Software Engineering Spring 2016

Group #15

Parking Garage Automation

<http://www.galuwa.com>

Report 2 Final

3/13/16

|  |  |
| --- | --- |
| Brandon Dunlap | brandon.dunlap@rutgers.edu |
| Vikram Krishna | vk235@rutgers.edu |
| Yufeng Liu | yufeng.liu@rutgers.edu |
| Luke Miller | luke.miller@rutgers.edu |
| Harshil Patel | harshil1029@gmail.com |
| Thomas Walters | twalters1012@gmail.com |
| Xiang Xing | xx52@scarletmail.rutgers.edu |

**Contributions**

All team members contributed equally.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Brandon Dunlap | Vikram  Krishna | Yufeng  Liu | Luke  Miller | Harshil  Patel | Thomas  Walters | Xiang  Xiang |
| Interaction  Diagrams | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| Class Diagram | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| Data Types | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| Traceability Matrix | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| System Architecture | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| Algorithms | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| User Interface | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| Testing | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| Project Management | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |
| References | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% | 14.29% |

Table of Contents:

**Interaction Diagrams**

UC-1 Registration……………………………………………………………………….4

UC-2 Reservation……………………………………………………………………….6

UC-3 Parking…………………………………………………………………………….7

UC-4 Overstay…………………………………………………………………………..9

UC 5 - Departure………………………………………………………………………10

UC-6 Policy Decision………………………………………………………………….12

**Class Diagram**………….…………………………………………………………………….13

**Data Types and Operation Signatures**……………………………………………………14

**Traceability Matrix**…………………………………………………………………………..23

**System Architecture and System Design**

Architectural Styles…………………………………………………………………….26

Identifying Subsystems………………………………………………………………..26

Mapping Subsystems to Hardware…………………………………………………..27

Persistent Data Storage………………………………………………………………28

Network Protocol………………………………………………………………………29

Global Control Flow……………………………………………………………...……29

Hardware Requirements………………………………………………………………30

**Algorithms and Data Structures**

Software Simulation…………………………………………………………………...32

Account Management…………………………………………………………...…….32

Arrivals…………………………………………………………………………….……33

Garage Policies and Scheduling……………………………………………….……33

Departure………………………………………………………………………….…...33

Statistics……………………………………………………………………………….34

Reservations……………………………………………………………………………34

**User Interface Design and Implementation**

Software Simulation………………………………………………………………..….36

Account Management…………………………………………………………..……..36

Arrivals………………………………………………………………………………….36

Garage Policies and Scheduling……………………………………………..………36

Departure……………………………………………………………………………....36

Statistics………………………………………………………………………………..37

Reservations…………………………………………………………………………..37

**Design of Tests**

Software Simulation…………………………………………………………….….…38

Account Management……………………………………………………………..…38

Arrivals……………………………………………………………………….…………39

Garage Policies and Scheduling………………………………………………..…...39

Departure……………………………………………………………………………….39

Statistics……………………………………………………………………….….…….39

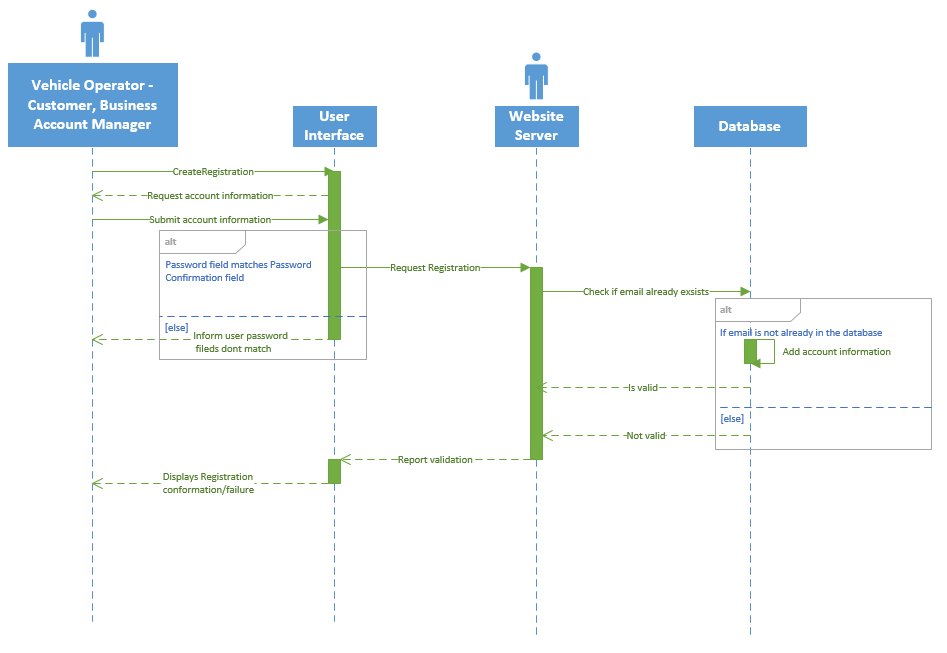
Reservations………………………………………………………………...….…….40

**Project Management and Plan of Work**……...…………………………………………41

**References** ………………………………………………………………………………..….42

**Interaction Diagrams**

UC-1 Registration



Description: This UML diagram for UC-1 Registration demonstrates the process of a user/customer creating an account on the Garage Automated system website using a method that involves the client making a request of the server, the server making a request of the database, and the result cascading back to the client.

Step 1

Customer→ User Interface: Customer presses button to begin registration process.

Step 2

User Interface→ Customer: Ask customer to fill out account information form.

Step 3

Customer→ User Interface: User fills out form and presses the “Submit” button.

Step 4

User Interface→ Website Server: Interface submits form to the Controller.

Step 4 alt

User Interface→ Customer: Informs user that both the password and password confirmation fields must match.

Step 5

Website Server→ Database: Check if customer’s email is already in use.

Step 6

Database → Website Server: Confirm customer account’s submission to database.

Step 6 alt

Database → Website Server: Customer email is already in use, registration not valid.

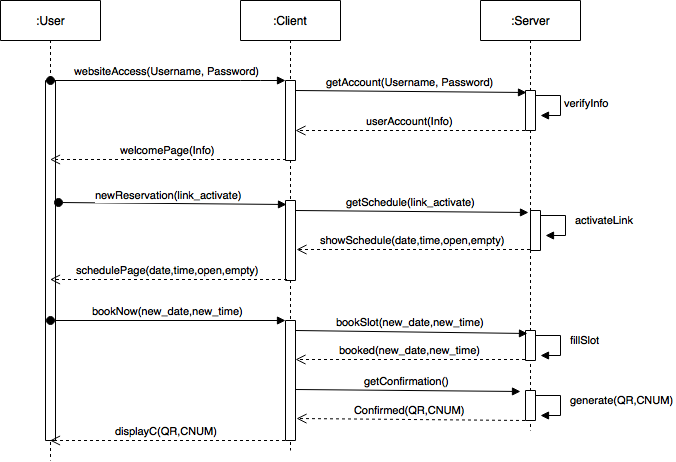
Step 7

Website server → User Interface: relay registration results.

Step 8

User Interface → Customer: show registration results to user.

UC-2 Reservation

[](https://drive.draw.io/#G0B1vkj8y_n1eFbkZHWnpnVXJNUFU)

Description: This UML Diagram for Reservation details the interaction between the User, Client and Server as a User attempts to create a new reservation online.

Step 1:

User → Client: User accesses website and types in username and password to log in to his account.

Client→ Server: The clients asks the server to get the account information with the given credentials.

Server→ Client: The server(database) verifies the info and gives the client the user info. Client→ User: The client displays the end-user with a welcome page and his account information.

Step 2:

User→ Client: User clicks link to create a new reservation.

Client→ Server: The client asks the server for the schedule of reservations.

Server→ Client: The server sees the activation and sends the client information of the schedule. Client→ User: The end-user is greeted to a schedule page with dates, times, and booked and unbooked slots.

Step 3:

User→ Client: The user sends information through client to book a slot with a new date and new time.

Client→ Server: The client forwards this information to the server.

Server→ Client: The server(database), after verifying the slot is available, books the slot and tells the client.

Client→ Server: Once receiving the booking, the client asks for confirmation.

Server→ Client: The server then generates a QR code and a confirmation number, after making sure the request is valid, and sends it to the client.

Client→ User: The client finally sends this confirmation information to the end-user who can then use it as he needs when parking.

UC-3 Parking

UC3-PARKING.png

Description: This UML Diagram for Parking details the interaction between the Customer, Elevator Console, Plate Scanner, Occupancy Photosensor and parking database as a Customer arrives at the garage.

Step 1

Customer -> Plate Scanner: Customer arrives at the garage and has their license plate scanned to confirm an existing reservation. This reservation is verified and a spot number is displayed to the user.

Step 1 Alternate

Customer -> Elevator Console: Customer arrives at the garage with an existing reservation, but the license plate scanner is unable to link license plate to a reservation. Customer will manually input a confirmation number, the information will be verified and the spot number is displayed to the user.

Step 2

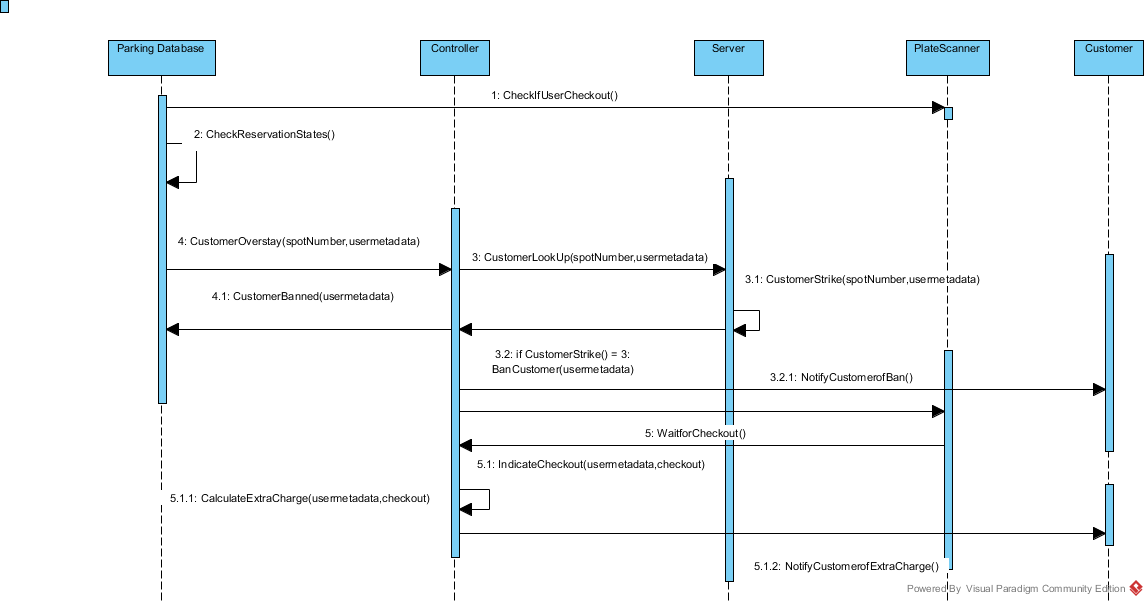
License Plate Scanner -> Occupancy Photosensor: The License Plate Scanner will listen through the Occupancy Photosensor at an assigned spot to determine that a user has parked in their spot within a timeout period.

Step 3 Alternate

Elevator Console -> Occupancy Photosensor: The Elevator Console will listen through the Occupancy Photosensor at an assigned spot to determine that a user has parked in their spot within a timeout period.

Step 4

Occupancy Photosensor -> Parking Database: Once a car has arrived, the Occupancy Photosensor will relay to the parking database that an additional vehicle has been parked in the garage.

UC-4 Overstay

Description: This UML Diagram for Overstay details the interaction between the Parking Database, Controller, Server, Plate Scanner and Customer as a Customer leaves the garage laste.

Step 1: Parking Database <-> PlateScanner Parking Database retrieves information from the PlateScanner as to whether the cars in timeperiod T have checked out

Step 2: Parking Database<-> Parking Database

Scans database for spots thats are overstaying in timeperiod T

Step 3: ParkingDataBase <-> Controller

Indicates that a customer or customers have indeed overstayed while passing their spotnumber and metadata(car #, plate number, model, etc.)

Step 4: Controller<->Server

Finds the customer with metadata and spotnumber provided

Step 5: Server <-> Server

Issues a strike for the customer for overstaying, and if the user has had 3 strikes(including current), then proceed with alternative path

Step 6: Controller<->PlateScanner

Controller waits for plate scanner to register user has left the garage

Step 7: PlateScanner <-> Controller

PlateScanner sends confirmation to Controller indicating the user has left, along with the usermetadata and checkout time

Step 8: Controller <-> Controller

Calculates the extra charge based upon the surcharge rate and how long the overstay was

Step 9: Controller <-> User

Send an email ,text, or user preference of communication to indicate that he has overstayed, gained a strike, and is paying this surcharge

Alternative Path:

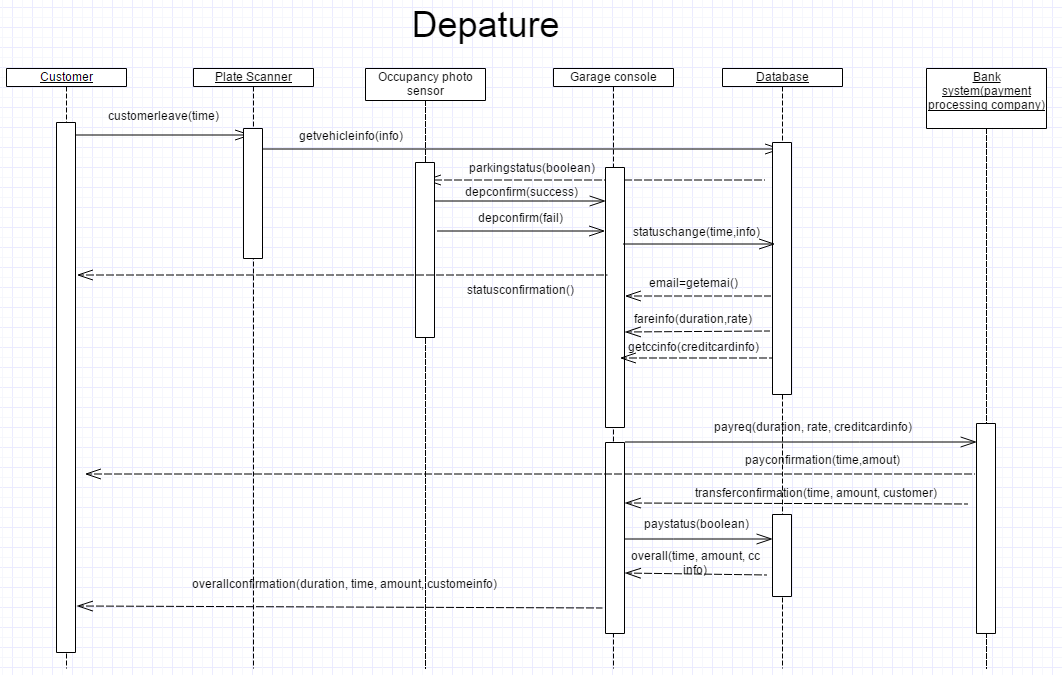
5.1: Controller<-> ParkingDatabase

Update the ParkingDatabase to indicate the user's ban from the parking garage along with his information(car, model, license plate)

5.2: Controller <-> User

Indicate to the user via email, that he has been banned from the parking garage for his overstay strikes

UC 5 - Departure



Description: This UML Diagram for Departure details the interaction between the Customer, Plate Scanner, Occupancy Photosensor, Garage Console, Database and Payment processing system as the

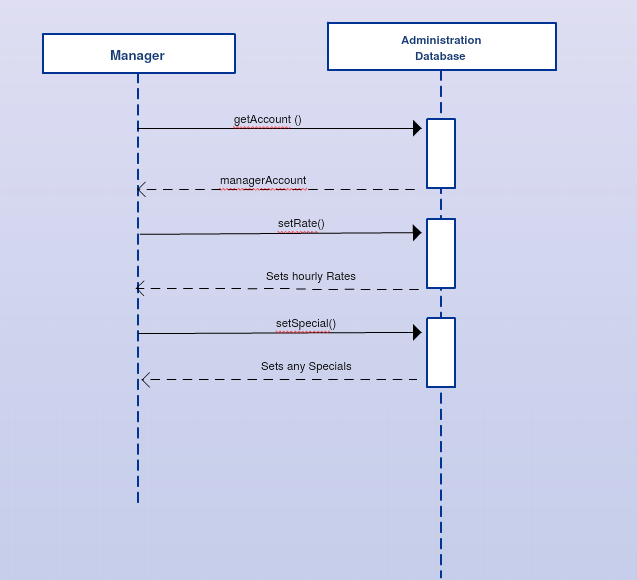
Step1. Customer->plate Scanner. Customer leave the garage and plate Scanner detect the leaving

Step 2. Plate Scanner->Occupancy Photo sensor. Plate Scanner send a message to Occupancy Photo sensor to make sure the leave

Step 3. Garage console->database. After confirmation, the Garage console ask for customer information from database

Step 4. Garage console->Payment processing company. Garage console post a payment requirement to bank system and wait for reply

Step 5. Garage console-> Customer and database. Garage console send a confirm e-mail to customer and a history log to database

UC-6 Policy Decision

Description:

Step 1

Manager→Administration Database: The manager first will login to his account on the administration database.

Manager→Administration Database: The manager can set hourly rates for parking in the garage.

Manager→Administration Database: The manager can set any specials for the upcoming week or month for the parking garage.

**Class Diagram**

ClassDiagran (2).png

**Data Types and Operation Signatures**

**Vehicle**: This class specifies the components of users’ vehicles information. And it will be stored to the database.

+ LicensePlateNumber: string: License plate number of users’ vehicles.

+ Cartype: string: The type of users’ vehicles.

+ Color: string: The color of users’ vehicles.

**Payment Method**: This class specifies the components of users’ payment information. And it will be stored to the database.

+ CardNumber: int: The card number of users.

+ CardHolder: string: The card holder of users.

+ ExpirationYear: int: The expiration year of users’ cards.

+ ExpirationMonth: int: The expiration month of users’ cards.

**ReservationDatabaseInterface:** This class serves as an interface between the parking garage controller and the reservation database. A set of functions will allow any developer to manipulate entries within the reservation database.

getName: Returns the Database name as a string.

addReservation(New: Reservation): This method will add a new vehicle to the parking database and return a boolean value indicating success.

removeReservation(R: Reservation): This method will remove and existing vehicle from the parking database and return a boolean value indicating success of the operation.

getReservation(R: Reservation): This method will return a Reservation object found in the database by the ReservationID. If this vehicle cannot be found then the method will return null.

**OccupancyPhotosensor**: This class serves as an object representing each Occupancy Photosensor within the garage. This class will be modified by the simulation to represent cars actively present in the garage.

isPresent: Represents the presence of a vehicle in the current parking spot. This is a boolean value.

sensorID: Represents the unique sensor ID for that sensor. This is used to relate a sensor to a specific parking spot. This is an integer value.

getStatus(): Returns the status of the sensor instance as a boolean value.

getID(): Returns the ID of the sensor instance as an integer.

setID(int newID): Sets the ID of the sensor instance as an integer and returns a boolean value indicating success.

**ParkingDatabaseInterface**: This class serves as an interface between the parking garage controller and the parking database. A set of functions will allow any developer to manipulate entries within the parking database.

getName: Returns the Database name as a string.

addVehicle(New: Vehicle): This method will add a new vehicle to the parking database and return a boolean value indicating success.

removeVehicle(V: Vehicle): This method will remove and existing vehicle from the parking database and return a boolean value indicating success of the operation.

getVehicle(int VID): This method will return a Vehicle object found in the database by the VID. If this vehicle cannot be found then the method will return null.

**Payment Processing:** This class serves as the method for which payment for spot reservation can be completed

retailerID:int : The identification number of retailer.

processPayment(CustomerID: int, CustomerName: String, CardNumber: int, ExpDate: int, Amount: int): process the payment when customers leave the garage.

sendReceipt(CustomerID, ResNum: int, Amount: int, Message: String): Boolean : send the receipt to the customers.

**Outdoor Display:** This class serves as the display on the outside of the garage which will show how many open spots are in the garage at any time.

vacancy: boolean: show the if the spot is still vacancy.

spotCount: int: show the number of spots which are vacancy.

operatingStatus: String: show the operating status.

printMessage(Message: String): show the message on the outdoor display.

setVacancy(Vacancy: boolean): set the spot to the vacany.

setStatus(Status: String): set the operating status.

**System User**

Operational Types:

+ getName(): String: get the name of the user.

+ getAddress(): String: get the address of the user.

+ getID(): String: get the user ID.

+ setPassword(Old: String, New: String): Boolean: change the password of the existed user ID

+ changeEmail(New: String): Boolean: Change the email of the User ID.

Data signatures:

- FirstName: String: User’s firstname.

- LastName: String: User’s lastname.

- Address: String: User’s Address.

- UserID: String: User’s account name.

- Password: String: User’s password of his account

- EmailAddress: String: User’s email address.

- CarList: List<Vehicle>: The list to store the users’ vehicle information.

- Payment: List<PaymentMethod>: The list which stores the users’ payment information.

**Web Server/UI**

Operational Types:

+requestReservation():void Allows the user to request a reservation for a specific date and time period

+requestCancellation():void The user will cancel the reservation

+requestpaymentchange():void The user will update/change or add a new payment option

+requestpasswordchange(Old: String, New: String):void The user will update his password

+DisplayReservationMap():void Display interactive map showing reservation times free

+DisplayAccountSetup():void Display interactive, user friendly account setup page

+DisplayAccountPage():void Display account page for anyone that has an account

+login(Username: String, Password: String):boolean Display sign in page

+logout(): Display logout page

Data Signatures:

ActiveUser: SystemUser: The user who is using the system right now.

**System Manager:** This class serves as an interface for manager log in. A manager is able to view the website and data as a manager. As a manager, you can add account and change the password through it.

UserID: Return the account ID as a string

ChangePassword: change the password of current account and return boolean if success

DeleteUser: delete current user or add a new manager’s account

**Manager account database:**

currentRate: Return current charging rate as double

Reservation info:Return customer information as string, return type may change according to Customer and database part.

Reservation Status: Return reservation status as string

Payment\_method(firstName : char, lastName : char, creditcardinfo: string): get payment method from database. Return boolean if it is valid.

get\_info(firstName: char, lastName:char, vehicle info: string, reservation status: string, payment status: boolean): void . get vehicle and customer info. Return void.

change\_rate(current\_rate double): double : change current charging rate. Return new rate as double.

changeReservation(vehicle: char, date:int, time int ); string : change the reservation information.

return string of vehicle&date&time

getpayment(customer name: string , creditcardinfo string): get payment method of customer, return boolean if success

changepayment\_status(): boolean change payment status to paid, or unpaid, and return corresponding boolean value if success

**Customer:** This class serves to hold and change all information regarding the customer and connects the customer to all other classes in some way.

Operation Signatures & Descriptions

customerInfo(firstName : string, lastName : string, zip : int, email : string, cPass : string) : customer

This method holds all the main info of the customer.

reservationHistory(rDates : string, rTimes : string) : string

This method holds the reservation history based on the customer’s account.

paymentInfo(cardFirst : string, cardLast : string, ccn : int, expDate : int, securityNum : int) : customer

This function is used to hold and manipulate the customer’s payment methods.

vehicleInfo(numofvehicles : int, make : string, model : string, year : int, plateNum : string) : customer

This function is used to hold and change the customer’s vehicle info.

changeCustomerInfo(firstName : string, lastName : string, zip : int, email : string, cPass : string) : customer

This method is used to change customer’s info when required by the customer

changePaymentInfo(cardFirst : string, cardLast : string, ccn : int, expDate : int, securityNum : int) : customer

This method is used to change the customer’s payment info when required by the customer.

changeVehicleInfo(make : string, model : string, year : int, plateNum : string) : customer

This function is used to change the vehicle info when required by the customer.

**Controller:** This class is made to serve as an intermediary between other classes as desired.

+<<GET>>getAccount(email : string, cPass : string) : int

This method is used to get the account information of the customer from the customer database when needed.

+<<GET>>getSchedule(schedDates : string) : schedule

This function is used to get the schedule of open and booked slots from the web server based on the dates provided.

+bookSlot(newDate : string, newTime : int) : boolean

This method is for passing a booking to the server once a customer passes in the date and time he wishes to book.

+<<GET>>getConfirmation(confNum : int, QR : string) : int

This method is for retrieving the confirmation number from the server once a booking is successfully done.

+<<GET>>getHistory(rDates : string, rTimes : int, email : string, cPass : string) : string

This function is used to get the reservation history of a customer in case it is needed such as when the system is trying to detect an overstay and needs the time and date.

**Elevator Console:** This class serve as a controller for user to interact with the system in the elevator.

ConfirmationNum: The confirmation number that users should provide in order to enter the upper level of parking garage. This is an integer.

SpotNumber: The spot number which assigned to the verified users. This is an integer.

Timeout: The time which the console will monitor if users’ vehicles have arrived the certain spots in that time period. This is a floating number.

LicensePlateNum: The license plate number of users’ vehicles. This is a string value.

void SpotAssignnment(SpotNumber: int): The Elevator console will assign the spot number through the elevator display to the users when they have been verified.

bool VerifyInfoFromDatabase(ConfirmationNum: int): Verifying that if users’ reservation information is valid by matching the data from database.

void ExpectVehicle(Timeout: float, LicensePlateNum: string): Console expects the vehicles will arrive the spot in certain time by checking the occupancy photosensor.

**Plate Scanner:** This device is used to scan the users’ license plate number when they enter the elevator and leave the parking garage.

LicensePlateNum: The license plate number of users’ vehicles. This is a string value.

SpotNumber: The spot number which assigned to the verified users. This is an integer.

void GetVehicleInfo(): The plate scanner will automatically get vehicles’ information when they enter or exit the parking garage.

void TransferToConsole(LicensePlateNum: string): Transferring the plate number to elevator console or controller.

void SpotAssignnment(SpotNumber: int): The plate scanner will automatically assign the plate number to users when their plate number has been verified by scanner.

**Traceability Matrix**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Domain Concepts | Website | Parking Interface | Outdoor Sign | Parking Database |
| Classes |  |  |  |  |
| Controller | X | X | X | X |
| Customer Database | X |  |  | X |
| Elevator Console | X | X |  | X |
| Manager account database | X |  |  |  |
| OccupancyPhotosensor |  |  |  | X |
| Outdoor Display |  | X | X | X |
| ParkingDatabaseInterface |  |  |  | X |
| Payment Processing | X |  |  |  |
| Plate Scanner | X |  |  | X |
| System manager | X |  |  |  |
| System User | X |  |  |  |
| Web Server | X | X |  |  |
| Reservation Database | X | X | X | X |

Classes:

a. Controller: derived from all the domains, as it became evident that a central manager was necessary if the domains where going to work together to form the whole system.

b. Customer Database: Derived chiefly from the Website domain, but also rounded out by Parking Database domain as well. It is through the website that customers can register and edit their accounts. So, it is logical that the database which holds all the customer information belongs in the Website domain

c. Elevator Console: Derived from the Website, Parking Interface, and Parking Database domains; the Elevator Console draws reservation information from the website, and confirms to the Parking database that a new arrival to the reserved portion of the garage has been approved. The Elevator Console is also a clear part of the Parking Interface, and customers must interact with it to park.

d. Manager account database: Derived from the Website domain, this class was created because a mechanism was needed to manage the database of customers. With potentially thousands of customers registering, making reservations, or editing their account information, a dedicated class to facilitate these features was necessary. It also serves a tool for Management to make business decisions, such as adjusting prices, or managing payments.

e. OccupancyPhotosensor: Derived from the Parking Database, the Occupancy Photosensor is needed to provide constant up to date information, regarding which spaces are being used, to the Parking Database.

f. Outdoor Display: Derived from the Parking Interface, the Outdoor Sign, and Parking Database, the purpose of the Outdoor Display is to inform walk-in customers if there is space available. The Outdoor Display was derived from the Outdoor Sign domain for obvious reasons, and while customers do not directly interact with the display, is still technically part of the Parking Interface. The information presented by the display is derived from the Parking Database.

g. ParkingDatabaseInterface: Derived from the Parking Database domain, the purpose of this class is to allow users direct access to information held by the Parking Database. This class is necessary as it gives management the tools to examine the state of the garage, and find specific vehicles.

h. Payment Processing: Derived from the Website domain, as payments are made through the website. This bridge allows the controller to communicate and process payments through an external company.

i. Plate Scanner: Derived from the Website and Parking Database domains; this class feeds the License Plate information to the Parking Database, and confirms reservations with corresponding account via the Website domain.

j. System manager: Derived from the Website domain, the purpose of this class is to provide a way that the garage manager can access account information. This class represents the physical manager of the system and can be accessed/manipulated within the databases.

k. System User: Derived from the Website domain, the purpose of this class is to provide a way that customers can manage their accounts and reservations. This class represents the physical user of the system and can be accessed/manipulated within the databases.

l. Web Server/UI: Derived from the Website and Parking Interface domains. While the Web Server is obviously derived from the Website domain, as a system that manages all the requests and submissions to the Website is needed for this automated system to function. It also communicates with the parking interface when reservations, and registered customer information needs to be confirmed.

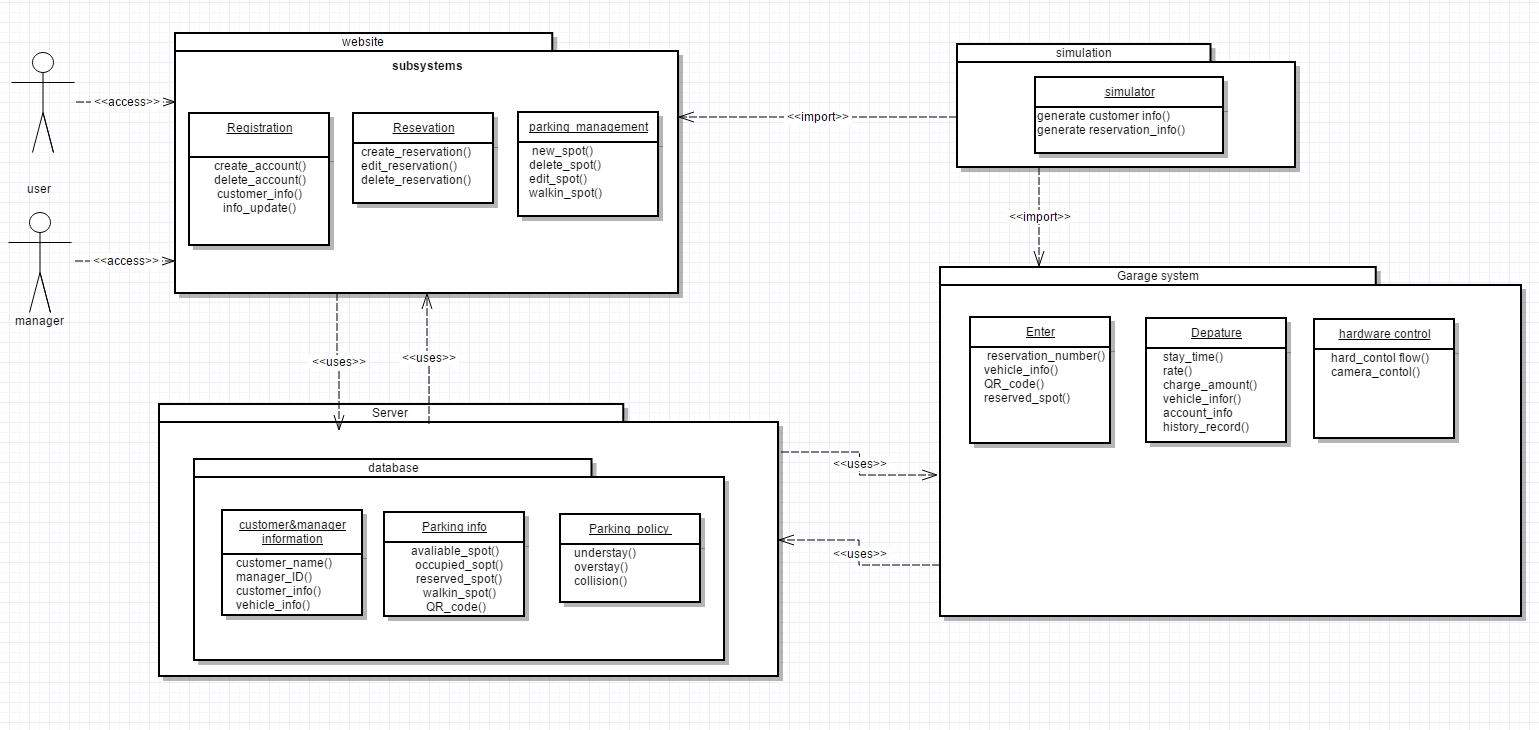
m. ReservationDatabaseInterface: This class serves as an interface between the reservation database and the rest of the system. Since many components within the Park-A-Lot system need to communicate with the reservation database, this is a public class which can be called by any aspect of the program.

**System Architecture and System Design**

Architectural Styles

The architectural style of this Garage Automated System includes principles from a variety of styles. Our system incorporates the Client-server model, as both customers and garage management access the system, and request actions, though the internet via a website. Service-oriented architecture is also apart the architectural style as the system provides a number of self-contained functions including; registering an account, checking current reservations, provide profit report from some time frame, and so on. Finally the Garage Automated System utilizes and Database-centric architecture, as the webserver that handles user requests submits and requests account and reservation information from a database. There is also a second database dedicated to the management of the garage itself, holding information regarding spot occupancy.

Identifying Subsystems



The System Architecture mainly composed of 4 parts.

The first part is website. The subsystem of the part are:

(1) Registration

(2) Reservation

(3) Parking management

Customers and managers are able to check the website through the web browser. Customer have limited access and managers have view all information included in the package.

The second part is Server, which include the database for the whole system.

The subsystem of the part are:

(1) customer and manager information database

(2) Parking information database, which includes reservation number, QR code, available parking spots, etc.

(3) Parking policy. This subsystem of database is comprised of understay, overstay, collisions records and details. Only managers have access of it

The third part is Garage system comprised of 3 subsystems:

(1) Enter the garage

(2) Departure

(3) Hardware control

The last package is a simulator , which generates and imports customer and reservations for the system.

Mapping Subsystem to Hardware

Registration/Parking Management/Reservation:

* Database
* Website Server

Customer and Manager Information Database:

* Database(Customer and Manager Part)

Parking Information Database:

* Database(Parking and Reservation part)

Parking Policy:

* Database(Manager Part)

Enter the garage:

* Outside Display
* License Plate Scanner(Entry)
* QR Scanner
* Elevator Console
* Elevator Display
* Elevator
* Occupancy Photosensor

Departure:

* Occupancy Photosensor
* Controller
* License Plate Scanner(Exit)

Hardware control :

* Controller

Persistent Data Storage

Persistent memory is an important facet of the parking garage. The main reasons being, metadata of customer information and reservation spots logically need to not be gone after 1 execution cycle, otherwise, for example the client would need to re enter his data every time or if the parking garage programs crash, the free spots bitmap needs to be rebuild easily. The persistent data will be saved on several databases, reservation, customer, and parking spot. The customer database stores all of the metadata associated with members, the reservation database stores the time, date of a reservation linked with a license-plate, confirmation number, QR scan. The parking spot database is important in cases of spontaneous failure, in order to rebuild the parking database from scratch, similar to how file systems are able to rebuild themselves.

Parking Database format: The database will contain these set of fields:

First Name, Last Name, Middle Name, Credit Card Number, Membership Number, CVC code, Zip Code, Date of Birth, Permanent Address, License Plate, Car Model, Car Type, Account Type. In addition 3 fields will be used to indicate whether this account is under a business account by a company, if so, the key of that membership account and the name of the person will be also be added.

Reservation Database format: License Plate, Confirmation Number, Date of reservation, Spot in use(this field indicates whether or not the driver has come yet),

Parking Spot Database: Spots in use

Network Protocol

Our System utilizes a website named <http://www.galuwa.com>.

The network protocol used is the HyperText Transfer Protocol (HTTP). We are using this as hypertext is a text structured so that it uses hyperlinks to navigate between nodes which contain text which is essentially what is needed. The user can click links to navigate seamlessly from page to page of the website. This protocol also consists of GET and POST requests. The GET requests can be used by the client to retrieve necessary information from the server depending on what operation is to be done. The POST method can be used to update information like new reservations, added vehicles and payment methods, and new customers. This utilizes a Transmission Control Protocol (TCP) which works with Internet Protocol (IP) to transmit data via Streams. This is preferred over User Datagram Protocol (UDP) because streams are highly reliable and the data transmitted can be ordered which is very helpful and useful.

Global Control Flow

**Execution Orderness**: Our system is mostly event-driven and each customer may not go through the same steps. Generally most customers will go through the same procedure, however there can be different events. Each customer will enter in their specific information before entering the garage. Some aspects can be considered procedure-driven, for instance each customer once the reservation is created will go through the same steps once they get to the garage.

**Time Dependency**: Our system does use timers. For example, timers are used to tell how long a customer has been parked in the garage. If a customer has been parked longer than their reservation duration, this is considered an overstay and they will be charged extra depending on the timer and how long they stayed passed their reservation time. Timers are also used to tell when a customer checked in and out of the garage. For example, a customer may leave prior to their reservation time. The timer will keep track of this and tell when spots are available. The system is based on real time and constantly will update information every second.

**Concurrency**: Our system does use multiple threads. There are many sensors, which require multiple threads in order for them to work all at the same time. There are also multiple cameras and plate readers, which also require multiple threads.

Hardware Requirements

**License Plate Scanner (Entry)**: The Entry License Plate Scanner serves as a gatekeeper for the parking garage. For cases in which a user has their license plate registered, the entry license plate scanner will recognize the plate and retrieve the User’s reservation information. When the license plate detects a license plate within the viewport, the value will be conveyed to the centralized controller for processing.

Scanner Requirements: The License Plate Scanner must be weather-proof and perform well in low light situations. One industry standard solution is to illuminate the license plate with infrared light when a vehicle is detected. The Scanner must have a minimum 25-foot scanning range and be resistant to vibration and shock. The system should use NTSC video format.

Possible models: Bosch VER-D2R5-2

**License Plate Scanner (Exit):** The Exit License Plate Scanner has the same hardware requirements as the Entry License Plate Scanner.

**Elevator Console:** The Elevator Console serves an an interface between the user and the Centralized Server. The Elevator Console is only necessary for cases in which the user does not have a License associated with their active reservation. The Elevator Console will present the option to manually enter a 6-digit reservation number.

Elevator Console Requirements: The console will feature a 10” Touch Screen.

**Elevator:** The Elevator serves as a division between the standard use “walk-in” garage and the reservation portion of our system. The elevator will only hold one car at a time and will process customer information before transporting them to the appropriate floor.

The elevator will be large enough to support oversized SUVs and personal trucks but larger extended trucks will be denied entrance.

**Occupancy Photosensor:** The Occupancy Photosensors will reside in every single garage parking spot to monitor the presence of a vehicle. The Occupancy Photosensor will be mounted in a position in the middle of the spot such that all shapes and sizes of vehicles will be detected. The sensor will be marked so as to encourage motorcyclists and other small vehicles to cover the sensor while parking.

**QR Code Scanner:** The QR Code Scanner serves as a straightforward way for customers to confirm their reservation without having to manually input the ID. The scanner will be a multi-purpose code scanner with a minimum reading range of one foot. The Scanner will be mounted next to the Elevator Console such that a user can easily hold up their phone or receipt to be read. The scanner will interface with the Elevator Console via a USB interface and will simply convey any scanned numbers to the existing system to check manually entered reservation numbers.

**Outside Display:** The Outside Display serves as a dynamic sign to represent the operating status and spot availability of the garage. The screen will be a 21:9, 1920x1080 weather-proof sign running a proprietary digital signage platform. Both the operating status and availability will be retrieved at a regular interval from the administrative panel and parking database. The Display will update over a LAN connection that can be wired or wireless.

**Centralized Server:** The centralized server will host the system controller that connects components. The server will be a Linux, Apache, MySQL, PHP platform running on a dual core processor with at least 4gb of RAM. The system will have a static IP and be housed at location off site with ports forwarded for respective services.

**Algorithms and Data Structures**

Software Simulation

**Algorithms:** To simulate realistic customer reservation and arrival, the software will employ two poisson processes. The arrivals will be governed by the following:

1. The database will be queried to verify availability. If spots are available, assign a spot randomly and mark it as occupied. If no parking spots are available, relay to the centralized server that an overbooked event has been registered.

2. To determine the next vehicle arrival time, generate a random number using the following equation when lambda is the average arrival rate and U is a random number between (0,1). This output will need to be scaled to obtain the next arrival time. For example: if rx = 0.3, then t(rx) = 0.3 60 minutes = 18 minutes.

**Data Structures:** The simulation will rely heavily on several objects that house data collected from various databases. The User object has a ArrayLists housing registered vehicles and active reservations.

The Garage object has ArrayLists for all currently parked vehicles, reservations and registered users.

Every program cycle, this information will be fetched from the respective database.

Account Management

**Algorithms:** There is no mathematical model included in this part.

**Data Structures:** The user information is stored in an Arraylist. The payment method and vehicle information are child classes of user information and they are saved in Arraylists as well.

Arrivals

One of the data structures used will be an array used to keep track of the state of every parking spot, specifically a bitmap vector. Each index in the bitmap, represents a spot in use or not in use. This is done for several reasons, mainly simplicity, easy reconstruction, and ease of use for the programmer. The biggest benefit is bitmap is an extremely efficient way to save space, not to mention it can be further broken up. However, if a garage is extremely large, then even the bitmap will fail however most garages are under 2000-3000 spots so overall, most garages this can work without further updates from programmers. Specifically, the algorithm assigned with this structure will be a quick and robust searching algorithm. It is linear search, and despite its efficiency; the number of times a spot will need to be found will always be at most few times per minute. Also, the algorithm is easy to code and will be less prone to bugs, errors, etc.

Garage Policies and Scheduling

**Algorithms:** There are no mathematical models in this section

**Data Structures:** In this section the manager will have access to the website and a database with all of the garage policies and scheduling information. This information will be stored in an Arraylist and the manager will be able to change the information.

Departure

**Algorithms:** Since the mathematical model of arrival for this system is a poisson distribution as mentioned in Report one. However, I am not apply and mathematical model for departure since we suppose that most of the customer will leave on time, that is to say , the schedule for departure is almost fixed by reservation. So we are not using any other algorithms for departure.

**Data Structures:** the information will be stored Arraylist since all previous part applied arraylist to store information and there will be not other complicated structure.

Statistics

**Algorithms:** The Reservation Statistics portion of the Parking Lot Automation system does not implement mathematical models from the Mathematical Model. Rather, the algorithms that govern are based on statistical analysis; requesting information from the garage database and producing models based on that information. At the user’s request, this system will generate a table, or graph displaying information including; the number of reservations, reservation frequency, concentration, account type (personal or business), and or length in any given time frame. The feature will also feature capability to determine most and least profitable day, and or the busiest and least busy days in a given time frame. The system will also be able to display statistical information regarding revenue from reservations, including; profits within a given time frame, and money from personal accounts and or business accounts, or overstay charges.

**Data Structures:** This Reservation Statistic portion of the Parking Lot Automation will organize relevant information into an ArrayList, which will be used in creating graphs.

Reservations

**Algorithms**: The part of the system that deals with reservations does not have anything to do with the mathematical models from report 1. Also, it does not use any complex algorithms. It will mainly extract information the user puts in the website and test and place it in another data structure alongside information from databases. The methods are as follows:

Normal Reservations - Based on the user’s login, the name will be extracted and placed in an array. Once the user clicks on the make reservations link, a schedule with empty and filled slots which will be held in a container will be displayed. Once the blocks are selected, that portion of the array’s contents will be extracted and made reserved, the block will be filled in. This will also be used to display the user’s reservation. His payment method will be extracted from the payment database and displayed alongside this. From the account database, if a business account is detected he will be discounted based on management discretion, a variable. A confirmation code of 6 numbers will be randomly generated using a random number generator (like the mersenne twister) and checked against a list of already existing numbers placed in a list. If it exists a new one will be generated until it is not repeated and then outputted to the user online. A QR code will be generated using a generator in the same manner.

Cancellations - Once the user logs into their account and clicks cancel reservation, the reservations made will be in a list and extracted for output to the user. The user can only select from these reservations to cancel. Once selected they may be charged a cancellation fee based on management discretion, a variable or fully refunded. The amount, calculated and extracted from the payment database will be displayed as a refund minus any cancellation fees.

Subscriptions - A subscription field will also display a similar schedule of blocks stored in an array as did the normal reservations part. Once the user clicks make a subscription they will be greeted to this. A list of frequencies (daily, weekly, or monthly) will be displayed. For these subscribers, one slot in the list of available slots will be filled in for the time available. The user will be displayed a monthly, weekly, or daily charge extracted from the payment database based on his selection. This depends on the management database as well and will be discounted if a business account is detected by an account type variable.

**Data Structures**: We will extract this information from the fields and place them in a *NESTED LIST*. This is useful because to make a reservations part of the data needed will require the above three variables. With a nested list, we can have the a list of a sets of lists which include these three variables. PHP has limited data structures available, namely arrays and lists, and I believe a nested list is the best way to go. SQL will be used as it goes hand in hand with PHP to get information from databases as needed.

**User Interface Design and Implementation**

Software Simulation

The simulation will rely solely on a command line interface for data representation and user interaction. Arrivals, new reservations and other data will be displayed as new lines in the command interface and at certain points, the user will have the opportunity to interact with the simulation.

Account Management

The account management part willrealize the functions ofaccount creation and vehicle and payment method edit. I will use php or java(base on the difficulty which they link to the mysql database.) to create a website which could reveal the process of user account creation and information edit.

Arrivals

A GUI will be needed for taking the user input's of confirmation code, and displaying their spot(with/out code). In addition, the GUI would be needed to take registered members using a walk-in reservation, who would enter a date and time ,and if needed, their membership number on a console. Hence the display would need to feed back their inputs. Lastly, there would be a display involving messages displaying the spot, or for an unregistered member to back away. The GUI main focus would be to be customer friendly and easy to use, rather than flashy.

Garage Policies and Scheduling

In this section as described before only the manager will have access to this database. He will be able to login to the website and there will be a database which stores all of the information regarding the garage policies and scheduling. This will include any specials or special dates. The interface will simply be a document style list of the policies and also contain a calendar with future events or specials.

Departure

There will not be too many interfaces interact with customer in this part. Upon departure, customer will receive a confirmation notification of their payment detail.

Statistics

The Statistics function of this Automated garage system will accessible via the system website. It will be implemented through php and java, and linked to a MySQL database. It will only be accessible through a manager’s account, and will reachable within one click after logging in. There have been no further design decisions departing from the mock up shown in Report #1.

Reservations

The initial screen mock up was changed slightly for the Reservation creation page. There is a table with with times and a way to change the dates easily. When a time slot is selected (by clicking and dragging), it displays in a list format next to the table and the user is allowed to select a payment method or add a new one once the total amount and savings are displayed. If not confirmed, the user will be greeted to a pop-up message to select another date/time, depending on the conflict in case multiple users are viewing the same page at the same time and one makes a reservation before another and the other users do not refresh. After that another screen is presented with a confirmation number and QR code once done and the user can proceed to do as they wish.

**Design of Tests**

Software Simulation

In essence, the simulation will be used to test the functionality and exchange of information between the other sections. For this reason, it is crucial that the simulation also be tested for accuracy.

Proposed Tests:

* Unit Tests will be developed for the following classes: User, Garage, Reservation, Database Interface and Vehicle. The unit tests will exercise the contained data structures and methods to verify the output matches the expected result.
* Integration testing the simulation is a key aspect of verifying its interaction with other aspects of the system. For every element of the system that the simulation interacts with, a test will be designed to verify that given a set of inputs, the expected output matches the real results. An example of this case is new reservation creation. The simulation is designed as a standalone system, so a new reservation object will be randomly generated and assigned to a user’s account. When the “New Reservation” portion of the system is also online, the simulation will automate the process of the reservation creation and register a new reservation based upon the object.
* Since this application relies heavily on multiple databases, it is important that the simulation accurately read and write from these tables. Tests will need to verify that information stored within objects can be accurately transferred to appropriate tables and read again with no error.

Account Management

* Username has already existed.
* Create an account and use it to log in.
* The process of how user reset their passwords.
* Add/Delete the vehicle and payment information.

There isn’t a program which specifically mimics those situations. I will show those situations when i present my part.

Arrivals

Testing, the software will be composed of 3 main sets of tests. First set will involve, cars that have registration and booked a reservation for this time x, and date t. Second set will involve cars that and need to enter a confirmation number or show a QR scan, for time x and date t. 3rd will involve cars that need to go to walk in, hence all free walk in spots need to be properly displayed. 4th will be registered members who are attempting walk-in reservation, and are indeed registered members(either license plate or membership number). 5th will be same as 4th except, they aren't registered. Each of these sets, will involve testing to see expected vs actual, this will be done through 10-20 trials to properly stress test the algorithms, and data structures. For the integrated testing, without question, departure, reservations, and arrival would have to work together and be closely integrated due to the massive overlaps.

Garage Policies and Scheduling

**sign of Tests:**

1. The manager will be able to access the website and test its effectiveness.
2. The test for this section will simply identify any issues within the website and database for policies and scheduling.
3. The manager will test the website and database regularly to determine any errors or uncertainties.

Departure

(a) The customer will be charged automatically as soon as leaving the garage

(b) As certain vehicle leave the garage, the database give out proper customer information

(c) understay will be fully charged, overstay will be charged more due to parking policy

(d) If the payment method is available, send a request to banksystem

(e) customer will receive a confirmation notification about payment and parking detail

Statistics

1. Test Cases
   1. Can the manager access the Statistics page on the website
   2. Can the program identify the best and worst hour, in terms of profit, on a given day
   3. Can the program identify the best and worst days, in terms of profit, a given week(s)/month(s)/year
   4. Can the program identify the best and worst hour, in terms of number of reservations, on a given day
   5. Can the program identify the best and worst days, in terms of number of reservations, a given week(s)/month(s)/year
2. Test Coverage: The above test cases will cover all code within this portion of this section of the Garage Automation System.

Reservations

The test cases programmed and used to test the reservations part of the system will be the following:

a. Test if a successful reservation can be made.

b. Test if a QR code and confirmation number is generated.

c. Test if the selected time slot was booked.

d. Test if a cancellation is done successfully.

e. Test if account type is detected.

f. Test if subscriptions work properly.

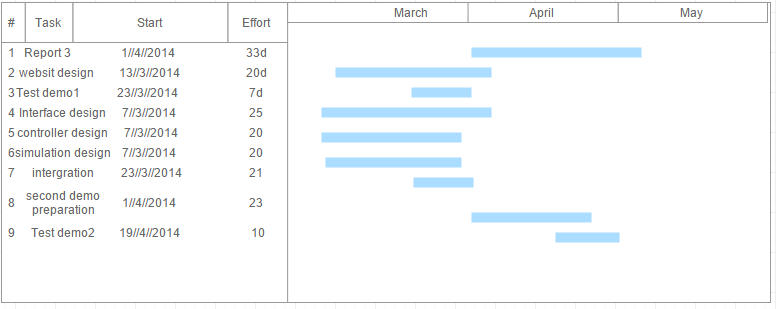
g.Test the amounts displayed are correct.

h. Test if payment information is transmitted to correct databases.

The integration testing coverage consists of collecting the tests of each unit and making sure they work.

**Project Management and Plan of Work**

Plan of Work



Breakdown of Responsibility

Basically we divided the project into 7 mini projects and everyone take the responsibility of one part

|  |  |  |
| --- | --- | --- |
| Group member name | Responsibility | Content |
| Brandon Dunlap | Statistics, Architectural Style | Statistics tool for management |
| Yufeng Liu | Account Management | Registration, add/delete vehicle and payment information. |
| Luke Miller | Hardware Requirements & Data Structures, Algorithms, UI and Testing for Software Simulation | Development of software Simulation for Park-A-Lot Garage - I detailed the components involved and how they will interface with other project sections. |
| Harshil Patel | Network Protocol & Algorithms, Data Structures, Testing, and UI for Reservations | The process of making, cancelling, and subscribing reservations accompanied by detailed explanations. |
| Thomas Walters | Garage Policies and Scheduling | This section will consist of a website and database that a manager will be able to edit all policies and specials involved with the garage. |
| Xiang Xing | Departure | payment , billing , overstay charging |
| Vikram Krishna | Arrival | The process of cars arriving for either walk in, or registration, the process of picking a spot also |

**References**

Freeman, Eric, Elisabeth Robson, Kathy Sierra, and Bert Bates. *Head First Design Patterns*. Sebastopol, CA: O'Reilly, 2004. Print.

Marsic, Ivan. "Software Engineering Project Report." *- Requirements*. Web. 21 Feb. 2016.

Marsic, Ivan. "Software Engineering." *Ece.rutgers.edu*. Web.

Marsic, Ivan. "1. Project Description." *Software Engineering ProjectParking Garage/Lot*. Web. 21 Feb. 2016.

"Parking Garage." *Parking Garage*. Web. 21 Feb. 2016. <https://sites.google.com/site/parkinggaragesp13/home>.

Pilone, Dan, and Russ Miles. *Head First Software Development*. Sebastopol, CA: O'Reilly, 2008. Print.