**Flipkart Clone**

Abhinand Bhat – PES1UG20CS007

Abhinav Vasireddy – PES1UG20CS009

Anirudh RH - PES1UG20CS057

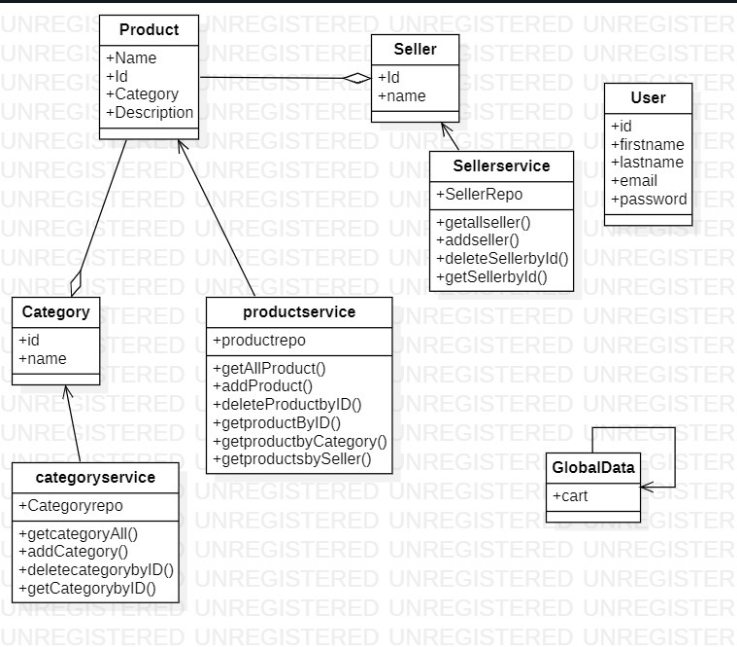
Synopsis

This mini project aims to create a basic Flipkart clone, where customers can browse products, view product details, add items to their cart, and complete checkout, while the admin can change the backend of the website. This is done using Spring Boot framework.

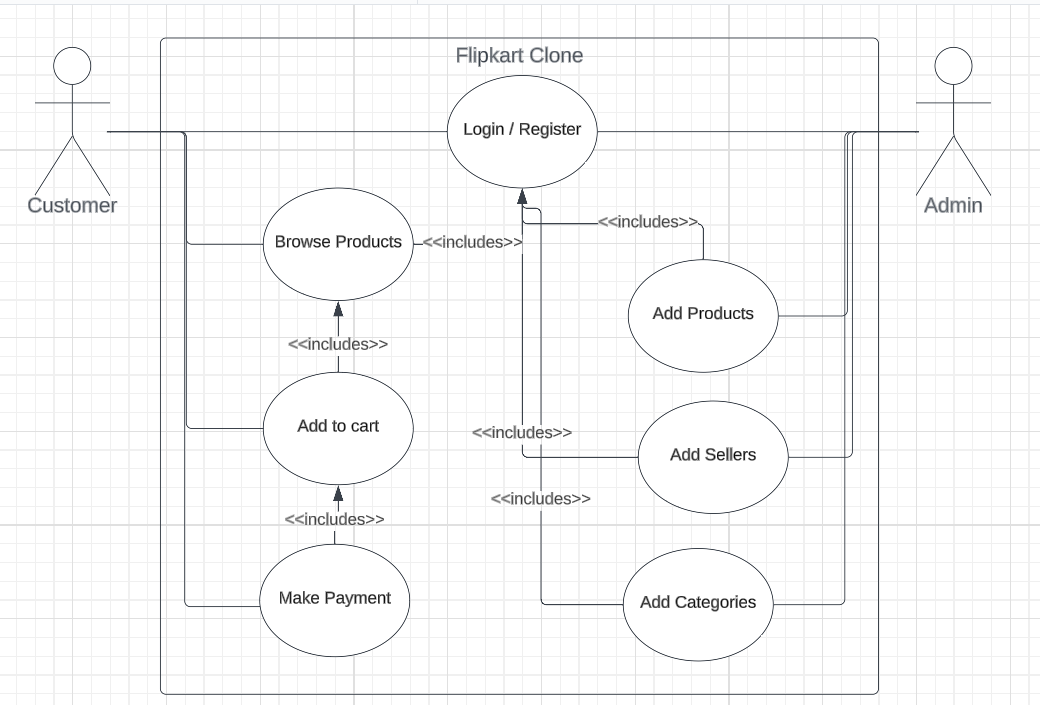
The front-end uses HTML, CSS, and JavaScript. The back-end will be powered by Java, with Spring Boot providing the necessary tools and libraries for building RESTful APIs

Diagrams Used

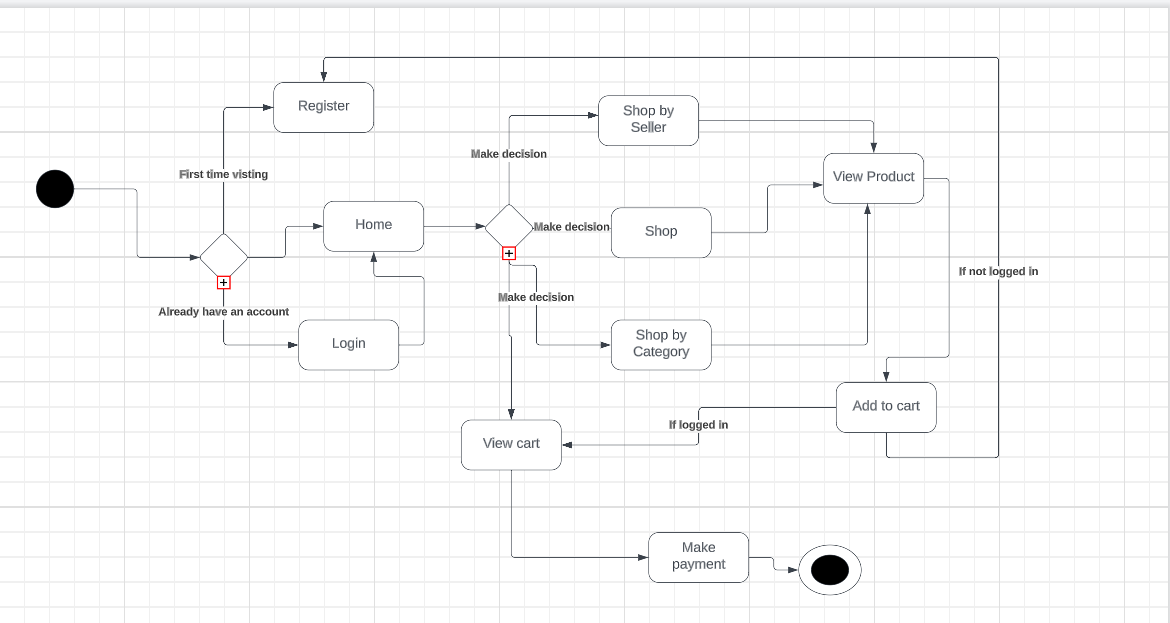
Class Diagram



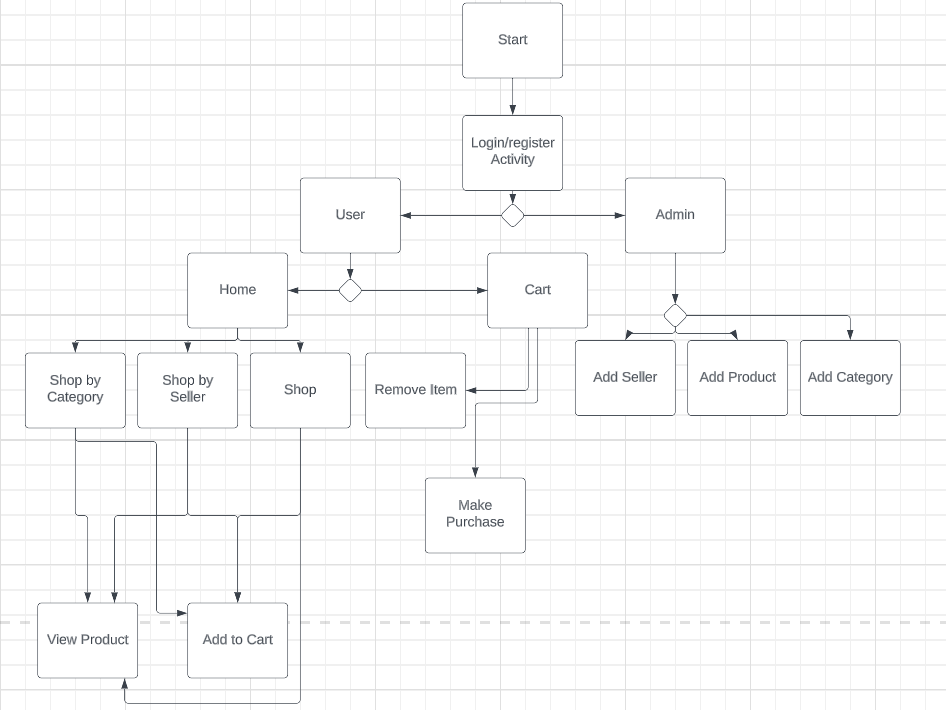
Use Case Diagram



State Diagram

****

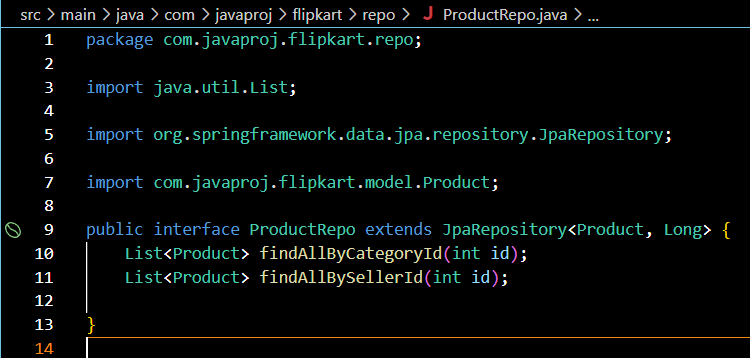
Activity Diagram



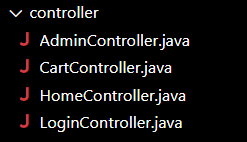
**Design Principles Used**

**Open-Close**

**In the following code, the ProductRepo interface extends JpaRepository. The repository provides implementations of basic crud operations. This code follows the Open Close Principle as the crud operations cannot be modified by us, but the interface is open to new functions, like findByCategoryId and findBySellerId functions.**

****

**Single Responsibility Principle**

****

**Instead of creating one massive controller class to deal with all the url mappings, we’ve split it up based on the usage.**

**Design Patterns Used**

**DTO (product)**

****

**The productDTO object is used to transfer data between different layers of the application, specifically between the view and the controller. The data submitted in the new product form is captured in the productDTO object and passed to the controller method as a parameter. This helps in encapsulating the data and passing it between different layers without having to worry about the underlying objects or entities.**

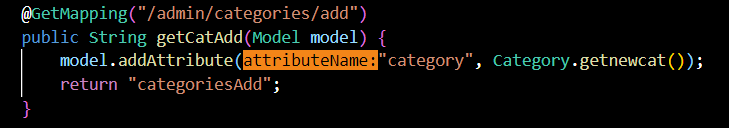
**Singleton(cart)**

****

**Here, the GlobalData class is being implemented with a singleton pattern.**

**In the class, the attribute cart can only have one instance in a particular session as multiple carts are meaningless. In the future if we**

**Factory**

****

**In the AdminController class, we implemented the factory pattern for adding new categories, sellers, products.**

**Instead of creating the new object in the admin class itself, we delegate the work to the Category class itself so that if we need to make any changes to the creation of a category in the future, we can implement it without having to change the admin class**