Process MeNtOR 3.o

Uni-SEP

Content adapted from the deliverables prepared by Dr. Kostas Kontogiannis

EVoltMarket

Design Document

Version:	Version 6
Print Date:	Nov 19, 2023
Release Date:	Nov 19, 2023
Release State:	Completed
Approval State:	Team Approved
Approved by:	Team
Prepared by:	Team
Reviewed by:	Nolan Lobo, Manasvi Jain

Document Change Control

Version	Date	Authors	Summary of Changes
v1	Oct 1, 2023	Manasvi Jain, Celina Su, Sara Soleymani	Initial readings and additions from first meeting
v2	Oct 6, 2023	Everyone	UML added
v3	Oct 11, 2023	Everyone	Added all diagrams required and edited them as required for final submission
v4	Oct 19, 2023	Everyone	Added Test cases as required
v5	Oct 20, 2023	Nolan Lobo, Manasvi Jain	Edited document for final submission.
v6	Nov 18, 2023	Manasvi Jain, Celina Su, Sara Soleymani, Nolan Lobo	Added deployment diagram, Added description of REST principles, Modified according to feedback received from deliverable 1

Document Sign-Off

Name (Position)	Signature	Date
Nolan Lobo	NOLAN	Nov 19, 2023
Celina Su	Celina	Nov 19, 2023
Sara Soleymani	Sara	Nov 19, 2023
Manasvi Jain	Manasvi	Nov 19, 2023

Contents

1 Introduction

1.1 Purpose

An online ecommerce platform for electric vehicles addresses several challenges and provides solutions to various problems in the electronic vehicle industry. An online ecommerce platform for electric vehicles can improve accessibility by providing shipping to locations where dealerships are not as common such as in remote places. Many dealerships also provide limited options but with our ecommerce platform we can provide multiple different electric cars with a whole myriad of options. We aim to solve the problem of long purchase times being a major deterrent to e-commerce sales through using a unique user sign-in method. When the user first accesses the website, they are automatically assigned an account to their cookies, and when they checkout the account details are saved in the website This should create a better user experience which promotes the sale of more environmentally friendly transport options.

The objective of the project is to provide an online e-commerce platform catering specifically to electric vehicle sales interactions. Specifically the aim is to create a complete user experience. The user can pick their preferred model through the search function, can customize the car to their needs, can compare their model to other available models and can also make smart financial decisions by calculating loan payment plans. The platform administrators can also run detailed reports on vehicle sales and application usage.

1.2 Overview

The goal of this project is to provide a specialized e-commerce platform to facilitate the sales of electric vehicles. The introduction section explains the purpose, overview as well as references used for the project report. The major design diagram includes the rationale behind the design decisions, including the choice of design patterns and architectural patterns. It also includes the description of the modularization criteria. The use case diagram action features a use case diagram that shows the use cases supported by the system. The next section, Sequence Diagrams, shows our sequence diagrams that correspond to 3 of our use cases: registering a customer, browsing a catalog and checking out. The next section, Architecture, focuses on breaking down the system into a package diagram, a component diagram, and a description of three modules used and their interactions. This section also includes a table form of the use cases shown in the diagram. The sixth section, Activities Plan, focuses on breaking down how the project will be developed. It includes a Gantt chart, as well as a meeting backlog. The last section is the Test Driven Development section, which covers 16 test cases and their expected outputs.

1.3 Resources - References

Paper 1: *E-commerce Architecture and System Design*

https://link.springer.com/chapter/10.1007/978-3-540-49645-8_8

This paper examines the history of E-commerce system design, as well as examines how they can be implemented today. These topics include basic principles, design methods, data flow structure, and system architecture principles which should be followed when designing an E-commerce system.

Paper 2: *CONCEPT OF E-COMMERCE: SYSTEMS ANALYSIS AND DESIGN FOR ONLINE-STORES*

https://www.researchgate.net/publication/335159397_CONCEPT_OF_E-COMMERCE_SYSTEMS_ANALYSIS_AND_DESIGN_FOR_ONLINE-STORES

This paper focuses on how security, profiling and control should be implemented in an e-commerce system. It also suggests that processes such as design of the project, payment systems, content management, and search management should be a coordinated effort. It also focuses on core concepts such as search management, catalog management, event notifications, and collaboration and testing.

Website 1: Why Use MongoDB: What It Is and What Are the Benefits

https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mongodb

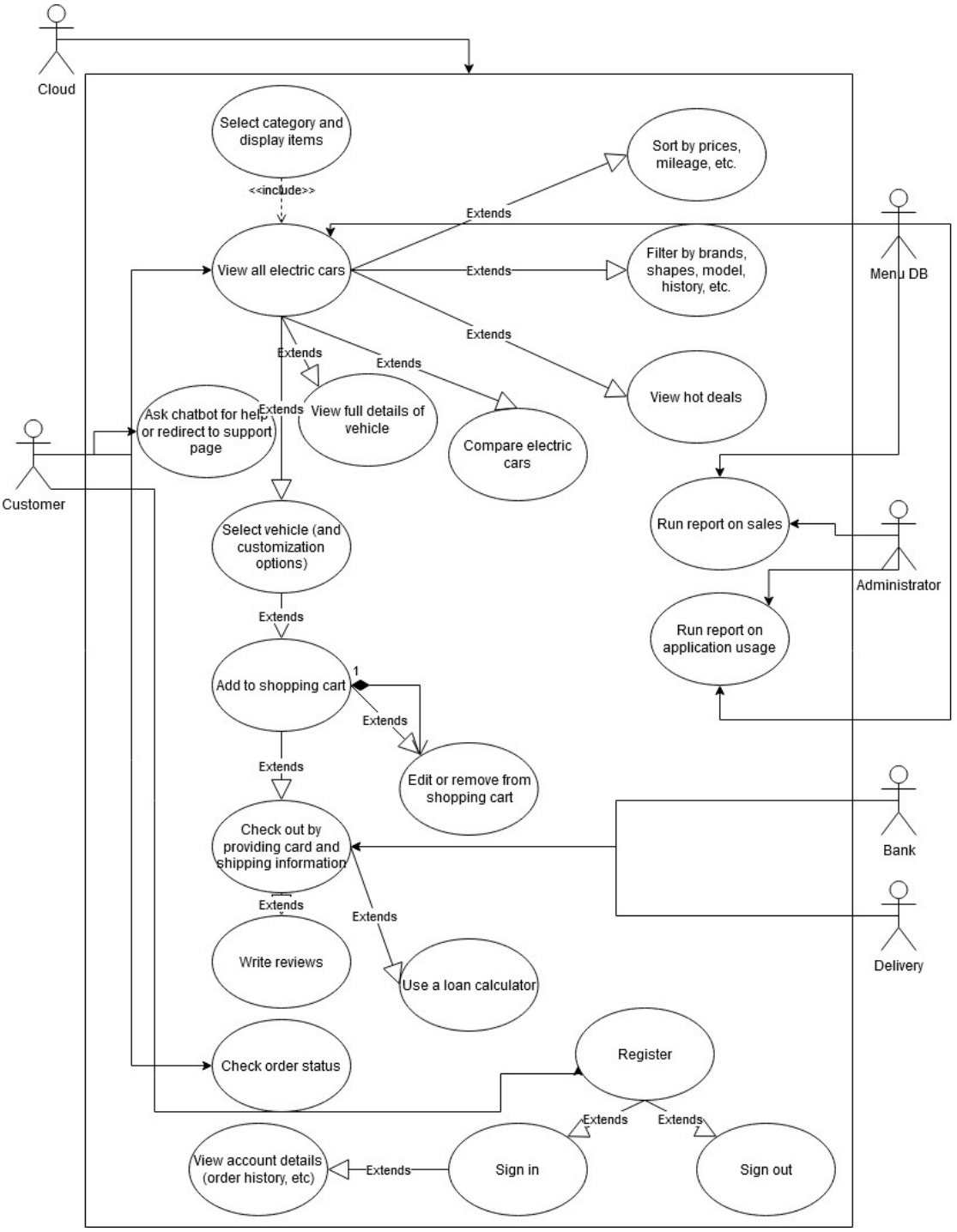
This website focuses on what MongoDB is, what it is and what are the benefits. It helped us decide on what database to use in our backend portion of the project.

2 Major Design Decisions

While the group did agree on most of the technologies to use in the project, there was rich debate regarding which architectural pattern to use. Specifically, there was discussion on whether to use a hierarchical We decided to use the Model-View-Control architectural style because it increases performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing to the user. We decided to split the presentation layer into 3 different components: Catalog view, the Customer Credentials view, and the cart view. In the business layer we decided to modularize the code by combining the related functions of the system together. In the data tier, we decided to add connections to the database so that we can perform operations on it such as add, delete, update, and read.

3 Use case Diagram

Below is the UML diagram created to show all the use cases covered in the system. It is intended to be a comprehensive diagram which showcases all the intended use cases

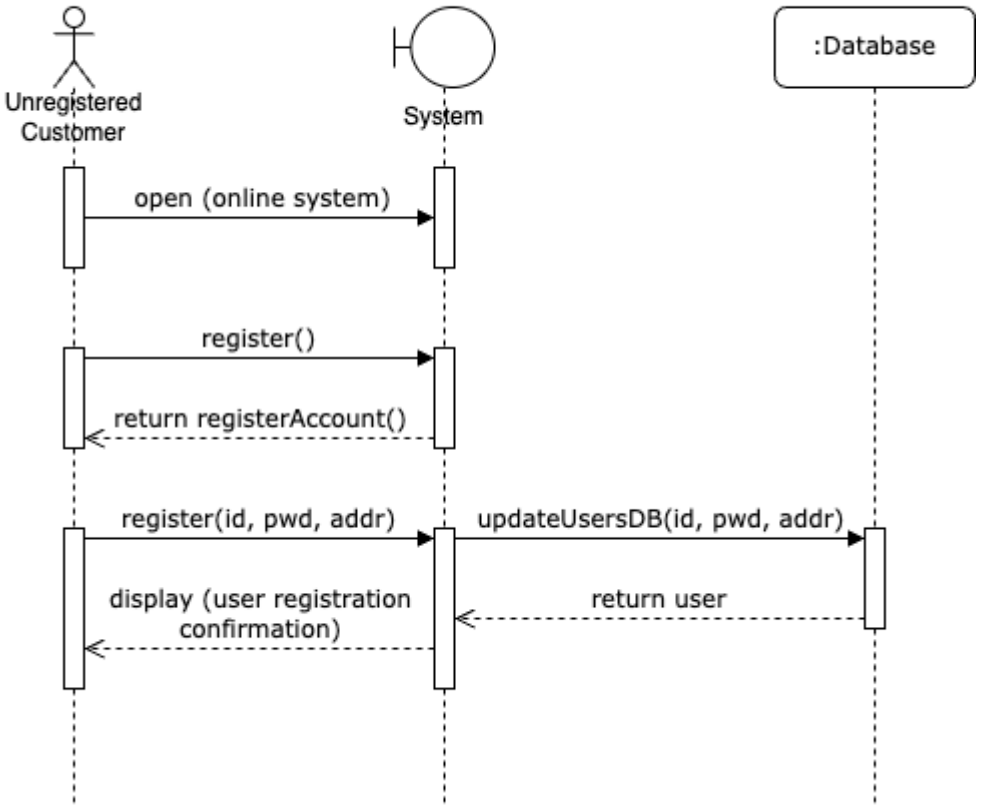


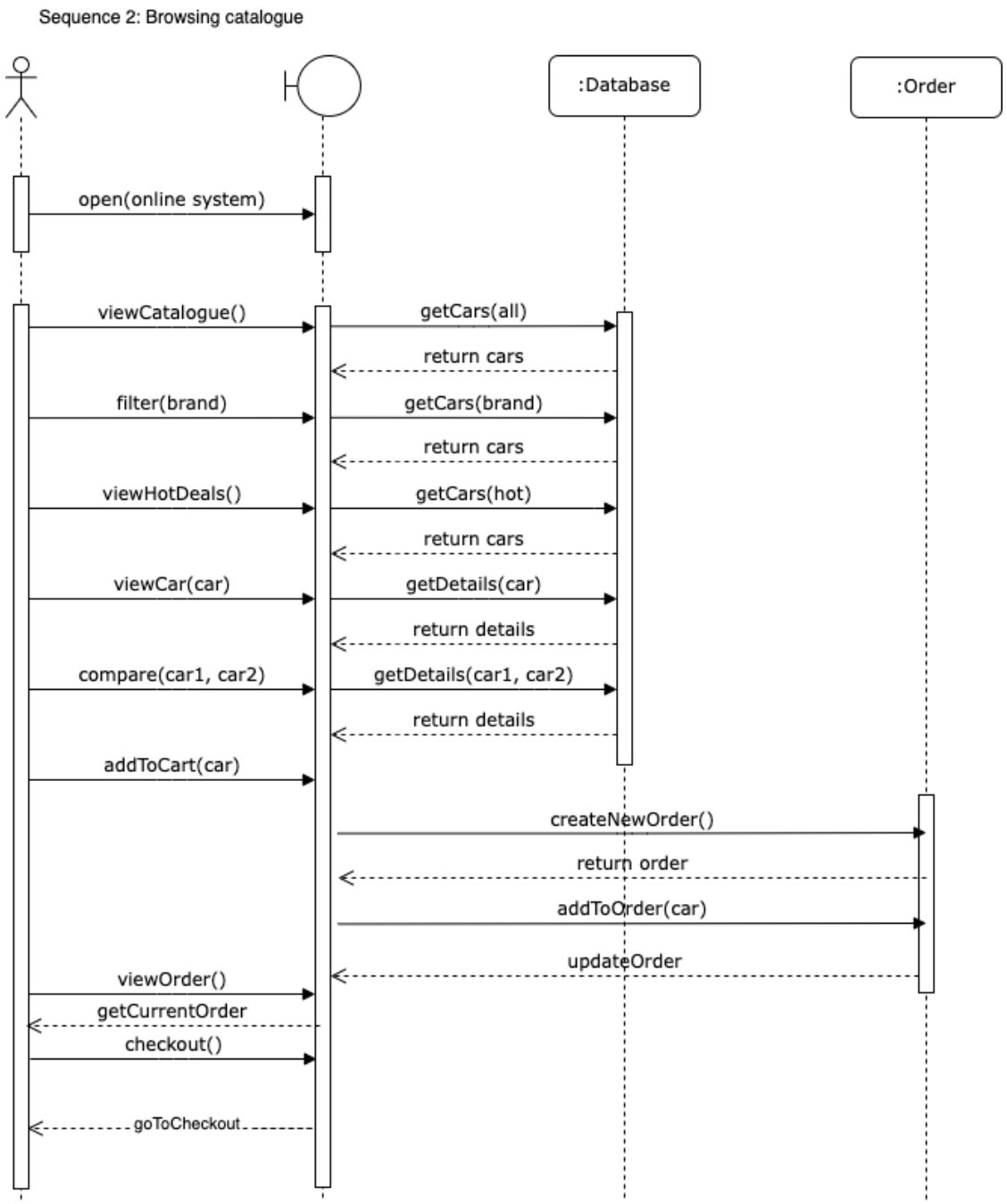
4 Sequence Diagrams

This section covers 3 sequence diagrams which depict use cases in the diagram. The 3 use cases covered are: registering a user, browsing the catalog and canceling an order. The first sequence is registering a customer. Here, an unregistered user would interact with the system to register an account. The system takes the user's request to create an account, interacts with the

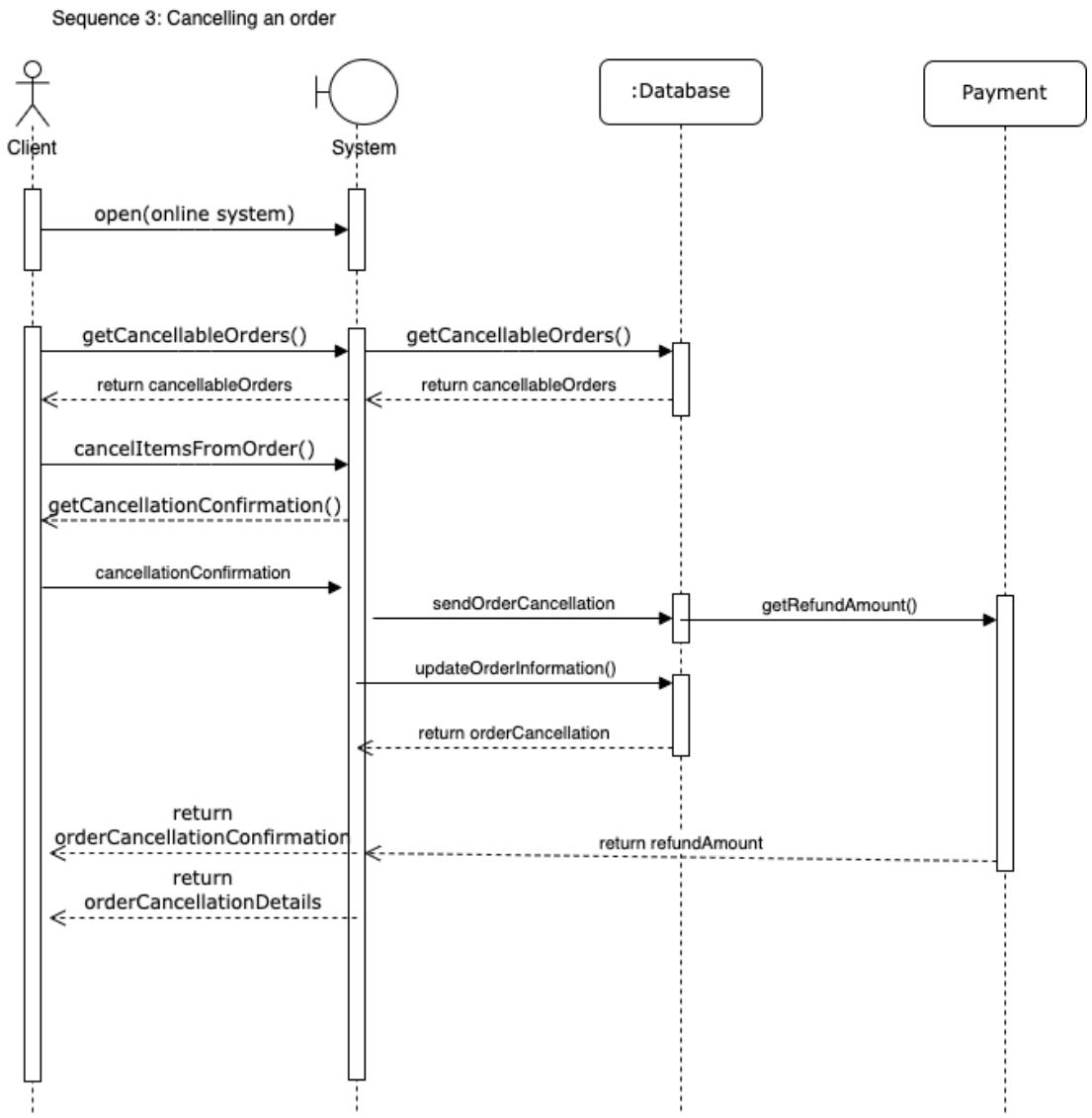
database to add a user to the database and returns the final account information to the user.

Sequence 1: Registering a customer





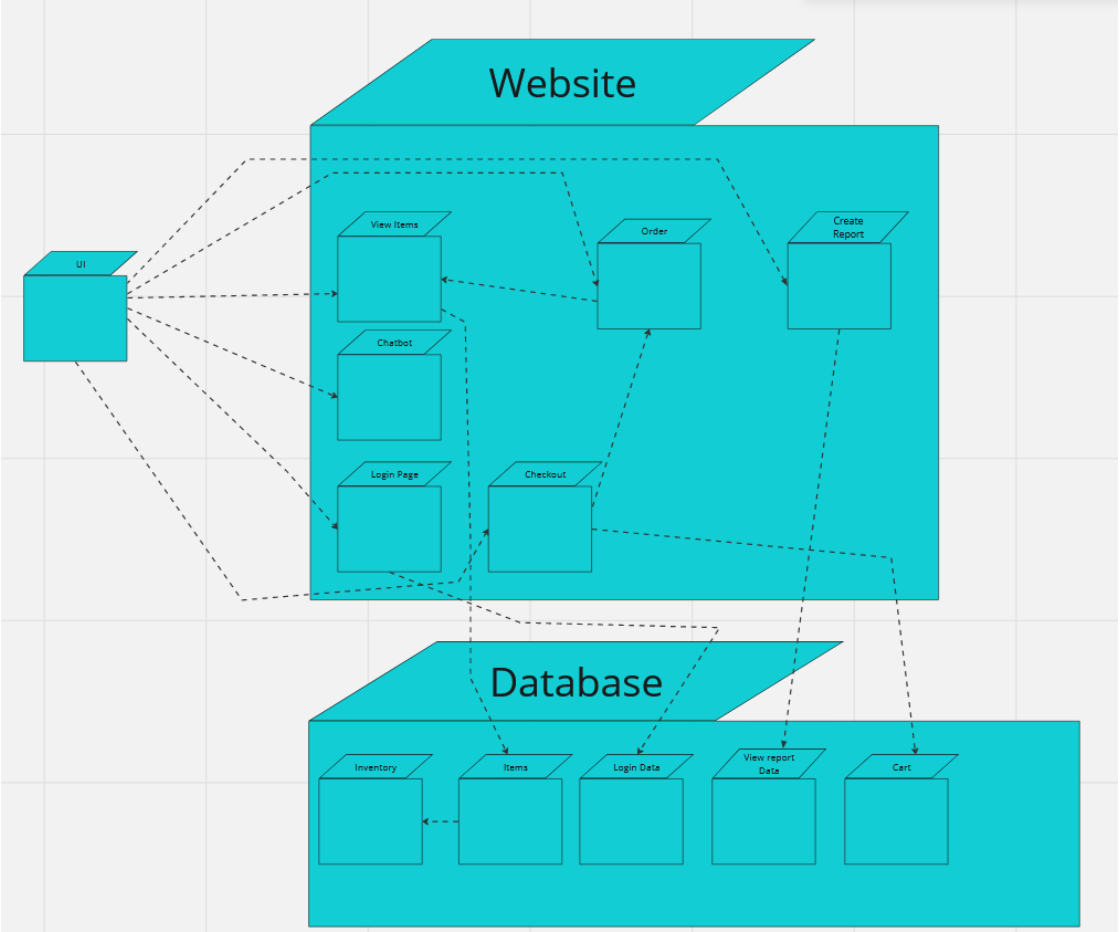
Above is the second sequence diagram. Here, the user browses the catalog provided to find their ideal car and to create an order. Whenever the user interacts with the system to find their preferred items in the catalog, the system will make a call to the database in order to pull the required data to show the user. Finally, when creating an order the user will create a new order and proceed to checkout.



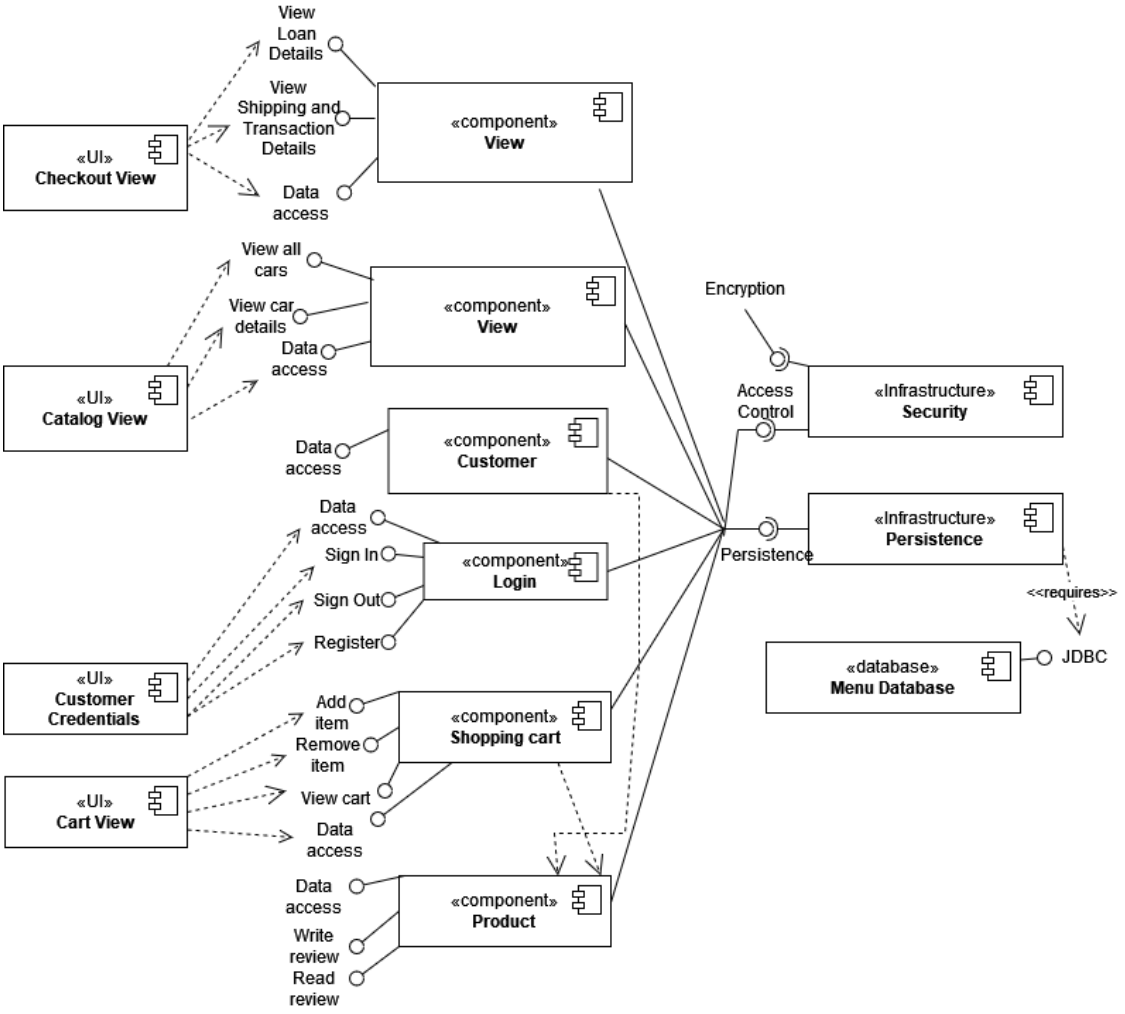
The above diagram shows the process as a user cancels their order. They get their cancellable orders, then after the system gets the confirmation from the user which allows them to delete the order, they delete the order from the database and contact the payment processor to get the final refund amount for the canceled order. This is then returned back to the system to display to the user.

5 Architecture

The figure below depicts the package diagram for the subsystems of the system. The UI represents what users as well as administrators see, the website is the formal logical representation of the system, and the database is where all the data is stored and updated as required by the website.



The component diagram below provides a decomposition of the system as a collection of the interacting modules. Below it is the description of each of the components.



Three-tier architecture – Presentation, Business (MVC), Data

Presentation:

- Divide into three main parts: catalog view (shopping), customer credentials (sign in, register, sign out), cart view (add item, remove item, view cart)
- These 4 UI's are shown to the user at different times. When the user views the catalog and customizes their car, they see the "Catalog View". When the user is doing credentials related activities, they see the "Customer Credentials" view. When the user is doing shopping cart related activities, they see the "Shopping Cart" view. When the user is at checkout, they see the "Checkout" view

Business:

- The components cover the business control sections.
- The view is updated according to how the logic is updated from the component section.
    - The customer view component changes which cars are shown as well as what car details are shown according to what filters the user applies.

- The customer component accesses data from the system for logical system use.
- The login component updates user credentials for the current user according to whether the user signs in, or signs out. Also, a new user is added to the system whenever a user visits the website as well as whenever a new user registers their data.
- The Cart View is updated as the user adds or removes items from the shopping cart component.
- The product component has writing and reading reviews, and can be accessed by the shopping cart component.

Data :

- The Data is persisted by the persistence infrastructure.
- The Persistence infrastructure connects via Mongo to the main database called “Menu Database”.
- The Security infrastructure uses encryption and access control to secure the system.

Use cases that registered customers can execute	**Use cases that administrators can execute**
List electric vehicles available in the catalog	Run Reports on vehicle sales
Sort vehicles by prices (sort electric vehicles based on ascending prices or based on the descending prices) or by mileage (i.e., low to high and high to low)	Run reports on application usage
Filter electric vehicles by brands, by shapes, by model year, by vehicle history (i.e., with reported accidents/damages, and without reported accidents/damages)	
View full details of each electric vehicle (including the vehicle history report if applicable) available in the catalog	
View hot deals	
Use a loan calculator (*)	
Be able to compare vehicles	
Select a vehicle customization options	
Add vehicles to the shopping cart	
Edit or remove vehicles from the shopping cart	
“Check out” by providing credit card information and	

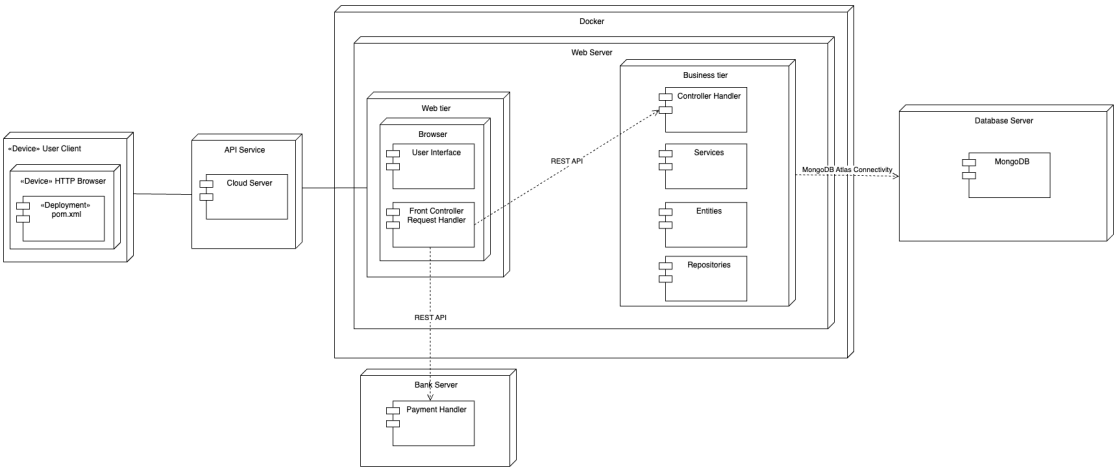
shipping information to purchase the vehicles in the shopping cart	
Write reviews on vehicles and rate vehicles using five stars	
Register	
Sign in	
Sign out	
Ask questions to a chatbot able to answer basic questions and help customers navigate to a specific vehicle or support page	
Distinguished use case: Cancel order	

Modules			
Module Name	**Description**	**Exposed Interface Names**	**Interface Description**
M1	Catalog view	M1:I1 M1:I2	M1:I1 “viewAllCars()” M1:I2 “viewCarDetails()”
M2	Customer Credentials	M2:I3 M2:I4 M2:I5	M2:I3 “signIn()” M2:I4 “signOut()” M2:I5 “register()”

Interfaces		
Interface Name	**Operations**	**Operation Descriptions**
M1:I1	<return List<Car>> I1:Op1() used by M2, M4	returns a list of cars for view in the catalog view
M1:I2	M1:I2:Op2() used by M9 <returns List <CarInfromation> I1:Op2()	returns a list of car informations for use in the catalog view
M2: I3	<return Customer > I3:Op() used by M4, M8, M9	returns the customer after the sign in has been completed for use in the other modules
M2: I4	<update database > I3:Op() used by M5	After signout, the user is automatically registered by M5
M2: I5	<return new Customer > I3:Op() used by M4, M8, M9	returns the customer after either registration process completed by user, or user without any login credentials stored in system visits website for first time.

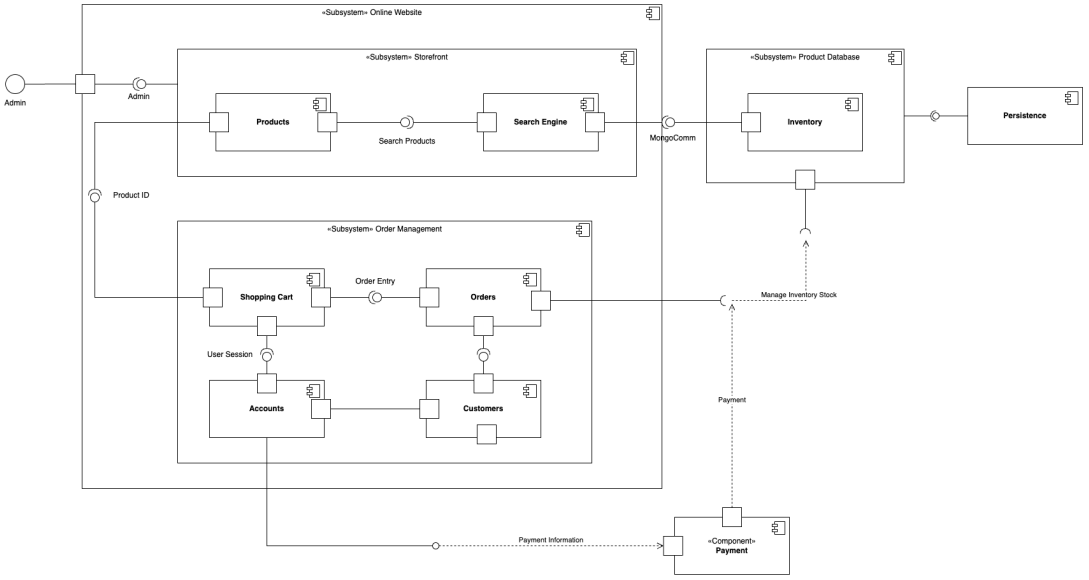
Here, we have two modules: M1, which represents the Catalog view, and M2, which represents the customer credentials view. Within the Catalog view, we find two interfaces: I1, where all cars can be viewed, and I2, where we can see the details of the car. In the customer credentials view, we have three interfaces: I3 for signing in, I4 for signing out, and I5 for registration.

Deployment Diagram



The above component diagram describes the architecture of our application. The User Client is how the User interacts with the website. The User will use their HTTP browser to connect to the application. Docker is used to containerize the Web Service. The user will use the API service to handle API requests and responses. There are then two parts of the web server: the Web Tier and the Business Tier. The Web tier encapsulates the User interface and the frontend request handler. This ensures that the user does not have control over the backend since the frontend request handler can only request data. The business tier encapsulates the logic and model component of the application. The controller handler includes business logic as well as processes the web tiers application requests. The Services are used to execute specific rules and logics. Entities are models which are used in the application. Repositories interact with the database and are responsible for storing and accessing the data. The bank server component has a PaymentHandler which deals with payment processing. The Database Server is hosted on MongoDB Atlas and is connected through MongoDB connectivity.

Component Diagram



The above component diagram depicts the organization of our ecommerce system. The storefront is a subsystem consisting of what the user clients are seeing and interacting with. This includes the search engine which the clients can use to search, filter, and browse through the products catalogue, and is also connected to the product database. The search engine connects to the product database, returning the user's requests for products through Mongo commands. The products are added to the shopping cart through the product IDs which inturn can be made into an order entry if the user is to purchase the items.If they are not purchased, they may remain in the user's shopping cart through the user session persistence. These are both stored within the customer's database. Once the order is placed, the product's database is updated with the new inventory count and the payment is completed through the payment gateway connected to the customer's account.

Design patterns:

For the design patterns we decided to use Observer, Singleton and Factory.

Observer pattern: Since Observer pattern works perfectly for defining one-to-many dependencies between objects, and also notifies and updates all of its dependents automatically, we thought it would be a good idea for us to use this pattern for our e-commerce systems for handling the situations such as changes in the inventory, changes in the cart, order status changes or inventory updates. Also, using the Observer pattern helps us maintain a clear separation of concerns which leads to reducing the security risks and scalability.

Singleton pattern: We thought Singleton pattern would fit well in our E-commerce system for things like managing our shared resources, memory caches or global configurations since Singleton pattern allows us to have only one instance for a

class and provides a global point of access to that instance.

Factory pattern: The database adds multiple objects created through the entity classes in a Factory pattern. This is extremely useful for scaling and reliability since the classes are able to obtain commonly required attributes easily. This also helps abstract object creation as required.

JEE patterns

Presentation pattern: For the presentation tier, we wanted to use the Front Controller pattern. This is because the front controller allows for a single, centralized point of request handling making it easier to manage all user requests. All the incoming requests will be handled by a single handler acting as the initial contact point, and then passed to the corresponding proper handlers where the requests will actually be fulfilled. This mechanism allows us to have a clearer view of the flow and navigation through our system. Having the front controller is also beneficial for managing and reducing code as there is less duplication across the different responses for handling each request. This makes the behavior for each request handler more easily maintainable as the code is not scattered, with the logic being in a centralized location, allowing for changes to be made in fewer places. However this was not able to be completely implemented in the final product. This is because we were still testing the code and thus were testing with multiple controllers. Due to this, the submission includes multiple controllers instead.

Data tier: For the data tier, we decided to use the Data access object pattern. The Data Access Object (DAO) separates the data-accessing to the database, such as obtaining and storing data, from the clients by abstracting the underlying data access implementation into a separate layer. By doing this, the data-persistence logic is hidden and the client is unaware about how the operations regarding the database are implemented. This allows for our system to be loosely coupled in regards to the business and data components, as well as centralizing all the data access points, making any changes we may need to perform later on easier to manage and maintain. Another big advantage of this was that the DAO pattern makes it easier to transfer to different database implementations. If we were to have problems with our current database, it would be a simple fix to change to a different database as we would only need to make changes to the DAO layer instead of rewriting our entire application. To implement this, we used multiple repositories in Spring Boot, which acted as specialized DAO for our backend components.

Business tier: For the business tier we decided to use the service locator pattern in our E-commerce system since it lets us centralize the process of obtaining a service with a centralized registry. We thought this would be appropriate specially to manage and locate different services like payment processing, shipping or

inventory management. This would give our program the flexibility to manage the business tier better, make it easier to maintain and update the system and improve the testability and scalability. We use our controllers to access certain service controllers and use multiple services to aid in finding and accessing data from the database.

Architectural patterns

One architectural pattern we decided to implement was REST. One reason why is due to the fact that the client and server implementations can be modular and separate. This allows us to freely make changes to either side without any concerns of how it may affect the operation of the opposite side. Both sides are thus not reliant on each other therefore allowing for good performance for availability and scalability. This made it easier for us to implement our system as we divided our work into either client and server sides and we would be able to freely develop our work independently of each other.

Along with the REST architectural style and the front controller design pattern, we also have the possibility of implementing the API gateway as the one single access point to the rest of the application.

We also decided to implement microservices to further break down our application into smaller components. This helps to further organize our system as each component would be independent from the others, and therefore can be updated or expanded upon without affecting the functionality of the rest. As a result of this, it also helps to ensure loose coupling between the components. Breaking down our services into smaller components is also beneficial as each component has its own focus on solving a particular problem, making testing and debugging easier for us.

For the structure of our system, we decided to go with the three tier architecture (presentation, business and data) and to incorporate the Model-View-Controller pattern within it as well. This is because this helps provide a separation between the different components within the tiers allowing for easier organization of the code and therefore maintenance as well. The similar actions are grouped together all resulting in separating into different components, resulting in coupling.

Description of the Server Side Implementation

For the server-side implementation, we use Spring Boot which acts as the controller in MVC and MongoDB Atlas for our database. We thought using these two would be a solid choice for building robust and scalable microservices following REST principles. Furthermore, using MongoDB would allow for easy scalability when deploying the application on the cloud.

MongoDB Atlas allows for easy integration with Docker, which is the technology that we would be using to deploy our application on the cloud which would ease the development and maintenance process. Furthermore, MongoDB has an excellent query language which allows for powerful querying such as searching text which is beneficial for us since it can allow for fast querying. MongoDB also has encryption while the data is being transported from the front-end to the back-end which would further improve the security of the application. Also, since MongoDB is a NoSQL database, it gives us flexibility in our data modeling which leads to scalability. We decided to use MongoDB since it is cloud-based and can provide us with high availability, scalability and security.

Spring boot acts as the controller in the MVC architecture. It facilitates rapid development, provides a comprehensive set of tools and simplifies the configuration. Spring Boot inherently supports the RESTful APIs which leads to Spring Boot to be our choice to implement the REST principles. Spring Boot handles the complex configuration tasks when creating REST services while also simplifying common tasks. For example, Spring provides us with annotation services which give us the ability to help define Restful endpoints which simplifies the process of defining API endpoints significantly. Spring Boot also gives us the ability to support statelessness in the RESTful APIs. Furthermore, using HTTP methods such as GET, POST, PUT and DELETE properly, leads us to properly following the REST principles.

MVC architecture was used to implement our e-commerce system and it represents the data model and is mapped to MongoDB. Its controllers handle the incoming HTTP requests, interacts with services and returns the appropriate responses. The repositories we used provide us with an interface with the MongoDB database, leading to a high-level abstraction for data access.

In our project we separated our application into entity classes, their controllers, and their services which follow REST principles. Entity classes represent the structure of the model i.e. a resource that our application will be using. Controller classes handle incoming HTTP requests and the data that comes with it as well as defining operations that will be done on the data. They are also responsible for invoking the operations associated with the incoming HTTP request. The service layer acts as the middleman between the controllers and data access layer. For example, in our application we developed the User entity class. The user entity class defines the structure of the user data.

Microservices

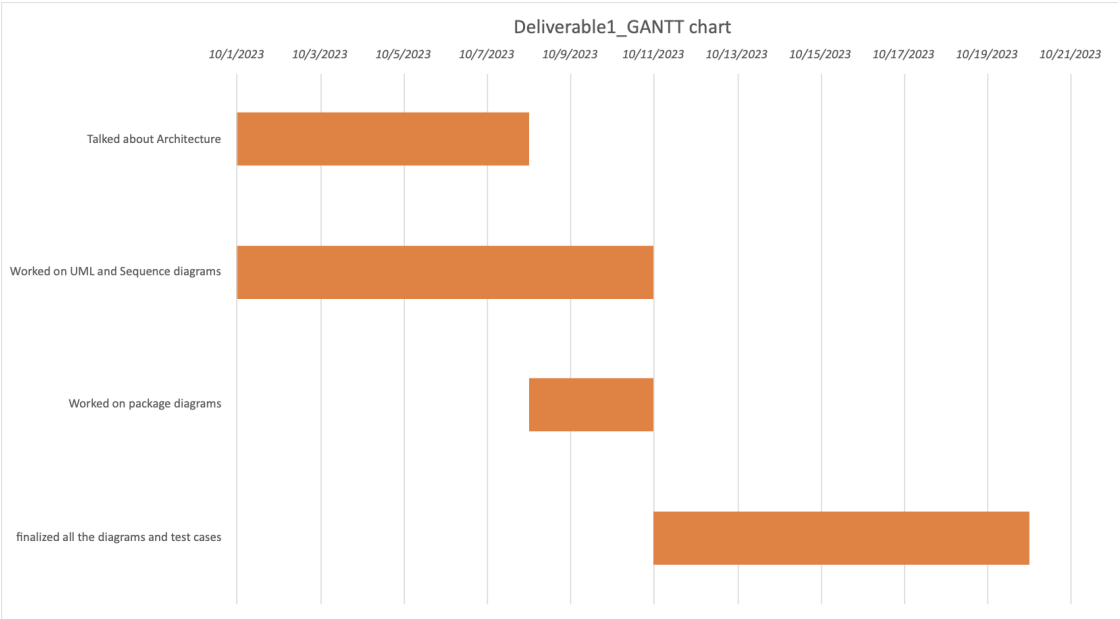
There are 3 microservices developed: CustomizationService, ProductService, and UserService. CustomizationService had multiple bugs and the code was removed, but it was meant to query availability for customization options, and add customized cars to cart, but was not able to make final submission. ProductService is used for adding products to the database and finding all the products in the database. UserService is used to provide specialized services related to user credential activities. It creates, and finds users.

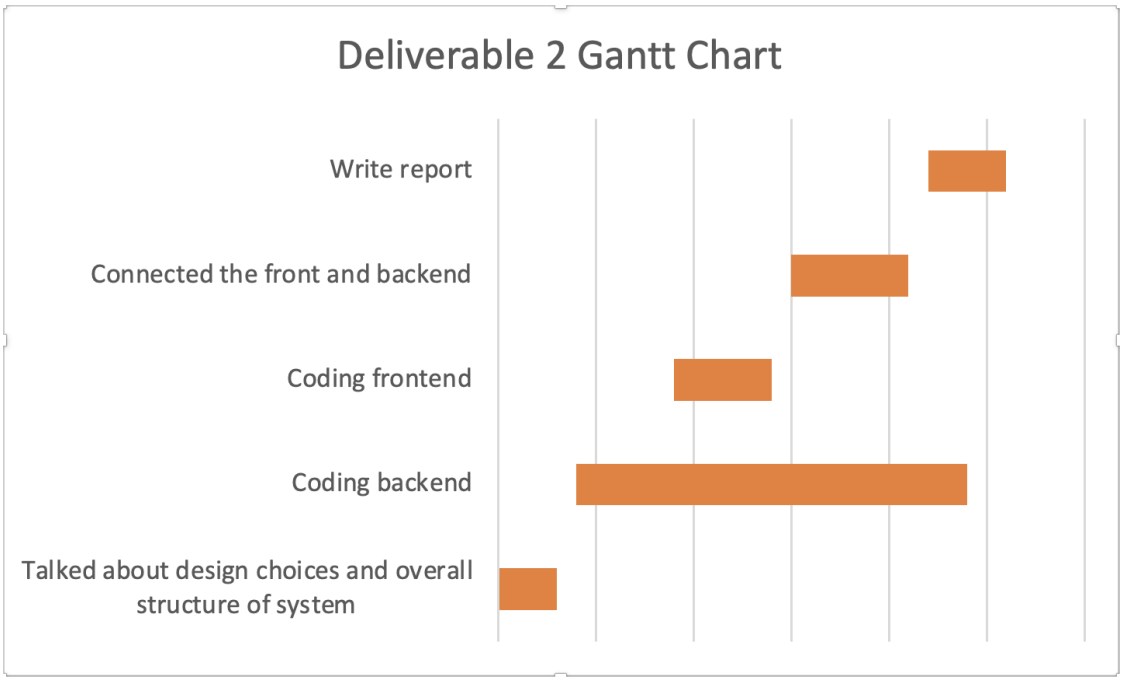
Chatbot Implementation Technologies

The technologies we are planning to use to implement the chatbot are: Amazon Lex. Amazon Lex would be able to integrate nicely with AWS which is a technology that we would be using to deploy our application on the cloud. A chatbot framework such as Amazon Lex can help build the conversational flow and handle the user interactions. While we did discuss training our own chatbot due to new technologies allowing for accessible and effective training options, we decided on using Amazon Lex instead since it would be much more accessible to work with and it integrated very well with AWS technologies.

6 Activities Plan

6.1 Project Backlog and Sprint Backlog





6.2 Group Meeting Logs

Present Group Members	Meeting Date	Meeting Time	Issues Discussed / Resolved
Celina Su, Manasvi Jain, Sara Soleymani	Oct 1, 2023	10:30 AM - 4:00 PM	Discussed the architecture of our E-commerce system, talked about the use cases for registered customers and administrators, discussed ways to draw our UMLs
Everyone	Oct 8, 2023	4:00 PM-7:00 PM	Agreed on uml, talked about sequence diagrams, split tasks
Everyone	Oct 11, 2023	8:00 PM - 10:30 PM	Talked about the diagrams we made and made changes in them
Everyone	Oct 20, 2023	90 minutes total	Finalized report detail
All members	Oct 24 2023	3PM-7PM	Reviewed the project outline and discussed issues with deliverable 1. Split up tasks for deliverable 2 and started coding.
All members	Nov 2 2023	7PM-8:30PM	Discussed the the design choices for deliverable 2 and updates for our code for the backend and started front end

Celina Su, Manasvi Jain, Sara Soleymani	Nov 12 2023	11:30AM-4PM	Discussed updates about our code and problems we encountered and what to do to resolve it (ran into an issue with springJPA. Continued to work on our code for back and front end.
All members	Nov 16 2023	5:30PM-7PM	Decided to switch to MongoDB. Continued to work on our code and the report.

7 Test Driven Development

Test cases will be provided in the form of a table as follows:

Test ID	1
Category	Evaluation of User credentials stored on database
Requirements Coverage	UC14- Successful-User-Login
Initial Condition	The user has signed up previously
Procedure	1. The user selects login 2. The user provides a user name 3. The user provides a password 4. The user logs-in into the system and is presented with the main UI window
Expected Outcome	The expected outcome of the test case (*e.g. the login form closes, and the user is presented with the main UI window*)
Notes	The user should provide only alphanumeric usernames

Test ID	2
Category	Evaluation of the checkout feature Which part of the system is tested (*e.g. evaluation of user credentials stored on file or DB*)
Requirements Coverage	UC11- Successful-Checkout
Initial Condition	The system has been initiated and runs Initial conditions required for the test case to run (*e.g. the system has been initiated and runs*)
Procedure	1. The user selects checkout 2. The checkout UI shows up with all the products in their cart 3. The user enters their credit card information 4. The user enters their shipping information

	5. The checkout is successful after verification of credit card information. 6. After checking out the user is presented back to the main UI.
Expected Outcome	The checkout page closes and the user is presented with the main UI window. The expected outcome of the test case (*e.g. the login form closes, and the user is presented with the main UI window*)
Notes	

Test ID	3
Category	Evaluation of the register feature Which part of the system is tested (*e.g. evaluation of user credentials stored on file or DB*)
Requirements Coverage	UC13- Successful-Register
Initial Condition	The system has been initiated and runs. The user doesn't exist in the database Initial conditions required for the test case to run (*e.g. the system has been initiated and runs*)
Procedure	1. The user selects the login page 2. The user enters their information (email address, first name, last name) 3. After registering the user information is added to the database 4. The user is taken back to the sign in page where they sign in and taken to the main UI.
Expected Outcome	The register page closes and the user is presented with the main UI window. The expected outcome of the test case (*e.g. the login form closes, and the user is presented with the main UI window*)

Notes	Any other notes you may want to add for this test case, which are also reflected in the requirements specification (*e.g. the user should provide only alphanumeric user names and passwords without any special characters*)

Test ID	4
Category	Evaluation of the loan calculator feature Which part of the system is tested (*e.g. evaluation of user credentials stored on file or DB*)
Requirements Coverage	UC6- Loan Calculator
Initial Condition	The system has been initiated and runs Initial conditions required for the test case to run (*e.g. the system has been initiated and runs*)
Procedure	1. The user selects the loan calculator feature 2. They are taken to the loan calculator page 3. They are allowed to enter the loan amount, loan term, interest rate, compounding period, and the payback period. 4. The final calculation is presented on the page
Expected Outcome	The final calculation is displayed on the page The expected outcome of the test case (*e.g. the login form closes, and the user is presented with the main UI window*)
Notes	Any other notes you may want to add for this test case, which are also reflected in the requirements specification (*e.g. the user should provide only alphanumeric user names and passwords without any special characters*)

Test ID	5
Category	Evaluation of the Filtering feature
Requirements Coverage	UC3- Successful-Filter
Initial Condition	The system has been initiated and runs Initial conditions required for the test case to run (*e.g. the system has been initiated and runs*)
Procedure	1. the user selects the filtering feature 2. the user chooses a feature for the filter

	3. the features are filtered
Expected Outcome	the information shown on the window is filtered based on the features that user chose
Notes	

Test ID	6
Category	Evaluation of Adding vehicles to the shopping cart
Requirements Coverage	UC10- Successful-Add
Initial Condition	The system has been initiated and runs Initial conditions required for the test case to run (*e.g. the system has been initiated and runs*)
Procedure	1. choose the vehicle 2. press the add to cart button 3. vehicle is added to cart
Expected Outcome	the vehicle is added to cart
Notes	

Test ID	7
Category	Evaluation of editing or removing vehicles from the shopping cart
Requirements Coverage	UC11- Successful-Edit
Initial Condition	The system has been initiated and runs Initial conditions required for the test case to run (*e.g. the system has been initiated and runs*)
Procedure	1. choses the vehicle from the cart 2. press the editing or removing button 3. the item is edited or removed from the cart
Expected Outcome	the vehicle is edited or removed from the cart
Notes	

Test ID	8
Category	Evaluation of Asking questions from a chatbot to get answer of basic questions and help customers navigate to a specific vehicle or support page
Requirements Coverage	UC17- Successful-Chat bot
Initial Condition	The system has been initiated and runs Initial conditions required for the test case to run (*e.g. the system has been initiated and runs*)
Procedure	1. the user presses on the chatbot button

	2. the chatbot window opens 3. the user asks a simple question and presses the button 4. the user gets the answer from the chatbot 5. the user closes the chatbot
Expected Outcome	the user successfully opens and closes the chat bot and gets the desired answers to simple questions that they ask
Notes	

Test ID	9
Category	Evaluation of the compare vehicles option
Requirements Coverage	UC7-Compare vehicles
Initial Condition	The system has been initiated and runs.
Procedure	1. The user selects the first car 2. The user selects the second car 3. The user presses “compare vehicles”
Expected Outcome	A list detailing the vehicle information of both cars in a chart format
Notes	

Test ID	10
Category	Evaluation of canceling orders
Requirements Coverage	UC17-Canceling an order
Initial Condition	The system has been initiated and runs. An order has been previously placed
Procedure	1. The user views their order history 2. The user selects the specific order 3. The order detail page is shown 4. The user submits a cancel order request
Expected Outcome	The user’s order is canceled and the user receives a refund
Notes	The request should be submitted before the deadline (i.e. within 30 days or before order is shipped) Cancellation is offered for both unregistered and registered customers

Test ID	11
Category	Evaluation of writing reviews
Requirements Coverage	UC12-Writing reviews and rating vehicles
Initial Condition	The system has been initiated and runs. User has been confirmed as a verified purchaser.
Procedure	1. The user goes to their order history 2. The user selects the specific order 3. The order detail page is shown 4. The user selects the specific product to review through the “Leave a review” option 5. The review input box is displayed 6. The user writes their review and rates the vehicle out of 5 7. The user submits their review
Expected Outcome	The review is displayed in the reviews on the product’s detail page along with the rating
Notes	Both unregistered and registered customers are allowed to leave a review as long as they are a verified buyer

Test ID	12
Category	Evaluation of administrator running a report
Requirements Coverage	UC18- Running a report
Initial Condition	The system has been initiated and runs
Procedure	1. The administrator logs in 2. The administrator selects “Run report” option 3. The administrator selects the specification option for their report 4. Report is displayed
Expected Outcome	Report for the desired information is displayed to administrator
Notes	

Test ID	13
Category	Evaluation of Vehicle listing in Catalog
Requirements Coverage	UC18- List Electric Vehicles in Catalog

Initial Condition	The system has been initiated and runs. User is currently in the catalog view.
Procedure	1. The user accesses the catalog view 2. A list of available cars is displayed
Expected Outcome	All available electric cars are displayed to the user, as per the “hot” by default or the latest filter according to the cookie.
Notes	According to filters, the list of available electric cars is shown

Test ID	14
Category	Evaluation of Vehicle Sorting in Catalog
Requirements Coverage	UC2- Sort Vehicles by prices or Mileage
Initial Condition	The system has been initiated and runs. User is currently in the catalog view.
Procedure	1. The user accesses the catalog view 2. A list of available cars is displayed 3. The user clicks on a button which holds the sorting options 4. The user selects their preferred filter with their options being: ascending prices, descending prices, low mileage, high mileage
Expected Outcome	The user should click on the button and have the correct options available, and by selecting the options change the sorting of the catalog as required by the filter.
Notes	The user can complete this action multiple times, with the sorting not carrying over between the sorting attempts. Also, a cookie should be created which saves the sorting preference

Test ID	15
Category	Evaluation of Full Detail View in Catalog
Requirements Coverage	UC4- View Full Details of Vehicles
Initial Condition	The system has been initiated and runs. User is currently in the catalog view.
Procedure	1. The user accesses the catalog view 2. A list of available cars is displayed 3. The user selects a car. 4. The car’s full details can be seen additionally with a comment box above the cars icon 5. The user can close the box if they prefer.

Expected Outcome	On clicking the car, a comment box will appear above the car's icon, with the full details including it's history report being presented to the user.
Notes	The box should be closeable by clicking on the "x" button on the top right.

Test ID	16
Category	Evaluation of View Hot Deals in Catalog
Requirements Coverage	UC5- View Hot Deals in Catalog
Initial Condition	The system has been initiated and runs. User is currently in the catalog view.
Procedure	1. The user accesses the catalog view 2. A list of available cars is displayed 3. The user clicks on a button which holds the sorting options 4. The user selects to see hot deals in Catalog
Expected Outcome	The user can see the catalog view of cars sorted according to hot deals available.
Notes	This sorting should also be affected by other sorting features such as sort by ascending price or least mileage. It also should be the default view for a new user.