Report 2

Course:

CSCI 441 VA Software Engineering

Project Title:

A Web Based Application for Vehicle Sales, Purchase and Inventory Management

URL of Project Website:

https://web-app-for-vehicle-sales-dev.herokuapp.com/

Submission Date:

03/15/2021

Team-Members:

Brady Neumann
Fatih Gurbuz
Ryan Shepard
Seth Whitaker
Taylor Cook

Breakdown of Contributions

All team members contributed equally. As such, each member is listed below.

Taylor Cook
Fatih Gurbuz
Brady Neumann
Seth Whitaker
Ryan Shepard

Table of Contents

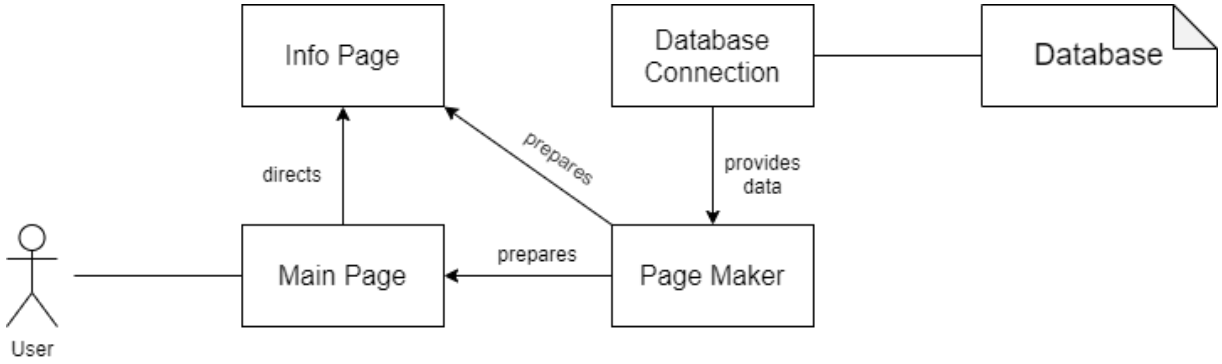
Cover Page: 1
Breakdown of Contributions: 2
Table of Contents: 3
Conceptual Model: 4
System Operation Contracts: 13
Data Model and Persistent Data Storage: 21
Interaction Diagrams: 22
Class Diagram: 24
Interface Specifications: 25
User Interface Design and Implementation: 32
Design of Tests: 37
Project Management and Plan of Work: 44

Conceptual Model

Domain Analysis

Given the use cases ViewCarInfo, Checkout, CrudSalesData, and CrudInventoryData, it is inevitable that there will be a Main Page, Info Page, Checkout Page, Login Page and a Cart Page. Almost all of the information that will be displayed on these pages will be stored in the Database. Using a concept, namely Database Connection, the process of retrieving information from the database can be better visualized. An intermediary platform for making connections between the database and the front-end pages is also needed. A concept named Page Maker could be useful in depicting such a platform. For more advanced operations such as user login to the website or filling out forms, concepts such as Authenticator and Data Confirmer (for checking whether the input data fits to the requirements) are necessary. When a user purchases an item, a message, possibly an email, is needed for both the user and the vendor. For this purpose, concepts such as Actor Updater and Email Sender could be used, which will update the actors and send emails if necessary. Lastly and most importantly, making CRUD operations would require a concept, which could be called simply as CRUD Operator, to be active.

ViewCarInfo (UC-5)



Concept Definitions

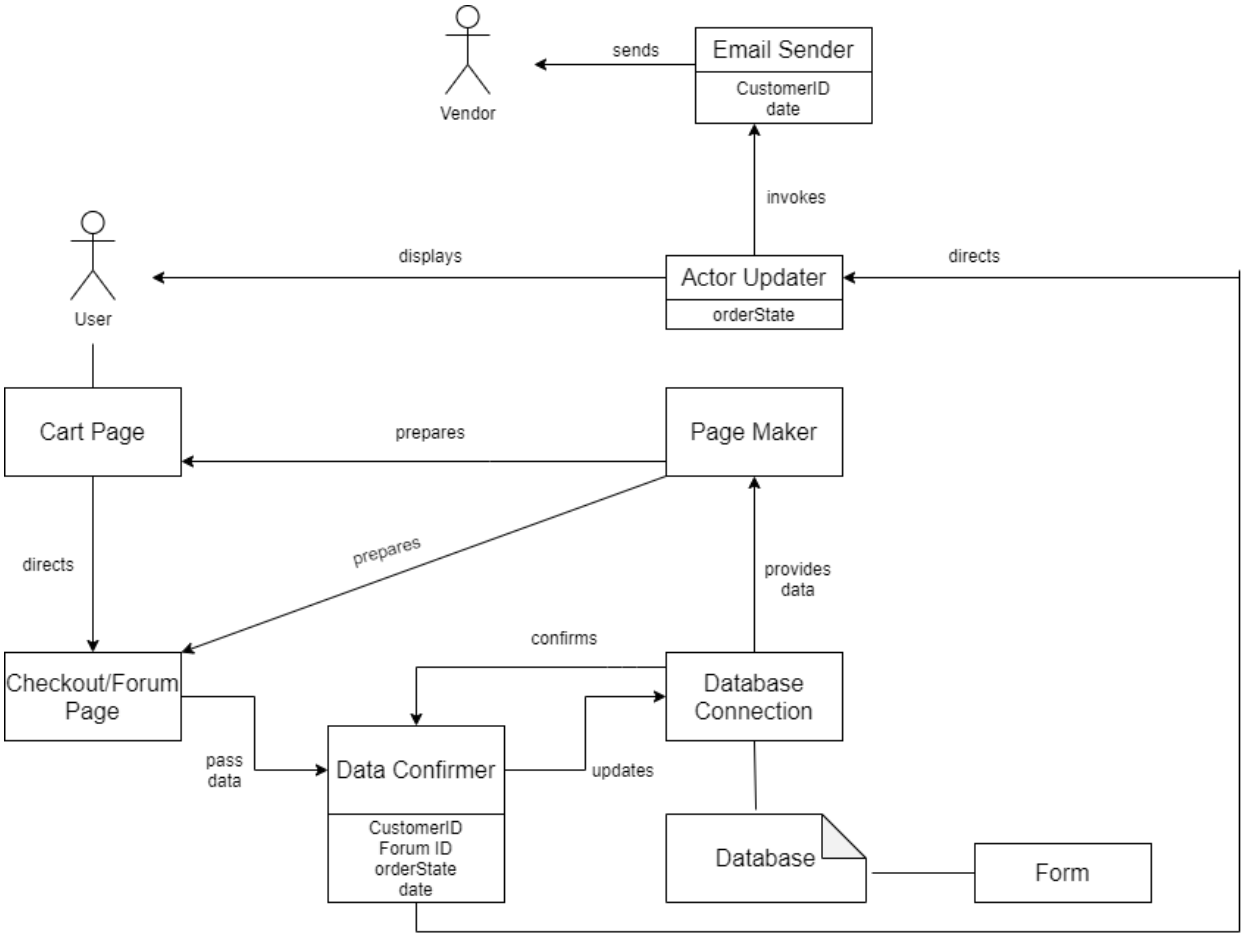
Responsibility Description	**Type**	**Concept Name**
Prepares a database query based on the specified cars.	D	Database Connection
The main HTML document that actor can use to interact with the application	K	Main Page
HTML document that shows information about the specified car in a conventional way.	K	Info Page
Render the retrieved records into an HTML document for sending to the actor's browser for display.	D	Page Maker

Association Definitions

Concept pair	**Association description**	**Association name**
Database Connection ↔ Page Maker	Database Connection passes the retrieved data to Page Maker to render them for display.	provides data
Page Maker ↔ Main Page	Page Maker prepares the associated page	prepares
Page Maker ↔ Info Page	Page Maker prepares the associated page	prepares
Main Page ↔ InfoPage	A click on a car card directs the actor from	directs

	the Main Page to the Info Page of the particular car.	

Checkout (UC-9)



The user starts the checkout process on the Cart Page. Then, a button on the page directs the user to the Checkout/Forum page. On this page, the user is expected to fill out a forum, which will send data to the database after all the data is confirmed. After the confirmation, the user will get a message on the screen about the success of the purchasing process. Meanwhile, the vendor will get an email that his/her product has been purchased.

Concept Definitions

Responsibility Description	Type	Concept Name
Prepares a database query based on the specified request.	D	Database Connection
The main checkout page that actors can use to fill out the Form.	K	Checkout/Form Page
HTML document in which items selected by the actor is displayed	K	Cart Page
Render the retrieved records into an HTML document for sending to the actor's browser for display.	D	Page Maker
Send email to the Vendor about purchase with the required information.	D	Email Sender
Check the data on Form Page and confirm the given data if it fits the requirements.	D	Data Confirmer
Display a success page and update the user regularly on the purchase	D	User/Actor Updater
Container for the actor’s information that is necessary for a possible purchase.	K	Form

Association Definitions

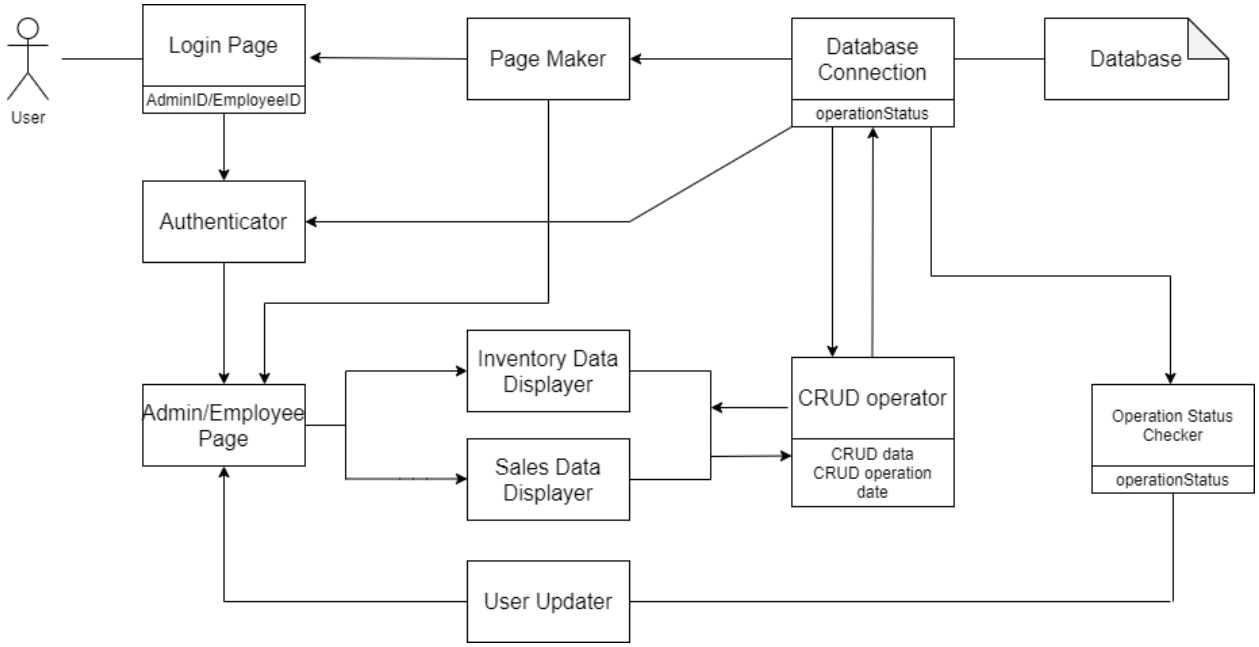
Concept pair	Association description	Association name
Database Connection ↔ Page Maker	Database Connection passes the retrieved data to Page Maker to render them for display.	provides data
Page Maker ↔ Checkout/Forum Page	Page Maker prepares the associated page	prepares
Page Maker ↔ Cart Page	Page Maker prepares the associated page	prepares
Cart Page ↔ Checkout/Forum Page	A click on an item in Cart Page directs the actor from Cart Page to the Checkout/Forum Page	directs
Checkout/Forum Page ↔ Data Confirmer	Checkout/Forum Page passes the data to the Data Confirmer	pass data

Data Confirmer ↔ Database Connection	Data Confirmer updates the Database Connection if data is confirmed	updates
Database Connection ↔ Data Confirmer	Database Connection confirms Data Confirmer on whether the updates have been successfully saved on the database.	confirms
Data Confirmer ↔ User/Actor Updater	Data Confirmer directs the actor from Checkout/Forum page to User/Actor Updater	directs
User/Actor Updater ↔ User	User/Actor Updater displays a success message/page to the actor	displays
User/Actor Updater ↔ Email Sender	User/Actor Updater activates Email Sender to make it perform its duty.	invokes
Email Sender ↔ Vendor	Email Sender sends an email to the Vendor, which includes related information about the purchase.	sends

Attribute Definitions

Concept	**Attribute**	**Attribute Description**
Data Confirmer	CustomerID	A specific ID that each Customer gets when the user account is created.
	FormID	A specific Id that each Form gets when a form is created.
	orderState	A value that describes the current state of an order
	date	The date in which order is given.
Actor Updater	orderState	The value that is changed after the order has been successfully given.
Email Sender	CustomerID	The same CustomerID that Data Confirmer contains. It is used for notifying the Vendor on the purchase.
	date	The date in which order is given.

CrudSalesData, CrudInventoryData (UC-10, UC-11)



The user will start the process by giving his/her credentials on Login Page. After it has been authenticated, the user will be directed to the Admin/Employee page, where he/she will have the privilege to access and update Inventory and Sales Data. When any update has been made to the database, the status of operation (the result of the query) is displayed to the user as a status message on the Admin/Employee page.

Concept Definitions

Responsibility Description	**Type**	**Concept Name**
Prepares a database query based on the specified cars.	D	Database Connection
The HTML document that will be used by the Admin/Employee to Login into the system	K	Login Page
Authenticates the user into the system by checking the given inputs and comparing them to the values	D	Authenticator

storing the database.		
The HTML document that is only used by Admin/Employee, that have the privileges of doing CRUD operations.	K	Admin/Employee Page
Displays Inventory Data by getting data from CRUD Operator and showing them to the actor	K	Inventory Data Displayer
Displays Sales Data by getting data from CRUD Operator and showing them to the actor	K	Sales Data Displayer
Creates SQL queries based on the request of the Admin/Employee Page. Transfers the results of the query to the Inventory and Sales Data Displayer	D	CRUD Operator
Checks the results of the operation and communicates them to the User Updater	D	Operation Status Checker
Render the retrieved records into an HTML document for sending to the actor's browser for display.	D	Page Maker
Display a success page and update the user regularly on the purchase	K	UserUpdater

Association Definitions

Concept pair	Association description	Association name
Database Connection ↔ Page Maker	Database Connection passes the retrieved data to Page Maker to render them for display.	provides data
Page Maker ↔ Login Page	Page Maker prepares the associated page	prepares
Page Maker ↔ Admin/Employee Page	Page Maker prepares the associated page	prepares
Login Page ↔ Authenticator	Login Page passes the data to the Authenticator	pass data
Database Connection ↔ Authenticator	Using the data from Login Page, Authenticator compares values stored in Database Connection and authenticates the user.	authenticates

Authenticator ↔ Admin/Employee Page	Authenticator directs user to the Admin/Employee Page after user credentials have been verified	directs
Admin/Employee Page ↔ Inventory/Sale Data Displayer	Admin/Employee Page displays data from the database through several other concepts.	displays
Inventory/Sale Data Displayer ↔ CRUD Operator	Inventory/Sale Data Displayer connects to the CRUD Operator in order to get required values.	connects
CRUD Operator ↔ Database Connection	CRUD Operator makes queries to the Database Connection and gets the results	makes query
Database Connection ↔ Operation Status Checker	Database Connection informs the Operation Status about the results of the query	informs
Operation Status Checker ↔ User Updater	Operation Status Checker activates User Updater to make User Updater display message about the status of database	invokes
User Updater ↔ Admin/Employee Page	User Updater displays a message on the results of the query.	displays

Attribute Definitions

Concept	Attribute	Attribute Description
Login Page	AdminID/EmployeeID	A specific ID that each Admin/Employee gets when the user account is created.
CRUD Operator	CRUD data	Data that will be created, read, updated or deleted.
	CRUD operation	The operation that determines whether the data will be created, read, updated or deleted.
	date	The date in which operation was executed.
Operation Status Checker	operationStatus	A description about the status of the operation that was executed.

Traceability Matrix

USE CASE	PW	Database Connection	Main Page	Login Page	Info Page	Admin/Employee Page	Cart Page	Checkout/Forum Page	Email Sender	Authenticator	Page Maker	Inventory Data displayer	CRUD Operator	Operation Status Checker	Data Confirmer
UC1	3	X	X								X	X	X		
UC2	3	X	X								X	X	X		
UC3	3	X									X	X	X		
UC4	4	X		X					X	X	X				X
UC5	4	X	X		X						X	X	X		
UC6	4	X	X	X	X			X	X	X	X				X
UC7	4	X	X			X				X	X	X	X	X	X
UC8	4	X	X			X				X	X	X	X	X	X
UC9	5	X					X	X	X	X	X		X		X
UC10	5	X		X		X				X	X	X	X	X	
UC11	5	X		X		X				X	X	X	X	X	
UC12	4	X	X	X		X				X	X		X	X	
UC13	4	X	X	X		X				X	X		X	X	
UC14	4	X		X					X	X	X		X	X	X

System Operation Contracts

I. ViewCarInfo

A.

Operation:	displayResults(search, filter, sort)
Cross references:	ViewCarInfo
Preconditions:	• The customer is viewing the application on their browser
Postconditions:	• A search instance was created. • Attributes of filter and sort have been modified.

B.

Operation:	carInfo()
Cross references:	ViewCarInfo
Preconditions:	• The customer is viewing the application on their browser
Postconditions:	• No notable post conditions. No instances or associations were formed or broken and no

	attributes have been changed.

II. Checkout

A.

Operation:	checkoutForm(customerID, formID)
Cross references:	Checkout
Preconditions:	● The customer has items in their cart ● The customer has clicked the checkout button
Postconditions:	● A form was created and associated with the customer. ● Basic attributes of the form were initialized.

B.

Operation:	submitForm(customerID, formID, date)
Cross references:	Checkout
Preconditions:	● The customer has items in their cart

	● The customer has clicked the checkout button
Postconditions:	● The form is placed into pending orders for the customer. ● The state of the order is changed upon approval. ● The order is associated with the customerID and the date.

III. CrudSalesData

A.

Operation:	readSalesData(sales:Sales)
Cross references:	CrudSalesData
Preconditions:	● Admin/employee is logged in ● Admin/employee is viewing application interface for CRUD changes
Postconditions:	● A connection is formed with the database and associated with sales data.

B.

Operation:	createSalesData(sales:Sales)
Cross references:	CrudSalesData
Preconditions:	● Admin/employee is logged in ● Admin/employee is viewing application interface for CRUD changes
Postconditions:	● A connection is formed with the database ● A new instance of sales is created. ● The new instance is associated with the database based on matching attributes

C.

Operation:	updateSalesData(sales:Sales)
Cross references:	CrudSalesData
Preconditions:	● Admin/employee is logged in ● Admin/employee is viewing application interface for CRUD changes
Postconditions:	● A connection is formed with the

	database. ● The attributes of a sales instance are modified. ● The modified sales instance is associated with the database based on matching attributes.

D.

Operation:	deleteSalesData(sales:Sales)
Cross references:	CrudSalesData
Preconditions:	● Admin/employee is logged in ● Admin/employee is viewing application interface for CRUD changes
Postconditions:	● A connection is formed with the database. ● An instance of sales is removed. ● The removed instance of sales is dissociated with the database.

IV. CrudInventoryData

A.

Operation:	readInventoryData(inventory:Inventory)
Cross references:	CrudInventoryData
Preconditions:	● Admin/employee is logged in ● Admin/employee is viewing application interface for CRUD changes
Postconditions:	● A connection is formed with the database and associated with inventory data.

B.

Operation:	createInventoryData(inventory:Inventory)
Cross references:	CrudInventoryData
Preconditions:	● Admin/employee is logged in ● Admin/employee is viewing application interface for CRUD changes
Postconditions:	● A connection is formed with the database. ● A new instance of inventory is created.

	• The new instance is associated with the database based on matching attributes.

C.

Operation:	updateInventoryData(inventory:Inventory)
Cross references:	CrudInventoryData
Preconditions:	• Admin/employee is logged in • Admin/employee is viewing application interface for CRUD changes
Postconditions:	• A connection is formed with the database. • The attributes of an inventory instance are modified. • The modified inventory instance is associated with the database based on matching attributes.

D.

Operation:	deleteInventoryData(inventory:Inventory)

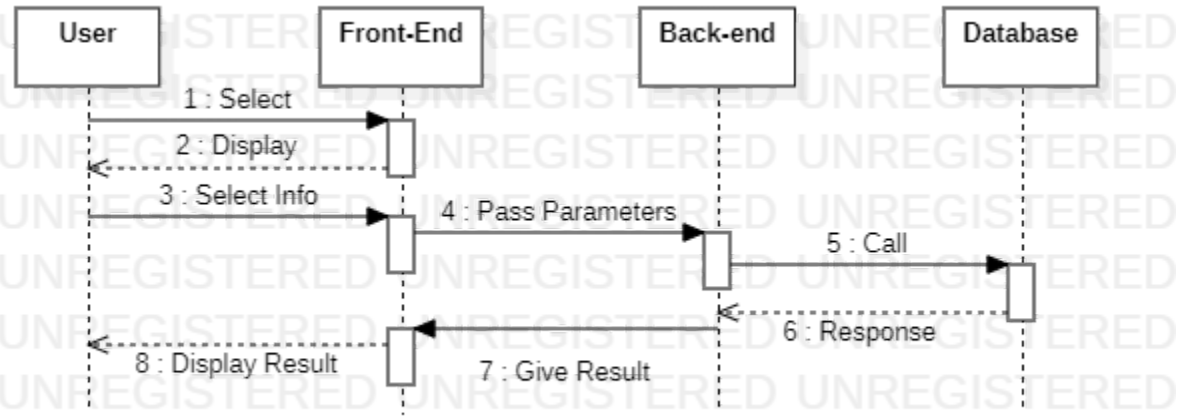
Cross references:	CrudInventoryData
Preconditions:	● Admin/employee is logged in ● Admin/employee is viewing application interface for CRUD changes
Postconditions:	● A connection is formed with the database. ● An instance of inventory is removed. ● The removed instance of inventory is dissociated with the database.

Data Model and Persistent Data Storage

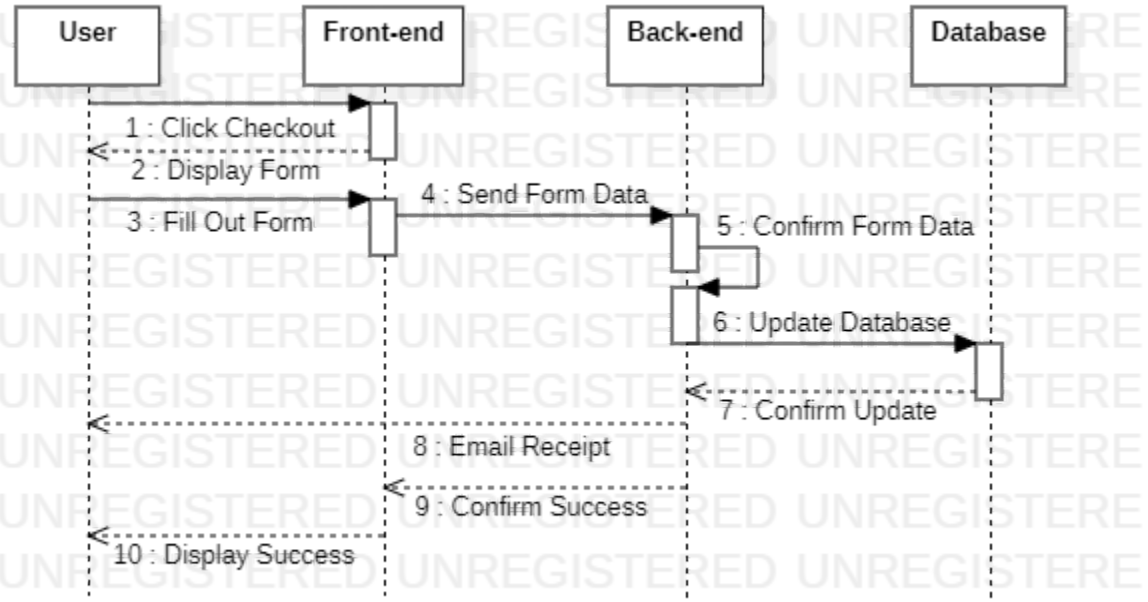
We will be using a relational database to store and reference data as well as deploying our web application using Heroku. The use of these will ensure that data can outlive a single execution of our web application. Objects that will be stored in the database include: sales, inventory, employees, customers, and cart.

Interaction Diagrams

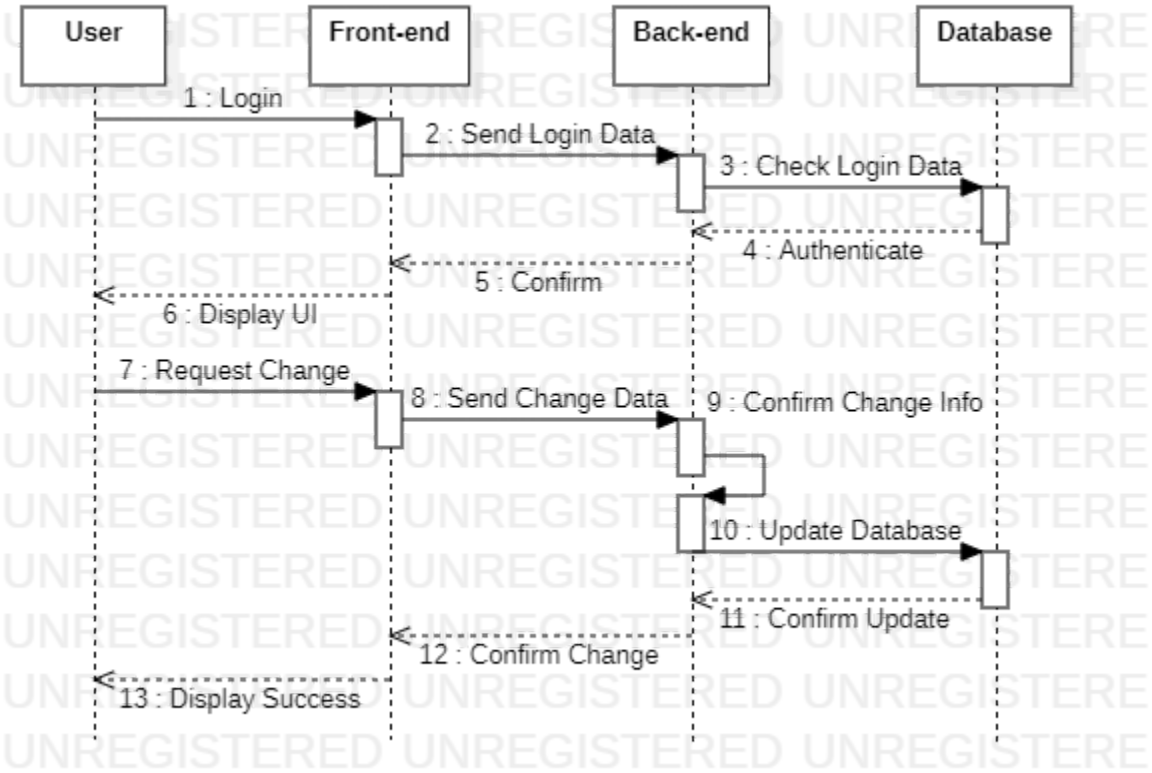
ViewCarInfo (UC-5)



Checkout (UC-9)

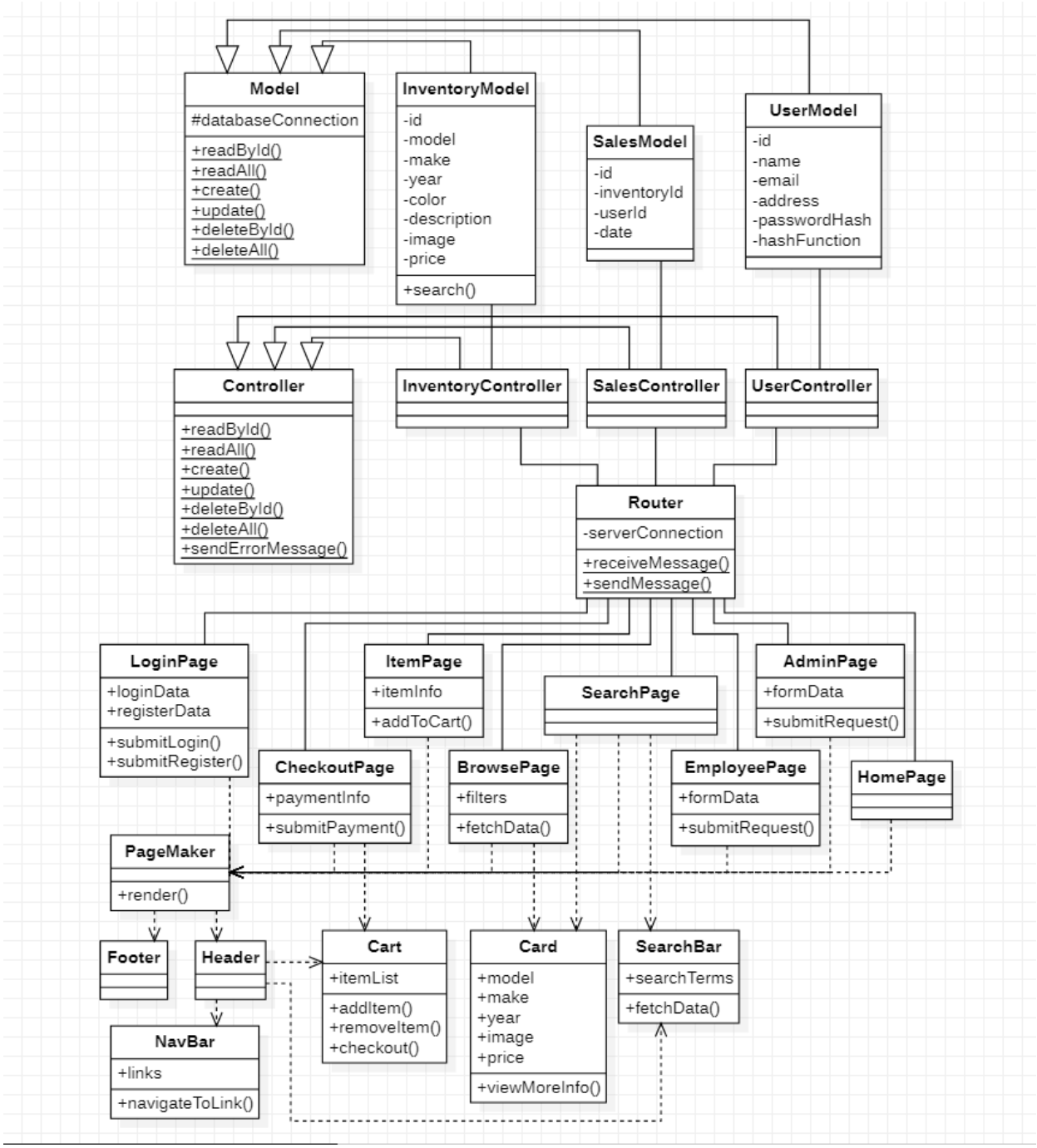


CrudSalesData, CrudInventoryData (UC-10, UC-11)



Responsibilities were given based on general guidelines for web application development. The user only interacts with the front-end, obscuring the moving gears of the back-end and database. This allows data entered by the user to be parsed correctly, and no malicious or accidental errors to be created.

Class Diagram



Interface Specifications

Data Types

Router: The router is the central hub that the website will use to connect with the server and to send and receive messages.

Inventory Controller: Mainly links to the central Controller and the Inventory Model.

Sales Controller: Mainly links to the central Controller and the Sales Model.

User Controller: Mainly links to the central Controller and the User Model.

Controller: Reads, creates, updates, deletes, or informs the user of an error using the Inventory, Sales, and User controllers.

Inventory Model: Will store the inventory of the car's features, such as the make and model, the price, descriptions of features, as well as a search function for the user to search the inventory.

Sales Model: Mainly tracks the sales and updates both the inventory and user when changes are made.

User Model: Stores user information such as username, password, email address, and any other information.

Model: Connects to a database and manages all information sent by the Inventory, Sales, and User models.

Login Page: Web page that displays login and registration information from users.

Item Page: Web page that displays information related to items, in this case vehicles.

Search Page: Web page that displays information related to user searches.

Admin Page: Web pages accessible only to site administrators showing various form data and features.

Checkout Page: Web page that displays checkout information after a user has placed an order.

Browse Page: Web page that allows users to browse the catalog.

Employee Page: Web pages accessible only to employees allowing them limited access to make necessary updates.

Home Page: Web page that is the first page a user will see when first accessing the website.

Page Maker: Renders all of the web pages using the provided information.

Footer: Static bottom information of the web page showing contact information and various links.

Header: Static top information of the web page showing links to parts of the website and necessary user information,

Nav Bar: Static navigation bar on the header of the page allowing the user easy access to the rest of the website.

Cart: Storage of user checkout information that displays on the header.

Card: Storing information and images related to vehicles.

Search Bar: Static bar on the web page allowing users to search the website.

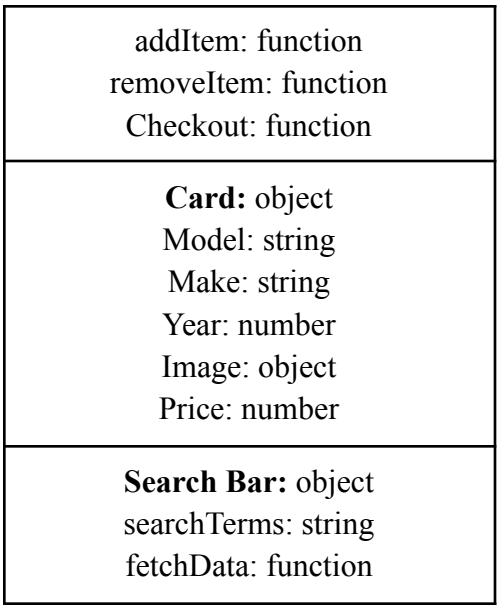
Operation Signatures

Router: object serverConnection: function receiveMessage: function sendMessage: function
Inventory Controller: object
Sales Controller: object
User Controller: object
Controller: object readByID: number

readAll: string create: string update: string deleteByID: number deleteAll: string sendErrorMessage: string
Inventory Model: object Id: number Model: string Make: string Year: number Color: string Description: string Image: object Price: number Search: function
Sales Model: object Id: number inventoryId: number userId: number Date: string
User Model: object Id: number Name: string Email: string Address: string passwordHash: string hashFunction: function
Model: object readByID: number readAll: string create: string update: string deleteByID: number deleteAll: string
Login Page: object

loginData: string registerData: string submitLogin: function submitRegister: function
Item Page: object itemInfo: string addToCart: function
Search Page: object
Admin Page: object formData: string submitRequest: function
Checkout Page: object paymentInfo: string submitPayment: function
Browse Page: object Filters: object fetchData: function
Employee Page: object formData: string submitRequest: function
Home Page: object
Page Maker: object Render: function
Footer: object
Header: object

Nav Bar: object Links: string navigateToLink: function
Cart: object itemList: string


addItem: function
removeItem: function
Checkout: function
Card: object
Model: string
Make: string
Year: number
Image: object
Price: number
Search Bar: object
searchTerms: string
fetchData: function

Traceability Matrix

Class	Database Connection	Main Page	Login Page	Info Page	Admin/Employee Page	Cart Page	Checkout/Form Page	Email Sender	Authenticator	Page Maker	Inventory Data Displayer	CRUD Operator	Operation Status Checker	Data Confirmer
PW	5	3	3	4	4	3	3	1	2	5	4	4	1	2
Model	X	X	X	X	X	X	X		X		X	X		
InventoryModel	X	X		X	X	X	X				X	X		
SalesModel	X			X	X		X					X		
UserModel	X		X		X	X	X		X			X		
Controller	X	X	X	X	X	X	X	X	X		X	X	X	X
InventoryController	X	X		X	X	X	X				X	X		X
SalesController	X				X		X					X		X
UserController	X		X		X	X	X	X	X			X		X
Router	X	X	X	X	X	X	X	X	X			X	X	
PageMaker		X	X	X	X	X	X			X	X			
LoginPage			X											
CheckoutPage							X							
ItemPage				X										
BrowsePage											X			
SearchPage											X			
EmployeePage					X							X		
AdminPage					X							X		
HomePage		X												
Footer										X				
Header										X				
NavBar										X				
Cart						X								
Card											X			
SearchBar										X				

Some of the classes evolved from the implementation of the MVC (Model, View, Controller) concept of web development. This type of design concept divides the file structure into three categories. The controller being the bridge between model (data) and view (front-end) and controlling the flow. Classes such as Model, InventoryModel, SalesModel, and UserModel are under the “Model” portion of the design. Similarly, any class with a name containing “controller” is part of the “Controller” portion (Router, and PageMaker class being the odd one out). This leaves all the rest of the classes in the “View” category, which is all user-facing.

The vast majority of the concepts were matched with a category of the MVC design class-structure in a simple manner. For example, the “Database Connection” and “CRUD Operator” concepts only interact with classes in the “Model” and “Controller” categories, as they only focus on behind-the-scenes work. Other classes were created simply to satisfy a singular concept. The “PageMaker” concept for example did not fall directly into any category, so a class was made to ensure the concept was included.

User Interface Design and Implementation

I. Modifications from Mock Ups

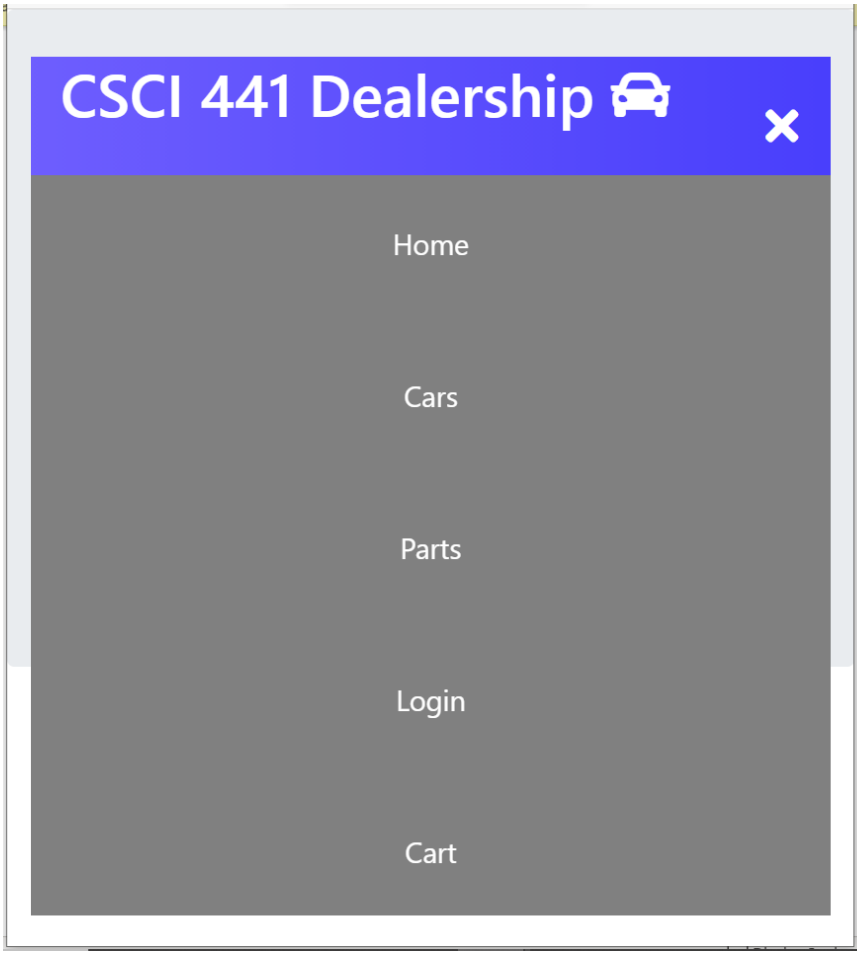
We still have a lot of implementation to do, but we will look at the difference between the mockups in report one with what we have implemented so far.

A. Navigation Menu

The Navigation menu has roughly the same functionality as the one in the mockup except for one thing, there is no search bar in our implementation. This will increase the user effort as they will not be able to search for a specific product directly from the navigation menu. We will most likely implement a search bar in the future to reduce the user effort and make it the same as the mockup.

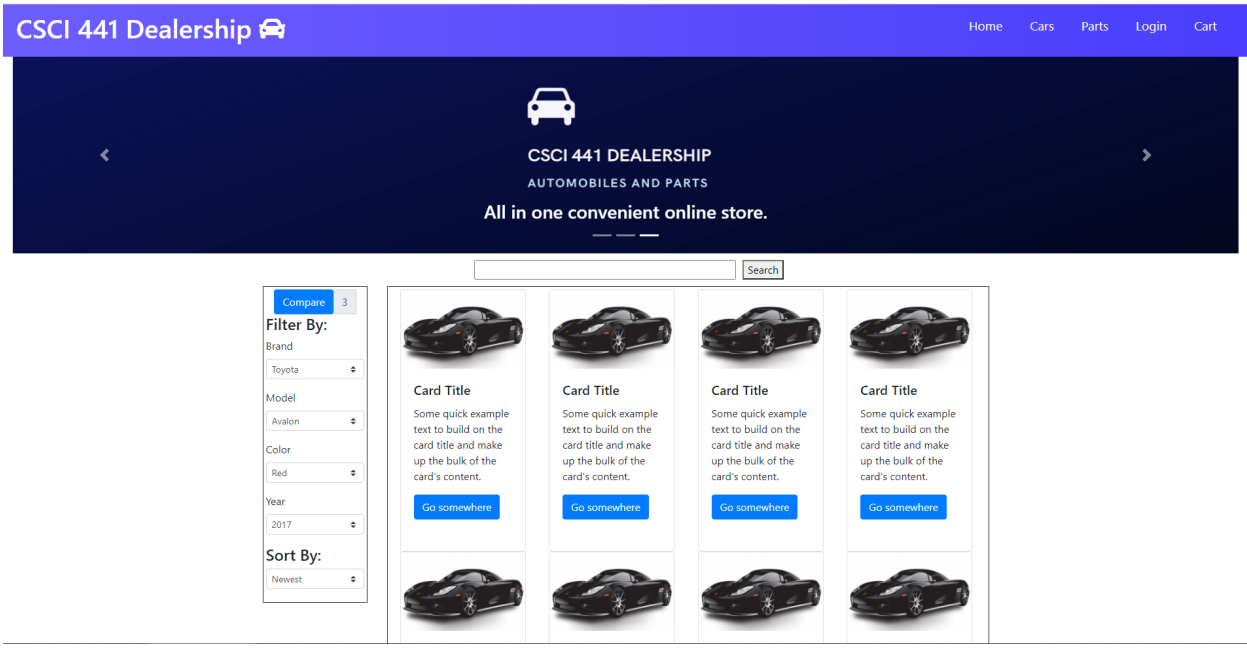
One improvement from the mockup is that there is styling for mobile devices. The navigation menu options are hidden for mobile devices until the user clicks the bars in the top right corner. From there, they have the same options that someone using a desktop has, just in an easier to navigate form factor. This does increase

the user effort by one click because they have to click the bars to see the page links. However, this feature increases the ease of use because it ensures that the users on mobile devices don't have to click tiny buttons from the navigation bar being shrunk down significantly.



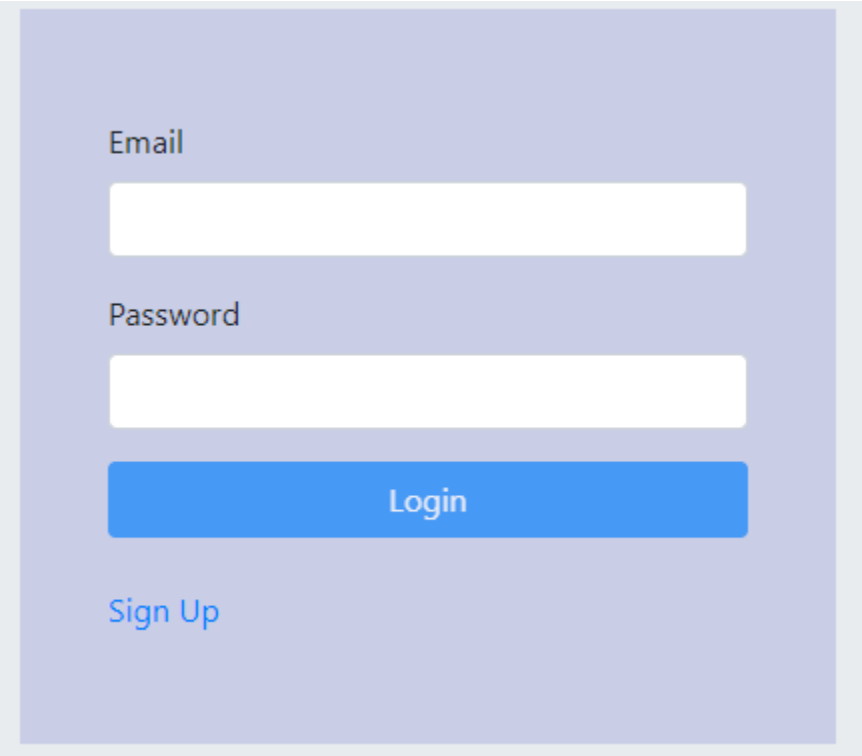
B. Homepage

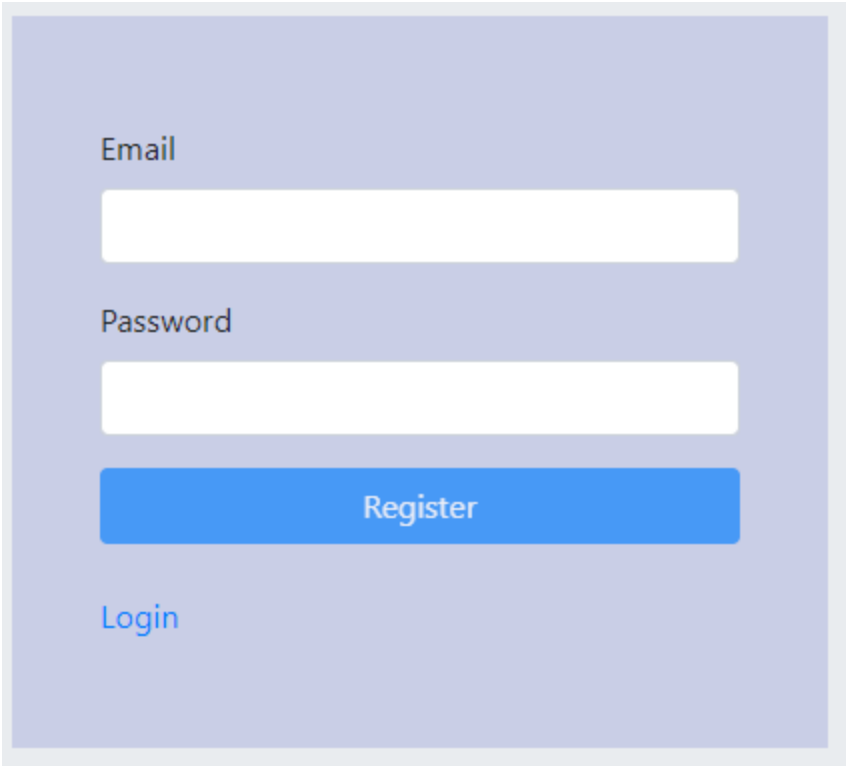
Our implementation of our homepage is a vast improvement from the one seen in the mock up. There are many different components to this homepage that will reduce the user effort. We have featured products that the user can click on to take them directly to the product page, rather than them having to go to a product listing page and then click on the product. There is a search bar at the top of the page so that a user can find any specific product that they want. There is a compare button that the user can use to easily compare two different products without having to go to each individual product page. Finally, there are filter and sort inputs that reduce the user's effort in finding a product that matches their criteria.



C. Login/Register

The Login page looks very familiar to the one that was shown in the mockup, a very simple form that asks for the email address and the password. There is one difference though; there is a link below the login button that the user can click to create an account. After the user clicks the sign up link, they will be taken to another form where they can enter an email and password to create an account. From there, they can login with their newly created account.




Email
Password
Register
Login

Design of Tests

Test Cases

Test-case Identifier: TC-5 **Use Case Tested:** UC-5 **Pass/fail Criteria:** The test passes if the user clicks on a car info button on the homescreen and successfully gets an information page about the selected car. **Input Data:** the car id, obtained by the user click	
Test Procedure	**Expected Results**
Step 1. Click on a car info button on the car card	System directs the user to a new page which displays information about the selected car.

Test-case Identifier: TC-9 **Use Case Tested:** UC-9 **Pass/fail Criteria:** The test passes if the user clicks on the checkout button, fills out a form, and gets a success message. **Input Data:** the car id, form data given by the user	
Test Procedure	**Expected Results**
Step 1. Click on checkout button, fill out the form correctly Step 2. Click on the checkout button, fill out the form incorrectly	System displays a success message informing the user. System displays an error message, requesting the user to try again.

Test-case Identifier: TC-10, TC-11 **Use Case Tested:** UC-10, UC-11 **Pass/fail Criteria:** The tests passes if the operation requested by the user is successfully done **Input Data:** a CRUD operation	
Test Procedure	**Expected Results**
Step 1. Perform a valid CRUD operation on the system Step 2. Perform an invalid CRUD	System displays a success message, the result of the CRUD operation can be instantly viewed on the system. System displays a fail message. The user can try

operation	to perform a different CRUD operation.

Test Coverage

There are mainly four code coverage criteria, namely, equivalence testing, boundary testing, control-flow testing and state-based testing. Due to the relatively simple design and implementation of the program and most of the testing strategies being overtly sophisticated, (thus making the testing process unnecessarily long and nonsense) only some of the strategies could be used when testing the program.

There are several important points that require attention in the application. These points can be divided into two. Firstly, the program should be seen as a product that should have a functional design. In this perspective, one should consider the transition of a program from one state or page to another, general structure of a program, and the usability of the program. Secondly, the program could be seen from a “data” point of view. This aspect is where most of the software testing strategies and methods could be applied and tested.

Equivalence Testing

The partitioning of programs is not very diverse in our application. Since the program mostly deals with simple data and the feedback to the this data is either success or failure, partitioning the value space into one valid (successful) and one invalid (failure) equivalence class, as in boolean values of FALSE and TRUE, seems to be the only plausible solution, if an equivalence testing will be applied.

Boundary Testing

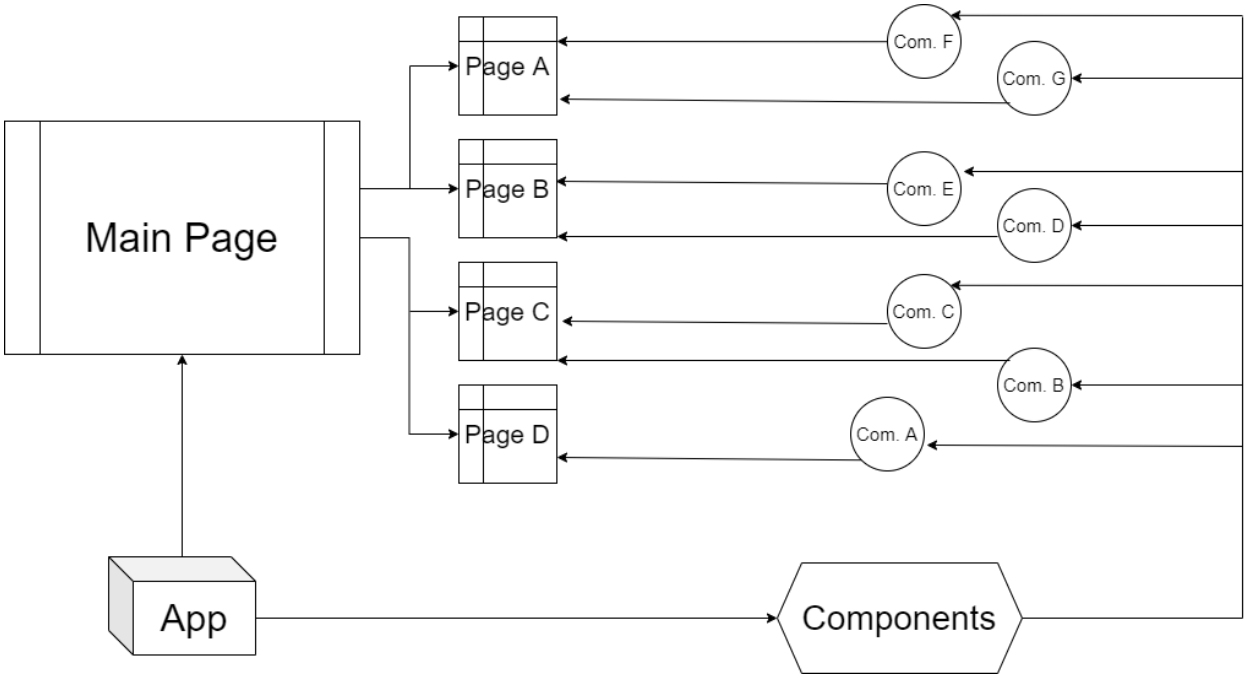
Boundary testing can be viewed as a part of equivalence testing in which boundaries, or “edges” are tested. Such boundaries in our programs could only be null values, or empty strings, which could easily be implemented as a part of equivalence testing.

Control Flow Testing

In order to make sure that the user does not face any errors, or software bugs while using the program, it is crucial to eliminate any possible situation that may lead to unexpected or unforeseen circumstances. Control flow testing allows a tester to reduce the number of bugs by minimizing the events that may create a problem for the user. Therefore, except the Path Coverage part of the control flow testing, which may not be applicable in all situations, all other coverages, such as Statement Coverage, Edge Coverage, and Condition Coverage could be applied.

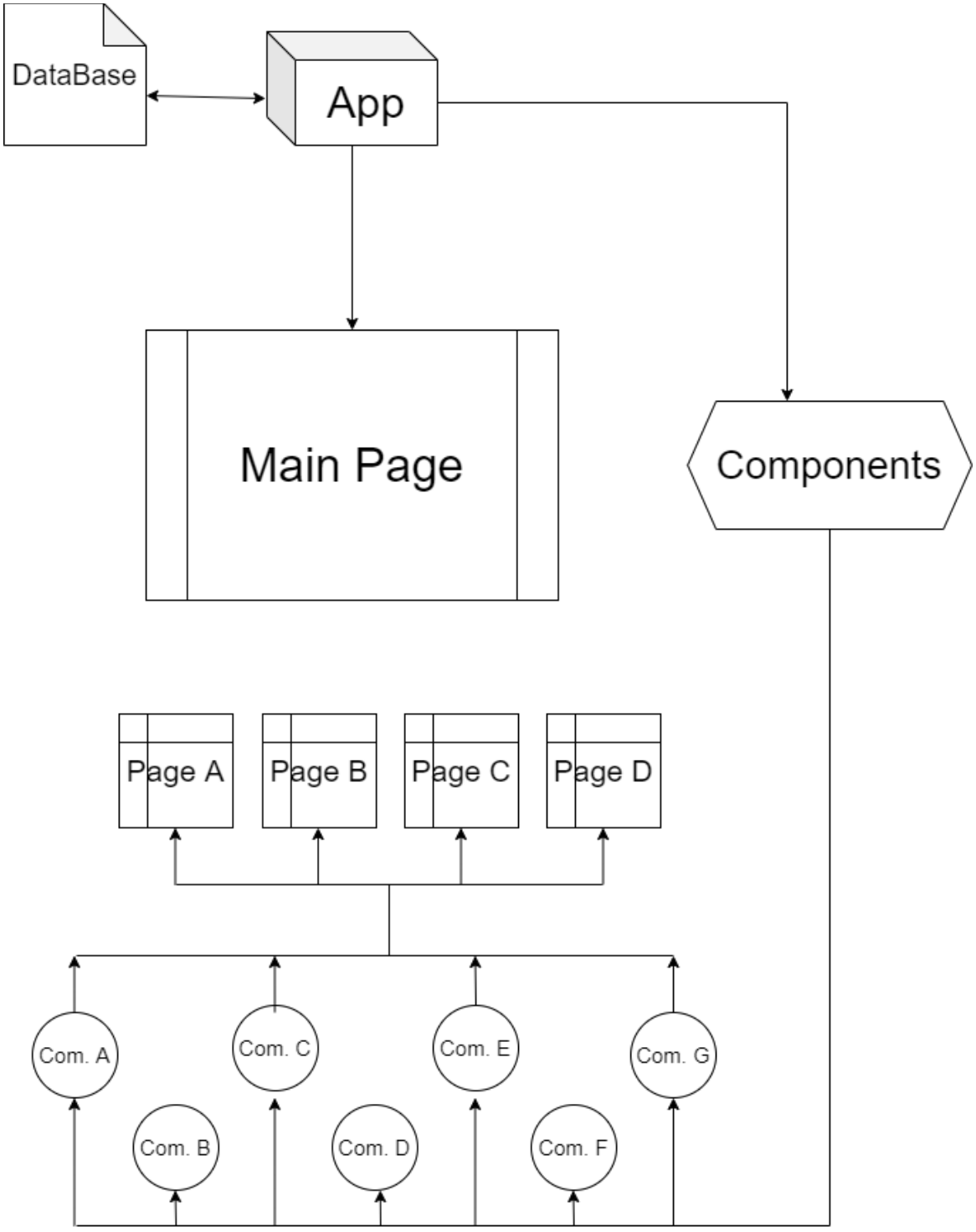
Integration Testing

Before mentioning about the Integration Testing, let’s have a look at the general conceptual structure of the application.



Conceptually, the main part of the program is a file named App, in which all the data communication is done. Main page is the second most important part of the program, which directs users to other pages. All the other pages are having a third place of importance and they contain components. Lastly, we have components, which are the smallest pieces of the program, also having the least level of importance.

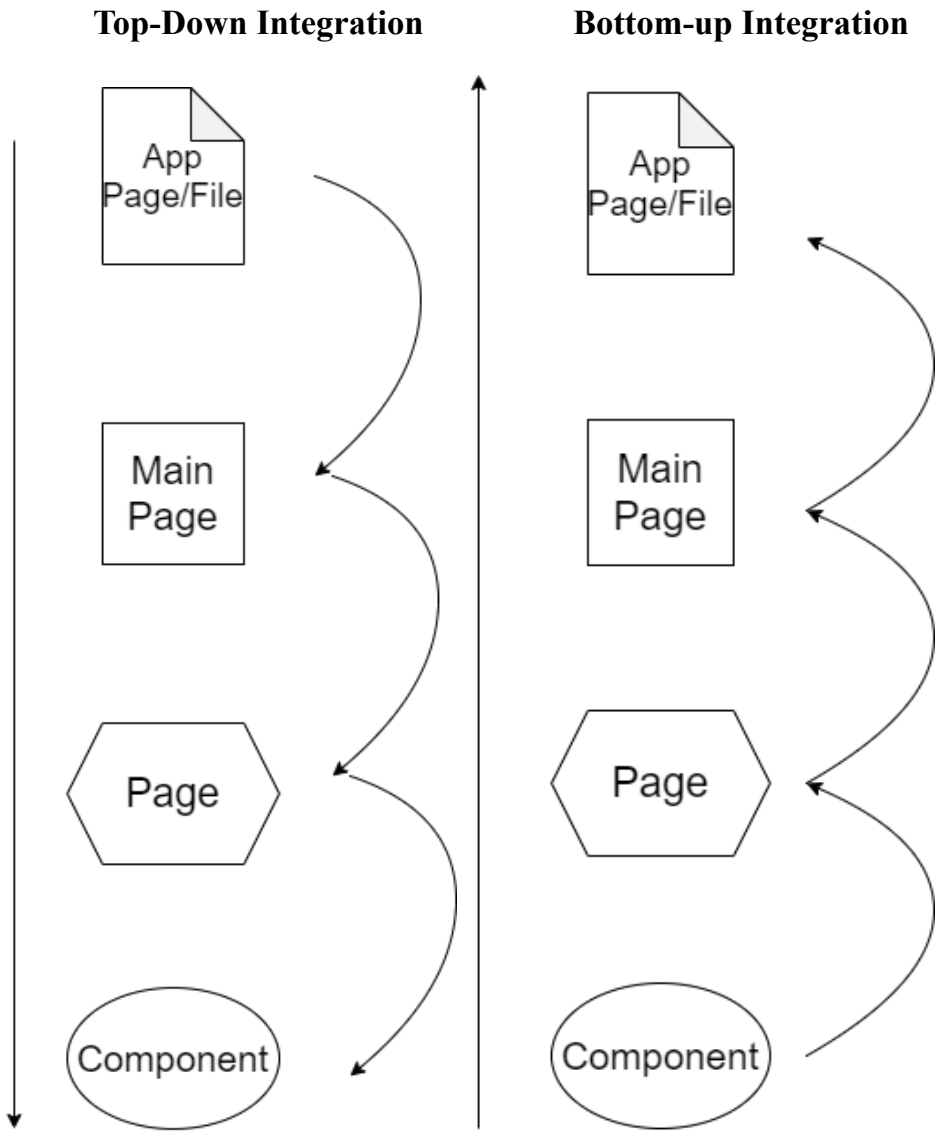
However, from the point of view of a software engineer or a software tester, the program can be restructured (viewed) in a much better way, as such:



When we look from this perspective, it is easier to have an idea of which kind of testing strategies and methods could be used.

Horizontal Integration Testing

Horizontal Integration Testing includes different approaches, such as bottom-up integration, top-down integration and sandwich integration, which combines both bottom-up and top-down integration into one by having a middle level “sandwiched” by top and bottom. Sandwich integration may not be efficient for this program but bottom-up integration and top-down integration could easily and effectively be used.



Vertical Integration Testing

Having only 14 user cases, vertical integration testing strategies doesn't seem to be a very plausible testing solution for a project of such a range.

Security Testing

There are, in total, two user cases in which security testing could be done. The use case, SignUp (UC-4) and Authenticate(UC-14), are two critical components of the program in which security leaks could occur. The authentication part of the program should work flawless in order to prevent security issues and smoothness of the program. The program also includes three user accounts: Admin, Employee, and Customer, with each user account having access to different privileges. Any of the mistakes that are made while implementing these user accounts may lead to unacceptable circumstances, in which a Customer account can invade privacy of other customers, creating ethical problems, or access to employee accounts, changing the internal structure of the program in an irreversible way.

Manual Testing

An alternative to all different testing approaches could be using manual testing as a primary method of testing. Considering the low number of Use Cases (with 14) and the results that could be obtained (a boolean value of TRUE or FALSE), manual testing could be a probable and clearly shorter method of testing. The use cases could be tested by the developers and some of them even by the non-developers. The fact that the testing process does not require any coding is also a plus.

Project Management and Plan of Work

Merging the Contributions from Individual Team Members

The team has tackled portions of the reports in sections individually while sharing their progress on a shared google drive. The final product has been merged through attaching all of the portions together and smoothing out any inconsistencies/formatting differences. The team then goes through the report in a group call and ensures that everything is correct before it is finalized. No issues were encountered, everyone had their portions done in time so that no rushing was needed by anyone last minute.

Project Coordination and Progress Report

No full use cases have been implemented as a significant portion of the work lies in a completely functional web app. The majority of the use cases will be completed upon the core functionality completion (approx. 03/21/2021). The rest of the use cases will be tackled between demo 1 and demo 2.

The hosting, database connection, and react setup have all been completed. The navigation/header, home page, and login/sign-up components have been coded as well. Configuring API endpoints and creating database models are underway on the back-end. The front-end ongoing work consists of creating the admin interface, adding functionality to the web application with react-router, and creating inventory cards and results components.

Plan of Work:

Group Meeting: 03/08/2021 (Divide up final work for Report 2)

Finish Individual Work on Report 2: 03/11/2021

Wireframe Functionality Complete: 03/14/2021

Core Functionality Complete: Nav, Template Item Page, Display 3 Cars, etc. 03/21/2021

Bug Fixes/Styling Complete: 03/28/2021

Record Demo: 03/28/2021

First Demo Presentation: 04/05/2021

Report 3 - Part 1: 04/12/2021

Report 3 - Part 2: 04/19/2021

Second Demo Presentation: 05/03/2021

Breakdown of Responsibilities

Generally speaking the responsibilities are broken up into two major sections. The front-end of the application is being tackled by Brady, Fatih, and Taylor. The back-end is being put together by Seth and Ryan. Further breaking up each section into individual tasks is being done by issues within github. For example, one of the members of the front-end section takes care of coding the login/sign-up page and puts in a merge request when their section is complete. There exists an issue for every feature of the site listed so far on the reports and are taken care of in order of importance.

Integration is a team effort, and will be done by any member available to help with a merge request. Thus far, Seth has set up the base layer of the project and has succeeded in ensuring that the team can work off of a solid skeleton of working parts.

Integration testing is done by creating a branch off of the dev branch and making code changes within the created branch. When unit testing by the individual is done, a merge request is submitted. Another member of the team then reviews the code and ensures accuracy before the merge is completed into the dev branch. The master branch is being left as only perfected code, in case any merge causes issues within the dev branch itself. Bugs are listed within github and taken care of accordingly.