

RUParking - Parking Garage Automation

Group No. 9

Team Members

Name	Email
Kevin Wong	wongwj@eden.rutgers.edu
Ikshat Gandhi	igandhi@eden.rutgers.edu
Prit Patel	pritp27@eden.rutgers.edu
Darryl Nable	dnable@eden.rutgers.edu
Luke Xu	lukexu@eden.rutgers.edu
Parth Desai	padesai@eden.rutgers.edu

Instructor: Prof. Ivan Marsic

Project URL: <https://sites.google.com/site/ruparking007/>

Revision History:

Version No.	Date of Revision
v.1.0	May 5, 2013
v.2.0	May 12, 2013

Effort Breakdown

Report Section	Contributors
Summary of Changes	Kevin
1.Customer Statement of Req	Equal
2.Glossary	Equal
3.System Req	Equal
4.Functional Req	Equal
5.Effort Estimation	Ikshat
6.Domain Analysis	Equal
7.Interaction Diagrams	Equal
Design Patterns	Kevin
8.Class Diag/Interface	Equal
Design Patterns	Ikshat
OCL Contract Spec	Kevin
9.System Arch / Design	Equal
10.Algorithms	Equal
11.UI Design/Implementation	Equal
12.Design of Tests	Equal
13.History of Work, etc	Ikshat, Kevin
14.References	Equal
Project Management	Equal

Summary of Changes

Summary of project changes from report 1&2:

- Added Management module, implemented by Java GUI.
- Allowing reservation cancels
- Removing spot sensor usage
- Reimagining walk-in zone functionality
- Implementing exit module with small Java GUI not requiring user interaction

Modifications for Report 3:

- Revised Proposed Solution to reflect all changes
- Removed occupancy photo-sensors from Devices section
- Removed assumption regarding spot sensors, and added assumption A7
- Revised business policies P2, P3, P6, P7, P8, and P11.
- Added new glossary terms; removed unnecessary ones
- Modified FURPS table/On-Screen Appearance requirements to reflect project changes
- Added new actor - Exit monitor
- Modified UC2, UC3, UC5, UC7, UC9, UC10, UC11 to reflect project changes
- Adjusted Use Case Diagram & Traceability matrix accordingly
- Fully-dressed descriptions have been modified to reflect changes
- Sequence Diagrams with Manager Module have been changed from website to GUI.
- User Effort Estimation added
- Domain Concept/Attribute Descriptions have been modified to reflect the current state of the project vision.
- Class diagram updated: Garage concept removed
- Traceability Description has been added.
- System Operation Contracts have been modified accordingly.
- Added design patterns section to Interaction Diagrams
- Added design patterns section to Class Diagrams
- Modified Acceptance Test Case 9 to reflect changes to exit module.
- Added History of Work, Summary of Changes

Table of Contents

1. Customer Statement of Requirements	1
2. Glossary of Terms	7
3. System Requirements.....	9
4. Functional Requirements Specification.....	15
5. User Effort Estimation	38
6. Domain Analysis	42
7. Interaction Diagrams	59
8. Class Diagram and Interface Specification	77
9. System Architecture and System Design.....	87
10. Algorithms and Data Structures.....	94
11. User Interface Design and Implementation.....	96
12. Design of Tests	105
13. History of Work, Current Status, and Future Work.....	116
14. References.....	118

1. Customer Statement of Requirements

Goals

The goal of this project is to design an automated garage system that is efficient and user friendly. The system will manage garage occupancy and include a feature that allows customers to reserve parking spots remotely ahead of time. Furthermore, it will generate statistical data to be viewed by the management, allowing them to make better business decisions.

Problem Statement

The parking garage currently operates without any automated system. The garage employees manage occupancy by walking around to check the status of parking spots. This not only causes congestion in the garage, but also reduces the profit. The current system is inefficient because there is no quick way of checking for vacant spots, which in effect discourages customers from using the garage. Bookkeeping is also done manually, which limits the number of diagnostics that can be obtained about garage usage. The management has requested for these inefficiencies to be replaced by an automated system that will increase profits and level of user simplicity.

Proposed Solution

In order to fix the inefficiencies mentioned in the problem statement, the management has requested for a system that would increase profits and occupancy of the garage.

The garage parking automation will have two separate parking zones, a reserved parking zone and a walk-in parking zone. The first floor will be dedicated as a walk-in zone. These two zones will be treated separately.

The solution to improve the current system will include a website exclusively for management to view statistical data of the parking garage and a mobile application for customers to make reservations. Customers looking to park in the reserve zone will be required to make a reservation through the mobile application. When making a reservation, the customer may provide a license plate number. Customers must create accounts through the mobile app in order to make all reservations.

The walk-in zone will be on the first floor and will be accessible through a separate entrance. Customers willing to park in the walk-in zone will be informed at the entrance whether or not there are vacant spots. Customers with a reserved spot will be required to use the garage elevator to reach the correct floor. This elevator will not move for customers without a reservation and will prompt them to park in the walk-in zone. This elevator will fix the problem of congestion that occurs with the current garage system by creating a one way entrance to the reservation lot. Customers will exit the garage through a ramp and their license plate number will be used to note their exit time. Since there is a one way elevator for entrance and a one way ramp for exit, there will be reduced congestion with a minimal chance of accidents.

This new system will not rely on employees to manually check for vacant parking spots. In place of the employees, there will be a camera based system that will keep track of vehicles as they enter and exit the reserved lot. Because parking spots are assigned to the customers, the camera alone is sufficient to accurately keep track of garage occupancy. Spot sensors on each parking spot will not be implemented.

If a registered customer forgets to make a reservation and wants to use the parking garage then they can use the walk-in lot if space is available. The walk-in space availability will be displayed at the entrance of the garage and will also be available through the mobile application. Customers who do not have a reservation and decide to park at the walk-in are known as walk-in customers.

If the elevator terminal cannot recognize the license plate, then the customer will be prompted to enter the reference number to see if a reservation exists. This system allows the customer to bring any vehicle they wish. If the license plate read by the system does not match the license plate in the database and the customer has a reservation, then the system will be updated with the new license plate. This will allow for proper billing and statistics.

When a customer makes a reservation, the parking spot will be held for the duration of the originally reserved time. This means that the customer will be able to park their vehicle in the reserved parking zone as long as their reservation time has not expired. If the customer does not show up for their reservation time, they will still be charged for their full reservation time. Any customer who fails to depart from the parking garage within the limits of their requested reservation time will be charged at higher rate for the time of overstay. If a customer with a reservation arrives and their spot is taken by an overstaying customer, the elevator terminal will check for another unreserved parking spot and if available will assign them that spot. If no other spot is available, then the customer will be offered one of the five emergency parking spots if available. If no spots are available, the customer will be fully refunded their money and will be directed to the walk-in zone if spots are available.

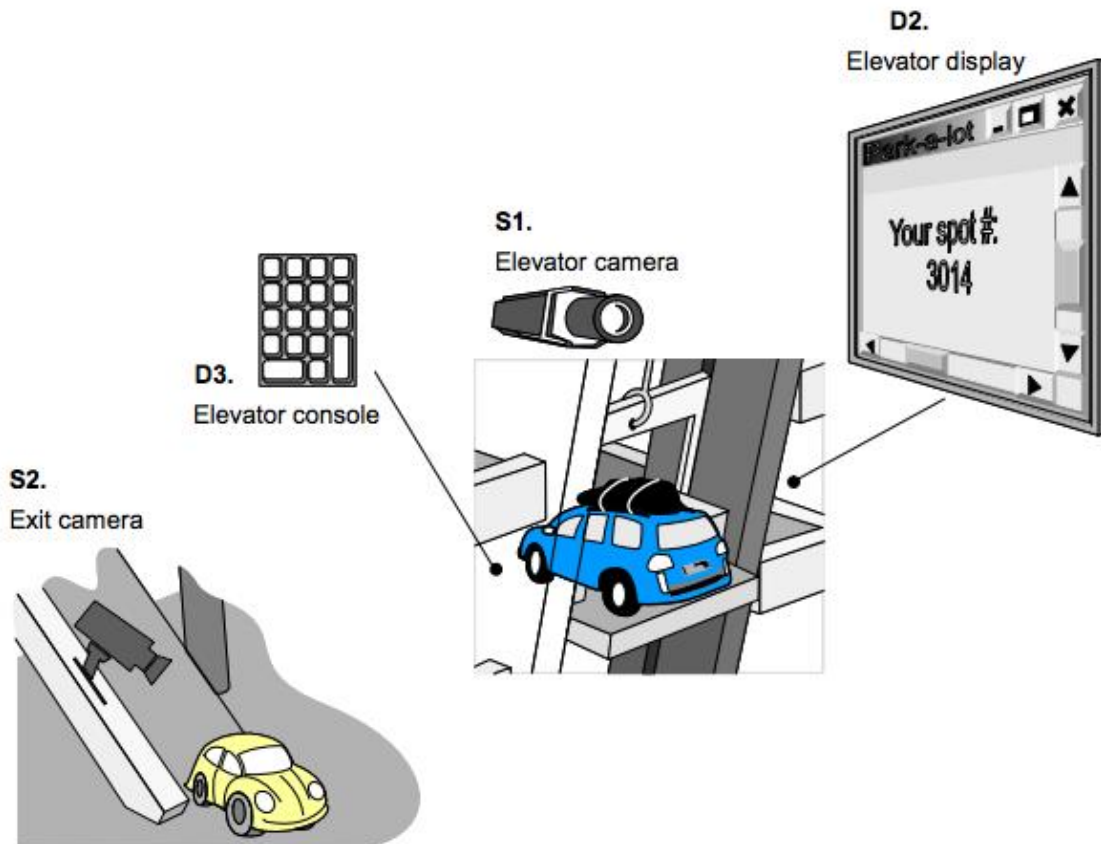
Within the parking garage, there are many machines working simultaneously to create an efficient and user-friendly system for customers to come in, park, and leave. With these machines, mistakes and failures can be made. One of the most important machines within the system is the license plate reading technology. This camera will read the customers license plate for the system to then confirm. The camera will be designed such that it will be able to read all license plates regardless of its condition. If the system is unable to recognize the license plate read by the camera, the system will require the customer to input their reference number into the console found inside the elevator. In the case that the inputted reference number is also not recognized by the software, then the system will not allow the customer to enter any of the reservation zones. All the machines working within the parking garage are expected to be working 100% of the time.

One of the major problems that occurred with the old parking garage is that customers entered the garage looking for a parking spot not knowing whether or not a vacant one was available. This system created congestion within the garage and lacked efficiency. With this new parking garage system, the customer will always know if a parking spot is available whether it is in the walk-in zone or the reservation zone. At the entrance of the walk-in zone, there will be a display that will let customers know how many parking spots are vacant for that particular zone. For the reservation zones, customers will be given a parking spot when their license plate or reference number is recognized by the system. The elevator will lift the customer with their vehicle to the appropriate reservation floor and never stop at an incorrect floor. This prevents the customer from searching for a parking spot on the wrong floor. This will also ensure that the customer will not park in a spot that was not assigned to them.

The management interface will provide garage operators with statistical data and the ability to modify business policies within the system. The goal is to design a user friendly interface to allow management easy access to the modification of the business policies. The homepage will take the user to a login page and can only be accessed by management who has been given a unique account name and password. As changes are made, the database will be adjusted through this GUI and it will be reflected in the customer's mobile application.

Devices

- S1: Elevator Camera
- S2: Exit Camera
- D2: Elevator Display
- D3: Elevator Console



Assumptions

- A1.** Customers will enter and exit the parking garage for each individual reservation.
- A2.** The license plate recognition system will work 100% of the time with no errors.
- A3.** Customers will leave elevator if license plate/reference number are not recognized.
- A4.** The vehicle elevator will work 100% of the time in lifting each customer and vehicle to the appropriate parking deck.
- A5.** Customers will always park in their assigned spots and not in any other vacant spots.
- A6.** Customers have access to a smartphone because reservations and extensions will be granted through the mobile application.
- A7.** Each car entering the elevator will have a physical license plate attached in a proper fashion on the car.

Policies

- P1.** The customer will be assigned only a reference number upon making the reservation. The customer will only be informed of his/her spot upon arrival.
- P2.** If the license-plate recognition system does not recognize the license plate number as associated with a registered customer, the platform will remain motionless and the display screen will notify the driver that the license plate number is not recognized, and will request them to enter their reservation confirmation number on the console installed in the elevator. If the customer does not have a reference number or enters the wrong one they will be asked to back away from the platform and park in the walk-in zone.
- P3.** Any customer, regardless of registration, without a reservation may park in the walk-in zone as long as there are vacant spots. These customers are not allowed to park in the reserved zone.
- P4.** The reserved parking spot will be held for the customer for the full allotted time independent of vehicle presence. If a customer departs before their reserved period expires ("understay"), he or she will be allowed to return to their spot at any time before their reservation expires.
- P5.** If the customer does not arrive during their reserved interval, he or she will be billed for the entire duration of their reserved interval. No refunds will be made.

P6. The customer is allowed to extend an existing reservation before parking if there is an available spot for the modified parking duration. The customer can extend or cancel their reservation before x amount of time in advance, where x is a modifiable time controlled by the management.

P7. If a customer fails to depart as scheduled after their reserved period expires, he or she will be billed for the duration of their reserved period at a regular rate and at a higher rate for the duration of their overstay. The rate will increase progressively with the duration of overstay. The customer will be notified of their overstay on the exit screen.

P8. Each customer is allowed to have multiple standing reservations under his or her name. Such reservations may be back-to-back, but not overlapping in time. For back-to-back reservations, customers will not be guaranteed the same spot; the customer must re-enter the garage (**A1**).

P9. If a customer arrives and his or her reserved spot is still occupied by a previous customer who failed to depart as scheduled, and if there are other available spots, then the arriving customer will be offered to park on another available spot.

P10. There will be 5 emergency parking spots available in the case that a customer arrives and his or her reserved spot is still occupied by a previous customer who failed to depart as scheduled, but there aren't other available spots. This customer will be offered to park on one of the 5 emergency parking spots. The message will be displayed on the display inside the vehicle elevator.

P11. If all 5 of the emergency spots are occupied, the customer will be offered a full refund. Further the customer may park in the walk-in if a spot is available.

P12. The parking garage system will not overbook the parking lot. That is, it will never assign spots that cannot be guaranteed to the customer.

2. Glossary of Terms

Application – A mobile application where customers can access the system to make a reservation.

Camera – A device for recording or reading visual images in the form of photographs and video, used to read license plates and send information to garage system.

Confirmed Reservation – A paid reservation placed by a registered user. A parking spot is guaranteed.

Customer – A person who wishes to use the garage's services.

- **Registered Customer** – A customer who has registered an account on the garage's mobile application prior to showing up to the garage.
- **Unregistered Customer** - A customer who does not have a registered account in the systems database.

Database – Entity that stores all the system's information.

Elevator – A platform used to raise vehicle to different floors.

Elevator Terminal – A console or screen inside the elevator where the customer can enter in necessary information.

License-Plate Recognition Software – A camera based system that reads the license plates of vehicles and checks the information against the database.

No-Show – A customer who does not show up for their reservation.

Overstay – When a customer doesn't leave the garage at the end of his reservation period.

Reference Number - A number that is given to the customer as confirmation of their requested reservation.

Refund - Returning money paid for reservation when reserved spot is occupied by an overstaying customer and emergency spots are occupied.

Reservation – The act of reserving a parking spot via the system's mobile application.

Sensors – A device that is placed on the floor of every parking spot that detects if that spot is occupied or vacant.

Understay – The act of leaving a parking spot before the reservation period is over.

Vehicle – A mobile machine used for transportation

Walk-In – When a customer requests an immediate parking spot without prior reservation.

Management Interface – An interface that the management can use to view statistical data and change parking rates.

3. System Requirements

User Stories

Actors: Garage Operator

Customer: Registered Customer

Customer: Unregistered Customer

Identifier	Size	User Story
ST-1	2	As a customer, I can check the number of vacant spots in the walk-in zone in person.
ST-2	4	As a customer, I can check the number of vacant spots in the walk-in zone via mobile application.
ST-3	1	As a customer, I can park in the walk-in zone.
ST-4	4	As an unregistered customer, I can register for an account using the mobile application.
ST-5	7	As a registered customer, I can reserve a parking spot for any time in the next seven days.
ST-6	1	As a registered customer, I can make multiple reservations using the same account on the mobile application.
ST-7	5	As a registered customer with a confirmed reservation, I can extend my reservation using the mobile application.
ST-8	4	As a registered customer with a reservation, I can cancel my reservation with accordance to the cancellation policy.
ST-9	6	As a garage operator, I can view statistical data about my garage.
ST-10	2	As a garage operator, I can modify the cancellation policy. I can set the required advance notification time for cancellations. I can also disallow cancellations altogether.
ST-11	3	As a garage operator, I can change the pricing policy of the garage at any time. This includes cancellation policy and parking rates.
ST- 12	2	As a user, I can overstay as long as I'm willing to pay extra.

Enumerated Nonfunctional Requirements

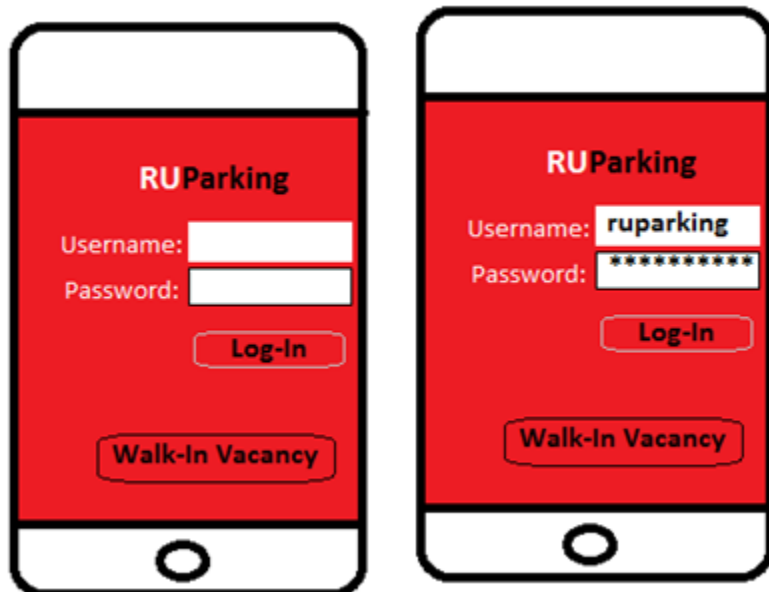
Identifier	Size	Description
REQ-1	5	Only authorized users will be allowed to access and edit a customer's profile. Customers should not be able to alter their amount owed in any way other than making a payment.
REQ-2	2	Only a valid credit card number will be accepted.
REQ-3	4	All user interfaces should be simple enough to allow registration and reservation to be completed in at most 4 minutes.
REQ-4	1	The elevator must bring the customer to the correct floor.


FURPS Table

Functionality	<ul style="list-style-type: none">• Includes a camera-based system that detects and reads the licence plate numbers of the vehicle entering or exiting the garage.• Contains a terminal-based system that allows the customer to enter in their reference number.• Allows users to check whether there are walk-in spots available.
Usability	<ul style="list-style-type: none">• There is consistency as all the app pages follow the same template.• There are proper instructions and guidelines that make it easy for the users to explore the app.• To accomplish simplicity, each page has multiple buttons to access the individual pages.
Reliability	<ul style="list-style-type: none">• System is designed to be fail-safe to minimize the chances of failure.• System uses cameras to check if customers have parked properly.• Emergency parking spots will be available if a customer with parking reservation is not able to park due to an overstay.• Database is stored to maintain record of all customers, their reservations, and the availability of all parking spots.
Performance	<ul style="list-style-type: none">• In order to be efficient, parking spots for the reservation zone will be assigned when reservations are made through the mobile app. The customer will not be notified of their spot until their arrival.• Throughput is increased as a result of the online reservation process.• By eliminating this step from the procedure done at the garage, it decreases the chance of traffic congestion due to payment transactions.

Supportability	<ul style="list-style-type: none"> • This design is very adaptable as far as the garage size is concerned. • If there are ever future plans to expand the garage, additional floors or any other renovation, the adjustment would be as simple as making an update in the database table. • Maintainability is also fairly straightforward as there isn't too much customer information to deal with.
----------------	--

On-Screen Appearance Requirements

REQ No.	Priority weight	Description
REQ-5	4	<p>[Mobile Application] Users will be required to register for an account in order to reserve a spot. The user will be asked to input their name, email address, billing address, credit card information and phone number.</p> 

		
REQ-6	3	[Mobile Application] The user will be required to make payment when making a reservation. They will be directed to a payment page where they will confirm the transaction.
REQ-7	4	[Mobile Application] To make a reservation, the user would need to log-in into their account on the app via their mobile phone.
REQ-8	4	[Mobile Application] If users want to cancel, modify, extend their reservation, they need to log into their account and choose whether they want to cancel or just extend it; if they want to modify/extend, they need to add the additional information: time, day, location, parking period, the number plate of their car, etc.
REQ-9	2	[Mobile Application] In order to edit the information of their parking account, the user needs to log on to the app for the garage, and click on an edit account button. After that, there should be several fields of information, the user should be able edit any field they want (including address, city, state, zip code, credit card number, new password, old password, etc.) then click on submit. The user should not be able to edit the email, because it is used to identify the user.
		[Elevator] The camera will read the license plate of the car in the elevator. The driver will be allowed to enter their reference number in case the license plate isn't recognized.

REQ-10	3	<div><h1>Welcome to RUParking</h1><p>Please wait while our camera detects your license plate...</p></div> <div><h1>License Plate Recognized</h1><p>Please remain in your vehicle as the elevator will lift you to the correct floor. Your parking assignment is: (parking space number here)</p></div> <div><h1>License Plate not recognized...</h1><p>Please input Reference Number:</p><div><div><div>1</div><div>2</div><div>3</div><div>A</div><div>4</div><div>5</div><div>6</div><div>B</div><div>7</div><div>8</div><div>9</div><div>C</div><div>*</div><div>0</div><div>#</div><div>D</div></div><div><div></div><div>Enter</div></div></div></div>
REQ-11	3	[Management Interface] Garage operators will be able to log into a management interface in order to modify parking rates, cancellation policies and to view statistical models.

		<div><h3>Parking Garage Admin Access</h3><p>Username <input type="text"/></p><p>Password <input type="password"/></p><p><input type="button" value="Login"/></p></div> <div><h3>Policy Modifications</h3><p><u>PARKING RATES</u></p><p>Cost per hour for reserved zone</p><p>\$ <input type="text"/></p><p>Cost per hour for walk-in zone</p><p>\$ <input type="text"/></p><p><u>CANCELLATION POLICY</u></p><p>Time given to customer for cancellation</p><p><input type="text"/></p><p><input type="button" value="Save Changes"/></p></div>
--	--	---

4. Functional Requirements Specification

Stakeholders

- 1) Customer - Owner of the garage
- 2) Construction architect
- 3) Maintenance officials
- 4) Security guard/operator
- 5) Database architect
- 6) Web-page designers
- 7) Developers
- 8) Users - Registered, Unregistered
- 9) Project Manager
- 10) Business Analyst
- 11) Third party bill payment services

Some of the stakeholders such as third party bill payment services are not involved in the current implementation of the project but will be of interest when the system is modified to include some of the suggested developments.

Actors and goals

Initiating

Garage Operator

- To set prices and rates through the Management Module.
- To view statistical data through the Management Module.
- Change cancellation policy through the Management Module.

Unregistered User

- Register an account with valid credit card information.
- Use the mobile app to view occupancy of walk-in zone.
- Park in the walk-in zone.

Registered User

- Enter registration details if the license plate is not recognized.
- To make multiple reservations in advance.
- Extend/cancel reservations.
- Use the mobile app to view occupancy of walk-in zone.
- Park in the walk-in zone.

Participating

Database:

- To maintain a record of parking spots and their availability.
- To manage registrations and reservations.
- To maintain history of customer overstay and usage of the parking lot.

Elevator System:

- To enable interaction between the users and the system by allowing the user to input reference number if license plate is not recognized.
- To direct the user to the assigned spot.
- To direct the user to the walk-in zone if the user doesn't have a reservation.
- To communicate with the database to validate customer license/reference numbers.

Elevator:

- To lift a vehicle to the corresponding floor of the assigned spot.
- To allow vehicles to access the reserved-zone.

Entrance Gate:

- Opens upon receiving an entry ticket and allows to decrement counter used to keep track of walk-in zone vacancy.

Exit Gate:

- Opens upon paying at the payment terminal and allows to increment counter used to keep track of walk-in zone vacancy.

License plate reader 1:

- To read the license plate number of a vehicle once it enters the elevator upon entrance.

License plate reader 2:

- To read the license plate number of the vehicle as it exits the garage.

Exit Monitor:

- To display an exit message as the user exits the garage.
- To display a message regarding overstay if user has overstayed past reservation time.

Mobile App:

- To allow customers to register with the system
- To allow customers to reserve multiple parking spots.
- To allow customers to view walk-in occupancy.
- To allow customers to extend/cancel reservations.

Operator:

- To process parking fare from walk-in customers upon exit.
- To attend to any customer issues at the garage.
- To notify the maintenance officials about any device malfunctions or problems.

Payment terminal:

- To process payments of unregistered customers at exit.

Ticketing terminal:

- To receive entry ticket for walk-in zone.

Vacancy Monitor:

- To display number of vacant spots of the walk-in zone.

Management Interface:

- To allow garage operator to view statistical data
- To allow garage operator to change cancellation policy
- To allow garage operator to set rates.

Use Cases

The following is the list of the use cases for the system and their description.

**All use cases will be implemented for our final demo.*

Casual Description:

UC-1 Registration:

In order to register with the system the user has to use the mobile app and create a user account. The user will be asked for some basic necessary details such as first name, last name, mailing address, phone number, credit/debit card number, email and password. When the user accepts the terms and conditions, an account will be established and maintained. The user can login with the email and password at any point to make or cancel reservations and make changes to the account. Only one account will be allowed per email address.

UC-2 Reservation:

Once the user has registered he/she can use the username and password to login to make reservations. Unregistered customers cannot make reservations. The user will be allowed to make reservations for only the next seven days. Upon selection of regular reservation the user can pick the desired date, start time and duration that he/she wants to reserve the spot. The app will check if there is an available spot for that selected time period and if there is a spot available the user will be notified of the total amount for the reservation based on the rates set by the garage operator. Once the user confirms the reservation he/she will be charged the total amount at this point and will be given a reference number.

Alternate Scenario:

If there are no spots available during the requested time period, the user will be advised to try again later in case their requested time period becomes available due to cancellations.

UC-3 Elevator Entry:

Once the car enters the elevator, license plate reader 1 will be activated and the license plate of the car will be read. If the number is recognized within the database as belonging to the reserved user, the software will fetch the spot number from the database. The user will be notified of the assigned spot through the elevator screen. The elevator will then move to the appropriate floor. Once the elevator reaches the appropriate floor the user will park at their assigned spot and the database will be updated to show occupancy of the current spot.

Alternate Scenario 1:

If the license plate is not recognized within the database then the interface will prompt the user to enter the reference number. The system will then check to see if the entered reference number is valid. Once validated, the system will then update the license plate on the database with the license plate that was read. Now, the software will fetch the spot number from the database. The user will be notified of the assigned spot through the elevator screen. The elevator will then move to the appropriate floor. Once the elevator reaches the appropriate floor the user will park at their assigned spot, and the database will be updated to show occupancy of the current spot.

Alternate Scenario 2:

If the parking spot that was assigned to a customer is occupied by an overstaying customer then the system will check the database to see if any other spot is available. If available then this spot will be displayed on the elevator screen.

Alternate Scenario 3:

If another spot is not available in the reservation zone then the system will check if one of the five emergency spots is vacant. If vacant, this spot will be assigned and the customer will be notified through the elevator screen.

Alternate Scenario 4:

If the emergency spot is not available then the system will notify the customer that their reservation cannot be fulfilled and will display the number of parking spots available in the walk-in zone. The customer will also be refunded the entire amount that was charged for the reservation that couldn't be fulfilled.

Alternate Scenario 5:

If the reference number is not recognized and the license plate is also not recognized then, the customer will be notified through the elevator screen that they do not have a

reservation. The user will be notified to exit the elevator entry and to use the walk-in zone.

UC-4 Walk-In

If a registered/unregistered user would like to park in the walk-in zone, he/she must enter through the walk-in zone entrance. At this entrance, there will be a display that will inform the user of the rate for parking in the walk-in zone and it will also display how many vacant spots are available in the walk-in zone. When the user enters the walk-in zone, he/she will be given a ticket that will have the date and time of entry. This ticket will be used to determine the total price of parking when exiting the parking garage. In the case that there are no vacant spots available within the walk-in zone, the gate will be locked and will prevent users from entering.

UC-5 Reservation Exit

All cars must leave the reservation zone through the exit ramp. As the user leaves the garage, the license plate reader 2 located at the exit ramp will read the license plate number and send the information to the database to check the user out of the garage. There will not be a gate located at the exit of the reservation zone since the camera will detect when a user leaves and if an overstay has occurred. There will be an exit monitor to display messages to the user as they exit the garage. If an overstay is detected, the monitor will display a message informing the user of this extra payment.

UC-6 Walk-in zone Exit

The user must exit the walk-in zone through the walk-in zone exit which consists of a payment terminal and gate. At this payment terminal, the user must give his/her ticket to the operator and pay for their parking time. When this transaction is complete, the gate will be opened and the user will be able to exit the parking garage. The counter in the database will be incremented for the number of vacant spots available in the walk-in zone.

UC-7 Reservation Management:

If a user wants to extend any previously made reservations he can do so by clicking the "Manage Reservations" button in the Mobile App. The program will check to see if it is before or during the reservation period. If it is before the reservation period then the program will look for any available parking spot; however, if it is during the reservation period, then the program will only look to see if the current parking spot that the user is in is available for the extension. If the user is successfully granted a reservation, then he/she will be billed for the extra duration.

Alternate Scenario 1:

The user requests an extension for a specific time period and the database checks if the modified reservation period is available. The database returns that there are no spots available for the requested time the user will be notified.

Alternate Scenario 2:

The user can also cancel a reservation. In this case the user clicks on “Cancel Reservation”. Once the user confirms that he/she wants to cancel reservation the reservation will be cancelled within the database. The user will also be given the full refund for the reservation cancellation.

UC-8 Mobile Payment:

Once the user selects a valid reservation period the total will be displayed. The user will be billed as soon as he/she confirms the payment and will be charged to his default credit card number. After completion, the user will be given a reference number.

UC-9 Database Read:

If a registered or unregistered user wants to check the vacancy of the walk-in zone they can use the mobile application. Once they open the mobile app, the user can click on the “Walk-in Vacancy” button. Once they click this button, the data will be fetched from the database and will be displayed on the screen of the mobile app.

Alternate Scenario:

If the garage operator wants to check statistical data, they can do so using the manager interface. The operator must log in with a username and password. Then, the operator must click on the “View Statistics” button. Once clicked, the data will be fetched from the database and will be displayed screen. The garage operator can log out once done viewing the data.

UC-10 Database Write:

When a registered customers uses the mobile application, he/she can reserve a new spot or make changes to an existing reservation. The database will be modified accordingly.

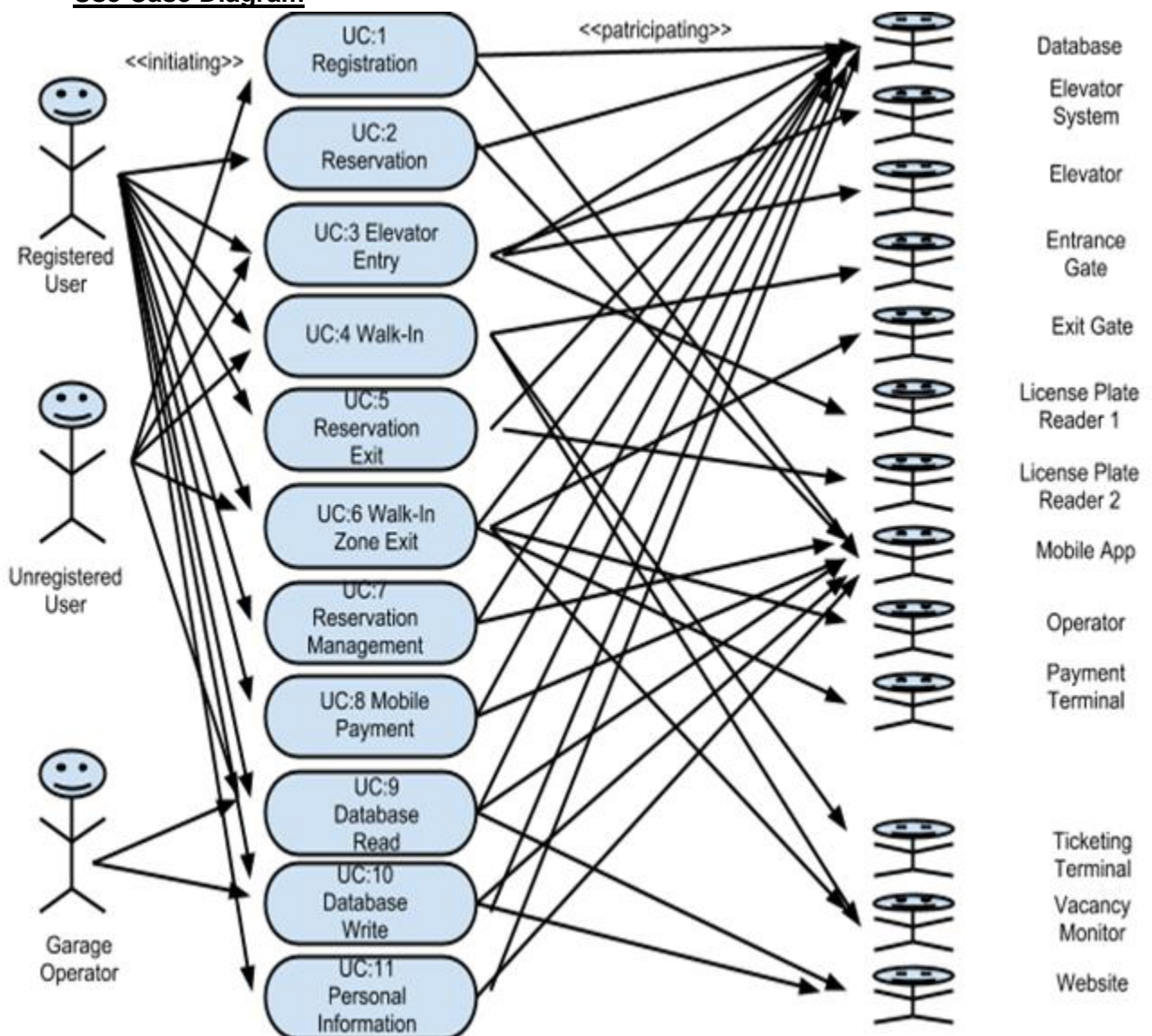
Alternate Scenario:

When the manager logs into the interface, he/she can modify the parking rates for reserved and walk-in zones. The operator can also change the cancellation policy from within the GUI. These actions will reflect changes in the database.

UC-11 Personal Information:

If a registered user wants to edit the details provided during registration, he can do so by logging into the mobile app and selecting the “Manage Profile” option under the Manage tab in the mobile app. The registered user will be able to manage his/her profile by changing initially input registration details such as address, phone number, credit card information, license plate number, etc. Once changed the registered user can click submit to update the changes.

Use Case Diagram



Traceability Matrix

Req	PW	UC1	UC2	UC3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC10	UC11
ST-1	2									x		
ST-2	4									x		
ST-3	1				x		x			x		x
ST-4	4	x									x	
ST-5	7		x						x		x	
ST-6	1		x						x		x	
ST-7	5		x					x	x		x	
ST-8	4							x			x	
ST-9	6									x		
ST-10	2										x	
ST-11	3										x	
ST-12	2					x			x		x	
REQ-1	5								x	x	x	x
REQ-2	2	x										
REQ-3	4	x										x
REQ-4	1			x								
REQ-5	4	x							x		x	x
REQ-6	3		x						x		x	
REQ-7	4		x									
REQ-8	4		x					x	x	x	x	
REQ-9	2									x	x	x
REQ-10	3			x						x		
Max PW	-	4	7	3	1	2	1	5	7	6	7	5
Total PW	-	14	24	4	1	2	1	5	31	27	46	16

Fully-Dressed Descriptions

Use Case UC-2	Reservation
Related Requirements:	ST-5, ST-6, ST-7, REQ-6, REQ-7, REQ-8
Initiating Actor:	Registered User
Actor's Goal:	To reserve a spot for a requested period of time.
Participating Actors:	Database, Mobile App
Preconditions:	The user is registered within the RUParking system.
Postconditions:	The user has a reservation for the requested period of time if spots are available.

Flow of Events for Main Success Scenario:

- 1. The user logs into their account on the Mobile App.
- ← 2. The database confirms username and password.
- ← 3. Display page with user options. ("Reserve a Spot", "Walk-in Vacancy", "Personal Info", etc.)
- 4. Customer clicks "Reserve a Spot".
- 5. Customer selects a date and start time and duration of reservation within the next 7 days and clicks "Submit".
- ← 6. Database checks for free spots during requested time period and calculates price.
- 7. User enters a license plate number (optional) and agrees to terms and conditions.
- ← 8. Database is updated with new reservation. Customer is billed and given a reference number.

Flow of Events for Extensions (Alternate Scenario):

- 3. a. Database checks for free spots and returns negative.
- ← 1. Customer is notified that no spots are available.

Use Case UC-3:	Elevator Entry
Related Requirements:	REQ-4, REQ-10
Initiating Actor:	Registered/Unregistered User
Actor's Goal:	To park within the reservation zone.
Participating Actors:	License Plate Reader 1, Elevator System, Elevator, Database
Preconditions:	The user arrives at the garage and enters the elevator.
Postconditions:	The user has parked in the reservation zone. The user has left the parking lot if there are no more available spots or if they do not have a reservation.

Flow of Events for Main Success Scenario:

- ← 1. The users arrival activates license plate reader 1 and it reads the license plate. This license plate is recognized within the system as belonging to a user with a reservation.
- ← 2. The system checks the database for the reservation.
- ← 3. The user is notified of their spot through the elevator screen.
- ← 4. The elevator moves to the appropriate floor.
- 5. The user parks in the assigned spot.
- ← 6. The elevator system relays the occupancy state of the spot back to the database.

Flow of Events for Extensions (Alternate Scenario #1):

- 1. a. The license plate is not recognized within the database.
 - ← 1. User is prompted to enter reference number.
 - 2. User enters reference number into elevator module.
 - ← 3. System validates reference number.
- Continue from number 2 of the Main Success Scenario.

Flow of Events for Extensions (Alternate Scenario #2):

- 2. a. Assigned parking is occupied by an overstaying customer.
 - ← 1. The system checks if there are other vacant spots within the reservation zone.
- Continue from number 3 of the Main Success Scenario.

Flow of Events for Extensions (Alternate Scenario #3):

2. b. Assigned parking is occupied by an overstaying customer and there are no more vacant spots left in the reservation zone.

← 1. The system checks if there are any emergency parking spots available.

Continue from number 3 of the Main Success Scenario.

Flow of Events for Extensions (Alternate Scenario #4):

2. c. Assigned parking is occupied by an overstaying customer, there are no more vacant spots left in the reservation zone, and there are no more vacant emergency parking spots.

← 1. The Elevator screen displays “The reservation cannot be fulfilled. You will be refunded your full transaction.” It will also display the number of vacant spots available in the walk-in zone and will prompt the customer to exit the elevator.

← 2. The system refunds the customer of his/her transaction.

Flow of Events for Extensions (Alternate Scenario #5):

1. b. License plate or reference number is not recognized within the system.

← 1. User will be notified that the reservation does not exist.

← 2. The customer will be prompted to exit the elevator.

Use Case UC-7	Reservation Management
Related Requirements:	ST-7, ST-8, REQ-8
Initiating Actor:	Registered User
Actor's Goal:	To extend or cancel reservation.
Participating Actors:	Mobile App, Database
Preconditions:	The user has a reservation within the system.
Postconditions:	The user has cancelled or extended their reservation.

Flow of Events for the Main Success Scenario:

- 1. The user logs into their account on the Mobile App.
- ← 2. The database confirms username and password.
- ← 3. Display page with user options. ("Reserve a Spot", "Walk-in Vacancy", "Personal Info", etc.)
- 4. The user clicks on the "Manage Reservations" button.
- ← 5. Display list of Reservations for user.
- 6. The user clicks on "Extend" on the reservation they wish to extend.
- 7. The user inputs the requested extension time.
- ← 8. The database checks if this requested extension time is available.
- ← 9. The Mobile App displays to the user that the extension has been made and reference number is returned.

Flow of Events for Extensions (Alternate Scenario #1):

- 8. a. The requested extension time is not available.
- ← 1. System notifies the user that the extension is not available.
Continue from number 1 from the Main Success Scenario.

Flow of Events for Extensions (Alternate Scenario #2):

- 6. a. User clicks on "Cancel" for the reservation they wish to cancel.
- ← 1. User is prompted to confirm cancellation.
- 2. User confirms cancellation by clicking "Confirm"
- ← 3. User is notified of their full refunded amount.
- ← 4. Database is updated with this cancellation and the refunded amount.

Use Case UC-9	Database Read
Related Requirements:	ST-1, ST-2, ST-3, ST-9, REQ-1, REQ-8, REQ-9, REQ-10
Initiating Actor:	Registered User, Operator
Actor's Goal:	To view walk-in zone vacancy or statistical data of the garage.
Participating Actors:	Database, Mobile App, Management GUI
Preconditions:	Registered User/Operator wants to check walk-in zone vacancy/statistical data of the parking garage.
Postconditions:	Registered User/Operator has knowledge of walk-in zone vacancy/statistical data of the parking garage.

Flow of Events for the Main Success Scenario:

- 1. The user opens up the Mobile App.
- 2. The user logs in with a username and password.
- ← 3. Database confirms username and password.
- ← 4. Display page with user options. ("Reserve a Spot", "Walk-in Vacancy", "Personal Info", etc.)
- 5. The user clicks on the "Walk-in Vacancy" button in the Mobile App.
- ← 6. The Mobile App displays the number of vacant parking spots within the walk-in zone of the parking garage.

Flow of Events for Extensions (Alternate Scenario #2):

- 1. a. The operator opens up the management interface.
- 1. The operator logs in with a username and password.
- ← 2. Database confirms username and password.
- ← 3. Display page with user options. ("Rate Changes", "View Statistics", etc.)
- 4. The operator clicks on the "View Statistics" button on the GUI.
- ← 5. The GUI displays the statistical data of the parking garage.

System Sequence Diagrams

UC-2: Reservation

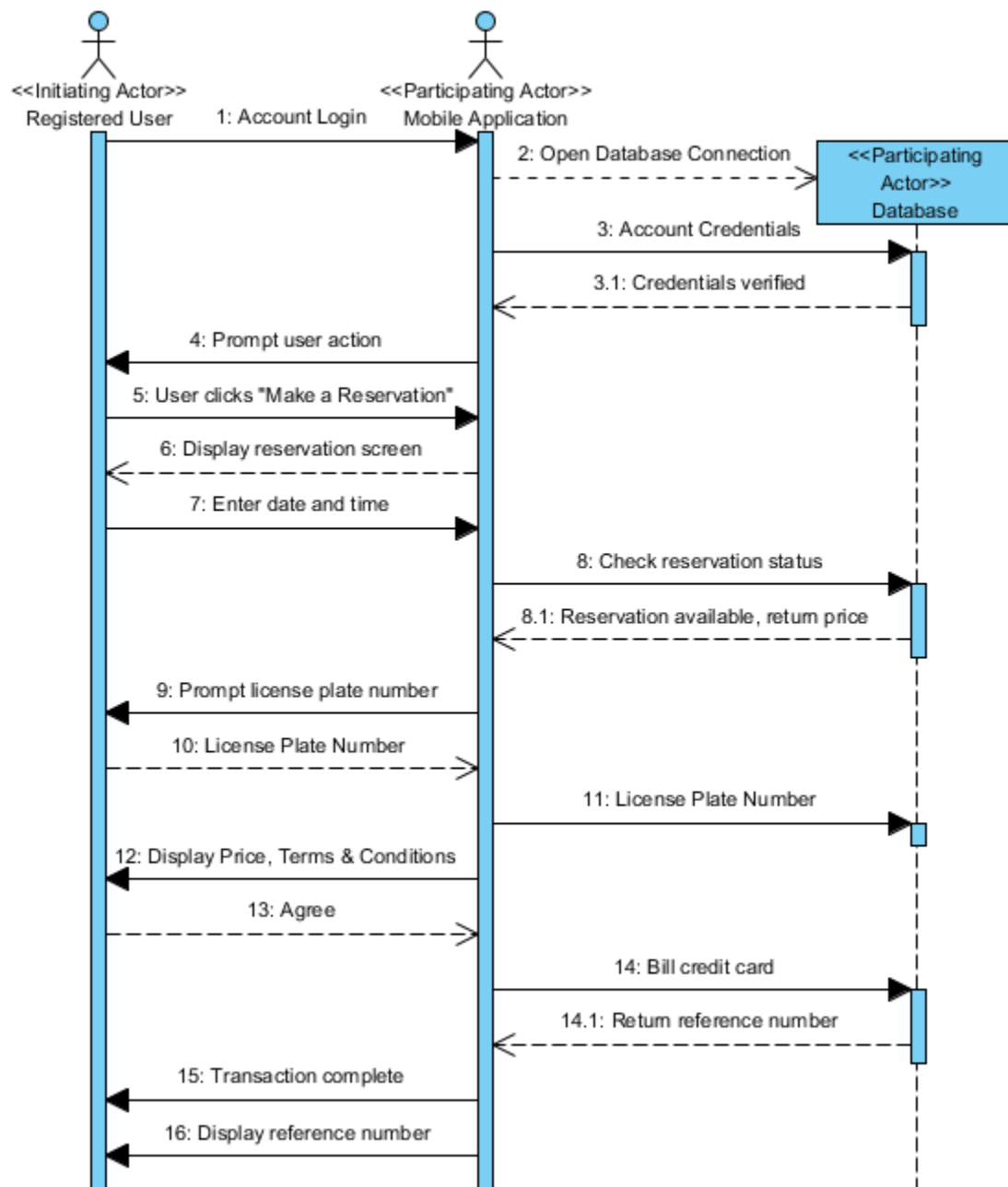


Fig 1. Main Success Scenario

UC-3: Elevator Entry

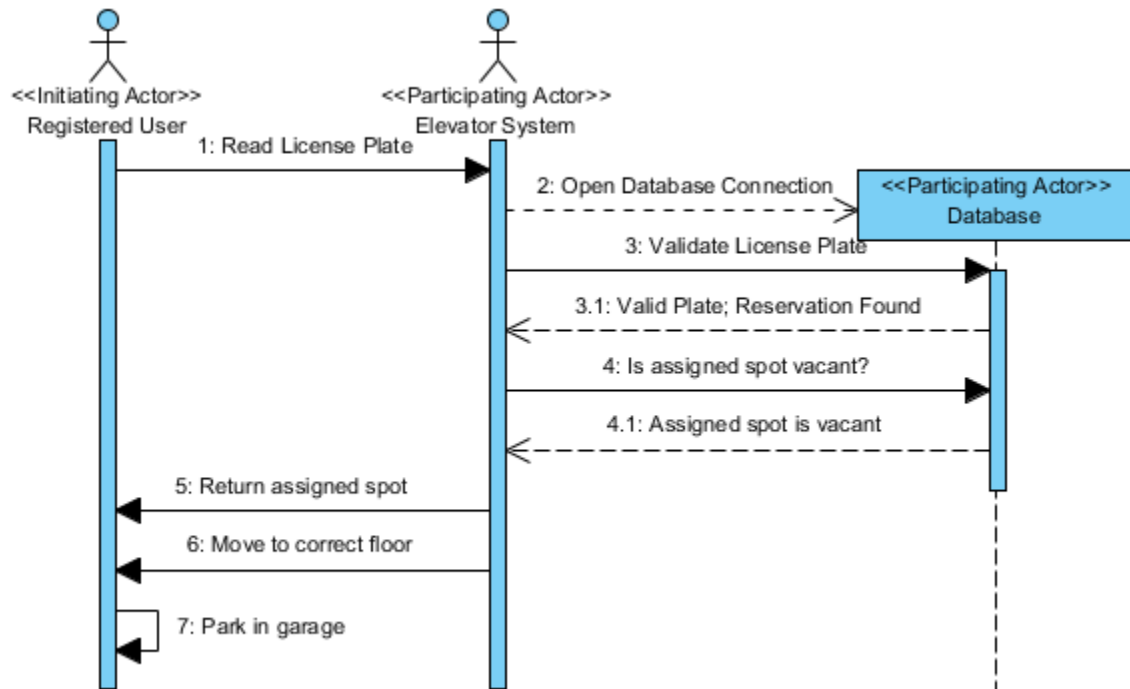


Fig 2a. Main Success Scenario

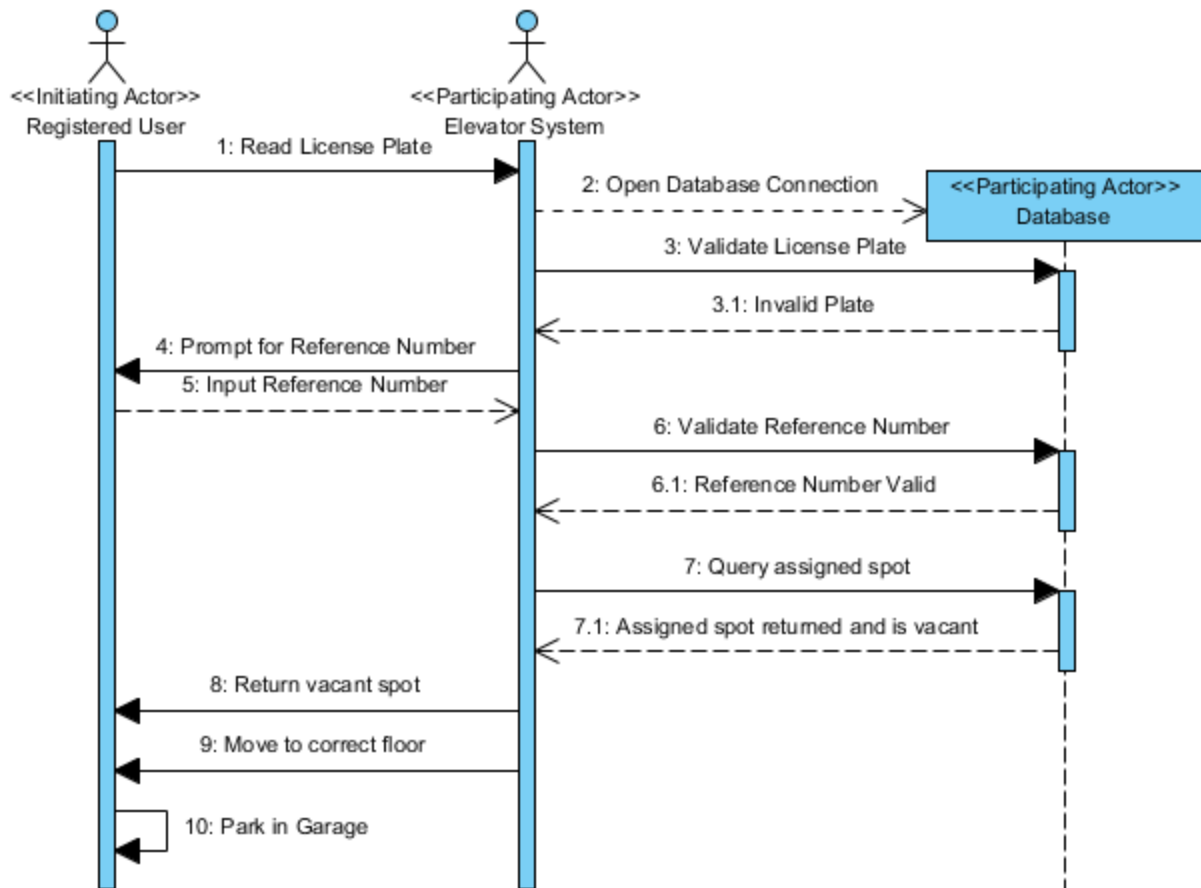


Fig 2b. Alternate Scenario 1

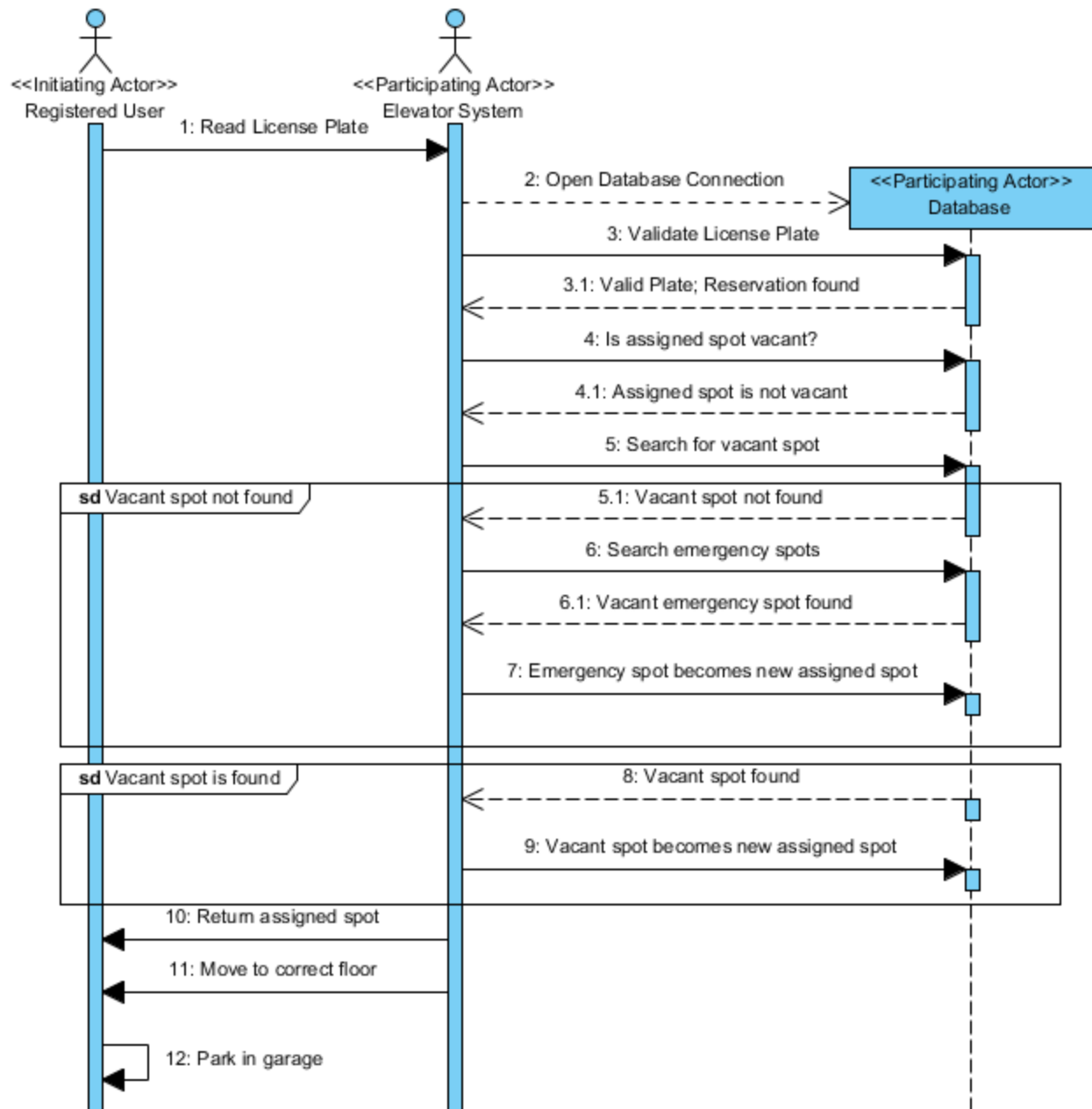


Fig 2c. Alternate Scenarios 2/3

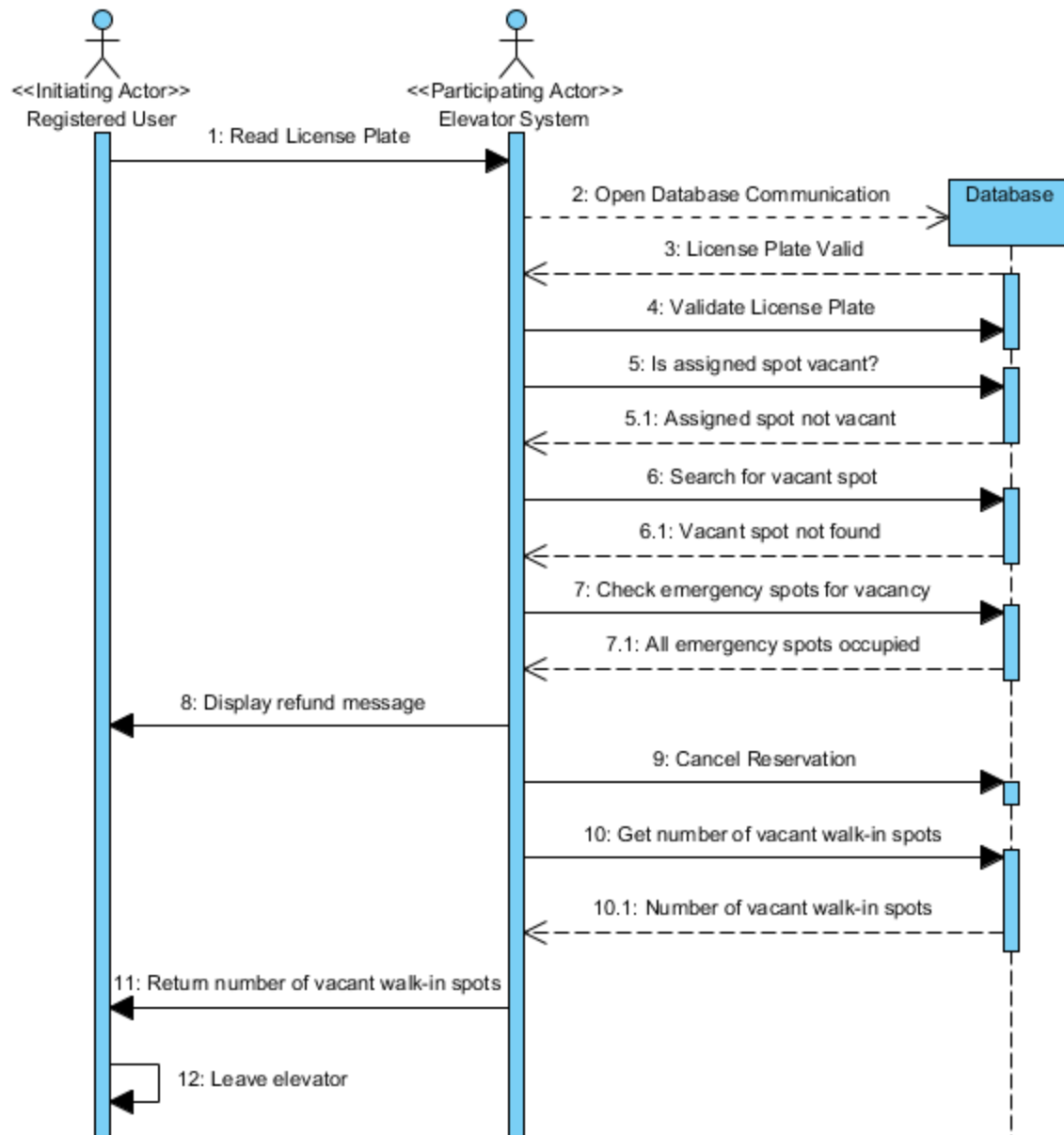


Fig 2d. Alternate Scenario 4

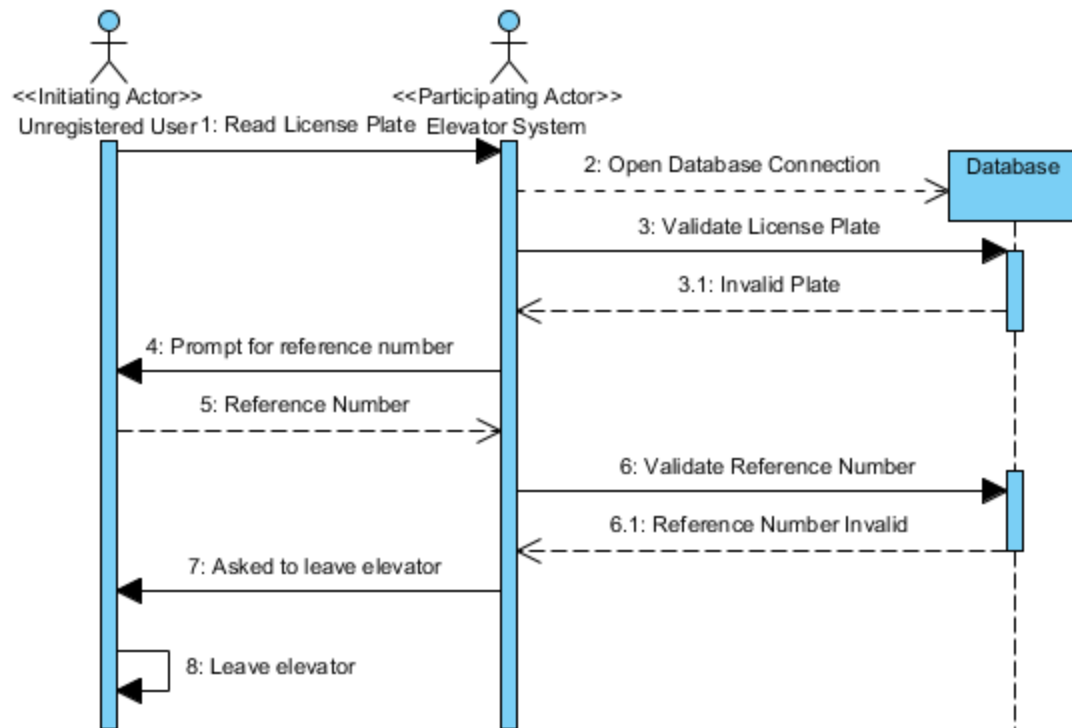


Fig 2e. Alternate Scenario 5

UC-7: Reservation Management (Figure-3)

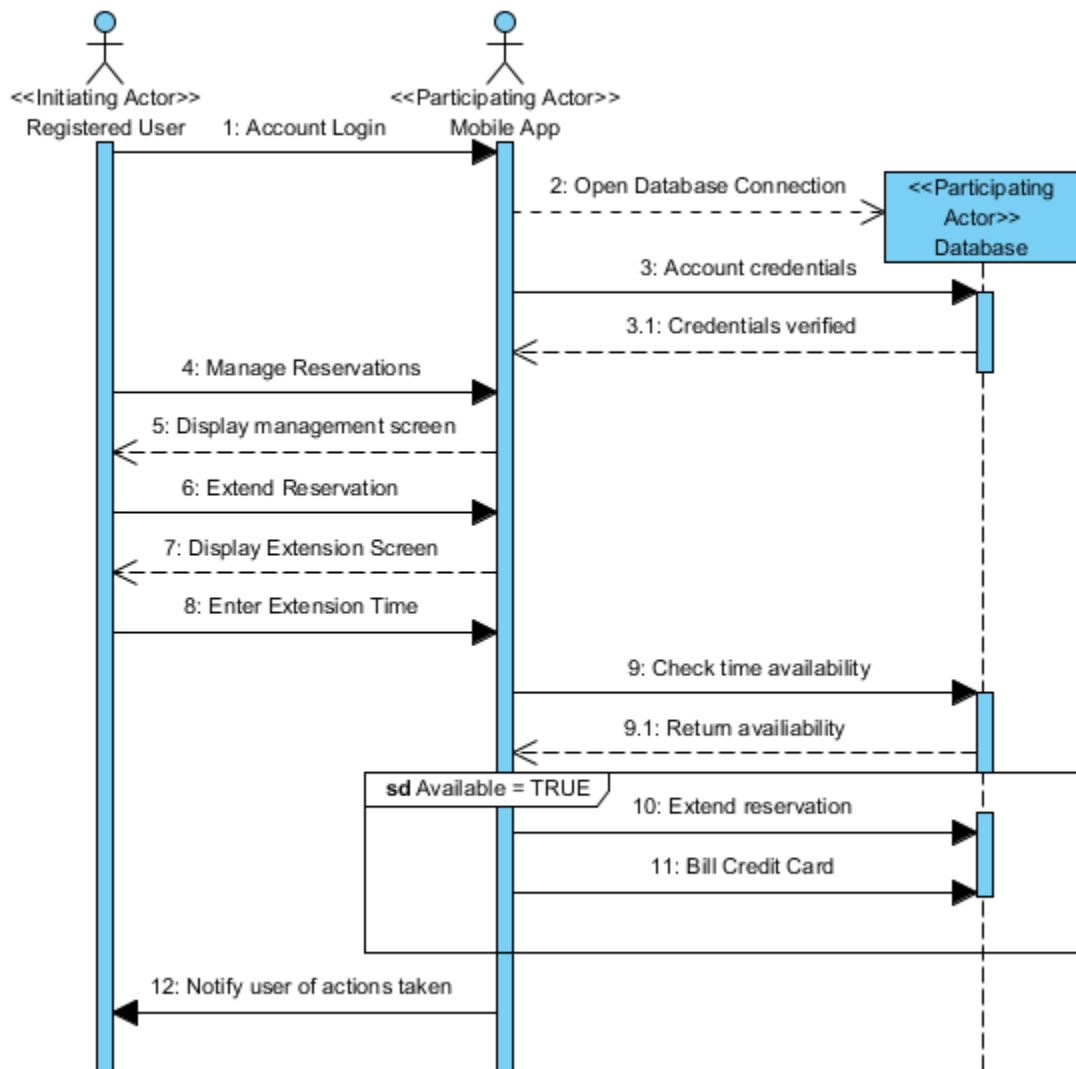


Fig 3a. Main Success Scenario

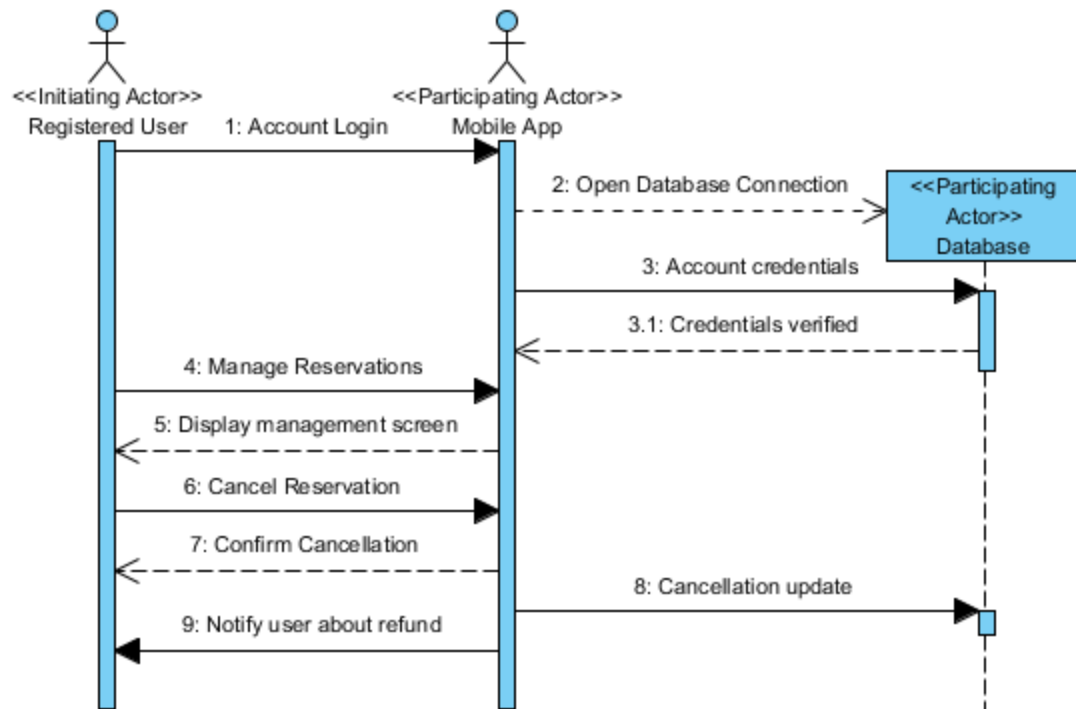


Fig 3b. Alternate Scenario

UC-9: Database Read

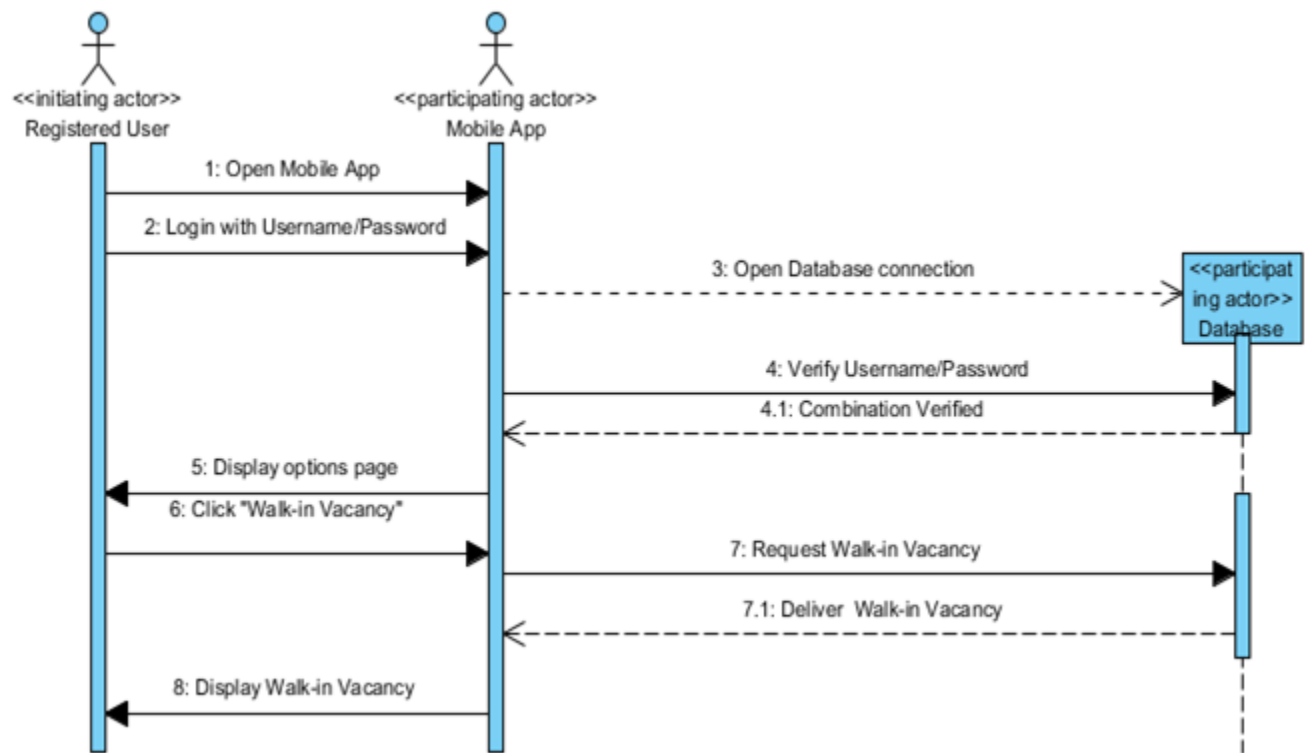


Fig 4a. Main Success Scenario

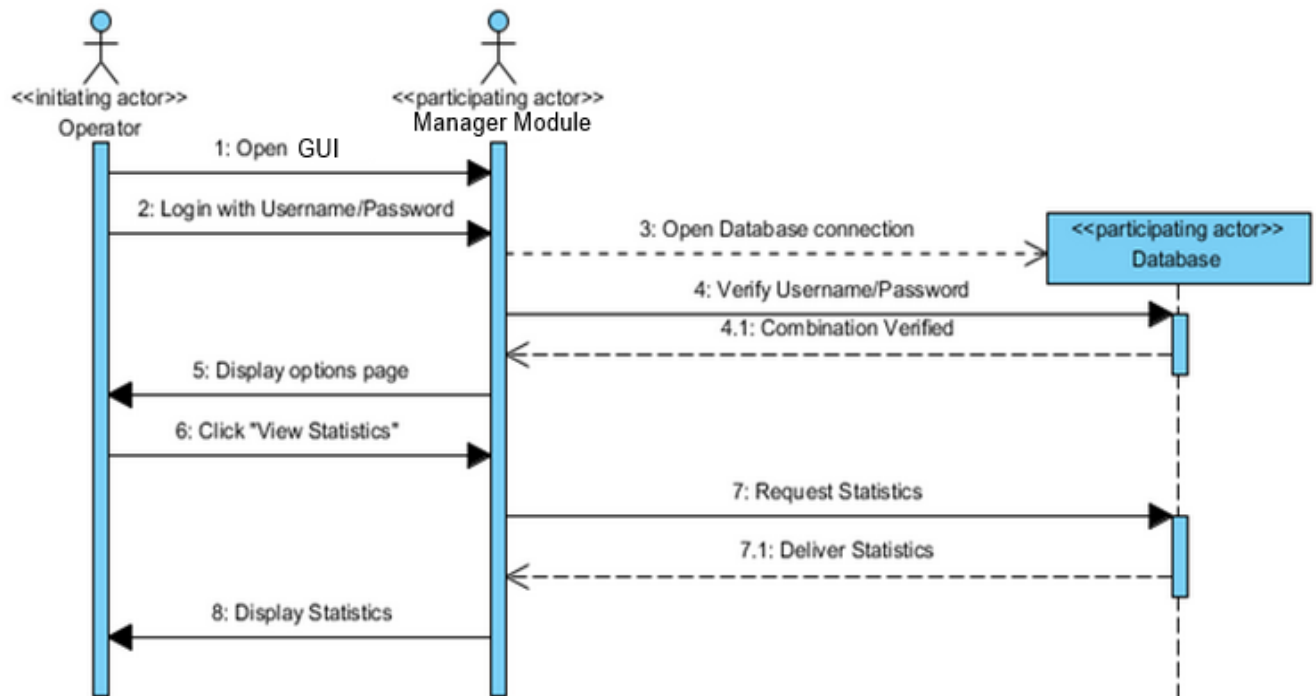


Fig 4b. Alternate Scenario

5. User Effort Estimation

UAW calculation:

Actor	Complexity	Weight
Garage Operator	Complex	3
Unregistered User	Average	2
Registered User	Average	2
Database	Average	2
Elevator System	Simple	1
Elevator	Complex	3
Entrance Gate	Simple	1
Exit Gate	Simple	1
License Plate Reader 1	Simple	1
License Plate Reader 2	Simple	1
Exit Monitor	Simple	1
Operator	Average	2
Payment Terminal	Simple	1
Ticketing Terminal	Simple	1
Vacancy Monitor	Simple	1
Management Interface	Average	2

$$\underline{\text{UAW}} = (9 \times \text{Simple}) + (5 \times \text{Average}) + (2 \times \text{Complex}) = (9 \times 1) + (5 \times 2) + (2 \times 3) = 25$$

UUCW Calculation:

Use Case	Category	Weight
UC-1 Registration	Average	10
UC-2 Reservation	Average	10
UC-3 Elevator Entry	Complex	15
UC-4 Walk-In	Simple	5
UC-5 Reservation Exit	Average	10
UC-6 Walk-In Zone Exit	Simple	5
UC-7 Reservation Management	Complex	15
UC-8 Mobile Payment	Average	10
UC-9 Database Read	Average	10
UC-10 Database Write	Average	10
UC-11 Personal Information	Simple	5

$$\underline{\text{UUCW}} = (3 \times \text{Simple}) + (6 \times \text{Average}) + (2 \times \text{Complex}) = (3 \times 5) + (6 \times 10) + (2 \times 15) = 105$$

$$\underline{\text{UUCP}} = \text{UAW} + \text{UUCW} = 25 + 105 = 130$$

TCF Calculation:

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	Mobile App integrated with database	2	3	$2 \times 3 = 6$
T2	Users expect ease of access with minimal complexity	1	3	$1 \times 3 = 3$
T3	Elevator interface must be integrated with database	2	3	$2 \times 3 = 6$
T4	Concurrent use	1	4	$1 \times 4 = 4$
T5	Elevator interface ease of use	1	3	$1 \times 3 = 3$
T6	Ease of install of the mobile app	1	3	$1 \times 3 = 3$
T7	No unique training needs	1	0	$1 \times 0 = 0$

Technical factor total = 25

TCF = $C1 + C2 \times \text{Technical Factor Total}$

$C1 = 0.6$

$C2 = 0.01$

$\text{TCF} = 0.6 + 0.01(25) = 0.85$

ECF Calculation:

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor
E1	Basic computer knowledge for operators	1.5	2	$1.5 \times 2 = 3$
E2	Highly motivated staff	1	4	$1 \times 4 = 4$
E3	Stable requirements expected	2	5	$2 \times 5 = 10$

Environmental Factor Total = 17

ECF = $C1 + C2 \times \text{Environmental Factor Total}$

$C1 = 1.4$

$C2 = -0.03$

$ECF = 1.4 + (-0.03)(17) = 0.89$

Use Case Points (UCP) Calculation:

$UCP = UUCP \times TCF \times ECF = 130 \times 0.85 \times 0.89 = 98.345$ or 98 points

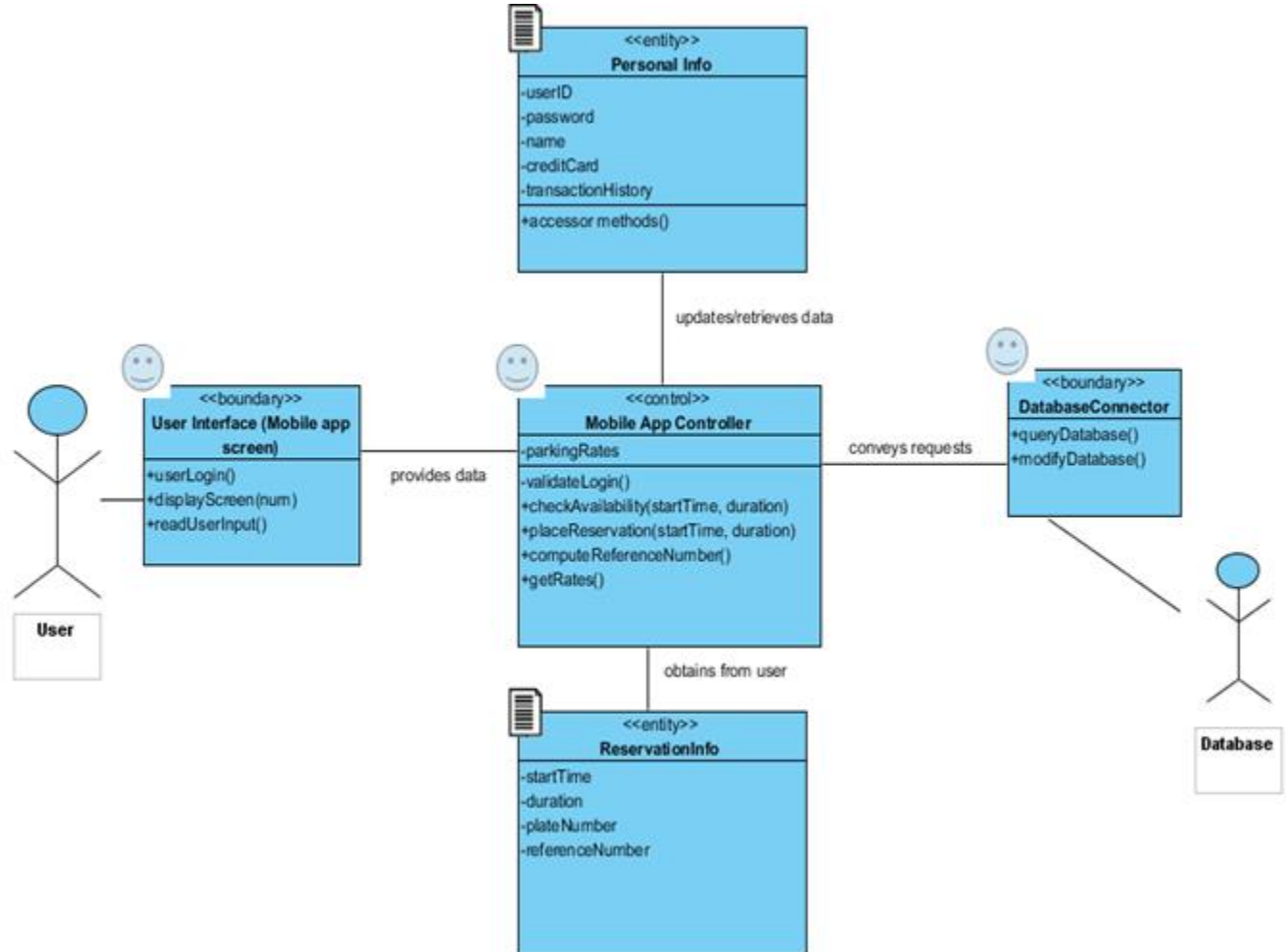
Productivity Factor (PF) = 28

Duration = $UCP \times PF = 98 \times 28 = 2744$

6. Domain Analysis

Domain Model

Domain Model for UC-2 (Reservation)



Rationale for Domain Model:

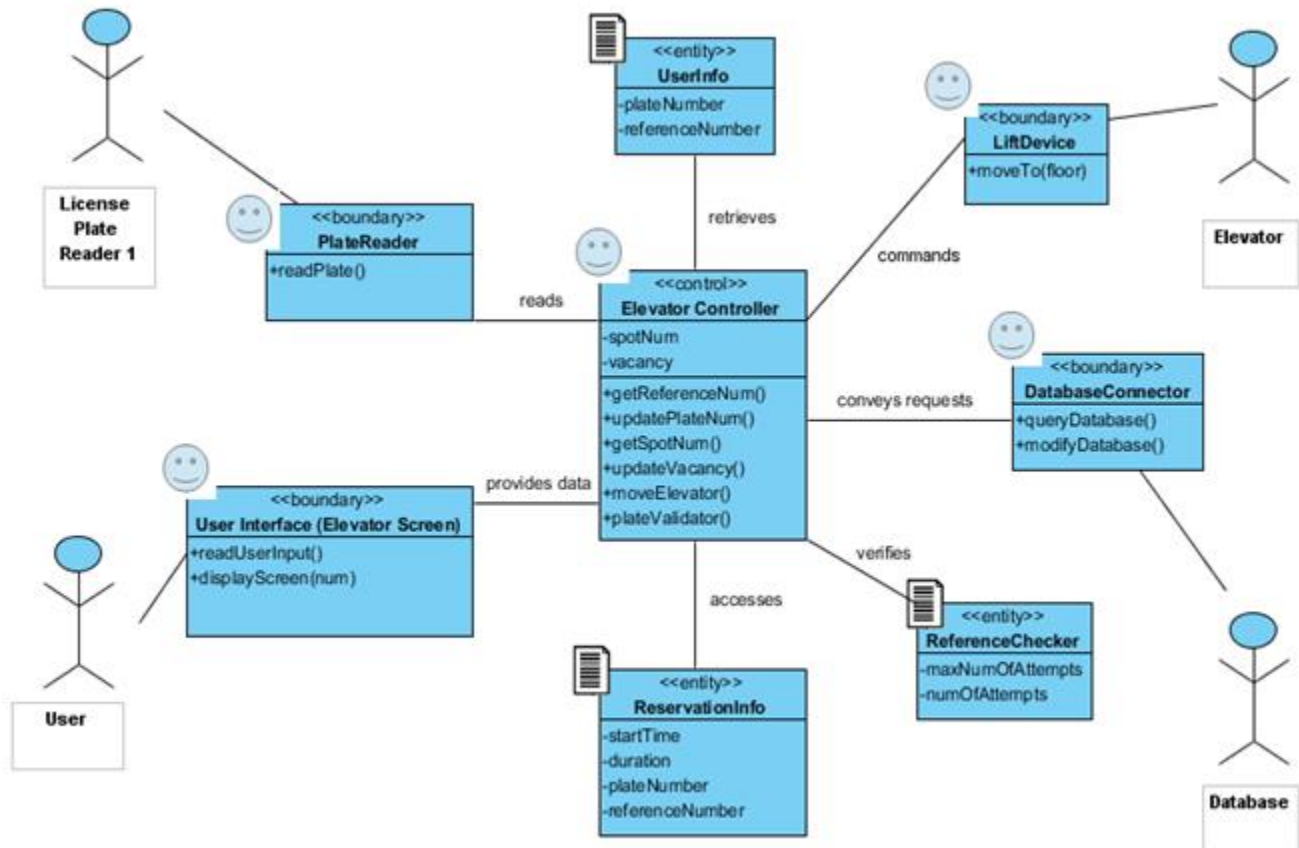
Two actors are required for UC-2 (Reservation). We develop the boundary classes first - the user interface and the database connector. Their tasks are simple - to act as the liaison between the system and their respective actors. The user interface receives prompts from the main controller to display to the user, while the database connector allows the main controller to read and write to the database, which contains all parking garage data.

Next, we created the mobile app controller, which handles all flow of information; all data coming to and from the system goes through this controller. Next, there are a number of tasks to be done - retrieve walk-in data, validate login information, keep track of the current parking rates, and place reservations. The traditional principle of high cohesion/low coupling states that we should create a separate class for each of these tasks. However, we opted to deviate from this principle for the following reason.

Each of these tasks involves opening a connection to the database. If we instead chose to create four separate classes to accomplish the task, they would all share the common responsibility of opening a connection to the database. This would violate the expert doer principle, as each class would have 100% of the garage information but only operate on a fraction of it. This is both redundant and inefficient; thus, we elected to place these responsibilities on the controller.

Finally, we created the containers ReservationInfo and PersonallInfo to store relevant information. We created two separate classes rather than a single generic storage class to maintain the principles of high cohesion and low coupling.

Domain Model for UC-3 (Elevator Entry)

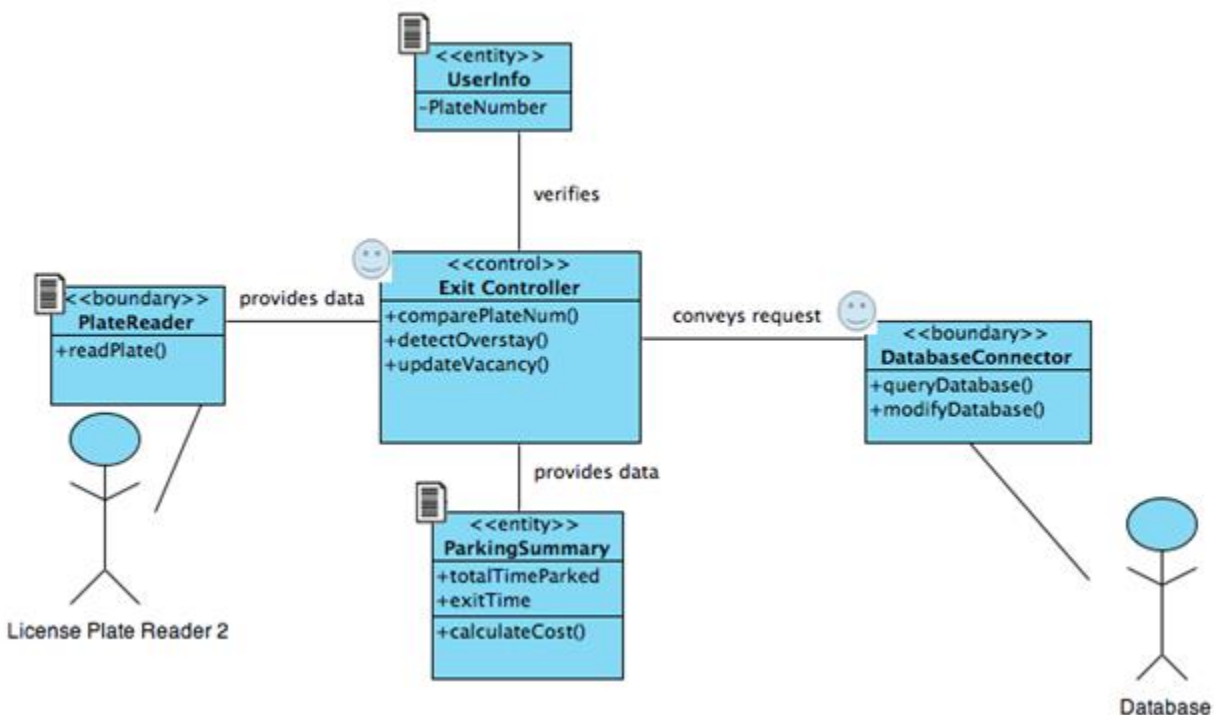


Rationale for Domain Model:

In this domain model, there are four actors involved. Again, we developed the boundaries first: User Interface, Database Connector, Plate Reader, and Lift Device. Next, we created the elevator controller; again, all input and output data must pass through this class first. It maintains organization across the system and is the central point of the system.

For the same reason as explained in the previous domain model, we chose to sacrifice high coupling/low cohesion in exchange for maintaining the expert doer principle by assigning responsibilities to the elevator controller. Yet, we chose to keep the relevant information separated into two categories - reservation and user. This action is in accordance with the principles of high cohesion and low coupling; there is no reason to deviate from it for this use

Domain Model for UC-5 (Reservation Exit)



Rationale for Domain Model:

In this domain model, there are two actors involved. As always, we developed the boundaries first: Plate Reader and Database Connector. Following the same design concept as the other domain modules, we created the exit controller; all input and output data is directed through this class.

Next, we developed the classes **UserInfo** and **ParkingSummary** to keep track of data relevant to the task at hand. Since **ParkingSummary** already keeps track of the total time parked and the exit time of the car, we give it the responsibility of computing the cost of overstay since it already knows the required information (Expert Doer Principle).

Concept Definitions

Key:

K - Knowing

D - Doing

UC-2 (Reservation)

Responsibility Description	Type	Concept Name
Reads input from the user and passes it to the Main Controller. Displays appropriate screens in correspondence with user inputs. This includes user login and user decisions once logged into the mobile app.	D	User Interface (Mobile Interface)
Contains all relevant information regarding the registered user. This information include name, credit card information, email, password and transactions history.	K	Personal Info
Responds to user input and directs program flow appropriately by interacting with the database. Tasks include: Validating user login, checking reservation availability for a given date and time, computing reference number for a given reservation, and sending the reference number to the user interface display. Also, it fetches the current parking rates in order to compute the total cost for a given reservation	D	Mobile App Controller
Establishes a connection to the database. Allows the mobile application to access the database in order to retrieve essential information as well as update the database with new reservations.	D	Database Connector
Contains all relevant information regarding the user's reservation. This includes start date, reservation as well as optional plate number.	K	Reservation Info

UC-3 (Elevator Entry)

Responsibility Description	Type	Concept Name
When prompted by the elevator system, it reads the plate number. Using pre-existing text recognition software, it is able to extract the correct license plate number. This value is passed to the controller.	D	PlateReader
Serves as the communication medium between the user and the elevator system. Receives messages from the elevator controller and displays them to the user.	D	User Interface(Elevator Screen)
Contains relevant information about the user: license plate number as read from the camera, and the reference number if needed.	K	UserInfo
Acts as the central unit that manages all flow of data. It reacts to user input and establishes a connection to the database when necessary.	D	ElevatorController
Contains all relevant information regarding the user's reservation, such as start time and duration.	K	ReservationInfo
Establishes a connection to the database. Allows the elevator system to access occupancy information and to update tables in real time.	D	Database Connector
Counts the number of times the user tries enter his/her reference number, and alerts the controller if too many failed attempts are made.	K	ReferenceChecker
Lifts the elevator to the appropriate floor as commanded by the elevator controller.	D	LiftDevice

UC-5 (Reservation Exit)

Responsibility Description	Type	Concept Name
Receives plate information from the PlateReader and opens a connection with the database. After retrieving reservation information, it checks for overstay and displays overstay on exit screen if applicable. Lastly, it updates the vacancy information in the database.	D	ExitController
The DatabaseConnector gets data from the Controller and modifies the database accordingly.	D	Database Connector
When prompted by the elevator system, it reads the license plate from the car. Using pre-existing text recognition software, it is able to extract the correct license plate number. This value is passed to the controller.	D	PlateReader
ParkingSummary calculates the total time parked by the user, records the exit time and calculates the total cost in case of an overstay.	K	ParkingSummary
UserInfo provides the ExitController with user's license plate number in order verify the customer leaving the garage.	K	UserInfo

Association Definitions

UC-2 (Reservation)

Concept Pair Name	Associated Description	Association Name
User Interface Mobile App Controller	Controller passes information to the user interface to be displayed to the screen. User Interface provides the controller with input data from users.	Provides data
Personal Information Mobile App Controller	Controller accesses and modifies a user's personal information as necessary.	Update/retrieve data
Reservation Info Mobile App Controller	Reservation info provides the controller with reservation data such as plate number, start time and reservation duration.	Provides data
DatabaseConnector Mobile App Controller	The controller provides database connector with data to be updated on the database. The controller also acquires data from database through database connector.	Conveys request

UC-3 (Elevator Entry)

Concept Pair Name	Associated Description	Association Name
PlateReader ElevatorController	License plate reader 1 provides the controller with the user's license plate number.	Reads
UserInterface ElevatorController	User Interface communicates with the user, receiving input via the attached keypad. It receives display prompts from the controller.	Provide data
UserInfo ElevatorController	Controller stores plate and reference numbers in UserInfo, retrieving when necessary.	Retrieve Data

ReservationInfo ElevatorController	Reservation info provides the controller with data such as plate number, start time and reservation duration upon prompt.	Provides data
ReferenceChecker ElevatorController	ReferenceChecker keeps track of the number of attempts for entering a reference number and the maximum number of attempts allowed.	Verifies
LiftDevice ElevatorController	ElevatorController commands the LiftDevice to move to the correct floor.	Commands
DatabaseConnector ElevatorController	The controller provides database controller with data to be updated within the database. The controller also acquires data from database through database connector.	Conveys request

UC-5 (Reservation Exit)

Concept Pair Name	Associated Description	Association Name
PlateReader Exit Controller	License plate reader 2 provides the controller with user's license plate number.	Reads
UserInfo Exit Controller	The license plate number of the user leaving the garage, as returned by the Plate Reader, is stored within UserInfo.	Verifies
ParkingSummary Controller	Controller retrieves the parking summary for the customer leaving the garage to calculate total cost and detect overstay and displays on exit screen if applicable.	Retrieves
Controller DatabaseController	The controller provides database controller with data to be updated within the database, regarding vacancy and/or additional billing. The controller also acquires data from the database through database connector.	Conveys request

Attribute Definitions

UC-2 (Reservation)

Concept	Attributes	Attribute Description
<i>Mobile App Controller</i>	parkingRates	Used for the system to charge the correct rates
<i>Personal Info</i>	Email	Used to identify the user by his/her username
	Password	Used to authenticate user identity
	Name	User's name
	CreditCard	Used to pay for a parking reservation
	transactionHistory	Keeps track of customer's previous reservations
<i>Reservation Info</i>	startTime	Used to determine when the reservation interval is to begin.
	duration	Length of the reservation, in hours.
	plateNumber	Optional field. Stores the license plate number of the car for which the spot is reserved.
	referenceNumber	Number unique to the reservation. It is generated by the reservation details.

UC-3 (Elevator Entry)

Concept	Attributes	Attribute Description
<i>User Info</i>	plateNumber	Optional license plate number of the user's car. Will be used for user reservations
	referenceNumber	User's inputted reference number, if prompted by the elevator interface.
<i>Elevator Controller</i>	spotNumber	Parking spot number. Used to indicate to user where his/her parking spot is located
	Vacancy	Current vacancy state of the walk-in zone.
<i>Reservation Time</i>	startTime	The starting time of the reservation.
	duration	Length of the reservation, in hours.
	plateNumber	The license plate number of the car for which the reservation is made.
	referenceNumber	Number unique to the reservation. It is generated by the reservation details.
<i>Reference Checker</i>	Number of attempts	Used to check the number of attempts the user has tried to enter his/her reference number

UC-5 (Reservation Exit)

Concept	Attributes	Attribute Description
<i>Exit Controller</i>	Vacancy	Used to update parking spots in the reservation zone.
<i>User Info</i>	plateNumber	The license plate of the car leaving the garage. It is matched to a reservation in the database.
<i>Parking Summary</i>	totalTimeParke d	Amount of time spent in the garage by the car. Used to detect overstay.
	exitTime	Time of exit. Compared with reservation end time to detect overstay.

Traceability Matrix

Req	P W	Reserve				Elevator							Exit				
		Personal Info	User Interface	Mobile App Controller	DatabaseConnector	ReservationInfo	PlateReader	ReferenceChecker	Elevator Controller	User Interface	ReservationInfo	DatabaseConnector	Exit Controller	PlateReader	ParkingSummary	UserInfo	DatabaseConnector
UC-1 Reg	14	x	x	x	x												
UC-2 Res	24	x	x	x	x	x											
UC-3 Elev Ent	4						x	x	x	x	x	x					
UC-4 Walk-in	1																
UC-5 Res Exit	2												x	x	x	x	x
UC-6 Walk Exit	1																
UC-7 Res Manage	5	x	x	x	x	x											
UC-8 Pay Mobile	31	x	x	x	x	x											
UC-9 DBR	27				x							x					x
UC-10 DBW	46				x							x					x
UC-11 Personal Info	16	x	x	x	x												

We have split the set of requirements into 3 categories : Reserve, Elevator, and Exit. This traceability matrix maps the requirements to our use cases. The Priority Weights have been copied from the total PW as calculated in the previous matrix, mapping the use cases with the

functional/non-functional requirements and user stories. There are a few use cases that do not map to any requirements – namely, the ones regarding the walk-in zone. This is because the walk-in zone is not a part of our system; RUParking is designed to manage the reservations for registered customers. This is further affirmed by the priority weights of UC4 and UC6 (they have PW 1).

System Operation Contracts

UC-2 Reservation

Operation	Place Reservation
Functionality	Reserve a parking spot on the RUParking System
Exceptions	None
Pre-Condition	The user is registered within the RUParking System
Post-Condition	The user has a parking spot reserved within the system for the requested time period.

Operation	Assign Parking Spot
Functionality	The mobile app assigns a parking spot but does not display it to the user. It also writes the reservation details to the database.
Exceptions	None
Pre-Condition	The mobile app has found a place for the user during the time period.
Post-Condition	The mobile app has assigned the user a parking spot and the change was passed to the database.

Operation	Compute Reference Number
Functionality	The system assigns a reference number to the user for specified reservation
Exceptions	None
Pre-Condition	The user has successfully paid for the reservation.
Post-Condition	The user is successfully assigned a reference number

UC-3 Elevator Entry

Operation	Read License Plate
Functionality	The license plate camera reads the license plate number of the car. Using external software it extracts the license plate number as a string to be passed to the elevator system.
Exceptions	None
Pre-Condition	A car has entered the elevator.
Post-Condition	The userInfo field was updated with the license plate number of the car in the elevator.

Operation	Read Reference Number
Functionality	Read the reference number is entered by the user in elevator user interface
Exceptions	None
Pre-Condition	The license plate is not recognized by the elevator system.

Post-Condition	The reference number field within the UserInfo class was updated with the read license plate number.
----------------	--

Operation	Re-assign Parking Spot
Functionality	The system finds a vacant spot for the user.
Exceptions	The elevator system is not able to assign the user a parking spot within the entire parking garage including emergency parking spots.
Pre-Condition	The user has entered the elevator within the allowed time limit of his/her reservation, and their assigned spot is being occupied by an overstaying customer.
Post-Condition	The user's reservation was moved to a different parking spot for the same time period as the original.

Operation	Display Parking Spot
Functionality	Display the parking spot to the user.
Exceptions	User does not have a reserved parking spot.
Pre-Condition	The user has an identified reservation. The car has entered the elevator within the allowed time limit during their reservation period.
Post-Condition	The user was informed of his or her parking spot assignment.

Operation	Operate Lift
Functionality	The elevator takes the user to the correct floor.

Exceptions	None
Pre-Condition	The user has an identified reservation and has been informed of the correct spot.
Post-Condition	The user was taken to the floor of the assigned parking spot.

UC-7 Reservation Management

Operation	Cancel Reservation
Functionality	Cancel the reservation that exists within the database.
Exceptions	Cancellations are disallowed under business policy, or a reservation has not yet been placed.
Pre-Condition	The reservation the user wants to cancel exists in the database.
Post-Condition	The reservation no longer exists in the database, and the spot is open to reservation by other users. The user is fully refunded for their cancelled reservation.

Operation	Extend Reservation
Functionality	Attempt to extend an existing reservation for the user.
Exceptions	Reservation cannot be extended for the time period requested by the user.
Pre-Condition	The reservation the user wants to extend exists in the database.
Post-Condition	The reservation has been extended

	successfully for the new requested time, or no change is made (if extension cannot be fulfilled). The user has been informed of the actions taken. User is billed if reservation is extended.
--	---

Operation	Check Vacancy
Functionality	The mobile app checks the database for parking vacancy within the timeslot requested by the user.
Exceptions	None
Pre-Condition	The user requests an extension.
Post-Condition	A boolean value has been returned to the mobile app - true if the timeslot is clear, and false otherwise.

UC-9 Database Read

Operation	Connect to Database
Functionality	The mobile app, entry module, exit module, elevator console establish a connection with the database, allowing for I/O commands.
Exceptions	An entity not allowed to access the database tries to connect to it.
Pre-Condition	A request to connect to the database is made.
Post-Condition	The connection to the database is established.

Operation	Get Data
Functionality	The requested data is fetched by the database.
Exceptions	A request is made to fetch data that doesn't exist within the system.
Pre-Condition	A request to fetch specific database entries is made.
Post-Condition	The requested data entry is fetched and returned.

Mathematical Model

There are no mathematical models utilized for this design.

7. Interaction Diagrams

UC-2: Reservation:

Goal: To make a reservation through the mobile app.

We used the expert doer principle in the process of assigning responsibilities. The expert doer principle states: that who knows should do the task. We assigned input/output responsibilities to the mobile app interface because it is the medium used to interact with the user. The mobile app controller knows how to interact with the user input and does this the best. Therefore, the mobile app controller is assigned computing as well as validating responsibilities. The mobile app controller also conveys requests to the database connector in order to achieve a specific goal. The database connector connects to the database and it queries and also modifies data.

The high cohesion principle could not be applied while assigning the tasks since the system was designed in a manner in which mobile app controller was responsible for all computations. The Low coupling principle could not be applied since the system was designed in a manner in which the database controller was responsible for all communication tasks with the database. Also the mobile app interface was responsible for all communication tasks with the user input/output.

Design Patterns:

Publisher-Subscriber: There are very few actors involved in this use case. Communication only occurs between the user and mobile app, and between mobile app and database. The pub-sub design pattern is inappropriate in this case, and our system would be better off without it.

Command: This use case implements the Command design pattern, for database queries. There is a single function called 'query', and its argument changes depending on the situation. Depending on information including the table to be queried, and the task at hand, the program builds a SQL command and executes it using the 'query' function. This design pattern improves our system because it opens our program to message passing - that is, in the case that the information needed to build a command is scattered across multiple classes, each class can add its own contribution to a common command string that will be used to execute a complete SQL command. It also decreases coupling, as it lessens the knowing requirements of each individual class. Further, it is a good demonstration of the expert doer principle, since the classes that know the key pieces of information are the ones that help assemble the command.

Decorator: The decorator pattern is not needed in this use case - there are no functionalities to be extended in this use case.

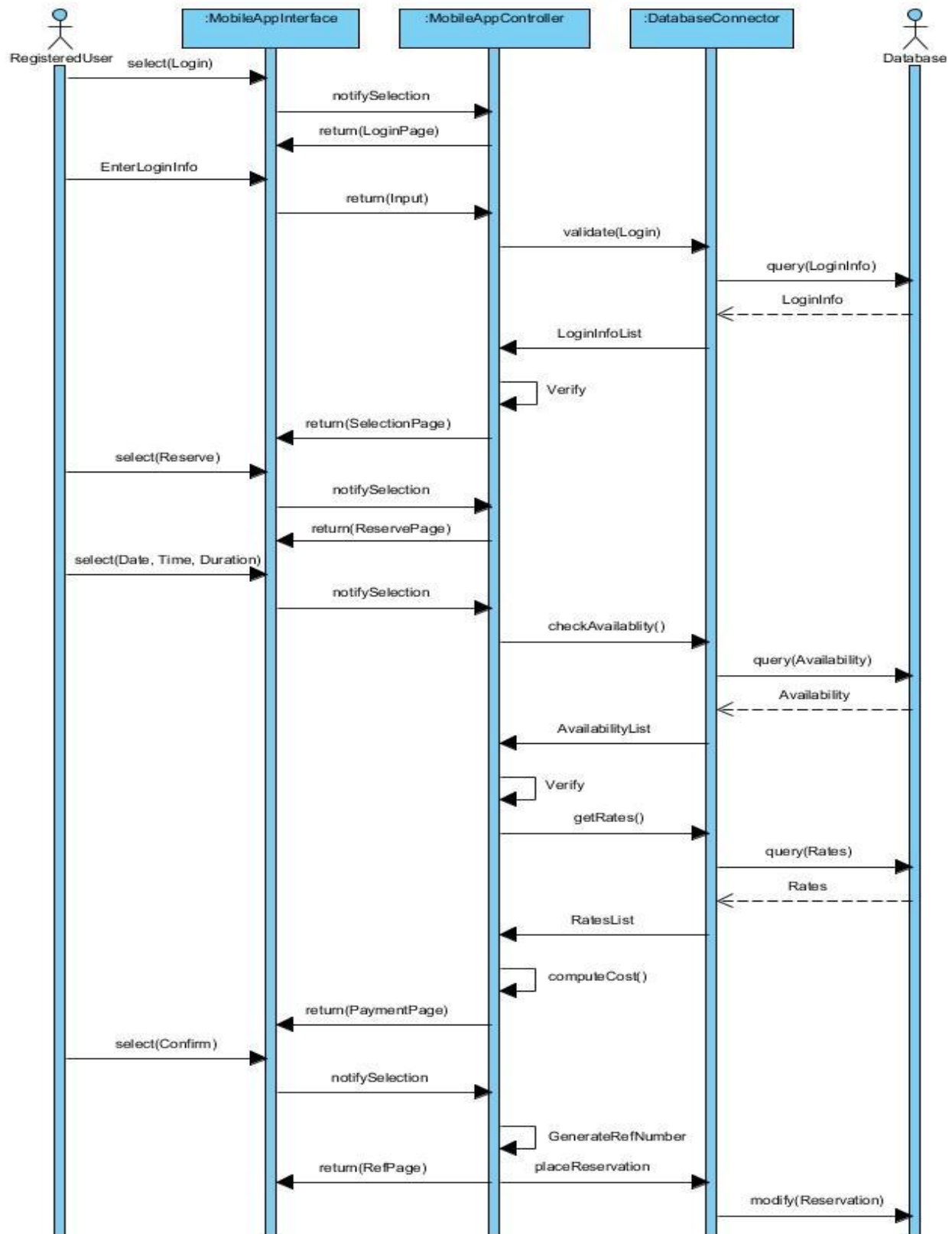
State: Because there is a natural flow of progression in this use case, implementing the state design pattern would unnecessarily complicate our design.

Proxy: Because the mobile app already restricts user input (values are selected from either lists or drop-down bars), there is no need to use the protection proxy.

Process (Main Success Scenario):

The user selects login on the mobile app interface. Next, the mobile app interface notifies the selection to the mobile app controller. The mobile app controller returns the login page. Next, the user enters their username as well as password. This input is transferred from the mobile app interface to the mobile app controller. Next the mobile app controller validates login. In order to validate login the database connector queries login information and this information is returned to the mobile app controller. Next, the mobile app controller verifies login and returns a selection page to the mobile app interface. Next, the user selects reserve and the mobile app interface notifies the mobile app controller of this input. The mobile app controller then returns a reserve page to the mobile app interface. The user then selects a date, time and the duration of their requested reservation time. This information is then passes to the mobile app controller. The mobile app controller then checks availability. In order to check availability the database connector queries the database for availability and this information is returned to the mobile app controller. Next, the mobile app controller verifies the availability and once verified it check the rates. The rates are queried by the database connector and the database then returns the rates to the mobile app controller. The mobile app controller then computes the cost and returns the payment page to the mobile app interface. Next, the user selects confirm and the mobile app controller is notified by the mobile app interface about the selection. The mobile app controller then generates a reference number, which is then returned to the mobile app interface. Next, the mobile app controller places the reservation. In order to place this reservation the database connector updates the reservation information as well as the reference number within the database.

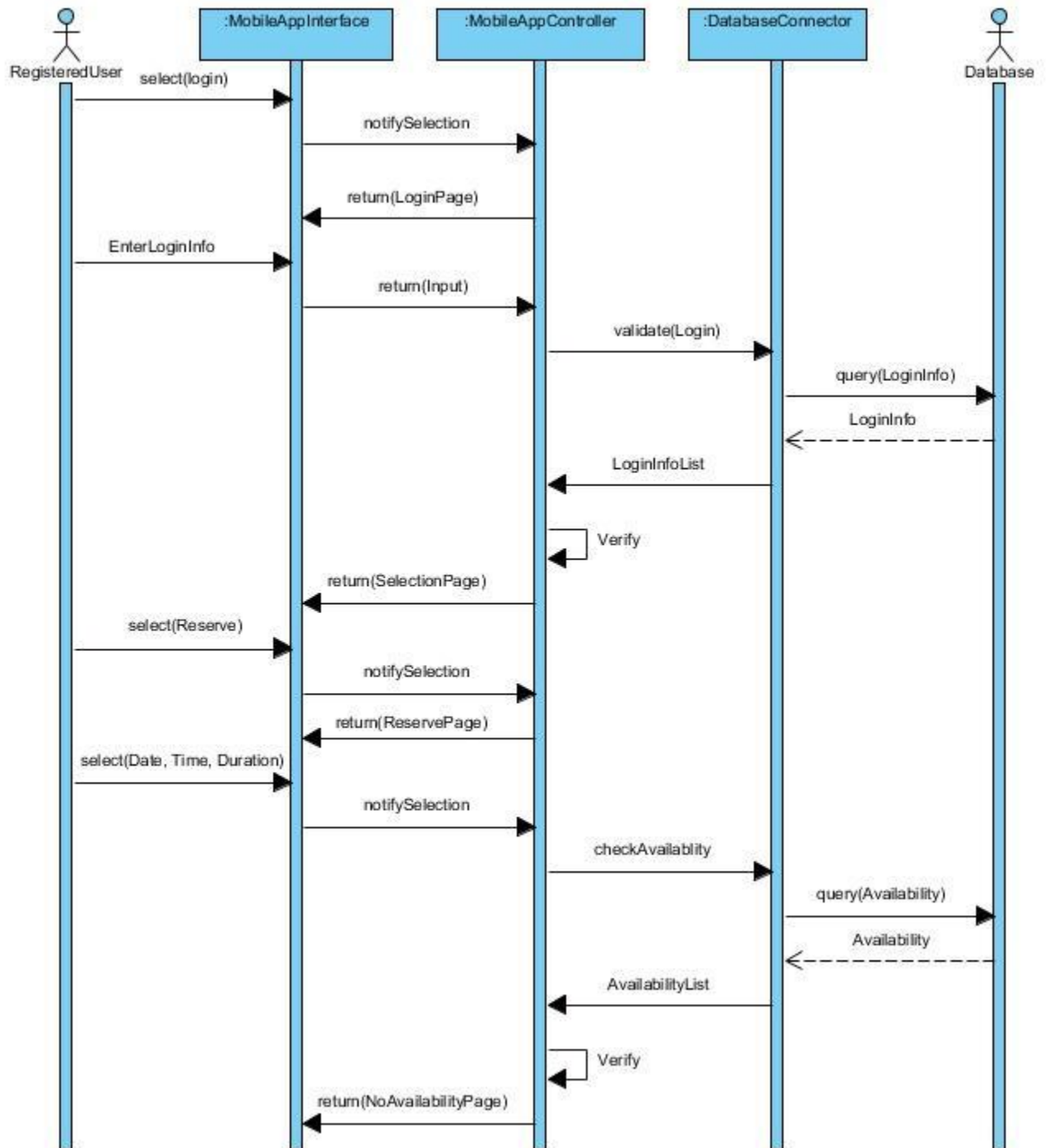
Main Success Scenario:



Process (Alternate Scenario):

The user selects login on the mobile app interface. Next, the mobile app interface notifies the selection to the mobile app controller. The mobile app controller returns the login page. Next, the user enters their username as well as password. This input is transferred from the mobile app interface to the mobile app controller. Next the mobile app controller validates login. In order to validate login the database connector queries login information and this information is returned to the mobile app controller. Next, the mobile app controller verifies login and returns a selection page to the mobile app interface. Next, the user selects reserve and the mobile app interface notifies the mobile app controller of this input. The mobile app controller then returns a reserve page to the mobile app interface. The user then selects a date, time and the duration of their requested reservation time. This information is then passes to the mobile app controller. The mobile app controller then checks availability. In order to check availability the database connector queries the database for availability and this information is returned to the mobile app controller. Next, the mobile app controller verifies the reservation, but the reservation is not available and the mobile app controller returns the no availability page to the mobile app interface.

Alternate Scenario:



UC-3 Elevator Entry (Main Success)

Goal: To enter the reservation zone of the parking garage.

The expert doer principle was implemented in the elevator entry use case. This principle was implemented because the elevator controller is responsible for most of the tasks within this system. The user input and output is implemented through the elevator controller and parking assignments are also queried and/or computed by the elevator controller. The elevator controller is responsible for most of these tasks since it interacts with the customer and is the most logical place for computations to be done. The database controller is responsible for all communication with the database in fetching/updating data and is also responsible for communicating with the elevator controller. This communication ensures proper flow of data and also allows for the one who knows the task best to implement the task.

Design Patterns:

Publisher-Subscriber: There are very few actors involved in this use case. Communication only occurs between the database and the elevator interface, and between the customer and the elevator screen - no more than one actor needs to be notified of any action at a time. Thus this design pattern would unnecessarily complicate our design.

Command: This use case implements the Command design pattern, for database queries. There is a single function called 'query', and its argument changes depending on the situation. Depending on information including the table to be queried, and the task at hand, the program builds a SQL command and executes it using the 'query' function. This design pattern improves our system because it opens our program to message passing - that is, in the case that the information needed to build a command is scattered across multiple classes, each class can add its own contribution to a common command string that will be used to execute a complete SQL command. It also decreases coupling, as it lessens the knowing requirements of each individual class. Further, it is a good demonstration of the expert doer principle, since the classes that know the key pieces of information are the ones that help assemble the command.

Decorator: The decorator pattern is not needed in this use case - there are no functionalities to be extended in this use case.

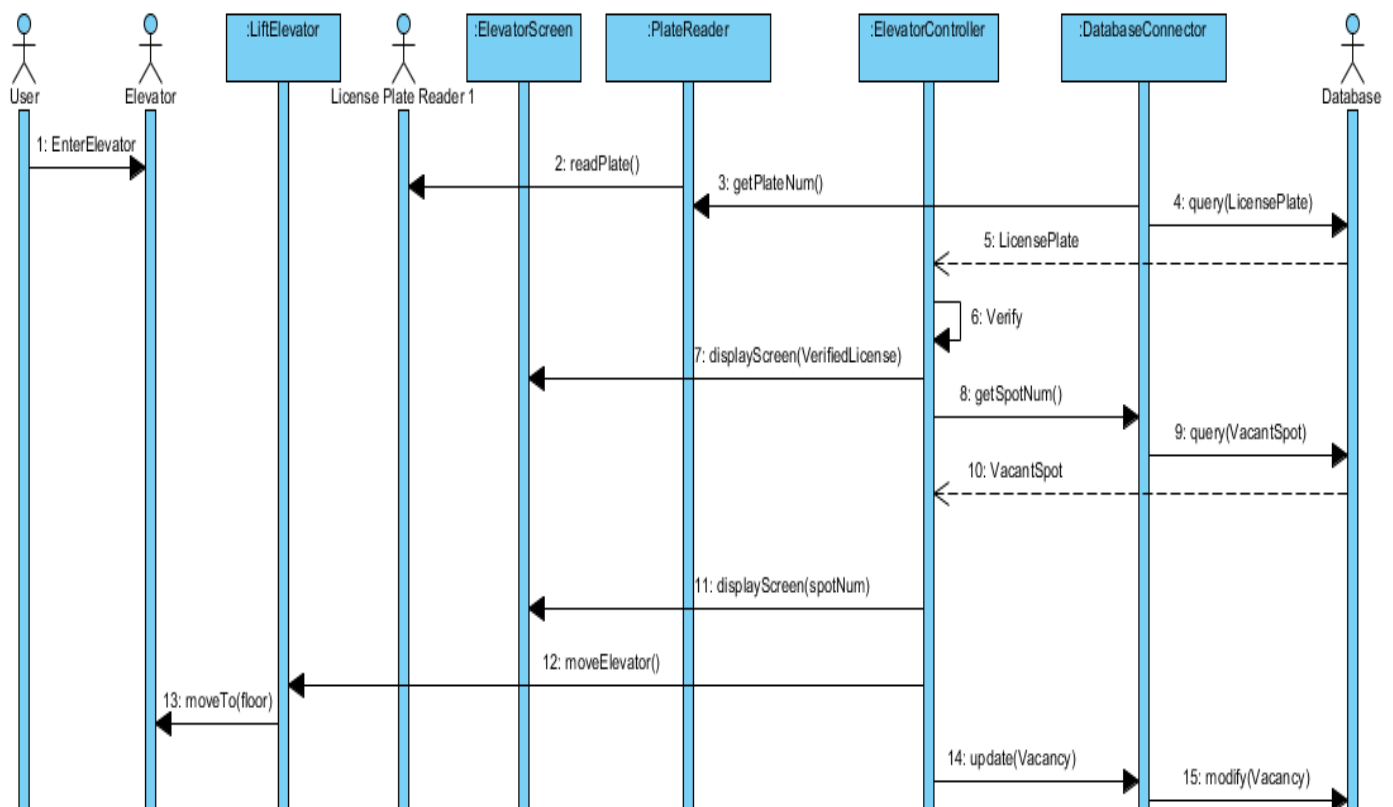
State: Because this use-case is linear by nature, implementing the state design pattern would unnecessarily complicate our design. Further, we do not expect the same event to occur again once our program has switched states while servicing any given customer.

Process (Main Success Scenario):

The user enters the elevator. The license plate reader 1 reads the license plate of the car. The ElevatorController retrieves the license plate number from license plate reader 1. The DatabaseConnector then queries the Database for a matching LicensePlate number to the one that was read. The Database returns the license plate to the ElevatorController and ElevatorController verifies the reservation. The ElevatorController then sends a request to display the message, "License Plate verified" on the ElevatorScreen. The ElevatorController sends a request to the DatabaseConnector to find a vacant parking spot. The

DatabaseConnector then queries the database for a vacant spot within the reserved zone. The Database finds the vacant spot associated with the user reservation and sends it to the ElevatorController. The ElevatorScreen will display to the user which parking spot number they are assigned to. The ElevatorController requests LiftElevator to move the elevator to the correct floor. The ElevatorController then tells the DatabaseConnector to update the Database with the occupied spot. The DatabaseConnector sends a request to modify the reservation vacancy within the database.

Main Success Scenario:

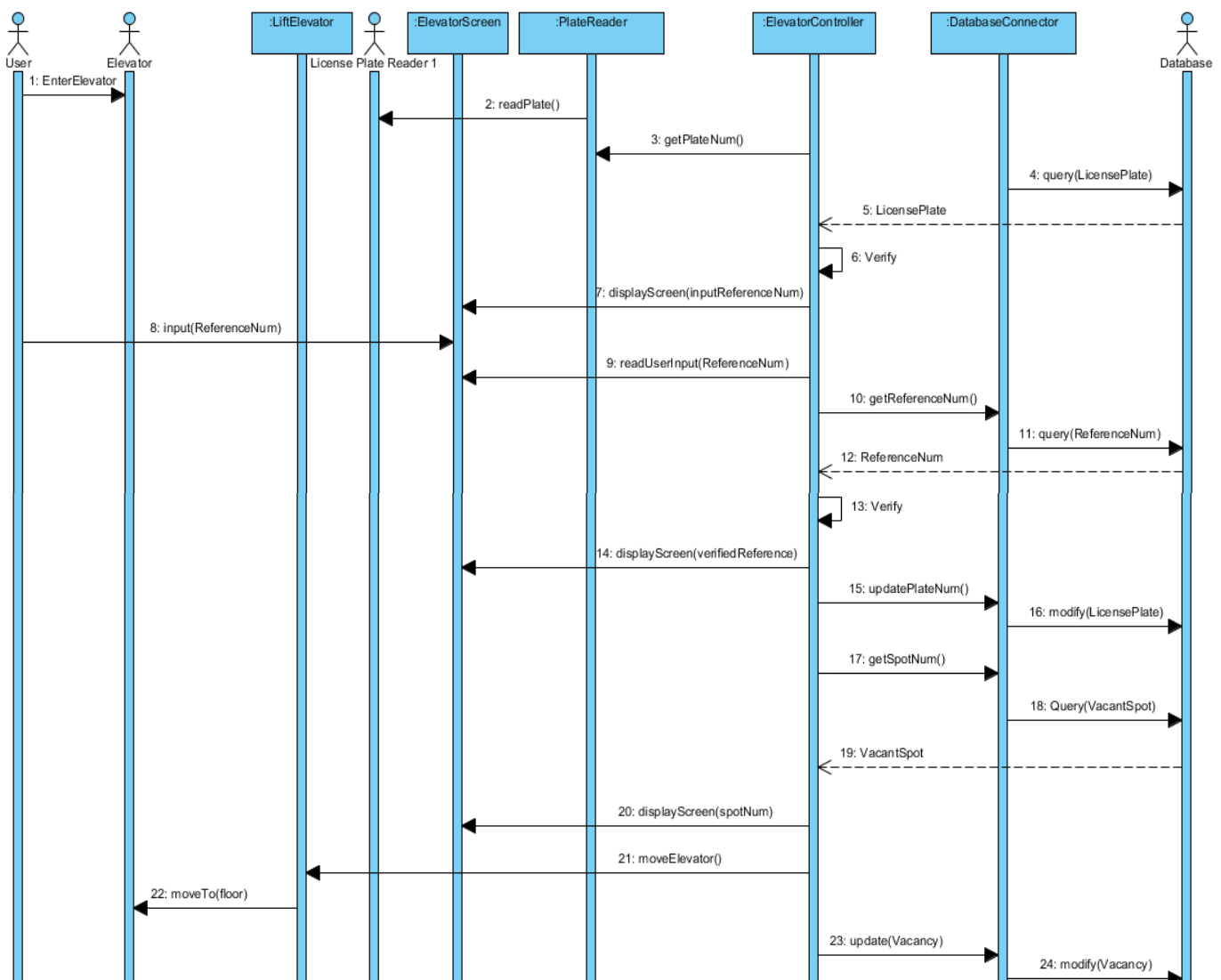


Process (Alternate Scenario 1):

The user enters the elevator. The license plate reader 1 reads the license plate of the car. The ElevatorController retrieves the license plate number from license plate reader 1. The DatabaseConnector then queries the Database for a matching LicensePlate number to the one that was read. In this case, the license plate number is not found within the Database. The ElevatorScreen prompts the user to input their reference number. The user inputs a reference number on the ElevatorScreen. The ElevatorController reads the reference number on the ElevatorScreen and sends a request to the DatabaseConnector to query the database for a matching reference number. The Database sends a matching reference number. The ElevatorController verifies it. The ElevatorScreen displays a message that the reference number

was verified. The Database then updates the reservation by matching the verified reference number with the license plate number. The ElevatorController sends a request to the DatabaseConnector to find a vacant parking spot. The DatabaseConnector then queries the database for a vacant spot within the reserved zone. The Database finds the vacant spot associated with the user reservation and sends it to the ElevatorController. The ElevatorScreen will display to the user which parking spot number they are assigned to. The ElevatorController requests LiftElevator to move the elevator to the correct floor. The ElevatorController then tells the DatabaseConnector to update the Database with the occupied spot. The DatabaseConnector sends a request to modify the reservation vacancy within the database.

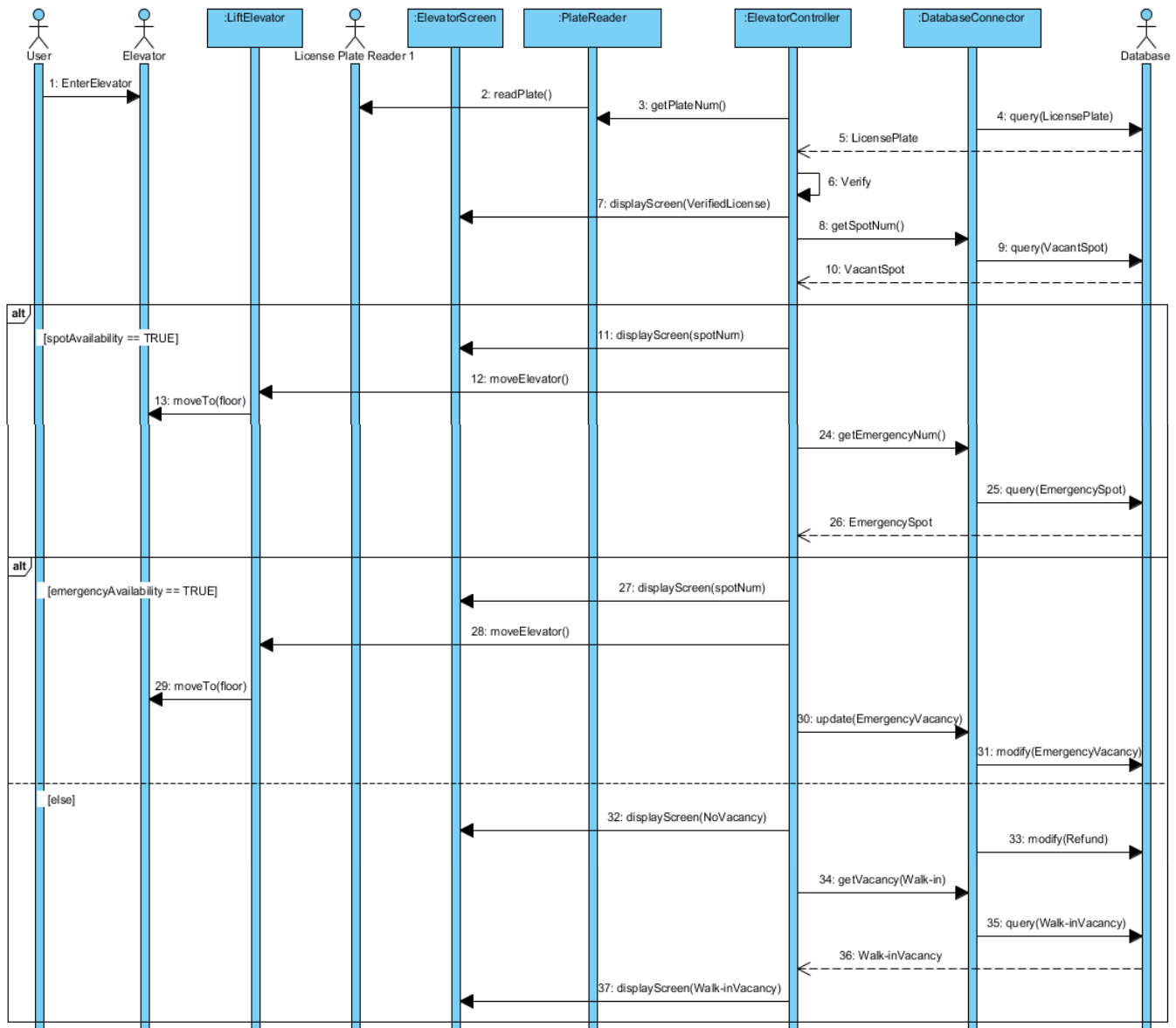
Alternate Scenario 1:



Process (Alternate Scenario 2/3/4):

The user enters the elevator. The license plate reader 1 reads the license plate of the car. The ElevatorController retrieves the license plate number from license plate reader 1. The DatabaseConnector then queries the Database for a matching LicensePlate number to the one that was read. The Database returns the license plate to the ElevatorController and ElevatorController verifies the reservation. The ElevatorController then sends a request to display the message, "License Plate verified" on the ElevatorScreen. The ElevatorController sends a request to the DatabaseConnector to find a vacant parking spot. The DatabaseConnector then queries the database for a vacant spot within the reserved zone. If the parking spot assigned to the user is vacant, then the system will follow the steps in the main success scenario. In the case of an overstaying customer, the ElevatorController will request for the database to search for any other vacant spots within the reserved zone. If one is found, the system will proceed like the main success scenario. Instead of the database updating the vacancy of the original assigned parking spot, the database will update with the new parking spot assigned to the user and . In the case, where an overstaying customer still occupies the user's spot and there are no other parking spots available within the reservation zone, the ElevatorController will request the database to search for a vacant parking spot from the list of Emergency parking spots. If one is found, the system will proceed like the main success scenario. Instead of the database updating the vacancy of the original assigned parking spot, the database will update by making the emergency spot occupied. In the case where no vacant emergency spot is found by the database, the ElevatorScreen will display to the user that there are no more vacant spots and that they will receive a full refund for their reservation. The user will also be suggested to park in the Walk-In zone if there are vacant spots. The DatabaseConnector will query the database to process the refund. The ElevatorController retrieves the vacancy for the Walk-In zone from the database and this is displayed to the user through the ElevatorScreen.

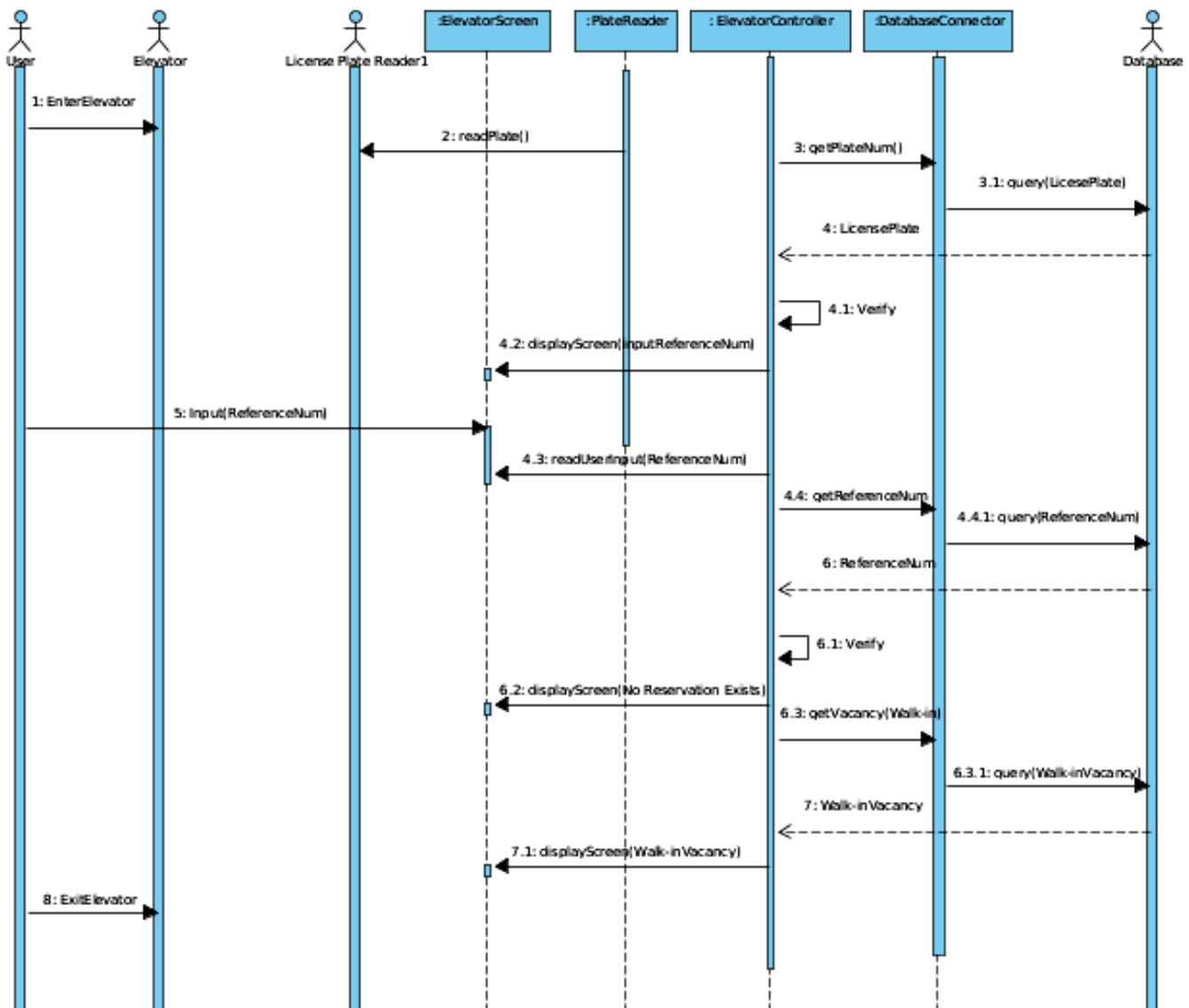
Alternate Scenario 2/3/4:



Process (Alternate Scenario 5):

The user enters the elevator. The license plate reader 1 reads the license plate of the car. The ElevatorController retrieves the license plate number from license plate reader 1. The DatabaseConnector then queries the Database for a matching LicensePlate number to the one that was read. The Database returns the license plate to the ElevatorController and ElevatorController verifies the reservation. The license plate is not verified and the user is prompted to enter a reference number through the ElevatorScreen. The user inputs the reference number and the ElevatorController retrieves this input and asks the DatabaseConnector to search the Database for a matching reference number. The reference number is not verified and the ElevatorScreen displays to the user that their reservation does not exist. The ElevatorController requests the Database for the vacancy of the Walk-In zone and this is displayed on the ElevatorScreen.

Alternate Scenario 5 (No Reservation):



UC-7: Reservation Management

Goal: To extend or cancel an existing reservation

Expert-doer principle states: that who knows should do the task. We decided to follow this principle for our model. DatabaseConnector is the main medium of communication between the system and the database. All requests to query/ update database go through the DatabaseConnector. Similarly, all user interactions are handled by the MobileAppInterface. By opting to follow this approach, we could not apply the low coupling principle to our design. The design closely follows the principle of high cohesion as each class has the responsibility to do one well defined job.

Design Patterns:

Publisher-Subscriber: There are very few actors involved in this use case. Communication only occurs between the user and mobile app, and between mobile app and database. The pub-sub design pattern is inappropriate in this case, and our system would be better off without it.

Command: This use case implements the Command design pattern, for database queries. There is a single function called 'query', and its argument changes depending on the situation. Depending on information including the table to be queried, and the task at hand, the program builds a SQL command and executes it using the 'query' function. This design pattern improves our system because it opens our program to message passing - that is, in the case that the information needed to build a command is scattered across multiple classes, each class can add its own contribution to a common command string that will be used to execute a complete SQL command. It also decreases coupling, as it lessens the knowing requirements of each individual class. Further, it is a good demonstration of the expert doer principle, since the classes that know the key pieces of information are the ones that help assemble the command.

Decorator: The decorator pattern is not needed in this use case - there are no functionalities to be extended in this use case.

State: Because there is a natural flow of progression in this use case, implementing the state design pattern would unnecessarily complicate our design. Further, all state diagrams in our design are relatively simple, and can be better-implemented without the added complexity of the state design pattern.

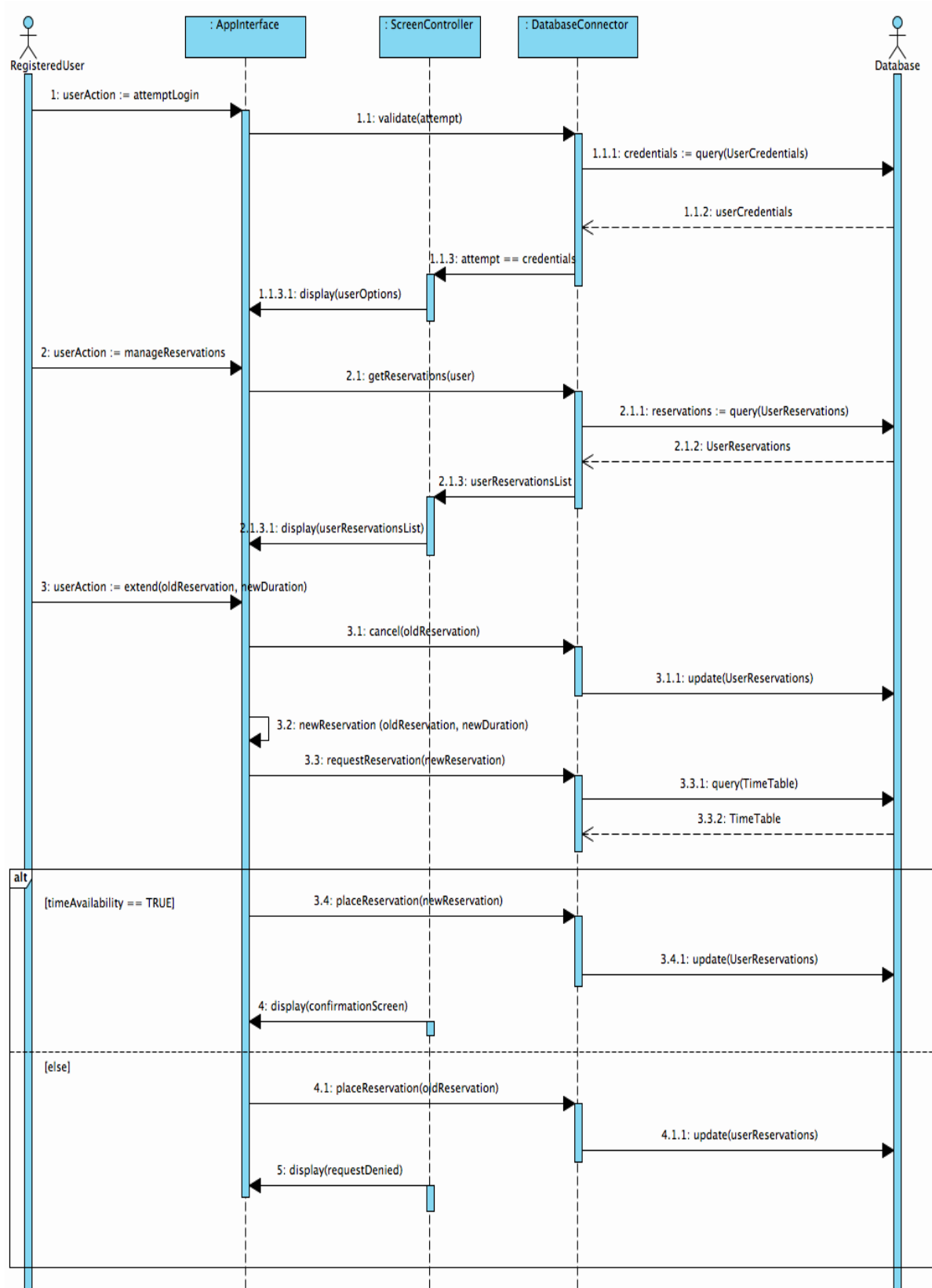
Proxy: Because the mobile app already restricts user input (values are selected from either

Process (Main Success Scenario/ Alternate Scenario 1):

The user logs in to their account using the mobile application. The login credentials are passed on to DatabaseConnector in order to be verified with credentials stored in database. Once verified, the user elects to manage existing reservations. Two options will be presented to the user: extend or cancel reservations.

If the user wants to extend an existing reservation, he/she must select the old reservation and provide the system with new duration. This causes the DatabaseConnector to cancel the old reservation and query database for new time duration. If a parking spot is available for the desired duration, it is assigned to the user and a confirmation screen is displayed to the user. On the other hand, if there is no availability for the desired time, the database is updated with the old reservation and user is notified.

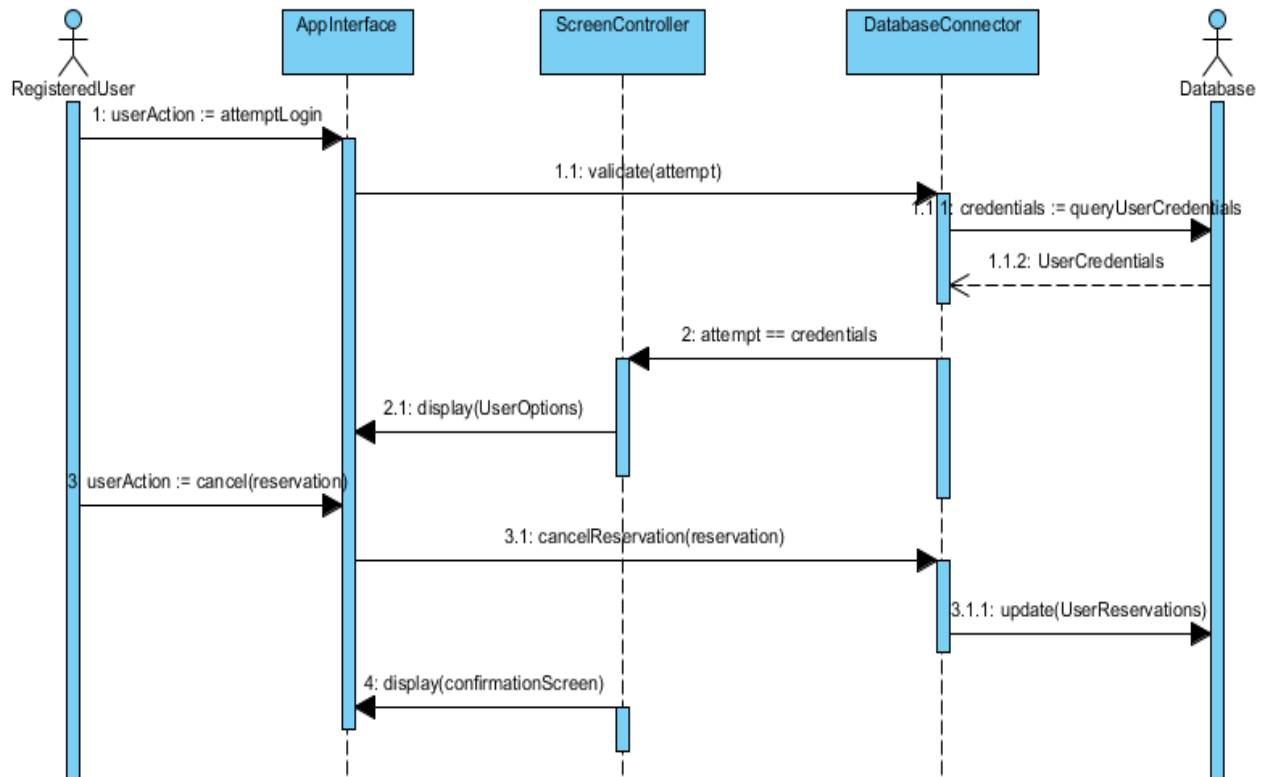
Main success scenario / Alternate scenario 1:



Process (Alternate Scenario 2):

If the user wants to cancel an existing reservation, the reservation info is passed to DatabaseConnector and the database is updated. A confirmation screen is then displayed to the user.

Alternate Scenario 2:



UC-9: Database Read

Goal: To obtain information about the garage by performing read operations on the database only.

The following design(s) satisfy the expert doer principle, as each class operates on incoming data and passes the processed information to the next. Each unique piece of information is only passed once as an input or output, to the class that has the remaining necessary information to operate on it. For example, the database connector only has one role - to send and retrieve information from the database. The action it takes depends on the processed information coming from the user or the controller.

Following this example, the designs also exhibit high cohesion, as the database only exists to communicate with the user, and the user interface only displays information to the user and receives inputs from the user for the rest of the program to process.

Design Patterns:

Publisher-Subscriber: There are very few actors involved in this use case. Communication only occurs between the database and the querying actor - no more than one actor needs to be notified of any action at a time. Thus this design pattern would unnecessarily complicate our design.

Command: This use case implements the Command design pattern, for database queries. There is a single function called 'query', and its argument changes depending on the situation. Depending on information including the table to be queried, and the task at hand, the program builds a SQL command and executes it using the 'query' function. This design pattern improves our system because it opens our program to message passing - that is, in the case that the information needed to build a command is scattered across multiple classes, each class can add its own contribution to a common command string that will be used to execute a complete SQL command. It also decreases coupling, as it lessens the knowing requirements of each individual class. Further, it is a good demonstration of the expert doer principle, since the classes that know the key pieces of information are the ones that help assemble the command.

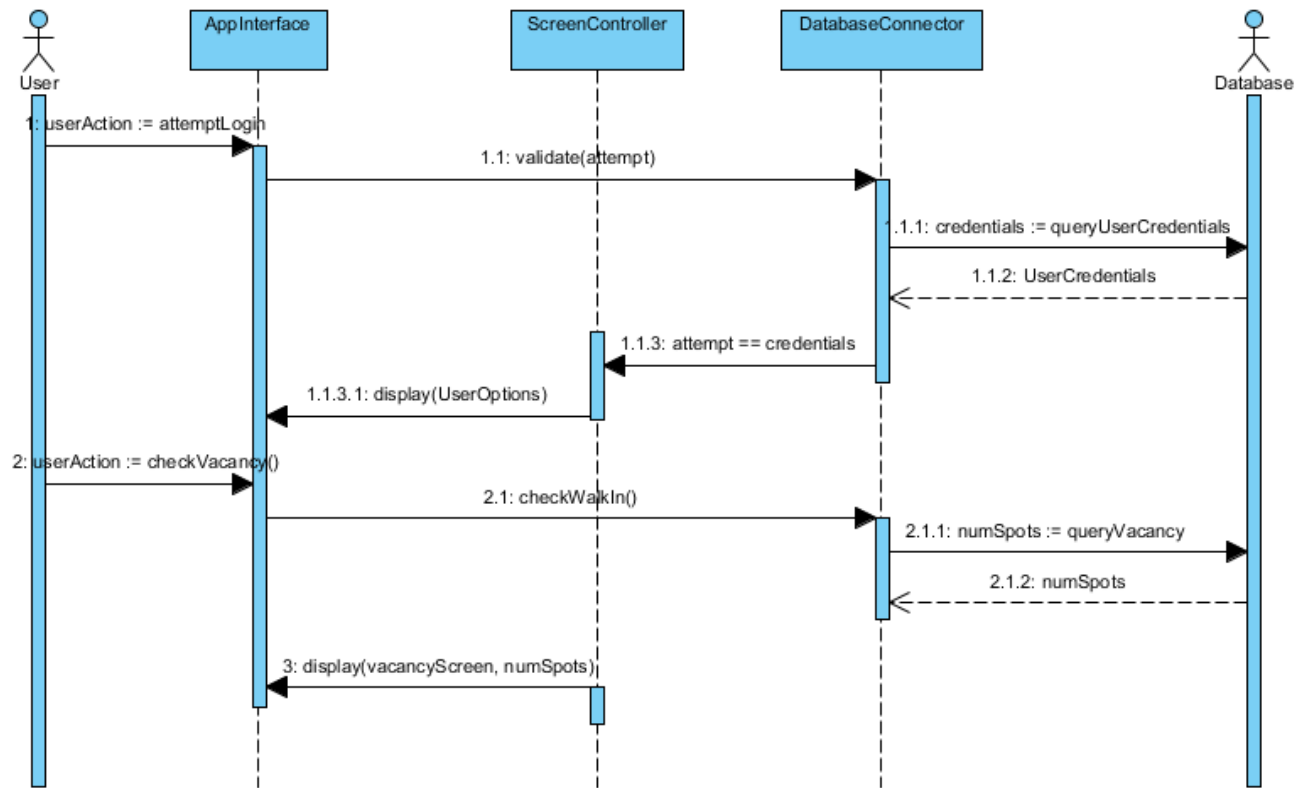
Decorator: The decorator pattern is not needed in this use case - there are no functionalities to be extended in this use case.

State: Because this use-case is very linear, implementing the state design pattern would unnecessarily complicate our design.

Process (Main Success Scenario):

The user attempts to log in to the mobile application. The log-in attempt is recorded and a connection with the database is established. After the mobile app receives the correct credentials from the database, the log-in attempt is compared against this information. A match has been found, so the User Options page will be displayed on-screen. The user chooses to check the vacancy of the garage. The mobile app retrieves this information from the database and displays it on the screen.

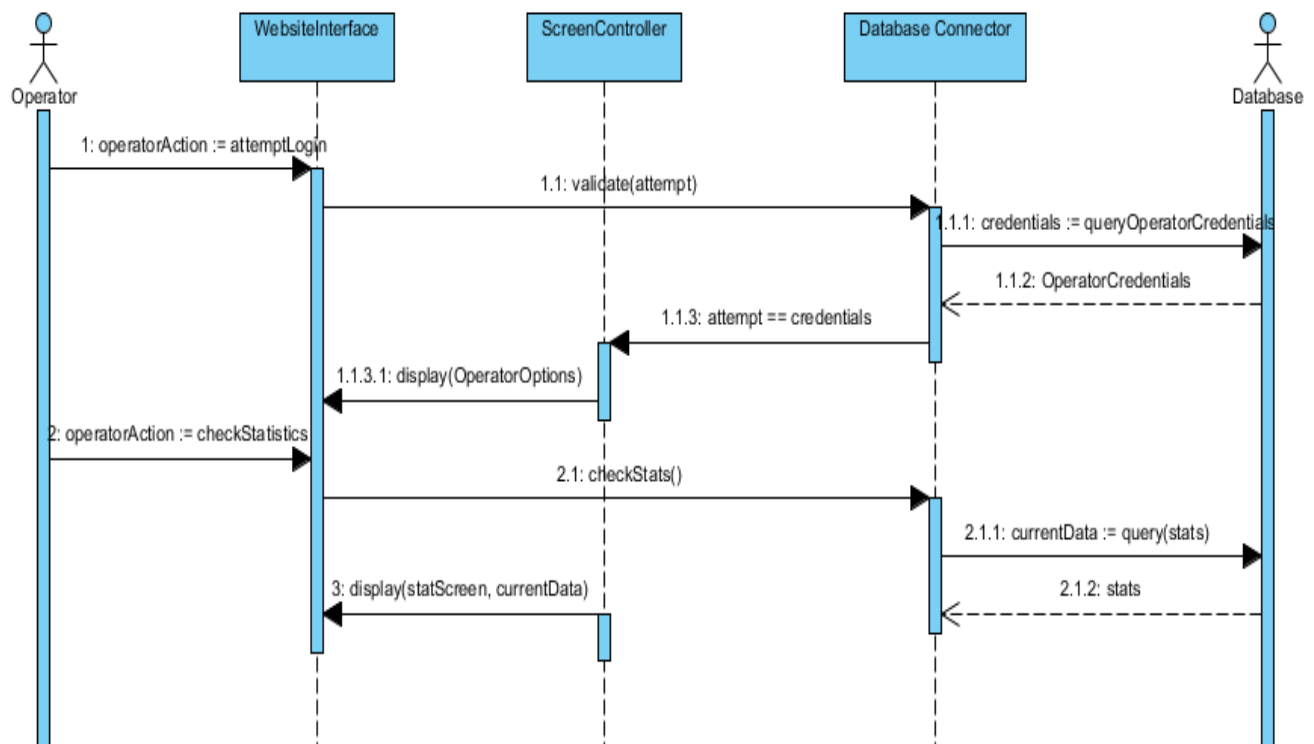
Main Success Scenario:



Process (Alternate Scenario):

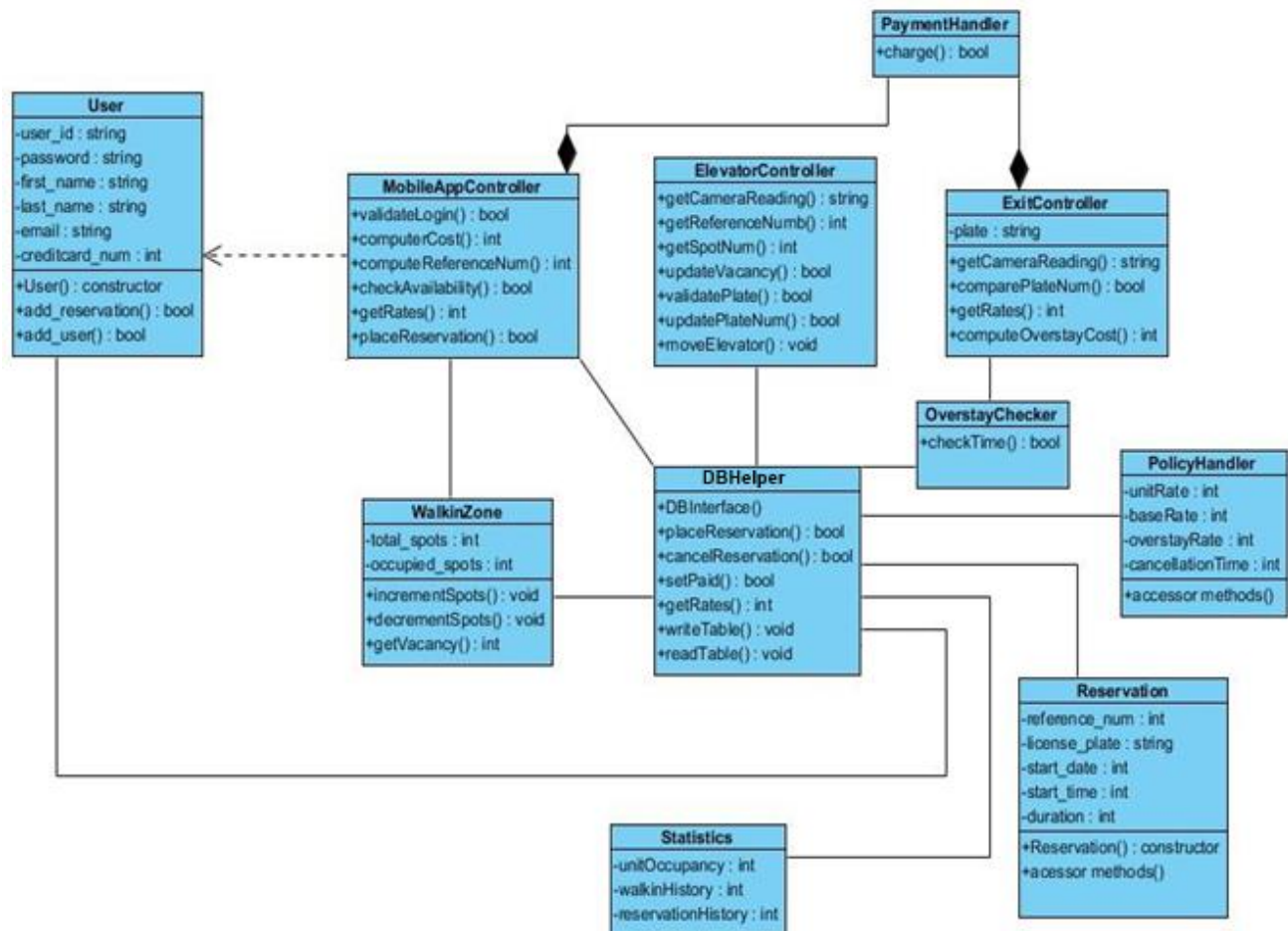
The garage operator attempts to log in to the manager's website. In a similar process to the main success scenario, the log-in attempt is recorded and a connection with the database is established. After the website receives the correct credentials from the database, the log-in attempt is compared against this information. A match has been found, so the Operator Options page will be displayed on-screen. The operator chooses to check statistics. The database retrieves all relevant data from the database and displays it on the screen.

Alternate Scenario:



8. Class Diagram and Interface Specification

Class Diagram



Design Patterns

- **Publish-Subscribe** - In our design, the *MobileAppController* has many need-to-know responsibilities. The controller needs to have direct communication with many other components of our system. Due to these restrictions, publish-subscribe design pattern doesn't apply to our system.
- **Command** - We have implemented the *command* design pattern in the class diagram above. The *MobileAppController* commands actions to other components of the system. The reason why we chose the controller as the command object is because it is the central player in the flow of events and, at the same time, it needs other objects to assist with accomplishing the task.
More specifically, this pattern is used for placing reservations through the database. It first creates a reservation command using parameters specified by the user. This contains all relevant values needed to identify the reservation - including start time and duration. The `placeReservation()` function executes this command and sets off a chain of database read/writes as directed by the command.
- **Decorator** - The decorator design pattern is suitable where there is one essential task and several optional tasks that may or may not occur together with the primary task. In our system design, there are no tasks that can be named optional. There are also tasks that need to perform alone, without any interaction with other tasks. Therefore, the decorator design pattern will not be implemented.
- **State** - Implementing the state design pattern would not visibly improve the existing design.
- **Proxy** - There appears to be no opportunity to use the proxy design pattern at the current stage of our system design.

Data Types and Operation Signatures

ReservationInfo

- Attributes:
 - reference_num: int [unique identifier of each reservation]
 - license_plate: string [intended license plate of car]
 - start_date: int [reservation start date]
 - start_time: int [reservation start time]
 - duration: int [duration of the requested reservation]
- Operations:
 - +ReservationInfo(): constructor
 - +accessor methods: all setters and getters

User

- Attributes:
 - user_id: string [unique username of a user]
 - password: string [password associated with the userid needed to access account]
 - first_name: string [first name of a user]
 - last_name: string [last name of a user]
 - email: string [an e-mail address of a user]
 - creditcard_num: int [credit card number used to make payments]
- Operations:
 - +User(): constructor
 - +bool add_reservation(array values): creates and links a reservation to user
 - +bool add_user(array values): creates user

PaymentHandler

- Operations:
 - +bool charge(creditcard, amount): charges the given amount on the given credit card, and returns a success value. If amount is negative, then a refund is issued.

DBHelper

- Operations
 - +DBHelper(): constructor
 - +bool placeReservation(Reservation): attempts to database with reservation. Returns false if unsuccessful
 - +bool cancelReservation(int reference): cancels the reservation corresponding to the given reference number, if it exists
 - +bool setPaid(reference):
 - +int getRates(): returns the current parking rates from the database.
 - void writeTable(string): <<overloaded>> internal function - performs actual database write to specified table
 - void readTable(string): retrieves requested information from database.

WalkinZone

- Attributes:
 - total_spots: int [total number of spots in walk-in zone]
 - occupied_spots: int [number of occupied spots in walk-in zone]
- Operations:
 - +void incrementSpots(): Increments spots available once a customer leaves
 - +void decrementSpots(): Decrements spots available once a customer enters
 - +int getVacancy() : Returns number of spots available at a particular time

PolicyHandler

- Attributes:
 - unitRate: int [parking rate per 30-minute interval]
 - baseRate: int [flat rate for using space]
 - overstayRate: int [rate for overstaying customers]
 - cancellationTime: int [amount of advance notice needed to cancel reservation]
- Operations:
 - + accessor methods: getters and setters

OverstayChecker

- Operations:
 - +bool checkTime(start_time, duration): compares actual vs expected parking duration & detects overstay

MobileAppController

- Operations:
 - +bool validateLogin(): validates login information of user
 - +int getRates(): gets current rates for cost computation.
 - +int computeCost(): computes cost of reservation considering current rates.
 - +int computeReferenceNum(): computes the reference number
 - +bool checkAvailability(): checks if the requested reservation is available
 - +bool placeReservation(): places reservation if exists, and user confirms

ElevatorController

- Operations:
 - +string getCameraReading(): gets license plate reading from camera
 - +int getReferenceNum(): reads inputted reference number from interface screen
 - +int getSpotNum(): gets assigned spot number for present user
 - +bool updateVacancy(): changes spot state from reserved to occupied
 - +bool validatePlate(): validates current license plate
 - +bool updatePlateNum(): updates plate number if not previously provided
 - +void moveElevator(): controls the physical elevator

ExitController

- Attributes:
 - plate: string [license plate of car at exit]
- Operations:
 - + string getCameraReading(): gets license plate reading from camera
 - + bool comparePlateNum(): matches vehicle plate number to a reservation and returns
 - +int getRates(): gets current rates for cost computation
 - +int computeOverstayCost(): computes cost if customer has overstayed reservation info.

Statistics

- Attributes:
 - unitOccupancy[]: int [stores occupancy data for each 30 minute period]
 - walkinHistory: int [stores walkin occupancy data for a given time period]
 - reservationHistory: int [stores reservation occupancy data for a given time period]
- Operations:
 - bool generateXLS(): creates excel spreadsheet, returns true on success
 - + computeStatistics(): computes statistics based on historic data
 - + accessor methods: setters and getters

Traceability Matrix

	User	Mobile App	Walk In Zone	Elevator Controller	Payment Handler	Exit controller	Overstay Checker	DB Helper	Policy Handler	ReservationInfo	Statistics
Personal Info	x	x			x			x		x	
Mobile App Screen	x										
MobileAppController	x		x		x			x		x	
Reservation Info	x							x			
Database Connector		x	x				x	x	x	x	x
PlateReader				x		x					
Elevator Screen				x				x			
UserInfo	x							x			
Elevator Controller				x				x			
Lift Device				x							
Reference Checker				x							
Exit Controller					x	x	x	x			
Parking Summary					x			x		x	

Class Evolution from Domain Concepts:

User: The user class was derived from multiple domain concepts including the Personal Info, Mobile App Screen , Mobile App Controller and UserInfo. Personal Info and UserInfo were the most essential part in determining the concept of the User. This class was derived in order to construct a user, when the user registers for an account. This constructor takes the user fields such as their name, credit card, and e-mail and constructs the user within the database. This class is then used for verification of the user on the Mobile App when the user logs in. Hence, this concept is derived from the Mobile App Controller and the Mobile App Screen.

Mobile App Controller: The domain concepts of Personal Info and Database Connector were used in designing the class Mobile App Controller. The Personal Info domain concept was used since it is associated with the different user fields such as the username and password along with other personal information. The acquisition of this information is an essential part for validating the identity of a user as well as placing a reservation. The Database Connector is also another essential concept used in designing the Mobile App Controller since the information needs to be pulled from the database via the use of the database connector in order to check availability of spots as well as computing the rates for a particular reservation.

Walk-in Zone: The concepts of Mobile App Controller and Database Connector were also used in designing the Walk in Zone class. This concept was used since the user can use the Mobile App in order to check the walk in zone vacancy at any given particular time. The Database Connector is also another concept used since data regarding the vacancy needs to be fetched as well as updated within the database.

Elevator Controller: The Elevator Controller class was derived from the Plate Reader, Elevator Screen, Elevator Controller, Lift Device and Reference Checker. The Plate Reader is an essential domain concept used in the elevator controller since the plate number needs to be read in order for it to be compared and verified within the database. Further, the concept of Elevator Screen needs to be used since the user input/output need to be received by the Elevator Controller class in the case where the license plate cannot be verified. Next, the domain concept of Elevator Controller was used since all the communication and computations were performed within this domain concept. This concept is essential since the major task of the Elevator Controller class is to assign the user a spot and also to communicate with other parts of the software to control the elevator. Moving the elevator to the correct floor is an essential part of the class and therefore the Lift Device domain concept was used within this class. Next, the concept of Reference Checker was used within the class as well since this domain concept is responsible for keeping track of the number of attempts the user has used while entering the reference number.

Payment Handler: This class uses the domain concepts of Personal Info, Mobile App Controller, Exit Controller, and Parking Summary. The main task of the Payment Handler is to charge the customers' credit card upon reservation and overstay. Also, another task is to make refunds when a customer cancels a particular reservation. The domain concept of Personal Info is essential since the credit card info needs to be pulled from the database in order to proceed with the reservation transactions. The domain concept of Mobile App Controller is also used since, the Mobile App is the medium through which the reservations occur, which eventually leads to payments. Further, the domain concept of Exit Controller is used since this concept is responsible for detecting overstay and charging the customer for their overstayed amount. Finally, the domain concept of Parking Summary is used since this is what allows the customer to view their transaction history for reservations made in the past.

Exit Controller: The domain concepts of Plate Reader and Exit Controller were used in implementing the Exit Controller class. The Plate Reader domain concept was used since the

Exit Controller class needs to detect overstays, and this can only be done through comparing the license plate number. The domain concept of Exit Controller was also used in implementing this class since the database needs to be updated with the vacancy of the given spot. Further this concept is also related to detecting overstay and charging the customer for the overstaying period.

Overstay Checker: The Overstay checker is derived from the domain concepts of the Exit Controller and Database Connector. The Overstay Checker class is only responsible for detecting overstay. In order to determine overstay, the inputs of plate number need to be given which is provided by the Exit Controller domain concept and the time and duration for a reservation also need to be fetched which is derived through the Database Connector domain concept.

DBHelper: The DBHelper class is a class derived from the greatest amount of domain concepts since the database stores all the data and the data needs to be pulled constantly for any computation to be achieved. The domain concepts of Personal Info, Mobile App Controller, Reservation Info, Database Connector, Elevator Screen, User Info, Elevator Controller, Exit Controller and Parking Summary were used in designing this class. The Personal Info domain concept, Reservation Info concept, User Info concept and the Parking Summary concept were used for their storage purposes within the database. Any sort of registration, reservation or change in personal information needs to be updated within the database and these concepts allow us to determine all the different types of data that needs to be stored within the database. The Mobile App Controller, Exit Controller and Elevator Controller are also used in this class since these controller systems perform specific computations within the software and these need to interact with the database in order for these computations to occur. Next, the elevator Screen is also a part of the DBHelper class since specific data pulled from the database such as the parking spot number need to be displayed on such a medium. Finally, the Database Connector is the most important concept used since queries and updates within the database are derived from this domain concept. This concept allows us to implement all the communication tasks of the database.

Policy Handler: The Policy Handler class is derived from the Database Connector concept. The policy handler is only used to update policies within the database. These policies might include updating parking rates as well as updating cancellation policies. Since all these policies will have to be updated in the database, the Database Connector domain concept allows us to derive communication concepts which allow for such updates.

Reservation: The reservation class is derived from the domain concepts of Personal Info, MobileAppController, Database Connector and Parking Summary. When attempting to place a reservation the domain concept of MobileAppController is essential since the Mobile App is the only medium through which a reservation can be placed. Since the Mobile App is used the domain concept of Personal Info is also essential since identity needs to be verified and credit card needs to be charged when a reservation is placed. Since the act of placing a reservation is being implemented, the Database Connector is also another essential concept since this allows

us to derive the communication links that need to be used in order to update the database with the placed reservation. Finally, the domain concept of Parking Summary is used since this allows us to derive information that needs to be updated in the database for the customer to access if needed.

Statistics: The Statistics class hails from the Database Connector domain concept. Its function is to compute diagnostics for the garage operator to view. As such, it requires constant access to the current state of the database. If the data is outdated, the computed diagnostics would be inaccurate thus rendering the entire management module useless and even detrimental.

OCL Contracts

Class User

Invariant: User is not registered in database

```
context DBHelper d;  
d->readTable(All_Users)->exists(user | user == User.user_id) == false;
```

Preconditions:

```
self->forAll(var | var == null) == true;
```

Postconditions:

```
self->forAll(var | var != null) == true;
```

Class ReservationInfo

Invariant: User is logged in and has credit card

```
User.password == valid && User.creditcard_num != null;
```

Preconditions:

```
self->forAll(var | var == null) == true;
```

Postconditions:

```
self.reference_num > 0;  
self.start_date > 0;  
self.start_time > 0;  
self.duration > 0;
```

Class PaymentHandler

Invariant: User is logged in and has credit card

```
User.password == valid && User.creditcard_num != null;
```

Preconditions:

```
Self->forAll(var | var == null) == true;
```

Postconditions:

```
DBHelper.setPaid(ReservationInfo.getReferenceNum)==true;
```

Class DBHelper

Invariant: database and user's IP addresses remain unchanged

Preconditions: No connection made

self.isConnected == false;

Postconditions: Connection is made

self.isConnected == true;

Class PolicyHandler

Invariant: Manager is logged in to the system

User.user_id.equals("manager") == true;

Preconditions: Rates/policies are stale

Postconditions: Rates/policies are updated according to Manager's vision.

Class Statistics

Invariant: Manager is logged in to the system

User.user_id.equals("manager") == true;

Preconditions:

Postconditions:

self.generateXLS() == true;

9. Systems Architecture and Systems Design

Architectural Styles

Our system closely follows the Event Driven Architecture (EDA). Event Driven Architecture is a framework that orchestrates behavior around the production, detection and consumption of events as well as responses they evoke. Our software architecture is based on event of a customer requesting to park in the garage.

There are multiple events that trigger responses in our reservation system. The reservation system starts functioning when a user creates a reservation and changes states based on user-actions. The following table depicts the possible states of an existing reservation and the possible events that can change occur over its lifetime:

Event	Pre-state	Post-state
Reserve	none	upcoming
Cancel	upcoming	cancelled
Extend	upcoming, active	upcoming, active
Overstay	active	overstay
Completed	upcoming, active, overstay	completed

State description:

Upcoming: A reservation is considered 'upcoming' when its starting time is later than the current time. [reservation.startTime > Time.time]

Cancelled: A reservation is 'cancelled' if it was placed but eventually removed at the request of the one who placed it. [reservation.cancel()]

Active: A reservation is marked 'active' during the time that the user holding the reservation is able to park in the garage. [reservation.startTime < currentTime < reservation.startTime + reservation.duration]

Overstay: A reservation is an 'overstay' if the reservation duration is over but the customer has not left the garage. [reservation.startTime+reservation.Duration < currentTime && userInfo.isParked == TRUE]

The diagram illustrates the following packages and their components:

- User**: A package with no visible components.
- Mobile App**:
 - Controller**: Contains `SendToDatabase()`, `AccessRates()`, `AccessGarageAvailability()`, `ChargeUser()`, and `UpdateDatabase()`.
 - User Interface**: Contains `RetrieveUserInfo()`, `DisplayPaymentAmount()`, and `SendtoController()`.
- Server**:
 - Tables**: Contains `Store:`, `User Info`, `Reservation Availability`, `Rates`, `Reservations`, and `Statistics`.
- Website**:
 - Contains `ChangeRates()`, `DetermineProfit()`, and `DisplayStatisticalData()`.
- Garage**:
 - Controller**: Contains `DetermineOverstay()`, `ChargeUser()`, and `UpdateDatabase()`.
 - Exit Interface**: Contains `RetrieveLicensePlate()` and `SendtoController()`.
- Elevator**:
 - Controller**: Contains `SendtoDatabase()`, `UpdateDatabase()`, and `RetrieveParkingSpot()`.
 - Elevator Interface**: Contains `RetrieveReferenceNumber()`, `RetrieveLicensePlate()`, `DisplayParkingSpot()`, and `SendtoController()`.

Relationships and Realizations:

- Realization** (dashed line with open triangle):
 - `Controller` realizes `User Interface` within the **Mobile App** package.
 - `Controller` realizes `Exit Interface` within the **Garage** package.
 - `Controller` realizes `Elevator Interface` within the **Elevator** package.
- Access** (dashed line with open arrow and `<<access>>`):
 - User** has access to the **Garage** package.
 - Mobile App** has access to the **Server** package.
 - Website** has access to the **Server** package.

The Mobile app is responsible for retrieving the user's request, sending the registration and/or reservation information entered by the user to the database, and sending appropriate response back to the user. The Mobile app needs to access the reservation information stored on the database; it needs to update the database with the registration and reservation information of the user.

The elevator interface depends on the server for confirming reservations. The exit interface depends on the server for determining & charging for overstay as well as for updating the database with parking spot availability.

Mapping Subsystems to Hardware

The system can be broken into three main parts, a mobile app, a server and a website.

Mobile App

The mobile app is where the majority of the reservation interaction occurs. It accepts input from a client-side system that allows reservations to be created and managed. It also allows users to create accounts and edit personal information. The mobile application will access the server to set up an account, make payment, make reservation and edit account information. The mobile application will be very light as we do not want to store anything on the phone because storage is very limited, therefore majority of the information will be transferred to database.

Server

The server maintains a database that stores all user information, reservation information, statistics and rates. The user information includes email, password, license plate, credit card information etc. Reservation information includes the date of reservation, duration of reservation and start time. The statistics include past occupancy information of the garage, per time unit (we set this as 30 minutes), as well as generated revenue. Rates include anything that management will need access to such as the price per hour. Only the rates and user information will be modifiable by outside sources.

Website

The website has two main functionalities. The first is to make changes to specific details in the database such as the rates per hour and how much to charge for overstay. The second use is to display a read only display of the statistical data from the database such as gross profit per day or occupancy per day. The website is one of the few ways to change information in the database and is strictly accessible only by management and not open to any other users.

Persistent Data Storage

Enumerated below are seven database tables we will be using in this design. These tables hold their respective attributes pertaining to the user and system information.

Database Tables			
Field	Type	Extra	Links
<i>userInfo</i>			
email	string		
license	string		
username	string		
password	string		
customer_id	int	auto increment	
<i>creditcard</i>			
cc_number	int		
customer_id	int		customer->customer_id
cc_type	enum('Visa', 'Mastercard', 'American Express')		
cc_id	int	auto increment	
<i>car</i>			
car_id	int	auto increment	
car_name	string		
license_plate	string		
customer_id	int		customer->customer_id

<i>reserve_zone</i>			
spot_id	int		
status	enum('vacant', occupied', 'reserved')	default:'vacant'	
<i>walkin_zone</i>			
counter	int		
vacancy	int	auto increment/decrement	
entrance	datetime		
departure	datetime		
<i>reservationInfo</i>			
reservation_id	int	auto increment	
start	datetime		
end	datetime		
customer_id	int		customer- >customer_id
cc_id	int		creditcard->cc_id
spot_id	int		reserve_zone- >spot_id
car_id	int		car->car_id
arrive	datetime		
depart	datetime		
<i>rates</i>			
rate_id	int		
reserve_rate	int		
walkin-rate	int		
overstay_rate	int		

Our design features several tables that will be stored in the database:

- “userInfo” stores all important information about garage customers
- “creditcard” stores the credit card information provided by customers
- “car” stores information about customer’s car in order to verify customer’s identity upon entry into the garage
- “reserve_zone” stores information for each parking spot in the reserve section of the garage
- “walkin_zone” keeps track of walkin zone’s vacancy by employing a counter to increment and decrement as customers drive in and out of the walkin zone
- “reservation_info” stores all relevant information about a customer’s reservation. This tables links with several other tables such as “creditcard”, “userInfo”, “reserve_zone”, and “car”.
- “rates” stores the current unit rates being used by the garage management.

Network Protocol

Our system runs on multiple machines: A mobile application, the manager’s website, and a number of systems that are active within the garage (Elevator/Exit System). In order to ensure data consistency, each program will perform reads and writes to a common database, hosted on a remote server. We adopt this protocol because database read/writes are a standard operation across all machines and programming languages. Thus, this data can be understood in the same way by all active components of our system.

Global Control Flow

- *Execution Order* - Our software is event driven, therefore it waits in a loop waiting for events created by users.
- *Time Dependency* - Our system does not rely on any timers.
- *Concurrency* - Our system does not use multiple threads.

Hardware Requirements

Touch Screen - Allows the customer to enter information at the elevator and displays the parking spot. This display should be around 800x600 resolution because it is small and close to the user. It should also be a color display

Server - Contains all of the user data and reservation data. There only needs to be one server to handle all of the garage reservations and user information. The server can start off small but would have to grow because we do not delete any of our user information so it would be safe to have 8GB of server space for the long run.

Cameras - Used to identify the incoming and exiting vehicles license plates. These cameras need to be of high quality as it needs to take a picture of a moving vehicle and read it’s license plate.

Vacancy Display - Displays the vacancy outside of the garage to notify if the garage is vacant or occupied. This sign should be of much greater resolution with a minimum of 1280x720. This does not need to be a color display but would attract more attention if it was.

Cell Phone - To access the mobile application, the user will be required to have a cell phone. The cell phone needs to be a smart phone with access to the internet and also running on an Android operating system.

10. Algorithms and Data Structures

Scheduling Reservation

Our design implements a complex algorithm that is used to schedule reservations for garage customers. Customer provides a desired start time and duration for which they require a parking spot reserved. Once the information is submitted, the database is queried to check whether the reservation can be made. Note that the customer will not be allowed to pick a specific parking spot and the time of creating a reservation so only the algorithm will only query database for available times.

The algorithm will mainly consist of several database queries. The database contains a *timetable* for each day. This timetable is broken down into 30-minute intervals for each parking spot in the garage. The desired reservation described by the customer will be compared with the *timetable* entries for each parking spot until a vacant spot is found.

The breakdown of the algorithm is as follows:

Function Prototype:

```
scheduleReservation(Time startTime, int duration, bool isAlternative = FALSE)
//isAlternative defaults to false - external calls to this function must set
isAlternative to False or only use 2 parameters.
```

1. The algorithm computes the number of 30-minute intervals in the requested reservation. For example, a 90-minute reservation has three 30-minute intervals, while a 91-minute reservation would span four 30-minute intervals.
2. Starting from parking spot 1, the program searches the timetable from the entry at [reservation.startTime] for [numIntervals] entries for vacancy.
3. If a spot is found with all requested timetable entries vacant, then the ID of that parking spot is returned.
4. If no spot was found after all parking spots were checked, the program returns spotID 0 - meaning it doesn't exist.

- ```
for(spotID = 1:end)

 for(entry = 0 : numIntervals)
 if(! timetable.isVacant(spotID, reservation.startTime +
 entry))
 break;
 if(entry == numIntervals)
 return spotID; // if inner for-loop has completed, then
 spot was found! Return this spot number.
```

5. If no spot was found, the algorithm will make one recursive call to itself, attempting to find alternate reservations with the same duration but starting time within 30 minutes. If this fails, no more attempts will be made and the algorithm will return 0.

*Description of return values:* Spots found for alternative times are indicated by adding an offset in the 4th digit. For example:

- If spot 130 is found vacant for the requested duration, the program will return 130.
- If spot 130 is found vacant for a reservation starting 30 minutes earlier, the return value will be 1130.
- If spot 130 is found vacant for a reservation starting 30 minutes later, the return value will be 2130.
- If no spot is found for any of these cases, the return value of the function will be 2000.
  - Note: while the algorithm returns 0 if no spot is found, the function's final return value will be 2000 if no spot OR alternative spot has been found.

- `if(spotID == end) //if outer for-loop has completed, then spot was not found`

```
 if(isAlternative)
 return 0; //if this was a search for an alternative
 time, do not attempt to search for more spots.
 else //if not, then search again.
 isAlternative = TRUE;
 int returnVal = 1000 + scheduleReservation(startTime-
 30, duration, isAlternative);
 if(returnVal == 1000)
 returnVal = 2000 +
 scheduleReservation(startTime+30, duration,
 isAlternative);
 return returnVal;

 end function
```

## **Data Structures**

Main storage of all our data will be in a database. By using technology such as MySQL we will be quickly able to access and modify all our data. Consequently, most of our algorithms will be applied to data stored in database using Structured Query Language (SQL).

Our system does not use complicated data structures such as hash tables or trees. Instead, we utilize arrays whenever database queries pull multiple entries. We chose arrays due to its simplicity and efficiency for our design.

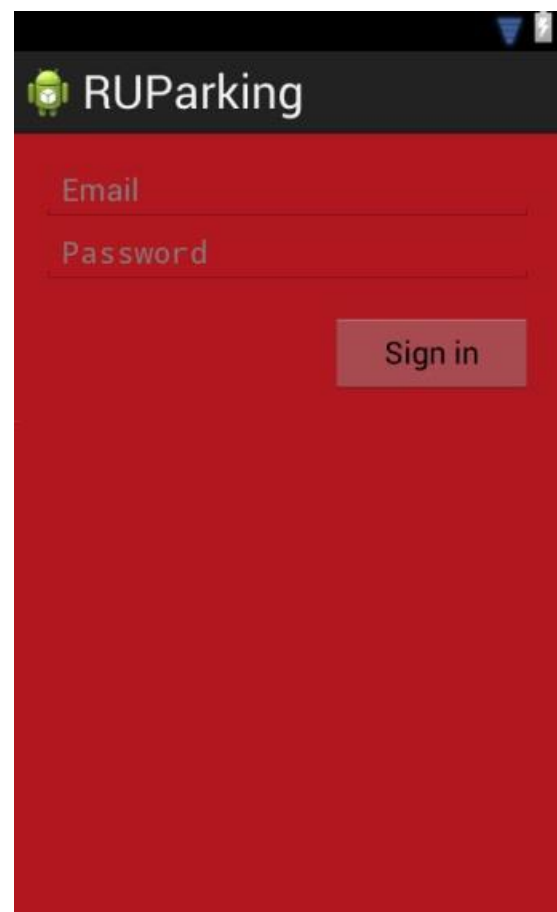
When querying the database, we will sometimes need to fetch tables of entries for Reference Numbers, License Plate numbers, Login Information, Parking Spot Numbers. We will need to store these entries in an array to be able to retrieve specific entries later on. We will use the array and not a linked list, because we will need to perform frequent search operations and we won't be performing insertion and deletion operations.

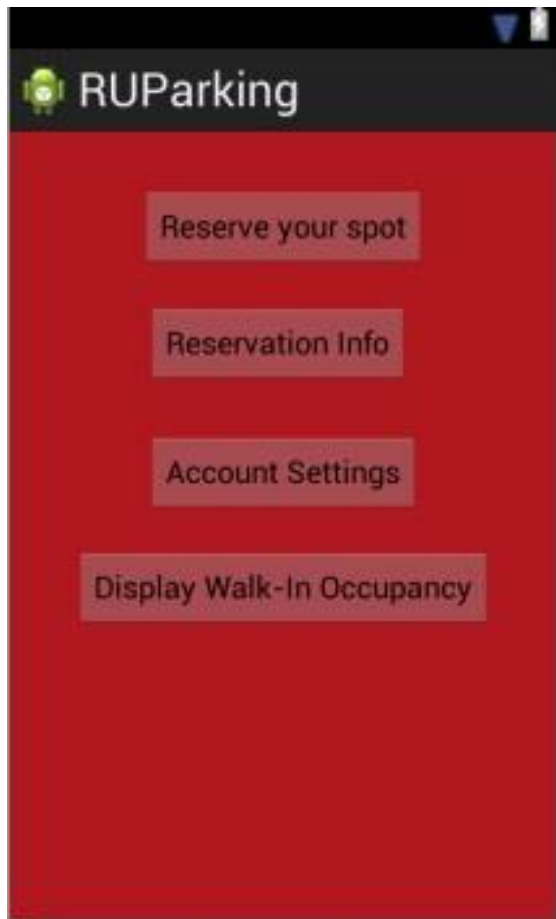
## 11. User Interface Design and Implementation

### User Interface Design and Implementation

#### *Mobile App*

The general mock-up screen was used to implement the RUParking Mobile App. The mock-up screen was made initially to be user friendly, and therefore the actual screens were made to be very much like the mock-up screen. No significant changes were made from the initial mock-up implementation. Shown below are the home screen, login page, registered user menu, account settings screen, and reservation placement screen.







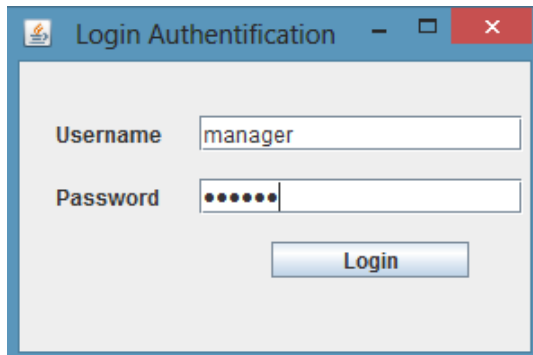
The screenshot shows the RUParking application interface on an Android device. The app has a black header bar with a green Android robot icon and the title "RUParking". Below the header, the background is red. The interface contains three input fields with labels and placeholder text:

- Enter Date:** with placeholder text "mm/dd/yy"
- Enter Start Time:** with placeholder text "hh:mm"
- Enter Duration:** with placeholder text "hh:mm"

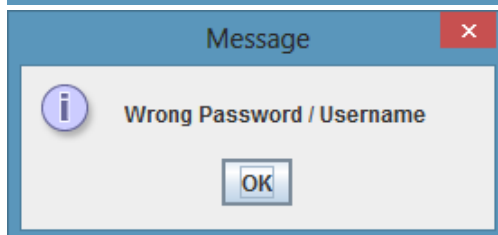
### ***Management Interface***

The management interface was created according to the customer specifications. Our current design is extremely easy to use, as it does not require any more clicks or user actions than is necessary.

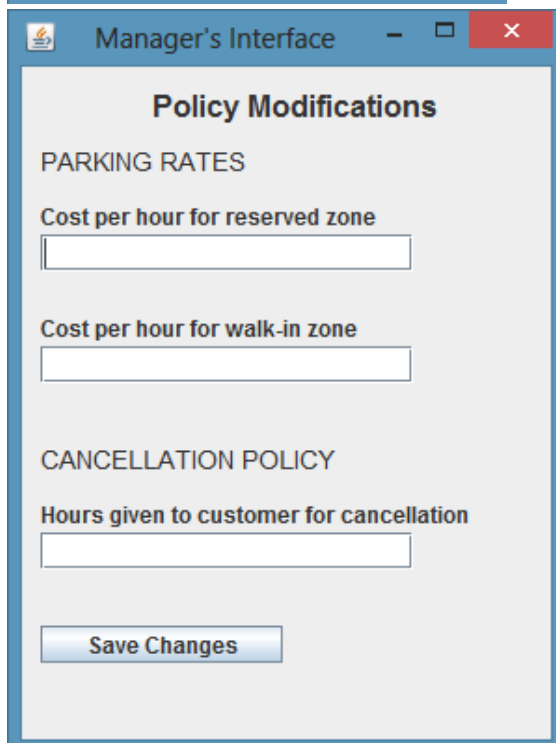
The first page is a prompt for a username-password combination. If the entry is invalid, an error message is displayed. If the login is valid, then the manager's interface opens and the user is granted access to the rates. These changes will be reflected in the database.



A screenshot of a 'Login Authentication' window. It features a title bar with a standard Windows icon, a minus sign, a maximize button, and a close button. The window contains two input fields: 'Username' with the text 'manager' and 'Password' with masked characters (dots). Below the password field is a 'Login' button.



A screenshot of a 'Message' window. It has a title bar with a close button. The message area contains an information icon (i) and the text 'Wrong Password / Username'. Below the message is an 'OK' button.



A screenshot of the 'Manager's Interface' window. The title bar includes a standard icon, a minus sign, a maximize button, and a close button. The main content area is titled 'Policy Modifications' and is divided into two sections. The first section, 'PARKING RATES', contains two input fields: 'Cost per hour for reserved zone' and 'Cost per hour for walk-in zone'. The second section, 'CANCELLATION POLICY', contains one input field: 'Hours given to customer for cancellation'. At the bottom of the window is a 'Save Changes' button.

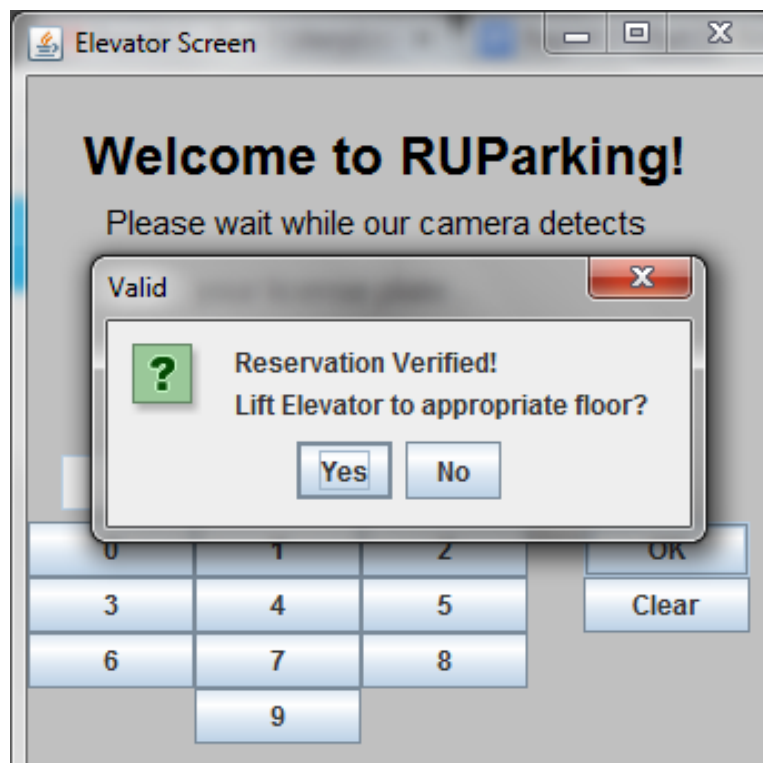
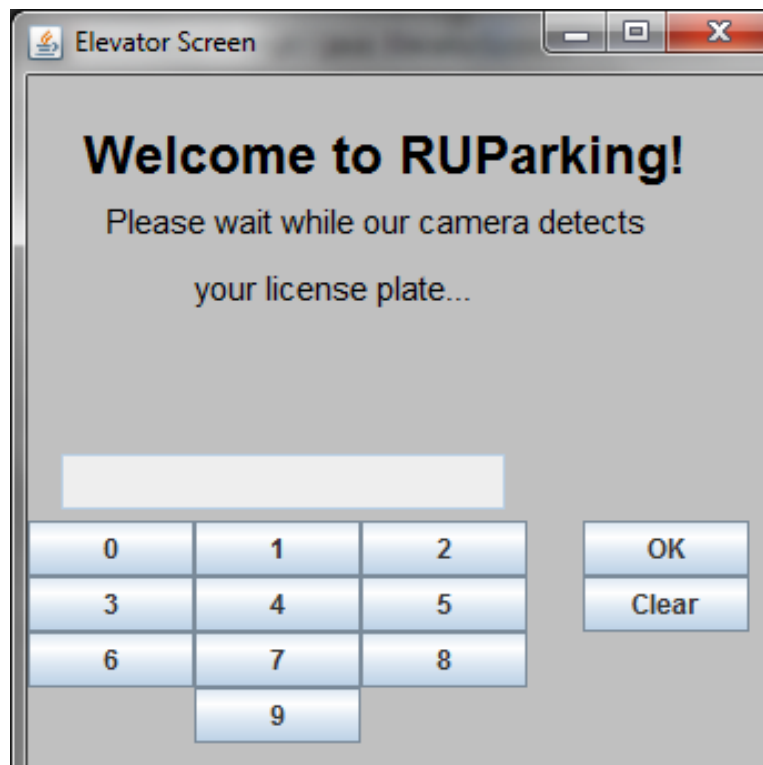
### ***Elevator Interface***

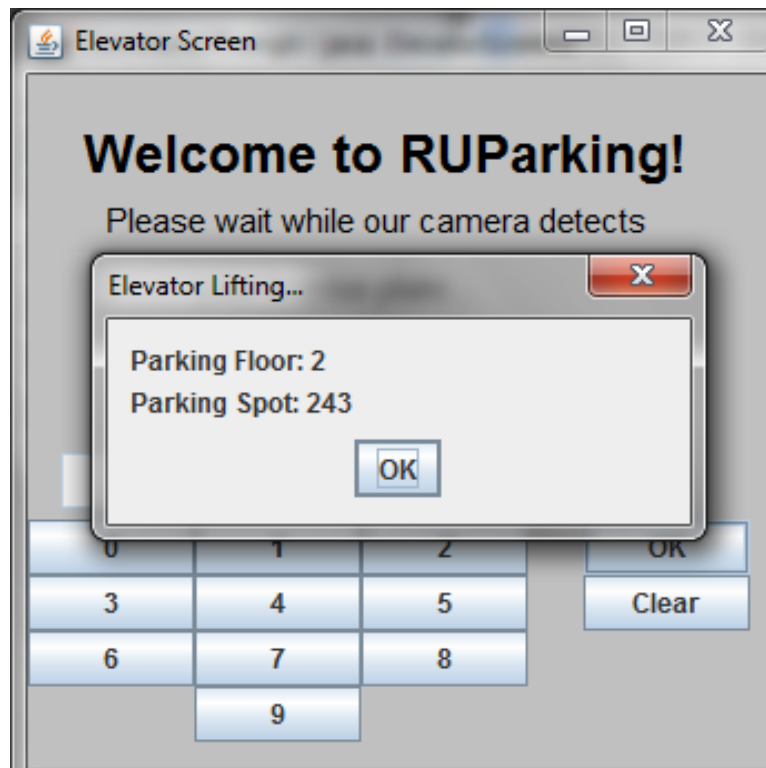
The changes to the initial screen mock ups of the Elevator Screen include:

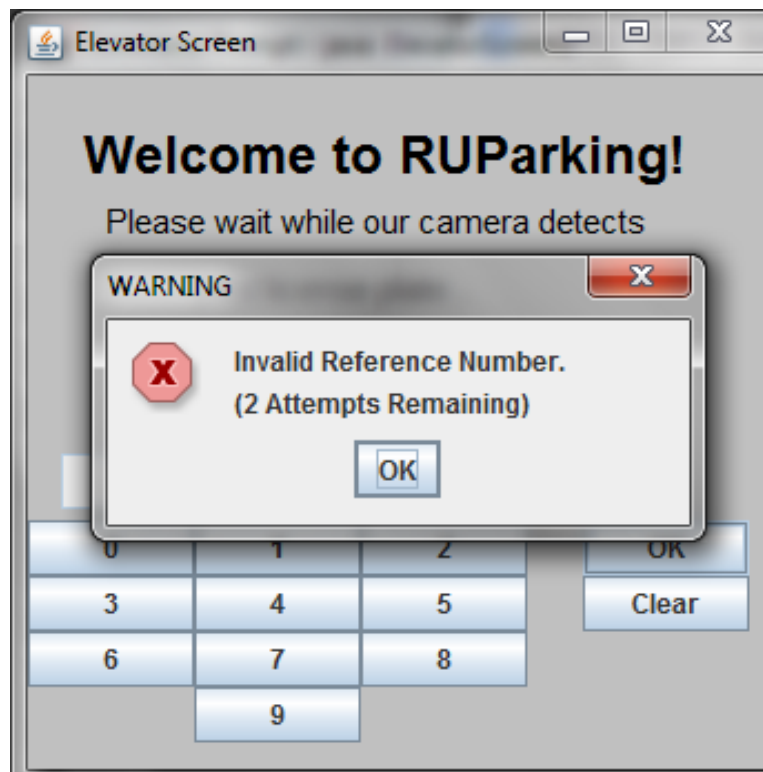
- Popup Windows to inform user of an event.
- Limit of 5 attempts for user to enter Reference Number.
- User cannot proceed before acknowledging popup window.
- User has option of telling elevator to lift to appropriate floor after reservation is verified or user may choose for elevator not to lift to appropriate floor in the case user wants to exit the parking garage before parking. If user selects “No”, the system will begin as a new transaction.

We will allow the user to have at most five attempts to enter their reference number. We will alert the user each time they enter an incorrect reference number and will provide them with the number of attempts they have left to enter a correct reference number. This will make the system easy to use not only for the customer currently using it, but also for the customers waiting in the queue to enter the elevator.



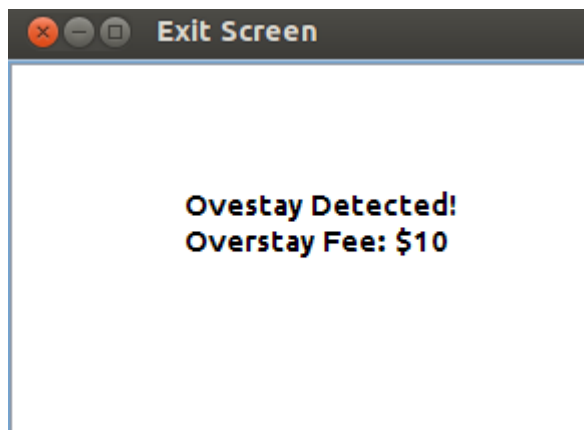
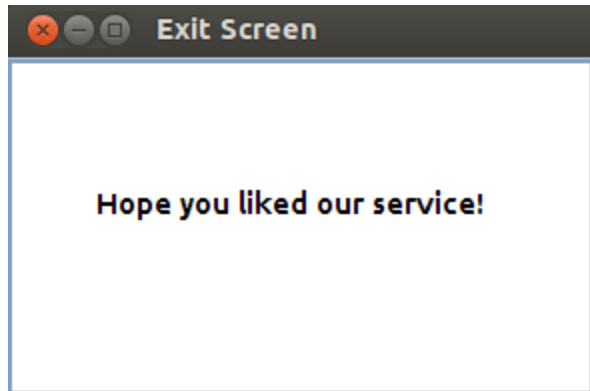






### ***Exit Interface***

This is a new addition to the project. Upon reservation zone exit, the system will detect whether the user has overstayed or not. In the design of this interface, we aim to minimize the number of words displayed, since the purpose of the exit module is to allow the user to cruise through the exit without stopping.



## 12. Design of Tests

### Integration Testing

Integration testing is a level of software testing process where individual units are combined and tested as a group. The purpose of this test is to expose errors in interaction between modules. For our system, we elected to utilize *Bottom-Up* integration testing. In the bottom-up approach, testing is conducted from low-level modules to main module. This approach is very efficient in detecting small bugs that may occur toward the bottom of the program.

After Bottom-Up testing, we will implement the *Big Bang* approach to integration testing. In this approach, all modules are combined and tested in one go. Note that we will use the Big Bang approach after completing Bottom-Up testing since Big Bang approach is inefficient in detecting small bugs that may exist in low-level modules.

### Test Cases

|                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                                                                                                                                                                                                                                                                        | TC-1 Registration                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Pass/Fail Criteria:</b>                                                                                                                                                                                                                                                                                                                                                                          | The user must enter their first name, last name, a valid email address, credit card number, phone number and password. If one of these fields is missing the registration fails.                                                                                                                                                                                                                                                                   |
| <b>Input Data:</b>                                                                                                                                                                                                                                                                                                                                                                                  | First name, Last name, Email, Credit Card, Password and Phone number                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Test Procedure:</b><br><br>Test 1. Submit a request without filling all the required fields.<br><br>Test 2. Submit the request with an email address that already exists in the database<br><br>Test 3. Submit a request where the two password fields do not match.<br><br>Test 4. Submit a unique email address, password fields that match, and the other requested (Input Data) is provided. | <b>Expected Result:</b><br><br>Mobile App Displays "Please fill out this field." and does not process the request.<br><br>Mobile App redirects the user to the registration page and displays "The email is already in use".<br><br>Mobile App redirects the user to the registration page and displays "The passwords do not match".<br><br>The system successfully registers the customer and will be redirected to home page of the Mobile App. |

|                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                                                                     | TC-2 Log In                                                                                                                                                                                                                                                                                                                                                              |
| <b>Pass/Fail Criteria:</b>                                                                                                                                                                       | The test passes if the user or operator enters an email password combination that is contained in the database and fails otherwise.                                                                                                                                                                                                                                      |
| <b>Input Data:</b>                                                                                                                                                                               | Email, Password                                                                                                                                                                                                                                                                                                                                                          |
| <b>Test Procedure:</b><br><br>Test 1. Submit blank fields<br><br>Test 2. Submit an unknown email address, password combination<br><br>Test 3. Submit a valid email address, password combination | <b>Expected Result:</b><br><br>User interface displays “Invalid email/password combination, please try again” and does not process the request.<br><br>User interface displays “Invalid email/password combination, please try again” and does not process the request.<br><br>Program logs in the customer successfully and redirects them to the Mobile App home page. |

|                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                                                                                                                                                               | TC-3 Reservation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Pass/Fail Criteria:</b>                                                                                                                                                                                                                                                                 | The reservation succeeds if the customer enters a start date, start time and duration for which a reservation is available. Else the test case fails.                                                                                                                                                                                                                                                                                                                                                    |
| <b>Input Data:</b>                                                                                                                                                                                                                                                                         | Start date, Start time, Duration, License plate (optional)                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Test Procedure:</b><br><br>Test 1. Submit a request without filling all the required fields<br><br>Test 2. Submit a request that does not pass the scheduling algorithm (no space available for that time).<br><br>Test 3. Submit a valid request that passes the scheduling algorithm. | <b>Expected Result:</b><br><br>Mobile App displays "Please fill out all the fields correctly before submitting" and does not process the request.<br><br>Mobile App redirects the user to the reservation page and displays "There are no spots available for that requested time frame, please try another request".<br><br>Mobile App redirects the user to the confirmation page for the user to confirm reservation as well as payment. Their reference number will be displayed after confirmation. |

|                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                                                                                                                                                                                                                                                                                                   | TC-4 Elevator Entry                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Pass/Fail Criteria:</b>                                                                                                                                                                                                                                                                                                                                                                                                     | The license plate is not recognized, and the user enters a valid reference number corresponding to a given reservation. If an invalid reference number is given it fails after maximum number of attempts (3 attempts) has been exhausted.                                                                                                                                                                                                                                                                                                                |
| <b>Input Data:</b>                                                                                                                                                                                                                                                                                                                                                                                                             | Reference number                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Test Procedure</b><br><br>Test 1. The user submits empty fields (does not enter a reference number).<br><br>Test 2. The user submits an invalid reference number.<br><br>Test 3. The user enters a valid reference number.<br><br>Test 4. The user selects "Yes" when asked to Lift Elevator after a valid reference number is entered.<br><br>Test 5. The user inputs invalid reference number maximum number of times(3). | <b>Expected Result</b><br><br>The Elevator Interface displays "Please input Reference Number."<br><br>The Elevator Interface displays "Invalid Reference Number. (2 Attempts Remaining)"<br><br>The Elevator Interface displays "Reservation Verified! Lift Elevator to appropriate floor?"<br><br>The Elevator Interface displays<br>"Parking floor: 2.<br>Parking Spot: 243"<br>The Elevator Lifts to the appropriate floor.<br><br>The Elevator Interface displays "Reservation does not exist. Please exit the elevator and park in the walk-in zone" |



|                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                                                                                  | TC-5 Password Update                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Pass/Fail Criteria:</b>                                                                                                                                                                                    | The test passes if the user puts in a request that does not conflict with another existing database entry and is valid                                                                                                                                                                                                                                                                       |
| <b>Input Data:</b>                                                                                                                                                                                            | New Password, Old Password                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Test Procedure:</b><br><br>Test 1. Submit blank field<br><br>Test 2. Submit incorrect Old Password<br><br>Test 3. New Passwords don't match<br><br>Test 4. New Passwords match and Old Password is correct | Expected Result:<br><br>Mobile App displays "please fill in the required fields" and does not process the request<br><br>Mobile App displays "enter the correct old password" and does not process the request<br><br>Mobile App displays "your passwords don't match" and does not process the request<br><br>Mobile App redirects the user to a page confirming that change has been made. |

|                                                                                                                                              |                                                                                                                                                                                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                 | TC-6 Credit Card Update                                                                                                                                                                                                                                                                                     |
| <b>Pass/Fail Criteria:</b>                                                                                                                   | The test passes if the user puts in a request that does not conflict with another existing database entry and is valid                                                                                                                                                                                      |
| <b>Input Data:</b>                                                                                                                           | New Credit Card                                                                                                                                                                                                                                                                                             |
| <b>Test Procedure:</b><br><br>Test 1. Submit blank field<br><br>Test 2. Submit invalid credit card<br><br>Test 3. Submit a valid credit card | Expected Result:<br><br>Mobile App displays "please fill in the required fields" and does not process the request<br><br>Mobile App displays "please fill a valid credit card number" and does not process the request<br><br>Mobile App redirects the user to a page confirming that change has been made. |

|                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                                     | TC-7 Phone Number Update                                                                                                                                                                                                                                                                                               |
| <b>Pass/Fail Criteria:</b>                                                                                                                                       | The test passes if the user puts in a request that does not conflict with another existing database entry and is valid                                                                                                                                                                                                 |
| <b>Input Data:</b>                                                                                                                                               | New Phone Number                                                                                                                                                                                                                                                                                                       |
| <b>Test Procedure:</b><br><br>Test 1. Submit blank field<br><br><br><br>Test 2. Submit a invalid phone number<br><br><br><br>Test 3. Submit a valid phone number | Expected Result:<br><br>Mobile App displays “please fill in the required fields” and does not process the request<br><br><br><br>Mobile App displays “please enter a valid phone number” and does not process the request<br><br><br><br>Mobile App redirects the user to a page confirming that change has been made. |

|                                                                                                                                                                                       |                                                                                                                                                                                                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                                                          | TC-8 Management Access                                                                                                                                                                                                                                                   |
| <b>Pass/Fail Criteria:</b>                                                                                                                                                            | The test passes if the user inputs valid (positive) rates.                                                                                                                                                                                                               |
| <b>Input Data:</b>                                                                                                                                                                    | Rates, Cancellation Duration                                                                                                                                                                                                                                             |
| <b>Test Procedure:</b><br><br>Test 1. Rates is an invalid rate such as a negative number<br><br>Test 2. Cancellation Duration is an invalid time<br><br>Test 3. All fields are valid. | Expected Result:<br><br>Website displays “Please enter a valid rate” and does not process the request.<br><br>Website displays “Please enter a valid duration” and does not process the request.<br><br>Website redirects the operator to a page confirming the changes. |

|                                                                                                                                                  |                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Test-Case Identifier:</b>                                                                                                                     | TC-9 Reservation Zone Exit                                                                                                                                                                                                                                              |
| <b>Pass/Fail Criteria:</b>                                                                                                                       | The test passes if system correctly detects overstay at time of exit.                                                                                                                                                                                                   |
| <b>Input Data:</b>                                                                                                                               | License Plate Number                                                                                                                                                                                                                                                    |
| <b>Test Procedure:</b><br><br>Test 1. The user exiting the garage has overstayed.<br><br>Test 2. The user exiting the garage has not overstayed. | <b>Expected Result</b><br><br>The system correctly detects the overstay and calculates the final cost to be charged on the exit interface.<br><br>The system recognizes this fact and displays a confirmation page after updating parking spot vacancy in the database. |

## **Unit Testing**

The unit testing is based upon 6 main tasks within our system. All of the use cases as well as test cases corresponding to these use cases can be placed under the six main categories. The categories include Registration, Reservation, Authorization, Elevator Entry, Management, and Reservation Zone Exit. Unit testing will be implemented using the test cases above to test components of the three main modules that make up the system.

The breakdown of the testing that will be performed on the six main categories is explained below:

### **Registration**

Registration is an essential part of the system as having an account is necessary in order to make reservations. An unregistered user will be able to register successfully if the email address provided by the unregistered user during registration does not match that of another user existing within the database. All users will be identified by their email, and their email will serve for the user to logging in along with a chosen password.

### **Reservation**

The reservation test cases are directed at testing the cases that do not pass the scheduling algorithm. The user in this case will be prompted into selecting a start date, start time as well as a duration for the requested reservation. If the user fails to input any of the fields, the request is not processed. If the user provides all valid fields the system checks to see if the reservation is available, and returns appropriate response to the user.

### **Authorization**

The authorization test cases are important as they check if the users/managers can log-in and also if the information parsed by the user matches with the database. Testing will involve asserting the modules with login credentials and reference numbers that both match and differentiate themselves from the ones in the database.

### **Elevator Entry**

The elevator interface asks the user to input a reference number this the case where their license plate number isn't recognized by the elevator camera. User will have 3 attempts to successfully enter a valid reference number. Once a valid number is entered, user will be taken to the correct floor. If the user fails to enter a valid reference number within the maximum number of attempts allowed, he/she will be asked to exit the elevator.

### **Management**

The management test cases are directed towards pricing and policies for the entire system. The goal for the design of these tests is to cover the interactions an operator has with the system making sure that all the changes they are making are valid. It is important to have cover all the scenarios, regardless of the situation.

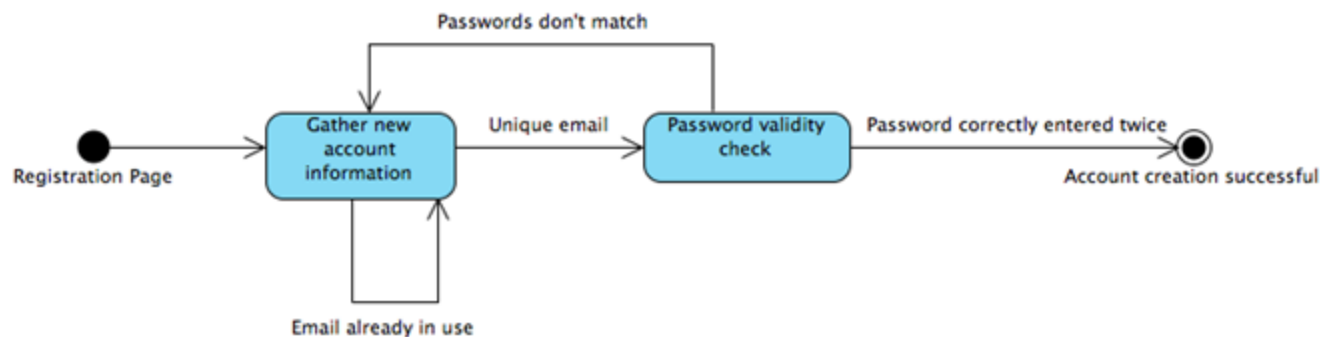
### Reservation Zone Exit

The reservation zone exit test cases must verify that the exit module correctly detects overstays upon exit and update the database vacancy correctly. It must also bill the user in case of overstay and display the action taken to the user.

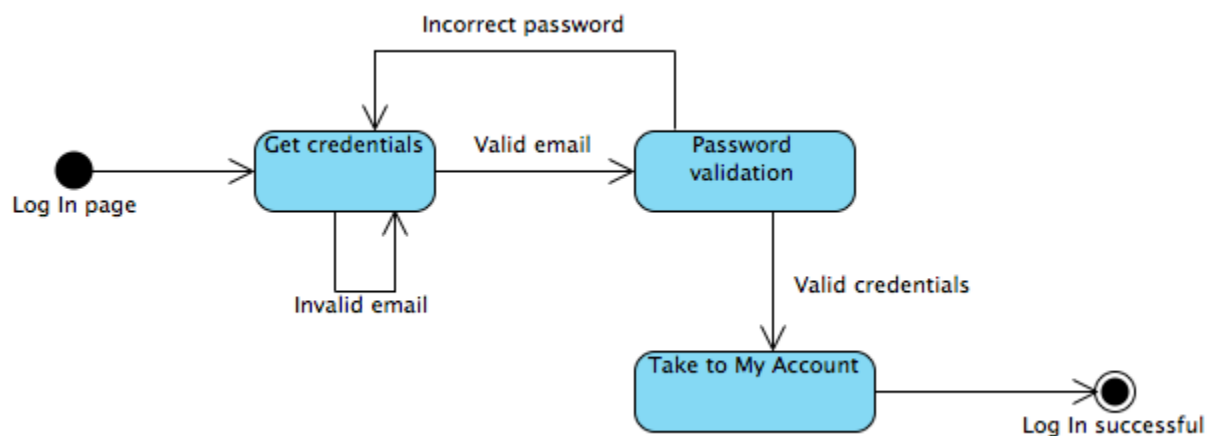
### **Test Coverage**

Test coverage was done using state diagrams. The test cases stated above cover all the states of the following state diagrams. There are also a number of test cases that cover the same states and/or transitions. In these cases, their utility is in code coverage – testing each line of code regardless of function.

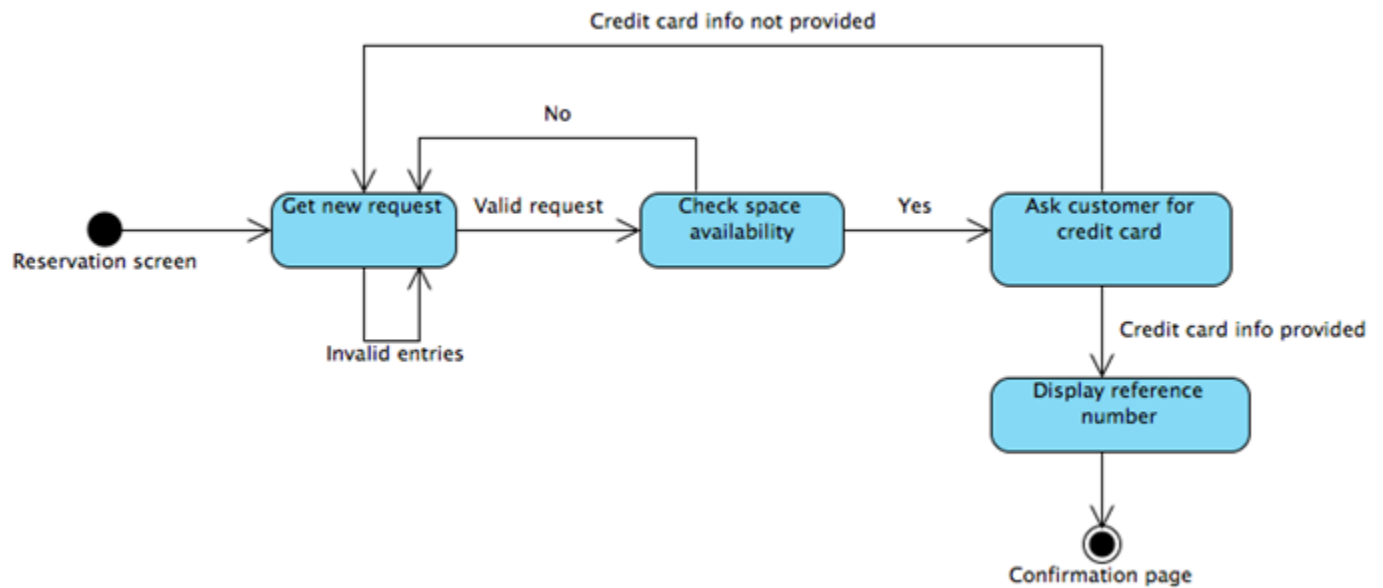
### Registration



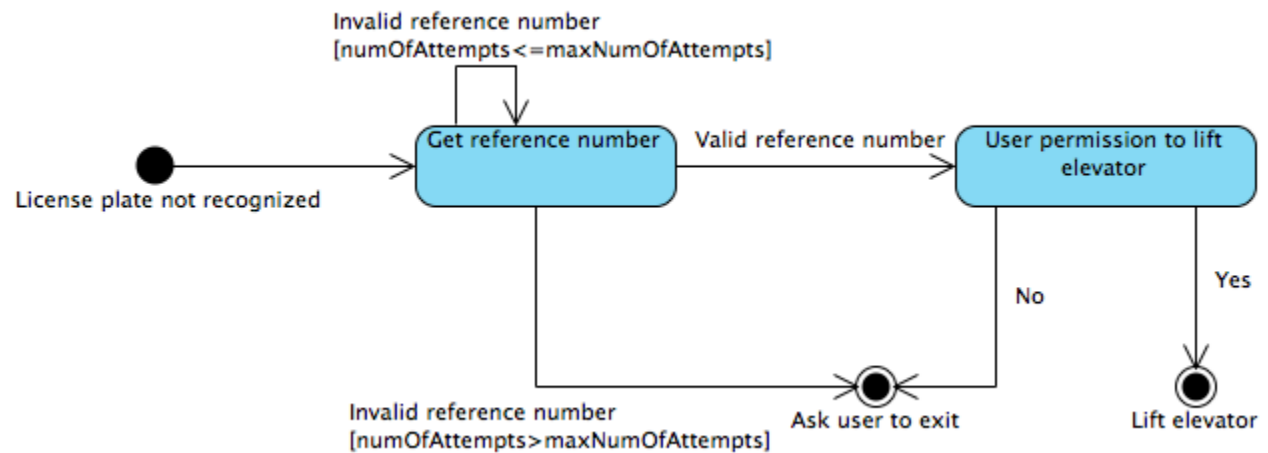
### Authorization



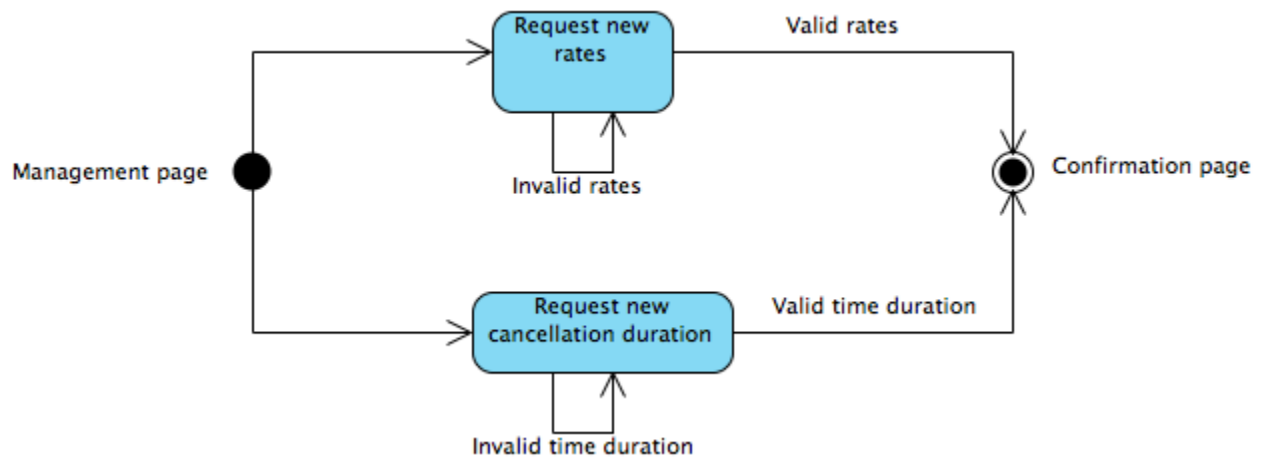
### Reservation



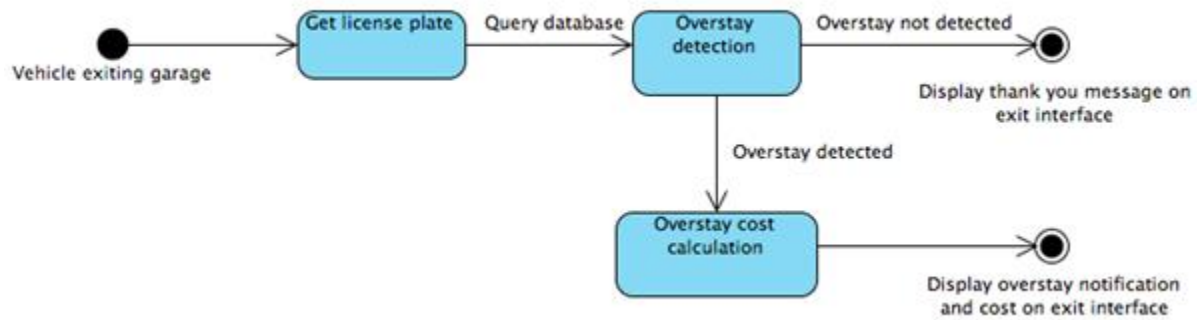
### Elevator Entry



### Management



### Reservation Zone Exit



## 13 History of Work, Current Status, and Future Work

Since the last demo, there have been many changes to the design and implementation of our project. Some of the major changes since the demo are:

- An exit screen that detects and confirms customers' exit. Customers are also shown whether or not they overstayed and the fine for their overstay.
- Garage operators can view statistical models in the management interface. These models will provide the management with useful information such as peak times and total revenue.

Our progress developed as follows:

|                                               |                   |
|-----------------------------------------------|-------------------|
| Completion of first report                    | February 22, 2013 |
| Completion of second report                   | March 15, 2013    |
| Database implementation                       | March 20, 2013    |
| Elevator interface and database integration   | April 1, 2013     |
| Management interface and database integration | April 1, 2013     |
| Completion of demo 1                          | April 2, 2013     |
| Mobile app and database integration           | May 2, 2013       |
| Exit interface and database integration       | May 2, 2013       |
| Completion of third report                    | May 5, 2013       |
| Completion of demo 2                          | May 10, 2013      |

We were able to meet all the deadlines for the milestones shown above.

### Key accomplishments

- Created a relational database to hold all the tables for ease of access
- We successfully wrote a reservation algorithm that correctly communicates with our database in order to find an open spot.
- Created an easy-to-use mobile app that is correctly integrated with our system
- Created a fully-functional management interface that garage operators can use to modify rates and cancellation policy
- Created entry and exit interfaces for garage customers in order to provide them information about their reservation



The system in its current state is functional. It contains all of the major functionalities described by our use cases and tested successfully using the test cases stated above. We were able to accomplish all the goals we set out with. In the next section, we will state some of the additional features and improvements that we think will work well with our system.

## **Future Work**

### *Mobile App:*

#### 1. Make the UI more user friendly

To make the mobile app more user-friendly, we can either consult experts on the matter, or use an iterative process with a small timebox using customer feedback to improve our design.

#### 2. Write an IOS version of the App.

Our Android app is written in Java. To create an iOS version of the app, it must be written again in Objective-C. The iOS version would greatly improve the number of potential customers to our garage.

#### 3. Database Interfacing with Android SQLite

Using SQLite, we can create a copy of the database locally on the Android, and using external programs this database would use the internet to sync with the garage database on a regular basis.

### *Management Module:*

#### 1. Website Integration:

The management module could be further modified to include a website that the manager can access from any location. This would involve rewriting all database interface functionalities for the webservice, likely in PHP.

#### 2. Multiple Garages with different policies:

The interface was designed around the RUParking garage. All fields were designed with the business policies and assumptions of this garage in mind. Suppose we were looking to expand the reach of our parking system to other garages. We would have to generate a new page layout allowing the respective managers to access their garage-specific business policies.

### *Elevator Module:*

#### Multiple-Elevator System:

We could remodel the parking garage entrance for the reserved zone to allow multiple users to enter the garage at the same time thus reducing congestion. This would involve adding multiple GUI's to the entrance and eliminating the elevator requiring customers to drive through to their assigned parking spot and floor.

## 14. References

1. Marsic, Ivan. *Software Engineering*. 2012
2. <http://softwaretestingfundamentals.com/integration-testing/>
3. Group 3 (2012) - Reserve-a-Spot
4. Group 4 (2012) - Parking Garage
5. Visual Paradigm for UML Software
6. GanttProject Software