Garage Automation

Software Engineering – 14:332:452

Group #10: Kendric Postrero, Gao Pan, Vatsal Pandya, Yunqi Shen, Arthur Rafal, Guy Rubinstein

URL:
https://vatsal09.github.io/Garage-Automation/

Submission Date:
May 1, 2017

Summary of Changes

- Customer Statement of Requirements
    - Proposed Solution (Essay and List Versions)
        - Fixed the clarity of the essay version of our proposed solution by clearly defining the type of parking garage our system aims to improve
        - Fixed the list version of our proposed solution by making it consistent with our final use cases
    - Glossary of Terms
        - Edited and alphabetized the glossary terms to match the current iteration of the system
- System Requirements
    - Enumerated Functional Requirements
        - Manager-side requirements had to amended to reflect the changes
    - On-Screen Appearance Requirements
        - As per the TA's notes, we converted the hand-drawn sketches
- Functional Requirements Specification
    - Stakeholders
        - The language of the stakeholders was changed for consistency
    - Actors and Goals
        - Added "System", "User Interface", "User Interface Controller", "Manager Interface", and "Manager Interface Controller"
    - Traceability Matrix - Use Case vs. Requirements
        - Added another Traceability Matrix for the role of Management
    - Use Cases - Casual Description
        - Changed the language to reflect the language of the key words
    - Use Cases - Fully Dressed Description
        - Edited all of them, removing any mention of Application and incorrect use of System
        - Made log-in a precondition so it does not need to be mentioned in the flow.
        - Included use of all actors mentioned in Actors and Goals
        - As per TA's notes, we added more description to the flow of events
    - System Sequence Diagrams
        - Redid all the diagrams (Use cases 2, 6, and 11) to be consistent with the new Fully-Dressed Descriptions.

        - As per TA's notes, we added a few more sequence diagrams (3, 4, 7, 10, and 12)
- Domain Analysis
    - Domain Models Analysis
        - Revised the initial Domain Models to match the use cases that we fulfilled in our demo
        - Added description of changes compared with the previous Domain Models
    - Traceability Matrix - Use Case vs. Domain Models
        - Revised the Traceability Matrix
        - Added descriptions of how Domain Models are mapped to each Use Case
- Class Diagrams and Interface Specification
    - Class Diagram
        - Revised the initial Class Diagram
        - Explained how the Class Diagram evolved
    - Object Constraint Language (OCL) Contracts
        - Added OCL Contracts that matched our Class Diagram specifications
- System Architectures
    - Identifying Subsystem
        - Made a new Package Diagram to identify the subsystem
- Algorithms and Data Structures
    - Algorithms
        - Changed the ambiguous language to reflect what we were initially trying to convey
- Interaction Diagrams
        - Diagrams for use cases 1, 2, 5, 6, 8, 9 and 13 updated to be consistent and removed all use of "Application", as instructed by the TA.
        - Design principles rewritten to reflect the changes in the diagrams.
- System Architecture and System Design
    - Identifying Subsystems
        - As per TA's comments, we created a digital rendering so it is easier to view now.
- Design of Tests
    - Test Cases
        - As per TA's comments, we changed our method of describing our tests by following the format of the lectures by defining test cases that link to important use cases and functionality

Effort Breakdown

	Vatsal	Kendric	Gao	Arthur	Yunqi	Guy
Summary of Changes	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Customer Statement of Requirements	5%	90%			5%	
Glossary of Terms	100%					
System Requirements		40%			60%	
Functional Requirements Specification	10%	5%	5%	25%	15%	40%
Effort Estimation		100%				
Domain Analysis		33.3%	33.3%		33.3%	
Interaction Diagrams	15%	10%		35%		40%
Design Patterns		20%		10%		70%
Class Diagram and Interface Specification			80%	20%		
OCL Contract Specifications		70%	30%			
System Architecture and System Design	25%				75%	
Algorithms and Data Structures	30%		70%			
User Interface Design and Implementation	40%			60%		
Design of Tests	5%	95%				
History of Work, Current Status, and Future Work	95%				5%	
References	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
Project Management	90%		5%		5%	

Table of Contents

Customer Statement of Requirements

Statement of Goals

The objective of this project is to design a software solution to streamline the parking process in commercial parking lots, by offering an **automated, ticket-less payment system**. This new procedure aims to provide an efficient car flow, minimize nearby traffic, and provide a quicker customer turnaround—thereby increasing occupancy and revenue. This is comparable to E-ZPass, but for parking lots.

Problem Statement

Our customer owns a multi-level, commercial parking garage without an automated system to fast-track the payment process or check for occupancy logistics. The current system relies on manual inspection of occupancy by employees, and this only leads to a slow and primitive way of indicating to customers whether a spot is available. At peak times, the garage could have free spots but no way of instantly checking or displaying the information, discouraging customers from parking at their garage. In another scenario, customers might search for open parking spots only to discover that there are no available spots. The displeasure and frustration felt by the customers could deter future business. **The current ticket-based payment system also adds to the congestion inside the parking lot. Each customer has to stand in a line to validate and pay their ticket, then fumble around their car to look for the ticket, and finally stretch their hands all the way out to insert the ticket.** This impedes swift exits and adds to the congestion of people trying to find a parking spot. It could be detrimental to the surrounding businesses, as well as for the owner of the parking lot, if customers are discouraged by the inconveniences of finding a parking spot or dealing with the traffic congestion caused inefficiency of the parking lot system.

Proposed Solution

Essay Version

A parking garage's main purpose is to give a potential user convenience and ease of access. The increased efficiency brings in more customers and thus, allows businesses to generate more revenue. Moreover, larger parking garages can result in greater inefficiency if managed incorrectly; larger lots require more time to traverse and congestion affects more consumers. **The type of parking garage our system aims to improve is the turnstile system where customers drive up to a gate, receive a ticket, and the gate unlocks; the driver pays the ticket when he wishes to leave the lot and hands off the paid ticket to leave the exit gate.**

As an owner of the parking garage, the new system will be based around a website that will give the user information about any commercial parking lot that has implemented this system. The information displayed to the user will consist of the available parking spots at each level, the ability to add/update payment method, and the payment history of the user. To achieve this, the system uses laser-based sensors to keep track of available parking spaces. **The sensor will determine if a car is occupying the parking spot and mark it in the system as such and when a car exits the spot, the system will know that the spot is vacant.** This streamlines the traditional process of manually checking parking spaces to judge occupancy state, and thus reduces labor cost. This also reduces congestion within the parking garage and reduces the time wasted searching for a spot. When customers are faced with choosing a parking garage, they're inclined to select the one that is guaranteed to have an open parking space.

For more specification on the hardware, each parking spot will have a sensor and the entrance and exit will have a camera for analytics purposes. The sensor will give constant updates to the system about the state of the parking spot (occupied or unoccupied). **The system will be aware the state of all the parking spots in the parking lot, and will display to the user a visual a map of the parking lot with parking spaces color-coded with green for occupied and red for unoccupied. Our system does not deal with reservations because our parking garage is not the elevator**

style parking garage but the turnstile system so we cannot reserve a parking spot, only recommend a free space.

The system also has the added benefit of automated transactions at the turnstile. A registered user will be able to pay using a linked credit card and license plate. At the entrance, a registered user will be able to drive up to the turnstile, and an entrance license-plate-reader will quickly identify the license plate, understand that they are registered in the system, note the arrival time, and allow entrance. Upon exiting the parking lot, an exit license-plate-reader will identify the car's license plate, calculate expenses based on the differences of the exit time and arrival time, and execute charges on the provided payment method.

We will have to take into account that not all users will register to this system. There are two kinds of guest users to consider. Credit card holding guest users and cash-only guest users. Credit card holding guest users that have not registered online for the system beforehand will have a different experience. Credit card holding guest users will have to approach the turnstile and swipe their credit card to be added into the system as a guest—linking their swiped credit card and analyzed license plate. The benefit that these credit card holding guest users have is that they will not have to go through the ticketing system and have an ease of exiting. For those guest users who prefer cash, have the option to still use the old ticketing system allowing them access to the parking lot without the beneficial features.

We must make some assumptions in this project in order to simplify it. If we did not, this project would be much more complicated, and we would run the risk of failing to complete at the due date. We must assume that the license-plate-reader will work with high accuracy (95%+), the registered user has an email address, a credit/debit card, and a mobile phone with GPS and internet connectivity capabilities.

One of the key benefits of this computerized system is that, in the future, a management-side dashboard can be implemented. The web-based dashboard could serve as a virtual gateway for management to check important metrics: number of parking spots utilized per day, average customer time of stay,

daily/weekly/monthly/yearly revenue, and revenue trend compared to historical revenue data. The analytics can help the owner make crucial business decisions which impact the future of the business. For example, if the owner sees that the occupancy rate data suggests that there is high demand for parking in the area, then he/she can use this information to expand his/her parking lot or invest in a nearby parking lot. Likewise, analytics can also allow an owner to understand a loss in demand so that they can divert or allocate resources as they see fit. The manager has the ability to deactivate a parking spot, i.e. marking it red for the app, for repairs, cleanup, maintenance, and etc.

Proposed Solution

List Version

1. The user shall be able to register for an account online and add user information such as Name and Credit Card Information.
2. The customer shall have the option to delete their account, change information, or add information.
3. The system shall be able to report via sensors which parking spots are occupied and which are unoccupied.
4. The user shall be able to access parking lot information, such as the visual occupancy within the parking lot.
5. The system shall be able to analyze a car's license plate via a license-plate-reader to determine whether or not they are a registered user in the system.
6. The system shall be able to create guest accounts by linking the license-plate-reader information with the swiped credit card by guest user.
7. The system shall be able to recognize a registered user, or guest account (if they swipe a credit card), or cash user (if they take a ticket) and process this information to open the turnstile for customers.
8. The system shall be able to calculate the cost of expense by subtracting exit time from enter time— for the customers using automatic (credit card) payment.
9. The system shall be able to charge the customer automatically upon exiting at the turnstile by analyzing the license plate, connect the license plate to the user, and execute a charge on an account (registered or guest) linked to the system.
10. If the user is registered, the system shall save transaction history.
11. The system shall collect various user data such as the amount of users in a specific parking lot, and time they have used the parking lot.
12. The management is able to register for a management account to initialize the system, on the management website.

13. On the management website, the management account is able to add parking lot, delete parking lot, add spots, delete spots, connect spots to sensor ids, and view lot details
14. The management website will have access to real-time occupancy data, i.e. visual indication of which spots are available and which ones are occupied for the management team.
15. The management account can disable parking spots that appear available on the customer's website.

Glossary of Terms

Automated Transactions	A payment transaction that has been conducted electronically by calculating the time of stay and charging that amount to the credit card associated with license plate, using the license plate reader, within the parking system.
Cash User	A customer who uses the ticketing system and pays for the parking in cash.
Computer Vision	A software tool that automatically extracts information from an image to be used for other software applications.
Customer	The owner of the vehicle that has been driven onto the parking lot.
Dashboard	A control panel GUI that organizes and displays information in user-friendly manner.
Efficiency	The ability to accomplish an action without wasting time or resources. In this context, ability to process payments quickly and manage car flow within the lot.
Guest User/Guest Account	A non-registered user of the service. These temporary users are not stored in the database entries.
GUI	A user interface that allows the user to interact with the system using images rather than text commands; acronym for graphical user interface.
License-Plate-Reader	A combination of hardware (camera) to take pictures of a license plate and computer vision to detect the license plate number.
Management	The owner/management of the parking lot and parking system. They are able to view the parking system, view availabilities and disable certain spots, and etc.

Occupancy Rate	The rate at which the parking spaces are used compared to the total amount of available space.
Parking History	History of locations, durations, and payments for utilizing parking lot.
Register	The action that a customer takes to sign up and send information needed to use the parking system. This action sends the required information to the necessary components.
Sensors	A hardware device used to detect if a parking spot is taken.
Time of Stay	Duration of time between when the user enters the parking lot and when the user exits the parking lot.
Turnstile	A mechanical gate used to physically facilitate the entrance and exit of customers.
User Accessibility	Design process to allow all users to interact with the system efficiently and with ease.
User/Registered User	A customer who has a registered account in the system. These users have entries and payment history stored in the database.
Website	A set of webpages linked together

System Requirements

Enumerated Functional Requirements

Identifier	User Story	Story Point
ST-1	As a user, I will be able to create an account on the app and add my credit card & license plate information.	8
ST-2	As a user, I can also update my information and delete information.	5
ST-3	As a user, the app will recommend me a spot to park	13
ST-4	As a user, the app will display number of spots unoccupied and a map of which spots are open, with red indicating occupied and green indicating unoccupied	5
ST-5	As a user, the app will automatically charge the credit card associated with my account, depending on the duration of the stay	8
ST-6	If I am a guest user, I will still gain the benefits of ST-5 with a guest account, but I will have to swipe my credit card at the entrance turnstile and not enjoy the benefits of ST-9	8
ST-7	As a user, I will be able to see history of my transactions, i.e. how much I paid, which date I paid, and the length of my stay	5
ST-8	As a manager, I will have a website where I can find and update key information, including adding/updating/deleting parking lots, parking spots, and connecting spots to sensor ids	5
ST-9	As a manager, I will be able to see which spots are taken on a visual parking map	2
ST-10	As a manager, I will be able to disable parking spots (mark red on the map) for repair and maintenance	3

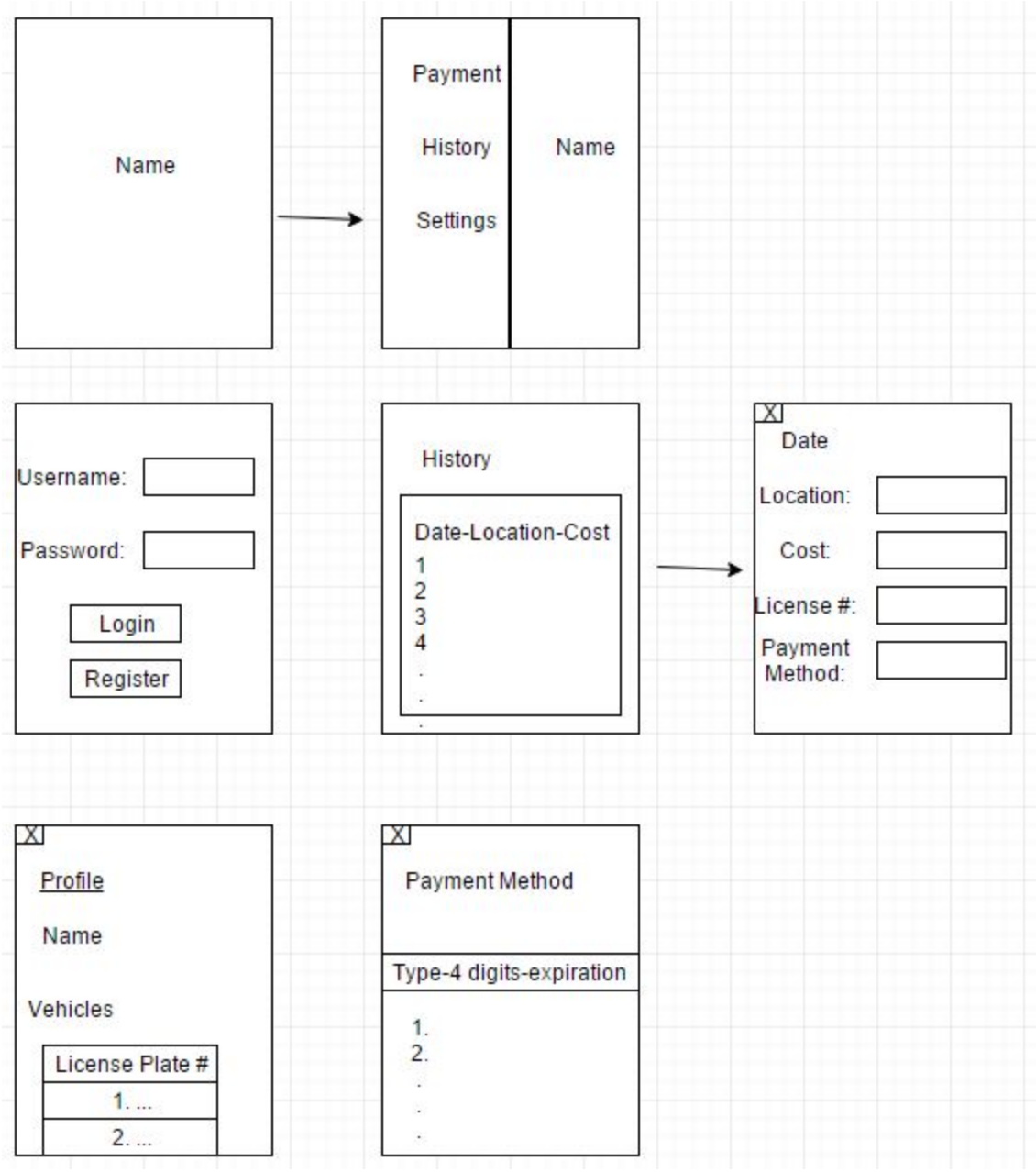
Enumerated Nonfunctional Requirements [7]

FURPS	
Functionality	● The laser-based sensors will be able to determine whether a parking spot is occupied ● Includes a camera-based system that is able to detect license plate numbers at the entrance and the exit point ● For guest users, there will be a system where they can swipe a credit card at the entrance
Usability	● There is a help dialog box on every page to provide the user with an idea of the steps they need to take for certain actions. ● There is consistency as all the web pages follow the same template. To keep it simple, the site was kept to roughly five pages. ● To accomplish simplicity, each page has a navigation bar to access the individual pages
Reliability	● Database is stored in the cloud to take advantage of upgrading storage at anytime, little to no downtime, and continued lowering of cloud storage prices
Performance	● To be efficient, the repeat customers will not have to go through the payment process themselves. Their linked account will charge the credit card associated with the license plate. ● Map of the parking lot and its spaces is available to customers so that they know where open spaces are ahead of time.
Supportability	● This design is very adaptable as far as the garage size is concerned. ○ If there are ever future plans to expand the garage additional floors or any other renovation, the adjustment would be as simple as making an update in the database table. ● Maintainability is also fairly straightforward as there isn't too much customer information to deal with. ○ If there was ever a need to remove or update a customer from the database, the action would be carried out to their credit card, vehicle, and reservation information automatically

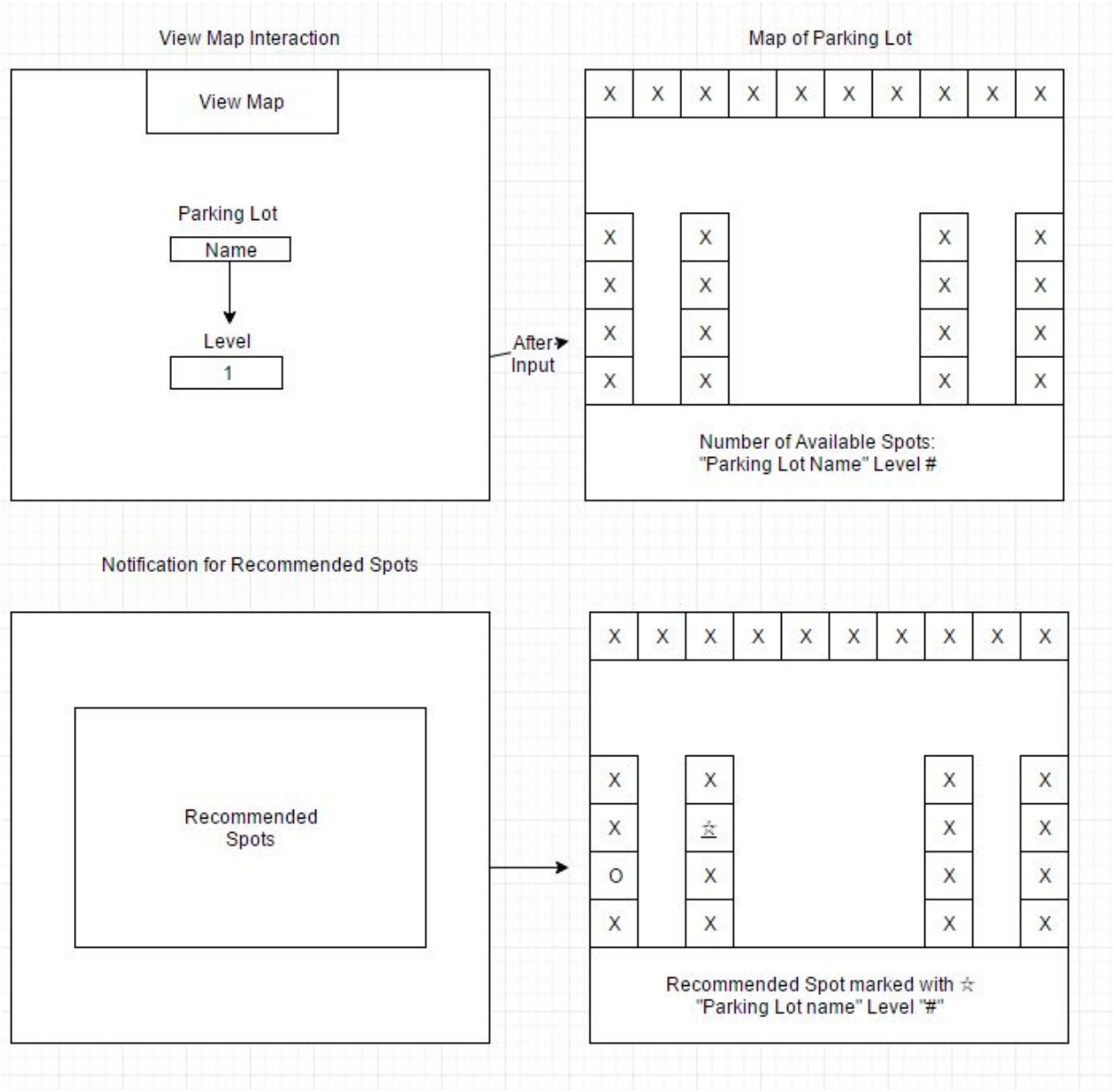
On-Screen Appearance Requirements

Our team is not building a product that relies on heavy graphics. Nonetheless, during the weekly meetings, we decided to create a mockup to truly understand what the user might see and how he/she will interact with the system. This gave us insight on which functions need to be added and how the team would have to subdivide the tasks. In addition, we were able better understand the design requirements of the shared API and the structural nature of the data.

Website Terminal for Registering, Adding User Information, and Viewing Transaction History

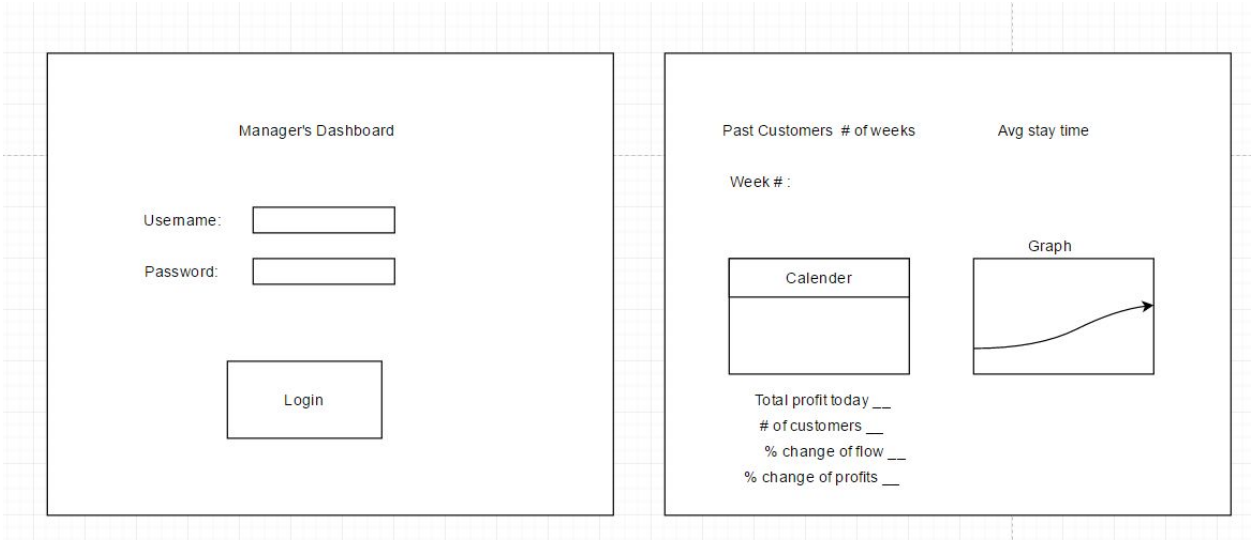


Map of the Parking Lot for the User displaying certain parking lot information



The user sees a map of the parking lot. He will see all the parking spots and information about the state of occupancy. A red X will indicate a spot is occupied. A green O will show that the space is available, and a star would show where the recommended spot is located.

Manager Dashboard (Future Work)



The manager first has to log in with the first web page. After successful authorization, the second page is shown. This page displays analytical information on the parking lot. It also provides tools for the business manager to track the transactions and build analytical reports concerning the revenue, customer behaviors, pathway traffics, so that the manager can figure out where to improve.

Functional Requirements Specification

Stakeholders

A stakeholder is someone who has interest or concern with the system-to-be. Due to the umbrella description, the system-to-be will have several types of stakeholders— all with varying degrees of stakes in the system-to-be. Individuals, teams, and organization with a stake include:

- Parking Garage Owners - sponsor of the system-to-be
- Maintenance - on-site individual whose job is inspect garage wear and tear, monitor manager-side website, and ensure everything is going smoothly
- End Users - individuals with and without accounts who will be parking their cars
- Database Manager - ensuring data is securely being updated, processed, and backed up
- Business Analyst - monitoring the manager-side dashboard to see if the revenue is aligned with projected revenue and looking for growth opportunities
- Camera and Sensor Companies - engineering the hardware components for the system-to-be
- Local business - business owners in the near proximity of the parking lot
- Payment service companies - credit card companies and banks involved with verifying, charging, and processing payment transactions

Actors and Goals

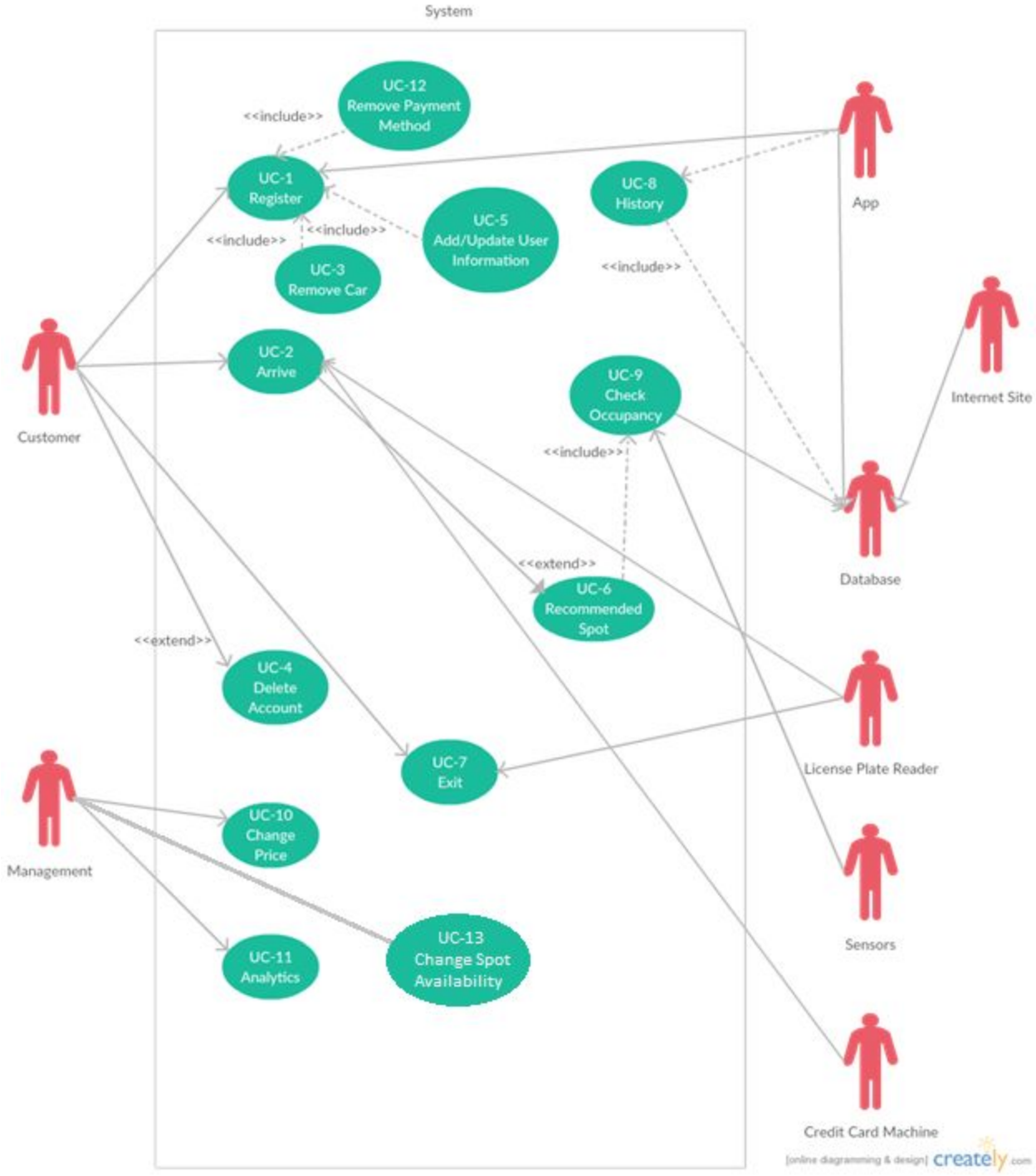
Actors	Goals
Management	To monitor the management-side website, disable spots when needed, and view the current occupancy of the spots
Customer	To park the car, and view a map of spots before they enter
User Interface	To allow customers to create an account, display transaction history, display recommended parking spot, display number of vacancies, and display occupied/unoccupied parking spaces
Database	To store all of the customer account information, duration of stays, and transaction history. To store parking occupancy data
License Plate Reader	To extract license plate number
Vacancy Display	To display the number of vacant spots within the garage
Turnstile	To prevent cars from accessing the parking lot for free
Sensors	To find out whether a spot is vacant or not
Credit Card Machine	Terminal to accept credit card payments
System	Controller for Turnstile, Sensor, License Plate Reader and Credit Card Machine.
User Interface Controller	Controller for User Interface
Manager Interface Controller	Controller for Manager Interface
Ticket Machine	Print and Receive tickets

Use Cases

Casual Description

Use Case	Name	Description
UC-1	Register	User registers for an account online, providing information including a payment method
UC-2	Arrive	User pulls up to the turnstile
UC-3	Remove Car	User removes a car from his/her account
UC-4	Delete Account	User deletes account
UC-5	Add/Update User Information	User adds/updates his address, credit card, or car information
UC-6	Recommended Spot	User is given a recommended parking space where the customer should park. **(Future work)**
UC-7	Exit	User exits the garage via exit gate
UC-8	History	User views their payment transaction history
UC-9	Check Occupancy	User checks the a map of the parking lot for which spaces are occupied/unoccupied and how many spots are open
UC-10	Change Price	Manager can adjust prices for parking **(Future work)**
UC-11	Analytics	Manager can view data statistics about parking lot revenue and logistics, and detailed payment history **(Future work)**
UC-12	Remove Payment Method	User deletes a payment method
UC-13	Change Spot Availability	Manager can manually change the availability of parking spots

Use Case Diagram



Traceability Matrix

Customer

Use Cases	Register	Arrive	Remove Car	Delete Account	Add/Update User Information	Recommend Spot	Exit	History	Check Occupancy	Change Price	Analytics	Remove Payment Method
Requirements												
Scan license plate		X					X					
Recognize registered customers		X										
Display number of available spots						X			X			
Display a map of which spots are occupied/unoccupied									X			
Create user account	X	X										
View past transactions								X				
Update user information					X							
Delete user information			X	X								X
Create temporary account		X		X								
Automatic payment		X					X					
Manager Analytics									X	X	X	
Total PW	5	23	5	13	5	10	15	5	17	3	7	5

Management

Use Cases	Register	Arrive	Remove Car	Delete Account	Add/Update User Info	Recommended Spot	Exit	History	Check Occupancy	Change Price	Analytics	Remove Payment
Requirements												
Scan license plate												
Recognize registered customers												
Display numbers of available spots												
Display a map of which spots are occupied/unoccupied									X			
Create user Account												
Update user info					X							
View past transactions								X				
Delete user information												
Create temporary account												
Automatic payment												
Manager Analytics						X				X	X	
Total PW					5	3		5	4	3	7	

Note: We found it difficult to find explicit way in which the traceability matrix is defined. For those reasons, we consulted a previous project for more direction. [13]

Fully-Dressed Descriptions

Use Case UC-1: Register
Related Requirements: 1, 6, 9, 11 **Initiating Actor:** Customer **Actor's Goal:** Create an Account that will be stored in the database and allow for faster entry. **Participating Actors:** User Interface, User Interface Controller, Database **Preconditions:** The User Interface will request all required information. **Postconditions:** The user's account will be stored in the database. **Flow of events for Main Success Scenario:** → 1. Customer accesses the website and chooses the "Register" option. ← 2. The User Interface returns the display that states the required information (Username, Password, Name, Phone Number, Payment Method). → 3. The Customer fills out the required data fields (listed in step 2). ← 4. The User Interface Controller verifies the information provided. → 5. Information is stored into the database and the customer is redirected to the login page. **Flow of Events for Extensions (Alternate Scenarios):** 5a. The information provided was not valid ← Same as in Step 2 above.

Use Case UC-2: Arrive

Related Requirements: 7, 8, 11, 12, 13

Initiating Actor: Customer

Actor's Goal: Get entrance to the parking lot

Participating Actors: System, Database, License Plate Reader, Credit Card Machine
Ticket Machine, Turnstile

Preconditions: Customer arrives in a vehicle with a licence plate and a valid credit/debit card registered to an account.

Postconditions: Customer's vehicle will be registered to his account and the turnstile will rise.

Flow of events for Main Success Scenario:
→ 1. Customer approaches the turnstile.
← 2. The License Plate Reader will scan the license plate and passed it to System
← 3. The license plate is registered to an account, so Turnstile rises

Flow of Events for Extensions (Alternate Scenarios):
← 3a. License plate is not registered so System requests the a payment method from Credit Card Machine
→ 4. Customer will swipe a credit/debit card and Credit Card Machine will pass it to System
← 5. If the provided credit/debit card is registered with a valid account, the car will be registered to that account in the database and System raises the Turnstile. Otherwise a
Temporary guest account will be created.

4a. The provided credit/debit card is invalid
← 1. System detects invalid payment and signals Customer.
→ 2. Customer will swipe a valid credit/debit card
← 3. Same as in Step 5 above.

4b. The customer only has cash
→ 1. Customer presses button on Ticket Machine
← 2. Ticket Machine informs System and prints ticket

← 3. System registers license plate to a temporary cash account
→ 4. Customer takes ticket
← 5. System raises Turnstile

Use Case UC-3: Remove Car

Related Requirements: 16

Initiating Actor: Customer

Actor's Goal: Remove a car/license plate from the account.

Participating Actors: User Interface, User Interface Controller, Database

Preconditions: The Customer has an account with a car listed and is logged in (*all* current account data is loaded).

Postconditions: The user's account will have a car removed.

Flow of events for Main Success Scenario:
→ 1. Customer accesses the website and goes to "Settings" page
← 2. The User Interface displays the customer's account information, including the cars registered to their account.
→ 3. The customer presses the x button next to the car they want to remove.
← 4. The User Interface requests the customer to confirm this action
→ 5. Customer confirms decision
← 6. User Interface Controller removes the car information from database.
Flow of Events for Extensions (Alternate Scenarios):
5a. Customer declines decision
← 6. The User Interface closes the confirmation window.

Use Case UC-4: Delete

Related Requirements: 16

Initiating Actor: Customer

Actor's Goal: Remove account data from the database.

Participating Actors: User Interface, User Interface Controller, Database

Preconditions: The Customer is logged in to their account (*all* current account data is loaded).

Postconditions: The user's account will be removed from the Database.

Flow of events for Main Success Scenario:
→ 1. Customer goes to "Settings" page.
← 2. The system returns the display that states their account information.
→ 3. The Customer presses the "Delete Account" button.
← 4. The User Interface presents a warning that this action is irreversible,
→ 5. Customer confirms decision
← 6. The User Interface Controller logs the user out and requests the data removed from the Database.

Flow of Events for Extensions (Alternate Scenarios):
5a. Customer declines decision
← 6. The User Interface closes the confirmation window and does nothing.

Use Case UC-5: Add/Update User Info

Related Requirements: 14, 15

Initiating Actor: Customer

Actor's Goal: Add a new payment option or update personal info (name, email, car plates).

Participating Actors: User Interface, User Interface Controller, Database

Preconditions: The Customer is logged in to their account (*all* current account data is loaded).

Postconditions: The user's account will have updated info or a new payment option.

Flow of events for Main Success Scenario:
→ 1. Customer accesses the "Payment" page and chooses the "Add Payment" option.
← 2. The User Interface displays a form for the user to fill in.
→ 3. The customer enters payment info.
← 4. The User Interface Controller validates the payment info, if invalid return to step 2 if valid, proceed.
→ 5. Payment information is added to database.

Flow of Events for Extensions (Alternate Scenarios):
→ 1a. Customer accesses the website and chooses the "Settings" option.
← 2. The User Interface displays the current information.
→ 3. The customer presses edit and fills in the fields.
← 4. The User Interface Controller validates new info, if invalid return to step 2 if valid, proceed.
→ 5. User Interface adds the updated information to Database.

Use Case UC-7: Exit

Related Requirements: 6, 7, 8, 9, 10, 11

Initiating Actor: Customer

Actor's Goal: To exit a parking lot.

Participating Actors: System, License Plate Reader, Database, Turnstile, Ticket Machine

Preconditions: The customer has a valid credit card and debit card that was verified on the Arrive UC-2.

Postconditions: The customer has successfully left the parking lot and was charged for their stay.

Flow of events for Main Success Scenario:
→ 1. Customer approaches the turnstile.
← 2. License Plate Reader scans the license plate
← 3. System will find a user account linked to license plate
← 4. System will calculate and execute payment with credit card/debit card linked to user based on current parking lot rate, and save the transaction to the user's history.
← 5. System raises the Turnstile.

Flow of Events for Extensions (Alternate Scenarios):
3a. The license plate is registered to as a guest account
← 4. The system will calculate and execute payment with credit card/debit card linked to guest user based on current parking lot rate.
← 5. The system will delete guest user information from database.
← 6. System raises the turnstile.

3b. The license plate is registered as a cash user
← 4. System requests ticket receipt
→ 5. Customer inserts ticket receipt into Ticket Machine
← 6. System raises the Turnstile

Use Case UC-8: View History of Transactions

Related Requirements: 11

Initiating Actor: Customer

Actor's Goal: To view a history of transactions for the user.

Participating Actors: Database, User Interface, User Interface Controller

Preconditions: The Database contains previous transactions for the Customer's account and the Customer is logged in to their account (*all* current account data is loaded).

Postconditions: The user will have viewed information of the transaction history.

Flow of events for Main Success Scenario:
→ 1. The customer goes on the "History" page.
← 2. The User Interface Controller queries for the Database for the collected history.
← 3. The User Interface displays the data of the collected history.
→ 4. The customer views the transaction history data.

Use Case UC-9: Check Occupancy of Parking Lot

Related Requirements: 2, 3

Initiating Actor: Customer

Actor's Goal: View display of a map to show available parking spots.

Participating Actors: System, Database, User Interface, User Interface Controller, Sensor

Preconditions: The user has an account and successfully logged in (*all* current account data is loaded).

Postconditions: The user will have viewed any necessary parking lot information

Flow of events for Main Success Scenario:
→ 1. The customer goes in the view parking lot map tab.
→ 2. The customer inputs the specific parking lot and level they want to view
← 3. The User Interface Controller accesses the Database for that parking lot
← 4. The User Interface display the parking lot map
→ 5. The customer views the map of the specific parking lot level where they are shown the state of all the parking spaces at the level.

Use Case UC-12: Remove Payment Method

Related Requirements: 17

Initiating Actor: Customer

Actor's Goal: Remove a payment method from the account.

Participating Actors: User Interface, User Interface Controller, Database

Preconditions: The Customer has an account with more than one payment method and is logged in (*all* current account data is loaded).

Postconditions: The user's account will have one payment method removed.

Flow of events for Main Success Scenario:
→ 1. Customer accesses the payment method they want to remove through the "Payment" page.
← 2. The User Interface displays information about the payment method and a remove button
→ 3. The Customer presses the remove button
← 4. The User Interface Controller queries the Database to delete the payment method if there is at least 1 other one

Use Case UC-13: Change Availability of Spots

Related Requirements: 17

Initiating Actor: Management

Actor's Goal: Mark certain spots as unavailable/available in case of emergency of special conditions in the parking system.

Participating Actors: Parking System, Interface Controller, Database

Preconditions: There are spots marked as available in the database and and the administrator has successfully logged in (*all* current account data is loaded).

Postconditions: The desired spots are marked as unavailable (or back to available) to users.

Flow of events for Main Success Scenario:
→ 1. Management accesses to the parking system and chooses the "Change Spot Availability" option.
← 2. The Interface displays the map of all spots and the availability of each.
→ 3. Management marks desired spots as "unavailable" (or "available"), with an option to add a remark displayed to users.
← 4. The Parking Controller marks those spots as unavailable/available in database.

Casual Descriptions for Future Work

Use Case UC-6: Given a recommended Parking Space *(Future Work)*

- Part of the User Application where it recommends the user a suitable parking space by searching for the closest available parking spot to the Ground Level.

Use Case UC-10: Change Price *(Future Work)*

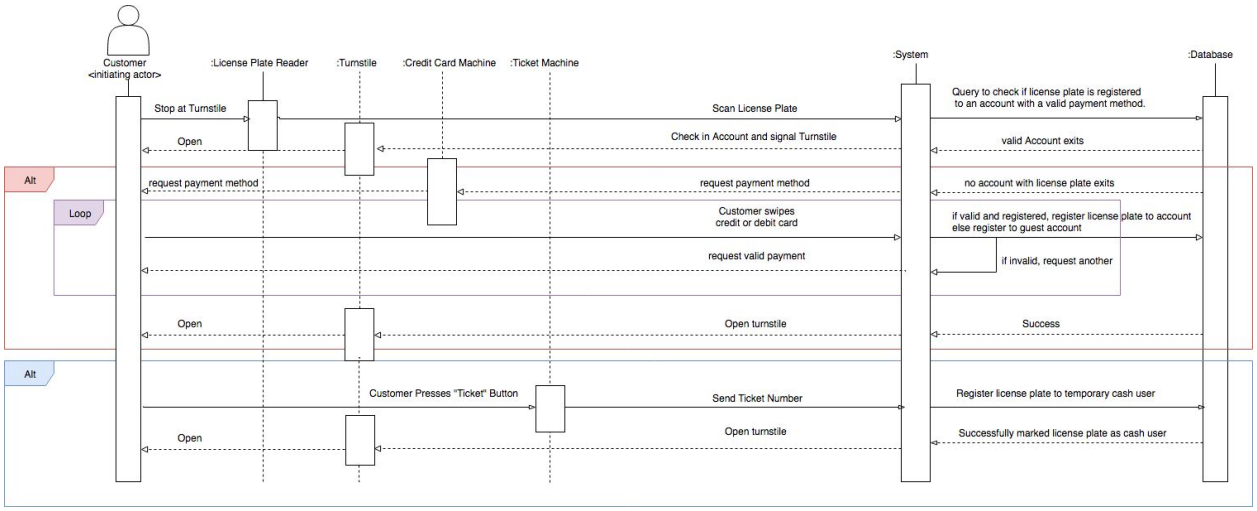
- Management clicks on the edit price button and enters the new price that he/she wants to decide.
- Manager Interface Controller queries the Database to update the price.

Use Case UC-11: Analytics *(Future Work)*

- Management is presented the login page and he/she enters the account credentials
- The credentials are verified by the Manager Interface Controller after it queries the Database
- Management takes in the information displayed and changes the range of information timeline where allowed. This includes sections such as the revenue, average customers, and a map of the parking lot with current availability status.
- If the credentials are deemed incorrect by the Manager Interface Controller, an error message is displayed by the Manager Interface.

System Sequence Diagram

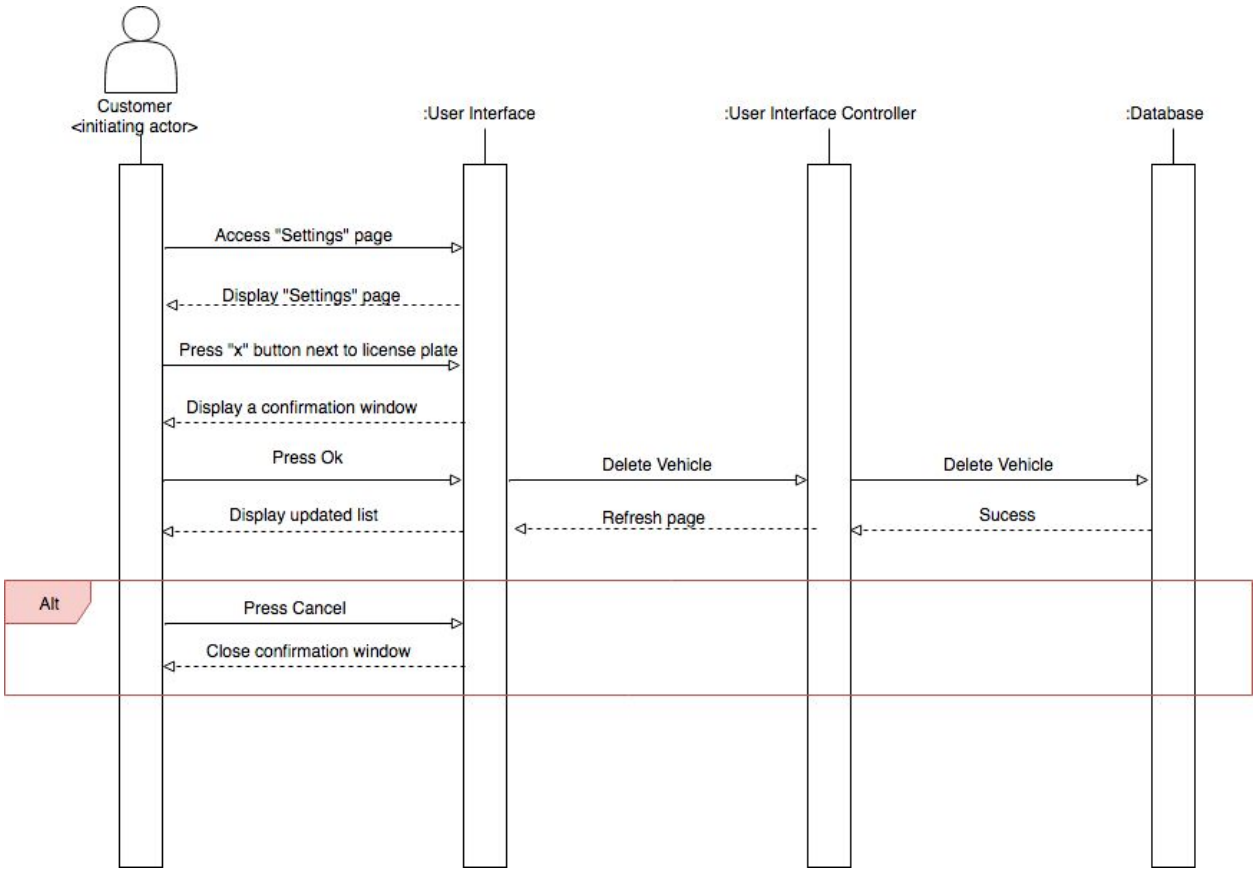
Use Case 2 - Arrive



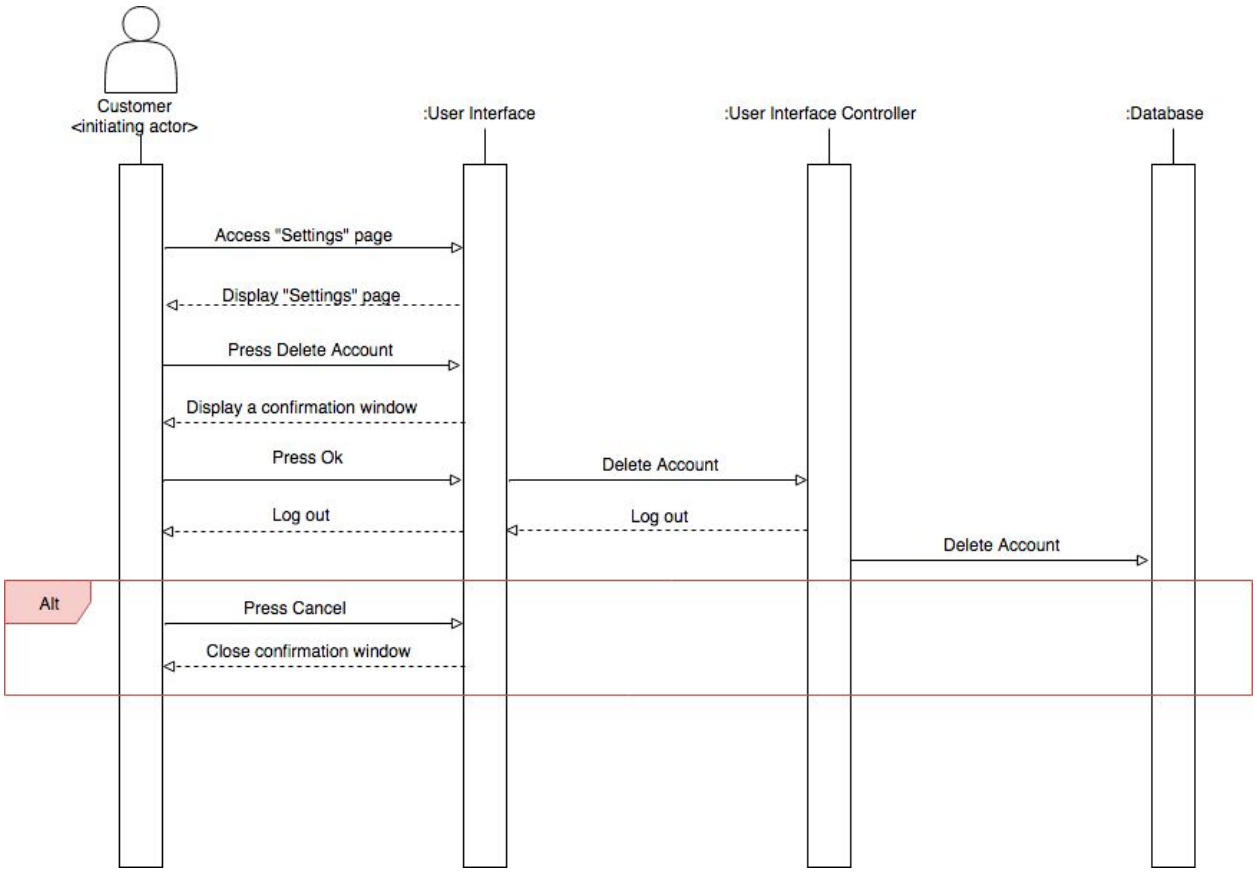
Please note that the diagrams are difficult to scale into the pdf. Please look here: https://github.com/Vatsal09/Garage-Automation/blob/master/Diagrams/arrive.png

This design followed our layered architecture approach. The application serves as the controller between the system and the database. This establishes a clear responsibility for each object: The System collects data, the Database stores data, and the Application transfers and manipulates data.

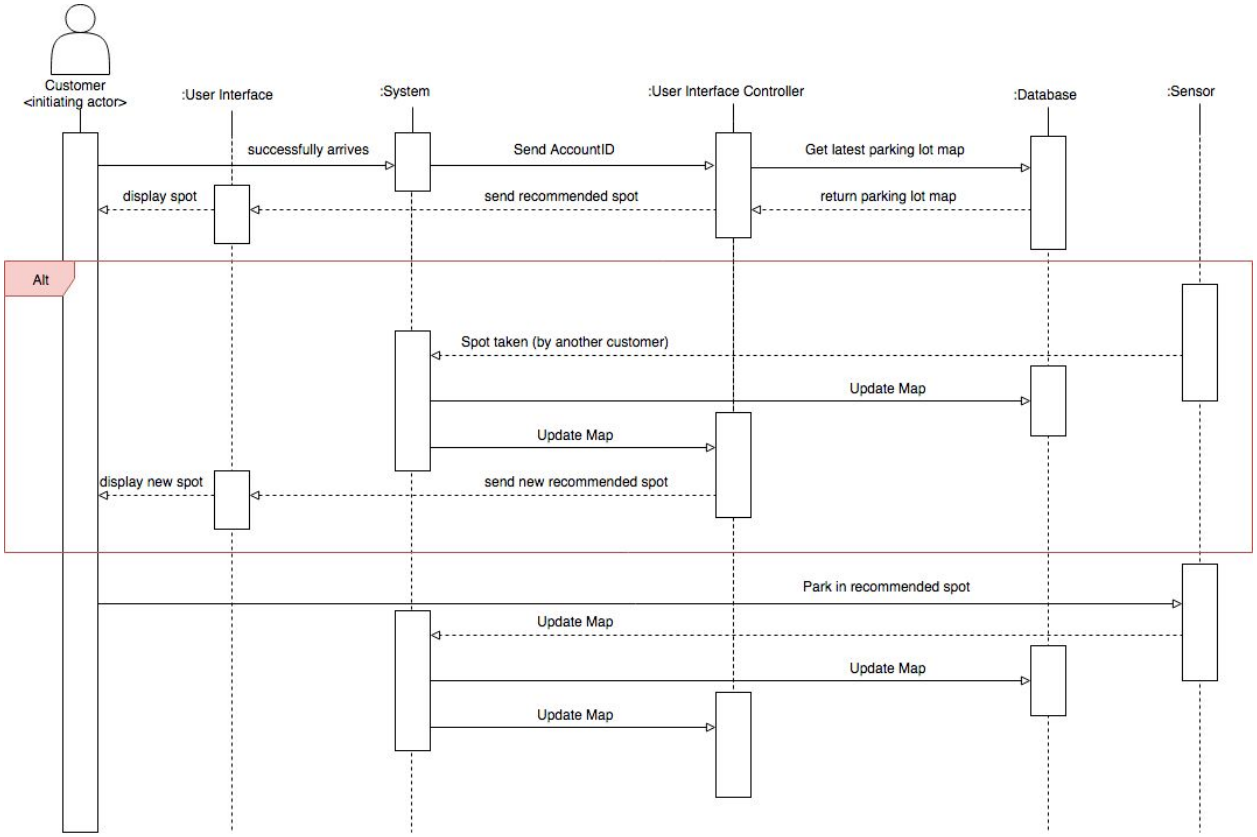
Use Case 3 - Remove Car



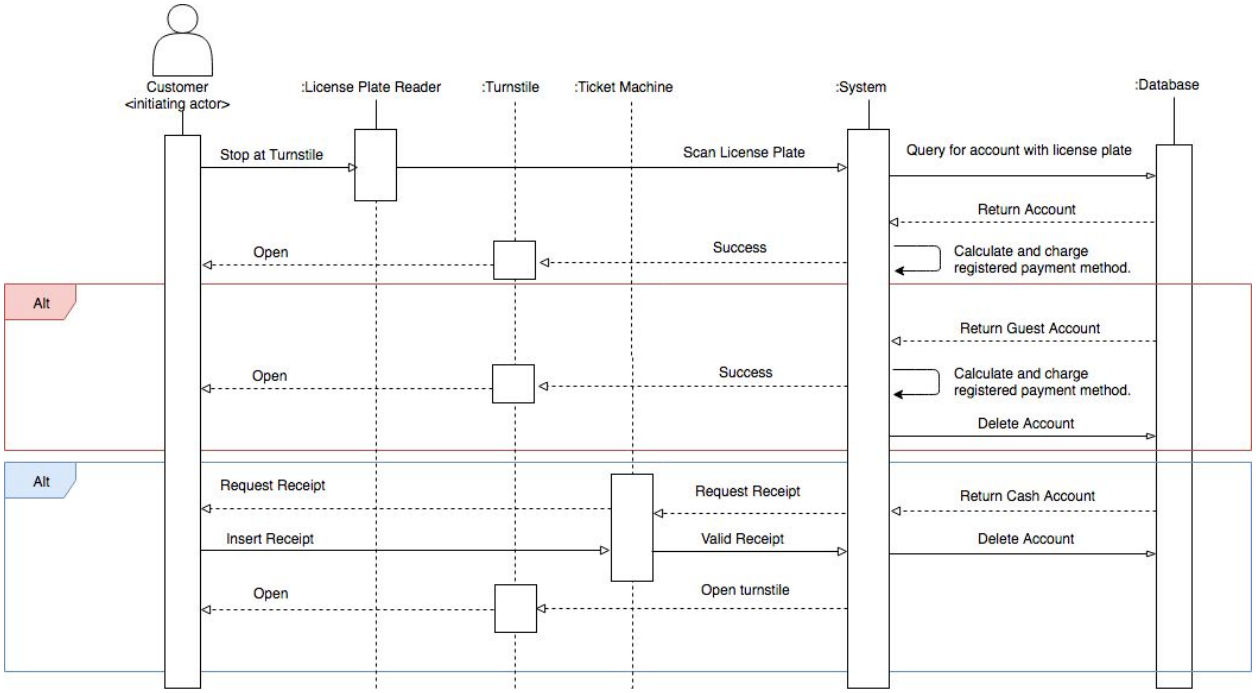
Use Case 4 - Delete Account



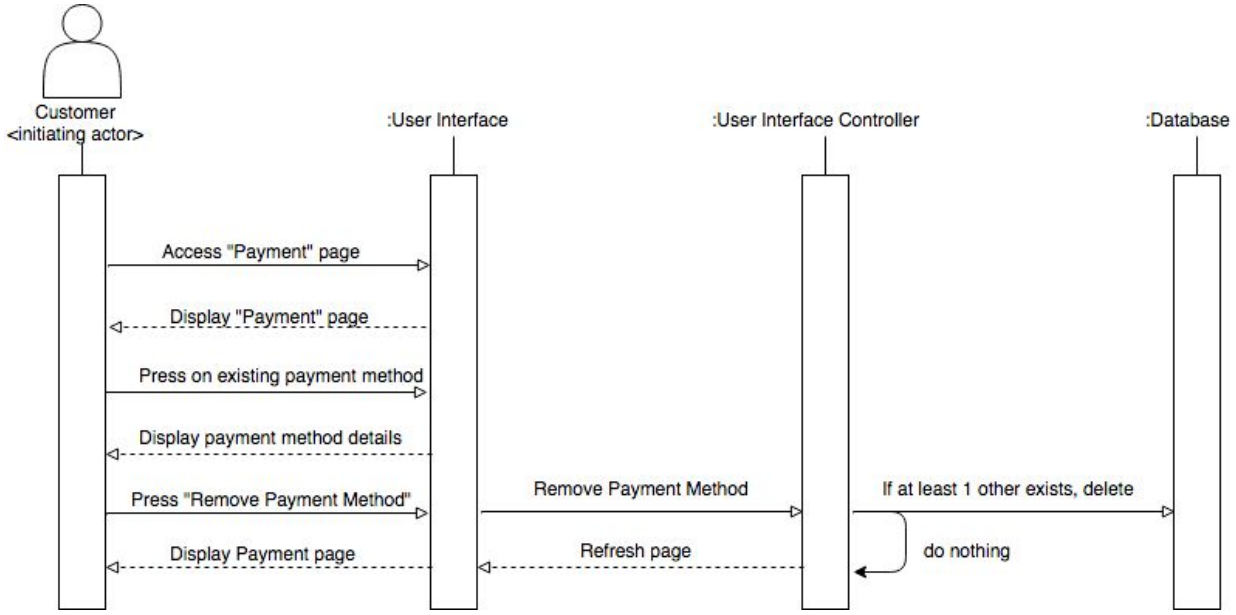
Use Case 6 - Give a Recommended Parking Space (Future Work)



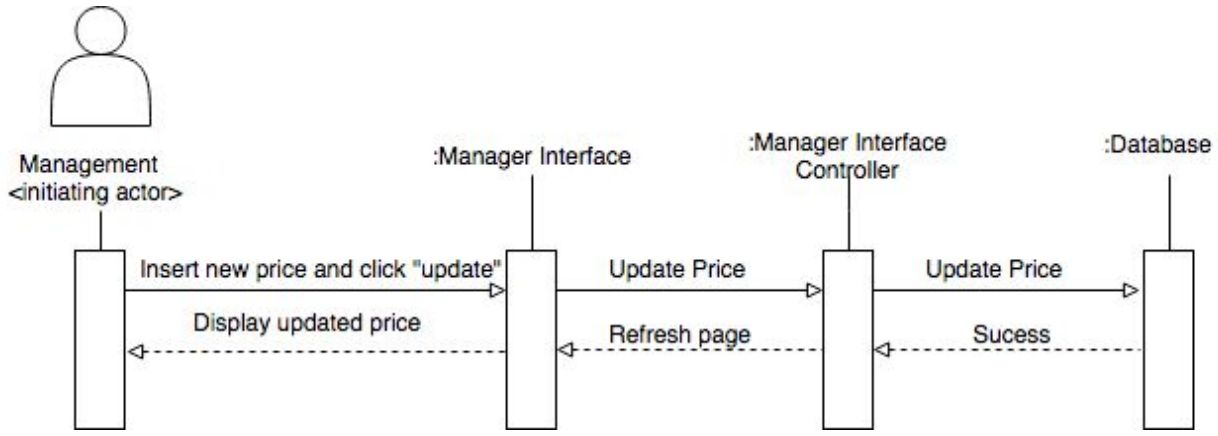
Use Case 7 - Exit



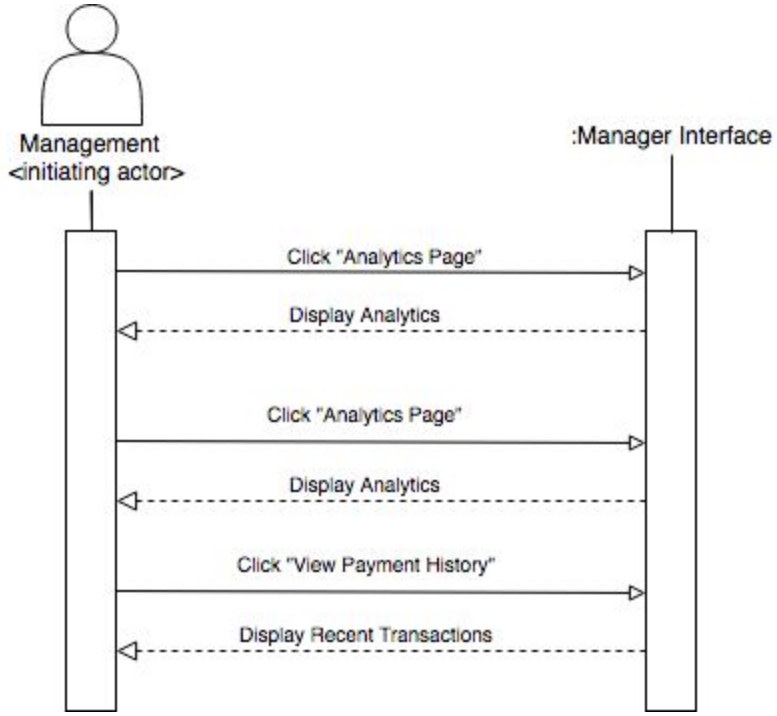
Use Case 12 - Remove Payment Method



Use Case 10 - Change Price (Future Work)



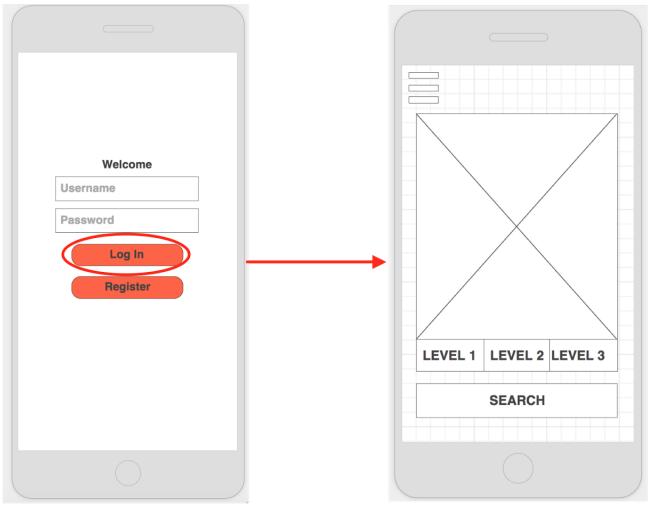
Use Case 11 - Analytics (Future Work)



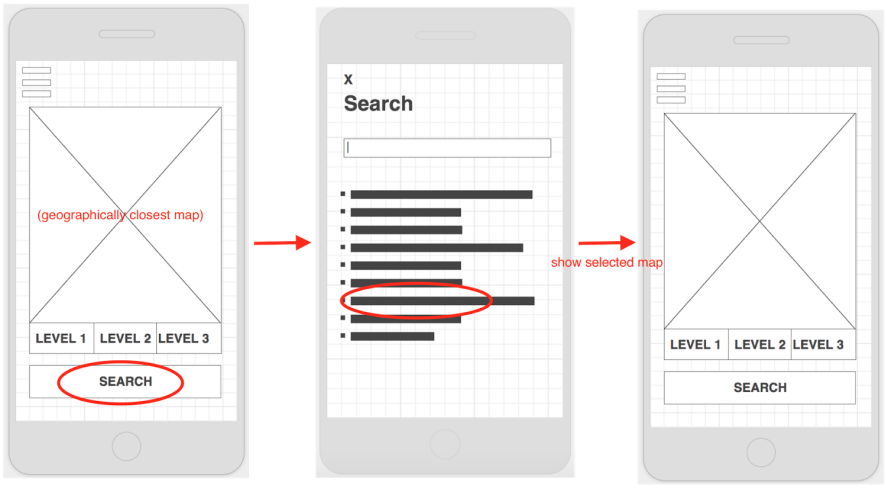
User Interface Specification

User Preliminary Design

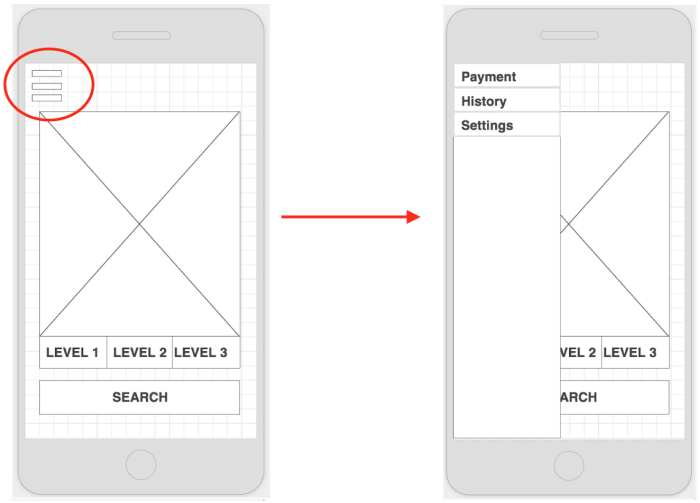
Application Functionality



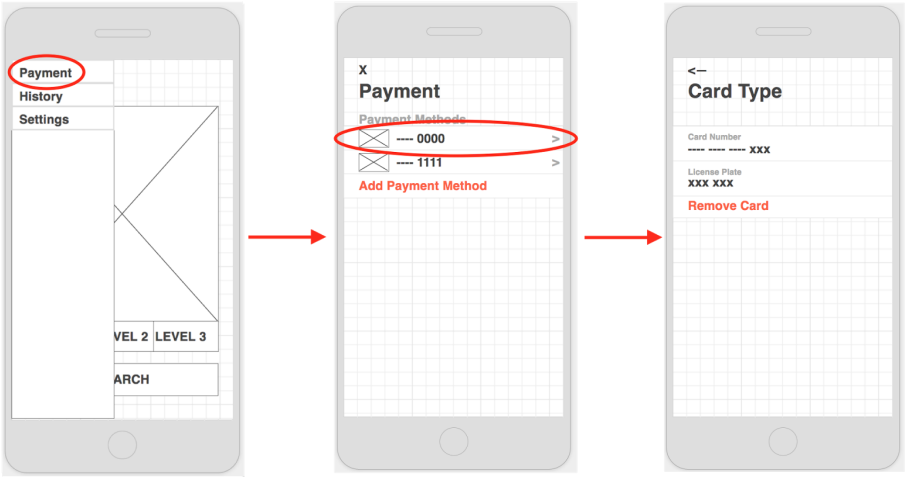
The user can log in or register from the Welcome page. When registering, the user will be required to provide a payment method. Then, will be directed to the home page. The home page displays the map of the geographically closest parking lot.



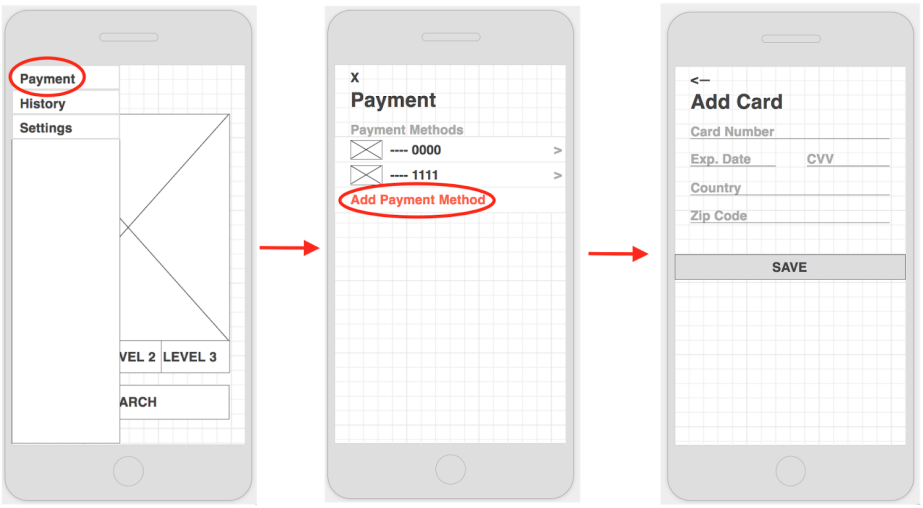
Pressing the Search Button brings up a search window. Here the user can search for a specific parking lot and select it from a list. Then they will be returned to the home page, where the selected parking lot's map will be displayed.



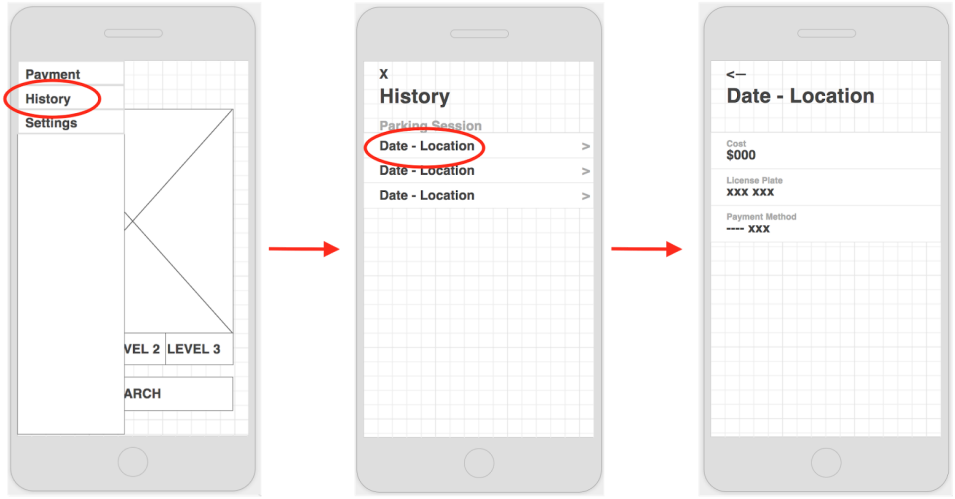
Pressing the menu button will bring up Payment, History, and Settings buttons



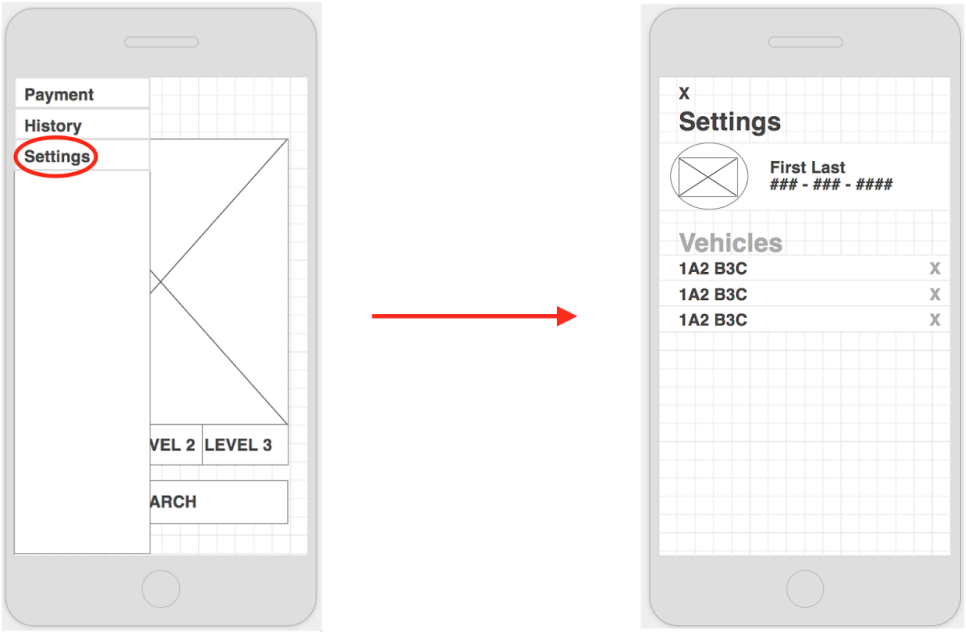
Pressing the payment button brings up the payment screen. Here the user has the option to review and remove payment methods. However, they must always have at least one valid payment method on their account.



To add a payment method, the user simply presses the Add Payment Method button and fills in the necessary credentials.



From the menu, the user can also access their history. The History page displays all of their previous parking sessions. Pressing on a specific session will bring up additional information, such as the cost, the licence plate they drove in with, and the payment method that was charged.



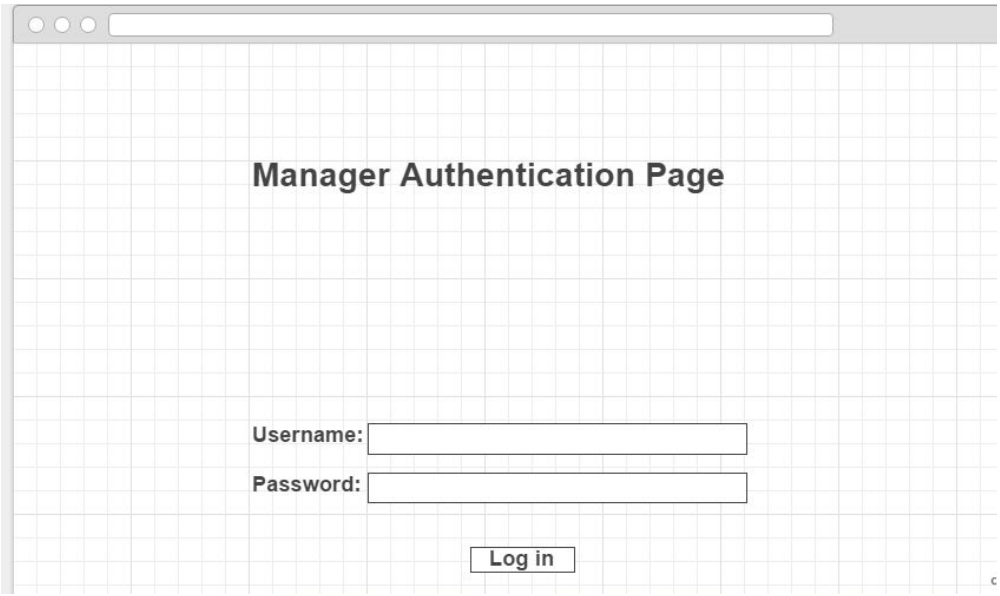
The settings page contains information regarding the user. This includes their name, phone number, and the license plates registered to their account. To remove a license plate from an account, the user presses the X next to the license plate.

If the user no longer wishes to pay for a vehicle's parking, they must remove it from their account. A vehicle cannot be removed if it is currently checked into a parking lot.

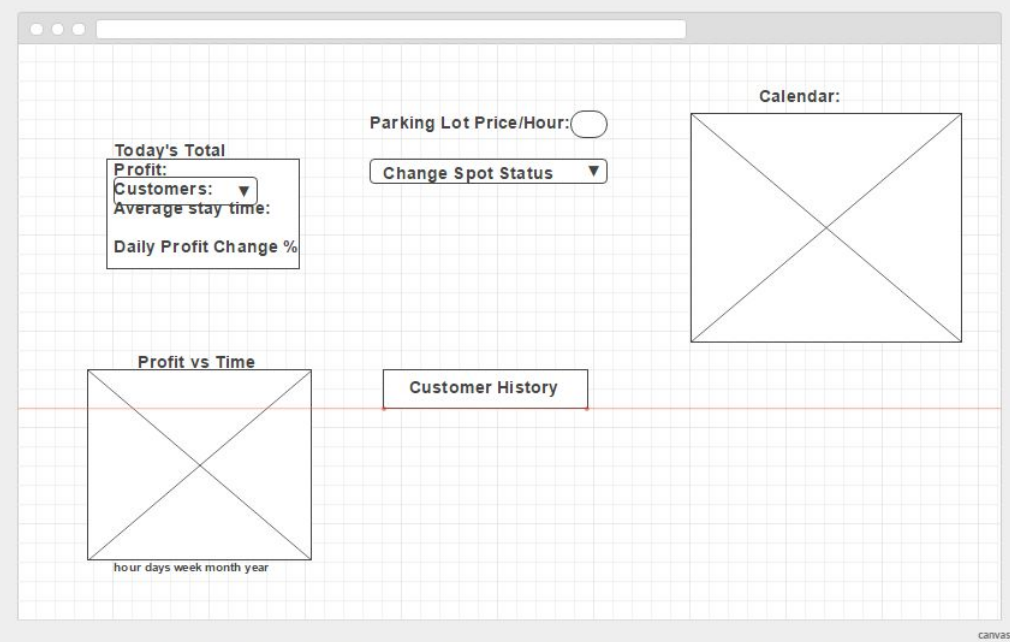
A license plate can only be registered to a single account, this is to avoid any confusion regarding who should be charged. To avoid any malicious abuse of this restriction, the user cannot manually add vehicles to their account.

The system will handle registration of a vehicle upon arrival. When a customer arrives with an unregistered license plate, they will be asked to provide a credit or debit card. If this payment method is registered to an account, the license plate will be added to that account for future visits.

Manager Dashboard (Future Work)

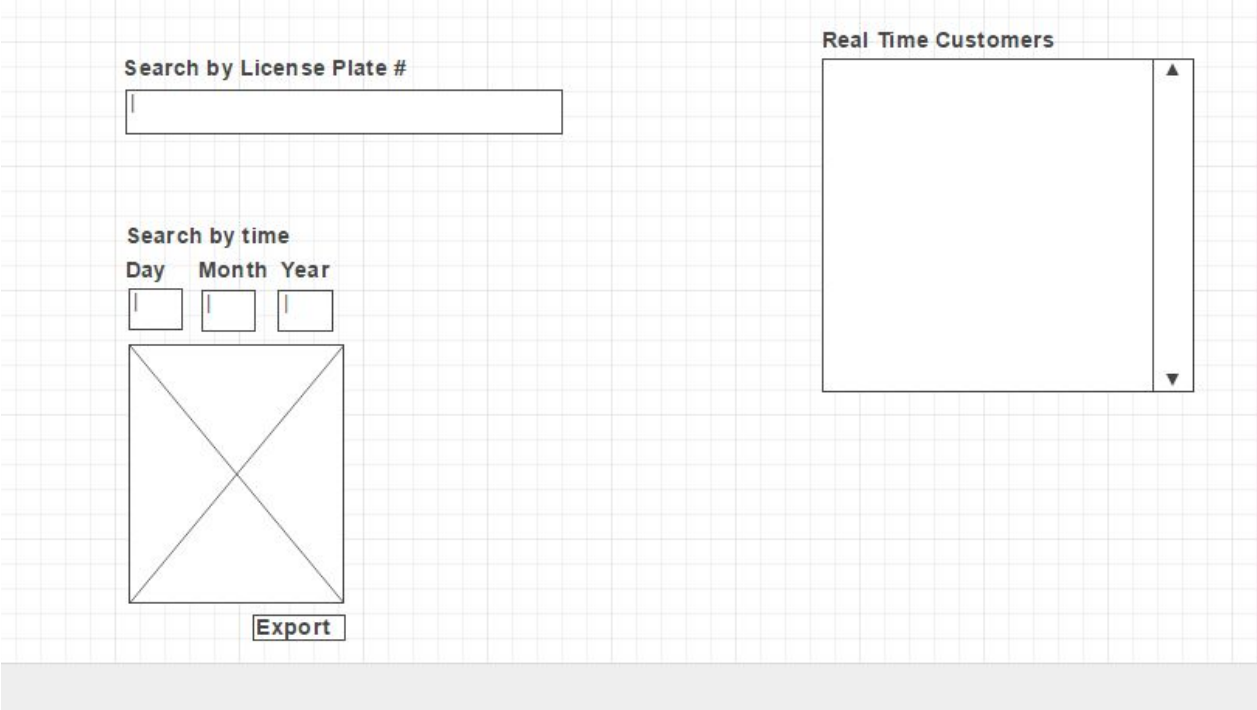


The first page prompts the manager to input the confidential and verify the identity. After successful login, the second page is shown.



In the second page, the manager is able to view the statistics of revenue earned and summary of customer behaviors (e.g. average stay time) for the current day on the top left box. The manager can set price on the top middle of page, where followed by a

option bar to change spot status. On the right shows the calendar. Graphs of data will be displayed on the left bottom. There is also a button for manager to click, which navigates to the third page.



On the third page, the manager can input license plate # (or user account #) to view payment history of that specific user. An alternate way of searching is to type date and time on the left bottom to export all payment records at that time. On the right displays the real-time information of spots and customers.

Effort Estimation Using Use Case Points

---

1. Unadjusted Actor Weight (UAW)

Actor Name	Complexity	Weight
Management	Complex	3
Customer	Complex	3
User Interface	Simple	1
Database	Average	2
License Plate Reader	Average	2
Vacancy Display	Simple	1
Turnstile	Simple	1
Sensors	Simple	1
Credit Card Machine	Simple	1
System	Average	2
User Interface Controller	Simple	1
Manager Interface Controller	Simple	1
Ticket Machine	Simple	1

UAW = 8 x Simple + 3 x Average + 2 x Complex = 8*1 + 3*2 + 2*3 = 20

2. Unadjusted Use Case Weight (UUCW)

Use Case	Category	Weight
UC-1 - Register	Simple	5
UC-2 - Arrive	Complex	15
UC-3 - Remove Car	Simple	5
UC-4 - Delete Account	Average	10
UC-5 - Add/Update User Information	Simple	5
UC-6 - Recommended Spot	Complex	15
UC-7 - Exit	Complex	15
UC-8 - History	Simple	5
UC-9 - Check Occupancy	Average	10
UC-10 - Change Price (future work)	Simple	(5)
UC-11 - Analytics (future work)	Average	(10)
UC-12 - Remove Payment Method	Simple	5
UC-13 - Change Spot Availability	Complex	15

UUCW = 6 x Simple + 3 x Average + 4 x Complex = 5*5 + 2*10 + 4*15 = 105

UUCW of future work = 1 x Simple + 1 x Average = 15

UUCP = UAW + UUCW = 22 + 120 = 142

3. Technical Complexity Factor (TCF)

Technical Factor	**Description**	**Weight**	**Perceived Complexity**	**Calculated Factor (Weight x Perceived Complexity)**
T1	Complex internal processing	1	2	1*2 = 2
T2	Easy to install	0.5	0	0.5*0 = 0
T3	Distributed, Web-based system	2	5	2*5 = 10
T4	Ease of use	0.5	3	0.5*3 = 1.5
T5	Concurrent use	1	3	1*3 =3
T6	Security requirements	1	4	1*4 = 4
T7	End-user efficiency	1	3	1*3 = 3
T8	Ease of change	1	2	1*2 = 2
T9	Reusable design	1	1	1*1 = 1
T10	Consistent performance	1	5	1*5 = 5
T11	Portability	2	0	2*0 = 0

TCF = Constant-1 + Constant-2 x Technical Factor =
= 0.6 + 0.01 x (2 + 0 + 10 + 1.5 + 3 + 4 + 3 + 2 + 1 + 5) = 0.915

3. Environmental Complexity Factor (ECF)

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor (Weight x Perceived Impact)
E1	Learning new framework Django	-1	3	-1x3 = -3
E2	Beginner familiarity with UML-based development	1.5	1	1.5x1 = 1.5
E3	Stable requirements expected	2	4	2x4 = 8
E4	Learning version control (Github)	-1	2	-1x2 = -2
E5	Report writing Skills	3	2	3x2 = 6
E6	Highly motivated but lose one team member	1	4	1x4 = 4
E7	Some knowledge of lead analyst	0.5	2	0.5x2 = 1
E8	Familiarity of application problem	0.5	3	0.5x3 = 1.5

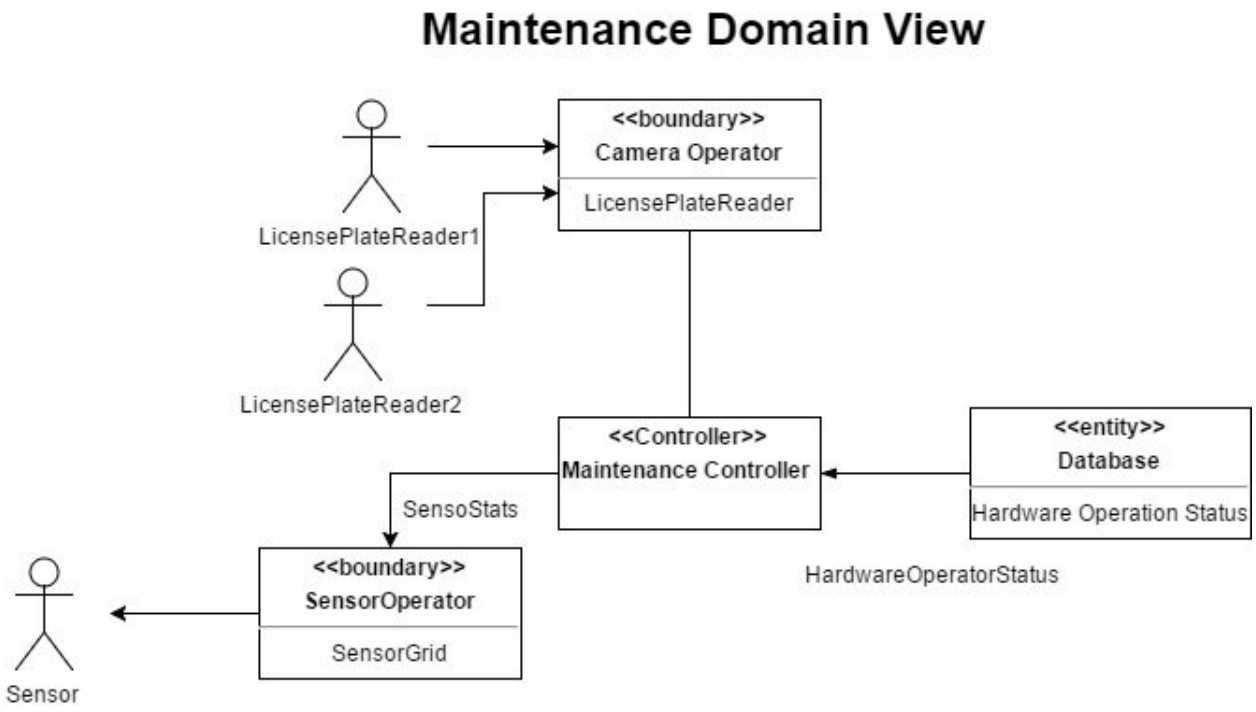
ECF = Constant-1 + Constant-2 x Environmental Factor =
= 1.4 - 0.03 x (-3 + 1.5 + 8 + -2 + 6 + 4 + 1 + 1.5) = 0.89

Use Case Points (UCP)

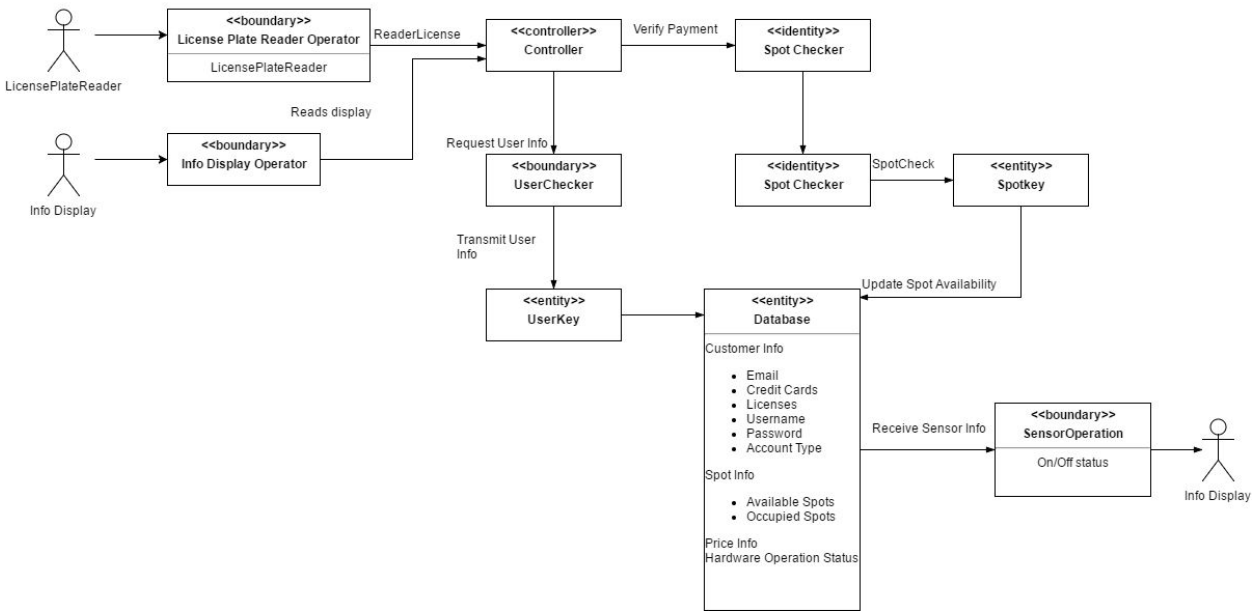
UCP = UUCP x TCF x ECF = 142 x 0.915 x 0.89 = 115.7

Domain Analysis

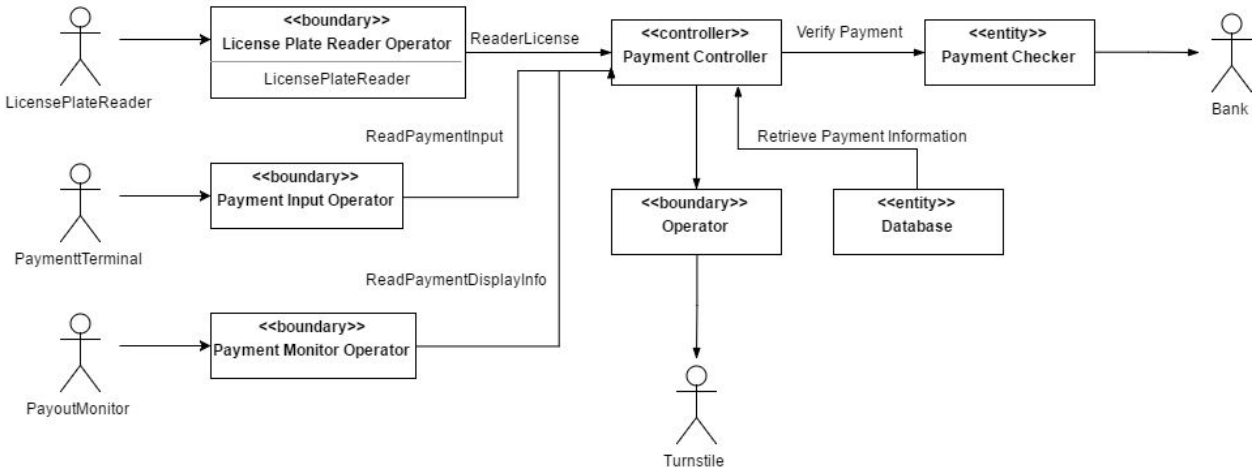
Original Domain Models from Report#2



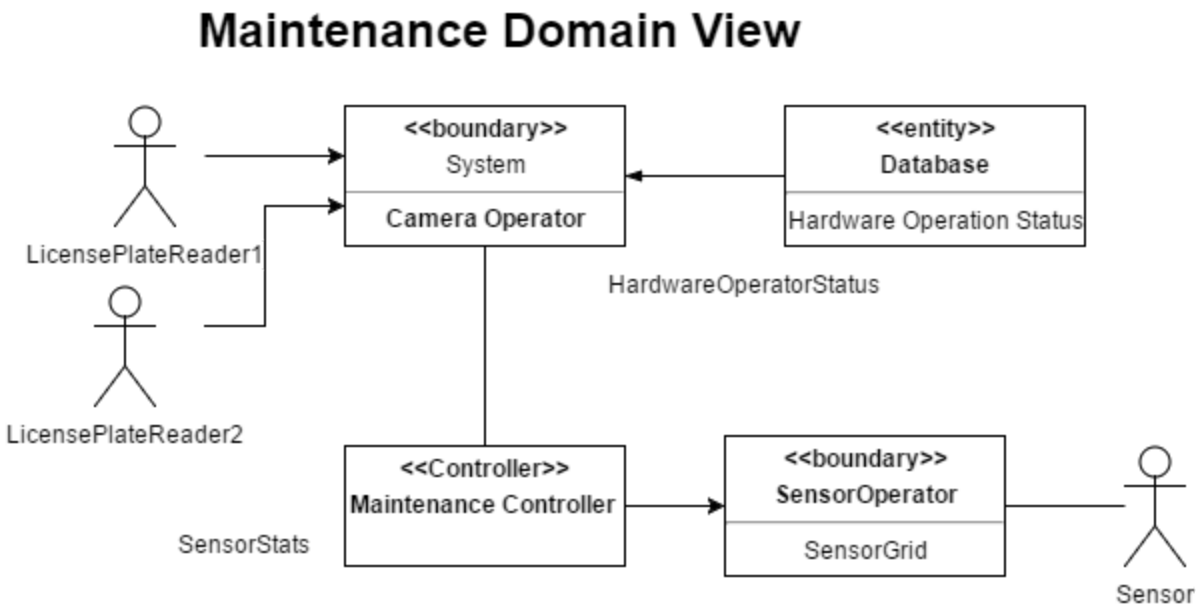
Park Domain View



Exit Domain View

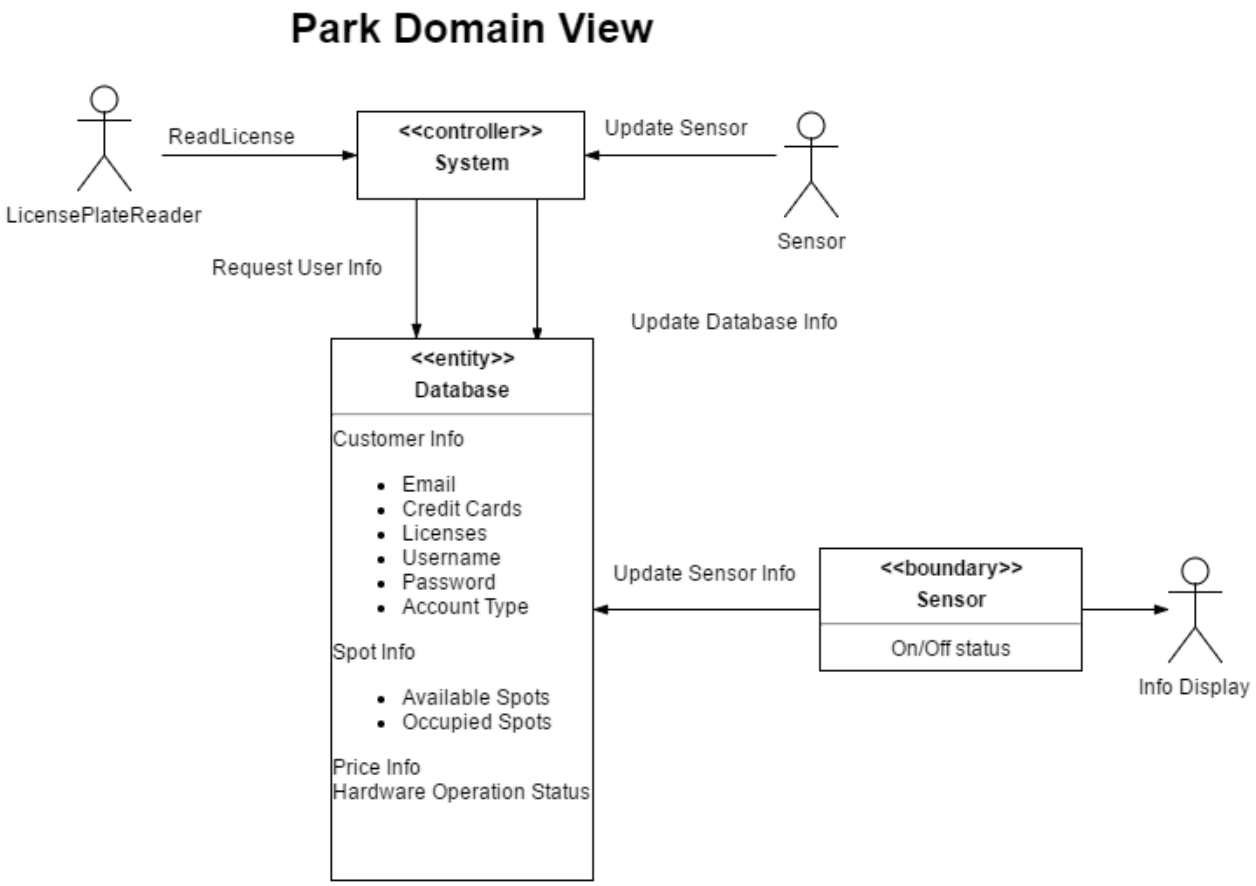


Revised Domain Models



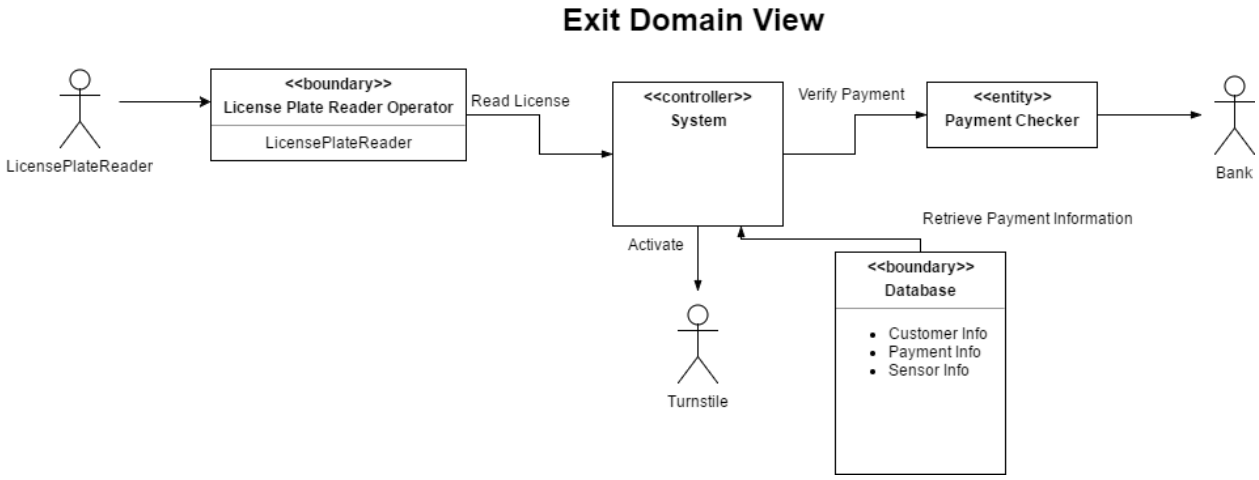
Notes of Changes:

The new Maintenance Domain View Model alters the time point - when the parking system adds the event of parking to the database. The time of customer's arrival and leaving should be calculated by the moments they enter and leave the gate, where the Camera Operator records the parking behaviors into the database. This method prevents customers from moving their vehicles to different spots inside the parking garage and confusing the system on whether they have left. That is a better approach compared to calculating customer's stay time based on the sensor at each spot in report#2.



Notes of Changes:

The new Parking Domain View Model is simplified compared to the one in Report#2, since the Info Display Operator is removed at the garage. Instead, we make it into a part of the customer's interface which is no longer included in the parking domain. In the parking domain, as soon as customers park their car in at the spots, the sensors will return the vehicle information, - the license plate #, and the spot information, - the spot IDs and statuses. The information from two aspects will be linked together in the back-end database, so that each spot is matched with each parking customer, which is saved as parking history in the database.



Notes of Changes:

To fully implement the automatic system, we decide to remove the Payment Input Operator and Payment Monitor Operator at the exits for those registered customers. Since the system already has their payment methods, their payments will be automatically processed in the back-end once the license plate reader at the one-way exit reads their license plate. It no longer needs a monitor in Report#2 to ensure that the customer leaves, as the license plate reader already solves the problem. In the meantime of processing their transactions, the turnstile opens, and a set of information is recorded in the database, including customer's leaving time, payment method they are using, along with the customer's account information.

I. Concept definitions

Responsibility Descriptions	Type	Concept Name
Reads inputs from the sensors to determine occupancy of each spot and informs the SensorChecker. If contradicting inputs are read in from particular spots dual sensors, a signal will be sent to the MaintenanceLogger to record the problem.	K	SensorOperator
Reads input from SensorOperator to update the FreeSpotKey.	D	SensorChecker
Reads input from LicensePlateReaderOperator and UserChecker. It determines whether or not to admit entry to the User. It uses the input form UserChecker and works with RecommendController to give a registered user a recommended parking spot.	D	TurnstileController
Outputs commands to the Turnstile	D	TurnstileOperator
Reads input from the License Plate Readers and communicates with the TurnstileController for arrivals and the PaymentController for exits.	D	LicensePlateReaderOperator
Serves to interact between the PaymentController to confirm session info has been logged and payment has been appropriated so the user can exit.	D	PaymentChecker
Receives request from the TurnstileController to obtain the type of customer that has arrived. It then accesses UserKey to determine the type of customer from the profiles where it will start a log for the session. It will also create a guest profile in the UserKey if the customer is not registered for a spot.	D	UserChecker
Receives input from LicensePlateReader of user approaching to exit. This plate number is then sent to the PaymentChecker to determine amount owed. Once payment criteria is confirmed through the PaymentChecker, the PaymentController will tell the GateOperator to open the gate.	D	PaymentController
Serves to interact between the PaymentController and the UserKey to confirm session info has been logged and payment has been appropriated so the user can exit.	D	PaymentChecker
Reads the payment received from the Payment terminal in the event a customer must pay before being allowed to exit and sends it to the Payment Controller.	D	PaymentOperator

Intermediary between Interface Page and Pagemaker	D	WebPageController
Gathers information to be displayed on the interface	D	PageMaker
Acts as the GUI between the system and the User	K	InterfacePage
Stores all pertinent variable information of the system	K	Database
Used to temporarily store all user info that it retrieved from the database for the UserChecker until the User is processed.	K/D	UserKey
Container for current number of parking spots available on each floor and where their subsequent location is, collaborates with the Database.	K	FreeSpotKey
Reads input from Sensor Operator to determine the state of all the parking spots. Reports to FreeSpotKey all the spots that are open.	D	SpotChecker

Revision of Concept Definitions:

- Removed RecommendController from our domain models because sending the user a recommendation spot is for future work.
- Removed PaymentMonitorOperator because we did not integrate a display system into our final product.
- Removed MaintenanceLogger because we did not integrate a system that recognizes abnormalities in hardware and notifies the system.

II. Association definitions

Concept Pair	Association Description	AssociationName
SensorChecker - SensorOperator	SensorChecker obtains sensor readings from SensorOperator in order to update spot availability (FreeSpotKey)	ReadSensor
TurnstileController - SpotChecker	TurnstileController obtains data from SpotChecker to check for spot availability (from FreeSpotKey)	RequestSpotInfo
TurnstileController - TurnstileOperator	TurnstileController sends command to TurnstileOperator to open turnstile and waits for execution confirmation	DecideAvailabilty
LicensePlateOperator - TurnstileController	TurnstileController reads information from LicensePlateOperator as a User pulls up to park	ReadsPlateInfo
PaymentController - LicensePlateOperator	PaymentController reads license number from LicensePlateOperator as Users pull up to attempt to exit.	ReadExitLicense
PaymentController - PaymentOperator	If it is a guest customer arrives and swipes their credit card at the turnstile, the PaymentOperator will send the payment info that it receives to the PaymentController.	ReadsPaymentInput
PaymentController - PaymentMonitorOperator	PaymentController will send the amount due to the PaymentMonitorOperator	ReadPaymentDisplay
UserChecker - UserKey	UserChecker asks the UserKey to retrieve any User info that is saved based on its information.	TransmitUserInfoRequest
UserKey - Database	UserKey loads any info pertaining to the request that is found in the Database	RetrieveUserInfo
PaymentController - PaymentChecker	PaymentController sends the license info of who is attempting to exit to the PaymentChecker. The amount owed that is	VerifyPayment

	returned is then sent to be displayed on the PaymentMonitor. Any payment that is received from the Payment Operator is verified by the PaymentChecker before the User can exit.	
PaymentChecker - PaymentKey	PaymentChecker will receive the amount due from PaymentKey and will update it if payment is received	RequestPaymentInfo
PaymentKey - Database	PaymentKey accesses info based on the info it receives from PaymentChecker from the Database	GetAmountDue
SpotChecker - FreeSpotKey	SpotChecker consults the FreeSpotKey to check for available spots	OpenSpotCheck
SensorChecker - FreeSpotKey	FreeSpotKey receives the input from the sensors through the SensorChecker	UpdateSpotKey
FreeSpotKey - Database	FreeSpotKey and Database keep each other synchronized	UpdateSpotAvailability
PaymentController - TurnstileOperator	PaymentController will send the signal to the TurnstileOperator to lift the gate once payment is confirmed to have been appropriated.	ExitSuccess
WebpageController - InterfacePage	WebpageController receives info from the InterfacePage and sends request to the PageMaker to build a response	WebPosts
InterfacePage - PageMaker	InterfacePage reads the information collected by the PageMaker	PreparesWebPage
PageMaker - Database	PageMaker queries database for pertinent information	ProvidesWebData
PageMaker WebPageController	PageMaker receives the request from the WebPageController	ConveysWebRequests
FreeSpotKey - RecommendController	Gives User a recommended from FreeSpotKey	GivesRecommendedSpot

Revision of Association Definitions:

- Removed ReadPaymentDisplay because we removed PaymentMonitorOperator due to it not being in the scope of our project.
- Removed SensorStatus, LicensePlateReaderStatus, and HardwareOperationStatus because we removed MaintenanceLogger due to it not being in the scope of our project.

III. Attribute definitions

SensorID	Assigned to map a sensor with a ID number
LicensePlateNumber	String of characters that represent the license plate information taken from the License Plate Readers
errorMessage	String pertaining to the type of error with the hardware
password	...
userID	...
SessionOwed	Amount owed that is taken every parking session unless they are a guaranteed customer during the extent of their contract time
TotalOwed	Total of all SessionOwed-SessionPaid
SessionPaid	Total amount paid per session
EntranceTurnstileStatus	present status of entrance turnstile
ExitTurnstileStatus	present status of exit turnstile
FreeSpots	Bitmap of spot availability
CreditData	Record of credit card information for guest users who swiped at the entrance and will later get charged at the exit
OpenSession	Time entered and pending duration of current stay

IV. Traceability Matrix

UC	1	2	3	4	5	6	7	8	9	12
PW	5	23	5	13	5	10	15	5	17	5
Domain Concept										
SensorOperator		X				X	X		X	
SensorChecker		X				X	X		X	
TurnstileController		X					X			
TurnstileOperator		X					X			
LicensePlateReaderOperator		X					X			
UserKey		X				X	X			X
PaymentChecker		X					X			
UserChecker		X				X	X			X
FreeSpotKey		X				X	X		X	
SpotChecker		X				X	X		X	
PaymentController										
PaymentChecker		X					X			
PaymentOperator		X					X			
WebPageController	X		X	X	X	X		X	X	X
PageMaker	X		X	X	X	X		X	X	X
InterfacePage	X		X	X	X	X		X	X	X
Database	X	X	X	X	X	X	X	X	X	X

V. Mapping Domain Concepts to Use Cases

Use Case 1 - Registration of User's Account corresponds to WebPage Controller, PageMaker and Interface Page, which render the registration page that prompts the new customers to input their personal info as well as payment info. The collected credentials are passed into the Database Domain.

Use Case 2 - Arrival of Vehicles corresponds to all the domains involving sensors, turnstiles, payments, spots and the database. The License Plate Reader Operator records customer's arrival as it catches the license info of the vehicle. Turnstile Operator and Turnstile Controller then open the turnstile to allow the entrance of the vehicle. UserKey and User Checker basically check the customer's identity, that is, whether the customer is already a registered user. If no, the customer will have to use the ticket machine for payment. Otherwise, the account info will pass to the Payment Checker and the Payment Operator Domain , which check if the payment methods are valid and prepare to process the payment. Spot Checker returns the availability of each spot to the Database. Free Spot Key Domain fetches the info of currently available spots from the Database Domain.

Use Case 3 - Removal of Vehicles allows customers to remove the vehicles that are scanned by the license plate reader and linked to their accounts. This involves with the domains of WebPage Controller, PageMaker and Interface Page that render the interface for users to do that. The operation will be passed to the Database Domain so that the changes are processed and stored.

Use Case 4 - Delete User' Account also corresponds to WebPage Controller, PageMaker, Interface Page and Database Domains. The process is similar with Use Case 3 - deleting vehicles.

Use Case 5 - Add/Update User Information, similarly, involves with the domains of WebPage Controller, PageMaker, Interface Page and Database. The user inputs or changes its personal info and the updated info will be passed to the Database Domain.

Use Case 7 - Exit maps to the same domains that are corresponded to the Use Case 2 - Arrival. The License Plate Reader Operator records customer's leaving as it catches the license info of the vehicle at the exit. Turnstile Operator and Turnstile Controller then open the turnstile to allow the exit of the vehicle. UserKey and User Checker basically check the customer's identity, that is, whether the customer is already a registered user. If no, the customer will have to use the ticket machine for payment. Otherwise, the

account info will pass to the Payment Checker and the Payment Operator Domain , which check if the payment methods are valid and immediately process the payment as they customer leaves. Spot Checker updates the new availability of each spot to the Database. Free Spot Key Domain fetches the info of currently available spots from the Database Domain. In the meantime, the Payment Operator Domain will pass the transaction into the parking history stored in Database Domain for future queries.

Use Case 8 - Parking History maps to WebPage Controller, PageMaker, Interface Page and Database Domains. WebPage Controller, PageMaker and Interface Page domains render the web page that display the user's parking and payment history, and the history is imported from the stored information in Database Domain.

Use Case 9 - Check Occupancy also corresponds to WebPage Controller, PageMaker, Interface Page and Database Domains, which render the list of occupancy of all the spots for the user's information. The occupancy info is stored in the Database Domain, which is returned by the Sensor Operator and Sensor Checker Domains. As WebPage Controller gets the user's request, the Sensor Domains pass the occupancy info into Database Domain, and the database passes the occupancy to PageMaker and Interface Page for rendering.

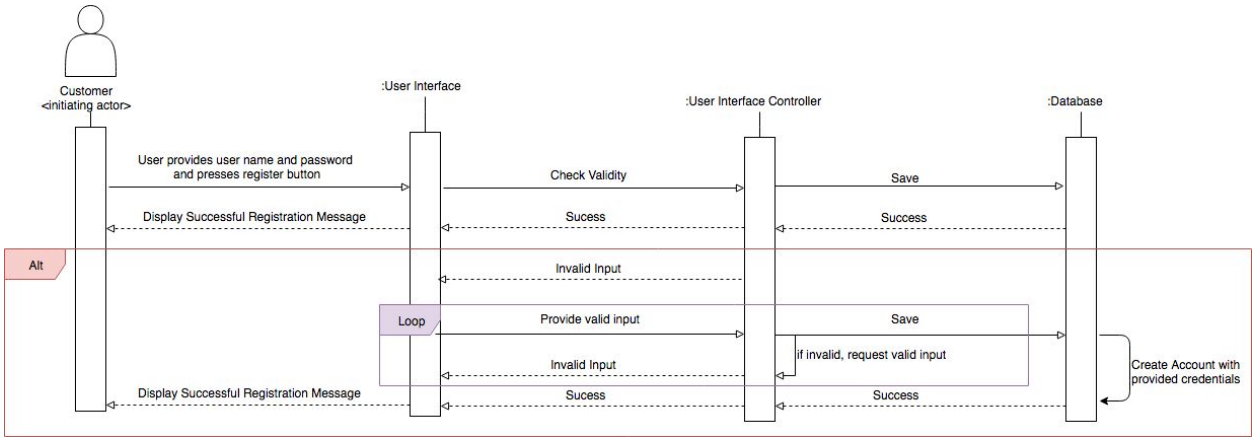
Use Case 12 - Removal of Payment Method is processed by the domains of WebPage Controller, PageMaker, Interface Page, Database, UserKey and User Checker. PageMaker and Interface Page render the interface where the user can choose which payment method that they wish to remove. The user's identity is verified by the domains of UserKey and User Checker, which import user info stored in the Database Domain. After the customer selects removal of payment method, the WebPage Controller passes the request to Database, so that the old payment method is permanently removed in the database.

VI. System Operation Contracts

Operation	**Exit**
Preconditions	● License plate is read by the exit license plate reader and sent to the PaymentController by the LicensePlateOperator. ● PaymentController sends license plate information to UserChecker to identify the user to determine payment information in the UserKey. ● PaymentController sends payment request to the PaymentOperator to obtain payment if applicable PaymentController confirms payments have been appropriated then sends message to TurnstileOperator to open exit gate. ● PaymentController sends message to update the UserKey of payments and the time the session has ended through the PaymentChecker
Postconditions	● TurnstileOperator is instructed to close turnstile by the PaymentController once the car has exited the garage

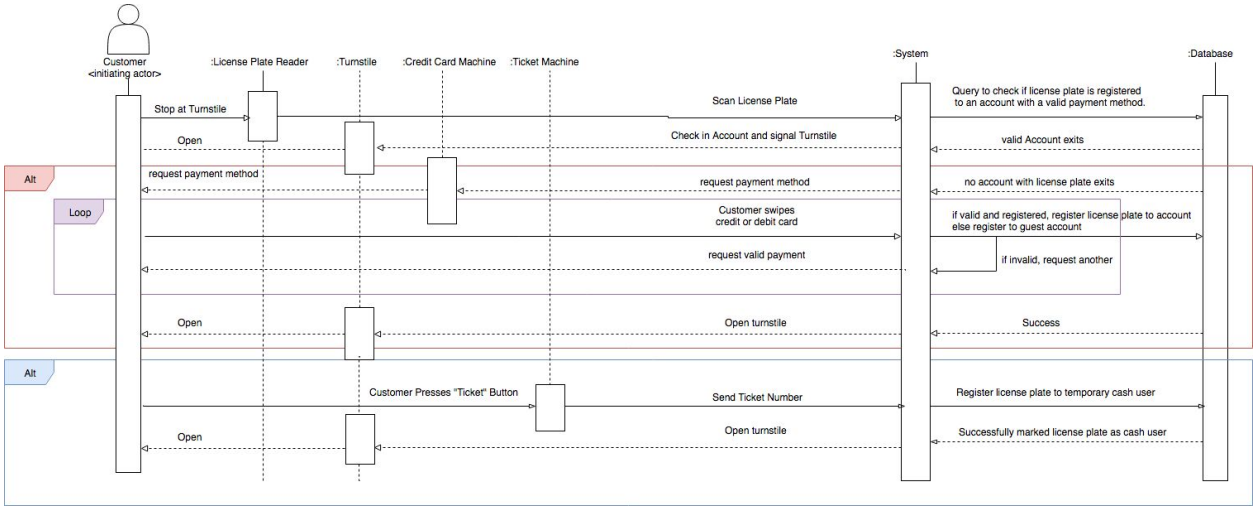
Interaction Diagrams

Use Case 1 - Register: Create an account



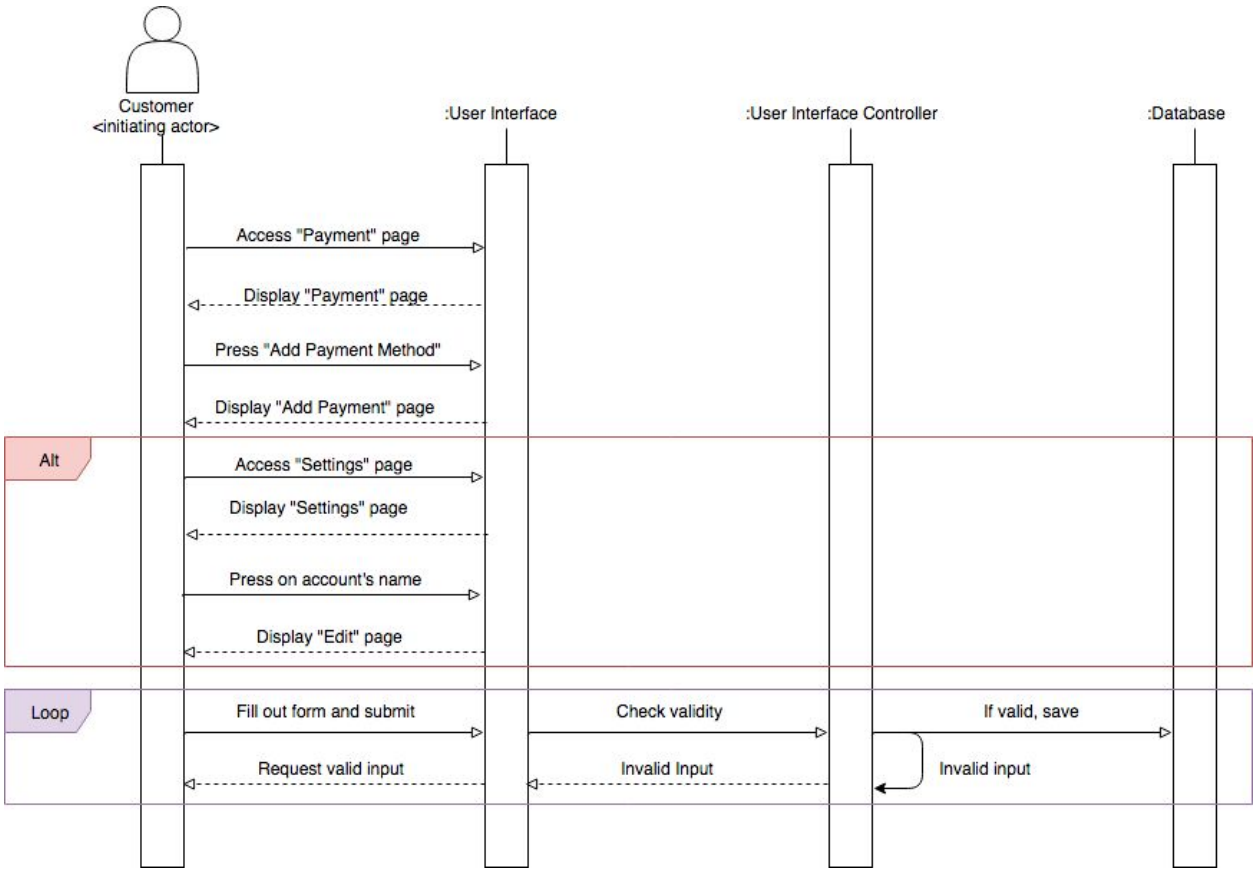
Design Principles: This design greatly focuses on high cohesion. The User Interface Controller mainly has a communication responsibility; it serves as the line of communication between the User Interface and the Database. The Database has a knowing responsibility; it simply stores the data provided by the application. Furthermore, there is a publisher-subscriber relationship between User Interface Controller and User Interface; User Interface Controller waits for events from User Interface to occur and then responds to them.

Use Case 2 - Arrive: Check-in a vehicle to parking lot



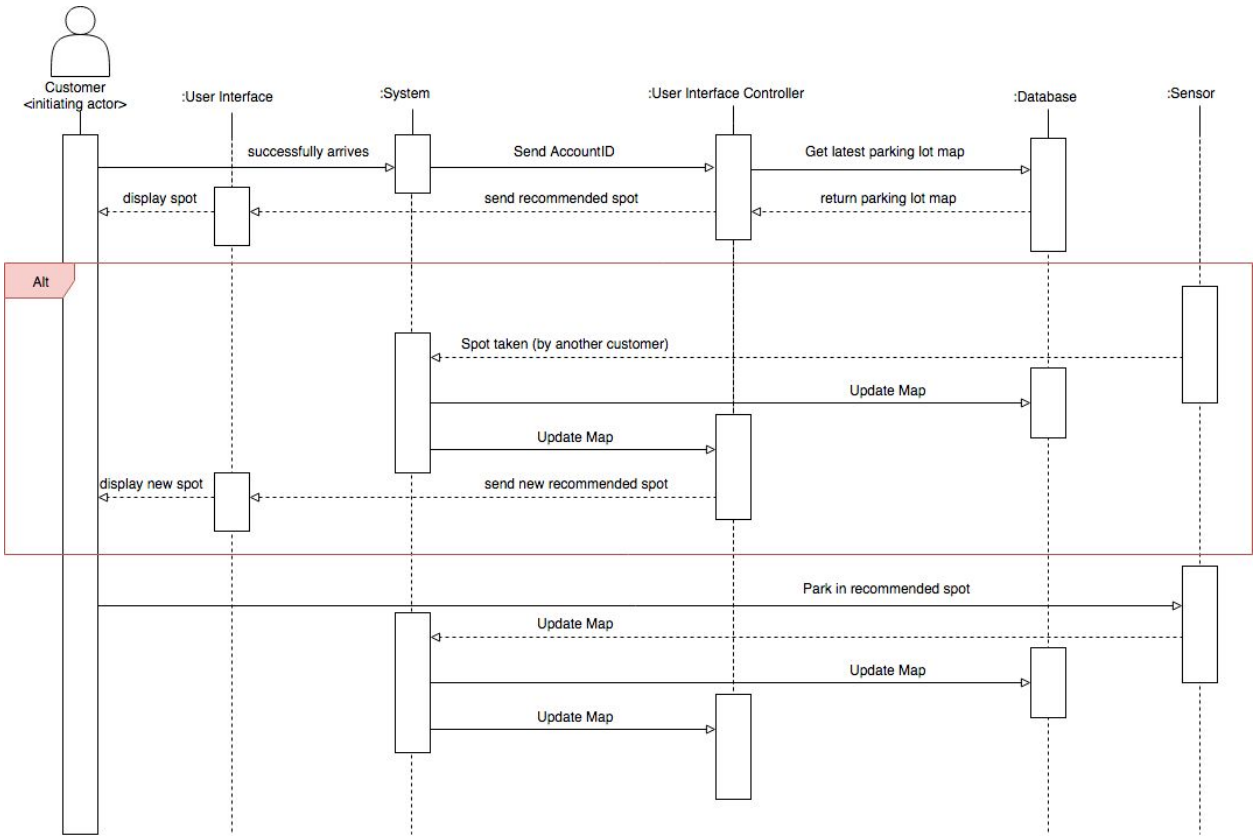
Design Principles: Here, the System takes on all the “doing” responsibility. It is the controller for all the hardware at the parking lot. This hardware includes the, Turnstile, Credit Card Machine, and Ticket Machine. Additionally, there is a publisher-subscriber relationship between System and License Plate Reader; License Plate Reader informs System of the event that a car has arrived. The Database has a purely knowing responsibility.

Use Case 5 - Add/Update User Information



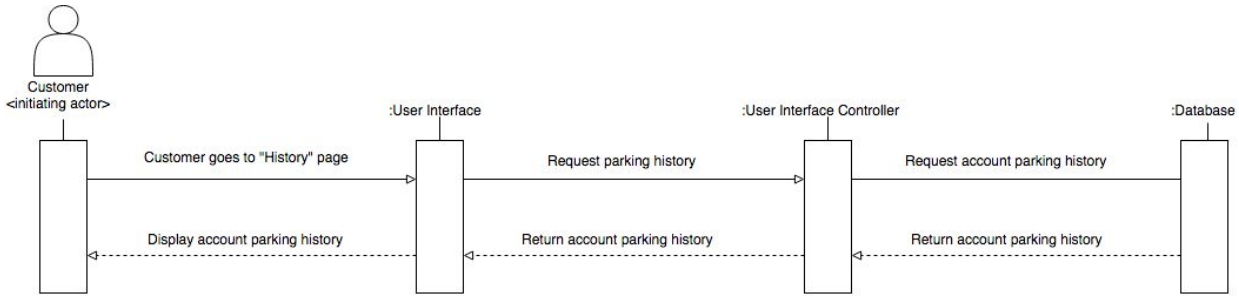
Design Principles: The User Interface Controller mainly has a communication responsibility; it serves as the line of communication between the User Interface (and Customer) and the Database. The Database has a purely knowing responsibility; all it needs to do is remove existing information from itself. Furthermore, there is a publisher-subscriber relationship between User Interface Controller and User Interface; User Interface Controller waits for events from User Interface to occur and then responds to them.

Use Case 6 - Recommended Spot: Navigate user to closest parking spot



Design Principles:
The high cohesion principle stands out here because now there is communication between System, the controller for the parking lot itself, and the User Interface Controller, the controller for the user application. The Sensor is completely isolated from all other actors beyond System. The same can be said for the User Interface to the User Interface Controller.

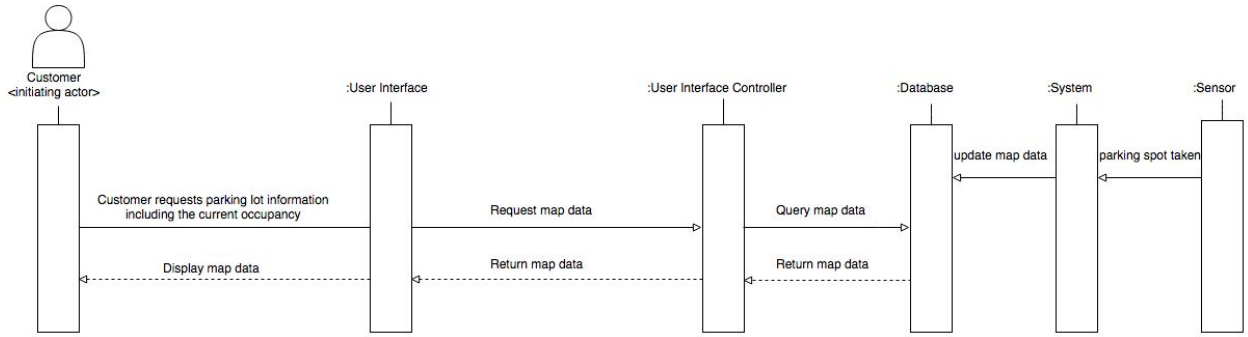
Use Case 8 - History: Customer checks previous parking sessions



Design Principles:
The application must communicate directly with the database. The database has a "Knowing" responsibility, as all the relevant information is stored in the database.

Furthermore, there is a publisher-subscriber relationship between User Interface Controller and User Interface; User Interface Controller waits for events from User Interface to occur and then responds to them.

Use Case 9 - Check Occupancy: Customer sees how full the parking lot is



Design Principles:
Sensor information that is to be utilized by the system should be requested by the SpotChecker due to the high cohesion principle. The checker does not want to have too many computation responsibilities. Its sole purpose is to check which spots are empty and which are taken. Furthermore, there is a publisher-subscriber relationship between User Interface Controller and User Interface; User Interface Controller waits for events from User Interface to occur and then responds to them. The same can be said for System and Sensor; Sensor publishes the event that a car is parked to System.

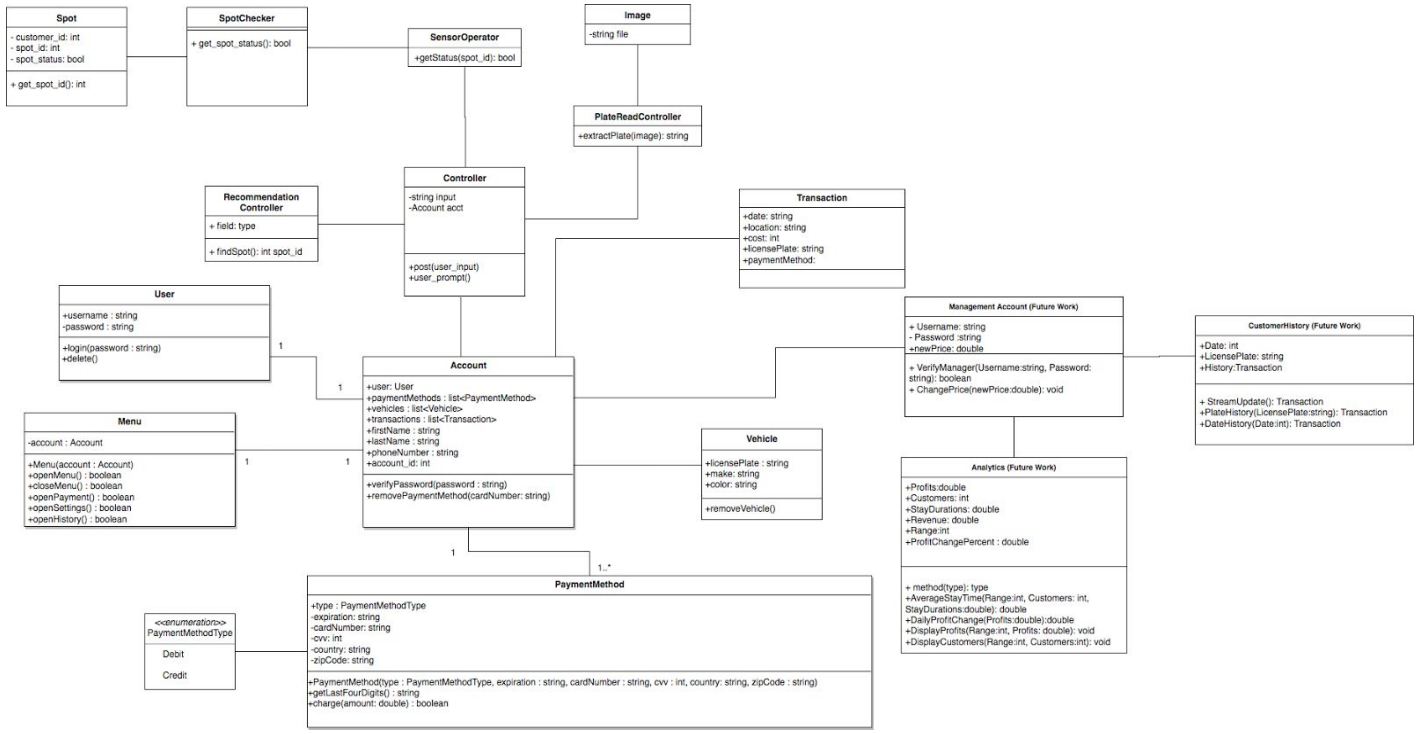
Use Case 13 - Change Spot Availability: Management can toggle a spot (Future Work)

Descriptions on Design Principles:

- Expert Doer Principle: When the Management uses the Manager Interface, that information is relayed to the Manager Interface Controller. The controller is the only actor that manipulates the data and queries the database.
- High Cohesion Principle: The responsibility of every actor are clear. The Manager Interface only displays information provided by the Manager Interface Controller. The Manager Interface Controller transfers data between the Manager Interface and the database.

Class Diagram and Interface Specification

Initial Class Diagram in Report#2



Descriptions:

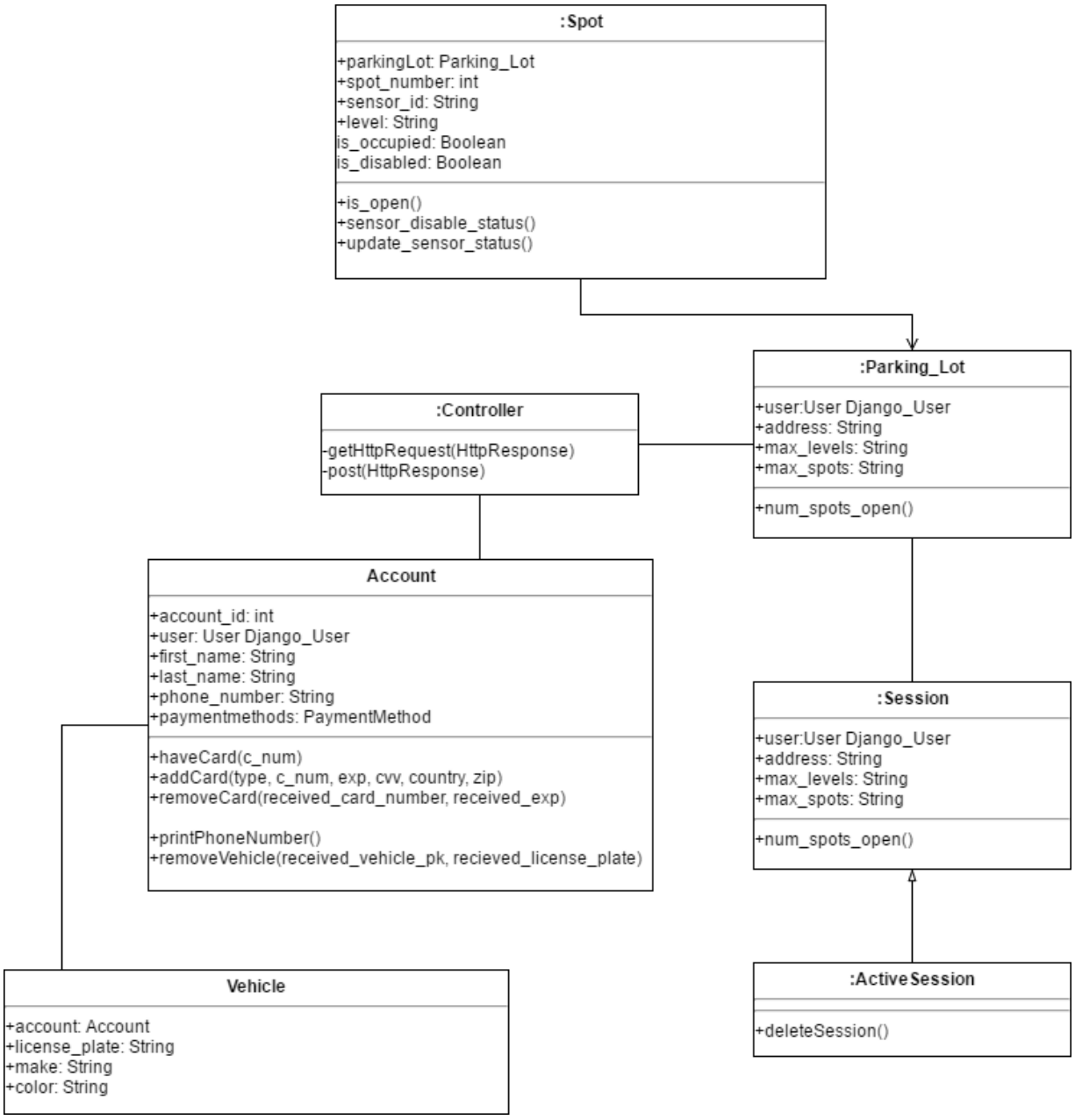
The class diagram above provides a broad understanding of all classes abstracted for the three parts of the application, the user's interface, the manager's dashboard, and the parking system.

- For the parking system, the Spot class contains all attributes of a parking spot, including the IDs of the spot and the customer that currently occupies the spot if there is one. The SpotChecker class is associated with the Sensor Operator class, that reports the status of each spot, either occupied or unoccupied. The Controller class then connects the license plate number with the user accounts as objects of the Account class. The Recommendation Controller class has a method

that takes the available spots into consideration for making a recommendation of where to park.

- For the user's interface, i.e. the customer's site, the Account class holds each customer's information, including the name, list of vehicles, former transactions, number, username and password, unique ID, also the method to verify the login confidentials and remove payment methods. The Menu class has a composition relation with the Account class, as it holds account objects as one of its attribute. The Menu class includes the following methods: open and close the menu, open settings, history, and payment methods. The Payment Method class contains complete credit card information as its attributes; it has methods to charge and remove.
- For the manager's site (Future Work), the management account contains the username and password that are to be checked to verify the identity of the manager. The account has a method to change the price of the spots. The class is associated to the Analytics class and the Customer History class. The Analytics class has as its attributes, the profits and revenues, and the information of customers that have completed their parking, i.e. left the parking lot, including the stay durations. The class has methods to calculate the average stay time of the customers that have already done the parking, and the daily profit changes. The Customer History class lists all the transactions that have done by the users.

Revised Class Diagram



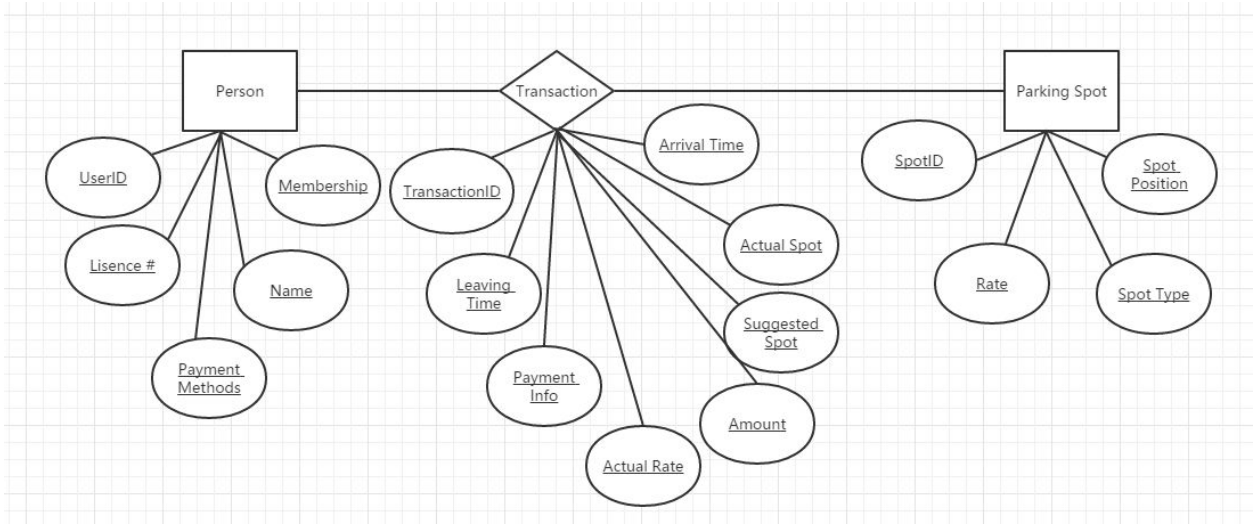
https://github.com/Vatsal09/Garage-Automation/blob/master/Diagrams/ClassDiagram.png

Descriptions of Changes:

The main change that was done was removing redundant and not created classes, such as management classes. These classes were not created due to extenuating circumstances. By removing these classes, the diagram gives a better understanding to the program.

A new class, ActiveSession, was added to facilitate the usage cases that were created. The function and variables of the classes were updated to the correct names and values. The relationship between classes was also updated to reflect inherited classes and dependent classes.

Database E-R Model for Payment History



Database of payment history is stored using SQL table implementing the Entity-Relation Model above.

Sample Relational Model:

Person:

UserID	Name	License #	Membership	PaymentMethod
00232	John Doer	4SAM123	General	VISA #...
00233	Jane Deere	BKXL246	General	MasterCard #...
00234	Bart Simpson	SAM000	General	VISA #...
00235	Homer	6VBV76	General	MasterCard #...

	Simpson			
00236	Marge Simpson	061 DGT	General	MasterCard #...

Entity-Relationship to Relations:

- Person (UserID, Name, License #, Membership, Payment Method)
- ParkingSpot (SpotID, Position, Spot Type, Default Rate)
- Transaction (TransactionID, Amount, Leaving Time, Arrival Time, Actual Spot, Suggested Spot, Payment Info, Actual Rate)

Data Types and Operation Signatures

PaymentMethod:
+ type: PaymentMethodType
- expiration: string
- cardNumber: string
- cvv: int
- country: string
- zipCode: string
+ PaymentMethod(type: PaymentMethodType, expiration : string, cardNumber : string, cvv : int, country: string, zipCode: string)
+ getLastFourDigits(): string
+ remove(): boolean
+ charge(amount: double): boolean

PaymentMethod is the object that will contain all the information regarding a credit or debit card. Many of the properties are kept private for security. The last four digits will allow users to know which card they have on their account. Remove will delete the data from the database and charge will charge the card once the parking session is over.

Account:
+ paymentMethods : list<PaymentMethod>
+ vehicles : list<Vehicle>
+ transactions : list<Transaction>

+ firstName : string
+ lastName : string
+ phoneNumber : string
+ username : string
- password : string
+ verifyPassword(password : string): boolean
+ remove(): boolean

Account contains all of the user's information. Passwords are kept private as a security measure.

Menu:
- account : Account
+ Menu(account : Account)
+ openMenu() : boolean
+ closeMenu() : boolean
+ openPayment() : boolean
+ openSettings() : boolean
+ openHistory() : boolean

Menu contain all the functions and information needed to display the menu to the user. It has access to all the data because it is constructed with an Account object.

Transaction:
+ date: string
+ location: string
+ cost: int
+ licensePlate: string
+ paymentMethod: PaymentMethod

Transaction provides template for storing transactions on users accounts.

Vehicle:
+ licensePlate: string
+ make: string
+ color: string

Vehicle is the format for storing the vehicles associated with user accounts.

Controller:
+ input: string
+ acct: Account
+ post(user_input)
+ user_prompt()

Controls the internal management between components

SensorOperator:
+ getStatus(spot_id:int): boolean

Triggers system wide check of spot state change

Spot:
- customer_id: int
- spot_id: int
- spot_status: bool
+ get_spot_id(): int

Holds the spot information

SpotChecker:
+ get_spot_status(): bool

Determines whether a spot is occupied or not

PlateReadController:
+ extractPlate(image: Image): string

Extracts the license plate number of the customer vehicle

Image:
- file: string

Analytics:
+ Profits:double
+ Customers: int
+ StayDurations: double

+ Revenue: double
+ Range:int
+ ProfitChangePercent : double
+ AverageStayTime(Range:int, Customers: int, StayDurations:double): double
+ DailyProfitChange(Profits:double):double
+ DisplayProfits(Range:int, Profits: double): void
+ DisplayCustomers(Range:int, Customers:int): void

The analytics are public because they don't need to be private. The methods take the information and do simple calculations to provide the analytics tailor to the input range.

Management Account:
+ Username: string
- Password :string
+ newPrice: double
+ VerifyManager(Username:string, Password: string): boolean
+ ChangePrice(newPrice:double): void

The password is kept private because it's better to keep it more secure. The newPrice value is used for the ChangePrice method so that the management account can change the price of the parking lot.

CustomerHistory:
+ Date: int
+ LicensePlate: string
+ History:Transaction
+ StreamUpdate(): Transaction
+ PlateHistory(LicensePlate:string): Transaction
+ DateHistory(Date:int): Transaction

The customer history class is used to find the history of the parking lot customers. It can be searched with Date or LicensePlate. The history is stored in History. Also, a live stream of the customers is displayed and updated with StreamUpdate().

Traceability Matrix

Register
Purpose: Customer registers online
Classes:
- Account: Used to store basic user information
- Vehicle: Vehicle information object tied to User's account
- PaymentMethod: Object used to store the payment methods on users account

Remove Vehicle
Purpose: Customer removes vehicle association from account.
Classes:
- Account: Deletes specific vehicle from account listing
- Vehicle: Object that will be deleted

Delete account
Purpose: Customer wishes to remove account from site.
Classes:
- Controller: Will delete the associated account from database
- Account: Object to be deleted

Add/Update User Information
Purpose: Customer wishes to update payment information or vehicle
Classes:
- Account: Will update/create stored vehicle or payment objects
- Vehicle: Object to be made or updated
- PaymentMethod: Object to be made or updated

Recommend Spot
Purpose: Customer is looking for a spot in parking garage.
Classes:
- RecommendationController: Will deliver best spot recommended action

History
Purpose: Customer would like to see transaction history.
Classes:
- CustomerHistory: Will keep a record of all transactions customer made.

Check Occupancy
Purpose: Customer would like to see how full/ how many available spots there are.
Classes:

Spot: Parking garage spot object
SpotChecker: Checks if spot is available

Change Price (Future Work)
Purpose: Manager wants to change price of parking
Classes:

ManagementAccount: Sets price of parking

Analytics (Future Work)
Purpose: Manager can view statistics about revenue and logistics and transaction history
Classes:

Analytics: has methods for average stay, daily profit change, customers

Remove Payment Method
Purpose: User would like to remove a method of payment.
Classes:

Account: Removes the payment method associated with the account

The domain concepts were adapted from the first report. The purposes of each domain concept is given and its corresponding classes are listed. The classes in use were created using these domain concepts and were created to satisfy the requirements of each use case. There are classes that are able to satisfy multiple domain concepts and were accounted for in the traceability matrix.

Object Constraint Language

Account
Invariants - Must be a valid registered user.
Pre-Conditions - The user must meet the restrictions placed on certain changes made to an
account.
Post-Conditions - Any valid transactions are recorded and then displayed on the account
Page.

Registration
Invariants - Can be any unregistered user
Pre-Conditions - The user can provide the basic information needed to create an account.
Post-Conditions - The new user's basic information is entered into the database and that user is re-directed to their account page.

License Plate Reader
Invariants - Cameras at the entrance and exit turnstile
Pre-Conditions - The camera is can take multiple photos when the car is stopped
Post-Conditions - After a picture is taken, the photo is analyzed and license information parsed.

User
Invariants - Must be a registered user with a confirmed email account.
Pre-Conditions - Needs to be registered with basic user information.
Post-Conditions - Once entered into the system, the user can further manage their account
by entering vehicles, credit cards, and have easy automatic payments.

Parking Lot
Invariants - Contains spots available for parking.
Pre-Conditions - Spots are available.
Post-Conditions - As parking spots begin to fill, the vacant space decreases till it hits zero.

Vehicle
Invariants - Must be a valid vehicle (contain a valid license plate as well as a licensed

driver)
Pre-Conditions - The vehicle has not been registered by another user.
Post-Conditions - The vehicle is entered into the database, under the respective user, and
used labeled with a unique name, to that user.

System Architecture and System Design

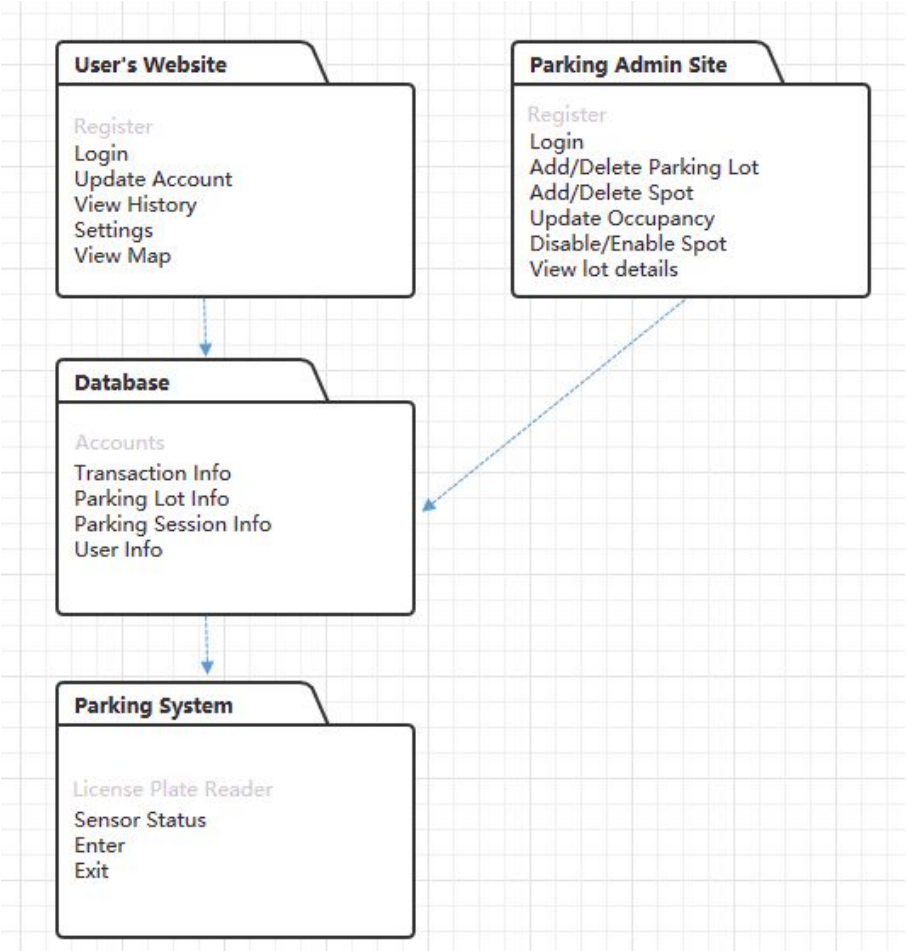
Architectural Styles

The complexity of this system requires the designers to implement various architectural styles. For this reason, the overall architectural style can be thought of as the subdivision of four categories: communication, deployment, domain, and structure. Communication design is constructed using an *Event-Driven* architecture—one that relies on triggers to cause a change in state. An event, in this instance, is defined as a "significant change in state" [10]. The idea is that each time a parking spot becomes occupied, an event occurs. The sensor messages the database that the spot is currently occupied. Likewise, an event occurs when the sensor realizes that a car is not parked in the spot—triggering in a message to the database. The utilization of this pattern means that the system can be more responsive to unpredictable environments. Event driven architectures have loose coupling within space, time and synchronization, providing a scalable infrastructure for information exchange and distributed workflows [10]. **For deployment architecture style, we implemented the frequently used *Client/Server* pattern.** This pattern segments the system into two applications, client and server, where the client makes request to the server (database) and the server provides data based on some logic. The rationale is that the data will be stored on the database and the clients (web-app and website) will help retrieve, update, add, and remove parking space state (using event-driven architecture), user accounts, user information, and etc. We will use this model to aid our interface design. The main benefits are high security because servers offer greater control of security; the centralized access so it's easier to update and access data; and the ease of maintenance helps the clients feel unaffected by server upgrades, repairs, or relocation.

The *Domain Driven Design* architectural design provides an object-oriented style focused on modelling a business domain and defining business objects based on

business domain. Heuristics of class nomenclature, functions, and data are representative of the domain concepts and terminology. We did this so that the communication between sub-teams is clear and the documentation is easy to understand for the Owner. Furthermore, it helps the maintenance, flexibility, and modularity of the database and/or codebase, if a third-party team becomes involved. Lastly, the overall architecture is based on *Component-Based* style for the structure pattern. This structure pattern allows us to break up the application design into reusable, functional components that bridges composing loosely coupled independent components into systems [10]. Each of the components—license-plate-reader, payment system, sensor system—can be thought of as their own entity. The components are able to function on their own and relay their information to the appropriate component. The aspects of component-based styles are that they are: reusable, replaceable, no context specific, extensible, encapsulated, and independent. The benefits of component-based styles is that they have ease of deployment, ease of development, reduced cost, and are reusable. These are the patterns that we used to design the overall architectural style.

Identifying Subsystems



A subsystem is a collection of classes, associations, operations, , events, and constraints that are interrelated [1]. The determination of subsystems helps organize the team and the main goal was to create a subsystem that each team could work on without the need of components from another subsystem. To do this, we assigned objects in one use case into the same subsystem, minimized the number of associations crossing subsystem boundaries, and ensured that all the objects in the same subsystem were functionally related. While Admin site and the User site depend on data from the database, the tables are completely independent. And while the database depends on the parking system data, we built predetermined data models so we could use dummy data to

simulate what would happen. The idea was to keep subsystems loosely coupled. However, within the subsystem, modules have strong cohesion and modular design so that each activity is performed by one component.

Mapping Subsystems to Hardware

The system is distributed across multiple computers. Users will access the the application through a web browser either on their mobile devices or desktops. Since it is expected that users will usually be using their phones, the focus will be on the mobile website. Manager have the same options, although it's expected that they will use the application on desktops. The benefit for a manager to use a desktop is that it can be connected to a display in the parking lot entrance. This allows the manager to display the currently available parking spots on a map. The website running the application will be hosted on a server containing the database. Another machine manages the sensors, turnstiles, ticketing machines. This machine is also responsible for transferring the data it collected to the server.

Persistent Data Storage

All of the data collected will be stored in a relational database. This data will include account information such as names, license plates, addresses, phone numbers, credit card information, and debit card information. Also parking lot data including the name and its map. It is expected that with time, the server will contain many different parking lot. Lastly analytical data will be collected such as parking history and transaction history. This will be used to provide manager with more insight regarding their performance and give users information regarding their history.

Network Protocol

Our service uses a website that will be hosted on a single server so it would only need HTTP and HTTPS as communication protocols. The database and source code would be on the main machine so we don't have to use any other communication protocols. Our system is versatile and easily upgradable therefore we could upgrade to the most secure communication protocol if the situation is necessary.

Global Control Flow

Execution Orderness:

Our system has both event driven and procedural driven aspects. The customers follow a general path. They have come up to the turnstile, enter, pay, park, and leave. However, the customer and the management has many events they can choose through. The manager decides the price of the parking lot and chooses the range of the timeline for the analytics to be displayed. The customer decides what they want to do. They can park where they please, leave and enter whenever they want, choose any payment method, and edit their account information.

Time constraints:
Yes there are timers in the system to measure the length of the stay. Our system takes into account the real time as well because the check in and out time log of each customer is tracked. The timers are not periodic because the exact time is recorded.

Concurrency:
No, it has to wait for the next event.

Hardware Requirements

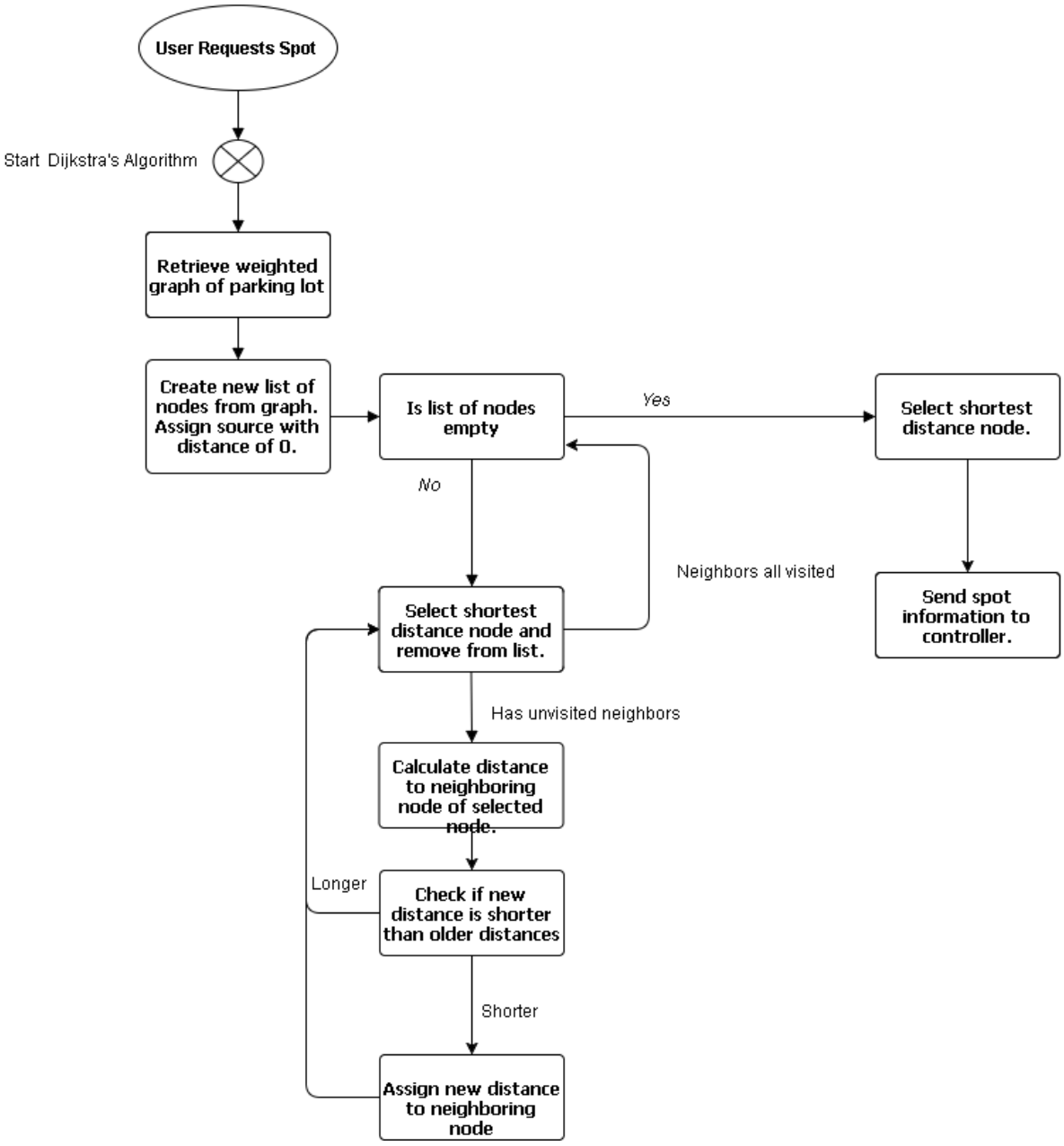
Sensor: A laser-based sensor that can identify the state of each parking spot (occupied or unoccupied)
Servers: At least one server to process the functionality between the user and the parking system.
Cameras: 2 cameras needed at the entrance and exit turnstiles that can provide an enough quality photo for license plate computer vision analysis.
SmartPhone: Our system can be used by both a smartphone and a computer for the customer to use. They can use the smartphone to fully utilize our recommend parking spot feature.
Wireless Adapter:
Hard-drive: At least 50GB of space allocated for our system for cached information.
Computer: Our system can be used by both a smartphone and a computer. Customer can use a computer to view, edit, and access their account information.
Manager's Computer: Used by the system to run all of the programs. Also a manager of the parking lot uses this computer to view their dashboard for logistics and analytics.
Database: We need at least 100 GB to store all the processed user information and data.

Algorithms and Data Structures

Algorithms

Due to the revision of the manager-side dashboard and recommended spot use cases to future work, any algorithms pertaining to the mathematical models described in Report #1 were discarded. For the manager-side dashboard the idea was to create a line of best-fit, using a linear regression model, to predict the total monthly and annual revenues. For future work, we also intend to implement a Recommended Spot feature that shows the user a recommended spot to park. To calculate the optimal parking spot, a weighted graph will be used along with a variation of Dijkstra's Algorithm. The parking lot will have each spot as a node of a graph along with a weighted value to symbolize the physical distance from the entrance of the parking lot to the spot. This algorithm is the most popular and most efficient method to calculate the shortest distance between the source and the node. On the following page, we provide an activity diagram used to describe the process of the algorithm.

Optimal Parking Spot Algorithm



Data Structures

While our system does not employ a variety of data structures, we do use arrays and graphs. The use of graphs aids with the representation of distance from the entrance to all of the available spots. We can perform the shortest path analysis using the graph data structure as the cornerstone piece to perform a variation of Dijkstra's Algorithm. Depending on the implementation, this data structure offers great advantage (performance) in adding and querying edges. The array is a rudimentary data structure that is often used to map an object with a key. As some database queries will pull multiple entries at once, an array data structure is used to store all these search query results in one place. It's really the only data structure that was implemented due to its simplicity and efficiency.

User Interface Design and Implementation

Customer Application Interface:

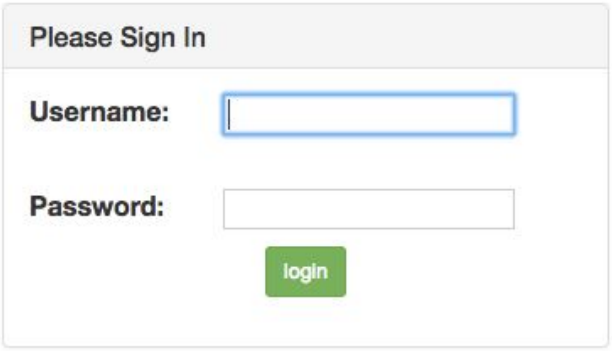
When designing the interface for report 1, we kept in mind the importance of a simple interface. Because our application contains a navigational functionality, we had to make sure customers would not be distracted while driving. To achieve this, we made ease-of-use and minimalism core foundations of our design from the beginning.

This required a good balance of information and button clicks. Managerial actions such as registering or adding a payment method require more button clicks, but these are actions that are expected to be done before or after parking; they contain information which we deemed to be non-critical during regular use.

(Log In): The user will provide a username and password and press login. If they wish to register they will press the
register button.

<u>To Log In **(5)**:</u>

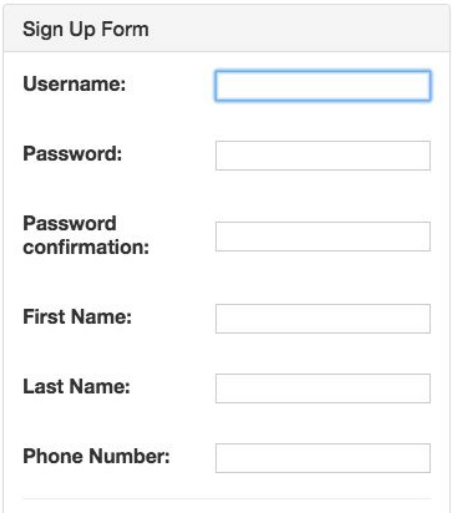
1. Tap username field
2. Type username
3. Tap password field
4. Type password
5. Press Log in

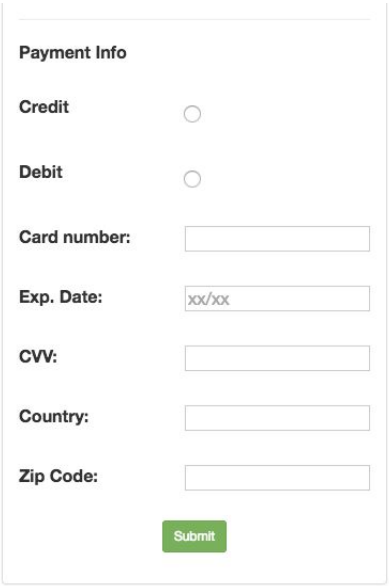


(Register): After filling in a username and password, pressing the register button will bring up this page. It will ask the user for their name, phone number, and a payment method.

<u>To Register **(20)**:</u>

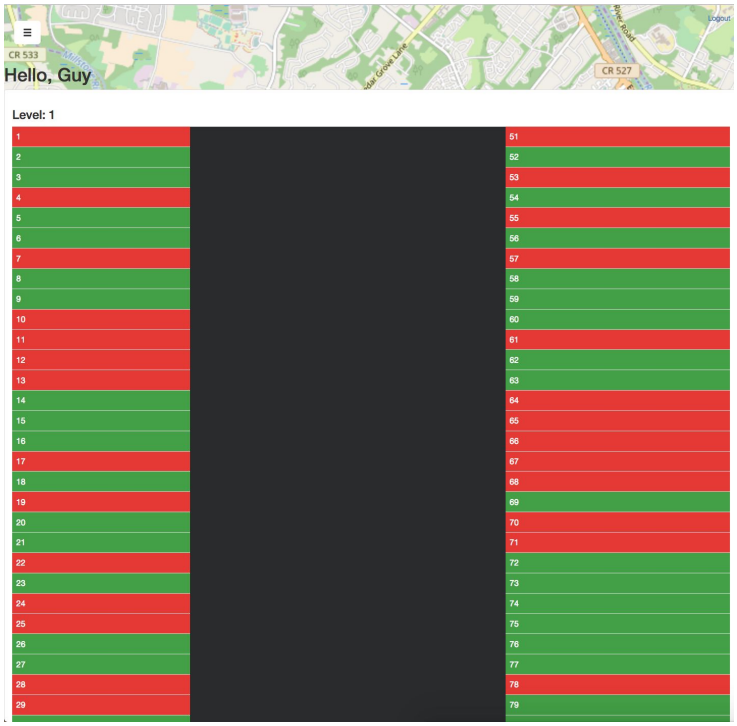
1. Tap username field
2. Type username
3. Tap password field
4. Type password
5. Press Register
6. Tap first name field
7. Type first name
8. Tap last name field
9. Type last name
10. Tap phone number field
11. Type phone number
12. Tap card number field
13. Type card number
14. Tap expiration date field
15. Type expiration date
16. Tap country field
17. Type country
18. Tap zip code field
19. Type zip code
20. Press Register





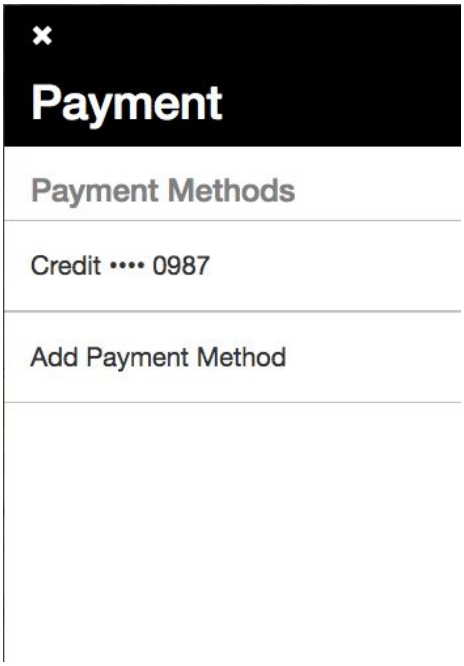
(Home): This page will display the parking lot map (with navigation) and a menu button. The menu button will show a small window displaying buttons for Payment, History, and Settings

To Display Map **(0)**



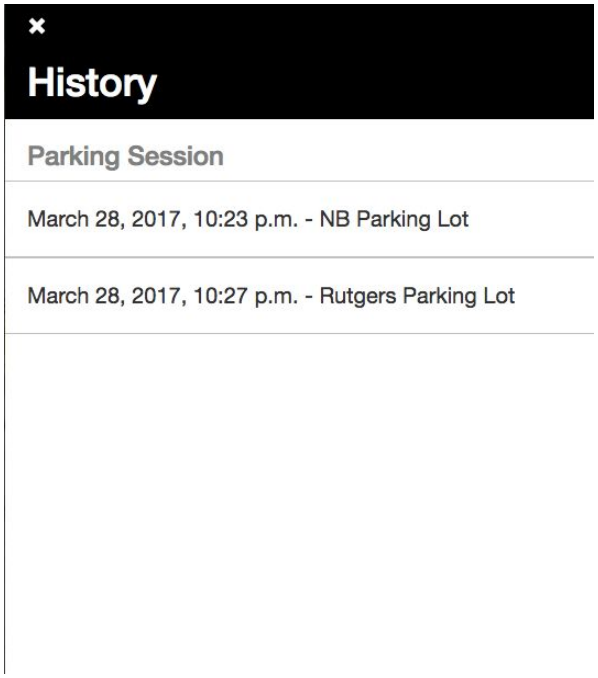
To display Payment Page **(2)**:

1. Tap Menu button
2. Tap Payment



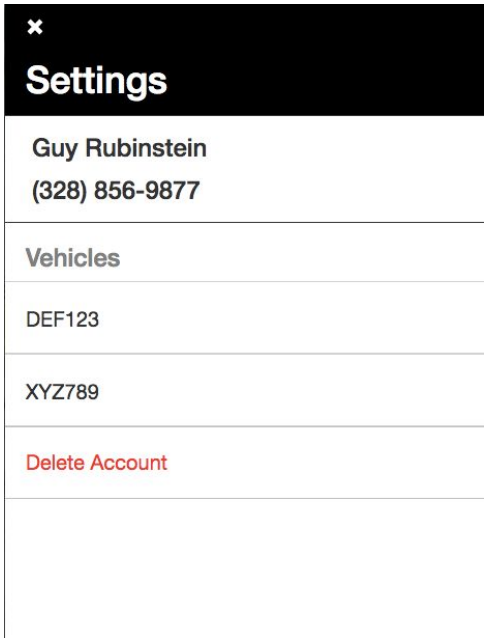
To display History Page **(2)**:

1. Tap Menu button
2. Tap History



To display Settings Page **(2)**:

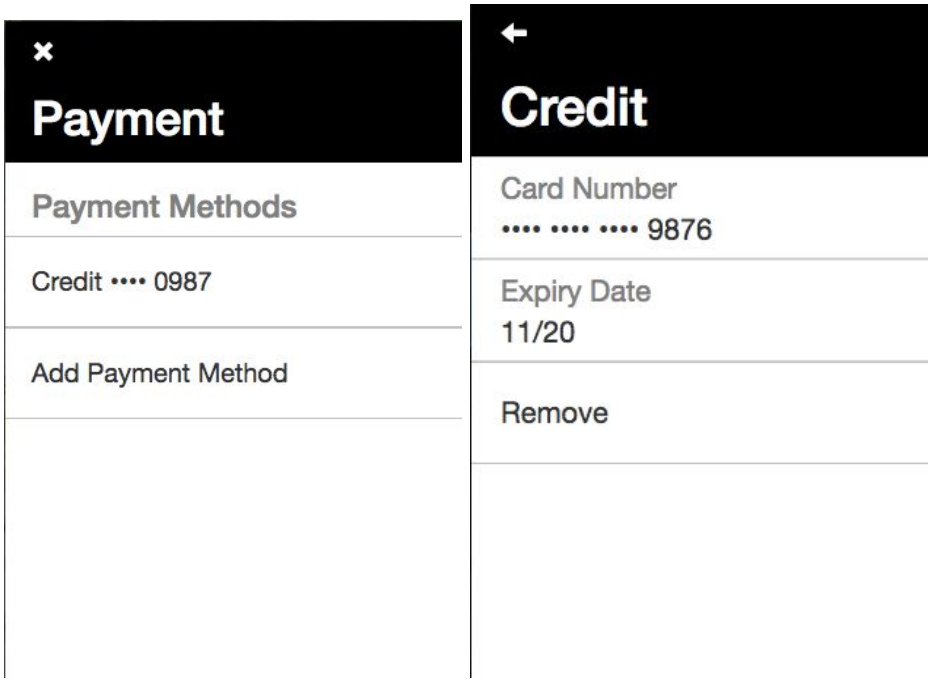
1. Tap Menu button
2. Tap Settings



(Payment): existing payment methods will be listed and a button to add a payment method.

To See Payment Method Details **(1)**:

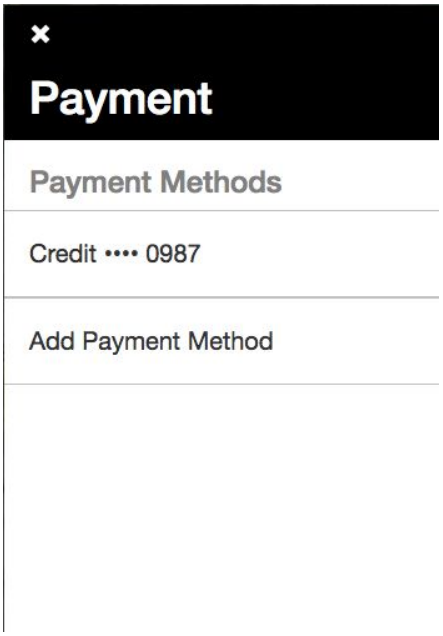
1. Tap on payment method in list



To Remove Payment Method **(3)**:

(Payment details): display details regarding payment method and option to remove it

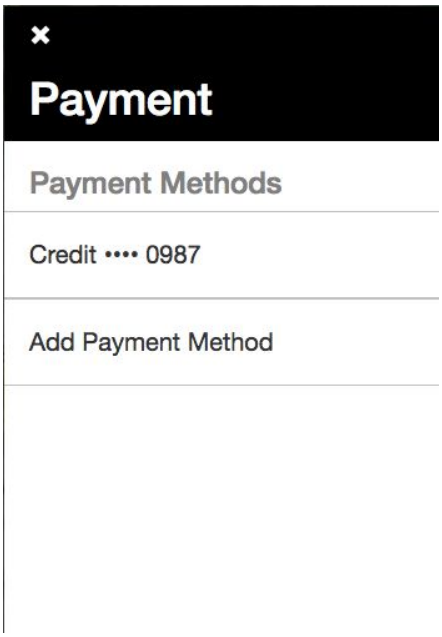
1. Tap payment method in list
2. Tap remove card

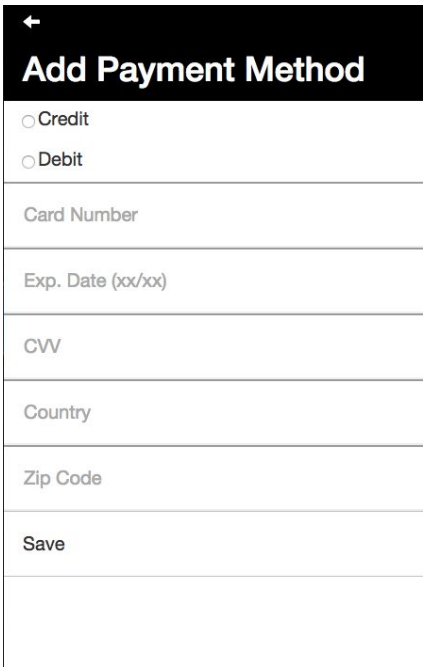


(Add Payment Method): display form for information required to add payment method

To Add Payment Method **(10)**:

1. Tap add payment method
2. Tap card number field
3. Type card number
4. Tap expiration date field
5. Type expiration date
6. Tap country field
7. Type country
8. Tap zip code field
9. Type zip code
10. Press save

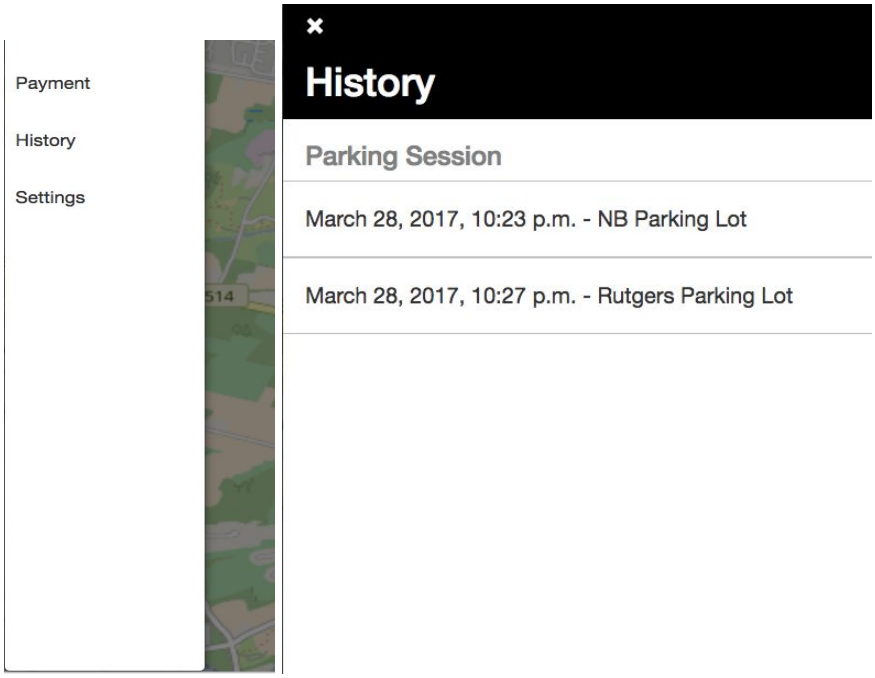




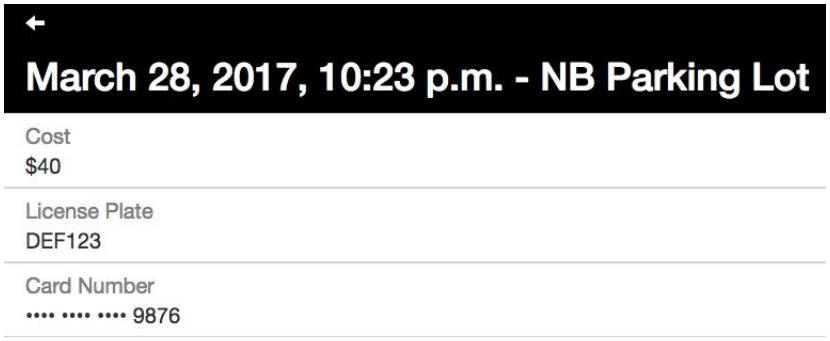
(History): Display list of past parking sessions

To Parking Session Details **(1)**:

1. Tap on parking session in list



(History details): display information about a parking session



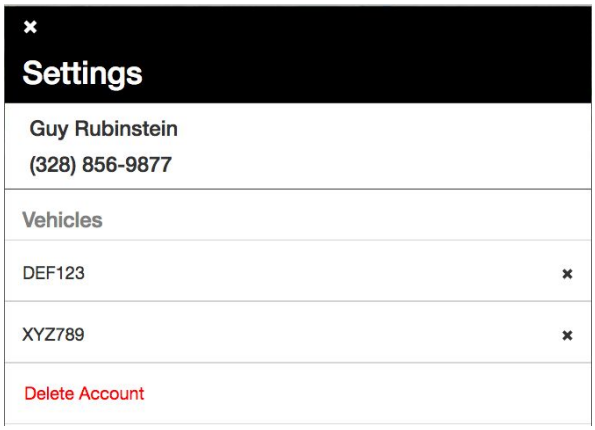
(Settings): Display information about the user and vehicles registered to account with option to remove each one

To Remove Vehicle **(2)**:

1. Tap on x next to license plate
2. Tap yes on confirmation pop up

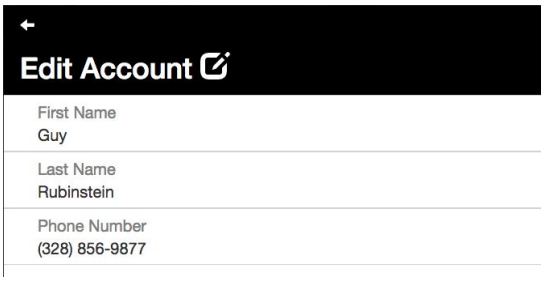
To Delete Account **(2)**:

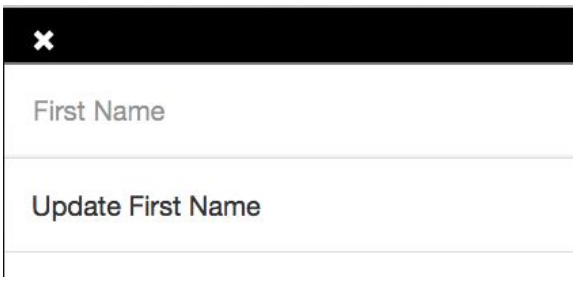
1. Tap delete account
2. Tap yes on confirmation pop up



To edit First Name/Last Name/Phone Number **(3)**:

1. Tap on current Name/Phone Number
2. Tap First Name/Last Name/Phone Number
3. Enter new First Name/Last Name/Phone Number and press Update

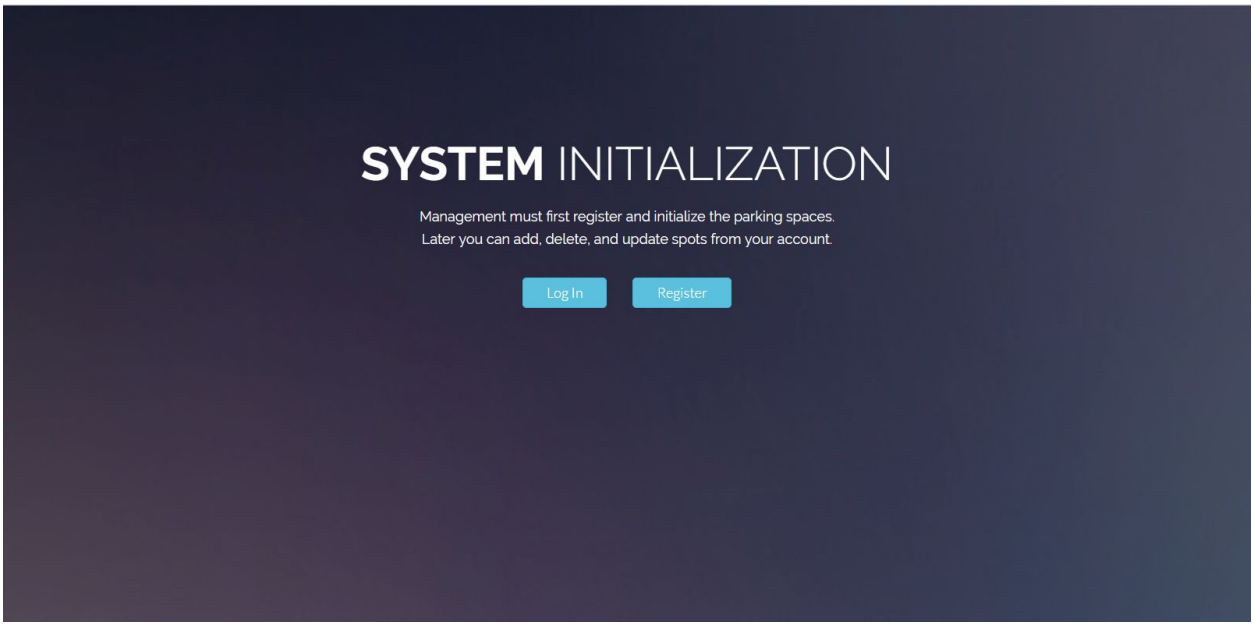




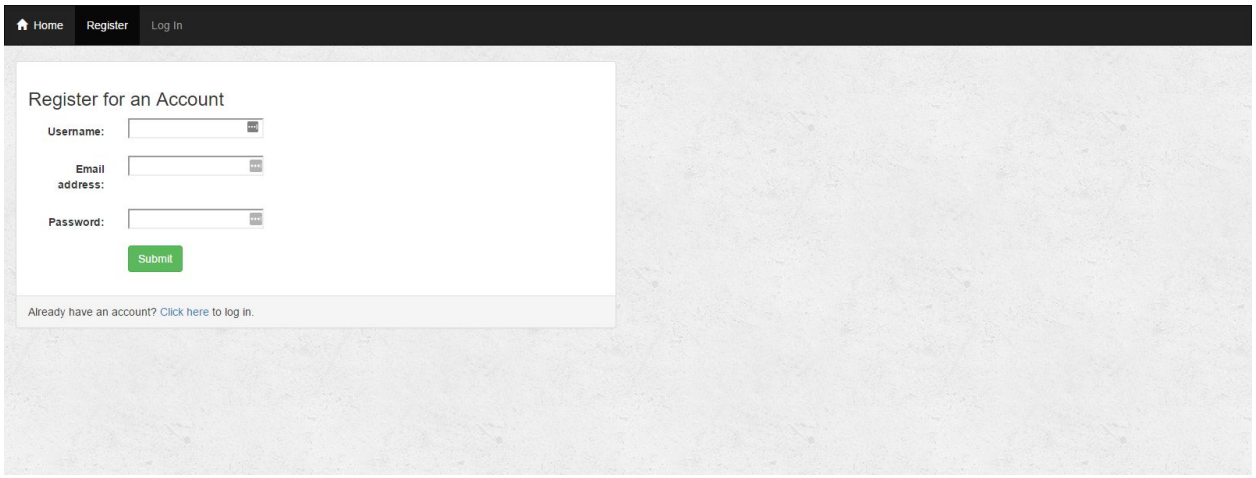
Management Website Interface:

The system needs to be initialized: the license plate needs to interface with the database, the sensor ids need to be tied to parking spots, and the sensors need to update their occupancy status in the database. From the owner/management's perspective, this can be a daunting task. They need a simple interface where they can view all their lots, their status, add/delete parking lot, add/delete spots, enable/disable spots, and much more with convenience and ease.

(Home): This landing page is the first part of the initialization process and will serve as a gateway to view and update lot information



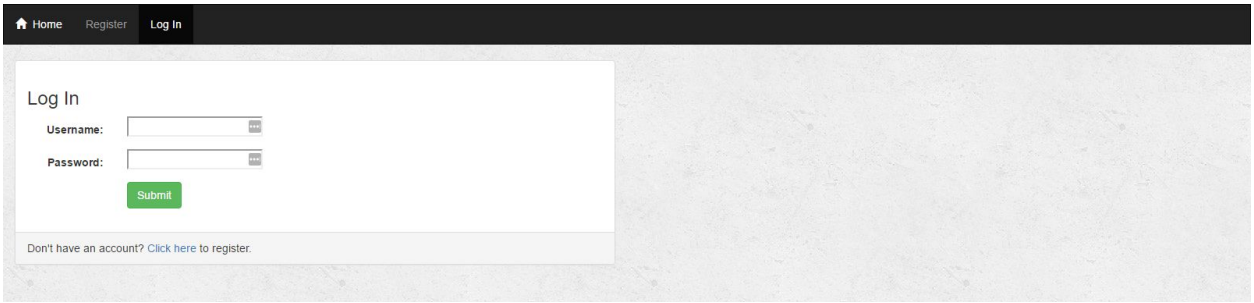
(Register): After clicking the register button, the management will be asked to register for an account by filling in a username, email address, and password. To complete the process, they must click submit button.



To Register **(7)**:

1. Click username field
2. Type username
3. Click email address field
4. Type email address
5. Click password field
6. Type password
7. Press Submit

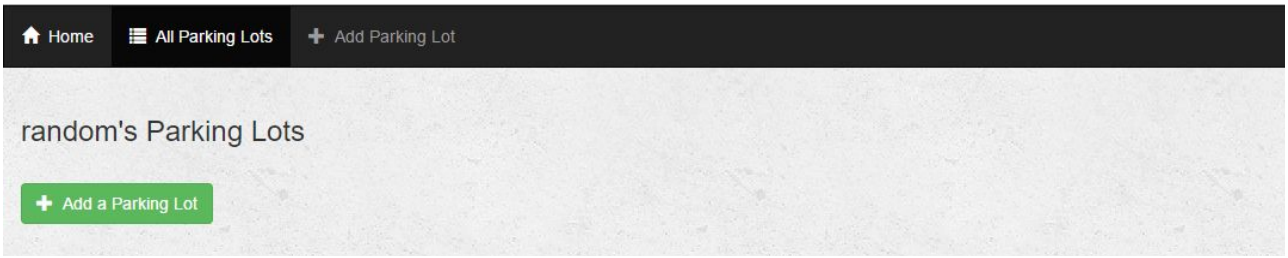
(Login): After registering the account, the management account is created and the user must click the "Log In" in the navigation bar to log in. They can also access this account from the home or landing page and click the "Log In" button.



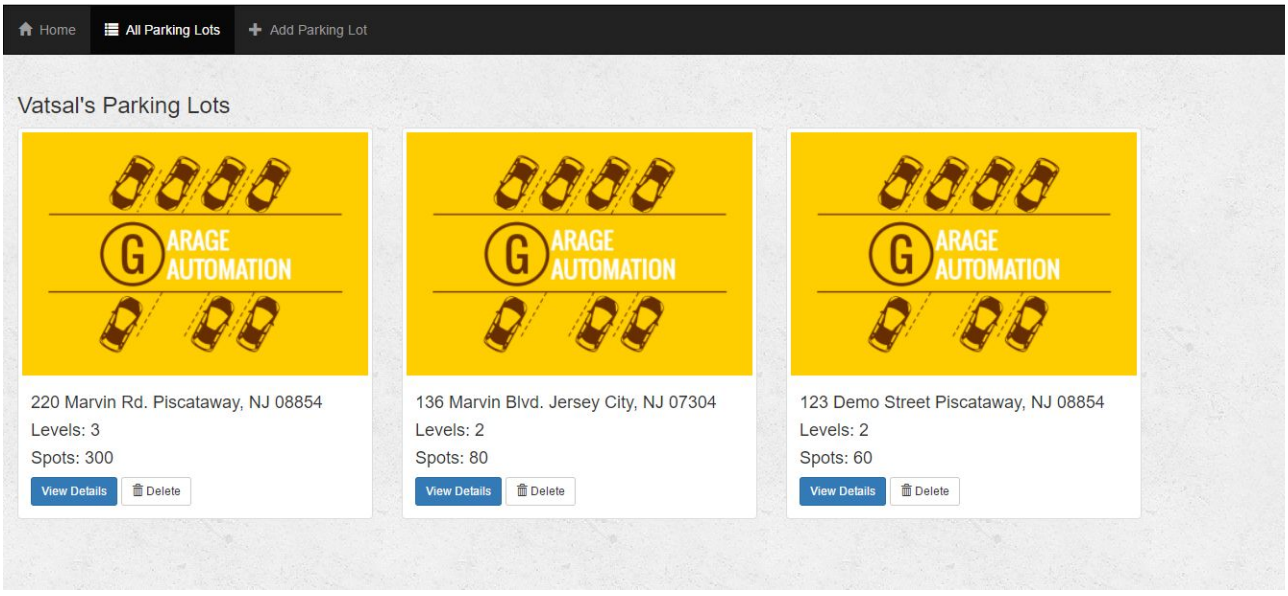
To Log In **(5)**:

1. Click username field
2. Type username
3. Click password field
4. Type password
5. Press Submit

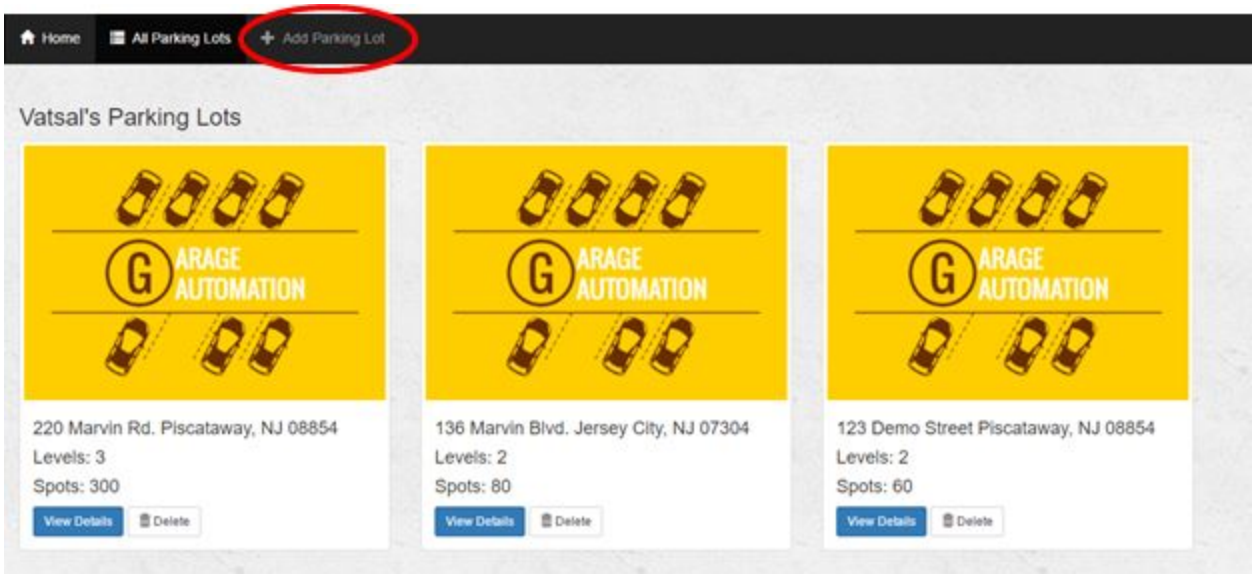
(Main Page [1 of 2]): After the logging in, if this is is your first time logging in or if you have deleted all parking lots, you will be viewing this page. It will prompt you to click the "Add a Parking Lot" button to add a parking lot.

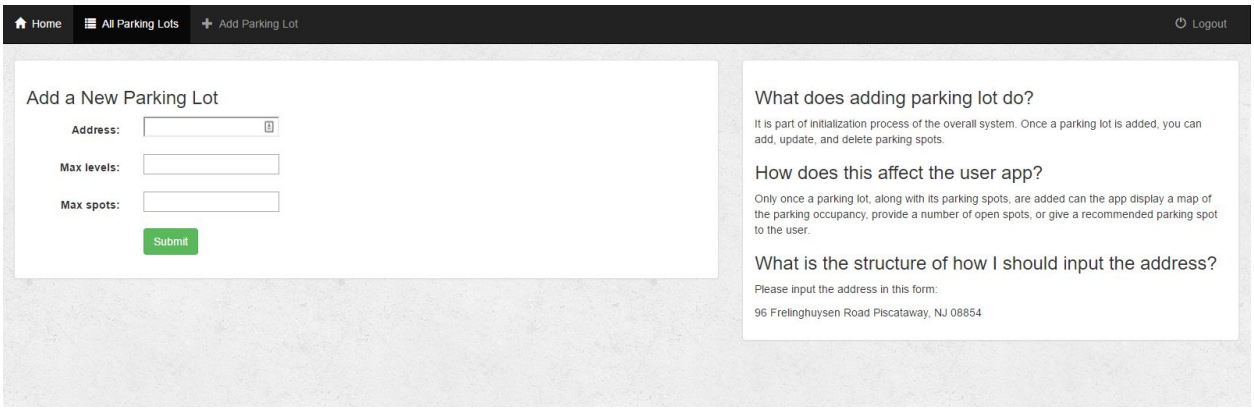


(Main Page [2 of 2]): After the logging in, if you already have parking lots added to your account, you will see all of your lots and the ability to view details of your lots or delete the lot from the system.



(Add a Lot): If you do not have any lots, you will see the green button (as in Main Page [1 of 2]). However, regardless if you have or do not have lots, you will see the "Add a Lot" feature in the navigation bar. Clicking this will lead you to a page where you can add lots. On this page, you enter the address of the lot and the number of levels and spots it possesses.



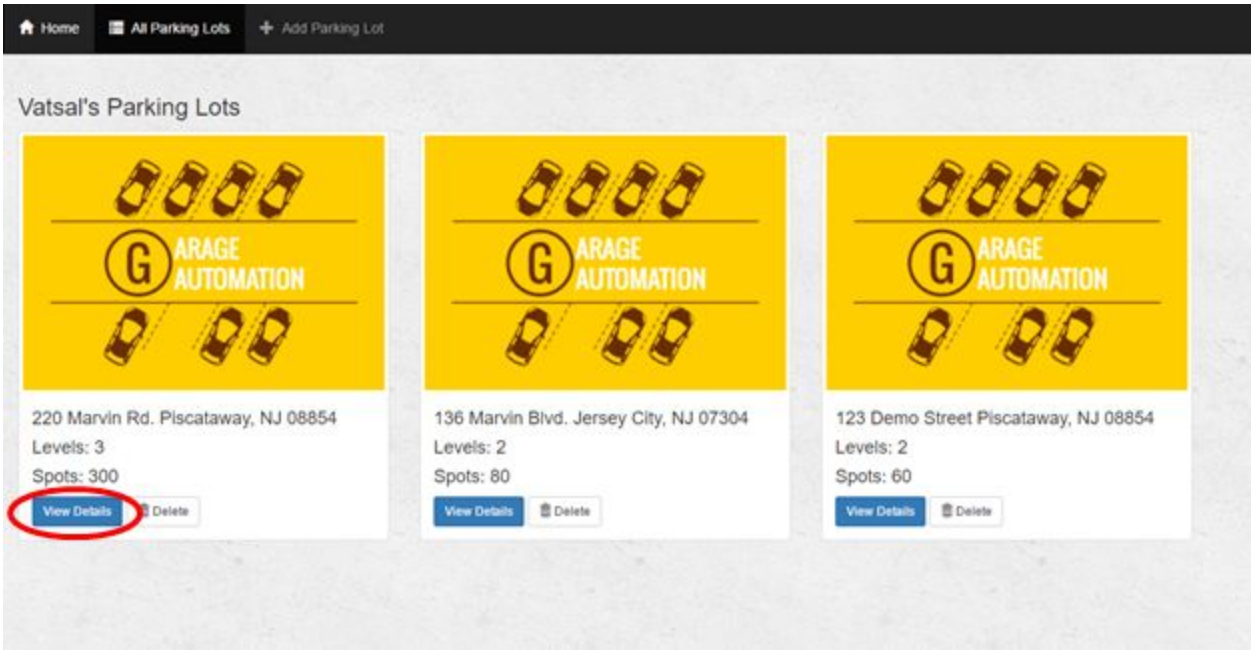


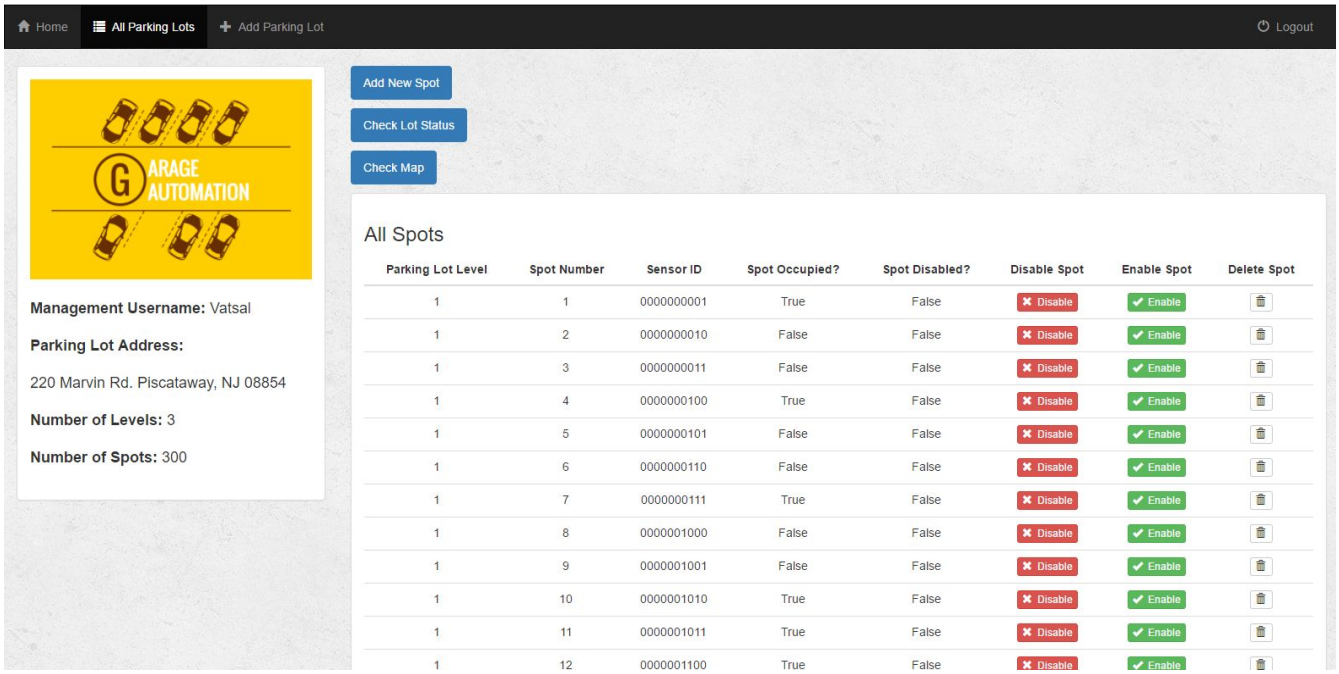
To Add a Lot **(8)**:

1. Click "Add a Lot" in navigation bar
2. Click address field
3. Enter address
4. Click max levels field

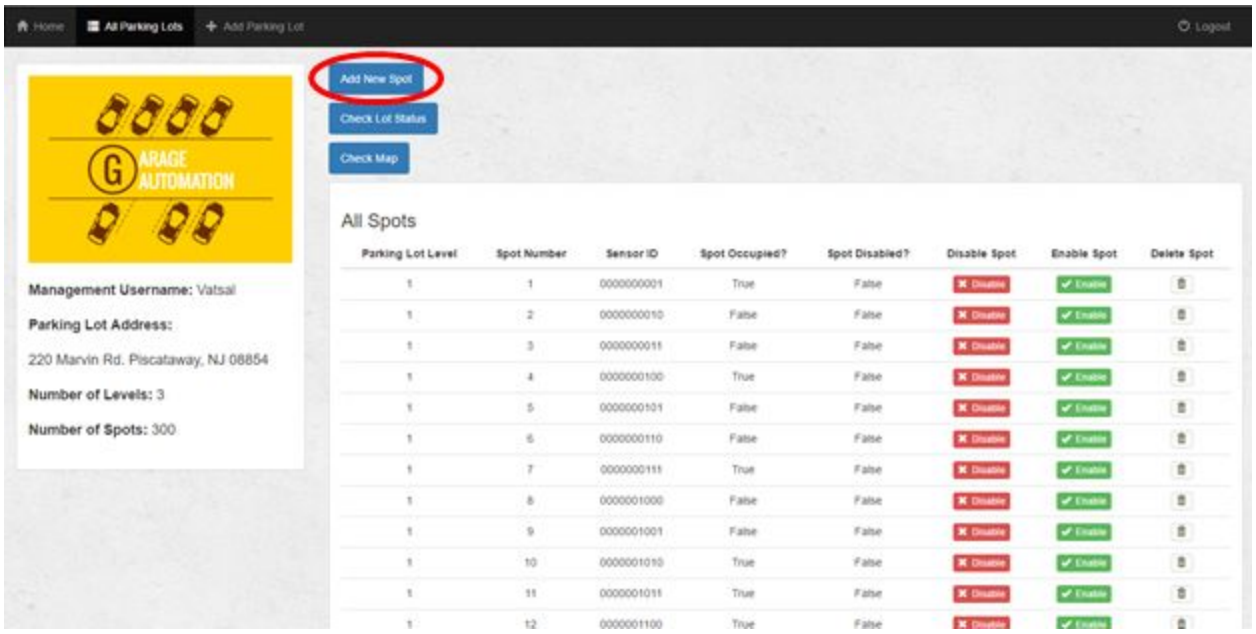
5. Enter number of levels the parking lot possesses
6. Click max spots field
7. Enter the number of spots the parking lot possesses
8. Press Submit

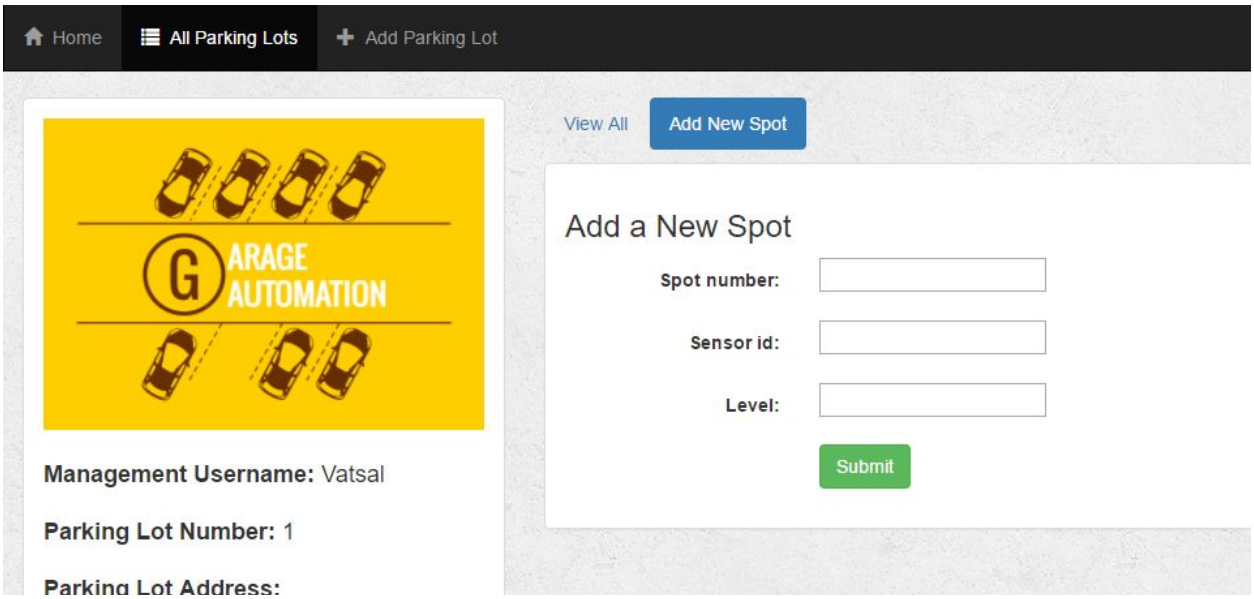
(Detail): From the Main page, you can look at the lot's details by clicking "View Details". This page displays a snapshot of the lot's information to the left (username, lot address, number of levels, and number of spots) and the spot details to the right. **The spot details include the parking spot number, the sensor attached to it, level where the spot is located, whether the spot is occupied, and whether it is disabled by the manager. This page also contains three buttons at the top to add a spot, check the occupancy map, and lot status (current and past customers). The manager can toggle disable button to make the spot appear to be be occupied, in case of maintenance.**





(Add a Spot): The sensor ids need to be tied to a parking spot. The management account must add all the spots in the parking lot, using a similar approach as adding a parking lot. You must click "Add a Spot" button at the top of the Detail page and then fill the information in the ensuing page.





To Add a Spot **(8)**:

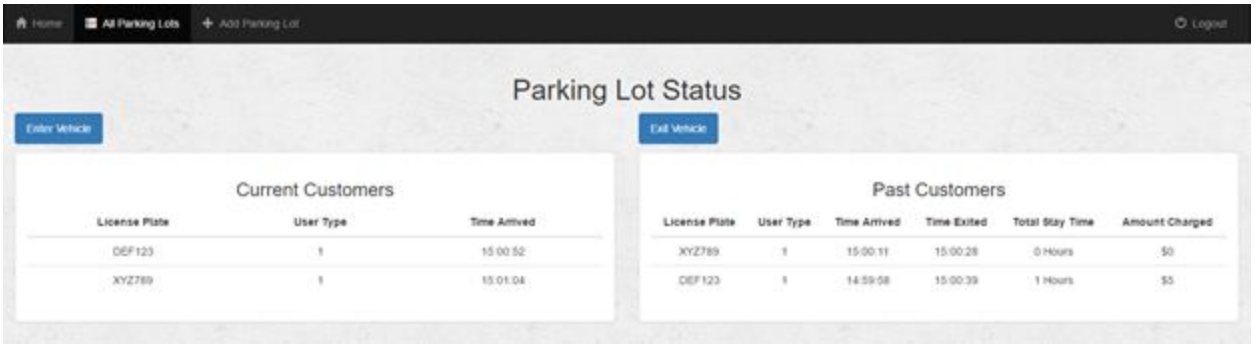
1. Click "Add a Spot" button on the Details page
2. Click spot number field
3. Enter the parking spot number
4. Click sensor id field
5. Enter sensor id attached to the parking spot
6. Click level field
7. Enter the level where the spot is located
8. Press Submit

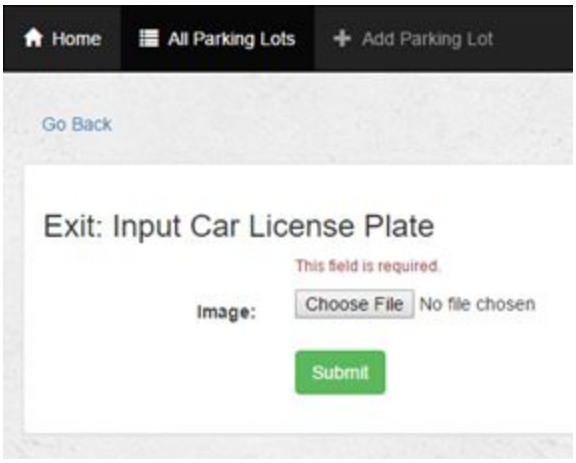
(View Occupancy Map): From the Details page, click the "Check Map" button to view the occupancy map. The map starts on level 1 but you can scroll all the way down and click "Next Level" to view the next levels.

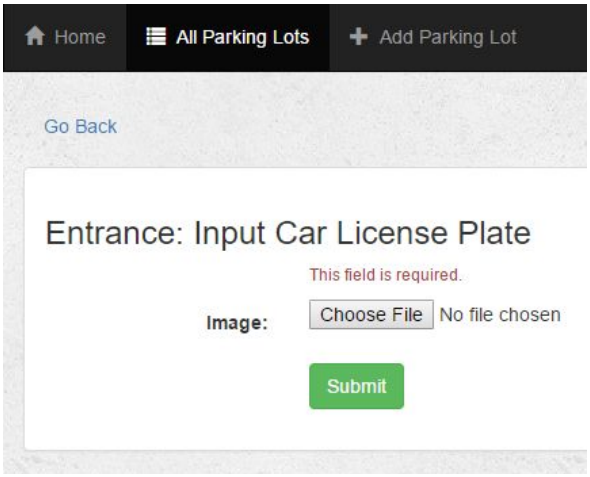
To Check Occupancy Map **(2+)**:

1. Click "View Map" button on the Details page
2. Click "Next Level" button by scrolling to the bottom

(Check Lot Status): From the Details page, click the "Check Lot Status" button to view the current customers and the past customers. For the current customers, you can view their license plate number, the user type (registered, guest, or cash), and the time they arrived. For the past customer, in addition to the information provided in the current customer section, the manager is able to view the ime exited, time of stay, and the total amount charged to the customer.







The Enter Vehicle button leads to an interface to simulate how the license plate will be used to automatically read the license plate number and the system will be able to automatically recognize the type of user and note the time arrived. The Exit Vehicle is used to simulate the license plate reader's ability to determine that the car has successfully exited. The enter and exit sequences are to be done automatically but for the demo, it was easier demonstrate the integration of the license plate reader with the system, using this interface.

Manager Dashboard Interface (Future Work):

(Login): User is presented with the login page, which they have to go through to access the dashboard
Login: **(5)**
1. Tap username field
2. Type username
3. Tap password field
4. Type password
5. Press Log in

(Manager Tools): Actions that the manager can take to change aspects of the parking lot.
Change Range: **(1)**
1. Click underneath the graph for hours, days, weeks, months, and years.

Change Price: **(2)**
1. Click on change price field
2. Enter new price

Change Spot Status: **(2)**
1. Click on change spot status drop down menu
2. Click on parking space that you want to change from available to not available or vis versa.

(Customer History): Manager can look at aspects of customer history and get a copy.

Look at live stream of customers: **(1)**
1. Click on Customer History button

Search Customer History: **(4)**
1. Click on Customer History button
2. Click on date or license plate number field, depending on what you want to search by.
3. Enter the date or license plate number
4. Click export

Design of Tests

Testing the Customer Website:

Goal: The Customer needs to be able to use our website for updating various information regarding his/her account.

Test Cases:

Customer signs up for an account

Test-case Identifier: TC-1 **Use Case Tested:** UC-1 **Pass/fail Criteria:** The test passes if the system can recognize bad inputs for sections of registration like payment methods and responds correctly back to user **Input Data:** Username, password, phone number, payment method, billing address
Test Procedure: Step 1: For username, test the system if it does not allow multiple accounts being added to the database. Step 2: For password, test the system if the re-enter password functionality works. Also test that the requirement for a secure password is being checked by the system. Step 3: For phone number, test the system if it checks a correct amount of characters for a phone number. Step 4: For payment method, test the system if it checks for a correct input of credit/debit card. Step 5: For billing address, test the system if it checks for a correct input of an address, city, and state. **Expected Result:** ● A username does not have duplicates in the database. ● A password should be at least 8 characters with at least one uppercase character. ● A phone number should have 10 numbers ● A payment method should be filled out completely to match the standard 16 CC number. ● A billing address should follow the standard format for address, city, state.

Customer changes account information

Test-case Identifier: TC-2

Use Case Tested: UC-5

Pass/fail Criteria: The test will pass if the user is able to change their account information accurately

Input Data: Password, phone number, payment method, billing address

Test Procedure:
Step 1: Test the functionality of your ability to change password and verify if the password changed in the database
Step 2: Test the functionality of your ability to change phone number and verify if the phone number changed in the database
Step 3: Test the functionality of your ability to change payment method and verify if the payment method changed in the database
Step 4: Test the functionality of your ability to change billing address and verify if the billing address changed in the database

Expected Result:

- When the password is changed, the new password replaces the old password
- When the phone number is changed, the new phone number replaces the old number
- When the payment method is changed, the new payment method replaces the old payment method
- When the billing address is changed, the new billing address is replaces the old billing address

Testing the Customer Parking Lot Application:

Goal: The Customer must be able to use an accurate Parking Lot Map successfully.

Test Cases:

Customer views Parking Lot Map

Test-case Identifier: TC-3 **Use Case Tested:** UC-9 **Pass/fail Criteria:** The test passes if the map that is being displayed to the user is accurate to the state of the sensors. **Input Data:** Change Sensor State
Test Procedure: Step 1: View an initialized parking lot map. Step 2: Take note of current state of some sensors at each level. Step 3: Switch to manager's side and change the state of the sensors you are taking note of. Step 4: Verify that the sensors you took note of changes to the new state. **Expected Result:** ● All sensors that get changed should be accurately displayed and change the map for the user. ● As soon as the sensor state changes, a new map is generated for the user

Testing the License Plate Reader:

Goal: The System must be able to successfully analyze a picture of a car's license plate and parse the license plate information accurately.

Test Cases:

System analyzes License Plate

Test-case Identifier: TC-4 **Use Case Tested:** UC-2 and 7 **Pass/fail Criteria:** The test passes if the system can successfully parse the license plate from a picture our camera took. **Input Data:** Picture of the rear of a car
Test Procedure: Step 1: Input a picture of a rear of a car Step 2: Take note of the license plate information for the car Step 3: Run the picture of the car through our computer vision API Step 4: Compare the output from the computer vision API with the license plate info we took note **Expected Result:** ● The license plate information that our computer vision API parses for us is the same as the license plate information in the picture

Testing Classes and Methods:

Goal: The classes and methods in our system are responsible for making the units work, these are the tools that the units use to accomplish their task.

Test Cases:

- ➔ ■ Refer to: Data Types and Operational Signatures
  - ◆ These are the classes and methods we will be testing
  - ◆ To test these we will see this we will use a combination of print and if statements to see if the output is valid

Integration Testing Strategy

Integration tests involve the testing of modules to see if the modules work with each other. For a Django project, this means that the different parts of the project – models, views, and URLs – work together. This is because Django receives an HTTP request and parses the structure of the URL to find a match in the mysite/urls.py file. Then, it relays that information to a specific Django view where the logic is processed and passed onto the template engine to render some client-side response. During the view construction phase, many times the database is queried or updated. So, if each view, in each Django app works, then integration test for Django project is successful. Inherently, testing the Django "view" output will require the use of the HTML, CSS, and JavaScript files we created, so it will test those component integration as well.

History of Work, Current Status, and Future Work

Current Status

After the completion of Report 2 and Demo 1, the team worked hard to implement all of the use cases, refactor the code to include the newly taught design patterns, and design tests. However, we suffered a major setback when one of our group members, Peter Luo, dropped the course. This setback prompted us to evaluate what was left to implement and whether we had enough resources to execute all parts of the project. We decided to narrow our scope and leave some use cases for future work. In particular, we decided that recommended spot and manager dashboard, displaying analytics, would be tabled for future work. After this decision, we were able to refactor our code, implement all use cases (except the ones listed above), clean up the user-side web interface, test individual components, test component integrations, and much more. We were able to fully integrate the license plate reader 3rd-party software into the system so that the system could automatically determine the type of customers, time of stay, and charge automatically. All of this was done prior to Demo 2. Since Demo 2, we continued to optimize the time that it took for the update_occupancy() function, improved our logic in views.py, removed unnecessary static components to decrease load time of web-pages, revised the comments, and etc. In addition, the key focus since demo 2 has been to enhance report 3 and improve the content— based on the TA feedback and meeting with the TA.

History of Work

Due to the magnitude of the scale, we sought to use a 3rd party Gantt chart application to meticulously note the deadlines and the current progress (https://app.smartsheet.com/b/publish?EQBCT=1bec5fd57c134f3f89dd6f8674c7f04c). Most of the milestones were based off of Dr. Marsic's deadlines because those were already set. Some milestones were team milestones needed to complete the bigger milestones. Thus, the timeline for the milestones were follows:

Milestone	Completion Date
Report 1	2/19/2017
Math	2/22/2017
Web Site Design	3/8/2017
Database Design	3/9/2017
Report 2	3/12/2017
Demo 1	3/24/2017
Demo 2	4/24/2017
Software Testing	4/25/2017
Third Report	5/1/2017

The first few weeks of the project, we made an unrealistic timeline of what we expected to accomplish. We wanted to get ahead of the curve, but it took us a few weeks to learn Django, set up the subteams, create a system of communication, and create the shared API. The big breakthrough in the first report occurred when we all took the time to

meetup and whiteboard all the use cases, user interaction, and system details. At this point, we were behind the aggressive timeline we created but we were able to catch up once we had a clear vision of the product. In Report 2, we had our deadlines planned out so that they were feasible and took advantage of Spring break. We were ahead of the established timeline when we completed Demo 1. However, with the departure of Peter, we initially fell behind on the timeline for Demo 2 and Report 2. Once we reevaluated our scope for the project, we were able to hit all the deadlines. We created We were able to follow the timeline set in Reports 1 and 2. Overall, we were able to maintain the deadlines in Reports 1 and 2.

Key Accomplishments:

- Fully functional parking system that can recognize and automatically charge registered users for their time of stay
- Ability for system to provide guest users with a similar automatic payment system
- Clean, well-designed responsive user web interface to make an account, add/update/remove information, view payment history, and much more.
- Ability to view occupancy map, for both users and management
- Management website to view lot details, add/remove/update details, enable/disable spots, view lot status, and much more

Future Work

There are many possible ways to continue developing on the initial concept. As alluded by the casual use case descriptions in the Functional Requirements section, two possible ventures include a recommended spot feature and manager analytics dashboard. The recommended spot can help alleviate additional congestion in the parking lot cause when the user is unsure where to park. The occupancy data we already have can be used to determine the best spot. In the Algorithms and Data Structures section, we described a potential procedure for finding a recommended spot.

In addition, we would like to incorporate a manager analytics dashboard to give the manager additional insights into their business. It would be a quick, one-stop shop for visualizations of daily/monthly/annual revenue, daily/monthly/annual number of customers, average stay time, and projected monthly/annual revenue — based on a best-fit line. Managers should also have a feature that allows them to change the cost of parking (per hour).

Future work for this project would entail the creation of native apps for iOS and Android for customers of the garage. This is because we would like to send push notifications for the recommended spot feature. It would allow the user to quickly access the necessary information, rather than fumbling through the website. These are some of the features we would like to implement in the future.

References

1. Marsic, Ivan. Software Engineering. 2012.
    a. Referenced the tic-tac-toe example as a basis for how to go about problem breakdown
    b. In addition, we utilized the online resources (website) associated with the textbook
2. Bruegge, Bernd, and Allen H. Dutoit. Object-oriented Software Engineering: Using UML, Patterns, and Java. Boston: Prentice Hall, 2010
    a. Consulted to get a better understanding of UML and view examples
3. Free flowchart maker and diagrams online." RSS. N.p., n.d. Web.
    a. <https://www.draw.io/>
    b. We wanted to view more examples of sequence diagrams before we started building ours
    c. The tools on the website were used to construct our sequence diagrams
4. "Minimal Wireframe Tool" Web.
    a. <https://wireframe.cc/>
    b. Used examples and the provided tool to create wireframes for the UI specification section
5. User Story Points
    a. https://www.atlassian.com/agile/estimation
    b. The website was consulted to determine how story points were awarded and the number awarded to each problem size
6. License Plate OCR
    a. https://github.com/openalpr/openalpr
    b. We were unsure about the current industry procedure for recognizing license plates. Furthermore, we were looking for an open source tool, from which we could better understand the recognition procedure—through viewing their code. We were able gain a strong understanding through their documentation, and we successfully integrated that into our code base.
    c. This allowed us to recognize car licence plate, make, model, color, and etc.
    d. We were also able to confirm that an image can be taken from various angles, not just straight.
7. Non-Functional Requirements

   a. <http://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2012-g3-ProjectFiles.zip>
   b. After much discourse, we were struggling to determine how to structure Non-Functional Requirements because most examples listed them as REQ list but we were using user stories. We consulted 2012 Group 3's documentation.
   c. Despite some years, non-functional requirements are similar to 2012 iteration of garage the automation track
8. Gantt Charts
   a. <http://en.wikipedia.org/wiki/Gantt_charts>
   b. We were familiar with the term, but we needed to understand why product managers used this tool frequently.
   c. Gained a step-by-step understanding of how to construct a Gantt chart
9. Effort Estimation Using Use Case Points
   a. <http://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2011-g2-report3.pdf>
   b. We used this as our basis for determining the effort estimation for our use cases.
   c. We used their approach using UAW, UUCW, TCF, and ECF to calculate Use Case Points
10. UML Domain Analysis Model
   a. <http://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2012-g4-report3.pdf>
   b. <http://eceweb1.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2013-g5-report3.pdf>
   c. Used 2012 Group 4 and 2013 Group 5's domain analysis models and edited it to take into account our new features.
11. Determining the Architectural Style
   a. <https://msdn.microsoft.com/en-us/library/ee658117.aspx>
   b. <https://en.wikipedia.org/wiki/Software_architecture#Architectural_styles_and_patterns>
   c. The report guide asked us to detail the Architectural Styles employed, to construct the system design, using Google search. So, we conducted a Google search and used the findings to direct and describe our design style.
12. Hardware Choices
   a. <http://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2012-g4-report3.pdf>

b. We decided this report had logical ideas for the specific hard drive, storage, and server choices.

13. Design of Tests
    a. <http://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2013-g5-report3.pdf>
    b. We followed this report's Test Cases format, and we used one blurb about Testing Classes and Methods.
14. Functional Requirements - Traceability Matrix
    a. http://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2013-g5-report3.pdf
    b. Since the book didn't provide explicit methodology, we sought to consult a previous group
15. Object Constraint Language (OCL) Contracts
    a. <http://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2012-g3-report3.pdf>
    b. This report's OCL matched our ideas when designing our system's structure