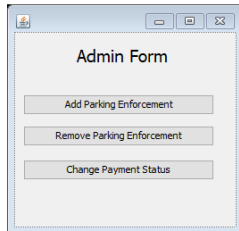
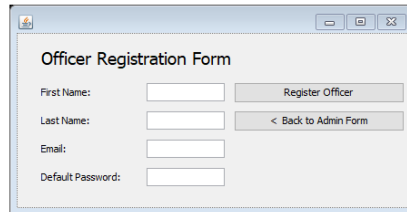
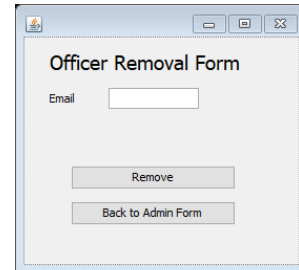


Requirements Satisfied with proof and comments:

4.1 Manage Parking Enforcement Officer

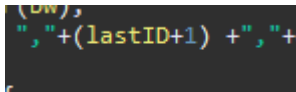
A screenshot of a web application window titled "Admin Form". It contains three buttons: "Add Parking Enforcement", "Remove Parking Enforcement", and "Change Payment Status".A screenshot of a web application window titled "Officer Registration Form". It contains input fields for "First Name:", "Last Name:", "Email:", and "Default Password:". There are two buttons: "Register Officer" and "< Back to Admin Form".A screenshot of a web application window titled "Officer Removal Form". It contains an input field for "Email:". There are two buttons: "Remove" and "Back to Admin Form".

4.1.1: Here we have the admin form that can be obtained by entering the right login. In the admin form you have the options to manage the parking enforcement officer. That is, add or remove the officer when you press the button add or remove parking enforcement.

4.1.2:

1. The system will ask information such as first name, last name, and email address is required to add a new Parking Enforcement Officer
2. A Parking Enforcement Officer can be removed by the administrator if their email address exists in the system

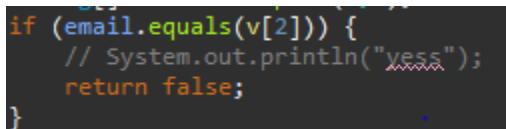
4.1.3-REQ-1: The system must assign a unique ID to each new parking enforcement officer added



```
(ID),  
", "+(lastID+1) +", "+
```

Whenever a new parking officer is created a unique ID is generated in the CSV database that goes by the name of "user.csv".

4.1.3-REQ-2: The system must verify the parking enforcement officer exists in the system before removing an officer



```
if (email.equals(v[2])) {  
    // System.out.println("yess");  
    return false;  
}
```

In the officer Registration method it checks whether the csv contains an user with the same email which should be unique if the officer user email does happen to exists it will return false and officer will not be added.

4.1.3-REQ-3: The system must verify a new parking enforcement officer's ID does not exist in the system already

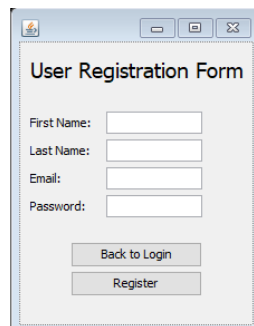
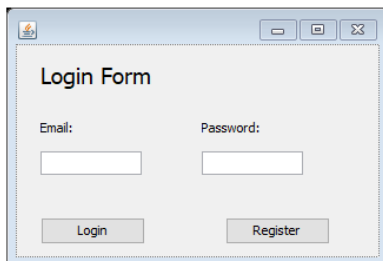
The system automatically generates the id based on the id already in the current database. It does this by iterating the id number by one each time it has been added.

4.1.3-REQ-4: The system must store a new parking enforcement officer's registration information

```
13 robo cop,12,robo@toof.com,of1,officer
14 yak yok,13,yak@yok,yak123,officer
```

The system automatically adds the officer's information to a csv and it removes from the system as can be seen above.

4.2 Customer Registration



4.2.1: Here is the Office registration form it can be obtained by pressing the register button on the login screen the user has to enter all of the different fields and he will be displayed with a success message if the user has successfully registered otherwise he will be displayed a warning message that the email currently exists.

4.2.2: New customers will be displayed with a button such as "REGISTER", leading to a separate window, where once clicked, the customer can enter their first name, last name and email along with a password.

4.2.3-REQ-1: The system must accept entries with duplicate first and last names

The system does this automatically allow this BUT the email must be unique.

4.2.3-REQ-2: The system must allow unique email addresses when registering

```
boolean x= cs.customerRegistrar(fname.getText(), lname.getText(), email.getText(), pass
if(x==false) {
    warning.setForeground(color.RED);
    warning.setText("Customer email is already in use");
}
```

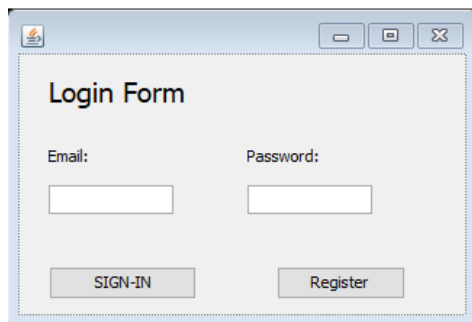
If the user enter an email that is already in use the user will be warned by the system that the email is actually in use.

4.2.3-REQ-3: The system must store a new user's login information for future authentication purposes

```
t4,4,t4@yorku.ca,t4t4,user  
t5,5,t5@yorku.ca,t5t5,user
```

The users information will be stored along with a reminder that he is just a plain user and not an officer or an admin.

4.3 User Login



4.3.1: Here is the login form which allows customer, admin and parking enforcement officer to enter their login information. If the email is associated with admin it will lead to admin form. If the email is associated with officer it will lead to officer form. If the email is associated with standard user it will lead to

4.3.2: Users will be displayed with a button such as "SIGN-IN", leading to a separate window, where once.

4.3.3-REQ-1: A user must be registered or added before logging in

```
customer= cs.isCustomer(email, password);  
admin = cs.isAdmin(email, password);  
officer = cs.isOfficer(email, password);
```

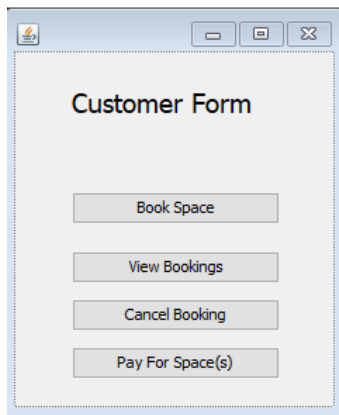
The email is checked as well as the password in the CSV database it also checks if the user is a customer or administrator or an officer to decide which from to give them.

4.3.3-REQ-2: The system must authenticate a user's login information before granting access to the application

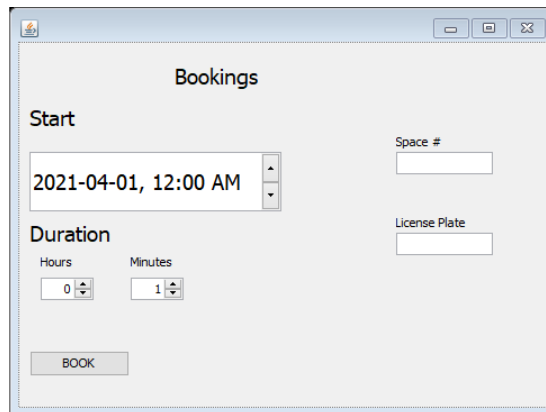
```
customer= cs.isCustomer(email, password);  
admin = cs.isAdmin(email, password);  
officer = cs.isOfficer(email, password);
```

The email is checked as well as the password in the CSV database it also checks if the user is a customer or administrator or an officer to decide which from to give them.

4.4 Book a Parking Space



A screenshot of a software window titled "Customer Form". It contains four buttons stacked vertically: "Book Space", "View Bookings", "Cancel Booking", and "Pay For Space(s)".



A screenshot of a software window titled "Bookings". It contains a "Start" section with a date and time picker showing "2021-04-01, 12:00 AM". To the right is a "Space #" input field. Below the start section is a "Duration" section with "Hours" and "Minutes" spinners, showing "0" and "1" respectively. To the right is a "License Plate" input field. At the bottom is a "BOOK" button.

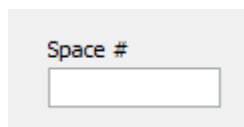
4.4.1: The user clicks a book a space and it leads to a booking form. Where the user can enter various information in regards to booking. The user must enter a date after current date and a certain amount of time. They must also enter space and license plate.

4.4.2: The customer will be displayed with a button such as "BOOK SPACE", leading to a separate window, where once clicked, the customer is prompted to enter the parking space number, booking time, and their license plate number.

4.4.3-REQ-1: Customer must be registered and logged-in before booking a parking space

This requirement is met because the user cannot access the customer form without having to login via the login form.

4.4.3-REQ-2: Customer must select which parking space they are booking



A close-up screenshot of a text input field with the label "Space #" above it.

The customer will have to select which space they want if they are booking a space in the booking form.

4.4.3-REQ-3: If the customer selects a parking space which is occupied, they are presented with an error message and must select a new space

```
boolean x1 = pv.isConflict(date1, (int) spinner.getValue(), (int) spinner_1.getValue(), textField.getText());
```

If the space is occupied cannot be added because there will be a conflict. Date is also taken into consideration in this case.

4.4.3-REQ-4: Customer must enter how long they want to book said parking space for



A form titled "Duration" with two spinners. The "Hours" spinner is set to 0 and the "Minutes" spinner is set to 1.

This shows the duration that users can book for. The minimum is 1 minute. The duration time is stored in the "bookings.csv".

4.4.3-REQ-5: The system allows a customer to book up to three parking spaces

```
public boolean ruleOfThree(String email) {  
    int count=0;
```

This method returns a Boolean value if the number of bookings for a particular email associated with a user is not less than 3.

4.4.3-REQ-6: Each booked parking space receives a unique booking ID

```
" " + (lastID + 1) + "  
+ " " + "unconfirmed"  
email +
```

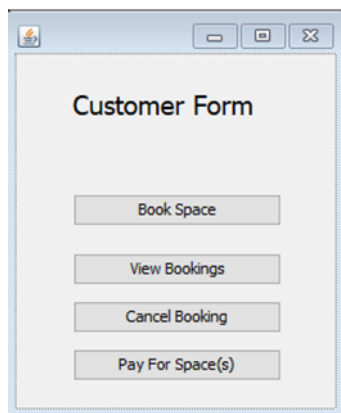
Whenever a new booking is created a unique ID is generated in the CSV database that goes by the name of "booking.csv".

4.4.3-REQ-7: The system must display an error message if the parking space the customer booked is occupied in the system

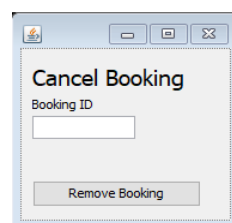
```
// System.out.println(date1);  
boolean x1 = pv.isConflict(date1, (int) spinner.getValue(), (int) spinner_1.getValue(), textField.getText());  
//System.out.println(x1);
```

This checks if the user can add a booking it takes the time duration and space number as values.

4.5 Cancel Parking Space



A window titled "Customer Form" with four buttons: "Book Space", "View Bookings", "Cancel Booking", and "Pay For Space(s)".



A window titled "Cancel Booking" with a text field labeled "Booking ID" and a button labeled "Remove Booking".

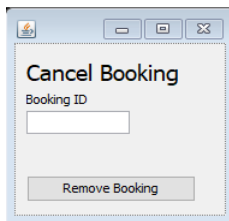
4.5.1: The user clicks cancel booking space and they are taken to a form that contains a textfield to enter the booking ID and remove it.

4.5.2: The customer clicks on a button labeled “Cancel Booking” and enters their booking number and then presses remove booking.

4.5.3-REQ-1: Customer must be registered and logged-in before cancelling a parking space

This is given taking into consideration the only way they could access the cancel spot would be through the login screen.

4.5.3-REQ-2: Customer must enter the booking ID which is associated with their name



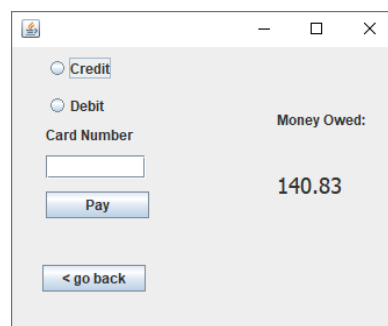
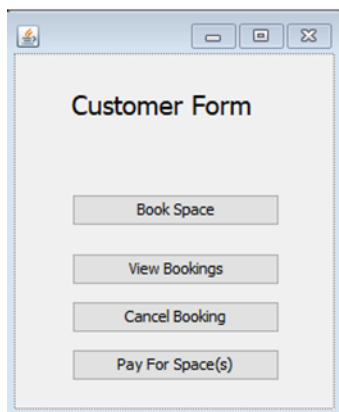
The booking id is entered and associated with the user email rather than their name due to issues regarding duplicate first and last names.

4.5.3-REQ-3: The cancellation can only go through if the time of cancellation is before the booking expiry time

```
if (!(bookingid.equals(v[3])&&email.equals(v[6])&&now.isBefore(date23))) {  
    // System.out.println("pass");  
}
```

The removebooking () method checks if there is a conflict with the current time: now.isbefore(date23).

4.6 Payment



4.6.1: The user clicks pay for spaces and he is taken to a form where he can pay for all of his spaces cumulatively.

4.6.2: The customer clicks on a button labeled “Pay” he then pays the total amount for his reservations and the money owed field is updated.

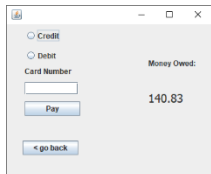
4.6.3-REQ-1: Customer must be registered and logged-in before making a payment

This is given taking into consideration the only way they could access the payment would be through the login screen.

4.6.3-REQ-2: Customer must have entered additional information such as which parking space they are booking before making payment

Yes it appears that this requirement is statisfied.

4.6.3-REQ-3: The system must accept different forms of payment (ex: Paypal, credit, debit, etc.)



As can be seen above there able to pay and must check mark credit or debit. Anything can be the card number considering the scope of the project the card number is the banks issue .

4.6.3-REQ-4: The system must automatically timestamp each payment confirmation

```
pv.paystamp(now,1);
```

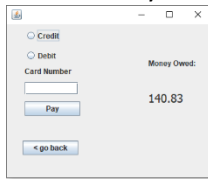
This method pay-stamps the current payment if all requirements are true by adding a record containing the time and the amount paid aswell as the time in a “payconfirmed.csv” .

4.6.3-REQ-5: The system automatically starts a countdown till expiry once payment is confirmed

```
1 2021-04-04T09:50,3,  
2 2021-04-09T15:50,6,  
3 2021-04-03T09:50,1,  
4 2021-04-05T09:50,2,  
5 2021-04-03T09:50,3,
```

The system stores all of the booking start and end time information in a booking.csv file and as such it the system is designed around this information.

4.6.3-REQ-6: The system allows a customer to pay for multiple parking space bookings (if any) cumulatively

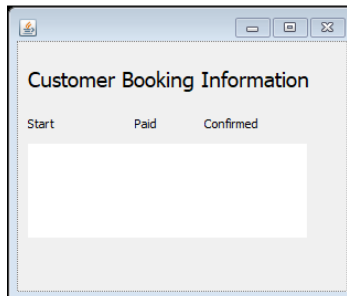
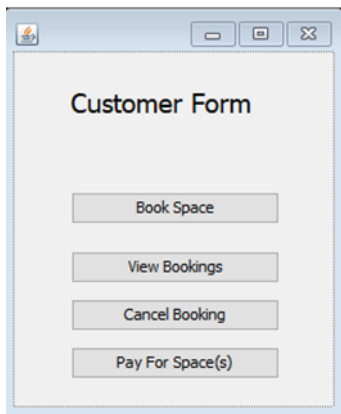


The system allows user to pay for all of his current bookings cumulatively automatically. This is due to a loop that marks the payment status of the booking as confirmed.

4.6 .3-REQ-7: The system must authenticate customer payment information before proceeding with confirmation of parking space

Since banks are not taken into consideration in the beta phase of this software any car information will automatically be considered valid.

4.7 View Bookings



4.7.1: This feature shows a customer's current parking booking. Information such as expiry time and payment status can be viewed.

4.7.2:

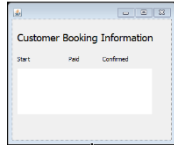
- After logging in, customers will be displayed with a button labeled "VIEW BOOKINGS", leading to a separate window displaying all their bookings.
- Customers can select and open any of their current booking to display additional information like expiry time and payment status
- Parking enforcement officers can enter customer information and view their booking details

Ahmad Baig
21580870

4.7.3-REQ-1: Only authorized users, such as parking enforcement officers , can view any customer's booking details

This is true for all of the parking information.

4.7.3-REQ-2: Customers can view their parking space booking information, including expiry time

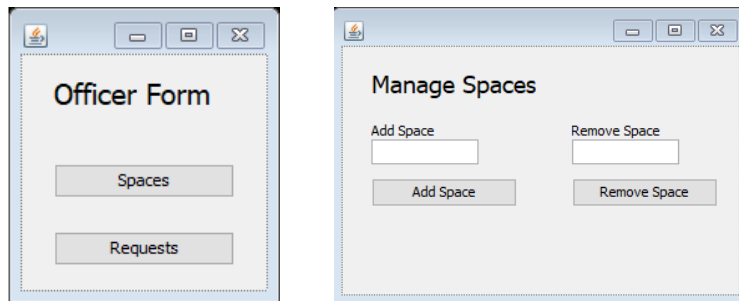


The user can see all the important booking information and EXPIRY TIME: is the same as start time.

4.7.3-REQ-3: The system must notify the customer when their parking space booking is expired

The parking space is going to be removed from viewbookings when the time is up.

4.8 Manage Parking Spaces



4.8.1: This feature allows the parking enforcement officer to manage parking spaces by adding or removing them.

4.8.2:

- Authorized parking enforcement officers will be displayed a button labeled "MANAGE PARKING", leading to a separate window displaying all parking spaces.
- Parking enforcement officers then can select any parking space and view information such as customer space requests and current user booking (if any)
- With buttons such as "ADD SPACE", "REMOVE SPACE", "CANCEL REQUEST", "GRANT REQUEST", parking enforcement officers can add, remove, cancel, and grant requests respectively

4.8.3-REQ-1: The user must be an authenticated parking enforcement officer.

This is given because user has to login as parking enforcement officer in order to access the parking enforcement officer form.

Ahmad Baig
21580870

4.8.3-REQ-2: Parking enforcement officers must verify a parking space is vacant before removing it from the system

```
if(s.isOccupied(textField_1.getText())) {
```

The system check if a space is occupied if it is it will warn the officer that it cannot remove space. Otherwise it will remove the space.

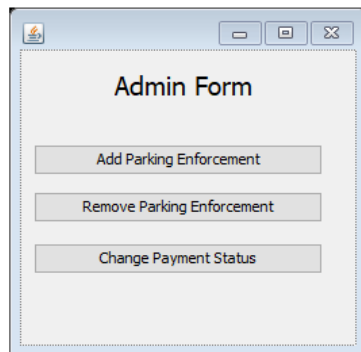
4.8.3-REQ-3: The system must have a minimum of one parking space

This will always be enforced throughout the system.

8.3-REQ-4: Parking enforcement officers must verify requested parking space is vacant in the system before granting requests

According to slack this requirement has been cancelled.

4.9 Manage Parking Enforcement Officer

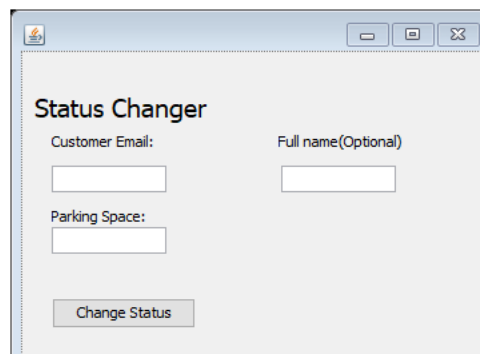


Admin Form

Add Parking Enforcement

Remove Parking Enforcement

Change Payment Status



Status Changer

Customer Email: Full name(Optional)

Parking Space:

Change Status

```
u.ca, unpaid, confirmed  
aid, unconfirmed  
g, unpaid, confirmed  
k, paid, unconfirmed  
ca, unpaid, unconfirmed  
ca, paid, unconfirmed  
unpaid, unconfirmed  
u.ca, unpaid, unconfirmed  
ca, unpaid, unconfirmed  
ca, unpaid, unconfirmed
```

4.9.1: This feature allows the systems administrator to update customers' payment status. The status is updated depending on the parking space and the email. The name is optional considering that the name can contain duplicates.

4.9.2: System administrator will be displayed with a button labeled "CHANGE STATUS", leading to a separate window, where they can enter customer first name, last name, email address and parking space number to change their payment status to "paid".

4.9.3-REQ-1: The user must be a systems administrator.

This is a given taking into consideration that they must login in order to access the admin forms.

4.9.3-REQ-2: The system must verify the customer's existence before changing their payment status.

This is true because in the booking and payment status are stored with the customer's email and this allows them to change the status whilst simultaneously having their actions verified.









4.9.3-REQ-3: The system must verify the customer's occupancy of the parking space before changing their payment status

This is up to the parking enforcement officer there is no way for us to examine real world events.

4.9.3-REQ-4: The system must verify the customer's payment of the parking space before changing their payment status

The system does this by checking payment status in the csv.

Use Jacoco to measure the code coverage of the test cases and show the coverage graph

csvStuff		54.1 %
▸ ParkingCSV.java		47.5 %
▸ CSVManager.java		57.4 %
▸ Spaces.java		79.7 %
test		96.6 %
▸ york.eecs.test		96.6 %
▸ testSpaces.java		88.0 %
▸ testCSVManager.java		100.0 %
▸ testParkingCSV.java		100.0 %

This is the coverage Graph for the test cases on the non GUI classes. These test cases were all done on the CSV classes because the CSV classes were all the non-GUI classes that I had.

IMPORTANT NOTE: It is important to mention that not all test cases reached 80% that is because most of the code in the CSV classes are designed for handling exceptions and/or adding and removing records and I cannot add and remove records without potentially damaging my CSV file with the dummy records.

Remarks and Difference Between Midterm Design and Final Project Design

Original Pattern (Builder):

The original pattern used during the midterm was builder and state for the login. This pattern was used with the idea of “building gui’s” in mind and the state pattern was designed to define context based on the login information.

New Pattern (Singleton):

The main pattern that was obtained was the singleton pattern. The singleton pattern was used because we only need one customer object to be handed around during our various activities on the customer form. Another pattern used was the filter pattern this was used in place of state for the login.

Comment on Differences:

The main difference is that in the original design during the midterm and the new design is that in this design the design pattern used was singleton and filter. Whereas, In the old(MIDTERM) design the builder and state pattern were used. The Builder patterns idea of dynamically building gui’s became difficult to implement. It was a much better idea to already create the guis and use design pattern singleton. The singleton design pattern was used because there was a need to maintain a single customer object while moving between various gui’s and forms.