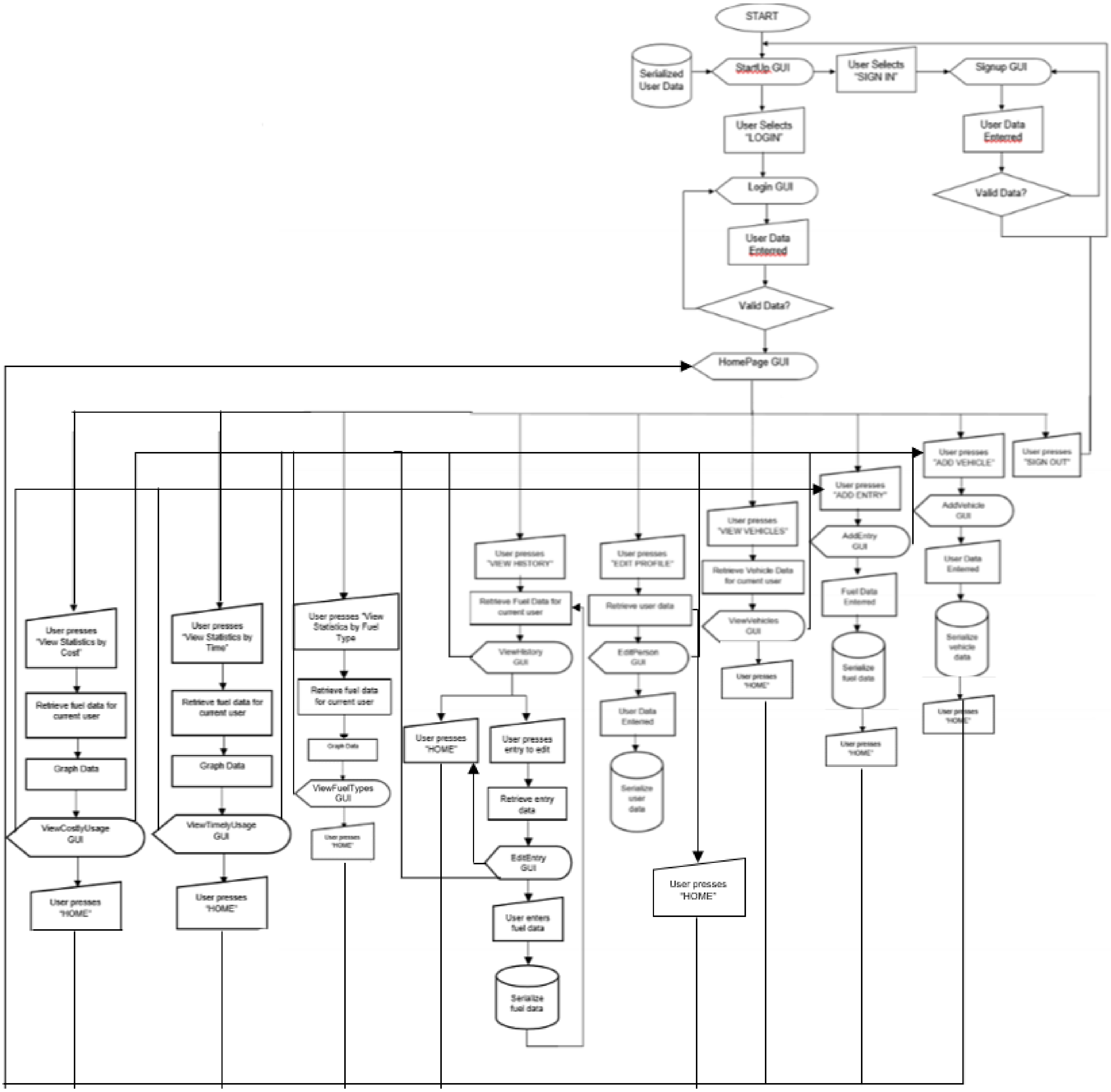
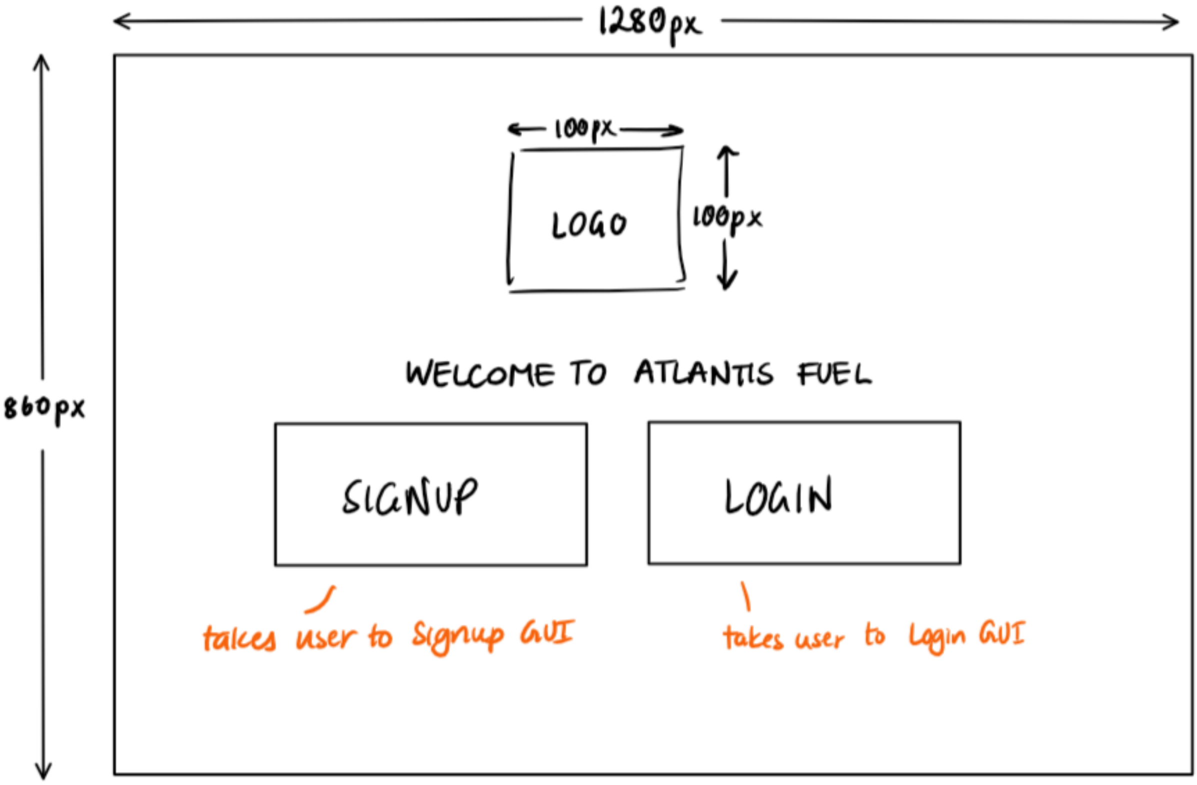
Table of Contents

System Overview

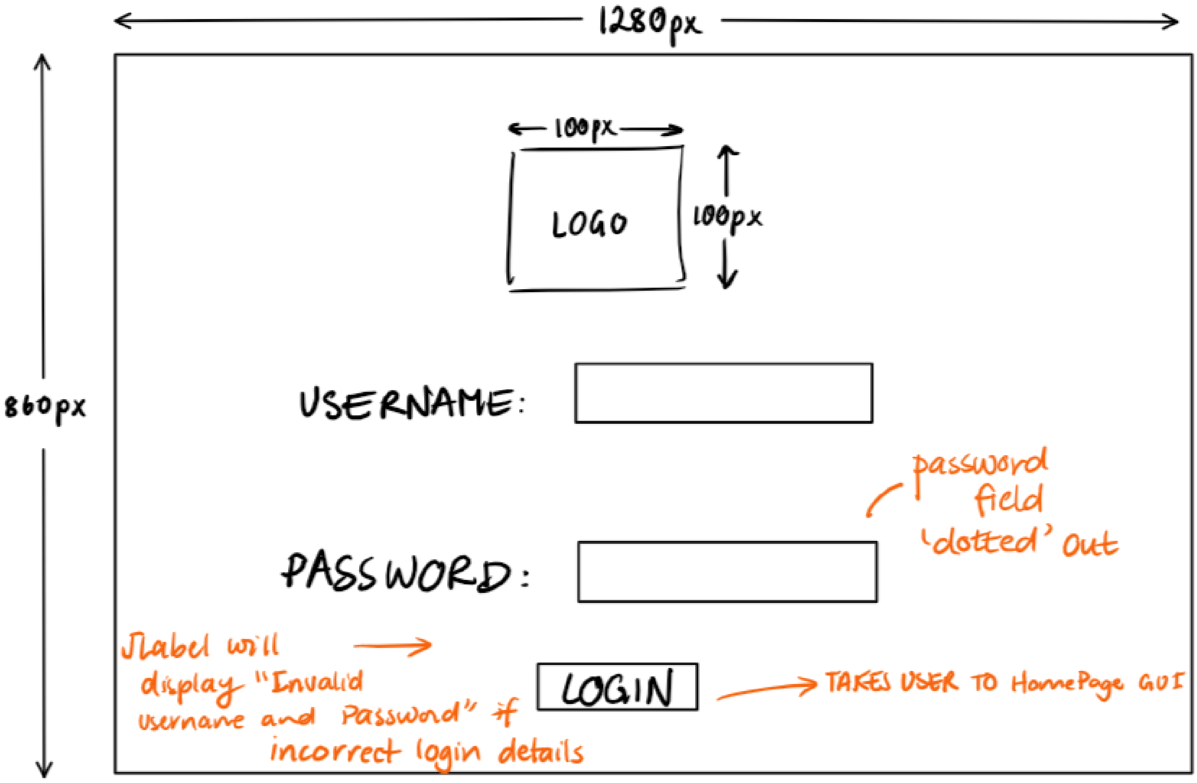


Graphical User Interface Designs

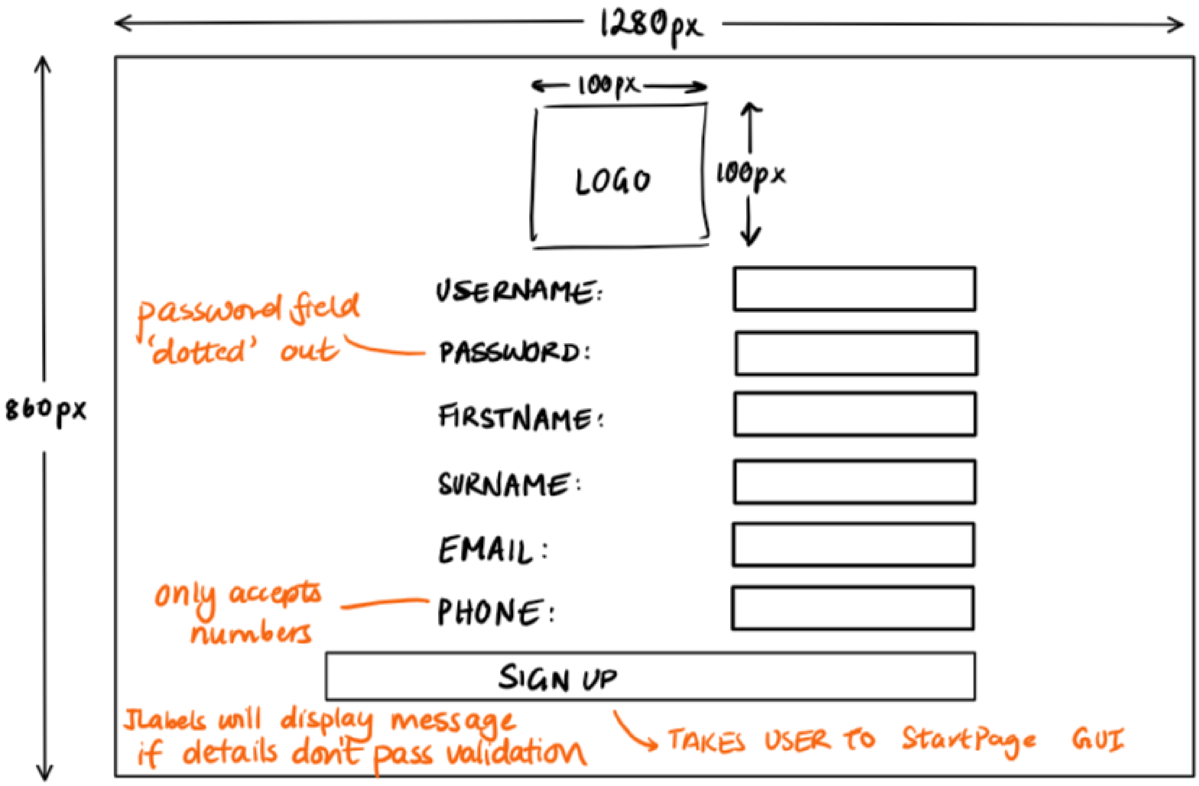
Main GUI



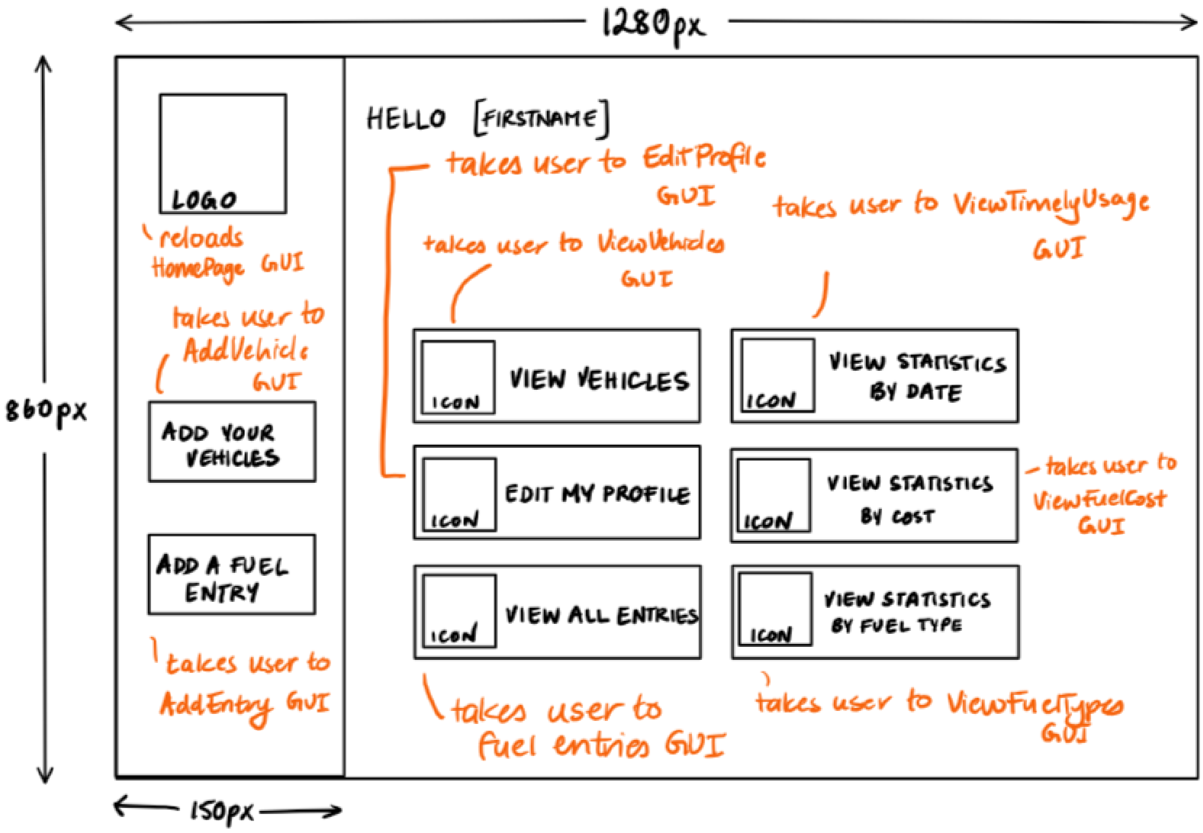
Login GUI


1280px
860px
100px
100px
LOGO
USERNAME:
PASSWORD:
password field 'dotted' out
JLabel will display "Invalid Username and Password" if incorrect login details
LOGIN
TAKES USER TO HomePage GUI

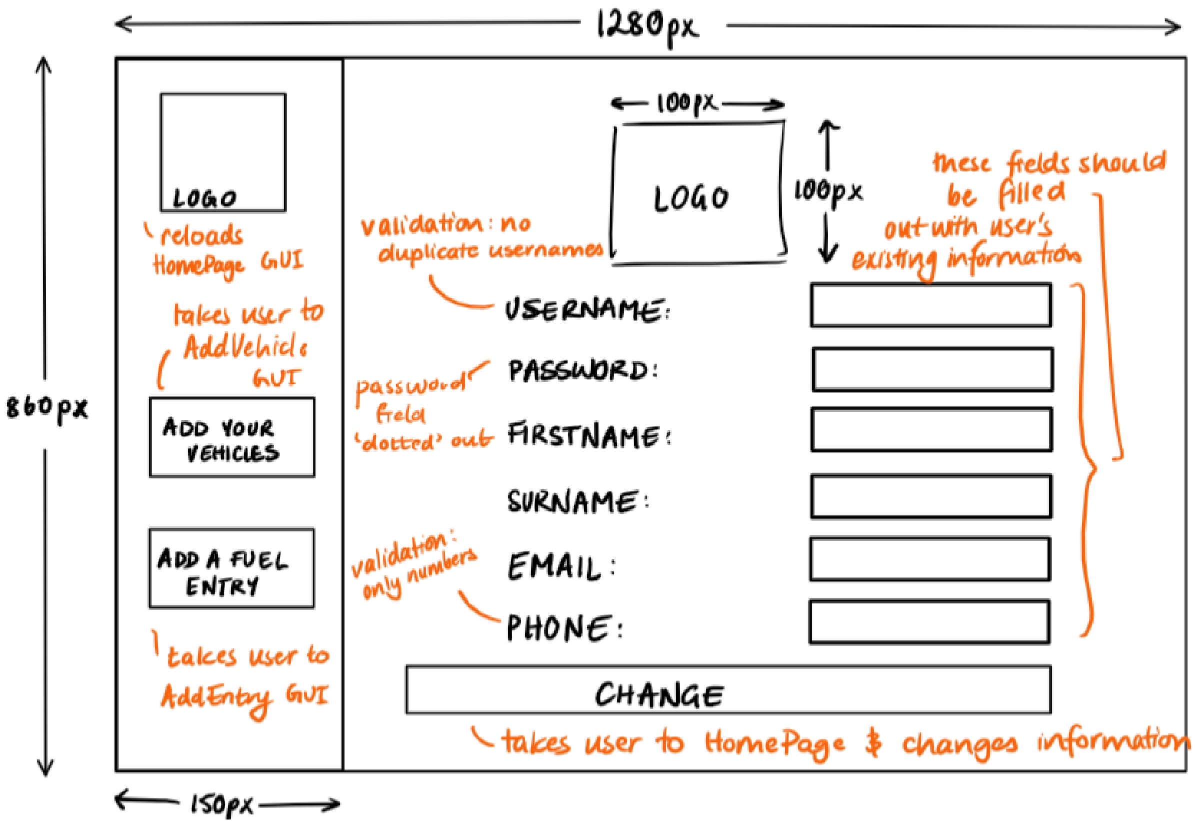
Signup GUI


1280px
860px
100px
100px
LOGO
USERNAME:
PASSWORD:
FIRSTNAME:
SURNAME:
EMAIL:
PHONE:
SIGN UP
password field 'dotted' out
only accepts numbers
JLabels will display message if details don't pass validation
TAKES USER TO StartPage GUI

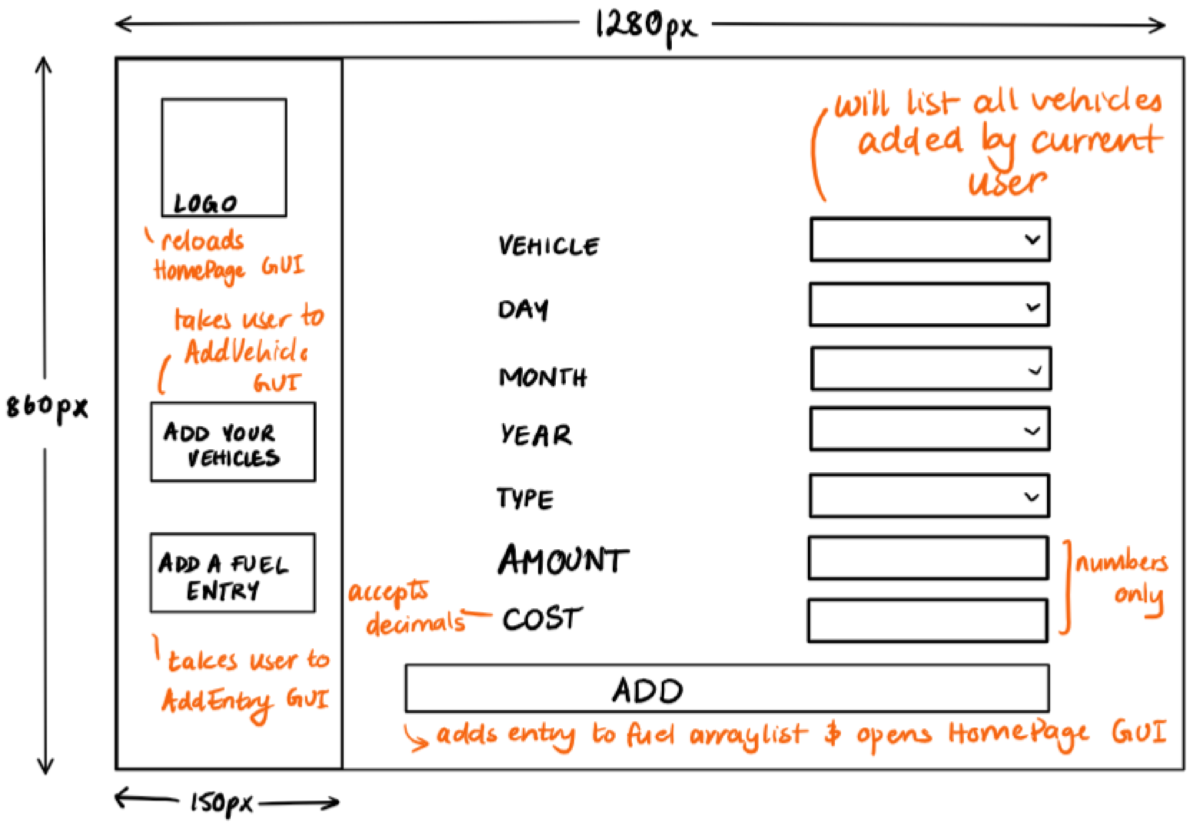
HomePage GUI


1280px
860px
150px
LOGO
reloads HomePage GUI
takes user to AddVehicle GUI
ADD YOUR VEHICLES
ADD A FUEL ENTRY
takes user to AddEntry GUI
HELLO [FIRSTNAME]
takes user to EditProfile GUI
takes user to ViewVehicles GUI
takes user to ViewTimelyUsage GUI
ICON
VIEW VEHICLES
ICON
VIEW STATISTICS BY DATE
ICON
EDIT MY PROFILE
ICON
VIEW STATISTICS BY COST
takes user to ViewFuelCost GUI
ICON
VIEW ALL ENTRIES
ICON
VIEW STATISTICS BY FUEL TYPE
takes user to fuel entries GUI
takes user to ViewFuelTypes GUI

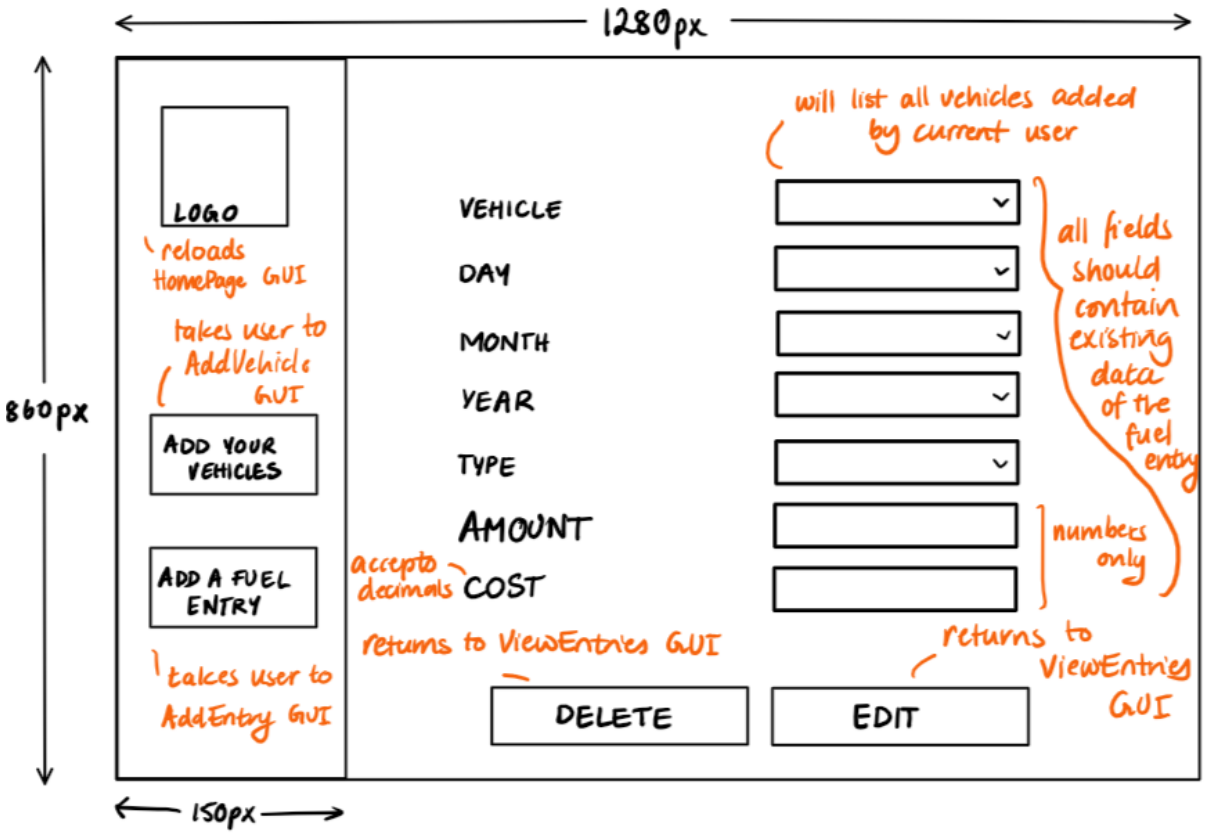
EditProfileGUI


1280px
860px
150px
LOGO
reloads HomePage GUI
takes user to AddVehicle GUI
ADD YOUR VEHICLES
ADD A FUEL ENTRY
takes user to AddEntry GUI
100px
100px
LOGO
validation: no duplicate usernames
these fields should be filled out with user's existing information
USERNAME:
password field 'dotted' out
PASSWORD:
FIRSTNAME:
SURNAME:
validation: only numbers
EMAIL:
PHONE:
CHANGE
takes user to HomePage & changes information

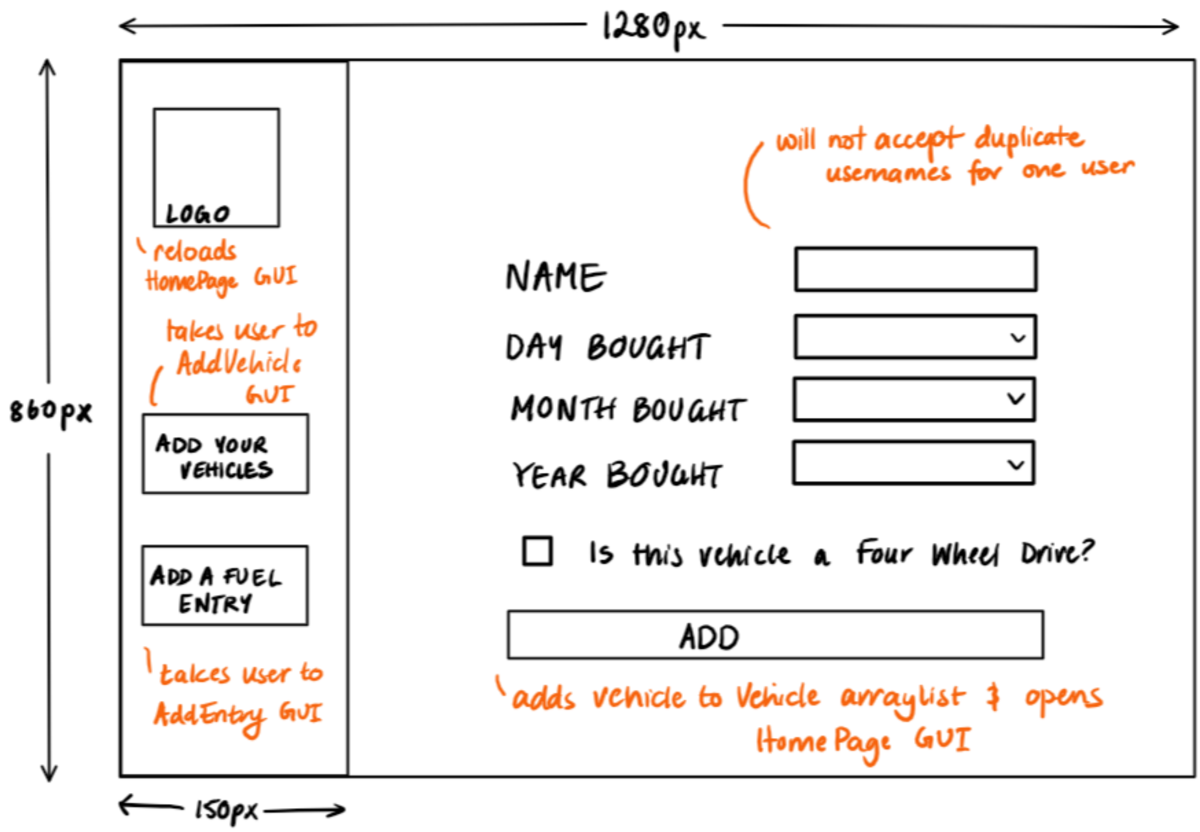
AddEntry GUI


1280px
860px
150px
LOGO
reloads HomePage GUI
takes user to AddVehicle GUI
ADD YOUR VEHICLES
ADD A FUEL ENTRY
takes user to AddEntry GUI
will list all vehicles added by current user
VEHICLE
DAY
MONTH
YEAR
TYPE
AMOUNT
accepts decimals
COST
numbers only
ADD
adds entry to fuel arraylist & opens HomePage GUI

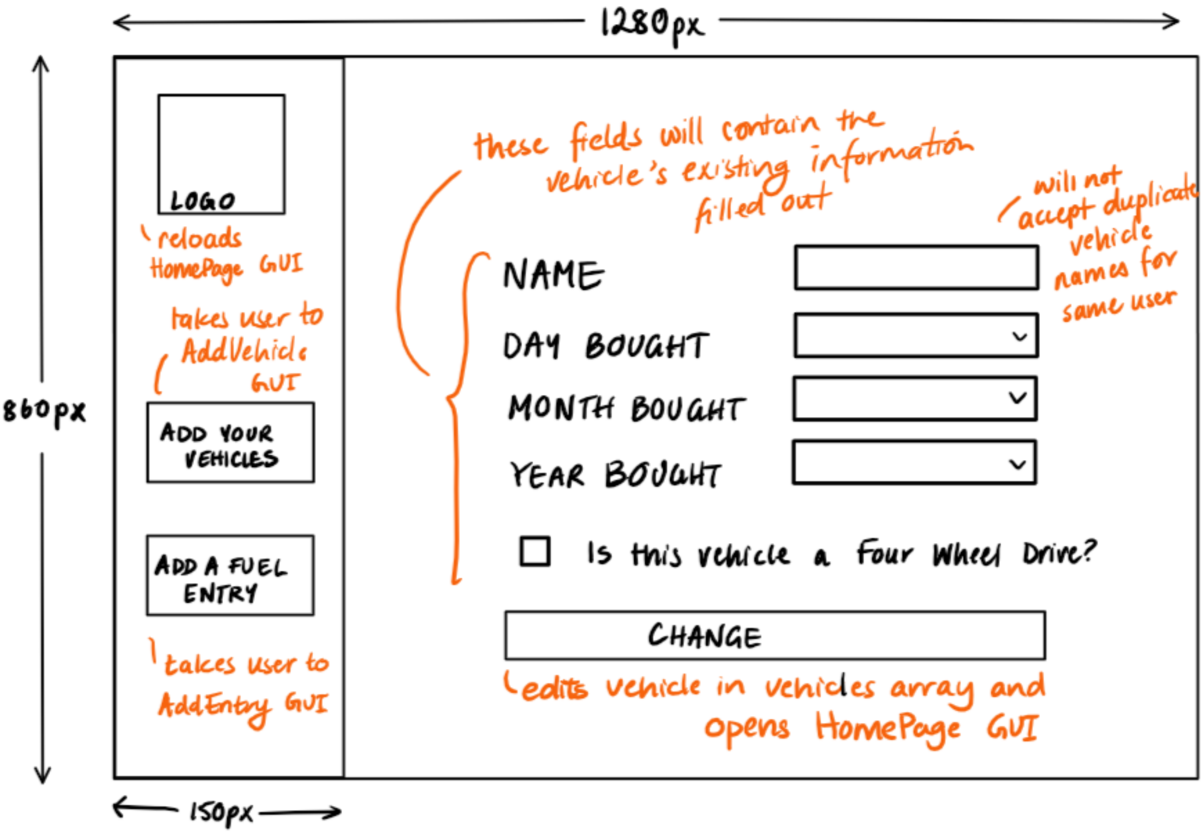
EditEntry GUI


1280px
860px
150px
LOGO
reloads HomePage GUI
takes user to AddVehicle GUI
ADD YOUR VEHICLES
ADD A FUEL ENTRY
takes user to AddEntry GUI
will list all vehicles added by current user
VEHICLE
DAY
MONTH
YEAR
TYPE
AMOUNT
COST
all fields should contain existing data of the fuel entry
numbers only
accepts decimals
returns to ViewEntries GUI
DELETE
EDIT
returns to ViewEntries GUI

AddVehicle GUI


1280px
860px
150px
LOGO
reloads HomePage GUI
takes user to AddVehicle GUI
ADD YOUR VEHICLES
ADD A FUEL ENTRY
takes user to AddEntry GUI
will not accept duplicate usernames for one user
NAME
DAY BOUGHT
MONTH BOUGHT
YEAR BOUGHT
□ Is this vehicle a Four Wheel Drive?
ADD
adds vehicle to Vehicle arraylist & opens Home Page GUI

EditVehicleGUI



reloads HomePage GUI

takes user to AddVehicle GUI

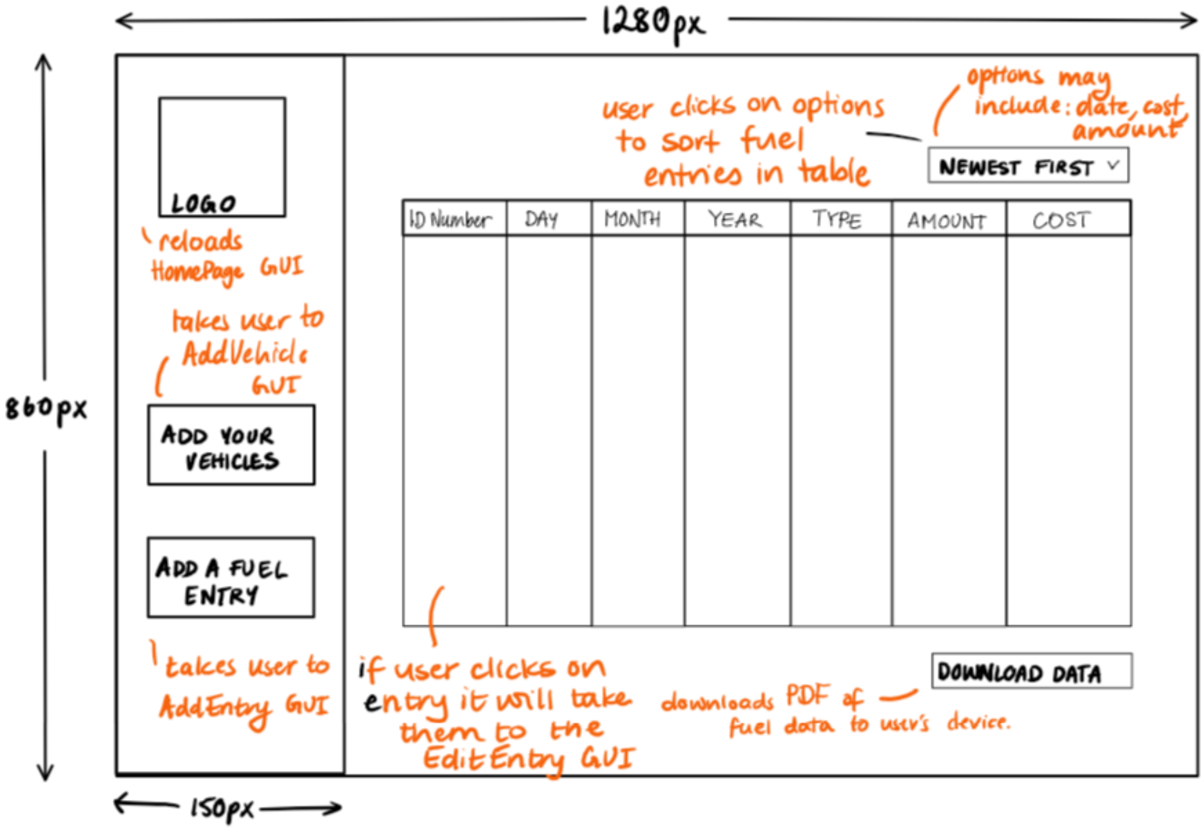
takes user to AddEntry GUI

these fields will contain the vehicle's existing information filled out

will not accept duplicate vehicle names for same user

edits vehicle in vehicles array and opens HomePage GUI

ViewEntries GUI



ID Number	DAY	MONTH	YEAR	TYPE	AMOUNT	COST

reloads HomePage GUI

takes user to AddVehicle GUI

takes user to AddEntry GUI

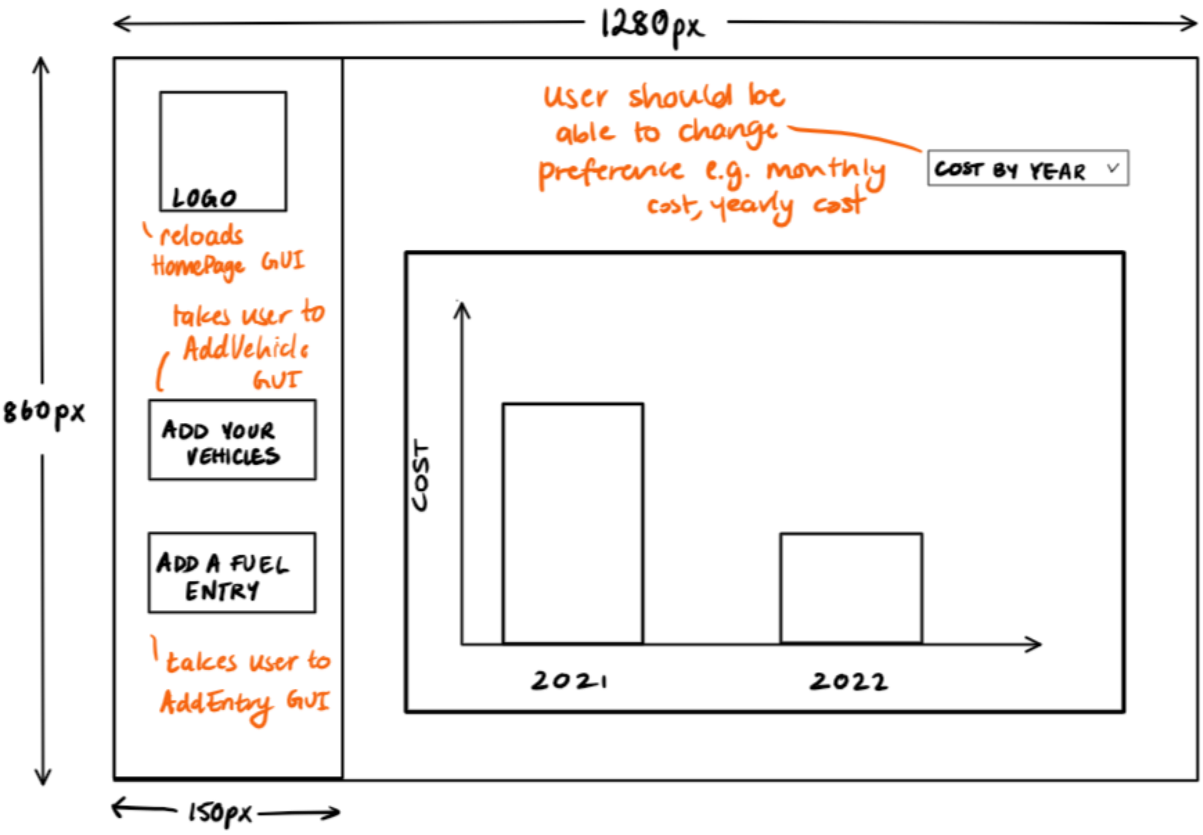
user clicks on options to sort fuel entries in table

options may include: date, cost, amount

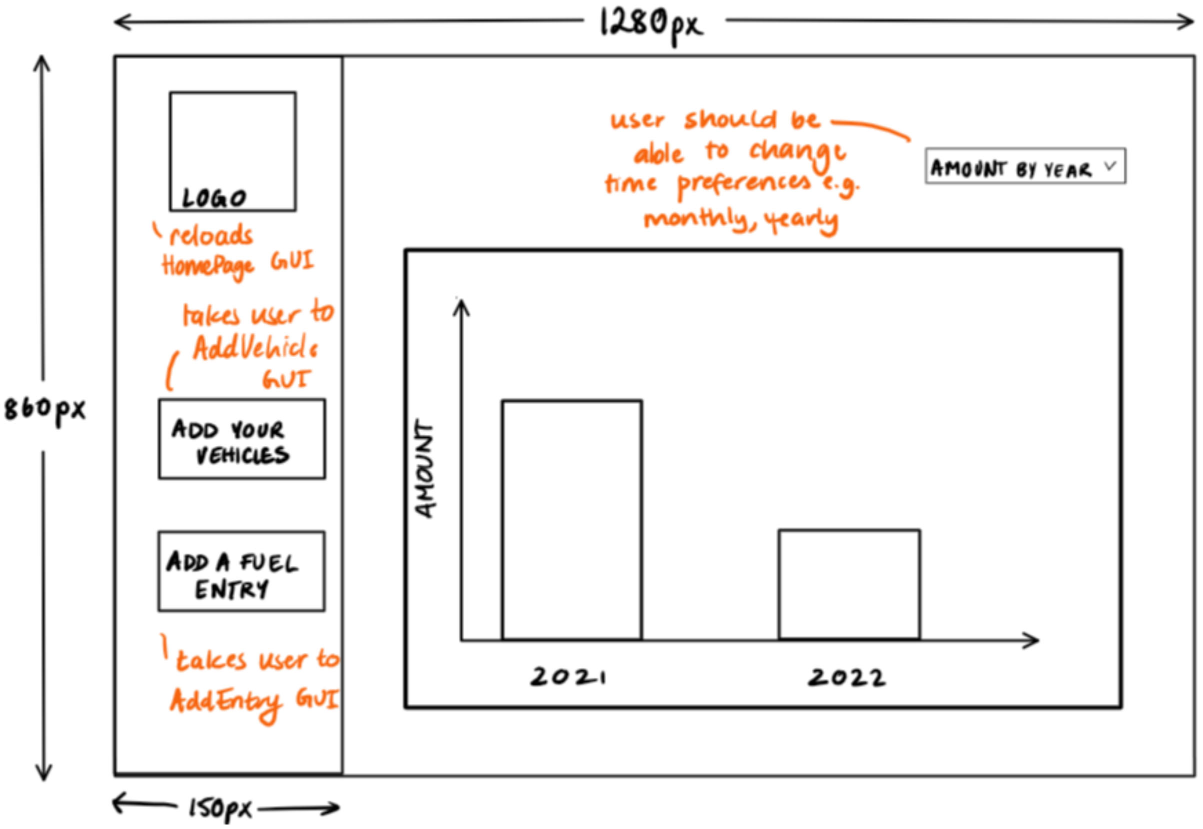
if user clicks on entry it will take them to the EditEntry GUI

downloads PDF of fuel data to user's device.

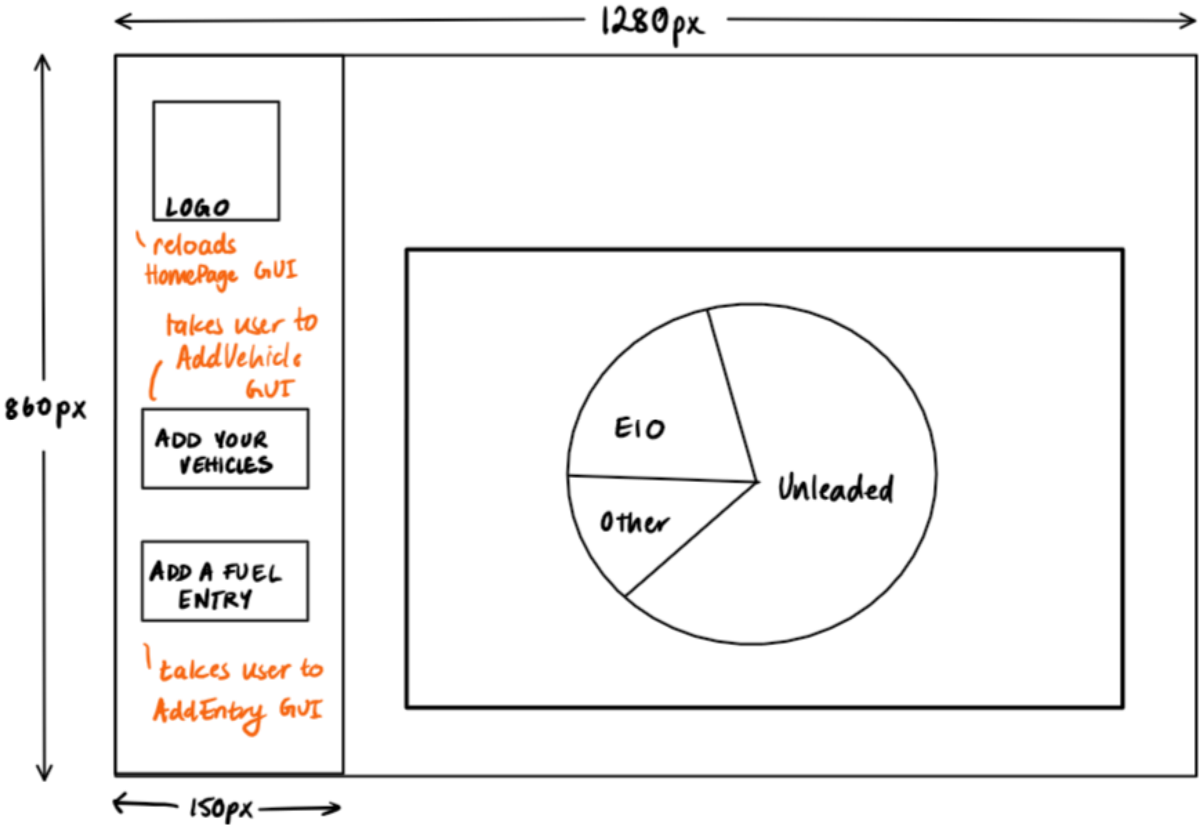
ViewFuelCost GUI


1280px
860px
150px
LOGO
reloads HomePage GUI
takes user to AddVehicle GUI
ADD YOUR VEHICLES
ADD A FUEL ENTRY
takes user to AddEntry GUI
user should be able to change preference e.g. monthly cost, yearly cost
COST BY YEAR
COST
2021
2022

ViewTimelyUsage GUI


1280px
860px
150px
LOGO
reloads HomePage GUI
takes user to AddVehicle GUI
ADD YOUR VEHICLES
ADD A FUEL ENTRY
takes user to AddEntry GUI
user should be able to change time preferences e.g. monthly, yearly
AMOUNT BY YEAR
AMOUNT
2021
2022

ViewFuelTypes GUI



ViewVehicles GUI

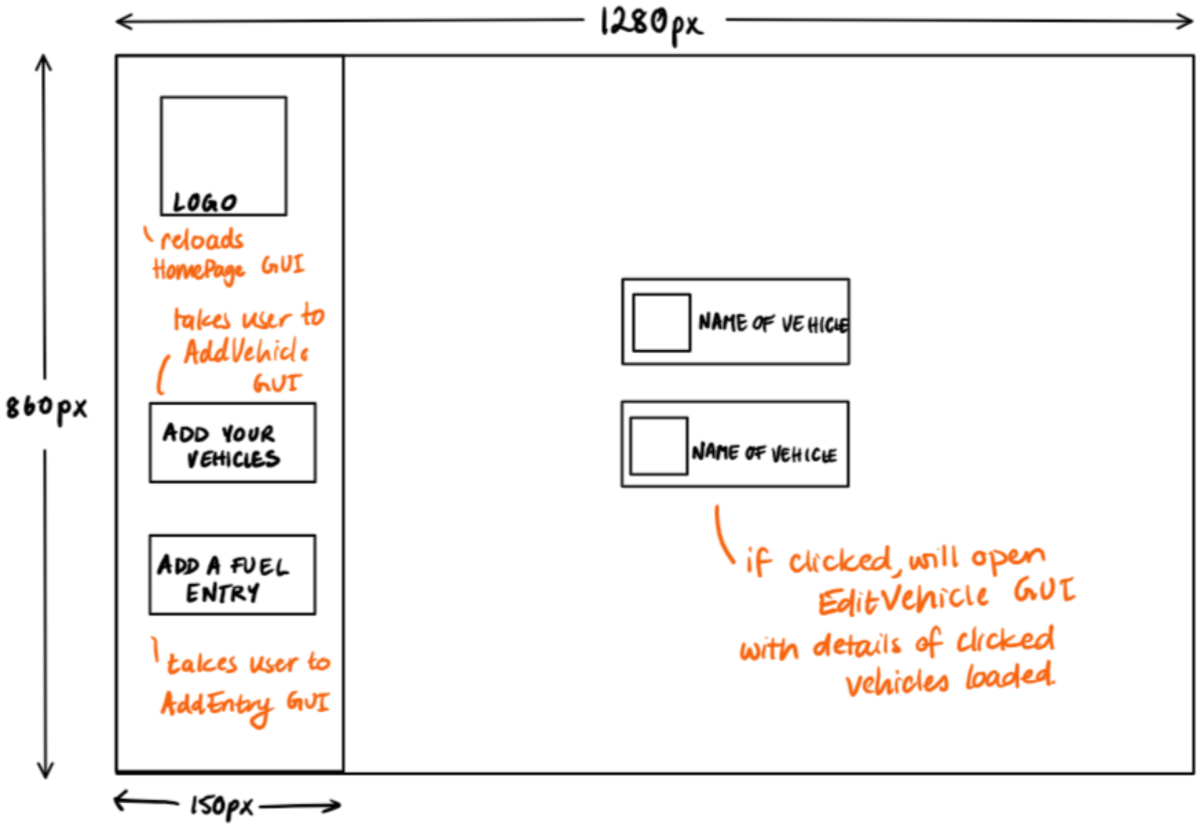

1280px
860px
150px
LOGO
reloads HomePage GUI
takes user to AddVehicle GUI
ADD YOUR VEHICLES
ADD A FUEL ENTRY
takes user to AddEntry GUI
NAME OF VEHICLE
NAME OF VEHICLE
if clicked, will open EditVehicle GUI with details of clicked vehicles loaded

Table of Classes

MVC (Model View Controller) package design will be used to organize the code.

Class Name (and Package)	Purpose
AtlantisFuel (Controller)	This class will contain methods and instantiated data structures used widely around the program such as the users array and serialization method. Its data will be accessible to every other class
Amounts (Model)	The Amounts class will contain the encapsulated variables in the amounts arraylist used in the ViewTimelyUsage GUI. It's data will be taken from the fuelConsumption arraylist when the GUI is opened.
CostsPerYear (Model)	This class will contain the encapsulated variables in the costsPerYear arraylist instantiated in the ViewFuelCost GUI.
FuelConsumption (Model)	The FuelConsumption class is a class containing the encapsulated variables in the fuelConsumption object arraylist.
Users (Model)	The Users class is a class containing the encapsulated variables in the users object arraylist storing user objects.
Vehicles (Model)	The Vehicles class is a class containing the encapsulated variables in the vehicles object arraylist storing vehicle objects.
AddEntry (View)	This class will contain the GUI where users add their fuel usage data. It should validate that this data is correct before adding it to the fuelConsumption arraylist.
AddVehicle (View)	This class will contain the GUI where users add their vehicle data. It should validate the data before adding it to the vehicles arraylist.
EditEntry (View)	This class will allow them to change any fuel usage info, again validating this before updating the fuelConsumption arraylist.
EditProfile (View)	This class will allow users to change their personal details. The class should validate the details before updating them in the users arraylist.
EditVehicle (View)	The EditVehicle class will allow users to edit information about their vehicles, validate the updated information and save it.
HomePage (View)	The HomePage will have access to most other pages in the program and should display "Welcome [first name]" at the top.

Login (View)	The Login should verify entered usernames and passwords and if correct, enter the HomePage.
Signup (View)	The Signup class will let users enter their information, ensure it is valid and avoids duplication of usernames, and lead users to the login page. If the information is not valid it will ask them to re-enter.
Main (View)	The Main GUI is the first GUI that opens when the program starts. It will allow the user to access the Login and Signup pages.
ViewEntries (View)	The ViewEntries class should display all the user's fuel usage data in a list. The user should be able to sort this data by data and cost. This class should also allow a download of the data.
ViewFuelCost (View)	The ViewFuelCost class should show the user's fuel data with relation to the cost per year through a bar graph.
ViewFuelTypes (View)	The ViewFuelTypes class will show the user's fuel data with relation to the types of fuel used. It should show this with a pie chart format.
ViewTimelyUsage (View)	The ViewTimelyUsage class should show the user's fuel data with relation to the amount per year through a bar graph.
ViewVehicles (View)	The ViewVehicles page should display all the logged in user's vehicles. If the user clicks on the vehicle names, it should take them to the EditVehicle GUI.

UML Diagrams[1]

Amounts
- year: String - amount: int
//Accessor and Mutator methods + getAmounts() :Arraylist + setAmounts(amounts: Arraylist) :void + getYear() :String + setYear(year: String) :void + getAmount() :int + setAmount(amount: int) :void

CostsPerYear
- year: String - cost: float
//Accessor and Mutator methods + getCostsPerYear() :Arraylist

+ setCostsPerYear(costsPerYear: Arraylist) :void + getYear() :String + setYear(year: String) :void + getCost() :float + setCost(cost: float) :void

FuelConsumption
- fuelID: long - username: String - fuelType: String - fuelAmount: int - cost: float - dateEnterred: Date - vehicleName: String
//Accessor and Mutator methods + getFuelConsumption() :Arraylist + setFuelConsumption(fuelConsumption: Arraylist) :void + getFuelID() :long + setFuelID(fuelID: long) :void + getFuelType() :String + setFuelType(fuelType: String) :void + getFuelAmount(): int + setFuelAmount(fuelAmount: int) :void + getUsername() :String + setUsername(username: String) :void + getCost() :float + setCost(cost: float) :void + getDateEnterred() :Date + setDateEnterred(dateEnterrred: Date) :void + getVehicleName() :String + setVehicleName(vehicleName: String) :void

Users
- username: String - password: String - firstname: String - surname: String - email: String - phone: int
//Accessor and Mutator methods + getUsers() :Arraylist + setUsers(users: Arraylist) :void + getUsername() :String + setUsername(username: String) :void + getPassword() :String

+ setPassword(password: String) :void
+ getFirstname() :String
+ setFirstname(firstname: String) :void
+ getSurname() :String
+ setSurname(surname: String) :void
+ getEmail() :String
+ setEmail(email: String) :void
+ getPhone() :int
+ setPhone(phone: int) :void

Vehicles
- username: String - nameOfVehicle: String - dayBought: int - monthBought: int - yearBought: int - fourWheelDrive: boolean
//Accessor and Mutator methods + getVehicles() :Arraylist + setVehicles(vehicles: Arraylist) :void + getUsername() :String + setUsername(username: String) :void + getNameOfVehicle() :String + setNameOfVehicle(nameOfVehicle: String) :void + getDayBought() :int + setDayBought(dayBought: int) :void + getMonthBought() :int + setMonthBought(monthBought: int) :void + getYearBought() :int + setYearBought(yearBought: int) :void + isFourWheelDrive() :Boolean + setFourWheelDrive(fourWheelDrive: boolean) :void

AddEntry
- GUIcomponents + fuelAmount: int + cost: float + fuelID: long = 0
- GUImethods() + addEntryButton(): void

AddVehicle
- GUIcomponents
- GUImethods() + addVehicleButton(): void

EditEntry
- GUIcomponents + fuelAmount: int + cost: float + monthEnterred: int + month: String
- GUImethods() + editEntryButton() :void

EditProfile
- GUIcomponents + previousName: String = Login.username + username: String
- GUImethods() + editProfileButton() :void

EditVehicle
- GUIcomponents + dateMonth: String + index: int = 0
- GUImethods() + changeButton() :void

HomePage
- GUIcomponents + username: String = Login.username
- GUImethods()

Login
- GUIcomponents + username: String
- GUImethods() + loginButton() :void

ViewEntries
- GUIcomponents + fuelID: long
- GUImethods() + downloadPDF() :void

ViewFuelCost

- GUIcomponents + found: boolean = false
- GUImethods() + findCostsPerYear() :void

ViewTimelyUsage
- GUIcomponents + found: Boolean = false
- GUImethods() + findYears() :void

ViewVehicles
- GUIcomponents
- GUImethods() + ViewVehicles(username: String) :void

AtlantisFuel
- GUIcomponents
- GUImethods() + addUser(username: String, password: String, firstname: String, surname: String, email: String, phone: int) :boolean + editUser(username: String, password: String, firstname: String, surname: String, email: String, phone: int, currentUser: int) :boolean + findUser(username: String): int + addFuelEntry(fuelID: long, username: String, fuelType: String, fuelAmount: int, cost: float, dateEnterred: Date) :boolean + editEntry(fuelID: long, username: String, fuelType: String, fuelAmount: int, cost: float, dateEnterred: Date) :boolean + addVehicle(username: String, nameOfVehicle: String, dayBought: int, monthBought: int, yearBought: int, fourWheelDrive: boolean) :boolean + editVehicle(username: String, nameOfVehicle: String, dayBought: int, monthBought: int, yearBought: int, fourWheelDrive: boolean) :boolean + displayDays() :boolean + displayMonths() :boolean + displayYear() :boolean + displayFuelTypes() :boolean

+ findVehicle(nameOfVehicle: String, username: String) :int
+ addRowToJTable() :void
+ chartThisMonth () :void

[1] The UMLs do not include iteration variables i.e., those used in loops within the program.

Table of Methods

This list does not include accessor/mutator methods.

Method Name	Purpose
addUser()	Adds parameters of a new user object as a user object to the users arraylist.
editUser()	Will update user objects in the users arraylist with the new details sent through the parameter list.
findUser()	Will find the index of a user in the users arraylist using their username.
addFuelEntry()	Will add FuelConsumption objects, sent as parameters, to the fuelConsumption arraylist.
editEntry()	Will edit and update FuelConsumption objects in the fuelConsumption arraylist.
addVehicle()	Will add Vehicle objects, sent through parameters, to the vehicles arraylist.
editVehicle()	Will edit and update Vehicle objects in the vehicles arraylist.
displayDays()	Will loop to display days of the month in a jComboBox (drop down list)
displayMonths()	Will loop to display months in a jComboBox (drop down list) in another GUI.
displayYear()	Will display the required years in a jComboBox (drop down list) in another GUI.
displayFuelTypes()	Will loop to display fuel types accepted by the program to drop down list (jComboBox)
findVehicle()	Will find vehicles using their names and usernames and return their index in the arraylist.
addRowToJTable()	Will display data from the fuelConsumption arraylist in a table (jTable) the ViewEntries GUI.
chartThisMonth()	Will convert data from the fuelConsumption array into a pie chart.
addEntryButton()	Will r validate the information before calling on the addEntry() method.
addVehicleButton()	Will validate user’s vehicle information before calling the addVehicle() method and returning to HomePage GUI.
editEntryButton()	Will validate information the user has changed about previous fuel entries and pass it to the editEntry() method.

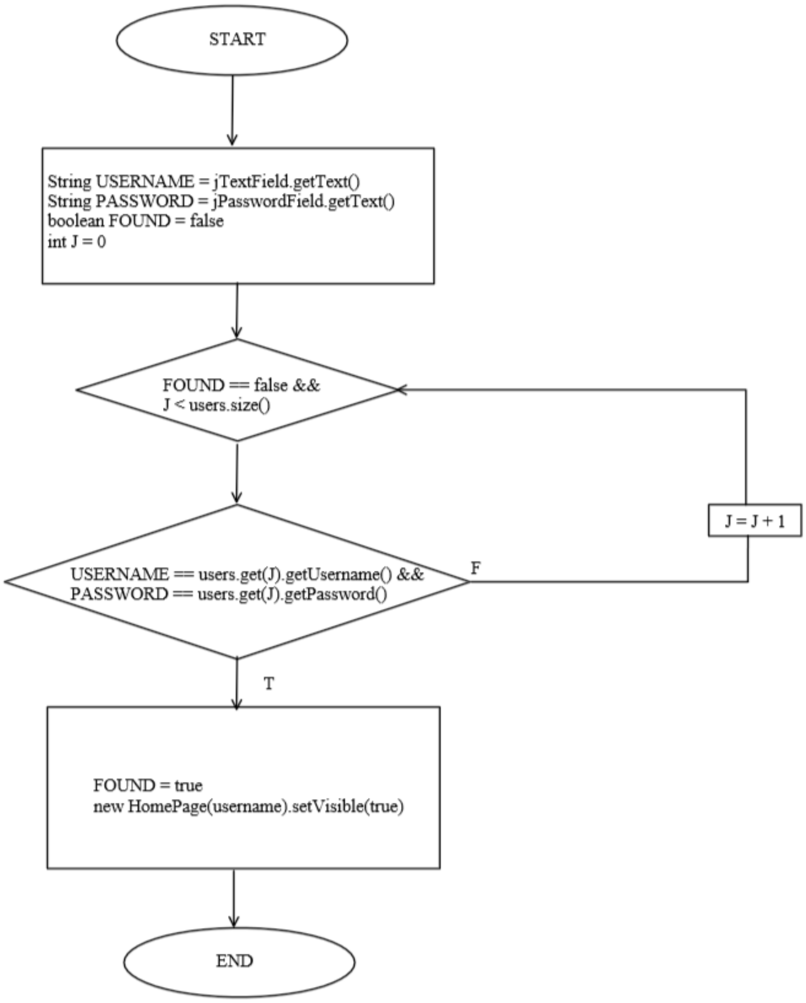
editProfileButton()	Will validate the user’s updated personal information and send it to the editUser() method.
changeButton()	Will validate updated information about the users’ vehicles and send it to the editVehicle() method.
loginButton()	Will verify usernames and passwords either open the HomePage GUI or ask the user to re-enter details.
signupButton()	This method will validate users’ details, send the details to the addUser() method and re-direct the user to the Main GUI.
downloadPDF()	This method will download the user’s fuel usage data to their downloads as a PDF.
findCostsPerYear()	Will create the data set for a bar chart through the arraylist.
findYears()	Will create the data set for a bar chart through the arraylist.

Justification of Data Structures

Name	Structure	Data involved	Justification
users	Arraylist	User objects	• Can store objects, in this case, user specific date and amount data, allowing easy accessibility • Unlike some other structures, they are extendable, meaning storage is flexible and saved. • Arraylists have their own methods e.g. removeAll(); • Allow full functionality: insertion, traversal, deletion, retrieval over other structures
vehicles	Arraylist	Vehicle objects	
fuelConsumption	Arraylist	Fuel objects	
costsPerYear	Arraylist	Money spent on fuel yearly for the current user only – taken from the fuelConsumption arraylist.	
amounts	Arraylist	Amount of fuel used yearly for the current user only – taken from the fuelConsumption arraylist.	

Pseudocode/Flowchart Algorithms

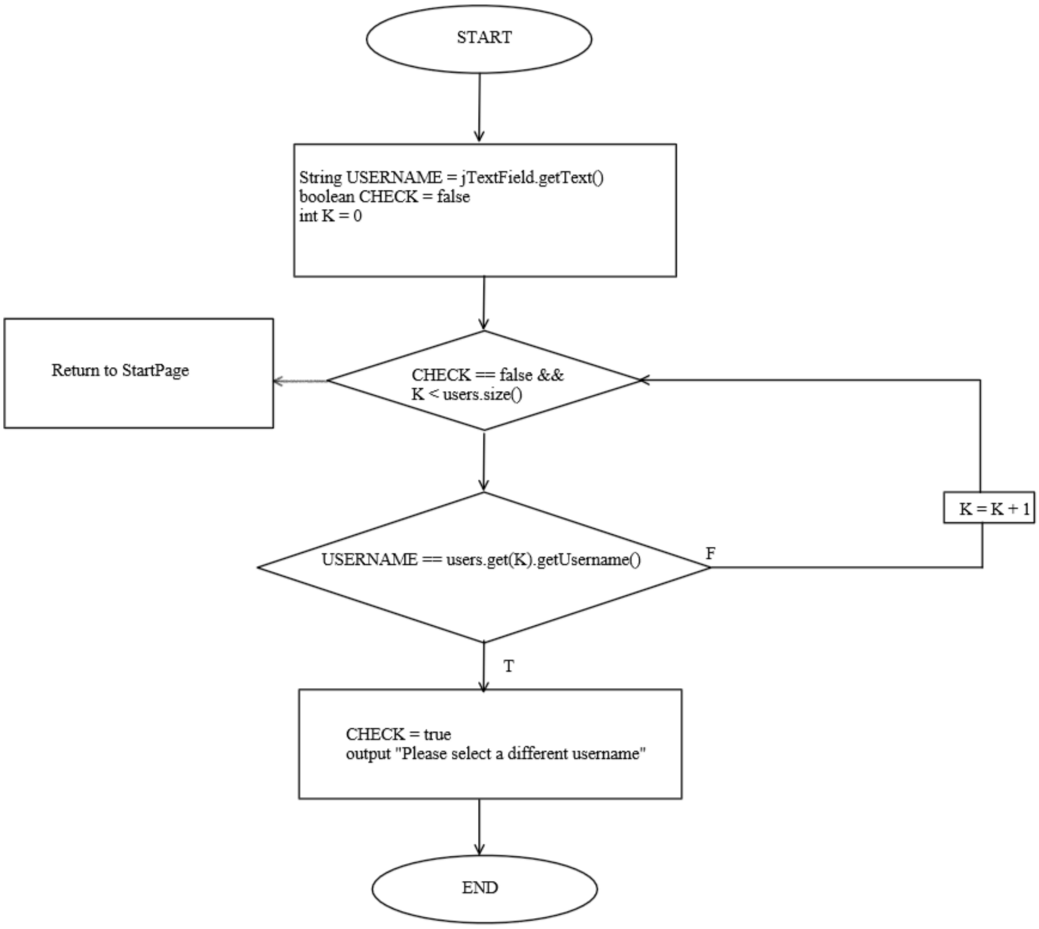
Login Username Validation


START
String USERNAME = jTextField.getText()
String PASSWORD = jPasswordField.getText()
boolean FOUND = false
int J = 0
FOUND == false &&
J < users.size()
USERNAME == users.get(J).getUsername() &&
PASSWORD == users.get(J).getPassword()
F
J = J + 1
T
FOUND = true
new HomePage(username).setVisible(true)
END

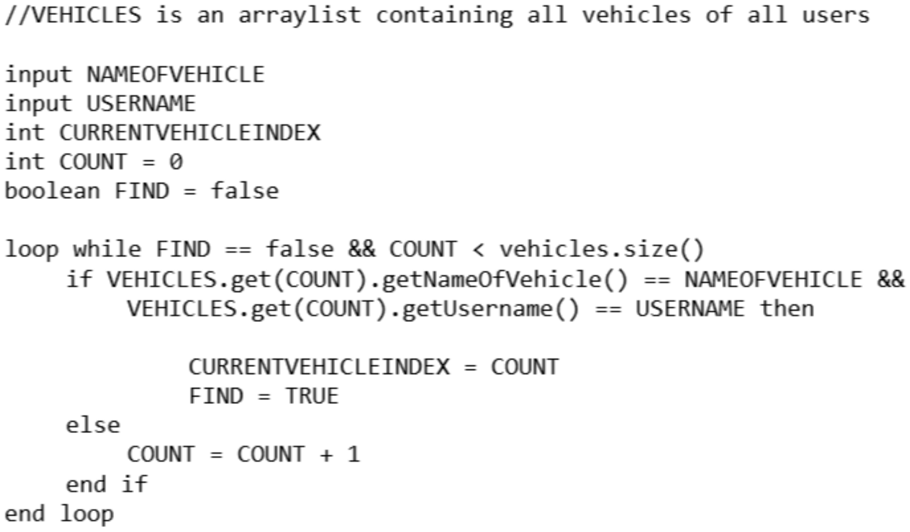
addRowToJTable() method: collects the user's fuel data to collate into a table

```
//fuelConsumption is an array containing fuel entries of all users
//model is an object of type DefaultTableModel that stores cell values
//USERNAME is the username of the current user logged in

loop I from 0 to fuelConsumption.size()
     if fuelConsumption.get(I).getUsername() == USERNAME then
        Object rowData = new Object[7]
        rowData[0] = String.valueOf(fuelConsumption.get(I).getFuelID)
        rowData[1] = String.valueOf(fuelConsumption.get(I).getDateEnterred().getDay())
        rowData[2] = String.valueOf(fuelConsumption.get(I).getDateEnterred().getMonth())
        rowData[3] = String.valueOf(fuelConsumption.get(I).getDateEnterred().getYear())
        rowData[4] = String.valueOf(fuelConsumption.get(I).getFuelType())
        rowData[5] = String.valueOf(fuelConsumption.get(I).getFuelAmount())
        rowData[6] = String.valueOf(fuelConsumption.get(I).get Cost())
        model.addRow(rowData)
    end if
end loop
```

Signup Username Verification

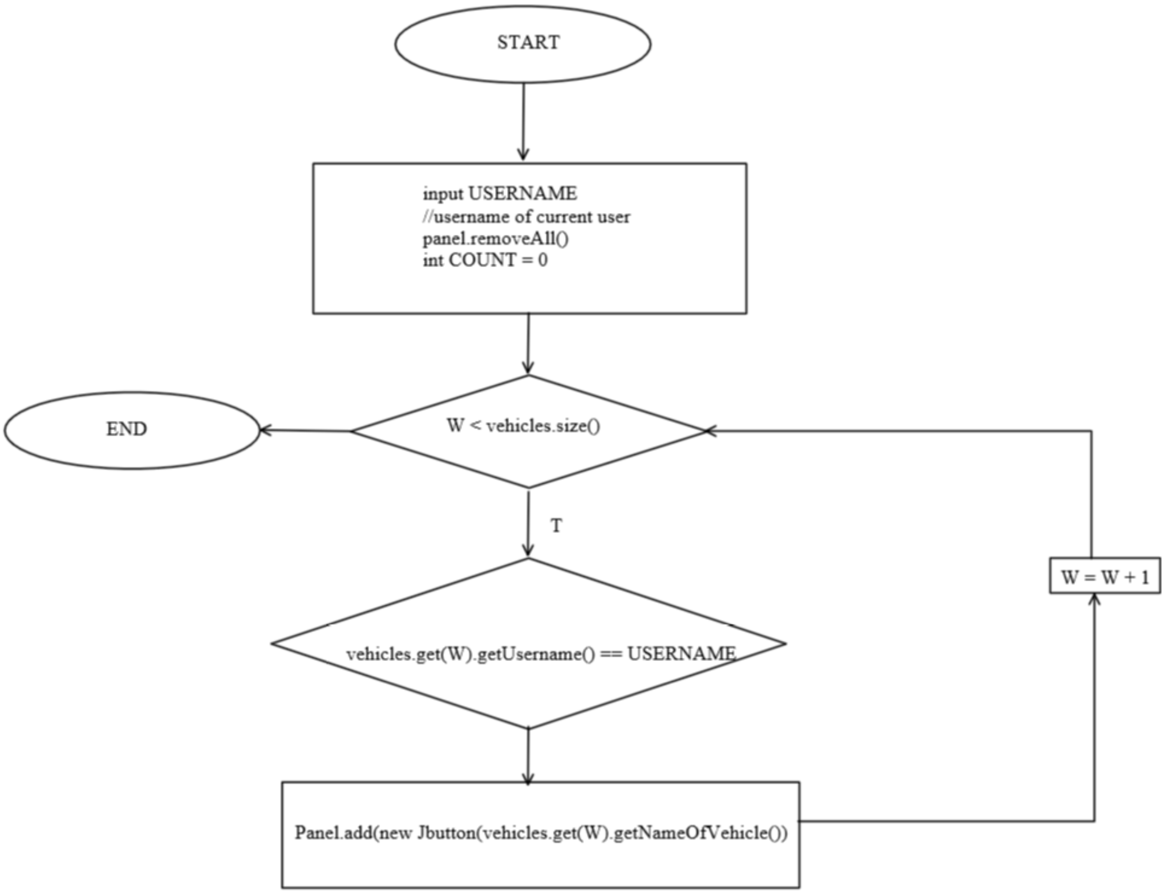

START
String USERNAME = jTextField.getText()
boolean CHECK = false
int K = 0
Return to StartPage
CHECK == false &&
K < users.size()
K = K + 1
USERNAME == users.get(K).getUsername()
F
T
CHECK = true
output "Please select a different username"
END

findVehicle() method: finds the index of the user's vehicle in the vehicle arraylist

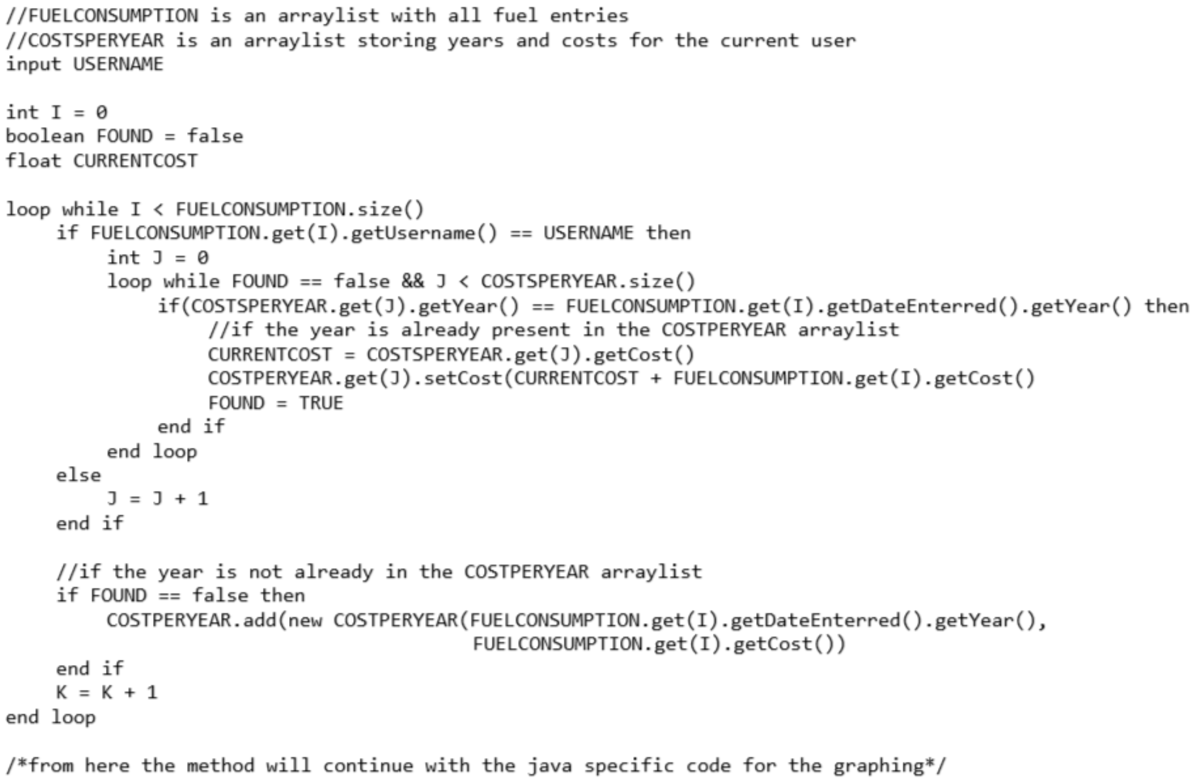


```
//VEHICLES is an arraylist containing all vehicles of all users

input NAMEOFVEHICLE
input USERNAME
int CURRENTVEHICLEINDEX
int COUNT = 0
boolean FIND = false

loop while FIND == false && COUNT < vehicles.size()
    if VEHICLES.get(COUNT).getNameOfVehicle() == NAMEOFVEHICLE &&
        VEHICLES.get(COUNT).getUsername() == USERNAME then

            CURRENTVEHICLEINDEX = COUNT
            FIND = TRUE
    else
        COUNT = COUNT + 1
    end if
end loop
```

Method to Display User’s Vehicles


START
input USERNAME
//username of current user
panel.removeAll()
int COUNT = 0
END
W < vehicles.size()
T
W = W + 1
vehicles.get(W).getUsername() == USERNAME
Panel.add(new Jbutton(vehicles.get(W).getNameOfVehicle())

findCostsPerYear() method: transfer’s user’s fuel data into another arraylist to prepare for graphing



```
//FUELCONSUMPTION is an arraylist with all fuel entries
//COSTSPERYEAR is an arraylist storing years and costs for the current user
input USERNAME

int I = 0
boolean FOUND = false
float CURRENTCOST

loop while I < FUELCONSUMPTION.size()
    if FUELCONSUMPTION.get(I).getUsername() == USERNAME then
        int J = 0
        loop while FOUND == false && J < COSTSPERYEAR.size()
            if(COSTSPERYEAR.get(J).getYear() == FUELCONSUMPTION.get(I).getDateEnterred().getYear() then
                //if the year is already present in the COSTPERYEAR arraylist
                CURRENTCOST = COSTSPERYEAR.get(J).getCost()
                COSTPERYEAR.get(J).setCost(CURRENTCOST + FUELCONSUMPTION.get(I).getCost()
                FOUND = TRUE
            end if
        end loop
    else
        J = J + 1
    end if

    //if the year is not already in the COSTPERYEAR arraylist
    if FOUND == false then
        COSTPERYEAR.add(new COSTPERYEAR(FUELCONSUMPTION.get(I).getDateEnterred().getYear(),
                                     FUELCONSUMPTION.get(I).getCost())
    end if
    K = K + 1
end loop

/*from here the method will continue with the java specific code for the graphing*/
```

Test Plan

Test/Criteria No.	Success Criteria	Expected Outcome
1	The user is able to signup and login to the application	• Users can sign up on signup page • Signup page ensures usernames are not duplicated. • Signup page ensures all other user information is valid e.g., phone number is numbers only. • Login page ensures username and password are entered correctly.
2	The user is able to edit personal information	• The EditProfile page allows users to edit personal information. • This class should still ensure no duplication of usernames.
3	The application enables users to enter and edit vehicle related data	• The AddVehicle page and EditVehicle page allow users to add and edit, respectively, their vehicles. • The classes should add to and update the vehicles arraylist.
4	The application enables users to enter and edit fuel related data	• The AddEntry and EditEntry pages allow users to add and edit fuel data respectively. • Both pages ensure amount, cost and date information are entered/formatted correctly e.g. amount is only numbers.
5	Users are able to view the data entries about fuel and sort the entries categorically	• The ViewEntries page displays a list of the user's entries. • The user can sort these by cost and date.
6	Users are able to download a PDF of the fuel data	• A button on the ViewEntries page automatically exports a PDF of fuel data to the user's downloads.
7	Users can view graphs of fuel data that is shown in relation to cost of fuel, timely usage of fuel and the type of fuel consumed.	• Users can view fuel data summarized as graphs based on different categories: ○ Cost ○ Type of Fuel ○ Date