

Garage Automation

Software Engineering – 14:332:452

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

URL:

<https://vatsal09.github.io/Garage-Automation/>

Submission Date:

March 12, 2017

Effort Breakdown for Report 1

	Vatsal	Kendric	Gao	Peter	Arthur	Yunqi	Guy
Project management	60%	20%					20%
Customer Statement of Requirements	30%	50%	20%				
System Requirements	40%	10%	10%	10%	10%	10%	10%
Functional Requirements Specification	15%	10%	15%	15%	15%	15%	15%
User Interface Specs				15%		15%	70%
Domain Analysis		50%	30%			20%	
Plan of Work	60%		40%				

Effort Breakdown for Report 2

	Vatsal	Kendric	Gao	Peter	Arthur	Yunqi	Guy
Interaction Diagrams			33%	16.5%		16.5%	33%
Interaction Diagram Descriptions			33%	16.5%		16.5%	33%
Class Diagrams and Interaction Specifications		16.5%	16.5%	16.5%		16.5%	33%
Data Types and Operation Signatures			33%	33%			33%
Traceability Matrix					100%		
Architectural Styles	100%						
Identifying Subsystems	100%						
Mapping Subsystems to Hardware							100%
Persistent Data Storage							100%
Network Protocol		100%					
Global Control Flow				100%			
Hardware Requirements		100%					

Algorithms			50%			50%	
Data Structures	50%					50%	
User Interface Design and Implementation		25%		25%			50%
Design of Tests		33%			33%	33%	
Merging of Contributions	14%	14%	14%	14%	14%	14%	14%
Project Coordination and Progress Report	100%						
Plan of Work	100%						
Breakdown of Responsibilities	14%	14%	14%	14%	14%	14%	14%

Table of Contents

Group Information	1
Effort Breakdowns.....	2
Table of Contents	5
Customer Statement of Requirements	
Statement of Goals	7
Problem Statement	7
Proposed Solution (Essay Version).....	8
Proposed Solution (List Version).....	10
Glossary of Terms	12
System Requirements	
Enumerated Functional Requirements	14
Enumerated Nonfunctional Requirements	15
On-Screen Appearance Requirements	16
Functional Requirement Specification	
Stakeholders.....	20
Actors and Goals.....	21
Use Cases	22
Use Case Diagram	23
Traceability Matrix	24
Full-Dressed Descriptions	25
System Sequence Diagrams	36
User Interface Specification	38
User Effort Estimation	45
Domain Analysis	
Concept Definitions	48
Association Definitions	50
Attributes Definitions	53
Traceability Matrix	54
System Operation Contracts	57

Interaction Diagrams	59
Class Diagram and Interface Specification	
Class Diagram	66
Data Types and Operation Signatures	68
Traceability Matrix	72
System Architecture and System Design	
Architectural Styles	74
Identifying Subsystems	75
Mapping Subsystems to Hardware	76
Persistent Data Storage	77
Network Protocol	77
Global Control Flow	77
Hardware Requirements	78
Algorithms and Data Structures	
Algorithms	79
Data Structures	81
User Interface Design and Implementation	82
Design of Tests	87
Project Management and Plan of Work	
Progress to Date Summary (Report 2)	91
Product Ownership	92
Merging Contributions	92
Project Coordination and Progress Report	92
Plan of Work	93
Breakdown of Responsibilities	94
References	96

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 with suggested parking spots. This new procedure aims to provide an efficient car flow, minimize nearby traffic, and provide a quicker customer turnaround—thereby increasing occupancy and revenue.

Problem Statement

Our customer owns a multi-level, commercial parking garage without an automated system to check for occupancy logistics or a fast-track payment processes. Additionally, 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. In the current system, drivers 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. Moreover, larger parking garages can result in greater inefficiency if managed incorrectly; larger lots require more time to traverse and congestion affects more consumers.

As an owner of the parking lot, the new system will be based around a website or mobile application 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 monetary cost of parking, available parking spots, and an overall recommended parking spot. To achieve this, the system uses laser 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 lot, 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. There will also be a visual on the app to indicate the recommended parking spot for the user.

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, and allow entrance. Upon exiting the parking lot, an exit license-plate-reader will identify the car's license plate, calculate expenses, 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 a management-side dashboard. The web-based dashboard serves 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.

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 number of available parking spots and visual occupancy within the parking lot.
5. The system shall be able to recommend a spot to a user, based on current parking lot availabilities.
6. The system shall be able to change the recommended spot to the user if the spot is occupied by another car in transit.
7. 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.
8. The system shall be able to create guest accounts by linking the license-plate-reader information with the swiped credit card by guest user.
9. 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.
10. 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.
11. The system shall be able to charge the customer automatically upon exiting at the turnstile by analyzing the license plate and execute a charge on an account (registered or guest) linked to the system.
12. If the user is registered, the system shall save transaction history.
13. 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.
14. The system shall have a web-based portal where the owner/management will have access to real-time occupancy data, i.e. visual indication of which spots are available and which ones are occupied.
15. The management portal should also provide an analytics portion where there will be key metric information, including the 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.

16. The manager can change the price per hour of a parking spot.

17. The manager can disable a parking space.

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 parking system.
Time of Stay	Duration of time between when the user enters the parking lot and when the user exits the parking lot.
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.
User/Registered User	A customer who has a registered account in the system. These users have entries and payment history stored in the database.
Guest User/Guest Account	A non-registered user of the service. These temporary users are not stored in the database entries.
Cash User	A customer who uses the ticketing system and pays for the parking in cash.
Dashboard	A control panel GUI that organizes and displays information in user-friendly manner.
GUI	A user interface that allows the user to interact with the system using images rather than text commands; acronym for graphical user interface.
Manager/management	The owner/management of the parking lot and parking system. They are able to view manager dashboard, fix rates, disable certain spots, and etc.

Mobile Application	An application designed to be used on a mobile device such as a smart phone.
Occupancy Rate	The rate at which the parking spaces are used compared to the total amount of available space.
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.
Payment History	History of locations, durations, and payments for utilizing parking lot.
Sensors	A hardware device used to detect if a parking spot is taken.
Website Application	An application designed to be used on a web browser.
License-Plate-Reader	A combination of hardware (camera) to take pictures of a license plate and computer vision to detect the license plate number.
User Accessibility	Design process to allow all users to interact with the system efficiently and with ease.
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.

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 an online portal where I can find key metric information, including the 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	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
ST-11	As a manager, I will be able set parking prices	2

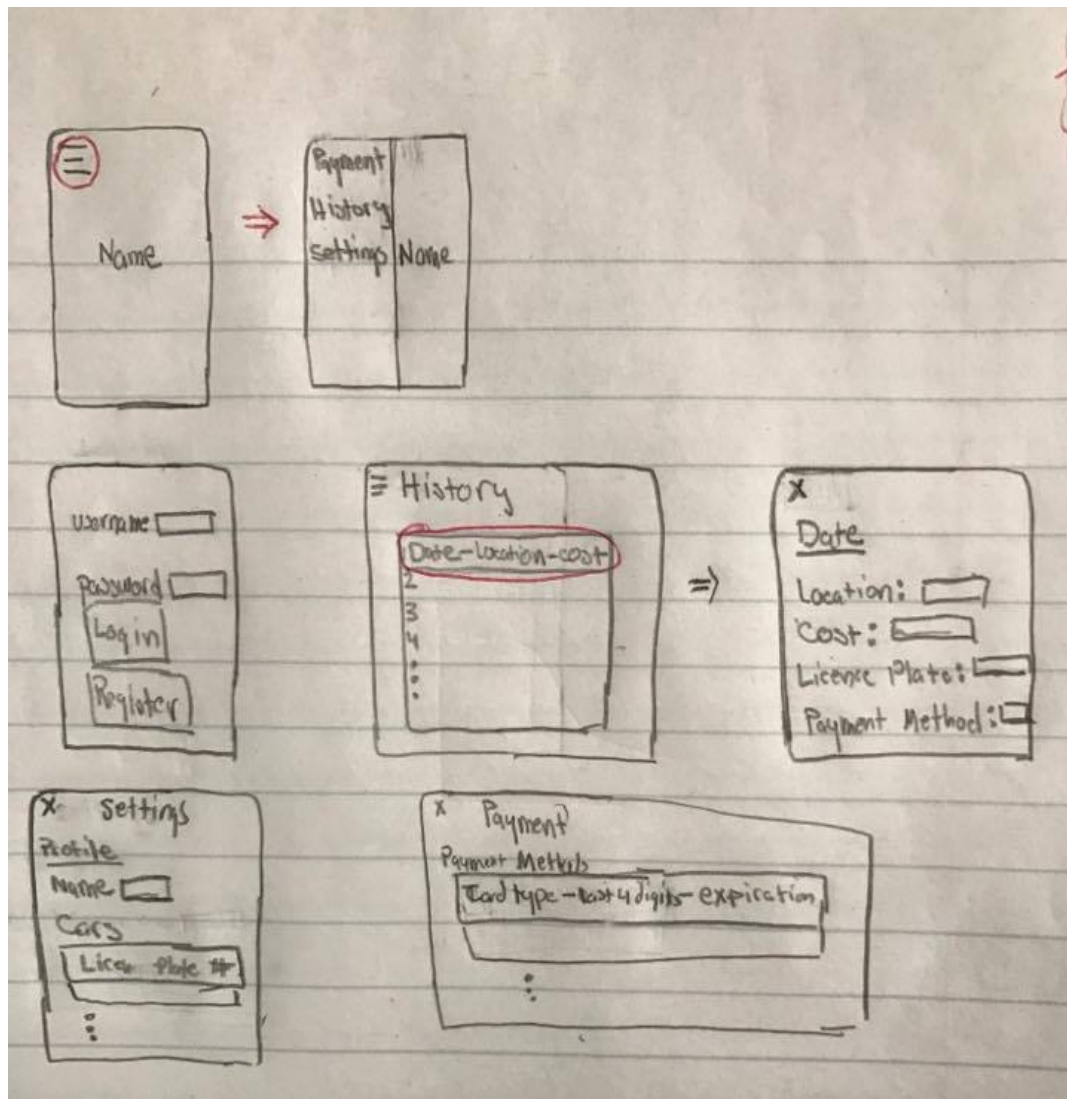
Enumerated Nonfunctional Requirements [6]

FURPS	
Functionality	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• This design is very adaptable as far as the garage size is concerned.<ul style="list-style-type: none">○ 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.<ul style="list-style-type: none">○ 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

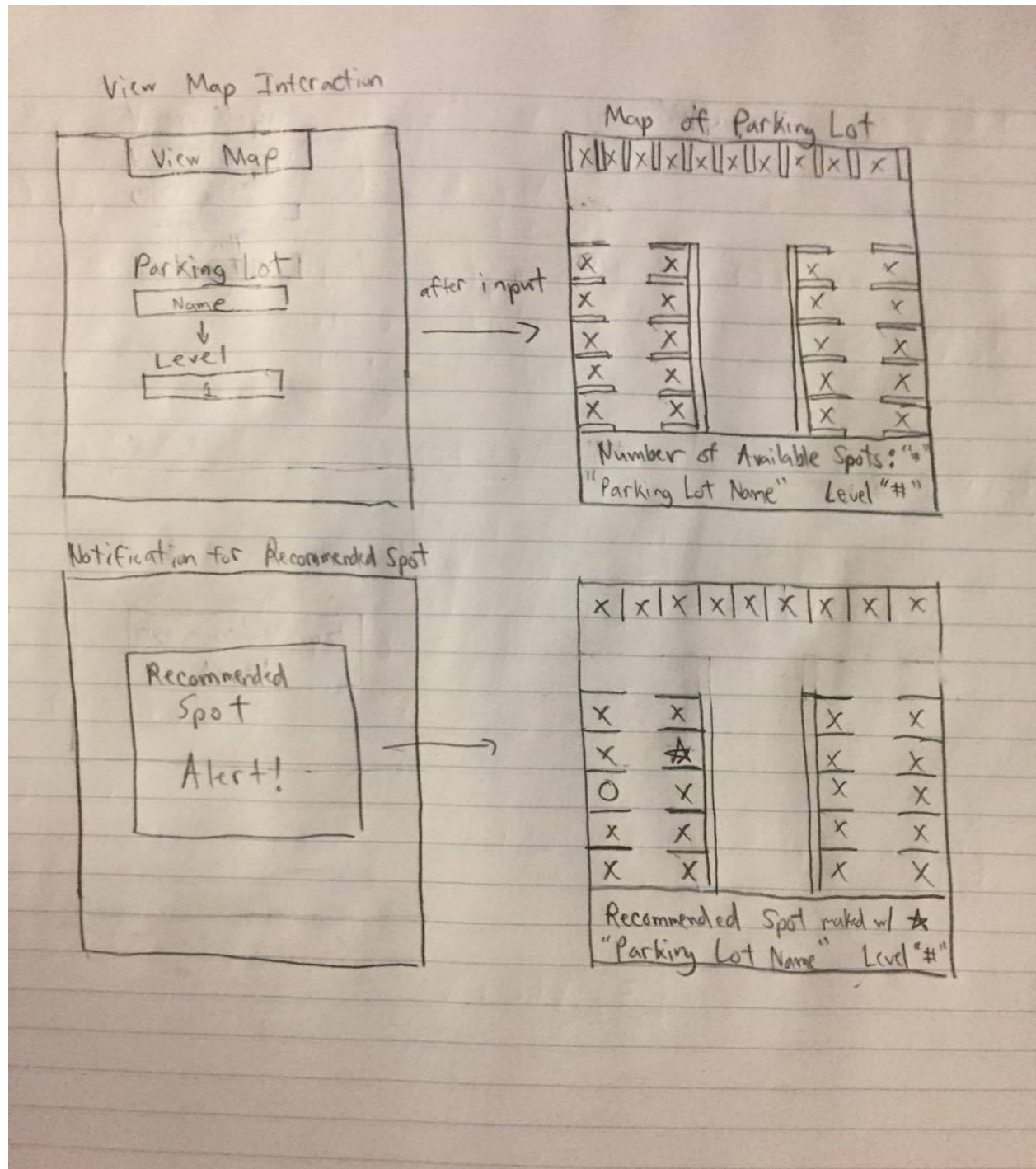
On-Screen Appearance Requirements

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



Manager Dashboard

Analytics Login

Username:

Password:

Login

Past Customers & weeks ☐ Avg Stay

week 1:

week 2:

week 3:

Calendar

Heat Graph

Total profit today

total customers

Average length of stay

% change month vs last month

Past 10 WEEKS Customers

Average stay of customer past weeks

Total profit ~~rest~~ & Total Customers

Calendar with Profit & customer at the day

% change in profit last month vs this month

Heat graph of amount of traffic

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, monitor manager-side dashboard, 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 set the parking prices, monitor the management-side dashboard, disable spots when needed, and managing employees
Customer	To park the car
App	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
Internet Site	To allow management to view occupancy statistics, revenue, and display current occupied/unoccupied parking spaces
License Plate Reader	To extract license plate number
Vacancy Display	To display the number of vacant spots within the garage
Turnstile	To stop the car
Sensors	To find out whether a spot is vacant or not
Credit Card Machine	Terminal to accept credit card payments

Use Cases

Casual Description

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

Use Case Diagram



Traceability Matrix

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

Fully-Dressed Descriptions

Use Case UC-1: Register
<p>Related Requirements: 1, 6, 9, 11</p> <p>Initiating Actor: Customer</p> <p>Actor's Goal: Create an Account that will be stored in the database and allow for faster entry.</p> <p>Participating Actors: Application, Database</p> <p>Preconditions: The application will request all required information.</p> <p>Postconditions: The user's account will be stored in the database.</p> <p>Flow of events for Main Success Scenario:</p> <ul style="list-style-type: none">→ 1. Customer accesses the website and chooses the "Register" option.← 2. The system returns the display that states the required information.→ 3. The customer fills out the required data fields.← 4. The system takes the information to verify it.If not valid, move back to 3 If valid, continue→ 5. Information is stored into the database.

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, Application, Database

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 system will scan the license plate
- ← 3. The system will request the a payment method
- 4. The customer will swipe a credit/debit card
- ← 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.

Flow of Events for Extensions (Alternate Scenarios):

3a. The license plate is registered to an account

- ← 1. Same as in Step 5 Above

4a. The provided credit/debit card is invalid

- ← 1. Application detects invalid payment and signals System, System 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
- ← 2. System signals Application to mark license plate as cash user
- ← 3. System prints ticket
- 4. Customer takes ticket
- ← 5. System raises turnstile

5a. The credit/debit card is not registered with a valid account

- ← 5. Vehicle data and credit/debit card are temporarily stored together in the database and System raises the 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: Application, Database

Preconditions: The user has an account with a car listed.

Postconditions: The user's account will have one or more cars removed.

Flow of events for Main Success Scenario:

- 1. Customer goes to settings page
- ← 2. The system displays account information, including registered vehicles.
- 3. The customer presses the X button correlating to the vehicle they want to remove
- ← 4. The system requests a confirmation
- 5. User confirms
- ← 6. Application request Database to remove the vehicle data from the account

Use Case UC-4: Delete

Related Requirements: 16

Initiating Actor: Customer

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

Participating Actors: Application, Database

Preconditions: The application will request all required information.

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

Flow of events for Main Success Scenario:

- 1. Customer accesses the website and logs in.
- ← 2. The system returns the display that states their account information.
- 3. The customer presses the "Delete Account" button in the "Settings" page.
- ← 4. The Application presents a warning that this action is irreversible,
- 5. Customer confirms decision
- ← 6. The Application logs the user out and requests the data removed from the Database.

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: Application, Database

Preconditions: The user has an account.

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 website and chooses the "Add Payment" option.
- ← 2. The system returns a form for the user to fill in.
- 3. The customer enters payment info.
- ← 4. The system 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):

- 1. Customer accesses the website and chooses the "Settings" option.
- ← 2. The system returns current information.
- 3. The customer presses edit and fills in the fields.
- ← 4. The system validates new info, if invalid return to step 2 if valid, proceed.
- 5. Updated information is added to database.

Use Case UC-6: Given a recommended a Parking Space

Related Requirements: 4, 5

Initiating Actor: Customer

Actor's Goal: Finds a suitable parking space to recommend to a Customer by searching for the closest available parking spot to the Ground Level.

Participating Actors: System, Database, Customer

Preconditions: The parking lot has enough parking spaces to give to the user even with cars in transit possibly taking the user's recommended spot.

Postconditions: The user will be in their recommended parking spot, and the system will recognize the changes in occupancy state.

Flow of events for Main Success Scenario:

- 1. The customer successfully goes through UC-2 Arrive.
- ← 2. The system gives the customer a recommended parking spot via a notification on the application.
- 3. The customer accesses the application to view the recommended parking spot.
- 4. The customer goes to the recommended parking spot.
- ← 5. The sensor indicates that the parking spot is taken and the system updates that information.

Flow of Events for Extensions (Alternate Scenarios):

- 4a. The customer attempts to go the recommended parking spot, but someone took it instead.
- ← 5. The system researches the parking lot and gives you another parking spot.
- 6. The customer goes to the new recommended parking spot.
- ← 7. The sensor indicates that the parking spot is taken and the system updates that information.

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, Database

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. The customer approaches the turnstile.
- ← 2. The system will scan the license plate
- ← 3. The system will find a user account linked to license plate
- ← 4. The 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
- ← 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: System, Database, Application

Preconditions: The system has saved a list of previous transactions for the user.

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

Flow of events for Main Success Scenario:

- 1. The customer successfully logs in their account on the website.
- 2. The customer goes on in the view transaction history tab.
- ← 3. The system accesses the database for the collected history.
- ← 4. The system displays the data of the collected history
- 5. 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, Application, Sensor

Preconditions: The user has an account.

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

Flow of events for Main Success Scenario:

- 1. The customer successfully logs in their account on the app.
- 2. The customer goes in the view parking lot map tab.
- 3. The customer inputs the specific parking lot and level they want to view
- ← 4. The system accesses the database for that parking lot and outputs a display of the parking lot.
- 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-10: Change Price

Related Requirements: 13,14,15

Initiating Actor: Management

Actor's Goal: Change the Price/Hour of the parking lot

Participating Actors: App,Credit Card Machine

Preconditions: The correct credentials have been provided to prove the user has authority to change the price

Postconditions: The price of the parking lot will be updated

Flow of events for Main Success Scenario:

- 1.User is presented the login page and he/she enters the account credentials
- ← 2.The credentials are accepted and the analytic page is displayed
- 3.User clicks on the edit price button and enters the new price that he/she wants to decide.

Flow of Events for Alternate Scenario:

- 1.User is presented the login page and he/she enters the account credentials
- ← 2.The credentials are incorrect and the analytics page is not displayed.

Use Case UC-11: Analytics

Related Requirements: 13,14,15

Initiating Actor: Management

Actor's Goal: Analyze the parking lot's statistics

Participating Actors: Database, Internet Site, Management

Preconditions: The required information has been attained by the parking lot system and stored in the database.

Postconditions: The pertinent information will be displayed in an easy to understand format.

Flow of events for Main Success Scenario:

- 1. User is presented the login page and he/she enters the account credentials
- ← 2. The credentials are accepted and the analytic page is displayed
- 3. User 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.

Flow of Events for Alternate Scenario:

- 1. User is presented the login page and he/she enters the account credentials
- ← 2. The credentials are incorrect and the analytics page is not displayed.

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: Application, Database

Preconditions: The user has an account with more than one payment method.

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

Flow of events for Main Success Scenario:

- 1. Customer accesses the website and chooses the "Remove Payment" option.
- ← 2. The system returns the list of available payment options. If only one payment option displays "Add new payment option before deleting."
- 3. The customer checks which payment option to remove.
- ← 4. The system deletes the payment option, if none selected returns same page(step 2), if selected go to step 5.
- 5. Payment information is deleted from database.

Use Case UC-13: Change Availability of Spots

Related Requirements: 17

Initiating Actor: Manager

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

Participating Actors: Application, Database

Preconditions: There are spots marked as available in the database.

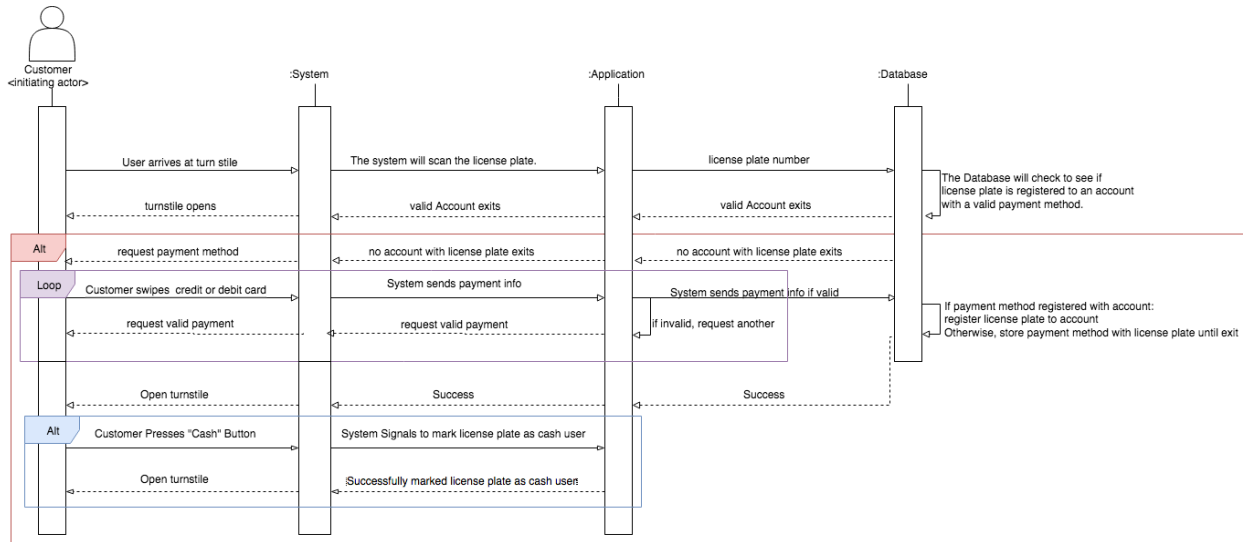
Postconditions: The desired spots are marked as unavailable (or back to available) to users.

Flow of events for Main Success Scenario:

- 1. The manager (Admin) accesses the website and chooses the "Change Spot Availability" option.
- ← 2. The system returns the map of all spots and the availability of each.
- 3. The manager marks desired spots as "unavailable" (or "available"), with an option to add a remark displayed to users.
- ← 4. The system marks those spots as unavailable/available in database.

System Sequence Diagram

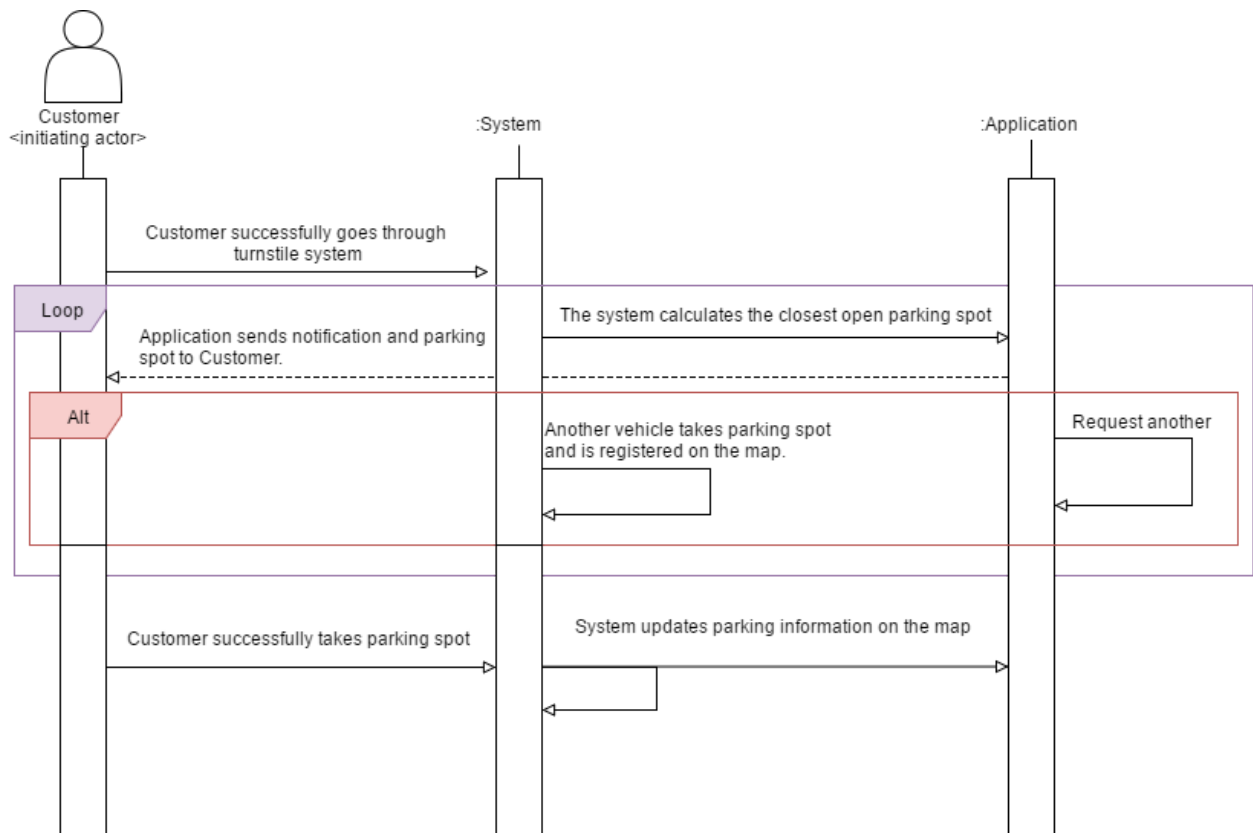
Use Case: UC-2



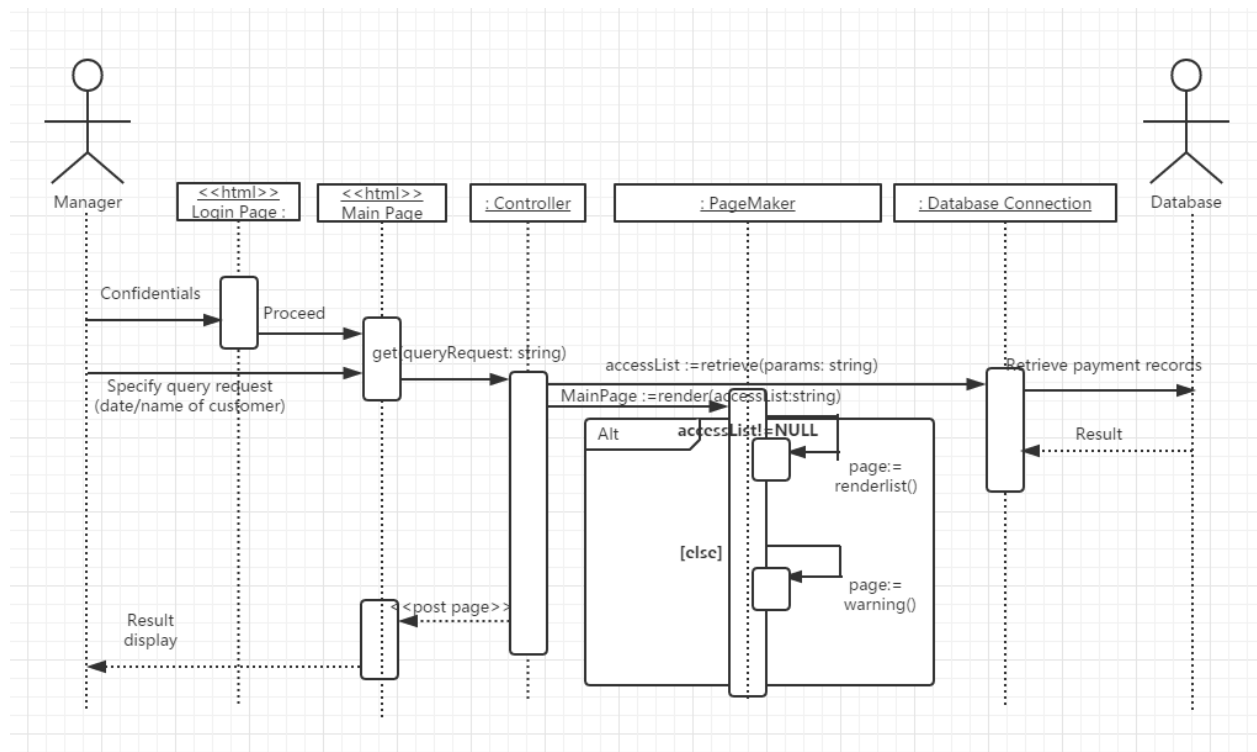
Please note that the diagrams are difficult to scale into the pdf. Please look in the “Diagram” folder in the GitHub repository for a high resolution image.

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: UC-6



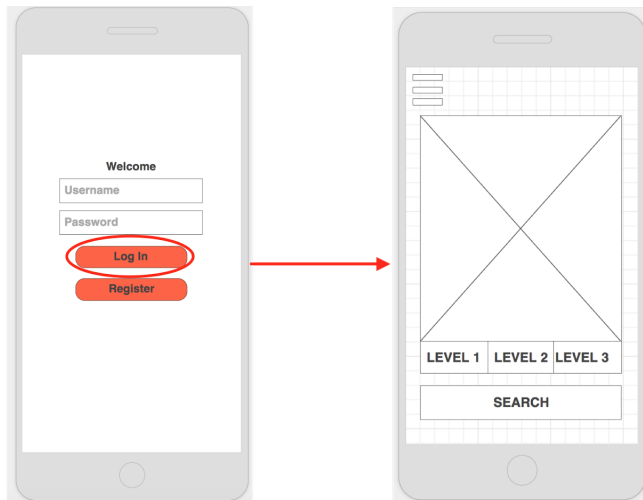
Use Case: UC-11



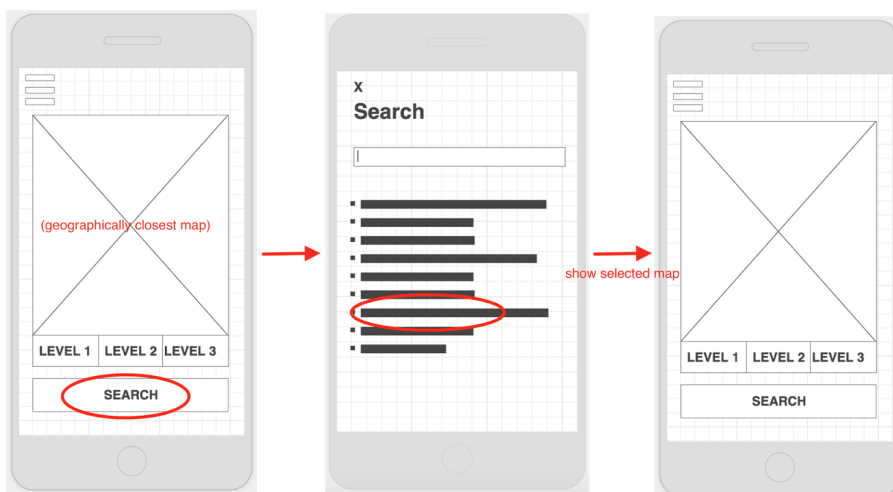
User Interface Specification

User Preliminary Design

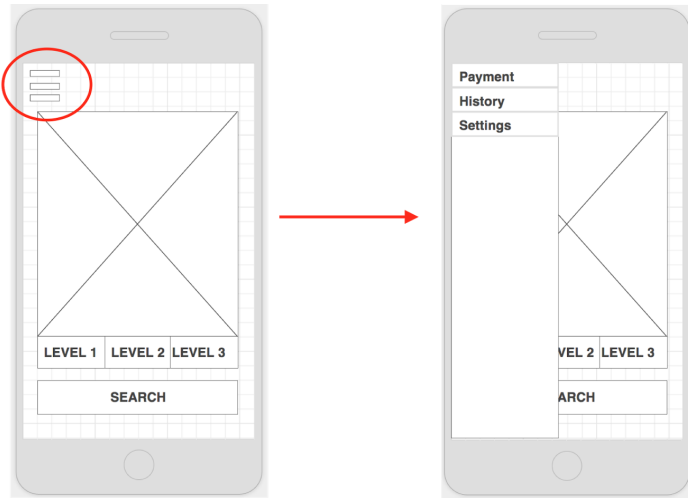
Application Functionality



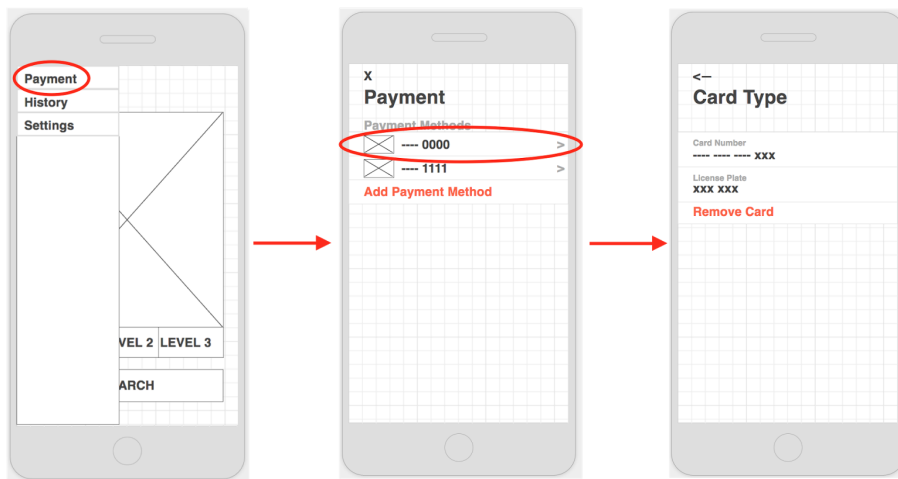
The user can log in or register from the Welcome page. When registering, the user will be required to provide a payment method. Then, will be directed to the home page. The home page displays the map of the geographically closest parking lot.



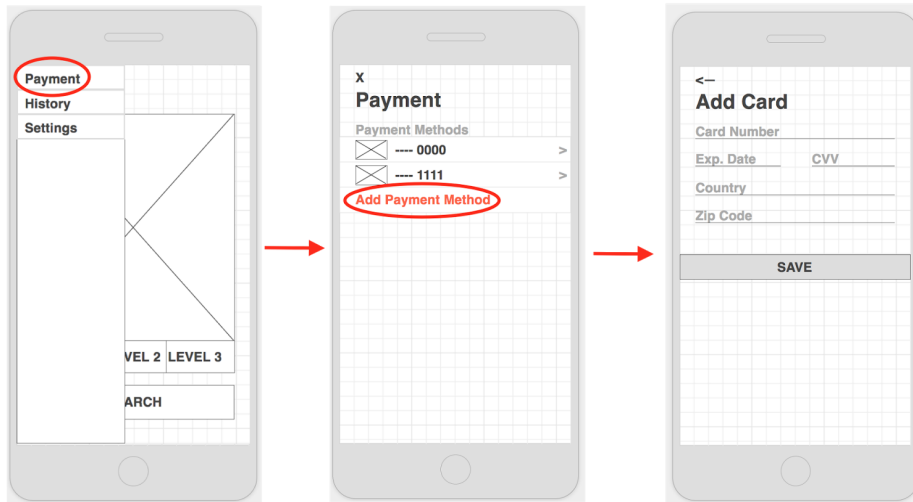
Pressing the Search Button brings up a search window. Here the user can search for a specific parking lot and select it from a list. Then they will be returned to the home page, where the selected parking lot's map will be displayed.



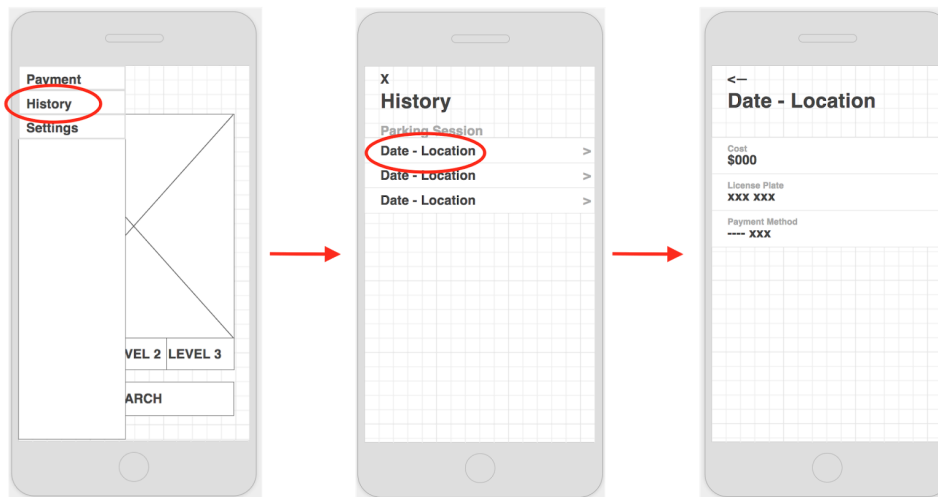
Pressing the menu button will bring up Payment, History, and Settings buttons



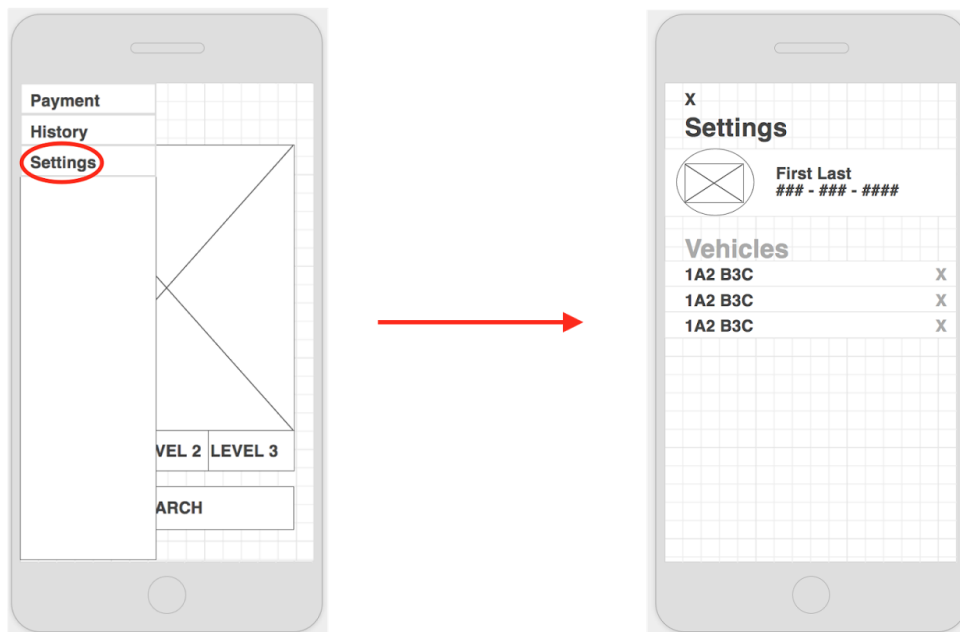
Pressing the payment button brings up the payment screen. Here the user has the option to review and remove payment methods. However, they must always have at least one valid payment method on their account.



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.



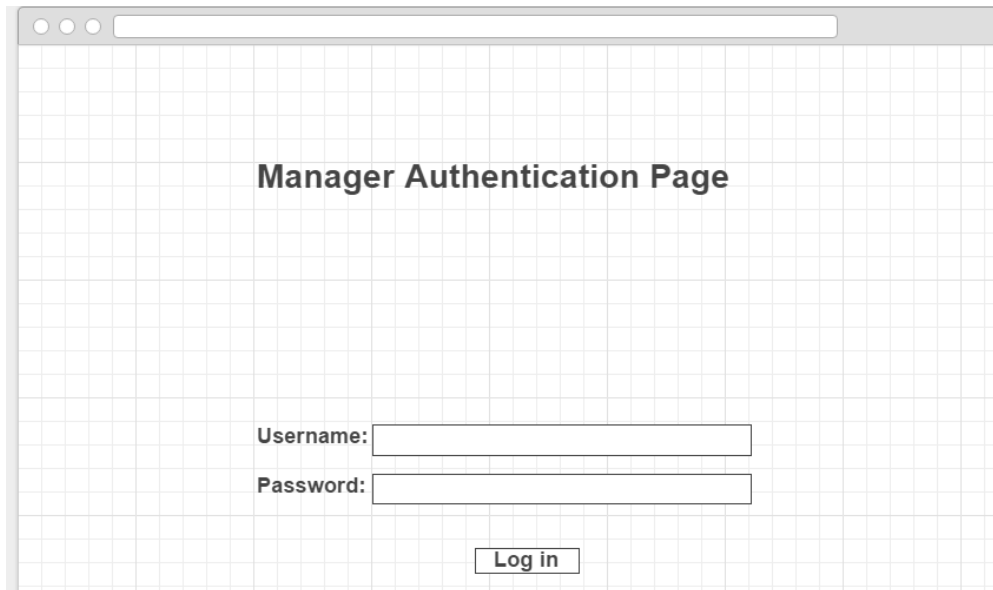
The settings page contains information regarding the user. This includes their name, phone number, and the license plates registered to their account. To remove a license plate from an account, the user presses the X next to the license plate.

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.

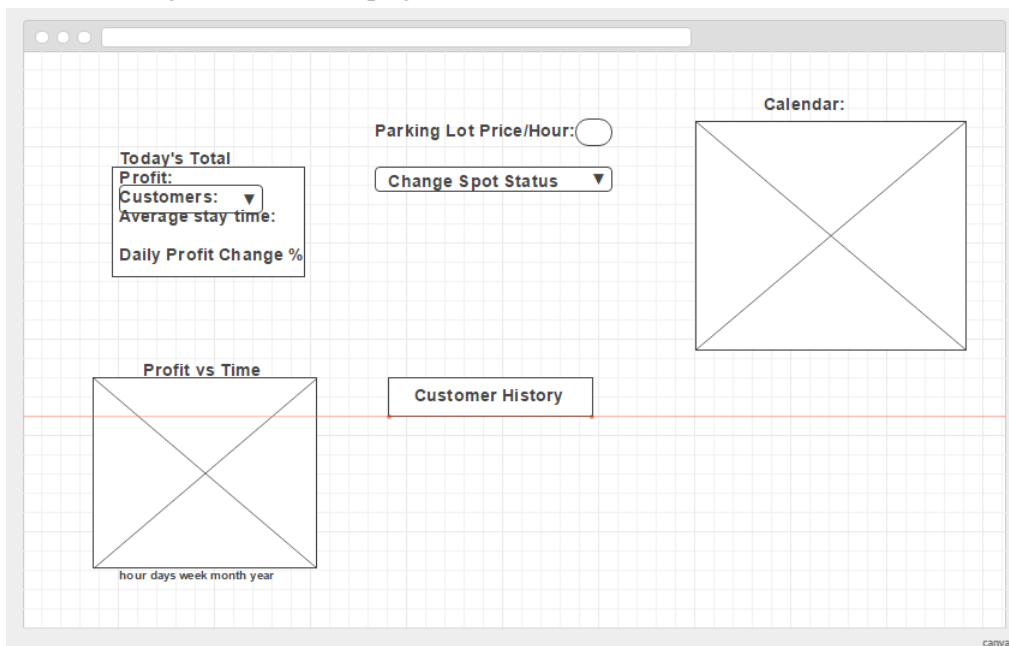
The system will handle registration of a vehicle upon arrival. When a customer arrives with an unregistered license plate, they will be asked to provide a credit or debit card. If this payment method is registered to an account, the license plate will be added to that account for future visits.

Manager Dashboard



The image shows a web browser window with a grid background. The title "Manager Authentication Page" is centered at the top. Below the title, there are two input fields: "Username:" and "Password:". Below these fields is a "Log in" button. The browser window has a standard address bar and window controls.

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



The image shows a web browser window with a grid background. The dashboard is divided into several sections:

- Today's Total:** A box containing "Profit:", "Customers:" (with a dropdown arrow), "Average stay time:", and "Daily Profit Change %".
- Parking Lot Price/Hour:** A text input field.
- Change Spot Status:** A dropdown menu.
- Calendar:** A large square box with a diagonal cross, representing a calendar.
- Profit vs Time:** A box with a diagonal cross, representing a graph. Below it, the text "hour days week month year" is visible.
- Customer History:** A button.

The browser window has a standard address bar and window controls.

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.

<div>Search by License Plate # <input type="text"/></div> <div>Search by time Day Month Year <input type="text"/> <input type="text"/> <input type="text"/> <div></div><div>Export</div></div> <div>Real Time Customers <div></div></div>

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.

User Effort Estimation

Accessing Management Analytics: 6 Clicks

Navigation: 1 Click

1. Click on the web page icon to open to login page.

Data Entry: 4 Clicks

1. User enters their username.
2. User presses "tab" key or click password data field.
3. User enters their password.
4. User presses "enter" key or click "Log in".

Analytics Navigation: 1 Click

1. User can click to view more information on License Plate information.

Accessing and Editing customer information: 8 Clicks

Log in: 4 Clicks

1. User enters their username.
2. User presses "tab" key or click password data field.
3. User enters their password.
4. User presses "enter" key or click "Log in".

Navigating to desired information: 2 Clicks

1. User clicks the top left menu icon.
2. User clicks "Payment" or "Settings" based on what they want to change.

Editing desired information: 2 Clicks

1. User selects the field they would like to edit.
2. User saves the information.

Accessing Parking Lot Information: 8 Clicks

Giving a recommended Parking Space: 1 Click

1. Clicking on the app will display the recommended parking spot.

Exiting the parking Lot: 0 Clicks

1. Nothing is needed to be done by the user

View History of Transactions: 4 Clicks

1. Click on the phone app/web app to open the login page.
2. Enter username and password.
3. Press enter/click login.
4. Navigate to history.

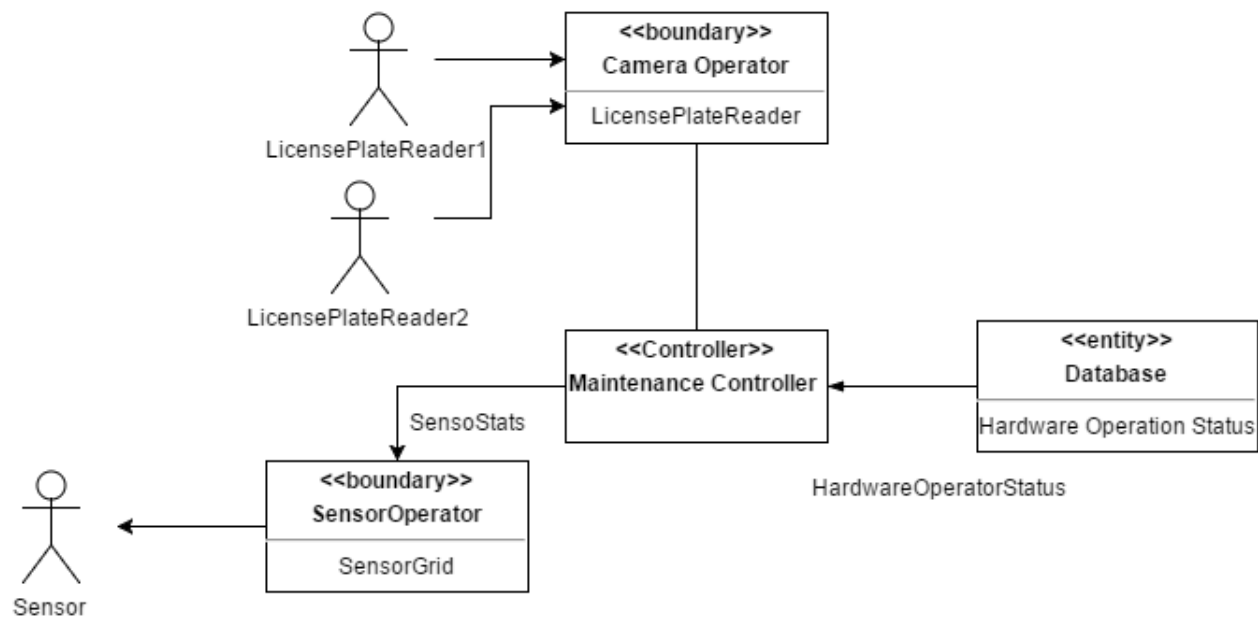
Check Occupancy of Parking Lot: 3 Clicks

1. Click on the phone app/web app to open the login page.
2. Enter username and password.
3. Press enter/click login.
4. Occupancy will automatically display at the right.

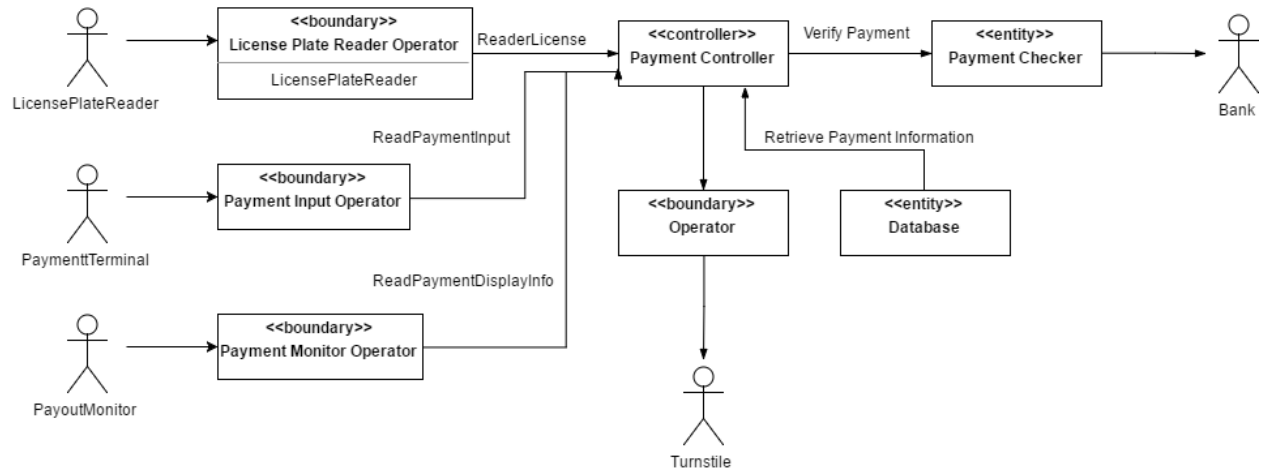
Domain Analysis

Domain Analysis UML Diagrams

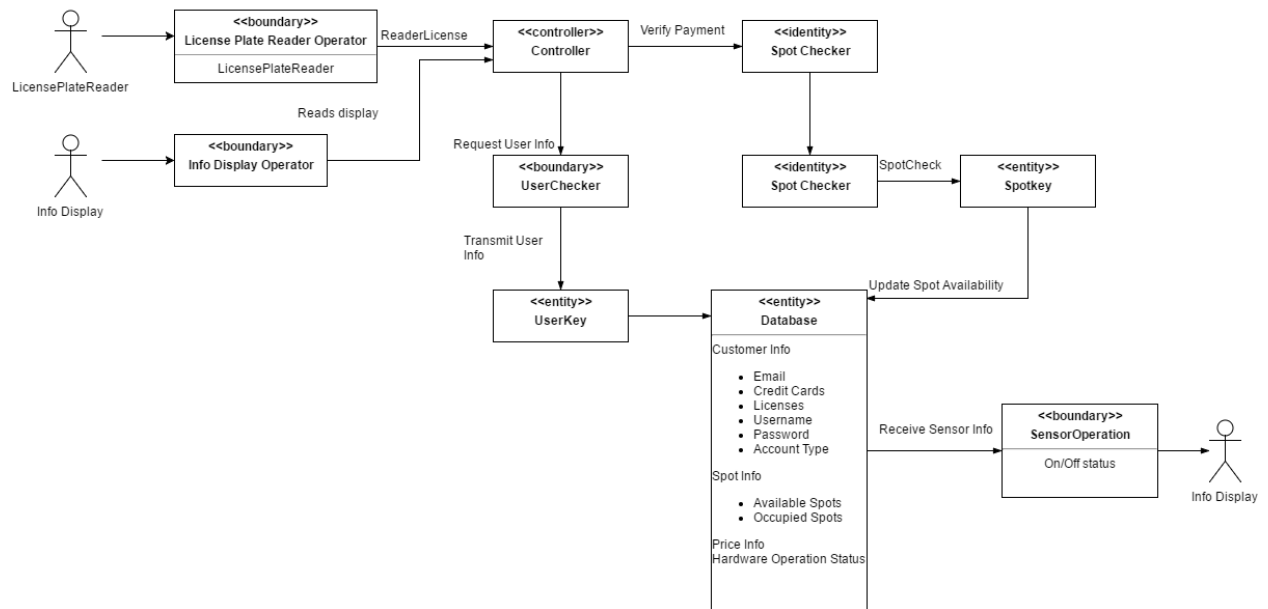
Maintenance Domain View



Exit Domain View



Park Domain View



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 from 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 notifications from hardware in the event of an abnormality, sends notifications to local service center and logs information in the Database	D	MaintenanceLogger
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
Receives input from FreeSpotKey and determines a vacant parking spot to recommend to the user who has passed through the turnstile. It can also change the spot if FreeSpotkey gives a notification that the spot has been taken by someone.	D	RecommendController
Container for current number of parking spots available on each floor and where their subsequent location is, collaborates with the Database.	K	FreeSpotKey
It will output necessary payment information through the Payment Monitor	D	PaymentMonitorOperator
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

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.	
MaintenanceLogger - SensorOperator	MaintenanceController receives notifications from SensorOperator in the event of any malfunction	SensorStatus
MaintenanceLogger - LicensePlateOperator	MaintenanceController receives notifications from LicensePlateOperator in the event of any malfunction	LicensePlateReaderStatus
MaintenanceLogger - ElevatorOperator	MaintenanceController receives notifications from ElevatorOperator in the event of any malfunction	ElevatorStatus
MaintenanceLogger - Database	Updates hardware statuses in the database	HardwareOperationStatus
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

iii. Attribute definitions

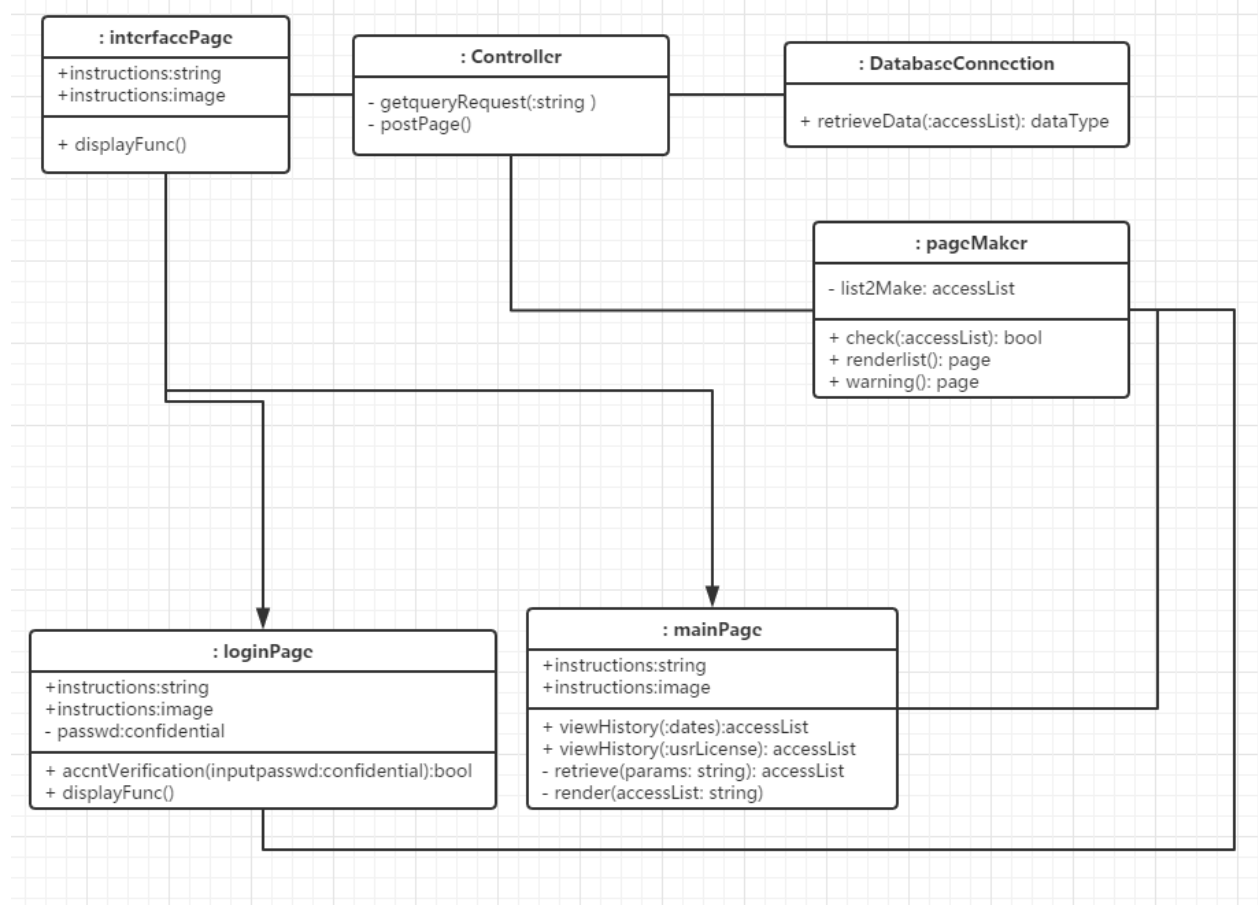
deviceName	Name of the hardware with the error
LicensePlateNumber	String of characters that represent the license plate information taken from the License Plate Readers
errorMessage	String pertaining to the type of error with the hardware
password	...
userID	...
SessionOwed	Amount owed that is taken 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
MaintenanceIssue	String to log into MaintenanceKey
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

[illegible]

Manager's Domain

I. UML Class Diagram



II. Concept Definition

Responsibility Description	Type	Concept Name
Coordinate actions of all concepts associated with Manager's analytics.	D	ManagerController
Contain login screen and verification of identity.	P	Login page
Contain responsive buttons by which the manager views the data analytic options.	K	Main page
Coordinate and display graphs and analytical information on the web page.	P	Pagemaker

Coordinate the connection to server database for fetching requested data.	P	Database Connector
---	---	--------------------

III. Association Definition

Concept pair	Association description	Association name
Controller - Pagemaker	Controller passes requests to pagemaker and receives back pages containing processed pages for display	Conveys requests
Pagemaker - Database Connection	Database Connection passes retrieved data to pagemaker for display	Provides data
Pagemaker - Main page	Pagemaker prepares the main page.	Prepares
Controller - Database Connection	Controller passes search requests to Database Connection.	Conveys requests

IV. Attribute Definition

Concept Pair	Attribute	Attribute Description
Controller	Price	Used to change the price per hour of the parking lot
	Search Customer Parameters	Used to search through history of customers. Includes date and time range.
	Date Parameters	The range of time that the analytics displayed should stretch. This can be range from a few hours up to years.
Log in Page	Username	Which account is attempting to log into the manager page.

	Password	Authentication that the account attempting to log in is the correct person.
PageMaker	Analytics	Used to convey the statistics on the parking lot business for managers to use.
MainPage	Graph Parameters	Changes the graph's date display range. Options include hours, days, weeks, months, and years.
Database Connector	Data Requested	The data that is requested from the Controller.

V. Traceability Matrix

Use Case	1	2	3	4	5	6	7	8	9	10	11	12	13
Controller								x	x	x	x		
Log in Page													
Page Maker									x	x	x		
Main Page													
Database Connector										x			x
PW								5	13	20	14		5

VI. System Operation Contracts

Operation	Perform Analytics
Precondition	The required information has been attained by the parking lot system and stored in the database.
Postcondition	The pertinent information will be displayed in an easy to understand

	format.
--	---------

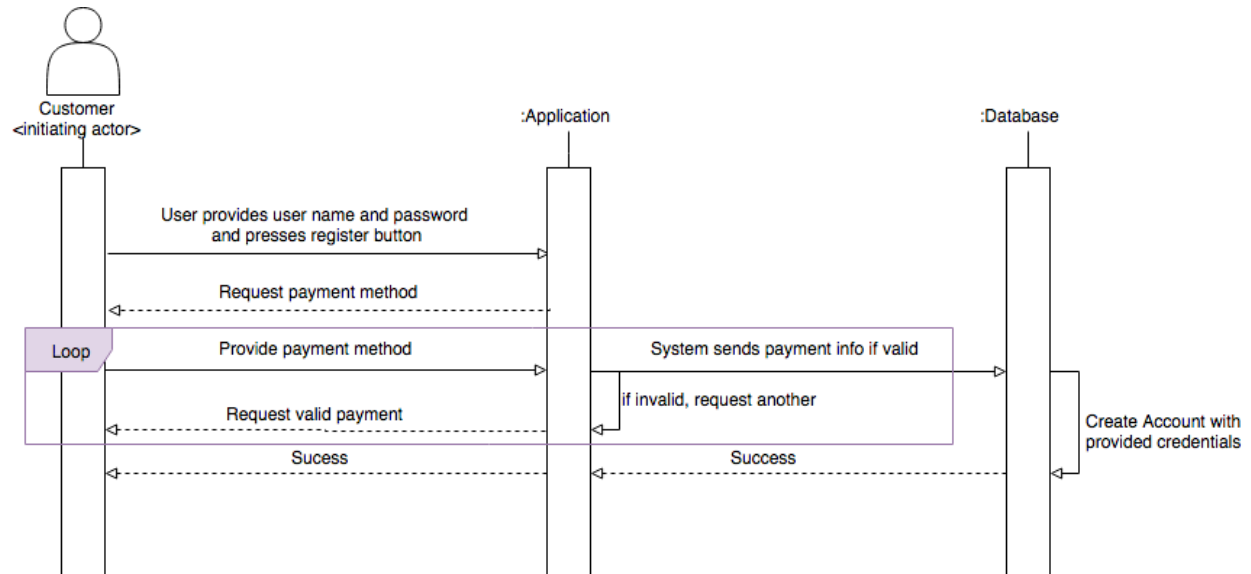
Operation	Authenticate Manager
Precondition	There is a database storing the username and passwords of managers so that the entered authentication parameters can be checked.
Postcondition	The manager attempting to log will know if the credentials he/she entered was correct.

VII. Mathematical Models

Regression may be used to study the customers' behaviors and predict the number of customers in a particular time period, based on real data collected in actual payments.

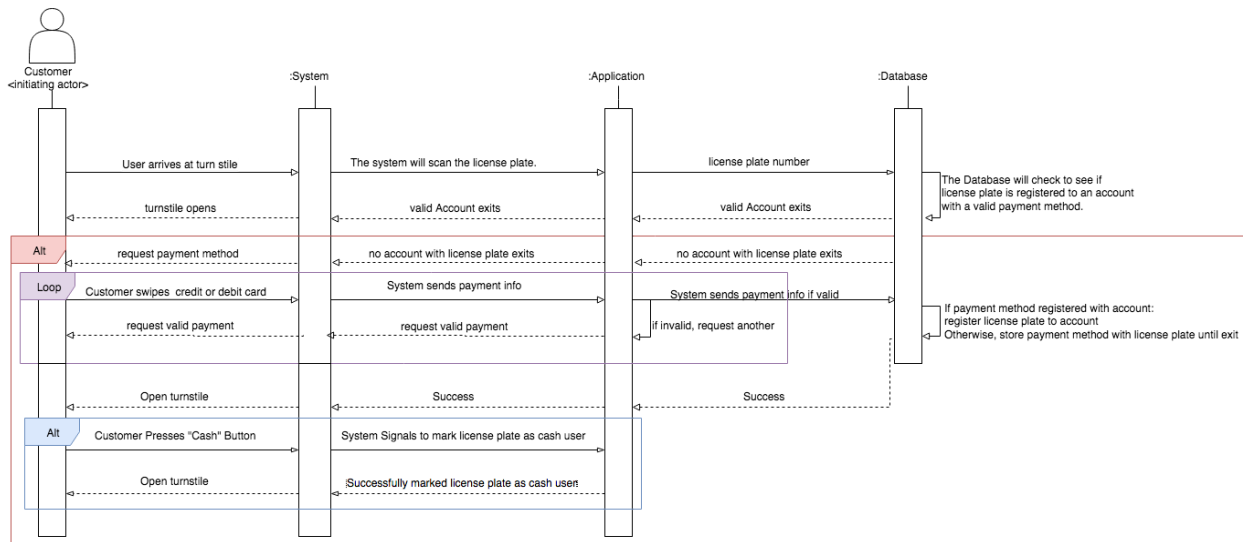
Interaction Diagrams

Use Case: UC-1



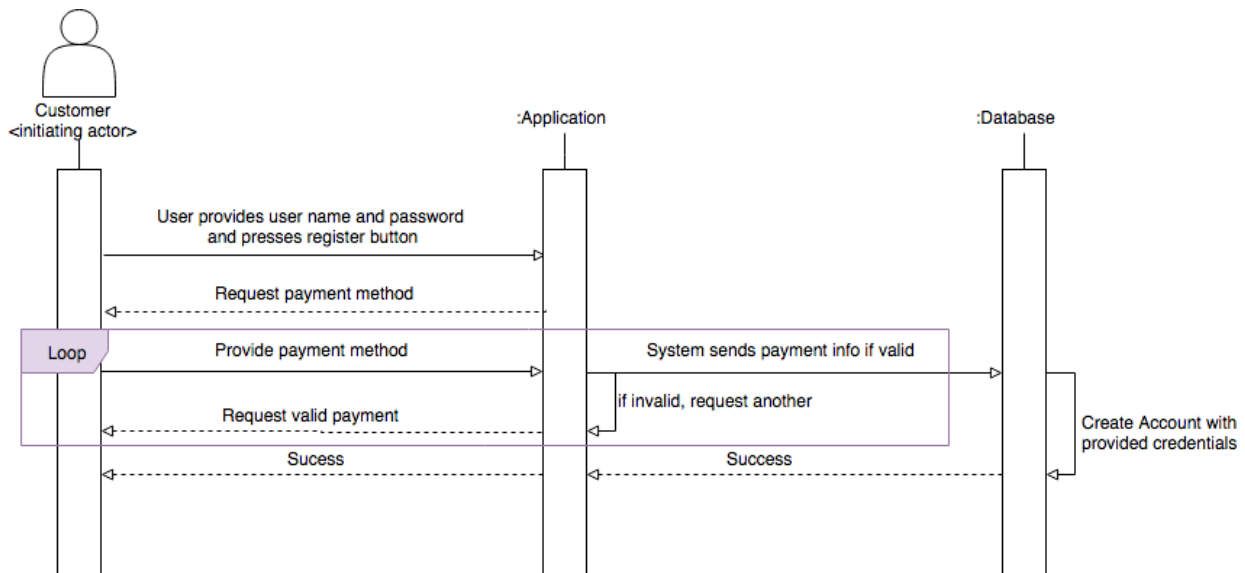
Design Principles: This design greatly focuses on high cohesion. The application mainly has a communication responsibility; it serves as the line of communication between the customer and the database. The database has a knowing responsibility; it simply stores the data provided by the application.

Use Case: UC-2



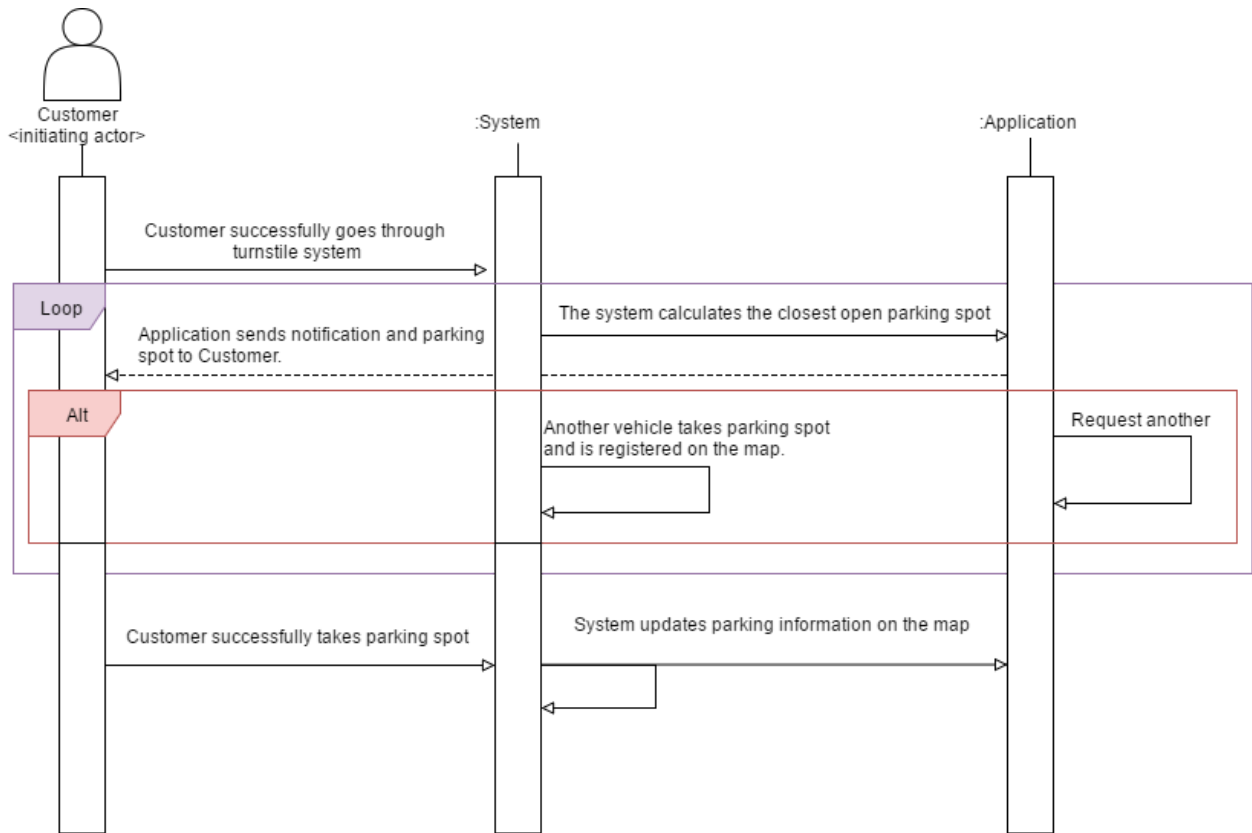
Design Principles: Here, the System takes on all the “doing” responsibility. Its job is only to control the process the data from the sensors and pass it to application. The application has the communicating responsibility; it mostly just transfers data between the System and the Database. The Database has a purely knowing responsibility and will only express what it knows to the Application.

Use Case: UC-3



Design Principles: The application mainly has a communication responsibility; it serves as the line of communication between the customer and the database. The database has a purely knowing responsibility; all it needs to do is remove existing information from itself.

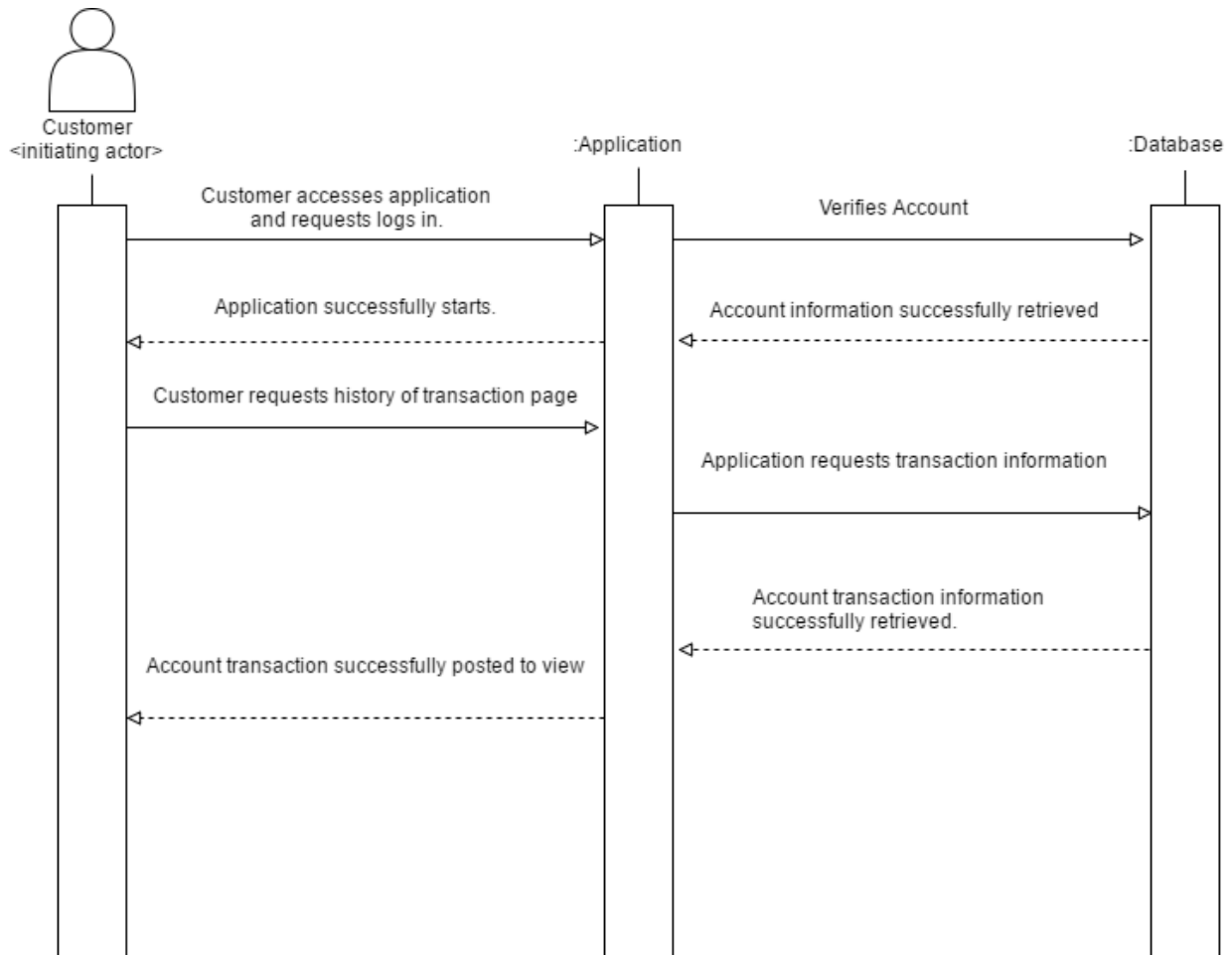
Use Case: UC-6



Design Principles:

The system communicates with the Recommend Controller to calculate the nearest parking spot to relay to the application. The RecommendController calculates the necessary information and relays it to the Application via the system.

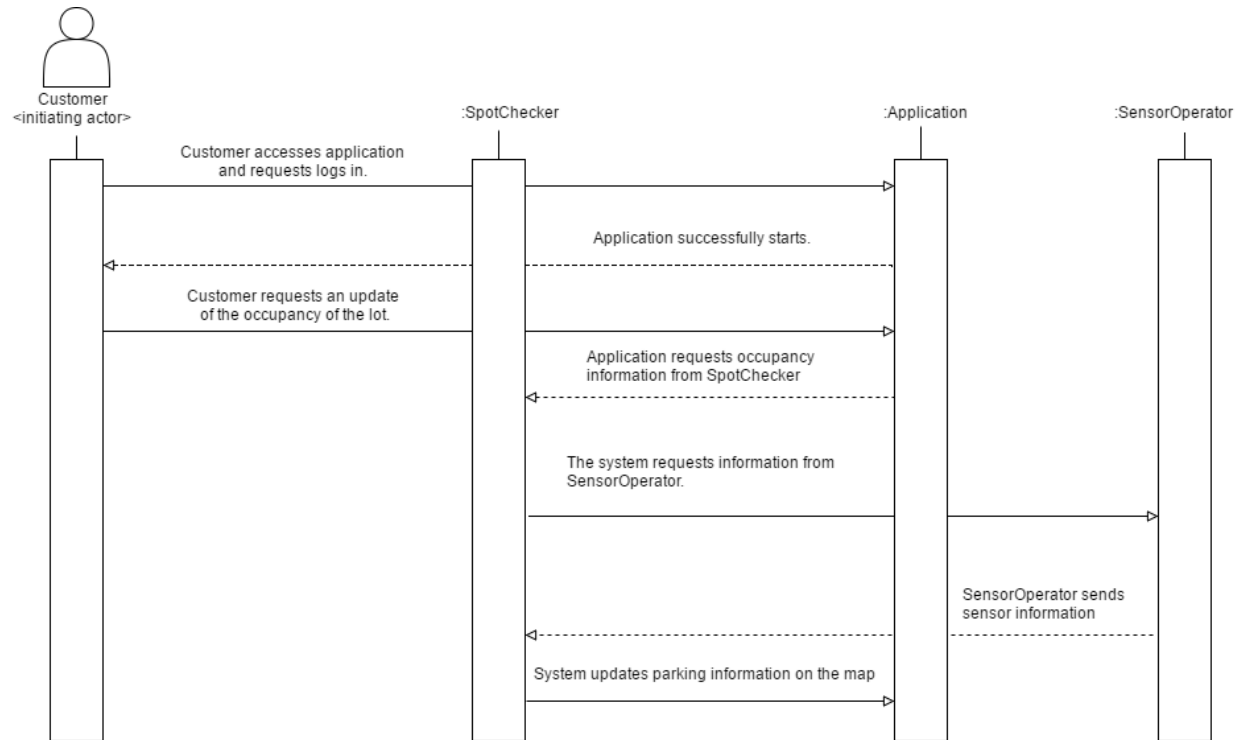
Use Case: UC-8



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.

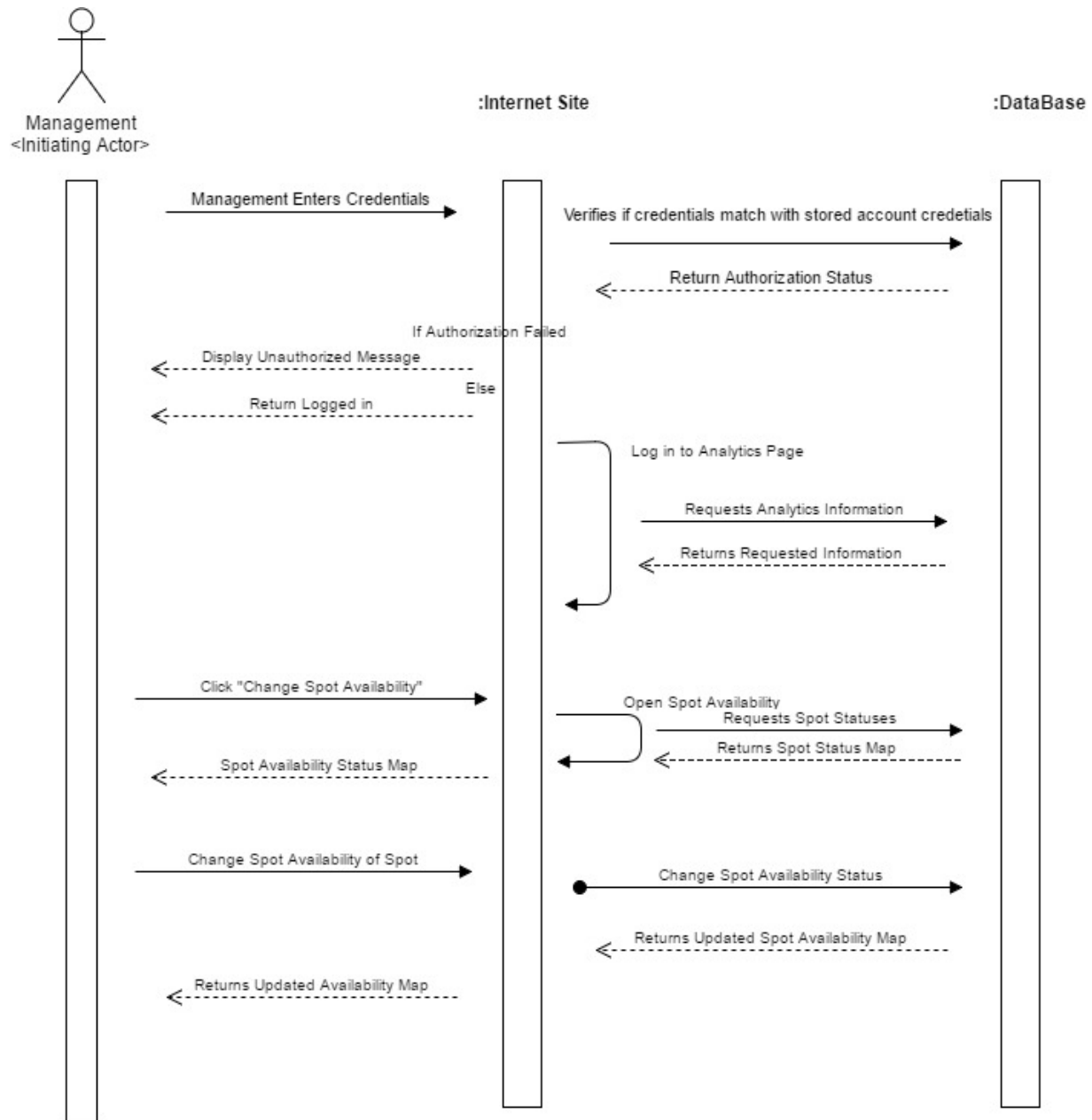
Use Case: UC-9



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.

Use Case: UC-13

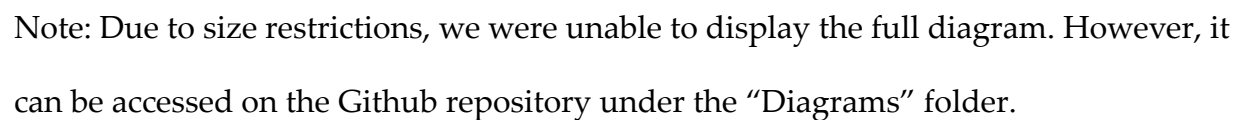


Design Principles:

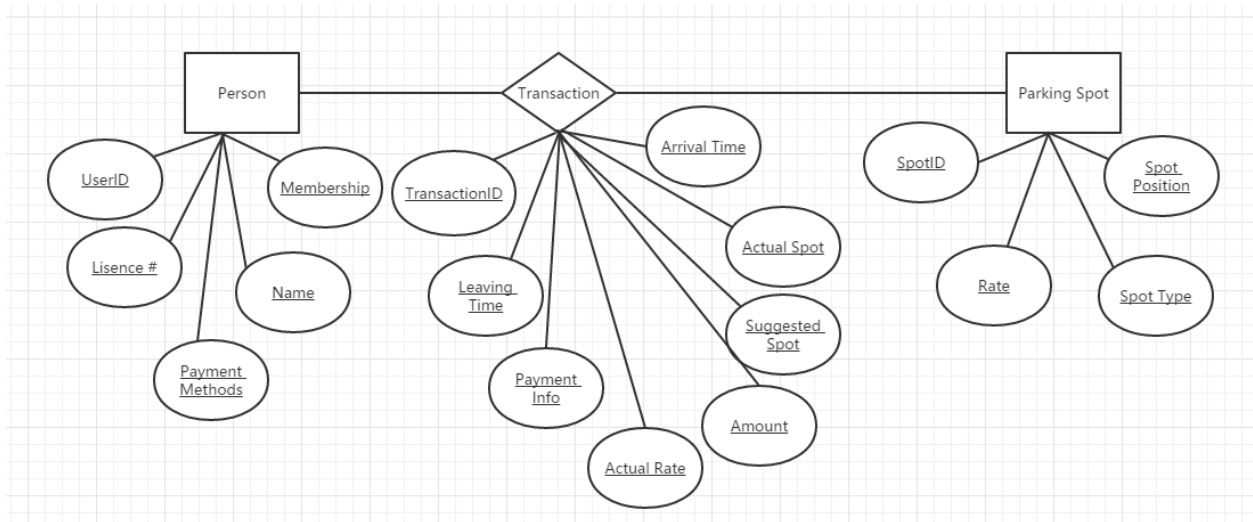
Expert Doer Principle: The management tells the internet site what it wants and the internet site relays the request to the database. The internet site is the first to learn of the request so it is the first to receive the message. This shortens the chain because it does this instead of going from management straight to the database without stopping at internet site.

High Cohesion Principle: The internet site isn't required to do any computations because it isn't necessary. The same goes for the management. They do very rudimentary actions.

Class Diagram



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 Simpson	6VBV76	General	MasterCard #...
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

Purpose: Manager wants to change price of parking

Classes:

ManagementAccount: Sets price of parking

Analytics

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.

System Architecture and System Design

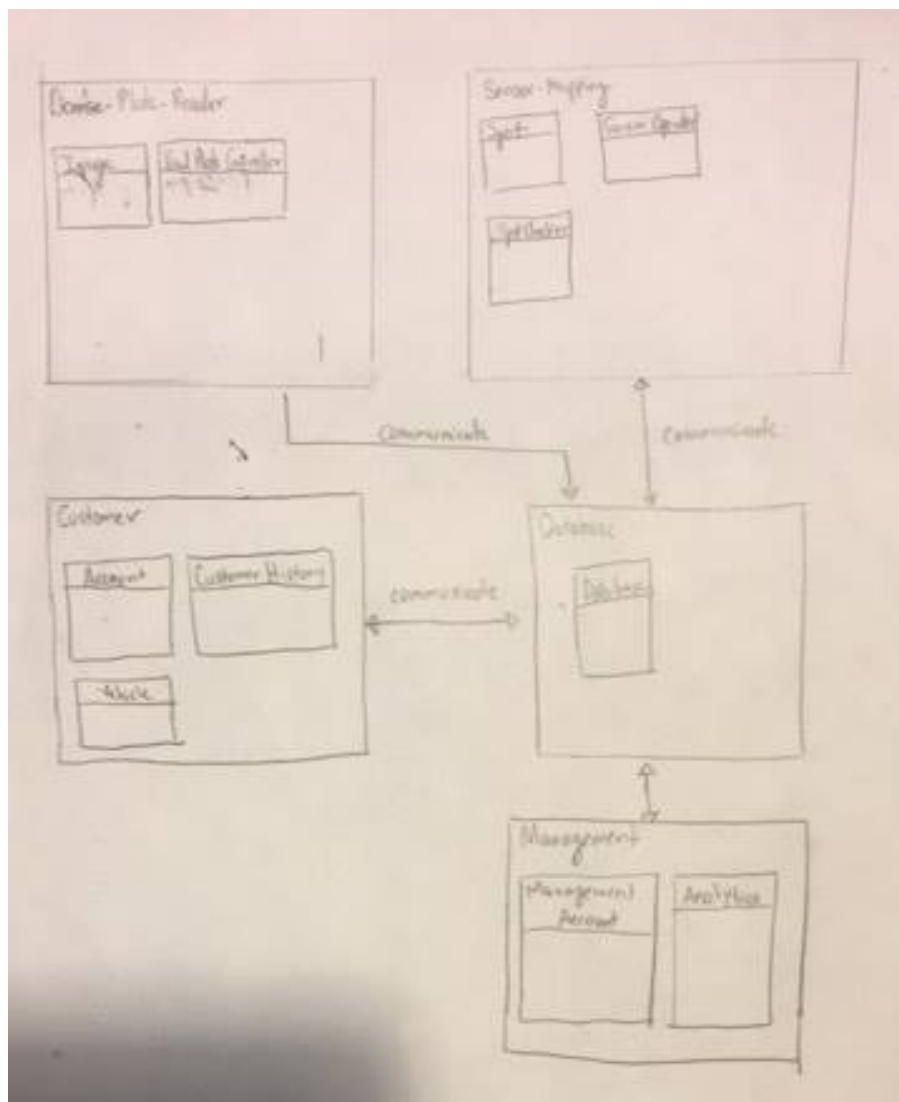
Architectural Styles

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



Since we are applying the domain architectural design pattern to design the system. We grouped classes into subsystems:

- License-Plate-Reader
 - Image
 - Read Plate Controller
- Sensor-Mapping
 - Spot
 - Sensor Operator
 - Spot Checker
- Customer
 - Account
 - Customer History
 - Vehicle
- Management
 - Management Account
 - Analytics

The license-plate-reader subsystem is in charge of reading the license plate and processing the license image into a string output to be stored into the system. The sensor-mapping is used to group similar classes which help with overall event-driven messaging system. The three components check, update, and maintain various data of the parking spot. Customer subsystem is used to hold, add, remove, and update customer information. It is the data that the client will use to provide things like recent transactions, updating credit card information, vehicle information, and more. The management subsystem is primarily used by the owner(s) to see the health of the business and update crucial information like cost of parking spot per hour.

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 Orderliness:

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 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

To figure out the patterns of customers' behavior, least-square regression can be implemented to study the curve of customer flow vs. time. A general linear model can be used if there is a linear relation. A relationship between factor x and y can be fitted by the following:

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i,$$

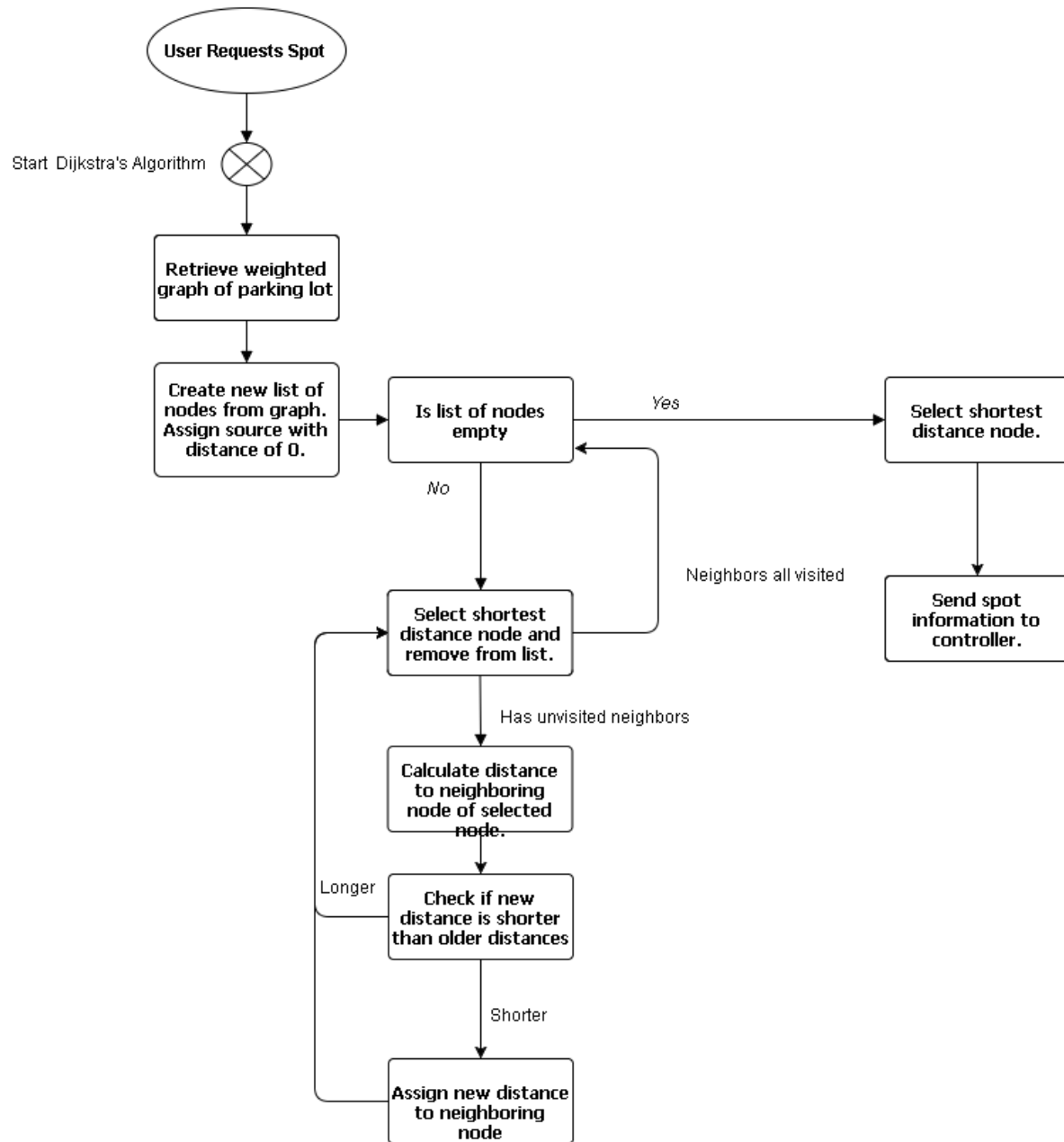
$$\varepsilon_i = y_i - \hat{\beta}_1 x_{i1} - \cdots - \hat{\beta}_p x_{ip}.$$

$$\sum_{i=1}^n \sum_{k=1}^p X_{ij} X_{ik} \hat{\beta}_k = \sum_{i=1}^n X_{ij} y_i, \quad j = 1, \dots, p.$$

Binary search is implemented for searching certain transaction in the payment history. If the payment ID or the time of transaction is known, since the payment IDs are listed from the earliest the latest, a direct binary search can be implemented without additional sorting required. If the exact transaction details are not known, a fuzzy query can be implemented by showing all the closest transactions that fit the input searching ranges of time, or usernames.

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. Below is an activity diagram used to displace 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.

To Log in (5):

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.

To Register (20):

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):

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

3. Tap yes in confirmation pop up

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

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

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

(Recommended Parking Spot): Alert/Notification for recommended parking spot

To Accept Notification (1):

1. Tap Ok

(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

Manager Application Interface:

No large changes to the UI have been made because it was already very simple and easy to use. Manager doesn't have to click any more times than necessary so we have a low user effort estimation. We only show information that is needed and it's displayed automatically on the pages. Just log in and preset settings for analytic display is shown. Manager just clicks if he/she wants to do anything else, like change the range or price.

(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
 - ◆ We will create a fictitious account representing a customer and use fictitious information and perform a check to see that the user information was stored in the database successfully.
- Customer logs in to account
 - ◆ To show that the customer is then able to login using the information provided at account creation and this test also verifies that information was successfully stored in the database.
- Customer adds a new payment method
 - ◆ We will add a payment method to the created account to demonstrate that the user is able to add payment methods and will be verified that it was stored in the database by a list of currently available payment methods.
- Customer adds a new vehicle
 - ◆ We will add a vehicle license plate number to demonstrate that a user is able to associate vehicles with his/her account. The updated list of vehicles associated with the account will show that it has been saved to the database.
- Customer removes an existing payment method
 - ◆ We will remove an existing payment method to show that payment methods are removed successfully. This test should also show that if there is only one payment method then the system will not allow deletion.
- Customer removes an existing vehicle
 - ◆ We will remove a vehicle from the account to check that the system is removing vehicles properly, the list of available vehicles will become shorter by the removed vehicle.

Testing the Customer Parking Lot Application:

Goal: The Customer must be able to use the Parking Lot Map and Recommended Spot successfully. Our parking lot turnstile system for entering and exiting should work integrated with our camera and sensors.

Test Cases:

- Customer receives notification for recommended parking spot
 - ◆ We will simulate a customer receiving a recommendation for a parking spot via a notification message. We'll send user a notification and do unit testing to see if our system reacts and marks the recommended spot as taken-in-transit.
- Customer views Parking Lot Map
 - ◆ We will have a fake parking lot map to demonstrate that the user can view a parking lot successfully. This test case should simulate the cases where the user opens and closes the map. It also tests for making sure you are viewing the correct parking lot map.
- Customer views Parking Lot information
 - ◆ We will assign data on each parking spot for the parking lot and test the sensors to see if our system sends state-change messages. This will show that the user will view the correct real time information.
- Customer searches for a Parking Lot
 - ◆ We will have list of fake choosable parking lots for this test. This should simulate the search bar for a list of parking lots. We will unit test to make sure system shows a suitable list of parking lots.
- System receives information from sensor state
 - ◆ We will have a test for every sensor to make sure they are reacting to state changes and sends the proper state to the system controller.
- System receives information from camera
 - ◆ We will have a test to see if the entrance and exit camera sends information to the system. We will simulate fake information sent to the system and checks if the information successfully sends.
- System processes information from camera
 - ◆ We will have a test to see if the system processes the information and executes the correct reaction for user input at the entrance turnstile (if registered user, if swiped the credit card, or if ticket user). It tests to see if the system executes the correct reaction for the user at the exit turnstile.

Testing the Management website:

Goal: The manager must be able to use the management dashboard to view the history of all transactions and see the report of profits and customer behaviors (i.e. high and low peaks of customers in certain periods of time during each day/week/month and occupancy of each spot). The website must allow the manager to manually change availabilities of spots as they want.

Test Cases:

- Manager logs into the account
 - ◆ To show that the manager is able to login using the default manager's admin username and password.
- Manager views payment history
 - ◆ We will perform plenty of testing transactions, and use manager's site to view all these transactions. This test verifies transactions are successfully stored in the database.
- Manager searches for one particular transaction
 - ◆ We will perform plenty of testing transactions, and perform a boundary test, that is, searching for the earliest and the latest transaction, using different input informations, including transaction ID, transaction date/time, username. This test too verifies integrity of transaction database, in addition to readability.
- Manager manually changes the availability of a spot
 - ◆ We will change availability of a randomly selected parking spot. The updated database will indicate the new status of that parking spot. This test verifies the revisability of the database.
- Manager view/search for all the user accounts
 - ◆ We will generate plenty of testing user accounts, and manager should be able to view all of them as a list. This test verifies the storage and readability of user accounts in the database.
- Manager generates a report about human traffic and profit
 - ◆ We will have plenty of testing user accounts and transactions generated, and use the manager's site to view the report of human traffic and profits. The profits should be consistent with the real money received and the human traffic report should clearly display the graphs made using the statistics stored in the database. This test verifies algorithm of building reports and the readability of the database.
- Manager deletes a user's payment method
 - ◆ We will generate testing user accounts and add testing payment methods to them. By deleting the user's' payment methods, those

methods will be permanently removed from the database and cannot be retrieved anymore.

→ Manager changes rate of certain spots

- ◆ We will store information of all the parking spots into the database including the rates. By clicking the button and input the desired rate of spots, the new rate will replace the former one but will not affect those reservations that are already did before it.

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

Since we are designing the user registrations, parking lot system, and manager dashboard isolated from each other, we will integrate our system at the end. The database has been agreed upon so integrating the database will be easy. We have to test whether or not each aspect of our system can effectively and successfully use the database to access and store information. Our manager dashboard needs the information from the parking lot system so we aim to test the dashboard to see if it will be getting the correct information from the database. Our parking lot system relies heavily on the user information in the database so we aim to test during every integration with the user system to see if our TurnstileController will react correctly with the User Database to important scenarios.

Project Management and Plan of Work

Progress to Date Summary (Report 2)

After the completion of Report 1 on February 19th, the team has since commenced to execute all the design specifications. The groundwork started when we all agreed to use Django framework with Python for our web-application, and HTML, CSS, and JavaScript to render interactive client-side webpages. We discovered that when each team added libraries, it was hard to communicate which dependencies had been added. The app would crash for the members who did not have all the correct dependencies. To rectify this, we started incorporating virtual environments and created a requirements.txt files to track the dependencies for the system. We have successfully implemented a system that takes an image of a car's license plate— at different angles — and renders a text file of the license plate number. Team 1 has created several modules to input/update user accounts in the database. They have been in discussion about creating a UX experience that minimizes the interaction with the app so that the driver does not become too distracted. Team 2 has figured out an efficient way of recommending spots to users in a way that helps the flow of traffic and reduces congestion in the parking lot. The team spent several days debating the best way to go about the solution, and through careful analysis and research, they have figured out an optimal solution. Team 2 has also started implementing internal tools to populate and display the occupied spots. Since teams 1 and 3 depend on team 2's data, teams 1 and 3 have started testing with dummy data that resembles the structure of the data team 2 is populating. Team 3 has implemented key algorithms used in determining manager-side analytics. They have looked at industry standards, as well as novel improvements, when considering the overall manager-side dashboard. On the project management side, we have moved away from the single TODO file and moved to weekly meeting notes, objectives, and deliverables. This is because often times we would forget the context of the TODO and then we would have to rework the content. By doing weekly files, everyone has a clear way of knowing what they have to and the particular points we talked about in the meeting.

Product Ownership

Teams:

1. Arthur Rafal and Guy Rubinstein
2. Kendric Postrero, Vatsal Pandya, and Gao Pan
3. Peter Luo, Yunqi Shen

Responsibility Breakdown:

- Team 1: User registration, turnstile implementation, user data verification, payment verification
- Team 2: Parking lot system implementation, parking space recommendation, parking lot testability, data collection for the parking lot
- Team 3: Manager access of setting price and enabling/disabling spots, Web portal dashboard for managers to view data, implement useful data analytics algorithms

Merging Contributions

Due to the large size of previous years' reports, the team decided to use the collaborative suite of tools offered by Google, primarily Google Drive and Google Docs. This way we were able to thwart spending man hours on compilation of everyone's work. People would simply add their content to specific sections and at the end, one person would format the titles, table of contents, and overall look and feel. We created a new copy of the report and edited that version for incremental changes. We ran into some problems when we first started making the diagrams because individuals would use different software, making the diagrams look inconsistent. We fixed this by having a designated website for creating wireframes and UML diagrams and each member was only allowed to use that one site.

Project Coordination and Progress Report

In the Gantt chart, from 2-18-17 to 2-26-17, the following changes were made:

- The "Create basic layout" deadline had to be pushed to 3-1-17
- The "Create basic layout" completion percentage changed from 20% to 50%
- The "Math" topic was completed on time

- The “Report 1 Part 1” had to be changed because “Class Diagram” were part of Report 2 Part 2
- We completed the “Interaction Diagrams”
- The “Plan of Work” history was changed to account for the progress and updated priorities

In the Gantt chart, from 2-26-17 to 3-5-17, the following progress/changes were made:

- The “Web Site Design” section was nearly finished but we still had to create a better mobile rendition and create better styling. We wanted to focus on functionality so we pushed the polishing the app further down the timeline.
- Each team had to do class diagrams and operation signatures. Each team submitted an XML file which was used to create one large UML file by Gao.
- We successfully assigned and divided up the responsibilities for Report 2 Part 2 subsection 3. This can be seen in the Gantt chart.
- Updated the “Software Testing” date from 3/17 to after the break

In the Gantt chart, from 3-5-17 to 3-12-17, the following progress/changes were made:

- Each team started designing tests for their modules, pushing the status from 0% completion to 35% completion
- The UI for internal tools had to be brainstormed and developed for Team 2
- Each team has met their deadlines since Report 1 (final)
- Prior to the break, the team met up to discuss their team’s progress and code integration that is set to take place during spring break

At this point, the basic layout for the website is completed. We have split off the work so that each sub-group has something they can show for Demo 1. Team 1 has the web-app running and they are checking whether data manipulation is successful with dummy data. Team 2 has finished implementing the algorithms involved for recommending spots. They will finish the mapping of the parking lot during the break. Team 3 is on track with creating modules that provide insightful metrics to the manager. They are looking to industry standards to find metrics and develop an interactive dashboard.

Plan of Work

The updated Gantt chart can be found here:

<https://app.smartsheet.com/b/publish?EQBCT=6e550bb75170435685b17b56898a4db5>

Breakdown of Responsibilities

As we progress, the bulk of the responsibilities and deadlines will be set by individual teams—in accordance to the overall timeline. For this reason, each team was tasked to further fine tune their timelines, objectives, and responsibilities, now that the design is clearer than ever. The dates, objectives, and responsibilities set by teams for Demo 1:

Team 1	<p>March 1</p> <ul style="list-style-type: none">• Create Log in Page with Register function - Arthur• Home Page with slide menu - Guy <p>March 7</p> <ul style="list-style-type: none">• Payment Page with Add payment method, remove card, and see details - Guy• History Page with see details - Arthur• Settings Page with remove vehicle, and edit personal details - Guy + Arthur <p>March 17</p> <ul style="list-style-type: none">• Fine tuning the app look and feel - Guy• Integrating the app with the database - Arthur• Testing the different Use Cases - Guy + Arthur
Team 2	<p>March 3</p> <ul style="list-style-type: none">• Creating the parkingLot class and establishing it's attributes – Vatsal, Gao, Kendric• Internal tool to map different parking lots - Gao• Implementing structure to track occupancy - Kendric• Implementing algorithm to determine recommended parking space - Vatsal, Gao, Kendric• Designing tests for the modules built to date - Vatsal, Gao, Kendric <p>March 8</p> <ul style="list-style-type: none">• Implementing data collection - Vatsal• Running tests - Vatsal, Gao, Kendric• Designing simulations - Vatsal, Gao, Kendric <p>March 17</p> <ul style="list-style-type: none">• Make the brochure for Demo 1 - Vatsal
Team 3	<p>March 8</p> <ul style="list-style-type: none">• Create Log-in Page that leads to analytics page, if correct - Peter

	<ul style="list-style-type: none"> • Analytics page that shows the stored information - Yunqi • Able to change Parking lot price - Peter • Calendar with info on dates - Yunqi • Page for Customer History - Peter <p>March 17</p> <ul style="list-style-type: none"> • Able to search for Customer History on the 3rd page by license plate or date - Yunqi+Peter • Able to change spot status - Peter • Able to choose timeframe of profit graph - Peter • Real time feed of customers - Yunqi • Sign out function - Peter
--	--

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. 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.
10. 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.
11. 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.
12. 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.