CSCI 4448: OOAD
Project Design for CarCustomizer
Nathan Mukooba and Abiriham Bitew

CarCustomizer

Design Plan

Table of Contents

Project Summary

Project Title: CarCustomizer
Team Members:

1. Nathan Mukooba
2. Abriham Bitew

High-Level Overview:
The CarCustomizer project is a web application developed in Java that allows users to personalize cars by choosing from a wide range of customization options. The system aims to simplify and streamline the process of creating custom vehicles, and it can be used as a standalone application or seamlessly integrated into other Java applications.

Project Description:
CarCustomizer offers users the ability to customize various aspects of a car, including its color, interior, sound system, wheels, and more. Users can mix and match these options to create a vehicle that matches their preferences and style.

Key Features:

1. **Car Customization Options:** CarCustomizer provides an extensive selection of options for customizing cars, enabling users to create a vehicle tailored to their preferences.
2. **Language Choice:** The project is developed using the Java programming language, ensuring platform independence and a robust development environment.
3. **Design Patterns:** CarCustomizer incorporates several design patterns to ensure clean, modular, and efficient code. These include:

    - Observer Pattern: Real-time updates are facilitated as users customize their cars. The interface remains synchronized with the current state of the car through this pattern.
    - Strategy Pattern: Users can select different strategies for customizing the car, offering flexibility in feature selection and configuration.
    - Factory Pattern: Car creation is centralized through the Factory pattern, ensuring consistent and maintainable object creation.
    - Decorator Pattern: Enhancements and features can be added to cars using the Decorator pattern, allowing users to modify various aspects of their vehicles.

    By the end of the project, the CarCustomizer system will provide users with a user-friendly and versatile platform for customizing cars, allowing them to create unique and personalized vehicles tailored to their preferences. The system's design patterns and

Java development ensure a robust, modular, and efficient application for both standalone use and integration into other Java applications.

Project Requirments

Functional Capabilities:

1. **Car Customization Options**
    a. Responsibility: Provide users with a diverse selection of car customization options, including color, interior, sound system, wheels, and more.
2. **Real-time Updates**
    a. Responsibility: Implement the Observer pattern to ensure that the interface remains synchronized with the current state of the car as users customize it.
3. **Customization Strategies**
    a. Responsibility: Enable users to choose from different strategies for customizing their cars, allowing flexibility in feature selection and configuration through the Strategy pattern.
4. **Car Creation**
    a. Responsibility: Centralize car creation using the Factory pattern to ensure consistent and maintainable object creation.
5. **Enhancements and Features**
    a. Responsibility: Allow users to modify various aspects of their vehicles by adding enhancements and features through the Decorator pattern.

Non-Functional Characteristics:

1. **Language Choice**
    a. Responsibility: Develop the project using the Java programming language, ensuring platform independence and providing a robust development environment.
2. **Usability**
    a. Responsibility: Design an intuitive and user-friendly interface to enhance the user experience, making car customization straightforward and enjoyable.
3. **Performance Goals**
    a. Responsibility: Ensure efficient performance in terms of response times and system resource utilization to provide a smooth and responsive user experience.
4. **Integration Capability**
    a. Responsibility: Design the system with flexibility in mind, allowing for seamless integration into other Java applications as mentioned in the project overview.
5. **Platform Independence:**

    a. Responsibility: Ensure that the application is platform-independent, allowing users to access and use CarCustomizer on various devices and operating systems.
6. **Security**
    a. Responsibility: Implement security measures to protect user data and prevent unauthorized access to the system.
7. **Scalability**
    a. Responsibility: Design the system with scalability in mind, allowing for potential growth and increased user demands.

Constraints:

1. **Development Timeframe**
    a. Responsibility: Plan the project to be completed within five weeks, with 2-week sprints, and ensure that 50% of the project is achieved with workable code by the end of the third week, as specified in the project workflow.
2. **Team Collaboration**
    a. Responsibility: Collaborate effectively with the project team members, Nathan Mukooba and Abriham Bitew, to ensure smooth and coordinated development efforts.

Users and Tasks: Use Cases via Text

User Types:

1. End User: The general user of the CarCustomizer web application who wants to personalize a car.

Use Cases:

1. Customize Car
    - User: End User
    - Task: The user wants to customize a car by selecting various options such as color, interior, sound system, and wheels.
    - Variations:
        - The user can choose a specific car model or start with a basic template.
        - The user can change customizations at any point during the process.
        - The user can save customizations for future reference.
        - The user can view a real-time preview of the customized car.

2. View Car Catalog
    - User: End User
    - Task: The user wants to view available car models and their base specifications.

- Variations:
    - The user can filter or search for specific car models.
    - The user can access detailed information about each car model.

3. Share Customization
    - User: End User
    - Task: The user wants to share their customized car with others.
    - Variations:
        - The user can share a link to the customized car's configuration.
        - The user can post the customization on social media.

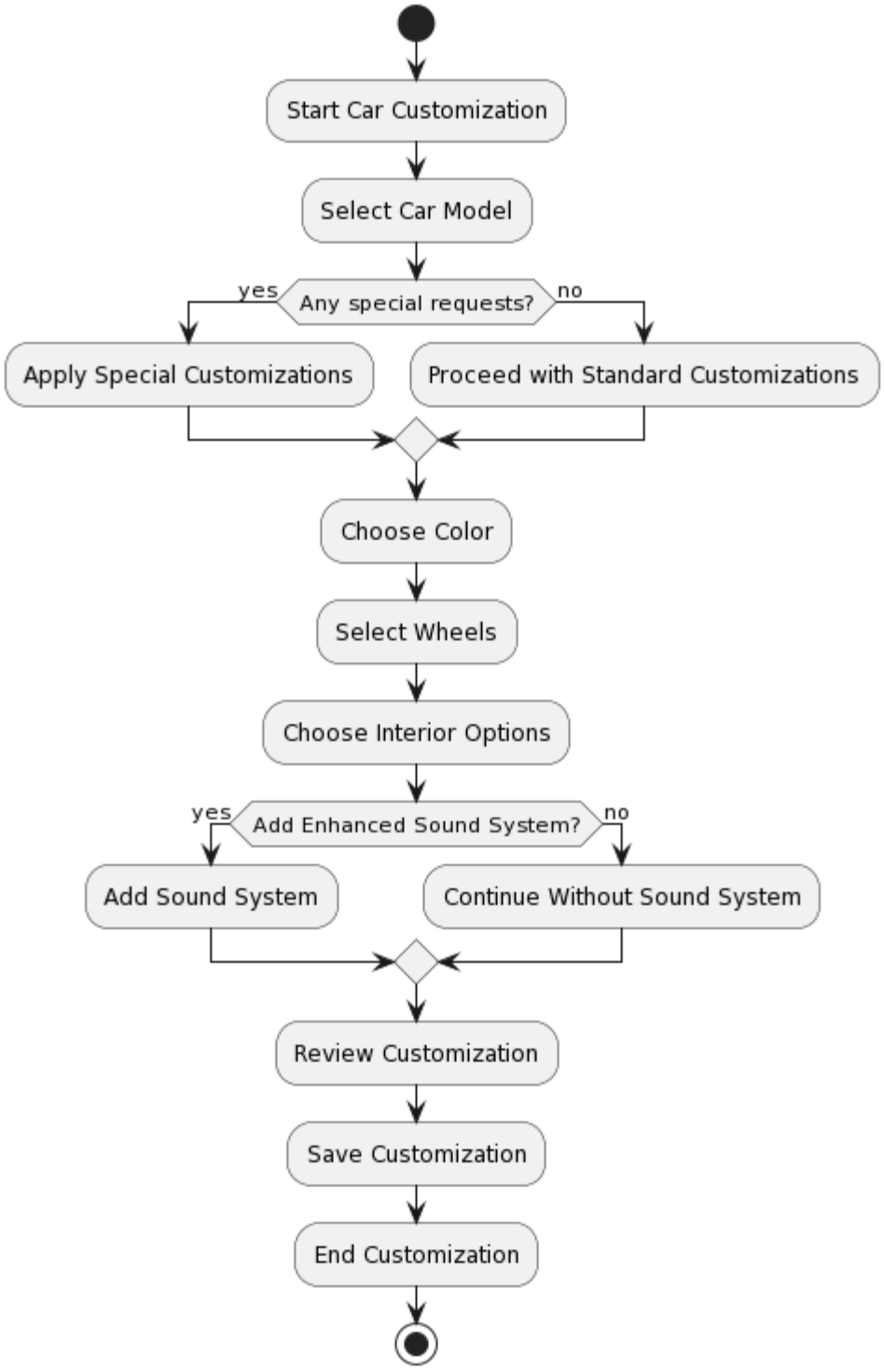
4. Manage Customizations
    - User: End User
    - Task: The user wants to manage and organize their saved customizations.
    - Variations:
        - The user can delete or edit saved customizations.
        - The user can organize customizations into folders or categories.

5. Administer Car Models
    - User: Admin (if applicable/optional)
    - Task: Administer car models, including adding new models, updating specifications, or removing models.
    - Variations:
        - The admin can upload images, descriptions, and pricing information for each car model.

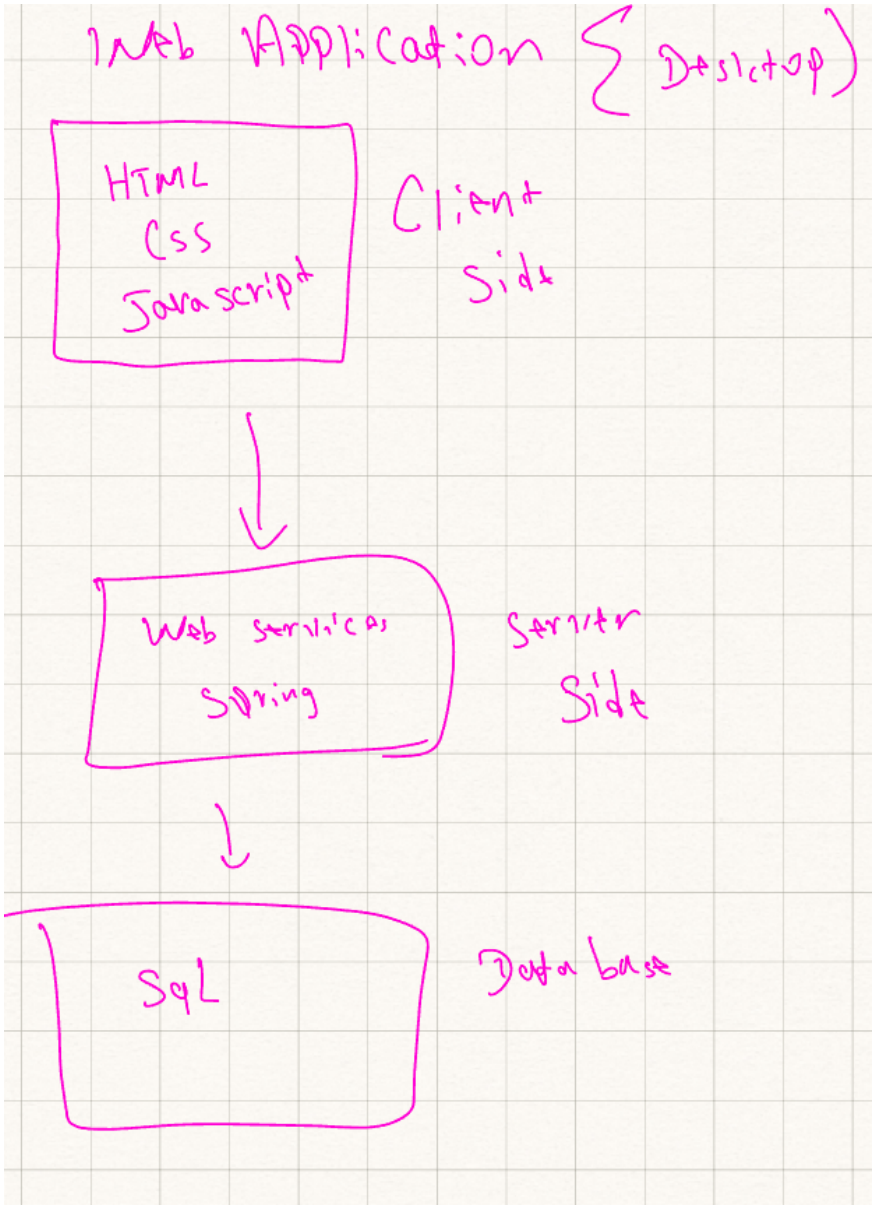
6. Monitor Real-time Updates
    - User: End User
    - Task: The user wants to see real-time updates as they customize the car.
    - Variations:
        - The user can observe how changes affect the car's appearance and specifications in real-time.

UML Activity or State Diagram

Given the dynamic nature of the CarCustomizer, where a user goes through a series of steps to customize their car, an Activity Diagram seems more appropriate. It can effectively map out each step in the car customization process, including decision points, such as choosing a color or a type of wheel, and illustrate the system's response to these actions.


Start Car Customization
Select Car Model
yes
Any special requests?
no
Apply Special Customizations
Proceed with Standard Customizations
Choose Color
Select Wheels
Choose Interior Options
yes
Add Enhanced Sound System?
no
Add Sound System
Continue Without Sound System
Review Customization
Save Customization
End Customization

Architecture Diagram



Data Storage

Storage Technology:
The application will use a relational database management system (RDBMS) to persist data. MySQL, a popular open-source RDBMS, is the chosen storage technology due to its reliability, scalability, and performance.

Data Storage:
The data will be stored in a MySQL database, hosted on a dedicated server or a cloud-based platform. This centralized storage approach ensures data consistency, easy access, and scalability.

Data Models:
The application will define the following data models:

- **User Profile Model:** This model stores user information, including usernames, passwords, and user-specific settings. It will have fields for user identification, authentication, and customization preferences.
- **Car Model Model:** This model represents car models available for customization, including information like model names, base specifications, images, and pricing. Each car model will have a unique identifier.
- **Car Customization Model:** This model tracks user-specific car customizations. It will include details on customized features, color, interior, sound system, and wheels, linked to the user profile and the selected car model.

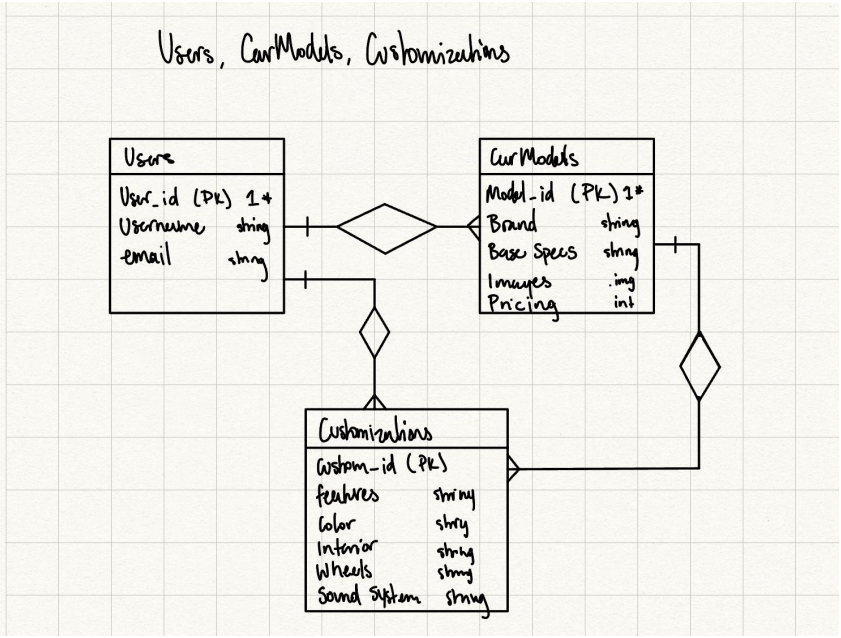
Data Relations:
Users will have a one-to-many relationship with customizations, meaning each user can have multiple car customizations. Car models will also have a one-to-many relationship with customizations, as multiple users can customize the same base model.

Data Access Libraries:
The application will utilize JDBC (Java Database Connectivity) to interact with the MySQL database. JDBC is a standard Java-based library that allows for database connection and SQL query execution. It provides a straightforward way to communicate with the MySQL database, including inserting, retrieving, updating, and deleting data.

Data Storage Diagram:
A simplified Entity-Relationship Diagram (ERD) provides an overview of the data structure for the application:

CarCustomizer ERD

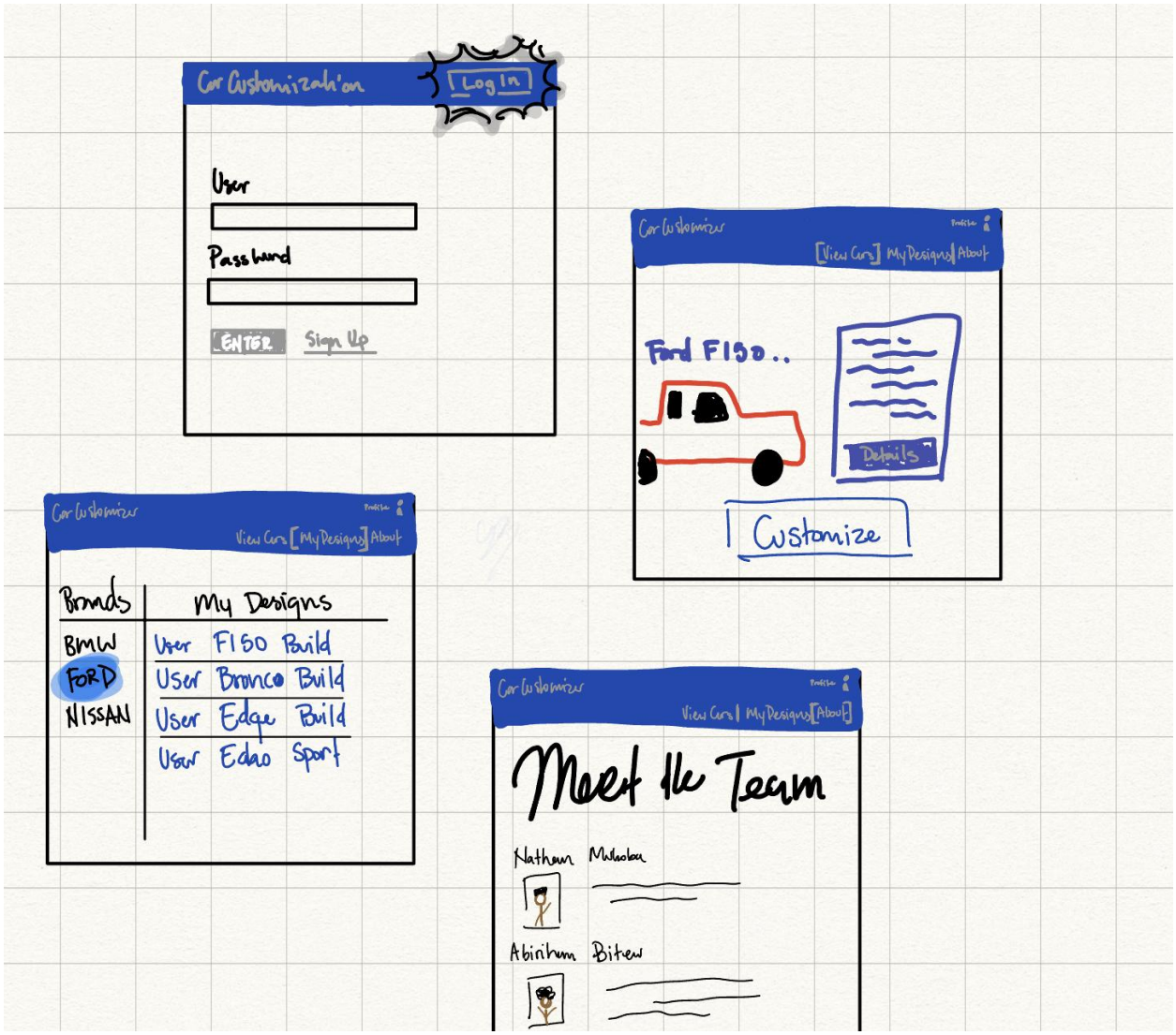
In the diagram:



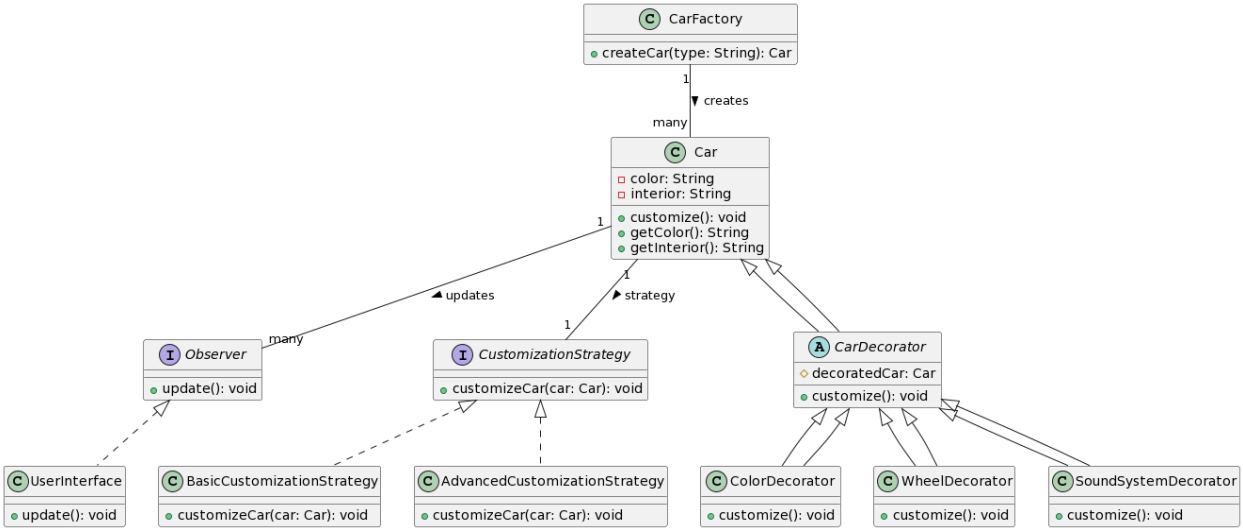
The "Users" table represents user profiles.
The "CarModels" table stores car model information.
The "Customizations" table links users and car models and includes customized feature details.
This diagram conveys the structure of the database and the relationships between the different data entities. It's important to note that this is a simplified representation, and the actual database design may include additional fields and tables as needed to fully support the application's functionality.

By utilizing MySQL and maintaining data in a structured database, we hope that the CarCustomizer application ensures data integrity and a scalable foundation for future enhancements.

UI Mockups/Sketches



UML Class Diagram & Pattern Use



This diagram includes:

- An ‘Observer’ interface for the Observer pattern, which will have a method update() that concrete observers must implement.
- A ‘CustomizationStrategy’ interface for the Strategy pattern, which defines a method for customizing cars.
- A ‘CarFactory’ class for the Factory pattern, which will be used to create ‘Car’ objects.
- A ‘Car’ class that will serve as the component to be decorated in the ‘Decorator’ pattern.
- Concrete ‘Observer’ classes like UserInterface that will be updated during the customization process.
- Concrete ‘CustomizationStrategy’ classes like ‘BasicCustomizationStrategy’ and ‘AdvancedCustomizationStrategy’ that define specific customization algorithms.
- Decorator classes such as ‘ColorDecorator’, ‘WheelDecorator’, and ‘SoundSystemDecorator’ that extend the Car class to add additional responsibilities dynamically.

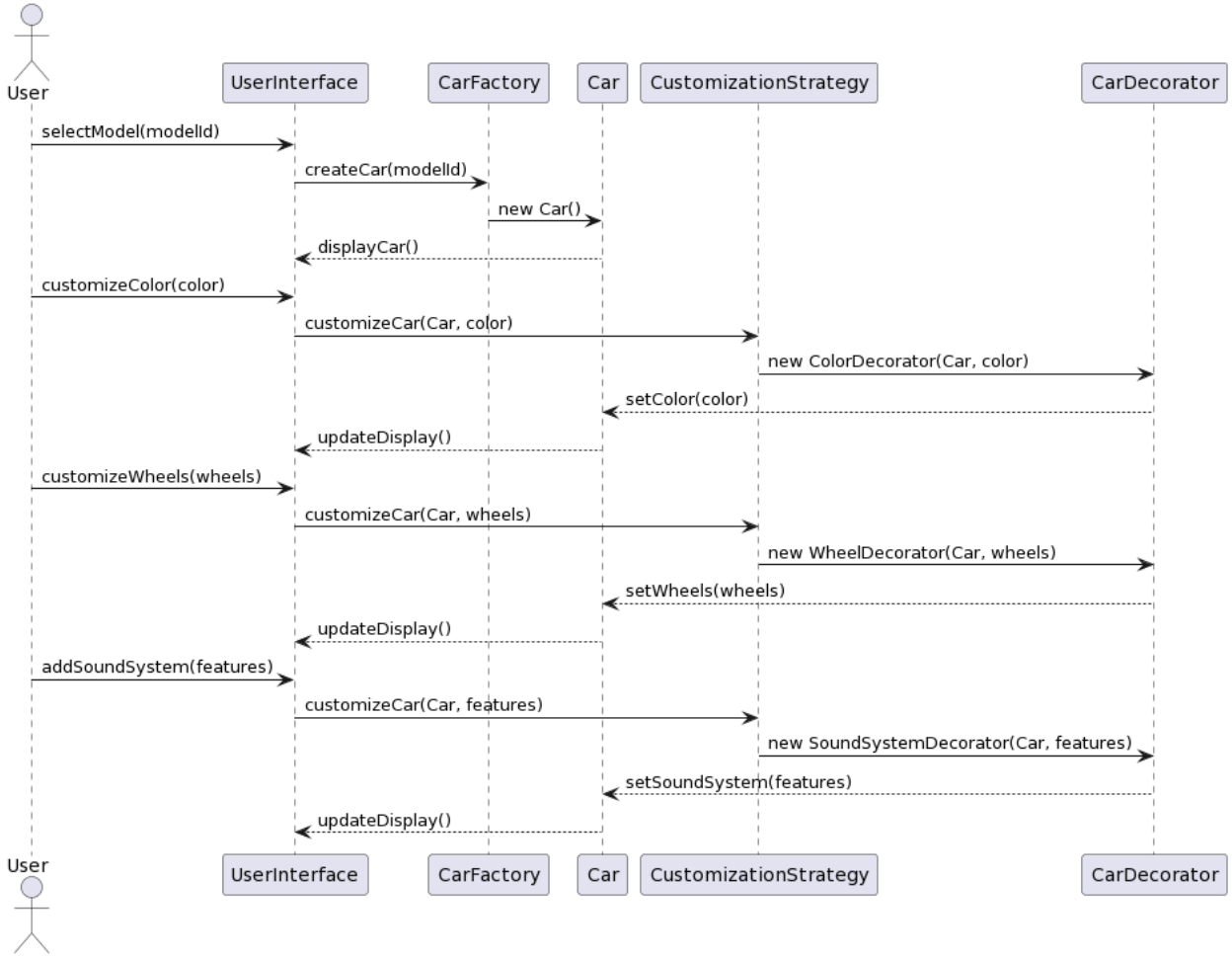
User Interactions/UML Sequence Diagrams

Customizing a Car: The user selects a car model and applies various customizations such as color, wheels, and sound system.

- The user starts by selecting a car model, which triggers the CarFactory to create a new Car object. The user then goes through a series of customization steps. Each step involves a CustomizationStrategy that applies the desired changes to the Car object. Decorator objects such as ColorDecorator, WheelDecorator, and SoundSystemDecorator are applied

to the car object to reflect the customizations. The UserInterface observes these changes and updates the display accordingly.

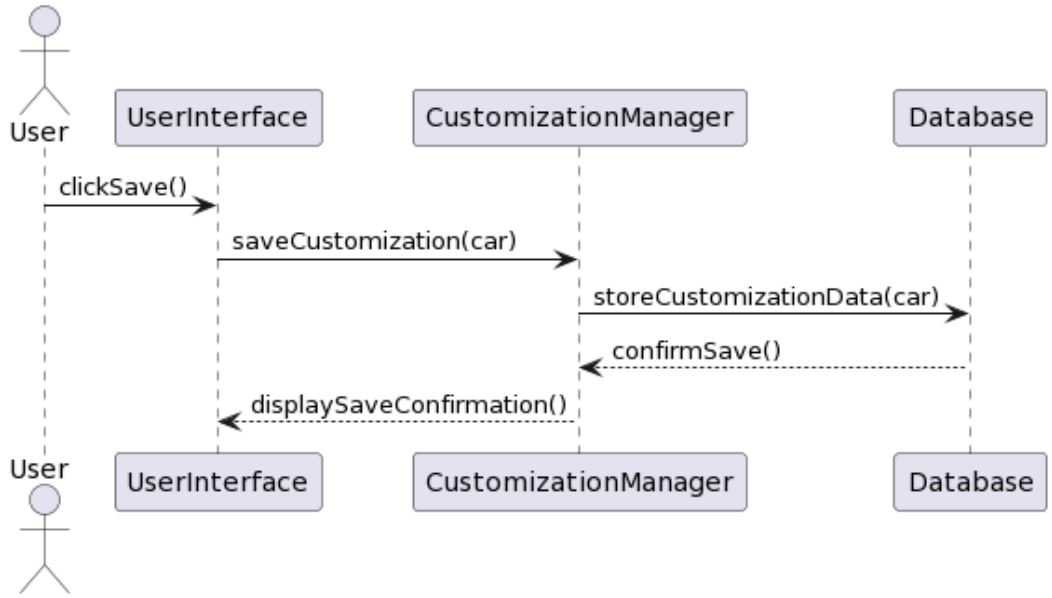
Sequence Diagram:



Saving Customization: After customizing the car, the user saves their configuration.

- Once the customization is complete, the user decides to save their configuration. The UserInterface sends a save request with the Car object's current state to a CustomizationManager controller, which then invokes the persistence layer to save the data into the database.

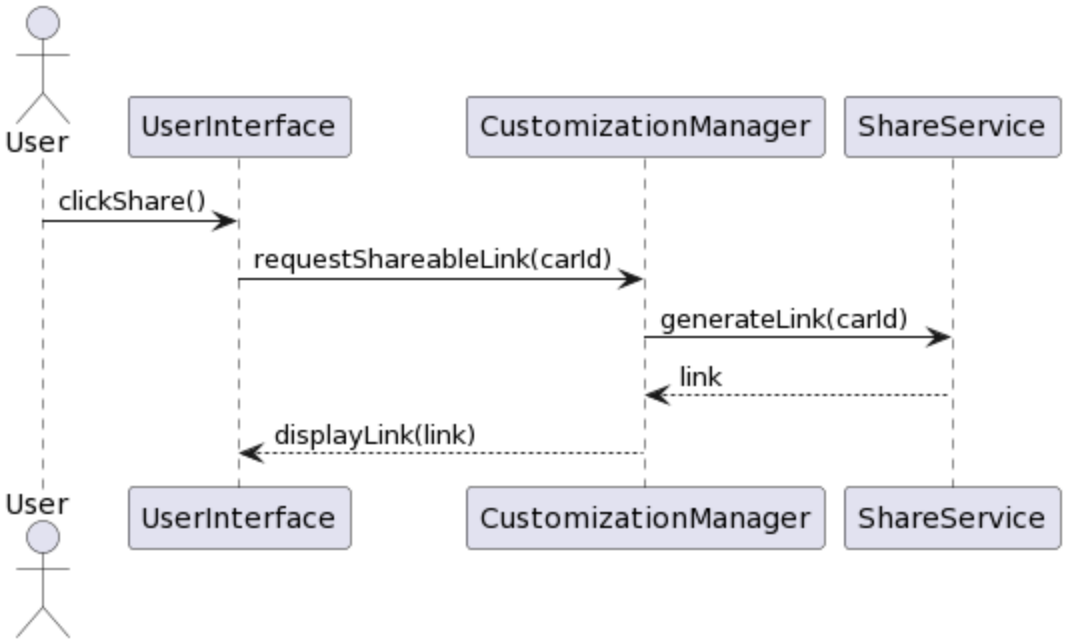
Sequence Diagram:



Sharing Customization: The user shares their customized car configuration.

- After saving the customization, the user wants to share their design. They initiate a share action, and the UserInterface requests a shareable link from the CustomizationManager. The CustomizationManager retrieves the appropriate data and generates a link, which is then displayed to the user for sharing.

Sequence Diagram:


User
UserInterface
CustomizationManager
ShareService
clickShare()
requestShareableLink(carId)
generateLink(carId)
link
displayLink(link)
User
UserInterface
CustomizationManager
ShareService