PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

MINI PROJECT REPORT

CHATBOT APPLICATION

Object Oriented Analysis and Design using Java (UE20CS352)

Submitted by:

Name	SRN
K N Pragathy Reddy	PES2UG20CS151
Kavya J	PES2UG20CS159
Anjali K	PES2UG20CS160
Manodyna K H	PES2UG20CS187

Team Number - 16

TABLE OF CONTENTS

Introduction to Problem statement

The problem statement chatbot is an application that uses Java technologies to help users converse with each other.This type of chatbot can be particularly useful in fields such as software development, where a well-defined problem statement is essential for success.

use Java frameworks like Spring Boot to build the chatbot's user interface, handle user input, and integrate with external systems such as project management tools or databases. Spring Boot provides a fast and easy way to build web-based applications, and it includes built-in support for popular web technologies like RESTful APIs and WebSocket communication

SYNOPSIS

A chatbot is a computer program that uses some techniques to simulate human conversations. In this project, we aim to develop a chatbot using Java programming language that can interact with multiple users simultaneously and can communicate with them via a server.

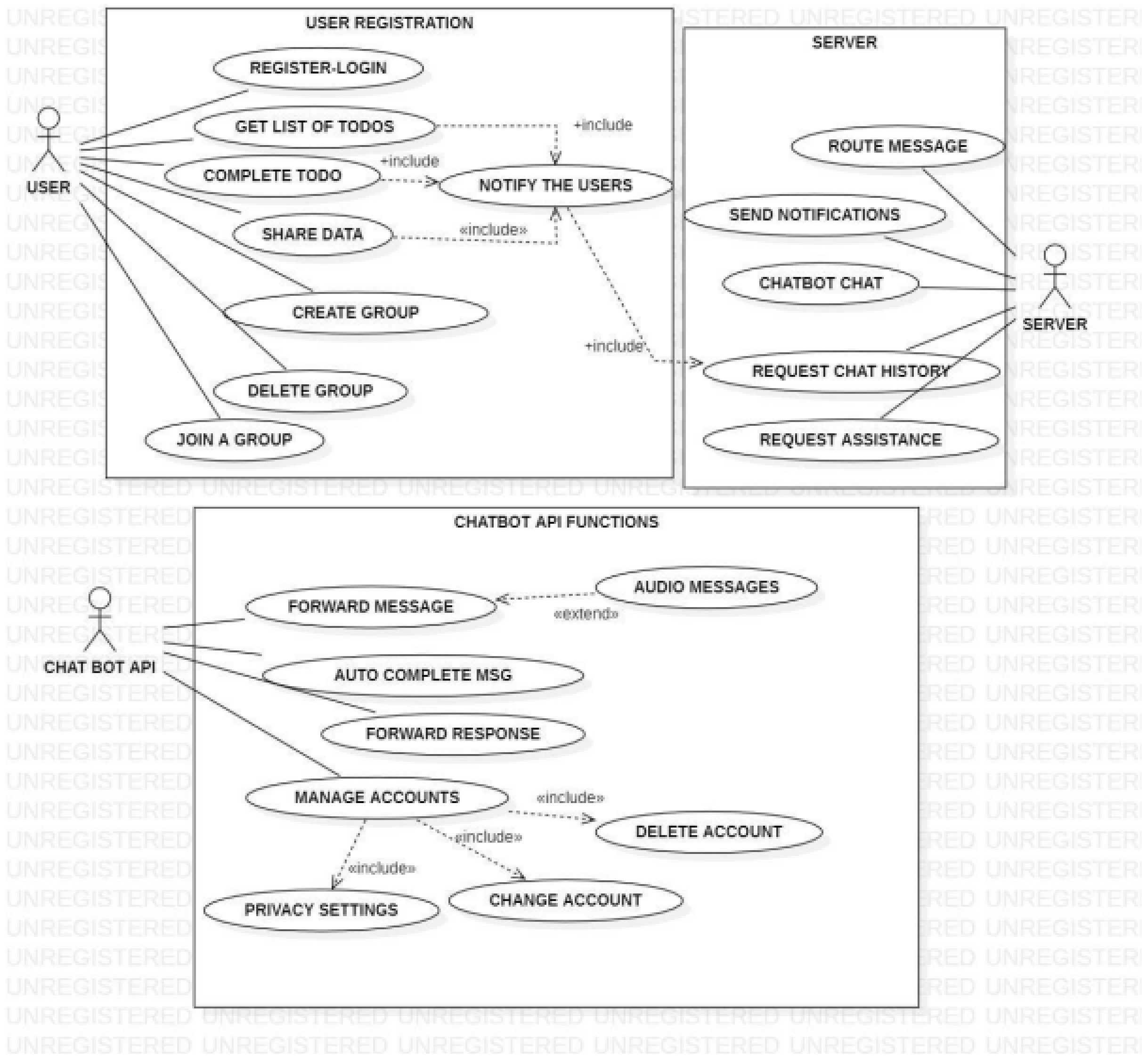
The chatbot will be built using the client-server architecture where the clients will connect to the server to communicate with the chatbot. The chatbot will have a database of predefined responses to user queries, and it will use some techniques to understand user queries and provide relevant responses. The chatbot will also be able to learn from its interactions with users and improve its responses over time.

The server will be responsible for managing user requests and responses, handling user authentication, and maintaining the database of predefined responses. It will also be responsible for routing user requests to the appropriate chatbot instance and ensuring that each user gets a personalized response. To enable multiple users to communicate with the chatbot simultaneously, we will use multithreading in Java. Each user will have a separate thread that will handle their requests and responses. The server will also use multi-threading to manage multiple client connections.

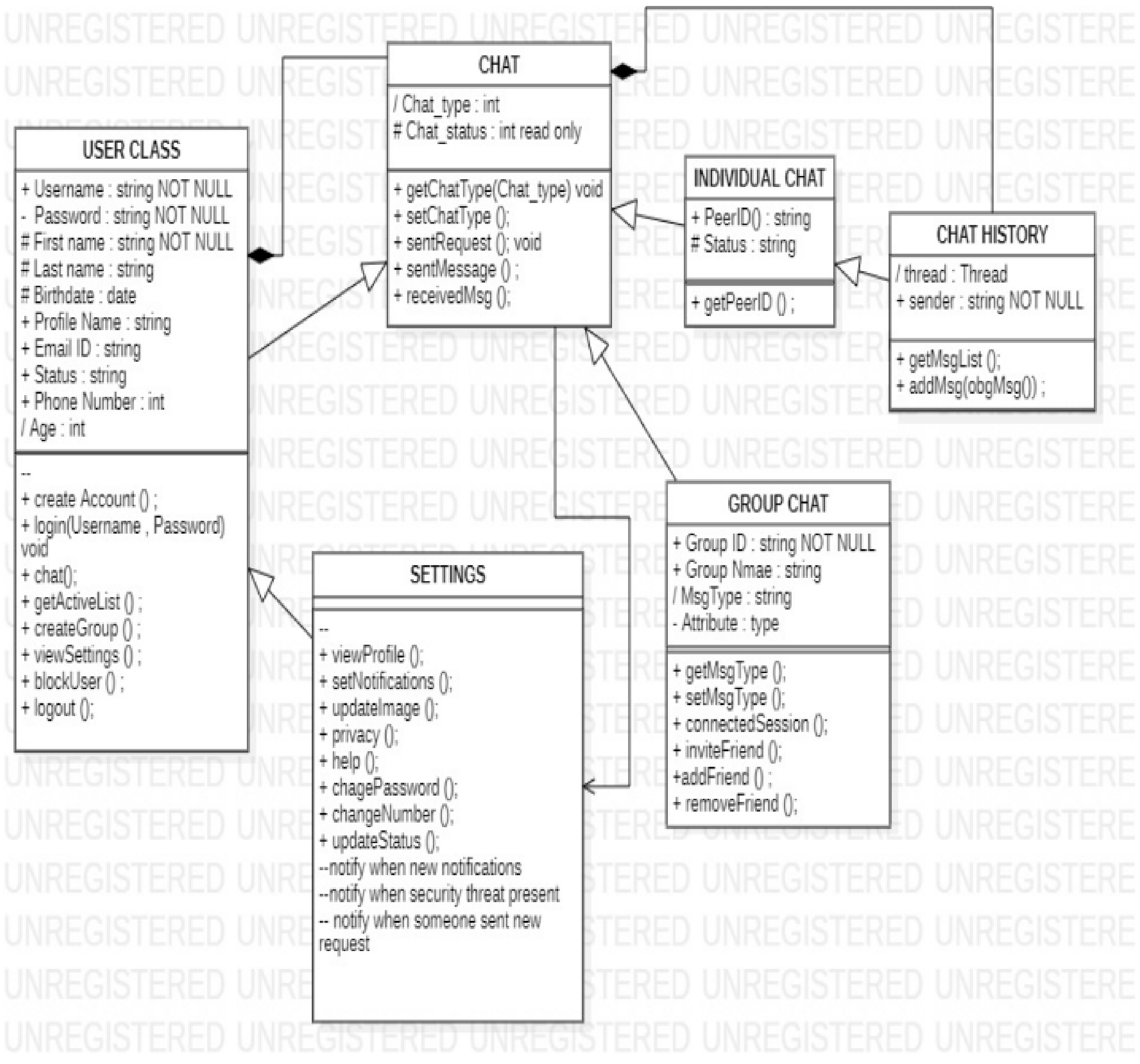
Overall, this project aims to develop a robust and scalable chatbot using Java programming language that can interact with multiple users and provide personalized responses using some techniques.

Models (Use Case and Class Models)

USE CASE DIAGRAM



CLASS DIAGRAM



Architecture Patterns, Design Principles and Design Patterns

Architecture Patterns:

Model-View-Controller (MVC) Architecture Pattern:

This pattern separates an application into three interconnected components: Model (business logic), View(user interface), and Controller (handles user input). This architecture pattern can be used for building chatbot applications as it provides a clear separation of concerns and promotes modular design.

Design Principles:

a) Single Responsibility Principle (SRP): This principle states that a class should have only one reason to change. This principle can be applied to chatbot development to ensure that each component of the application has a single, well-defined responsibility.

b) Open / Closed Principle (OCP): The Open/Closed Principle (OCP) is a design principle in object-oriented programming that states that software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification. In other words, once a class or module has been written and tested, it should not be modified, but rather extended to add new functionality.

c) Dependency Inversion (DI): This principle promotes loose coupling between components by allowing them to depend on abstractions rather than concrete implementations. This principle can be used to make chatbot development more modular and testable.

Design Patterns:

a) Facade Pattern: This pattern provides an interface for creating objects, but allows subclasses to decide which classes to instantiate. This pattern can be used for creating chatbot components such as message parsers and response generators.

b) Chain of Responsibility Pattern: This pattern allows multiple objects to handle a request in a chain-like structure. This pattern can be used for building chatbot message processing pipelines where each component can handle a specific type of message.

c) Strategy Pattern: The strategy pattern can be used in chatbot development to allow for interchangeable behavior based on the user's input or context.In this chatbot application may need to respond differently to different types of messages or user inputs.

Github link to the Code base

Pragathy Reddy PES2UG20CS151:
https://github.com/pragathy1572/PES2UG20CS151_OOAD-PROJECT.git

Kavya J PES2UG20CS159:
https://github.com/Kavyaj554/OOAD_Project_Chatbot.git

Anjali K PES2UG20CS160:
https://github.com/anjalikesanapalli/OOAD_PROJECT.git

Manodyna KH PES2UG20CS187:
https://github.com/manodyna/OOAD_PROJECT.git

Individual contributions of the team members

Pragathy Reddy PES2UG20CS151:

Pragathy Reddy implemented design patterns such as the Strategy pattern to allow for interchangeable behavior based on user inputs or context. Additionally,she refactored the frontend of the chatbot application to follow the Facade pattern, simplifying the interface and providing a unified interface for the user to interact with the chatbot backend. The use of the Facade pattern promotes loose coupling between the frontend and backend components and simplifies maintenance and future development of the chatbot application.

Kavya J PES2UG20CS159:

Kavya J implemented a chatbot application using Java technologies with MVC framework. To ensure the maintainability and scalability of the application, she has followed design principles like the Single Responsibility Principle (SRP) and the Dependency Inversion (DI) principle, and have utilized design patterns such as the Factory Pattern and Chain of Responsibility Pattern. Additionally, she has applied the Open/Closed Principle (OCP) to enable easy extension of the application with new features and APIs. She has integrated WebSockets into the chatbot application to enable real-time communication between the chatbot and users.

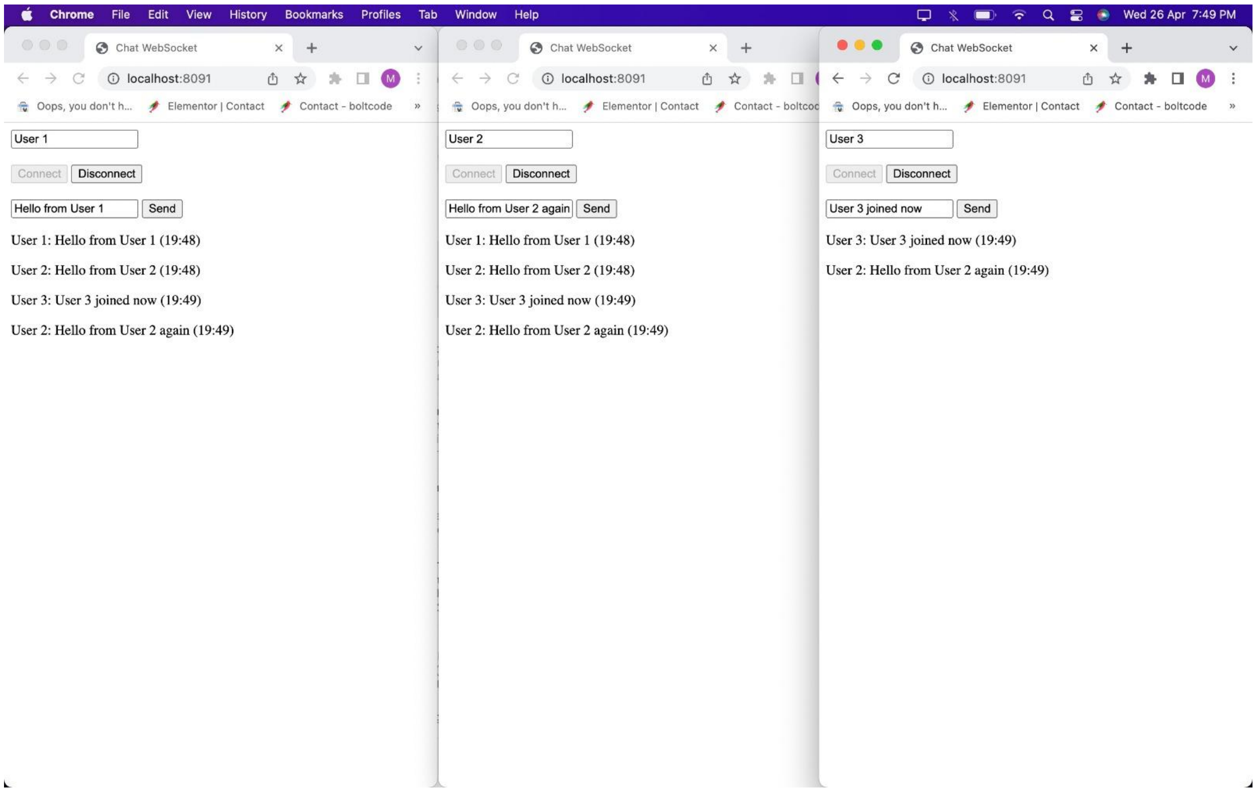
Anjali K PES2UG20CS160:

Anjali K designed and implemented a chat system that allows for two users to engage in a conversation, as well as the ability to chat with chatbots. She utilized the Model-View-Controller (MVC) architectural pattern to create a robust and scalable chat platform. Through this framework, they developed and integrated various components such as user authentication, message processing, and API integrations, resulting in a seamless and user-friendly chat experience. Overall, she did chat system that facilitates human-to-human and human-to-bot interactions.

Manodyna KH PES2UG20CS187:

To ensure the application is easily extensible and maintainable over time,Manodyna K H has refactored the code to follow the Strategy pattern.Specifically, she has implemented a family of response strategies that encapsulate different response behaviors based on the user's input or context. This allows the chatbot to select and apply the appropriate response strategy at runtime, making it more flexible and customizable.Additionally, she has used the Strategy pattern to encapsulate external APIs or services, such as weather APIs, making them interchangeable and easily replaceable if needed.

Screenshots with input values populated and output shown



User 1

Hello from User 1

User 1: Hello from User 1 (19:48)

User 2: Hello from User 2 (19:48)

User 3: User 3 joined now (19:49)

User 2: Hello from User 2 again (19:49)

User 2

Hello from User 2 again

User 1: Hello from User 1 (19:48)

User 2: Hello from User 2 (19:48)

User 3: User 3 joined now (19:49)

User 2: Hello from User 2 again (19:49)

User 3

User 3 joined now

User 3: User 3 joined now (19:49)

User 2: Hello from User 2 again (19:49)