Anthony Pompili
CSC 415
Dr. Pulimood
Assignment 5 - Open Source Software: Analysis and Design
December 1, 2017
Link: https://github.com/pomps8/CSC415_AnthonyPompili_ECar
Github Repository: pomps8/CSC415_AnthonyPompili_ECar
Project: E-Car

Use Case Descriptions

<table>
<tr><td colspan="2">Use case: Check Singular Car in Carlist</td></tr>
<tr><td>Iteration: 1, last modification: December 1, 2017

Primary Actor: E-car App user

Goal in context: Browse the catalog of cars and view specific car

Preconditions: Application must be downloaded on iOS Device, and the user must have an account created

Trigger: User clicks on the “View car info” button in the main menu

Scenario:
User goes into the E-Car app
User selects the “View Car info” button
User selects the brand of car
User selects the model of car
User selects the year of the car
User selects the transmission of the car
User Selects the cylinders of the car
Car data is outputted to the users iOS Device if car exists, else a UIAlert pops up saying the car doesn’t exist</td><td>Exceptions:
Data base is not up to date and does not contain a specific car
Car data was pulled incorrectly, and data is not correct
Data stored wrong and cannot pull correct car
Priority: High priority, to be implemented after foundations to the application are complete

When available: second increment

Frequency of use: Frequent

Chanel to actor: Through E-Car Application

Secondary Actors: Database to pull information in list

Channels to secondary actors:
SQLite: Database access
Open Issues:
If querying a car that exists, app says car doesn’t exist, must hit button again to see result</td></tr>
</table>

Use case: Add Data (Gas consumption and Mileage)	
Iteration: 1, last modification: November 9, 2017 **Primary Actor**: E-car App user **Goal in context**: Add gas consumption and mileage to users data **Preconditions**: Application must be downloaded on iOS Device, the user must have an account created and a personal car set. **Trigger**: User clicks on the “View Personal Car” button in the main menu **Scenario**: 1. User logs into the E-Car app 2. User selects the “View Personal Car” button 3. User selects the “Add gas” button at the bottom 4. User inputs valid info for gas consumed and current mileage 5. User info is updated (Footprint, MPG, mileage, etc.)	**Exceptions**: 1. User does not have personal car set 2. Negative / zero gas is inputted into the system 3. Mileage set is less than last inputted data set **Priority**: High priority, to be implemented after foundations to the application are complete **When available**: third increment **Frequency of use**: Highly Frequent **Chanel to actor**: Through E-Car Application **Secondary Actors**: None **Channels to secondary actors**: 1. None **Open Issues**: 1. How to make sure user sets up personal car first? 2. How to calculate mpg, footprint? 3. How to set up list of inputs? 4. How to edit / delete from this list?

Use case: Change Personal Car

Iteration: 1, last modification: December 1, 2017

Primary Actor: E-car App user

Goal in context: Browse the catalog of cars and find new car to set as personal car

Preconditions: Application must be downloaded on iOS Device, and the user must have an account created

Trigger: User clicks on the “Change Personal Car” button in the main menu

Scenario:

1. User goes into the E-Car app
2. User selects the “View Car info” button
3. User selects the brand of car
4. User selects the model of car
5. User selects the year of the car
6. User selects the transmission of the car
7. User Selects the cylinders of the car
8. Car data is outputted to the users iOS Device if car exists, else a UIAlert pops up saying the car doesn’t exist
9. User has option to change personal car

Exceptions:

1. Data base is not up to date and does not contain a specific car
2. Car data was pulled incorrectly, and data is not correct
3. Data stored wrong and cannot pull correct car

Priority: Medium priority, to be implemented after basics to the application are completed

When available: second increment

Frequency of use: Semi-Frequent

Chanel to actor: Through E-Car Application

Secondary Actors: None

Channels to secondary actors:

1. SQLite: Database access

Open Issues:

1. If querying a car that exists, app says car doesn’t exist, must hit button again to see result

Use case: Set Personal Car

Iteration: 1, last modification: November 9, 2017

Primary Actor: E-car App user

Goal in context: Browse the catalog of cars and select personal car for app usage

Preconditions: Application must be downloaded on iOS Device, and the user must have an account created

Trigger: User clicks on the “View Car Info” button in the main menu

Scenario:

1. User logs into the E-Car app
2. User selects the “View Car Info” button
3. User is presented a pop-up saying “You do not have a current personal car set up, would you like to make one?”
4. User selects “Yes” to continue
5. User browse list just like searching for a car in the car list, and finds their car
6. Data about car is inputted once car is found (Mileage)

Exceptions:

1. Data base is not up to date and does not contain the specific car
2. Use selects “No” and goes back to main menu
3. Bad info about mileage is inputted

Priority: High priority, to be implemented after foundations to the application are complete

When available: second increment

Frequency of use: Highly Frequent

Chanel to actor: Through E-Car Application

Secondary Actors: None

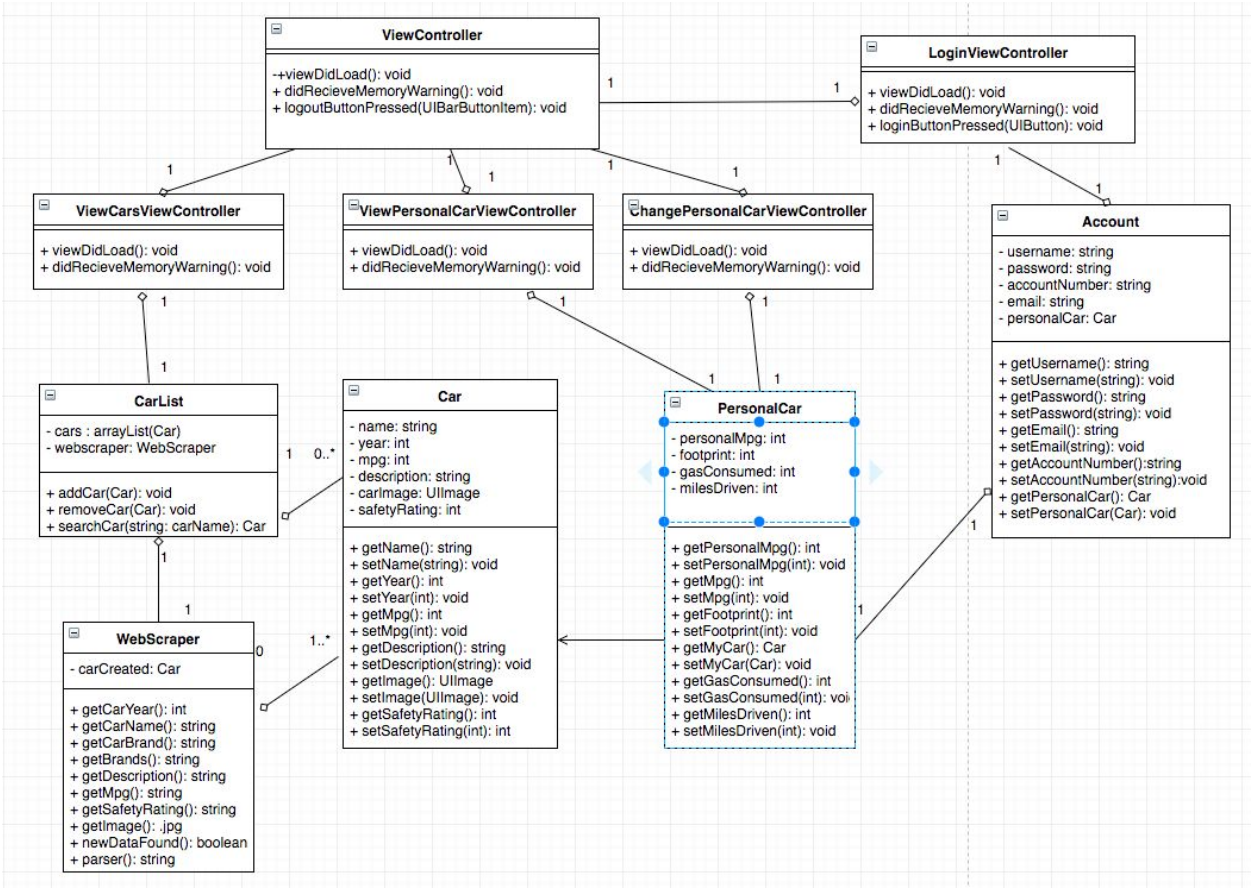
Channels to secondary actors:

1. None

Open Issues:

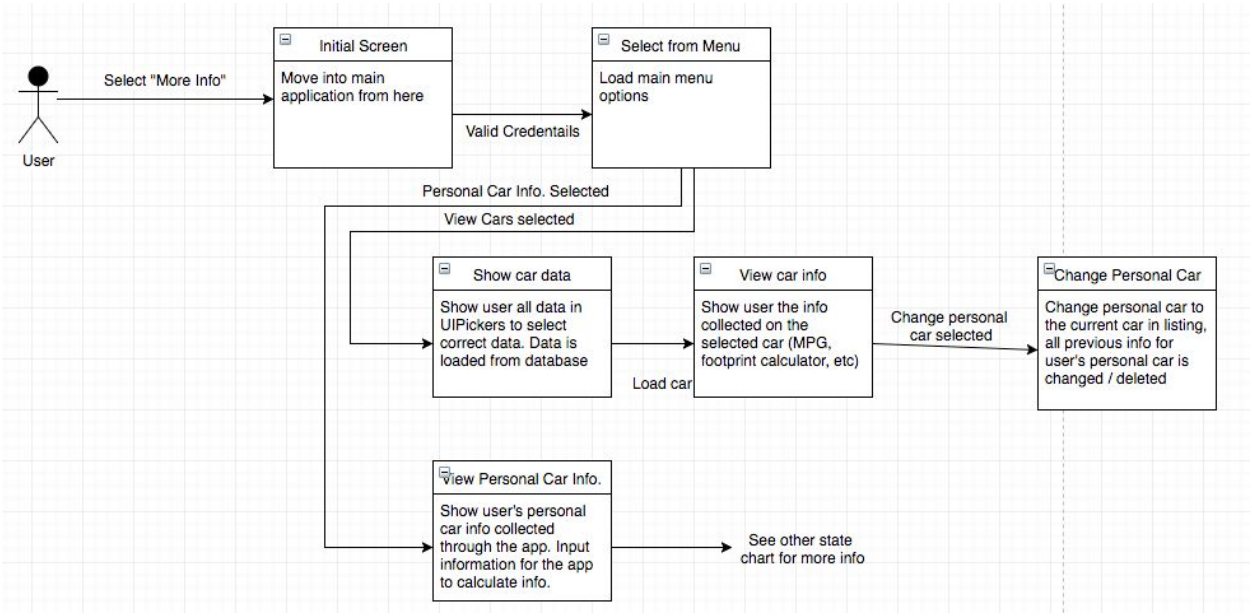
1. Create way for user to input data about car to estimate footprint size thus far prior to app use? (Est. mileage per week + average MPG = est. gas usage)

Detailed Design class diagram

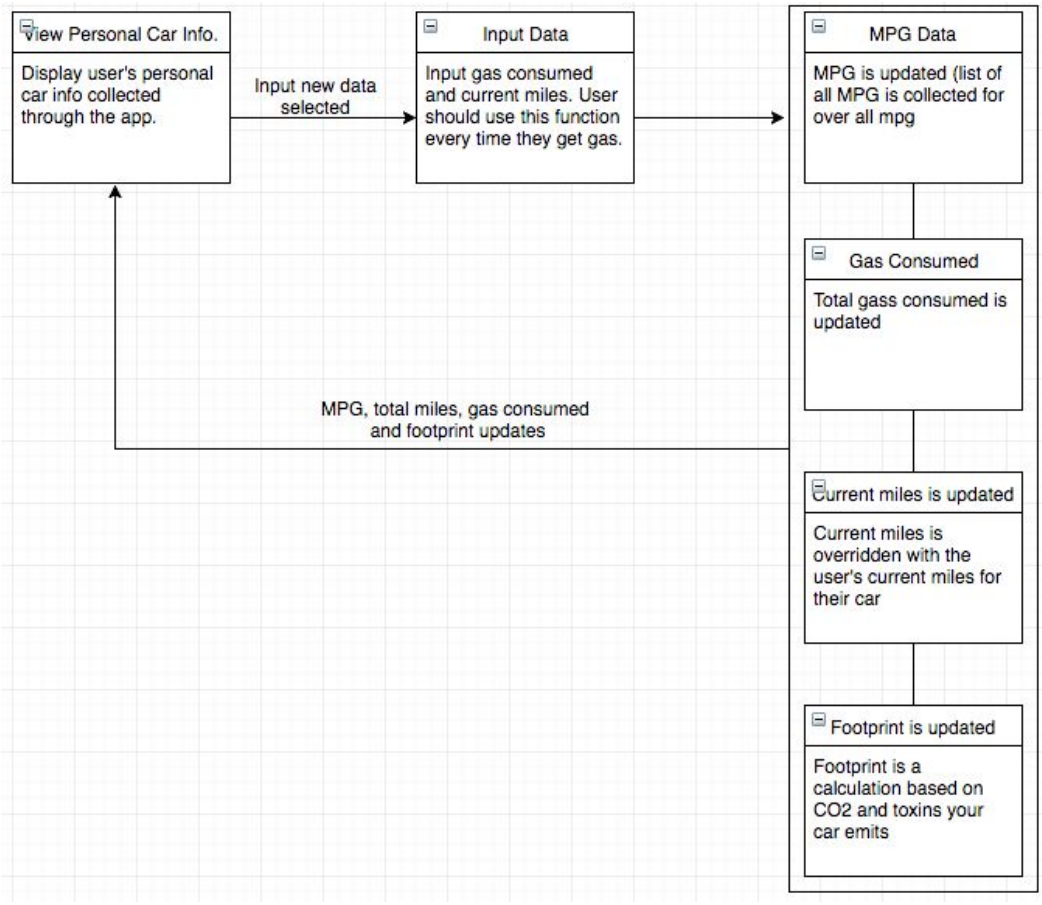


Statecharts

Overall System

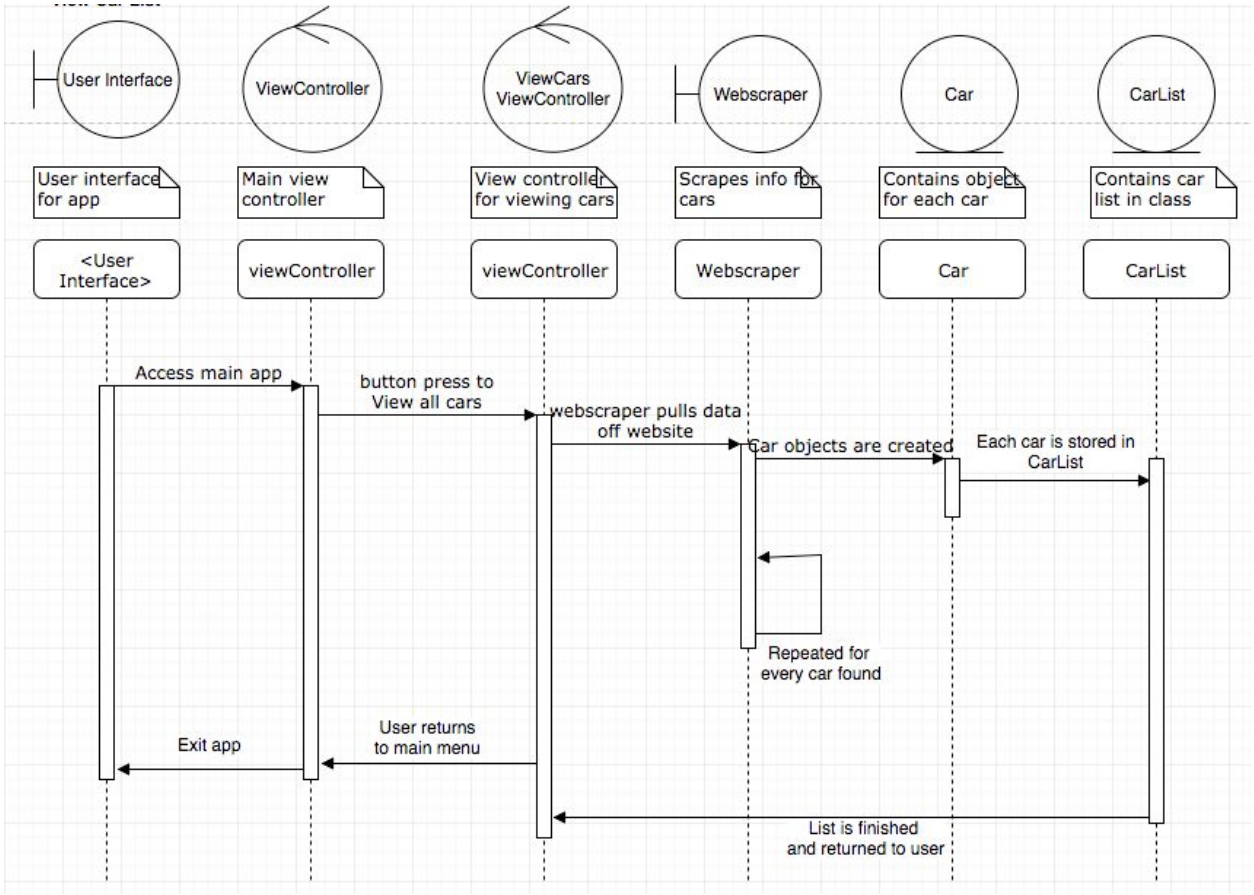


Background calculations

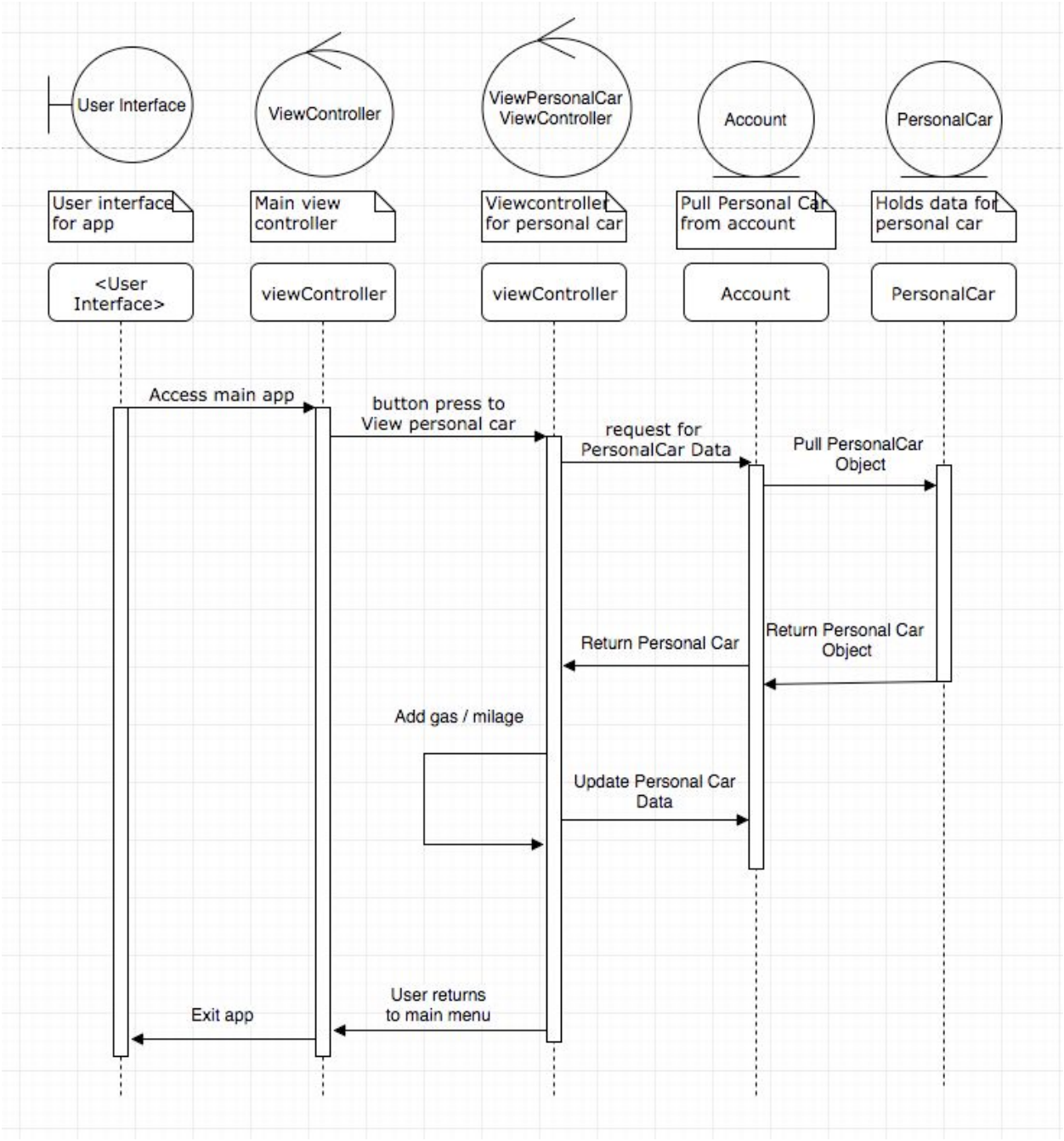


Detailed SSD(s)

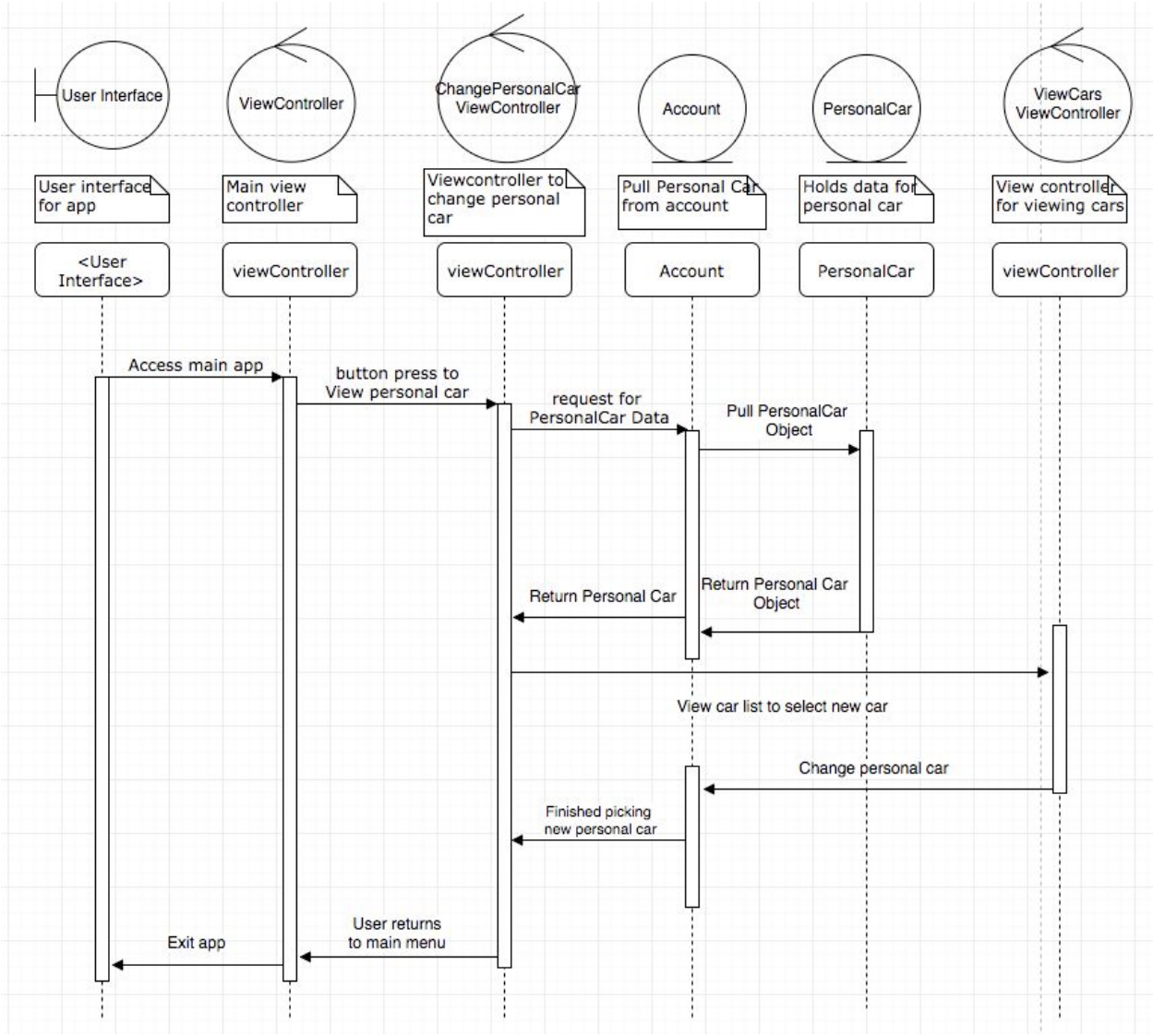
Check Singular Car in Carlist



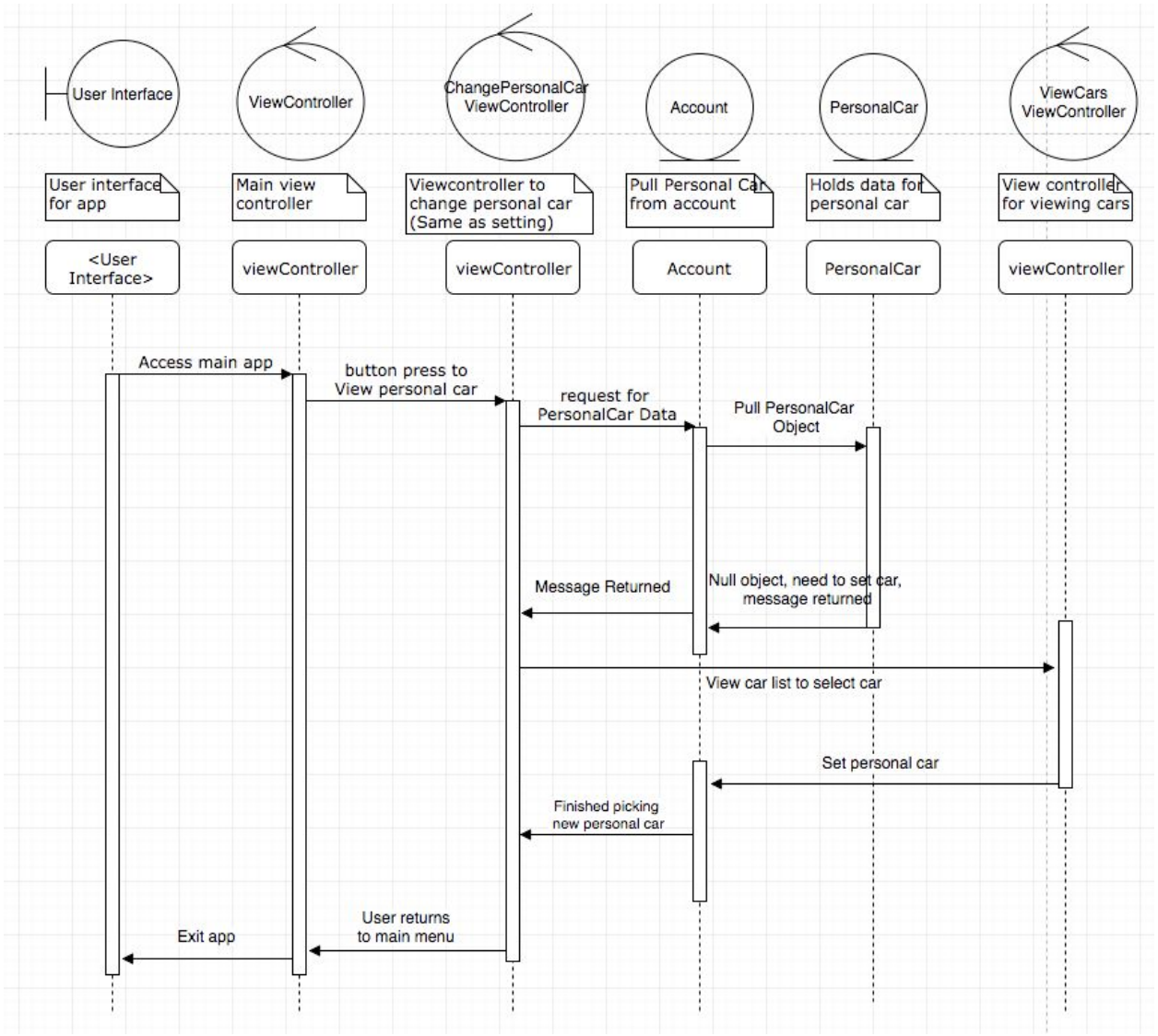
Add Data (Gas consumption and Mileage)



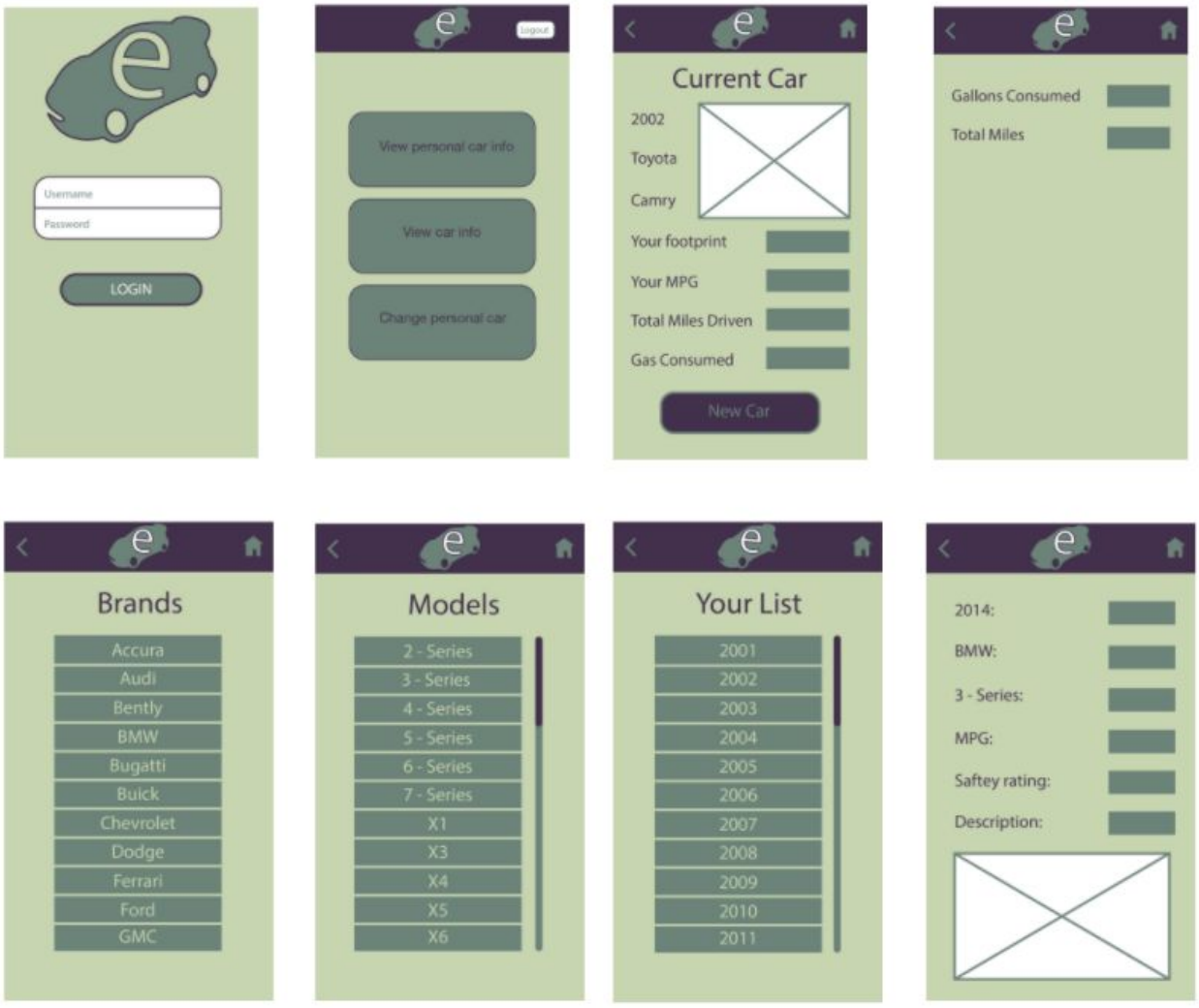
Change Personal Car



Set Personal Car



User Interface Design



From top left to bottom right:

1. Log in screen
2. Main menu
3. Personal Car info
4. Gas + Mileage input screen
5. View car info Brand list
6. View car info Model list
7. View car info Year list
8. View car info for specific car

How my design reflects each of the 8 Golden Rules of design:

Rule 1: Strive for Consistency

The user interface in each panel is completely consistent, from the color pallet, down to the layout of lists, and button layout. Green colors were chosen due to more "environmental" applications choose this color to reflect nature, and its cleanness. Additionally, the top portion of the application act as the navigation area, allowing the user to go back or to the home (main menu) whenever necessary.

Rule 2: Enable Frequent Users to Use Shortcuts

The main shortcuts allowed in this application will be found on the top bar: the home and back buttons. Being that this interface allows the user to be nested in a very linear path, we want those users who know the app and do not want to go all the way through picking a car year for a specific car, to know they can hit the home button and go back to the beginning. A back button is embedded in the application so the user may go back any number of screens in the application.

Rule 3: Offer Informative Feedback

Any type of buttons pressed in the application will appear like they are being physically pressed, allowing the user to get the feedback they need to know that something was pressed. Any invalid information will not be accepted, resulting in a default pop-up box appearing on the screen.

Rule 4: Design Dialog to Yield Closure

Any action that requires the user to input data, such as gas consumed and current miles, will have a "submit" button, which when pressed, and the data is validated, then a "success" message will be displayed that it was accepted.

Rule 5: Offer Simple Error Handling

Although I attempt to make the system in such a way that errors will not happen, I am sure certain data input will result in errors. Having data checks on negative / extremely large numbers, making sure the user has a valid account, etc. will all have the correct error handling to ensure the data is checked properly / errors are dealt with in a proper manner to ensure the application will not fail.

Rule 6: Permit Easy Reversal of Actions

To allow the user to reverse their actions, a back button was placed to go back to every pane in the path they are currently on. One main area they may wish to update / change is their current car, which will allow the user to change their car later on if they

need to do so. Additionally, seeing their full list of their gas inputs allows the user to edit / delete any input, in case of incorrect input.

Rule 7: Support Internal Locus of Control

This application allows the user to freely browse and input any data they need into the application. This data input covers their current car as well as gas and milage input. Anything the user wants to do, the user has the option to do, allowing them to believe that they have full control over anything inside the application.

Rule 8: Reduce Short-Term Memory Load

The user will have a delightful time using the application due to its ease of use. The flow of the application is easy to pick up for first time users, and these users will become very experienced shortly after use, and will become in tune with using any short cuts. This will allow them to just focus on what they're currently working on. All data is shown in a progressive manner, and the defaults for the application establish a mindful start up to the application.

Modularity, Encapsulation, ALgorithms, Data Structures

Modularity gives any application the ability to find any issues with testing, if an issue were to arise, and make the application more easily understood. My application will be broken down into very modular components to suit this, as well as encapsulate data by only having the data the particular objects needs within it. Additionally, data hiding will be used to make any class attribute private, and having respective getters and setters to mutate / access the data respectively. The main algorithm of calculating the footprint for a user based on gas, miles driven, personal vehicle, etc. will be created in an elegant manner. In speaking with the environmental department at TCNJ, I have the contact for a social scientist (Dr. Miriam Shakow) in the department to get an idea of the direction for the algorithm basis, and an environmental chemist (Dr. Mike Aucott) for the computational part of this application. Together, they will guide me in my search for an efficient algorithmic design for computing the carbon footprint for this application. Searching will be another taxing algorithm my application will have to face, going through all the different vehicles on the EPA's website through my webscraper, as well as storing and searching through them once scraped. A Binary Search Tree will aid me in storing all of the data for the cars, allowing the user to parse through this data as quick as possible. With all things considered, this application's code base will take encapsulation and modularity, efficiency of algorithms, and appropriate data structures into consideration.

Test Case Design

Testing a system such as the one I am designing will require a lot of unit, integration and system testing. For unit testing, I see 3 main components with my project (View personal car and adding gas, view car list, and changing the user's personal car), that intend to test individually throughout development of each component. As the project progresses, I will be integrating each component together, and testing the integration of these parts to ensure that they work well together. Along each step of the way, I will be doing system testing, ensure integration of a certain component does not break, as well as at the end when I believe the project is complete.

For debugging, Xcode has a very robust system for debugging built in. This includes the debug area, debug navigator, adding breakpoints and a source editor. All of this is encapsulated in the IDE itself, so I can test while I am developing in a very easy manner. The following table will be followed throughout development for testing of the various components of the system.

Test Cases			
Functionality Test	Input	Expected Output	Actual Output
View Personal Car	Select “View Personal Car”	Correct car is selected	
Change Personal Car	Query database for car, if it exists, info is displayed. User hits change personal car, UIAlert notifies user if this change too effect	Current car is changed to new car, UIAlert notifies user of change	
Change Personal Car	Query database for car, but car isn’t found	UIAlert pops up saying “invalid car”	
View Personal Car info	Go view personal car data	See all personal data regarding car, all data should be correct.	
View Cars in database	Go to query cars in database with valid car	Valid car is output to the user’s screen	
View Cars in database	Go to query cars in database with invalid car	UIAlert says car doesn’t exist, no car is pulled	