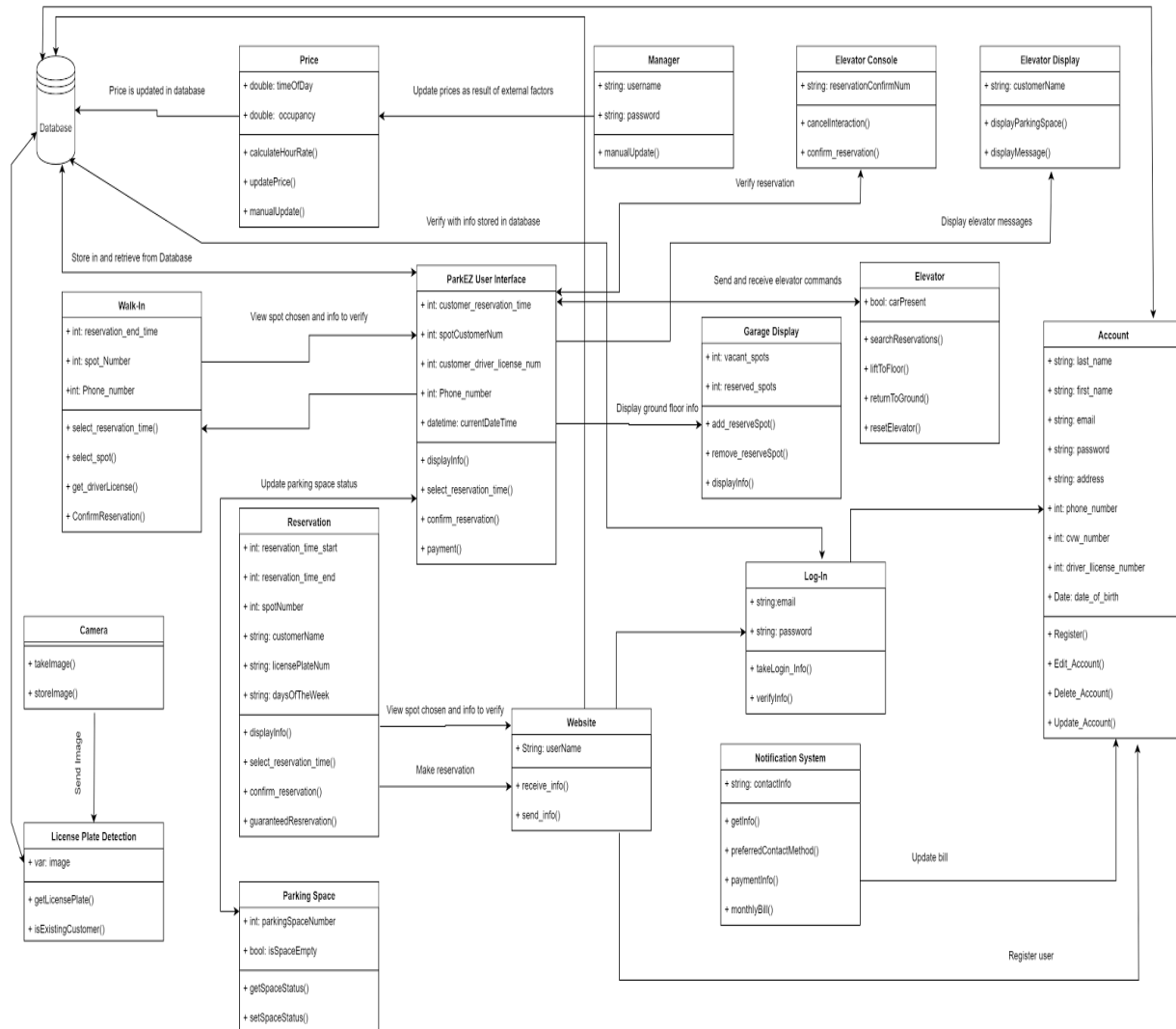
## UC22 Alternate Interaction: Registered Customer Without a Reservation or a Walk-in Customer Arrives in the Parking Garage Elevator



The third Interaction Diagram of Use Case 22 represents the scenario of a registered user without a reservation or a walk-in customer pulling onto the parking garage elevator. The customer arrives at the garage and pulls into the elevator. The LPR camera scans the customer's license plate, which sends the image to the API for processing. The processed license plate number is sent to the ParkeZ UI, which then requests a database connection. Once the database connection is established (dbConnection() = true) the ParkeZ UI queries the database for outstanding reservations that satisfy the current date, current time and license plate number criteria (searchReservations(date, time, LP)). In this scenario searchReservations(date, time, LP) returns false to the ParkeZ UI because the license plate number does not match any of the current reservations for the current date and time. The database connection

is closed. The UI sends a request to the elevator console for the user to enter a reservation confirmation number to confirm the reservation. The UI also sends a message to the elevator display for the customer that states, "The vehicle license plate number does not match any current reservations. Please enter your reservation confirmation number on the console and press "Enter" to confirm your reservation or "Cancel" to quit and exit the elevator." The customer presses "Cancel." The console sends the quit command (Cancel = true) to the ParkeZ UI. The UI sends a message to the display that states, "Please exit the elevator. See the first-floor display for available ground floor parking." The UI requests a database connection. Once the database connection is established (dbConnection() = true) the ParkeZ UI creates a record of the elevator interaction and saves it to the database. The database connection is then closed. The camera, API, console, display and UI are returned to the original state.

## Section 2: Class Diagram



## Data Types and Operation Signatures

**Account:** Includes registered customer information obtained from the ParkEZ 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-in 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.

## **Section 3: System Architecture**

### **Architectural Styles**

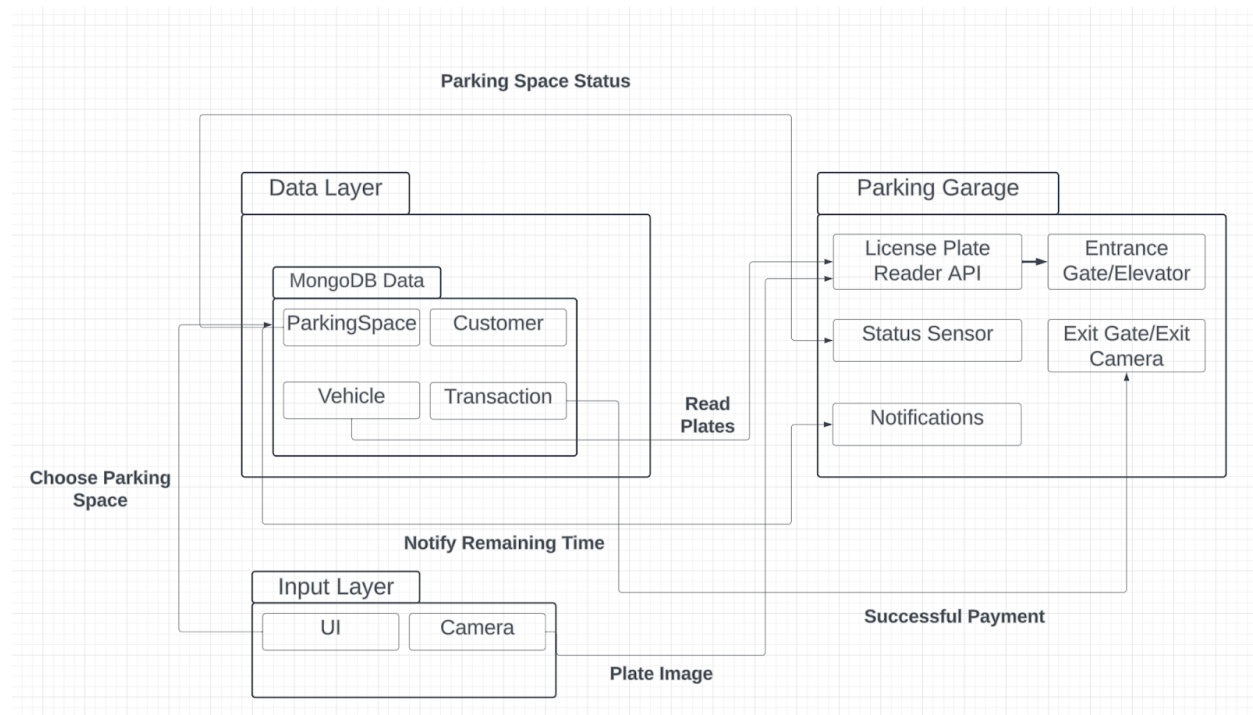
There are many architectural styles to choose from, each tailored towards specific types of applications. However, there did not seem to be a single style that fit the needs of the ParKEZ Automated Garage system. The ParKEZ system uses a web page to create customer accounts and make reservations, but also uses a container-based system to control the parts of the automated garage. Therefore, we elected to go with a hybrid system that uses an Event-Driven system with many containerized components making our system a union of the Event-Driven and Component-Based styles.

The Event-Driven style means that specific events take place that trigger communications between components or services. Some examples of this in the ParKEZ system can be seen in interactions that occur when using the website. When registering or reserving a parking space on the website each mouse click triggers another webpage or a popup with some information. In the garage itself there are also event driven actions taking place. When a registered customer's reservation ends and they exit from the parking space, the parking space sensor toggles and causes the space to show as vacant in the system. Also, when the same customer leaves the garage the LPR camera scans the vehicle license plate and triggers a database update with the exit time and addition to the customer's bill.

Besides the heavy use of the Event-Driven style, the ParKEZ Automated Garage system also uses the Component-Based style to break down many of the parts of the system into components and containers that can be tested individually. Some of these components can be reused for separate purposes after slight alterations. An example of this would be the two displays in the parking garage. One is used for the garage first floor and the other display is tailored for use in the elevator.

The ParKEZ Automated Garage system also uses a Database Centralized system. The website creates and updates database records when customers register on the site, make reservations, or cancel/edit existing reservations. The garage creates and updates database records when registered customers with reservations show up to use their reserved parking spaces or leave their reserved spaces. The garage also uses the database for walk-in customers that park on the first floor of the garage.

## Identifying Subsystems



## Mapping Subsystems to Hardware

### Mapping Subsystems to Hardware

Our system comprises several interconnected components, including Docker, web services, Visual User Interface, the Information Database, Camera, and License Plate Detector. The goal is to create a cohesive environment where these components work harmoniously to deliver the desired functionality.

#### 1. Docker Functionality

- Hardware Component: The Docker function operates within the cloud infrastructure.
- Description: Docker is a containerization technology that provides a flexible and scalable environment for our system. It can span multiple servers, ensuring high availability and scalability. The Docker containers can be hosted on cloud servers or virtual machines with adequate processing power and memory.

#### 2. Web Services:

- Hardware Component: The web services are hosted on dedicated or virtualized servers.
- Description: These servers should have the necessary computing power and memory to handle the incoming requests and serve web content efficiently. Load balancers can also be employed to distribute traffic across multiple servers to ensure high availability.

#### 3. Visual User Interface:

- Hardware Component: Visual User Interface runs on client devices (computers, smartphones, tablets).

- Description: The Visual User Interface requires client devices with sufficient processing power and graphical capabilities to render a user-friendly interface. The choice of hardware here mainly depends on the devices our customers are using, ensuring compatibility and responsiveness.

#### 4. Information Database:

- Hardware Component: The Information Database is hosted on dedicated database servers.
- Description: The database servers need to be robust with high storage capacity and redundancy to ensure data integrity and availability. Advanced storage solutions like RAID configurations can be implemented for data protection.

#### 5. Camera and License Plate Detector:

- Hardware Component: Cameras and License Plate Detectors are IoT devices installed in the garage.
- Description: These devices are specialized hardware components installed in the garage environment. They are designed to capture data efficiently, and they should be connected to a local network or IoT gateway for data transmission to the Information Database. The hardware should be capable of handling image and data processing tasks.

## 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 Park-Ez'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 4: Algorithms and Data Structures**

### **Algorithms:**

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

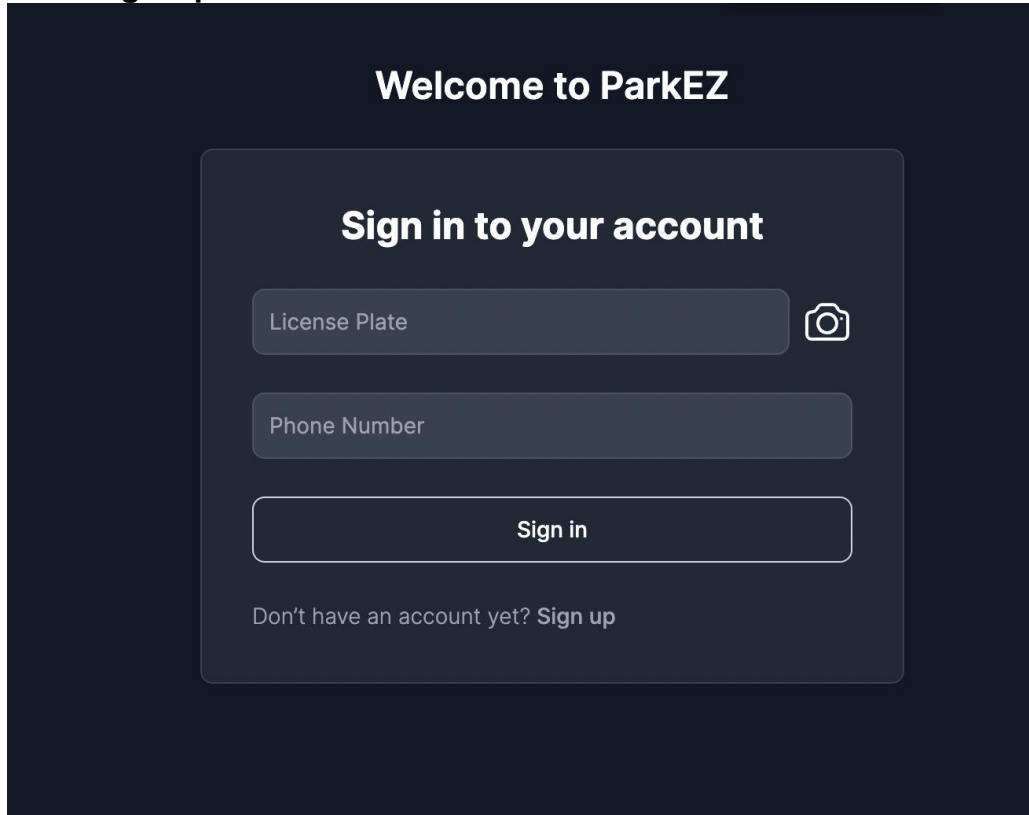
### **Data Structures:**

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

## Section 5: User Interface and Design 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 sign-in screen. At the top, the text "Welcome to ParkeZ" is displayed in a bold, white font. Below this, a central white box contains 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 "Sign in" button. 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**, **Available parking spaces**, and **Parking Space Map**.

- Date and Time:** Includes a date input field (mm/dd/yyyy) with a calendar icon and a time input field (--:-- --) with a clock icon.
- Available parking spaces:** A list of four options, each with a distance, location, and a green 'Go' button:
  - 5 feet away (Level A - Space 12)
  - 20 feet away (Level A - Space 22)
  - 30 feet away (Level B - Space 11)
  - 40 feet away (Level C - Space 13)
- Parking Space Map:** Three maps labeled Level A, Level B, and Level C. Each map shows a grid of 12 columns and 3 rows of green squares representing parking spaces.

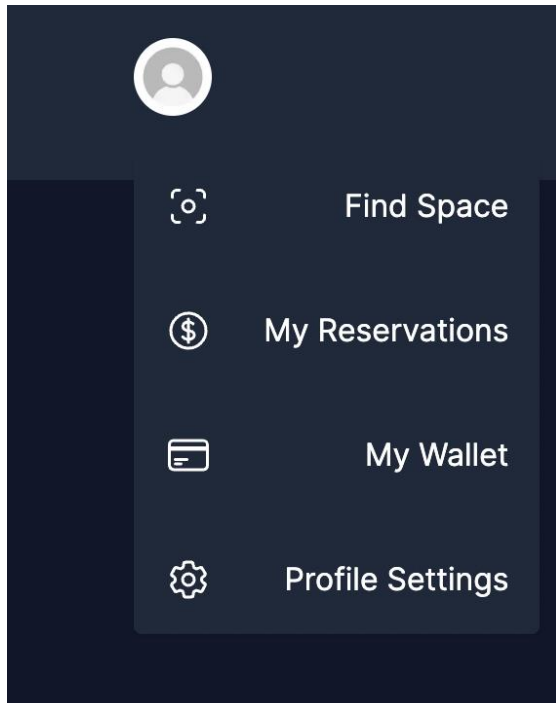
## UC-5 Walk-in

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.

**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 a 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**

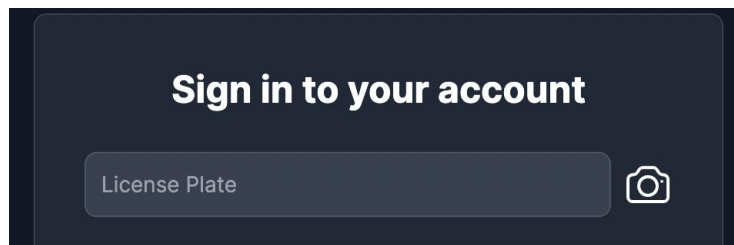
A password less, phone authenticated sign in will ensure all users are authorized and data will be better secured. Transactions will be encrypted and secured using Stripe API.

### **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**

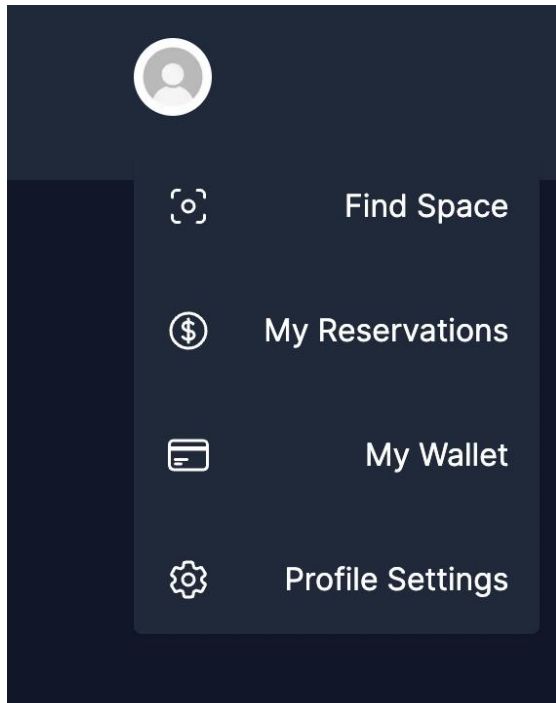
Customers can reserve a “one time” parking space or a monthly guaranteed reservation through /parking-space route.

**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 the user profile for faster transaction implementation.

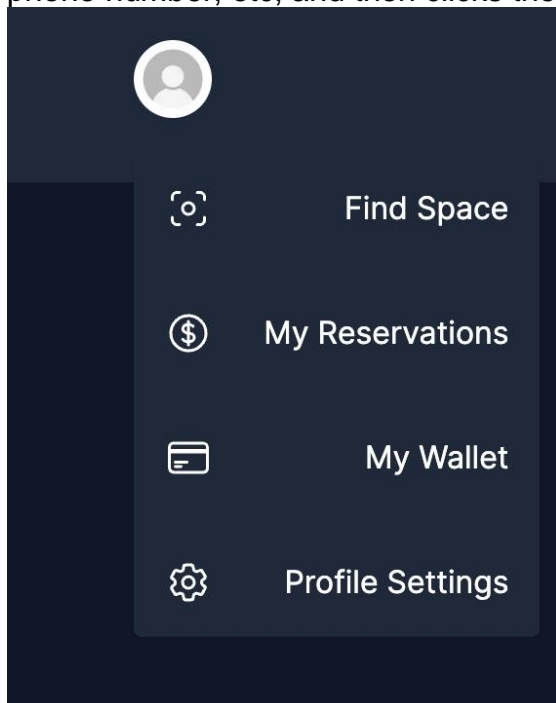




**Estimation:** To update payment method takes 2 clicks and text inputs.

### 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

localhost:3000/sign-in

Convert a string L... Feed | LinkedIn x Setup a contact fe... react - w3collective React user registr... Firebase web cod... Fort Hays State U... science RegEz: Learn, Bu... An Introduction to... Python MySQL De... All Bookmarks

Home Parking About Us

Welcome to ParkEZ

Sign in to your account

License Plate

Phone Number

☐ Remember me [Forgot password?](#)

Sign in

Don't have an account yet? Sign up

**Registration:**        /sign-up

localhost:3000/sign-up

Convert a string L... Feed | LinkedIn x Setup a contact fe... react - w3collective React user registr... Firebase web cod... Fort Hays State U... science RegEz: Learn, Bu... An Introduction to... Python MySQL De... All Bookmarks

Home Parking About Us

Park EZ

Create an account

Continue with Google

Continue with Facebook

Continue with Github

Continue with Apple

Make

Model Year

Payment Information

Bitcoin PayPal Apple Pay

Pay with AmEx VISA Pay with Visa

Pay with MasterCard

Create an account

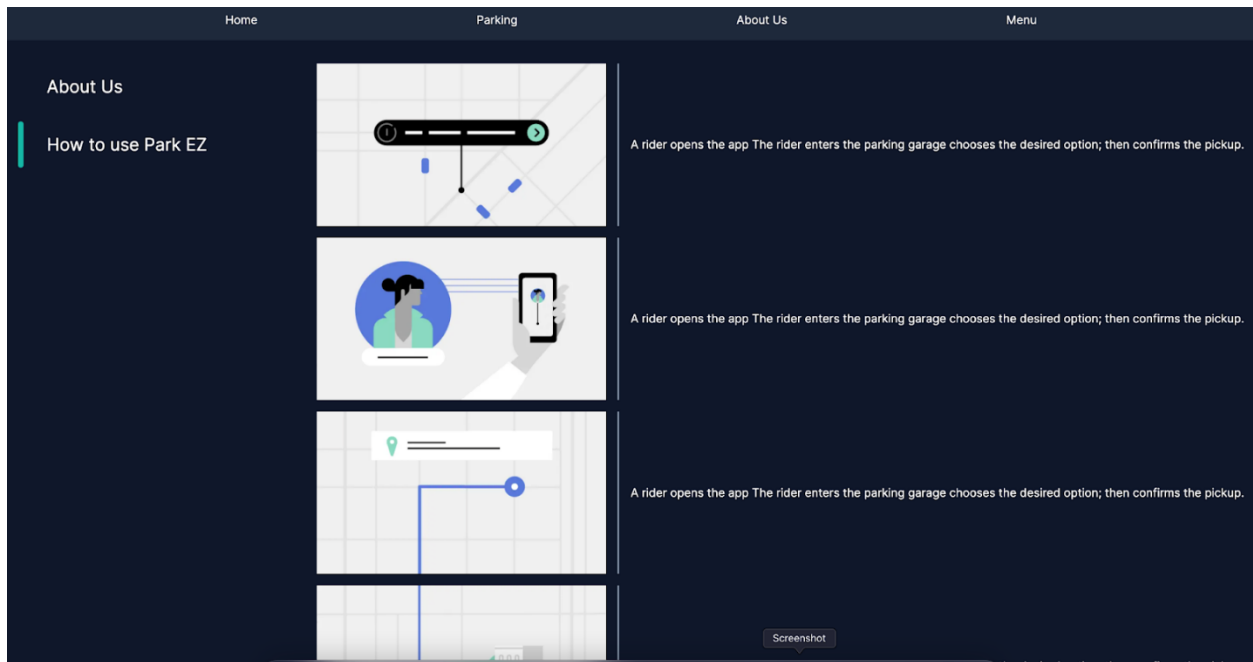
**Parking:**      </parking-space>

The screenshot shows a web application with a dark blue header containing navigation links: Home, Parking, About, and Menu. The main content area is divided into three sections. On the left, 'Available parking spaces' lists four options: '5 feet away' (Level A - Space 12), '20 feet away' (Level A - Space 22), '30 feet away' (Level B - Space 11), and '40 feet away' (Level C - Space 13). Each option has a green 'Go' button. The middle section, 'Parking Space Map', displays three maps for Level A, Level B, and Level C, each showing several red dots representing parking spaces. A green dot is visible on the Level A map. On the right, the 'Payment' section contains a form with fields for 'Card Number' (placeholder: 0000 0000 0000 0000), 'Expiration Date' (placeholder: MM / YY), 'CVV' (placeholder: 000), and 'Card Holder' (placeholder: Full Name). A green 'Submit' button is at the bottom of the form. A small 'TextEdit' button is located at the bottom right of the map area.

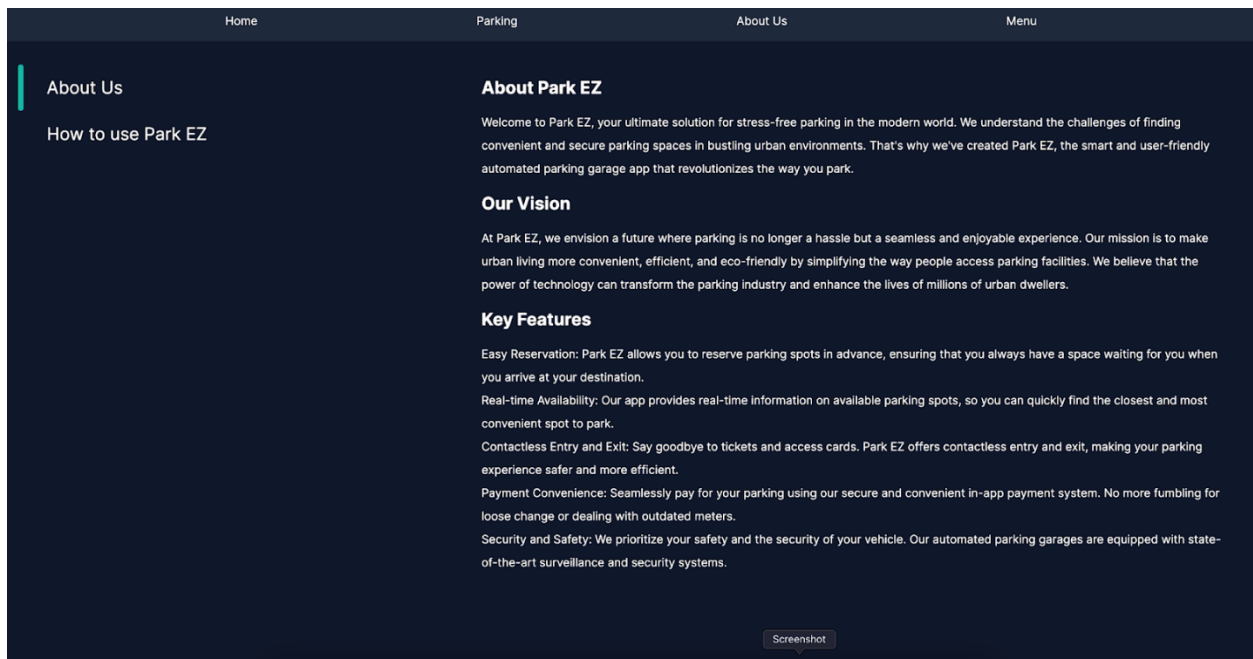
**Payment Confirm:**      </parking-space>

This screenshot shows the same web application as the previous one, but with the 'Payment' section updated. The 'Payment' text is now followed by a large green checkmark, indicating a successful transaction. The 'Available parking spaces' and 'Parking Space Map' sections remain unchanged. The 'TextEdit' button is still present at the bottom right of the map area.

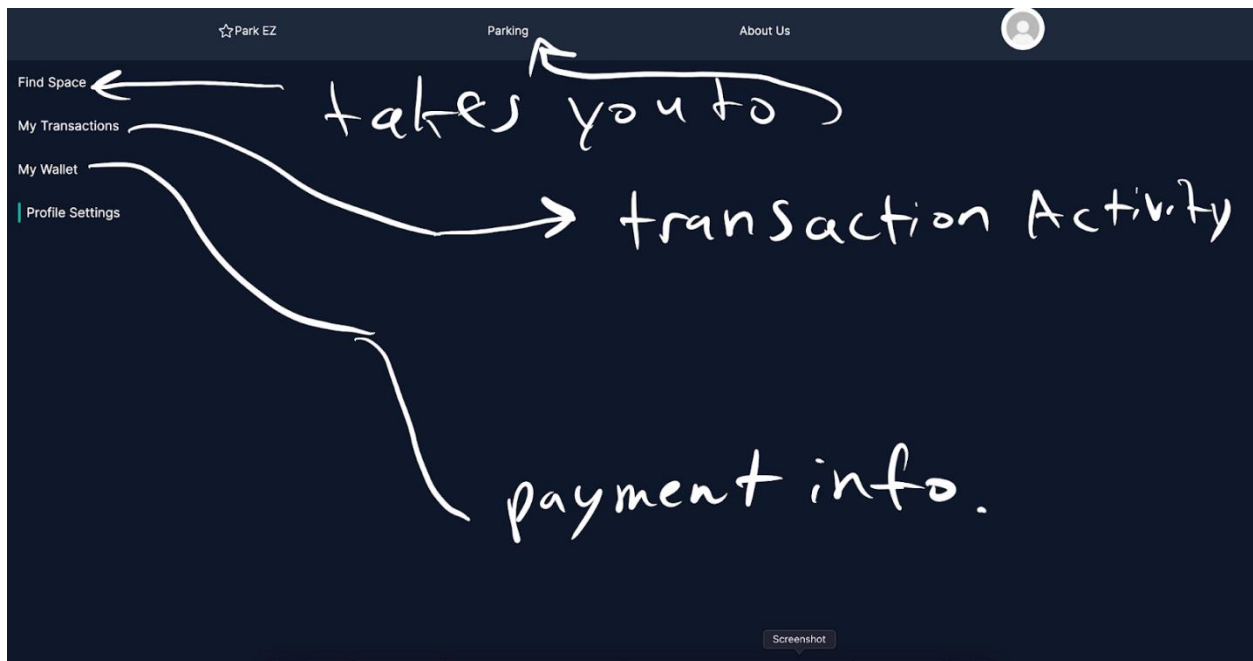
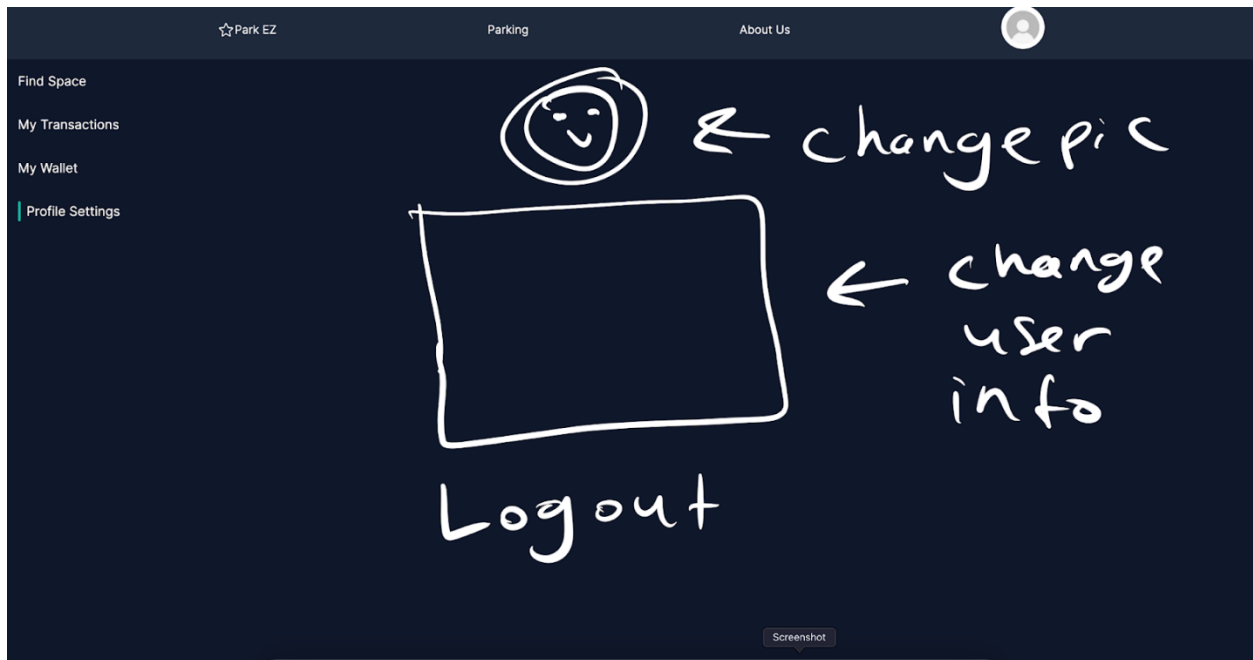
**About :**      </about/how-to-use>



## /about/about-us



Menu: /user-menu



## **Section 6: 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 use case code through a series of tests in order to ensure that the Customer and the System run 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 etc.
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 a LPR API to simulate a 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 to the screen. The elevator sensor will be toggled to false to show the vehicle 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 to the screen. The elevator sensor will be toggled to false to show the vehicle 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 to 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 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 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.

# **Project Management**

## **Merging the Contributions from Individual Team Members:**

Putting together a final report out of pieces from six team members, with sometimes much different schedules, does present its challenges. What we have discovered is that the earlier we can get our individual parts completed the better. Aside from getting things done as early as possible, the use of Google Docs has helped us come together, as a team, on the final report by allowing everyone to share, update and edit a single document. It is easy to see which parts of the report have been addressed and what still needs to be done. Once everyone has completed their assigned sections the final report gets a final read through and is edited before submission.

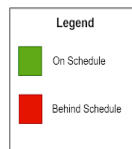
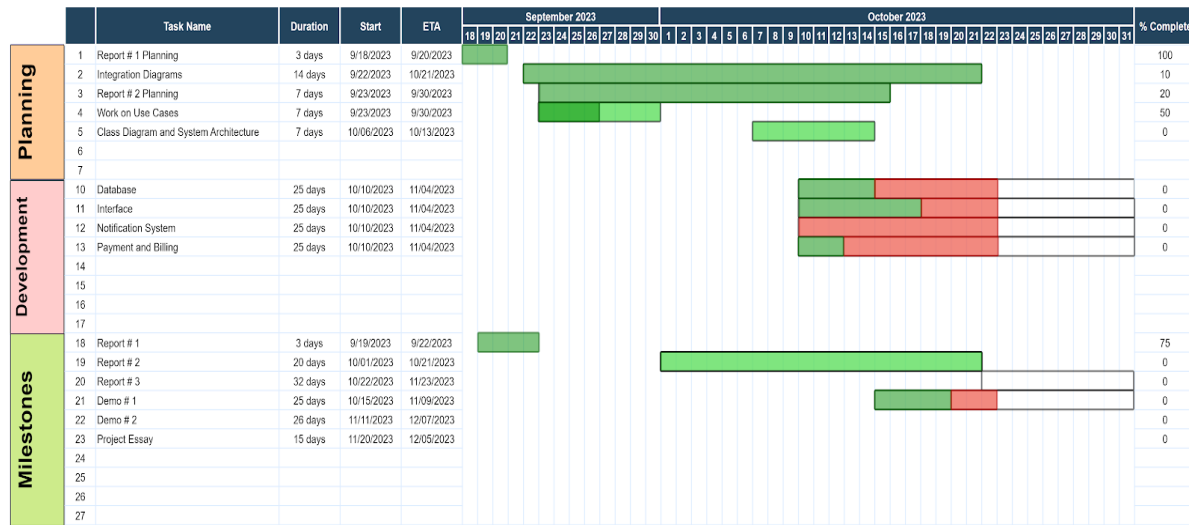
## **Progress Coordination and Progress Report:**

A basic home page shell for the ParKEZ Automated Parking Garage has been developed along with pages for Parking and About Us. Also, there is a drop-down site menu present, but is not fully functional. Also, there is a sign in form that has been developed. The MongoDB database is in the design phase.

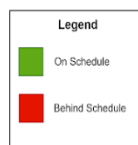
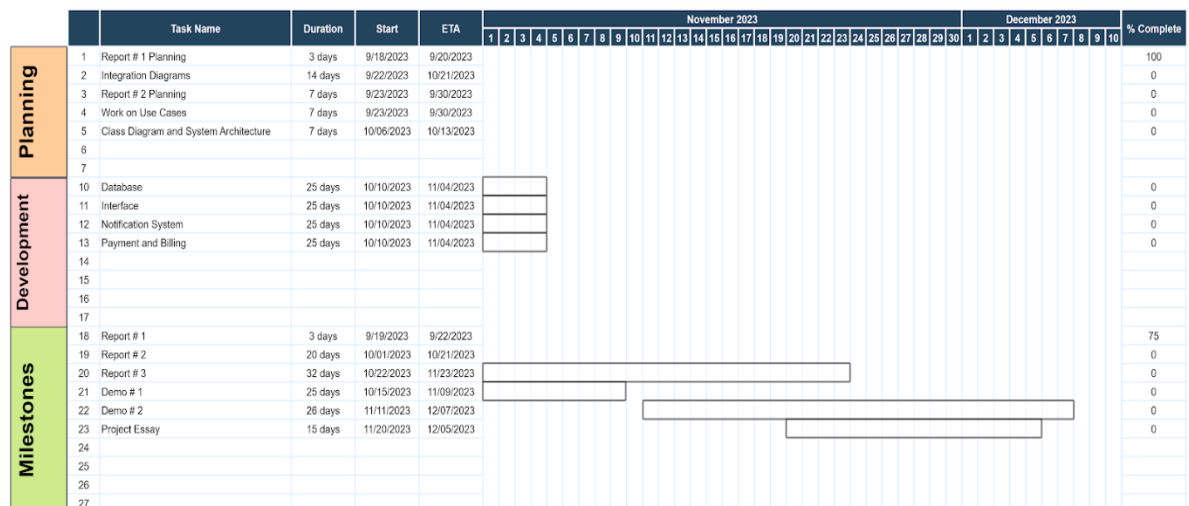
## **Plan to Work:**

This project is in the planning phase and all the foundation laid so far has been a team effort. Our group discussions have been conducted on a Discord text server that was created for the purpose of this project. We have also attended a weekly Discord voice chat for project discussion, planning and design. The Report was created on Google Docs and was shared with the entire group so that each member could contribute in real time to the content of the document and participate in the editing process. We will continue to use Discord for meetings and Google Docs for future reports.

### 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"> <li>Customer registration and profile management.</li> </ul> Mikael Mikaelian <ul style="list-style-type: none"> <li>Payment processing and billing.</li> </ul>	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.
Team Pair # 2: Christopher Smith Adrian Elgin	Christopher Smith <ul style="list-style-type: none"> <li>Real-time parking space availability updates.</li> </ul> Adrian Elgin <ul style="list-style-type: none"> <li>Reservation system with confirmation.</li> </ul>	Functionalities: Real-time parking space availability updates by integrating parking space sensors, license plate cameras and customer check-in software.  Qualitative Property: Ensure system performance to display availability within Seconds.  Treatment: Alleviate the problem of customer frustration due to inefficient parking space searches.
Team Pair # 3: Phongsavanh Mongkhonvilay Geoffrey Sarpong	Phongsavanh Mongkhonvilay <ul style="list-style-type: none"> <li>Integration with security and surveillance.</li> </ul>	Functionalities: Automated entry and exit for vehicles using license plate recognition camera system.

	<p>Geoffrey Sarpong</p> <ul style="list-style-type: none"> <li>Automated entry and exit for vehicles.</li> </ul>	<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>
--	--	--



## References

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

[https://www.ece.rutgers.edu/~marsic/books/SE/book-SE\\_marsic.pdf](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>