LE\EECS3311 Z

Winter 2020-2021

# Final Project Report

Full name: Gleb Ignatski

Student ID: 215536667

Login ID: gleb99

April 15, 2021

## Table of Contents

## Introduction

I would like to start off by commenting on the major differences between my previous design and the current one. Prior to starting the design diagram during the midterm, I had very little knowledge and experience with GUI. I ended up only including the JButton fields in the gui package of the design diagram, without taking into consideration the essential components JTextField, JLabel, JFrame, JPanel, and JTextArea. I also had not considered using methods in the GUI class to transition between each window. As a result, this project helped me learn and develop graphic user interfaces skills at the front and back-end level as well as improve my problem-solving skills.

Moreover, I realized the importance of using text/csv files to store the parking lot data as I had previously considered storing the data in memory using a HashMap. This allowed my application to be interactive and dynamic rather than dissociated and static. Not only that, but I did not organize my Java files in relevant packages and consequently missed an important feature which is that the system administrator just like the officer and the client must log in to the system to manage administrative tasks. I also did not consider the administrative work that an officer could do besides the system administrator.

Furthermore, I added additional features to the application. One was the date of birth verification feature, which does not allow a customer under 16 years of age to register in the system. The date, such as "04/09/1999" that the user was born is subtracted from the current date and is verified. Another feature was encryption. I decided to store the card information in Card.card encrypted, which upon system access to deduct user balance after paying gets decrypted and encrypted automatically back after the card balance update. More information on this can be found below in the *Running/Testing the Application* heading.

## Running/Testing the Application

To execute the application, go to the GUI.java class in the gui package and run the program. The window that opens has two fields (username/email and password) in which you can enter the login information. You would then locate the corresponding account type button under the login fields (customer, officer, or system) and click it. An account to try for a **customer** is (user = gi@y.ca, pass = gi5555), (user = ab4, pass = 73!) for an **officer**, and (user = ab10, pass = 4321) for an **administrator**.

Additionally, if you would like to refer to the Card.card file, it is encrypted. To decrypt it, run the XOREncryption.java class located in the payment package. The main method will encrypt/decrypt the file. It is also important to encrypt it back (run the class again) after looking at it as the pay methods will not be able to read it if it is not encrypted. To test the application, run the Junit test in the testing package Tests.Java. There are 60 tests.

The coverage compiled by Jacoco was not very good. I struggled to get more than 50% coverage on average per non-GUI class. I was able to test the application thoroughly and get good results, but the coverage ended up being very low as a result of many auxiliary (helper, extra testing) methods I added. I believe that in the next project, I have to find ways to cut down on lower-level methods as well as accomplish tasks by integrating key methods, not creating auxiliary ones.

## Design Patterns

I was able to further comprehend the observer pattern and why it fits in with this application. Previously, I thought I was using the observer pattern when working on the midterm, but it ended up being an overly complicated implementation of the singleton pattern. This is because the users of the system are in fact the observers who “observe” the subjects, which are the corresponding databases. The dependence mechanism “change-update” was more prevalent and noticeable than in the midterm because of testing on top of the design that I was not able to do during the midterm. For example, when a customer books a parking space, an object-oriented update to the ParkingSpaces.txt database takes place along with the BookedSpaces.txt.

The objects used to display parking data in Order.java are automatically updated by another object in the SignIN.java class to enhance the user’s experience and be able to see the changes they have made to their order list as well as what is available after they pay. Another example would be when a parking officer observes the requests made by the client and when he or she intervenes and grants the request, the client is notified of an available space to select. The parking officer depends on the database, which is a subject since it frequently updates according to its subject’s state. Another example would be a system feature observing the client’s database for duplicate emails upon the client signing up.

Another major difference was with the payment procedure. I had not considered that the user needs to have the balance on their card updated (an amount deducted) after each transaction. Since I read over the instructions this time more carefully prior to coding, I also realized that I did not have an option in my design diagram that would enable the user to pay for one of multiple parking spots. I took this into account and created two methods: pay(String, Card, String) and payTotal(String, Card, String) that would handle the user payment requests.

## System Features

### 4.1 Manage Parking Enforcement Officers

Upon registration of an officer, the first name, surname, username, email address, and password are prompted. But it is not the officer that signs up themselves; the system administrator has the authority to add officer data to the database as well as ask the officer to come up with a password. The system assigns a unique identification to the officer because the officer uses their username, not their email to log in. The identification is unique because it does not only depend on the name, surname, and email address, but also on the number of officers in the system. If there are currently eight officers signed up, and the new officer registering has the name Bob Smith with the email address bobsmith@yahoo.com, then the generated ID will be "bs9". This way if there was another Bob Smith, the username would not match this one since the previous Bob Smith got registered earlier and has username bs1-bs8. Additionally, there is no need to decrease the variable storing the number of officers in the file OfficerID.txt upon removal of an officer as it will prevent a username collision.

When removing an officer, the administrator enters the unique username and presses the "remove officer" button, after which a message will be displayed on the screen that the officer has been successfully removed. If the officer does not exist with the entered username, then an error message will be returned. On the contrary, the same procedure is put in place for adding an officer with the only difference being that this officer does not already exist in the system. This will be done by checking the email address since two Bob Smiths' with the same email address having different username raises a red flag and is an indication that there are two accounts linked to the same officer. All the parking officer information is stored in the Database_officers.txt file.

### 4.2 Customer Registration

When a customer is registering, they are prompted for their first name, surname, date of birth, email address and password. To sign up, they must be at least 16 years old. It is also crucial that they stick to the date format specified near the entry filed ("dd/mm/yyyy"). If the date format is incorrect, or they entered an invalid date (e.g., December 32, February 29 when it is not leap year), they will need to resubmit.The system also checks the number of characters they entered on each line (first name and last name need to be at least two characters long, email address is at least four characters, and the password is at least 6 characters. The system will accept duplicate entries provided that the emails do not match. A corresponding error will be provided in case the email entered already exists in the database. The customer data are stored in Database_customers.txt.

## 4.3 User Login

The first page of the application lets the user sign in. They must type in their email address and password, then click "Sign in as Customer". An error will be displayed if the database does not contain this user or if there is an error with the credentials. Once the application verifies that the email address and password match what is in the database, the customer will be redirected to a separate window where they can reserve, cancel, request, view and book their parking space(s).

## 4.4 Book a Parking Space

For this part of the application, extra classes needed to be created for user convenience. First, prior to booking a parking space, the system verifies that the user is in fact logged in. a Boolean variable is present to either allow the user to proceed or reject them from booking a parking space. For customer convenience, at most 25 free parking spaces will be displayed from which a customer can choose from. A text area below the prompt will be visible to the user so that they know what kind of identification is required. It will not be the case that the user will be told that the parking space they are selecting is occupied, since the system will always present those that are currently available. Once the customer enters the booking ID, and presses the "Book Space" button, they will be redirected to a new window where they can enter the start and finish times. They can also request a parking space not in the available list which simply adds the parking space to the requested database that the officer has the authority to manage. The database is stored in a RequestedSpaces.txt file.

The date is in the "dd/mm/yyyy" format while the start and finish times are in the 24-hour format (e.g., 06:30, 17:30, up until 23:59). A helper TimeBooked.java class is used to pass and convert the times entered as well as calculate intervals. The times available to book are between 4 AM and 23:59 (20-hour window). Once the customer enters the times and click on “Book Space ##” (dynamic button), then they can check their booking details by clicking “View Bookings”. There is a text area that displays user relevant parking details upon activation.

The ParkingSpaces.txt database stores the parking space IDs as well as whether a verification code has been assigned to the parking space. If in place of the verification code there is a “0”, then the space is vacant. Otherwise, a 13-14-digit code is assigned that is determined by adding the four-digit parking space identification code with the hash values of the start and finish times selected. This is the code the user will enter upon paying/canceling their space.

Another key aspect of the system is prohibiting users from booking more than three spaces. This is done by counting the number of booked spaces associated with the user’s email address in the BookedSpaces.txt database. An error message is also returned to the user informing them that they cannot book another parking space. The users also can view their booking to keep track of how many more spaces they can book. Additionally, a parking space identification code is the unique 13-14-digit code mentioned above.

### 4.5 Cancel a Parking Space

A customer can cancel their ordered parking space by clicking the cancel button after passing a valid parking space identification that they were given when they first booked their spot. The customer must also be signed into the system to execute this function. The customer can check whether their parking space was successfully removed after cancellation by clicking on the “View Bookings” button. The customer must also keep in mind that they can only cancel their parking space prior to the start of booking time.

### 4.6 Payment

The customer is presented with the “PAY” button and a field to enter the parking space identification code (13–14-digit one) which they could simply copy/paste from the text area after clicking “View bookings”. They can also pay for all the booked space at once using the “Pay for all” button. The customer must also be logged in as usual to proceed with payment. In the text area that displays booking details, the total price for each booked space as well as the overall total if there are two or three spaces booked will be displayed.

Once they are redirected to the payment window, they need to fill out their credit/debit card details. These details are the type of card, card number, expiry date, first name, surname, and security code. The database Card.card will be accessed to verify all the details passed by the customer. The Card.card file is encrypted with the special security code using XOR encryption that the user received. You can enter (Visa, 2410024527339075, 08/24, Ben, Gibbin, 4286) into the fields in that order. The Card.card file not only includes the essential details that the user has to enter to pay but also the amount of money available. Once the fields are If the transaction is successful, the BookedSpaces.txt database will be updated by having the transaction removed. The user can also go back to the previous window in case they would like to book/cancel a spot or would like to review the booking details.

## 4.7 View Bookings

After the customers log in, they are redirected to the main windows where all the booking procedures get executed which includes the "View Bookings" button. Once the user presses this button, they can see line by line each booked spot and its associated details. The account with credentials (gi@y.ca, gi5555) currently has two spaces booked. The details include the full booking ID, the start time, finish time, and the cost. The cost for one hour is $1.80 in the system. The officer can also view booking details along with the associated email appearing prior to all other details mentioned above.

## 4.8 Manage Parking Spaces

When an officer logs in, they will click a button "Manage Parking Spaces" and will be redirected to another window where they can observe the ongoing requests. They are displayed with the emails of the users who booked parking spaces and the associated details. There will be four buttons: "Check Requests with email:", "Add Space", "Remove Space", "Cancel Request", "Grant Request", and "Sign out". They are also presented with a list of emails, since they must enter them to check if the customer with the associated email has made any requests. The officer will check whether the time requested for the currently unavailable parking space is valid and then remove the request with the associated email, while also notifying the customer with an email that they can book the spot they requested. If it is not valid, then the officer will press the “Cancel Request” button and an email will be sent to the customer informing them that the request is still denied. Other officer administrative functions such as “Add space” and “Remove space” are for managing the ParkingSpaces.txt database. The “Add space” button will make the specified parking space available by having a “0” replace the current hash value in its place.The “Remove space” button assigns “Unavailable” to the parking spot entered in the ParkingSpaces.txt file.

## 4.9 Change Payment Status

For this feature, the successful payment status is indicated by the “Success!” message printed on the screen . The system completes this step automatically after there is confirmation of a successful payment made by the customer. The Card.card database will get updated before the message is displayed.