



UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

“UPI CLONE”

Submitted by:

TEJA YADAV S-PES1UG21CS668
TEASHWIN SJ-PES1UG21CS667
DHENU SARANG -PES1UG21CS660
SUMUKHA GANESHA-PES1UG21CS641

6th Semester K Section

Prof. Bhargavi Mokashi
Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING

Synopsis

This project is a Unified Payments Interface (UPI) clone developed using design principles in the Java programming language. The Unified Payments Interface is an instant real-time payment system developed by the National Payments Corporation of India. The project focuses on implementing key design principles, including but not limited to SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion), to ensure a robust, scalable, and maintainable UPI system. The implementation encompasses features such as account management, fund transfers, transaction history, and security protocols. By applying design patterns and principles in Java, this project aims to provide a streamlined and secure UPI platform following best practices in software design and architecture.

Architecture:

The application employs the model - view - controller (MVC) architecture pattern, a widely used design patterns in software engineering. MVC separates an application into three interconnected components, facilitating better organization, maintainability, and scalability.

1. Model Component

The Model component manages the application's data, business logic, and rules independently of the user interface. It handles data storage, retrieval, and manipulation, ensuring consistency and integrity.

Characteristics:

- Data Management: Models store and manipulate data structures pertinent to the application's domain.
- Business Logic: Contains algorithms and rules governing data processing and behavior.
- Change Notification: Notifies views of any data changes, enabling real-time updates.

This includes entities such as venues,vendors, events, bookings, and user details.

2 . View Component

The View component manages the user interface (UI) and presentation logic of the application.

Characteristics

- User Interaction: Renders graphical or textual output to users and captures user input.
- Multiple Views: Supports multiple views for a single model, facilitating diverse presentation options.
- Passive Strategy: Implements a passive view approach, separating UI logic from data processing.

It displays data from the Model to users and forwards user commands to the Controller.

Views display information retrieved from the Model and provide user-friendly interfaces for performing actions such as booking venues, and managing events.

3 . Controller Component The Controller component acts as an intermediary between the Model and View, handling user input and application logic. It interprets user actions, manipulates data in the Model,and updates the View accordingly It handles user requests, processes input data, and updates the Model based on user interactions.

Characteristics:

- Event Handling: Processes user events initiated in the View and triggers corresponding actions.
- Coordinating Role: Determines appropriate responses to user interactions, orchestrating interactions between Model and View.
- Input Conversion: Converts user inputs into commands for the Model or View, facilitating data processing.

In our event management portal, Controllers receive requests from users, invoke appropriate methods in the Model to perform actions (e.g., adding a booking, updating event details), and render the corresponding Views to display the results.

Design Principles

Single Responsibility Principle (SRP):

Each method in the controller seems to have a single responsibility. For example, methods like `home()`, `login()`, `register()`, `contactus()`, etc., handle specific HTTP requests and return corresponding views. The controller delegates business logic to service classes, ensuring separation of concerns.'

Open/Closed Principle (OCP):

The controller methods are open for extension (e.g., adding new functionalities) but closed for modification. Adding new features would involve adding new methods or modifying existing methods, without changing their core implementation.

Low Coupling:

We have made sure that there is not much of dependency between two components by making a separate model, view, controller for each component.

Design Pattern

Factory Design Pattern:

In the Factory design pattern, a factory class (e.g., `ServiceFactory`) is responsible for creating instances of various service classes (e.g., `HotelService`, `CateringService`). This decouples the client code from the concrete implementations of services, allowing for easier maintenance and flexibility in changing or adding new service types.

Adapter Design Pattern:

The Adapter design pattern is a structural pattern that allows the interface of an existing class to be used as another interface. It acts as a bridge between two incompatible interfaces, making them work together. This is done paying either through upi id or phone number.

Singleton Design Pattern:

The Singleton design pattern ensures that a class has only one instance and provides a global point of access to that instance. In this pattern, a class (e.g., ConfigurationManager) ensures that only one instance of itself is created and provides a static method (e.g., getInstance()) to access that instance. This pattern is useful for managing resources that should be shared across the application, such as configuration settings or database connections.


LOGIN:



The image shows a login interface for a 'UPI system'. The background is a solid blue color with a yellow diagonal stripe on the left side. At the top center, there is a small square image showing a hand holding a smartphone with a UPI app interface. Below this, the text 'UPI system' is displayed in a large, white, sans-serif font. Underneath the title, there are two input fields: 'Username:' followed by a white rectangular box, and 'PIN:' followed by another white rectangular box. Below the input fields, there are three buttons: 'SIGN IN' and 'CLEAR' are small, white, rounded rectangular buttons with black text, positioned side-by-side. Below these two buttons is a larger, white, rounded rectangular button with the text 'SIGN UP' in black. In the bottom right corner, there is a small square image with a green and yellow gradient background, featuring the UPI logo.

SIGNUP PAGE:

APPLICATION FORM



Sign up Details

Page 1
Personal Details

User name

Pin

Phone number

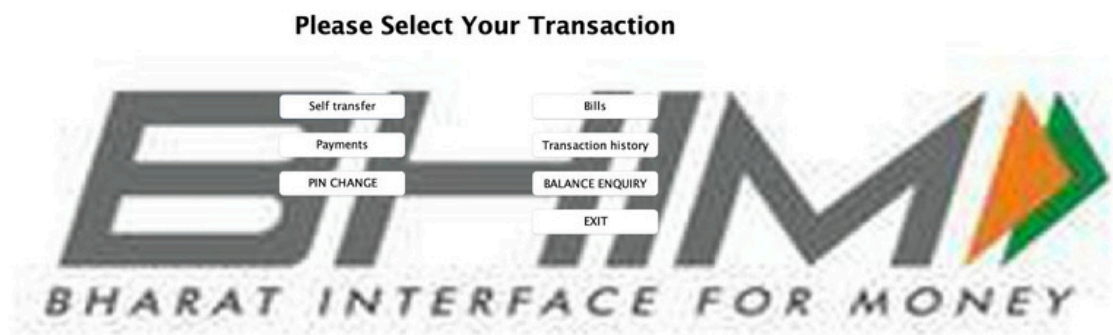
Bank Name

full name

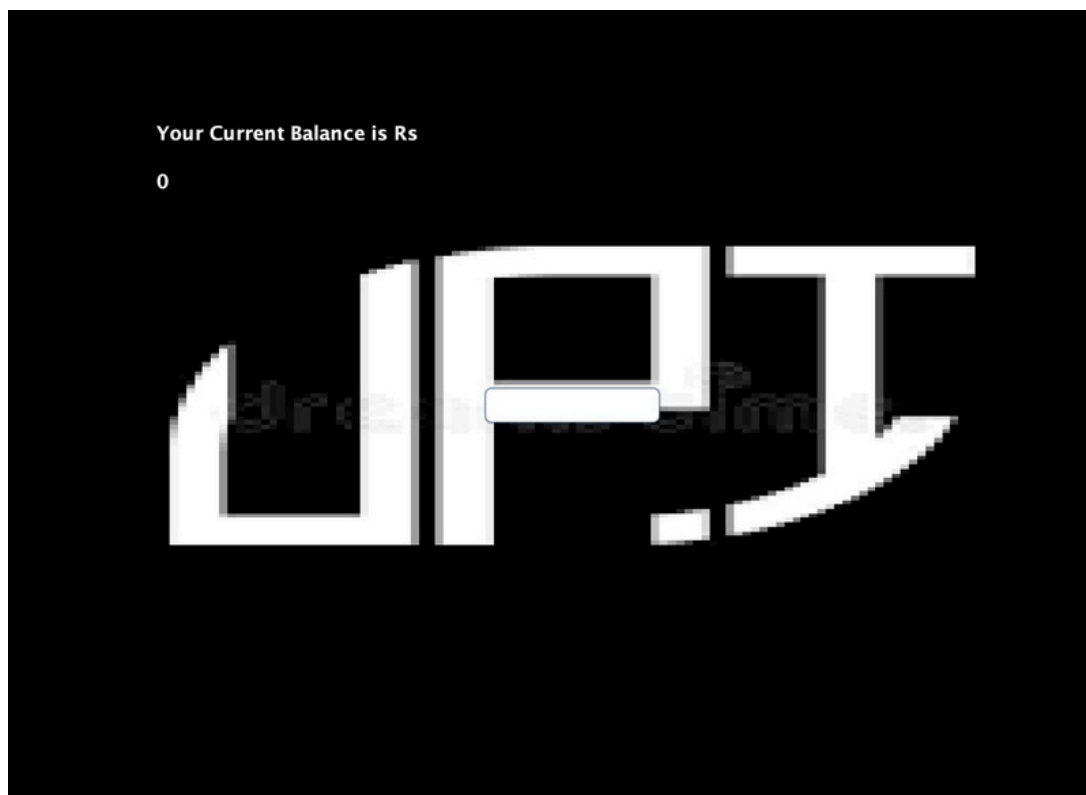
DOB

Address

PAYMENT INTERFACE



BALANCE ENQUIRY:



BILL PAYMENTS:

Payment System

Select Bill Type

☐ Water Bill

☐ Electricity Bill

☐ Gas Bill

Enter Amount

Pay

UPI PIN CHANGE :

CHANGE YOUR PIN

New PIN:

Re-Enter New PIN:

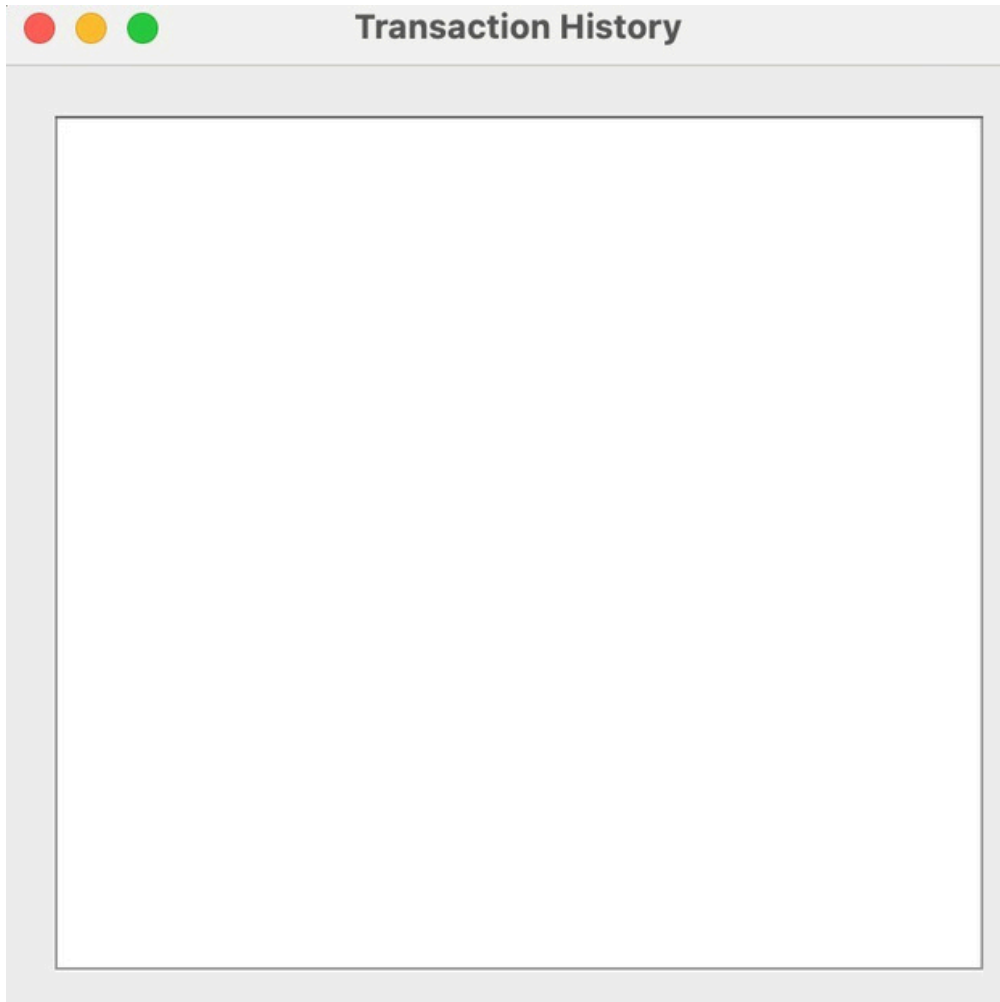
CHANGE

BACK

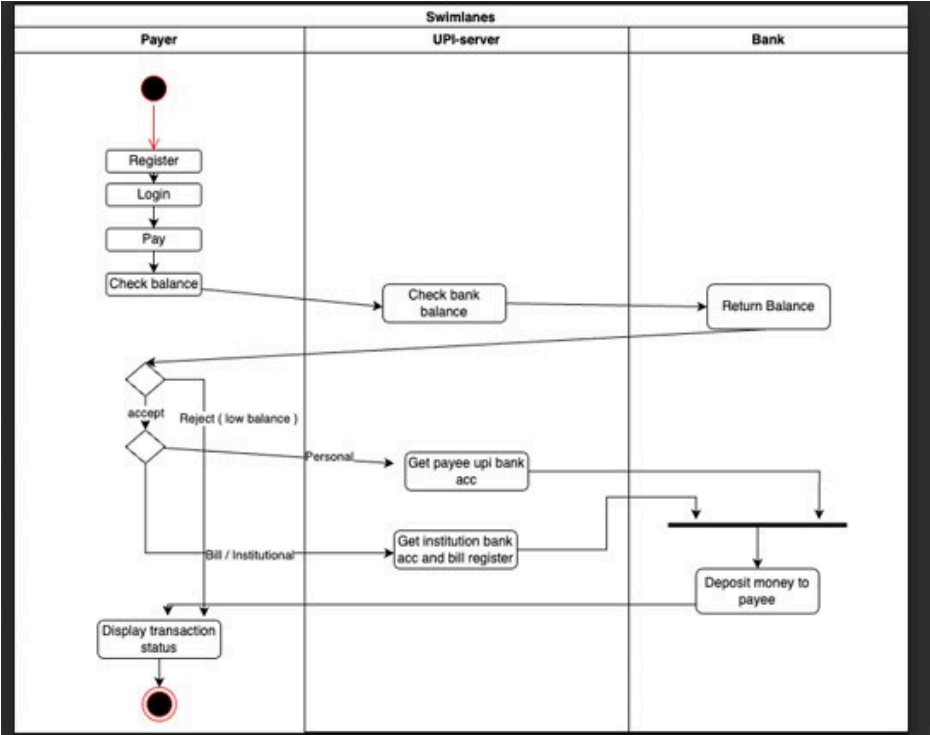
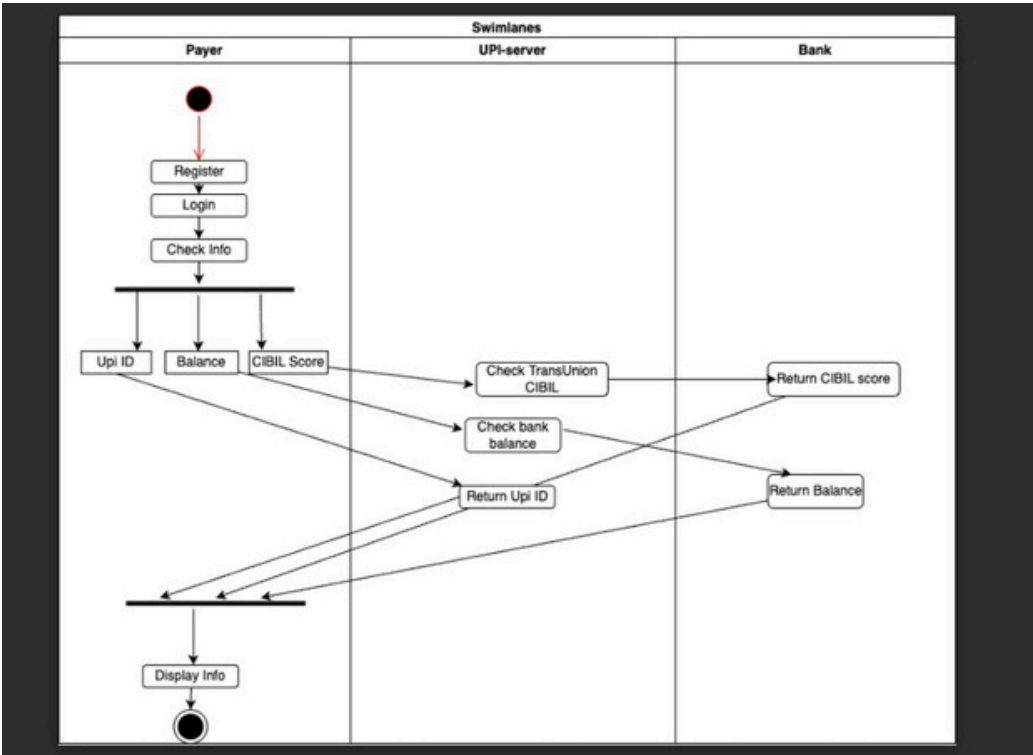
BT PIN

AT INTERFACE FOR

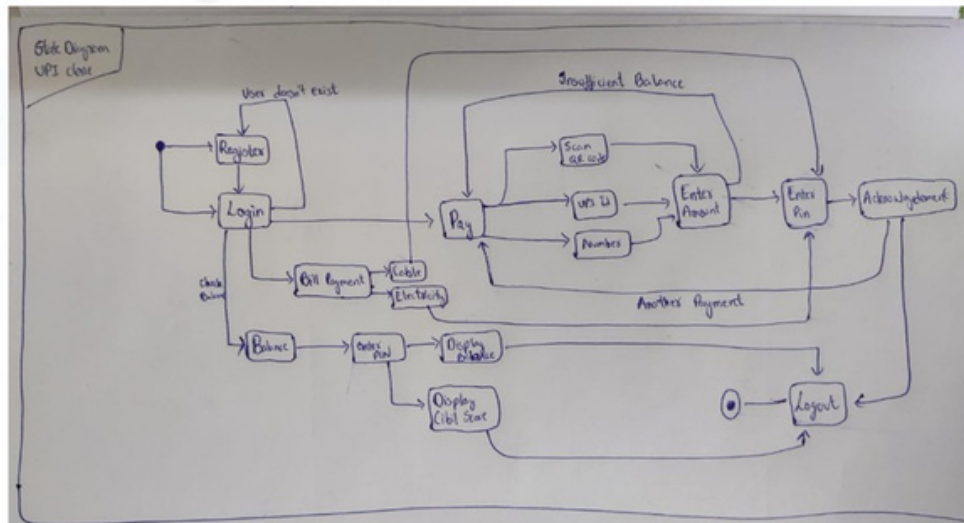
HISTORY TAB SYNCHRONIZED WITH SQL



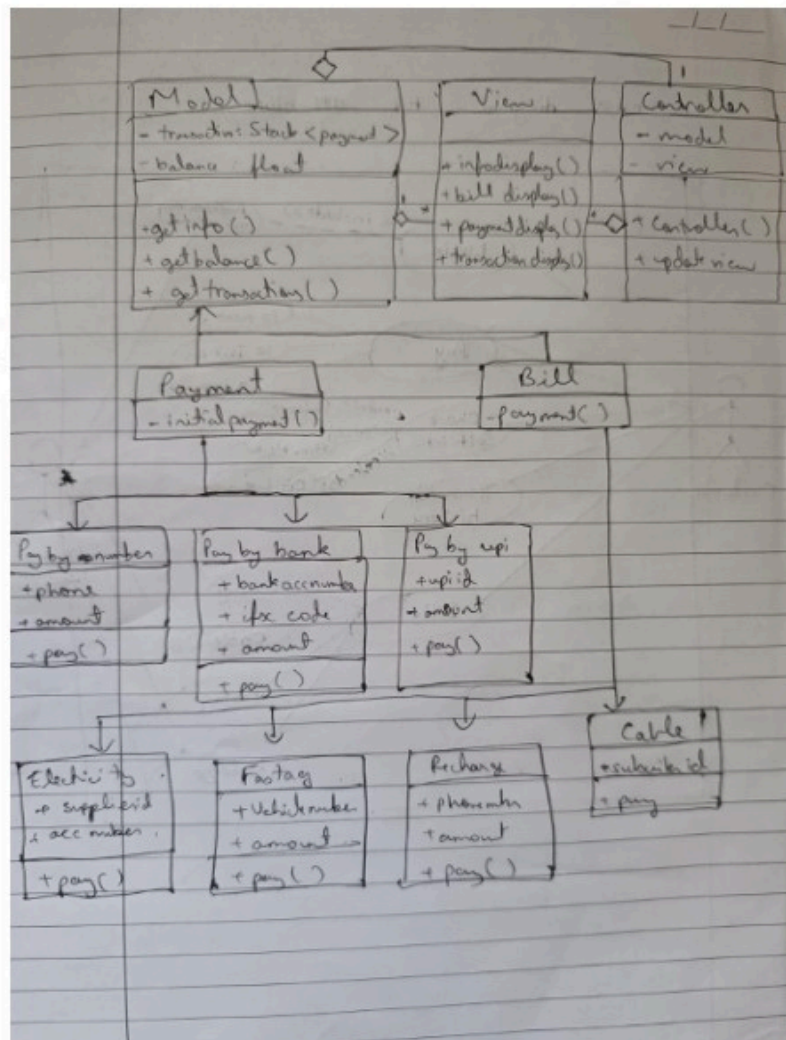
ACTIVITY DIAGRAM



STATE DIAGRAM



CLASS DIAGRAM



USE CASE DIAGRAM