Complaint / Helpdesk Management System

Software Engineering Course Work

B.Tech – Computer Science Engineering, MNIT Jaipur

Kavyansh Bagdi	2023UCP1701
Nikhil Kumawat	2023UCP1713
Ankit Sharma	2023UCP1697

Github Link : https://github.com/Kavyansh-Bagdi/helpdesk-management/

Software Requirements Specification (SRS)

1. Introduction

This document outlines the Software Requirements Specification (SRS) for the Complaint / Helpdesk Management System. This system is designed to streamline the process of managing customer support tickets, providing a platform for users to submit issues and for agents to track and resolve them efficiently.

2. Objective and Scope

2.1 Objective

The primary objective of the Complaint / Helpdesk Management System is to provide a robust, user-friendly, and efficient platform for managing support tickets. It aims to:

- Facilitate easy submission and tracking of support tickets by customers.
- Enable support agents to effectively manage, prioritize, and resolve tickets.
- Improve communication between customers and support staff.
- Provide administrators with tools to oversee system operations and user management.

2.2 Scope

The system will encompass functionalities for user authentication (registration, login), ticket management (creation, viewing, updating, commenting), and user role management (customer, agent, admin). It will provide a clear separation between frontend user interfaces and backend API services.

3. Functional Requirements

The system shall provide the following functional capabilities:

3.1 User Management

- **FR1.1 User Registration:** Users shall be able to register for an account with a unique username and email.
- **FR1.2 User Login:** Registered users shall be able to log in to the system using their credentials.
- **FR1.3 Role-Based Access Control:** The system shall support different user roles (customer, agent, admin) with varying levels of access to features.
    - Customers can create, view, and comment on their own tickets.
    - Agents can view all tickets, update ticket status, and add comments.
    - Admins can manage all users and tickets.

3.2 Ticket Management

- **FR2.1 Create Ticket:** Customers shall be able to create new support tickets, providing a title and description.
- **FR2.2 View Tickets:** Users shall be able to view a list of tickets relevant to their role.
    - Customers see their own tickets.
    - Agents and Admins see all tickets.
- **FR2.3 View Ticket Detail:** Users shall be able to view the detailed information of a specific ticket, including its description, status, and associated comments.
- **FR2.4 Update Ticket Status:** Agents and Admins shall be able to update the status of a ticket (e.g., 'open', 'in_progress', 'closed').
- **FR2.5 Add Comment:** Users (customers, agents, admins) shall be able to add comments to a ticket.

4. Non-Functional Requirements

4.1 Performance

- **NFR1.1 Response Time:** The system shall respond to user requests (e.g., login, ticket creation, viewing tickets) within 2 seconds under normal load.
- **NFR1.2 Scalability:** The system shall be capable of handling an increasing number of users and tickets without significant degradation in performance.

4.2 Security

- **NFR2.1 Authentication:** User passwords shall be securely hashed and stored.
- **NFR2.2 Authorization:** Access to system functionalities and data shall be strictly enforced based on user roles.
- **NFR2.3 Data Protection:** Sensitive user and ticket data shall be protected against unauthorized access and disclosure.

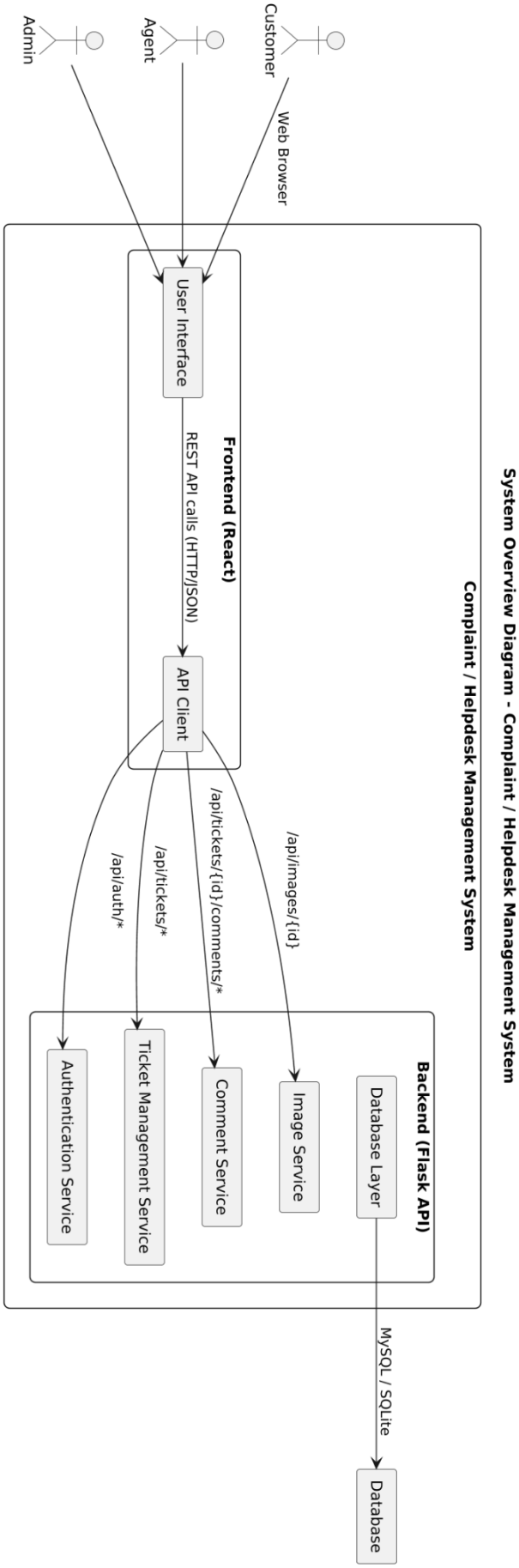
4.3 Usability

- **NFR3.1 User Interface:** The user interface shall be intuitive and easy to navigate for all user roles.
- **NFR3.2 Accessibility:** The system shall adhere to common accessibility standards to ensure usability for a diverse range of users.

4.4 Maintainability

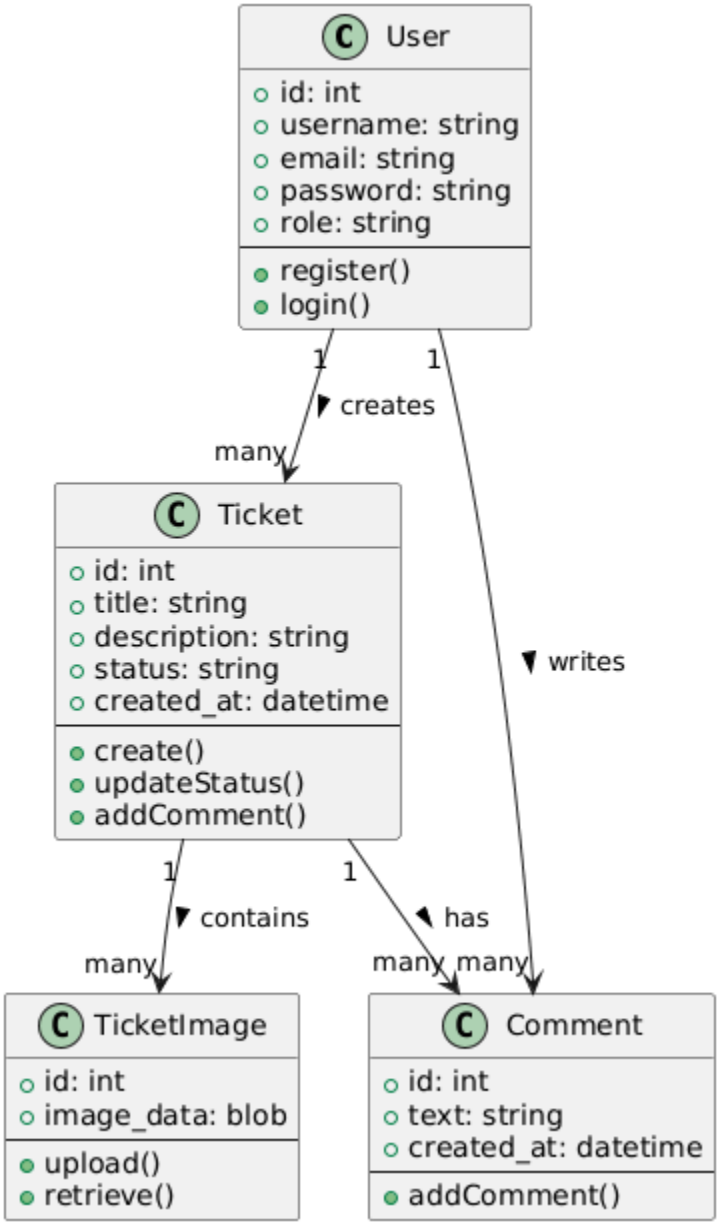
- **NFR4.1 Code Quality:** The codebase shall be well-structured, documented, and follow established coding standards to facilitate future maintenance and enhancements.
- **NFR4.2 Testability:** The system components shall be designed to be easily testable.

5. System Overview Diagram

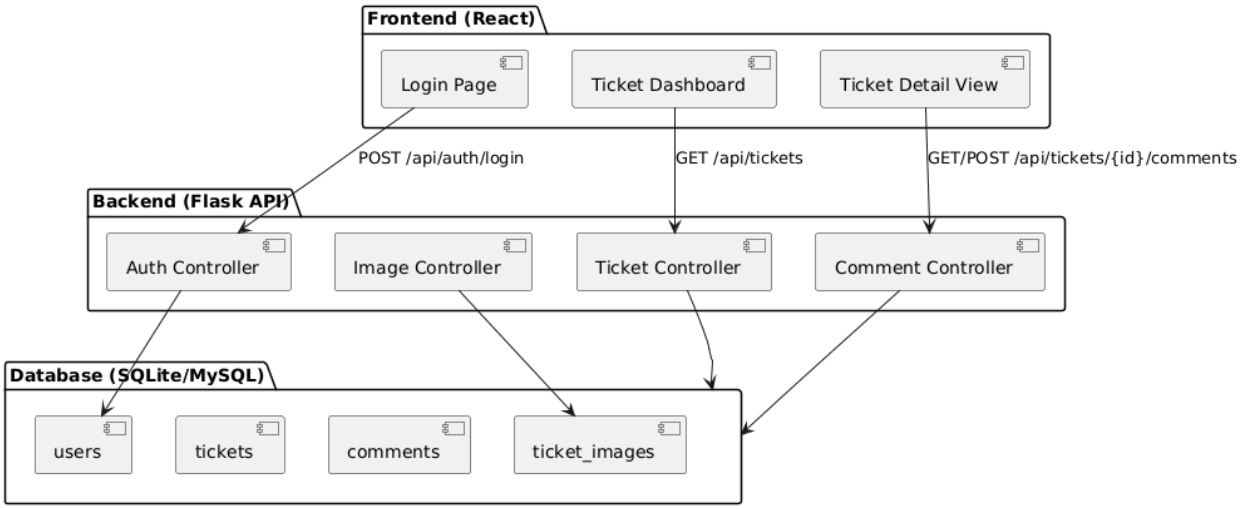


Design Documentation

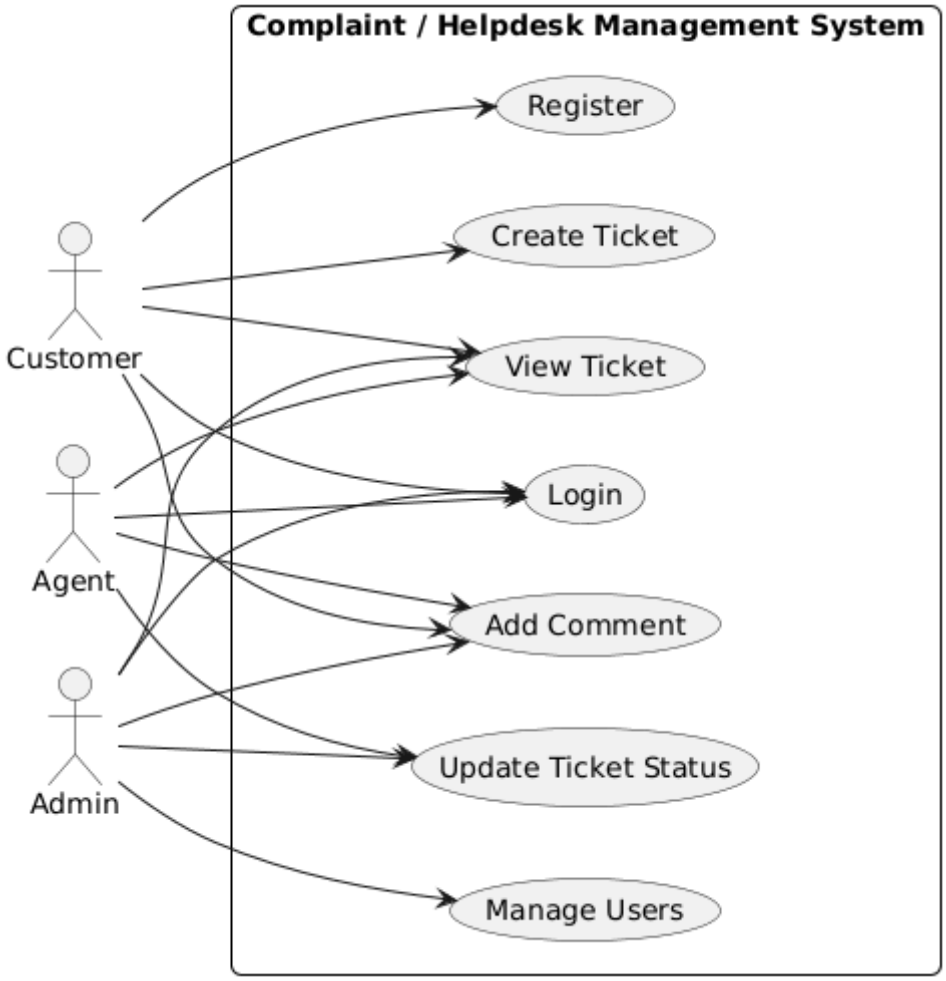
Class



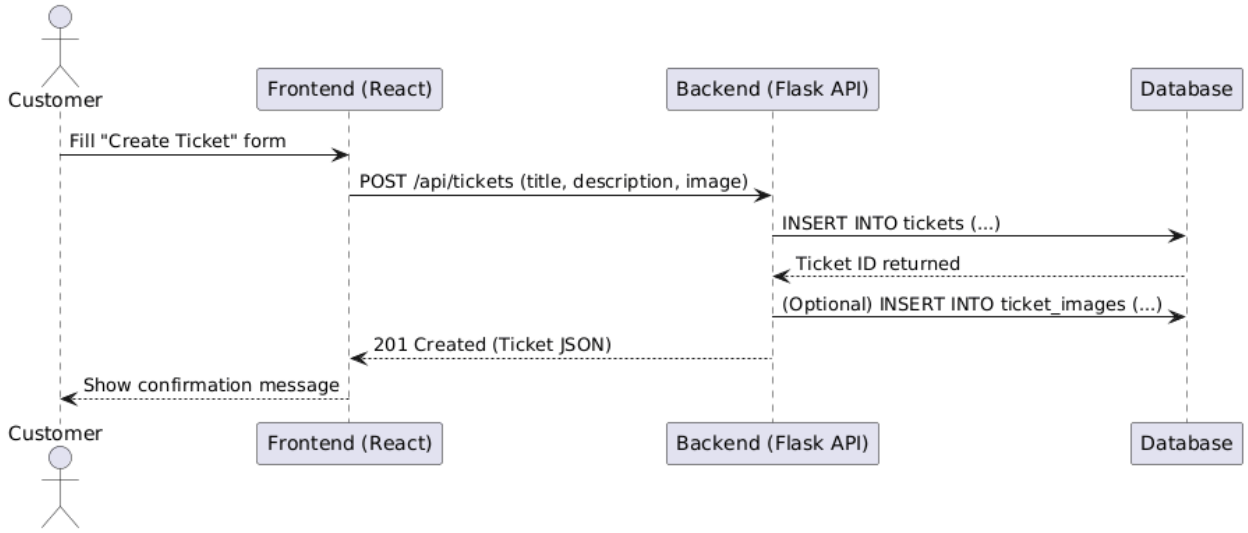
Component

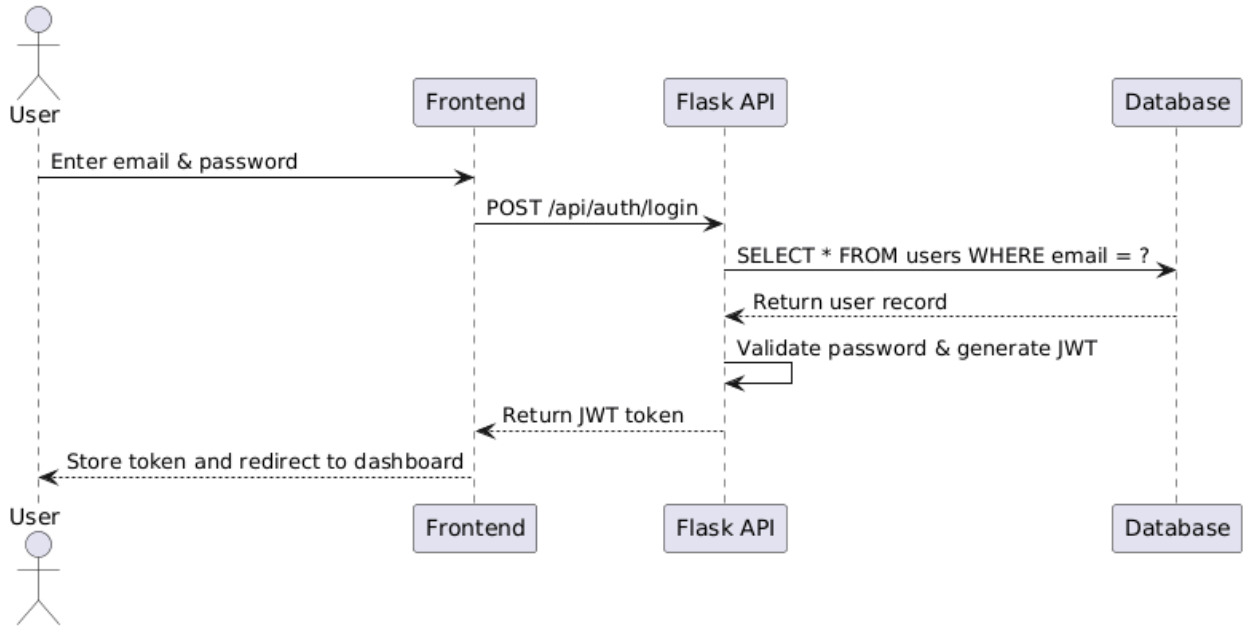


Use Case



Sequence


Customer
Frontend (React)
Backend (Flask API)
Database
Fill "Create Ticket" form
POST /api/tickets (title, description, image)
INSERT INTO tickets (...)
Ticket ID returned
(Optional) INSERT INTO ticket_images (...)
201 Created (Ticket JSON)
Show confirmation message
Customer
Frontend (React)
Backend (Flask API)
Database


User
Frontend
Flask API
Database
Enter email & password
POST /api/auth/login
SELECT * FROM users WHERE email = ?
Return user record
Validate password & generate JWT
Return JWT token
Store token and redirect to dashboard
User
Frontend
Flask API
Database

Implementation Report for Complaint / Helpdesk Management System

1. Tech Stack

This project utilizes a modern full-stack architecture, separating the backend API from the frontend user interface.

1.1 Backend

- **Framework:** Flask (Python)
- **Database ORM:** Flask-SQLAlchemy
- **Database Migrations:** Flask-Migrate
- **Authentication:** PyJWT (JSON Web Tokens)
- **Environment Variables:** python-dotenv
- **Password Hashing:** bcrypt

1.2 Frontend

- **Framework:** React (JavaScript)
- **Build Tool:** Vite
- **Routing:** React Router
- **Styling:** Standard CSS

2. Architecture

The application follows a client-server architecture:

- **Frontend (Client):** A React single-page application (SPA) that consumes the backend API. It handles user interactions, data presentation, and routing.
- **Backend (Server):** A Flask API that provides RESTful endpoints for user authentication, ticket management, and comment management. It interacts with the database and handles business logic.

Communication between the frontend and backend occurs via HTTP requests, with data exchanged in JSON format. JWTs are used for secure authentication and authorization.

3. Key Module Descriptions

3.1 Backend Modules

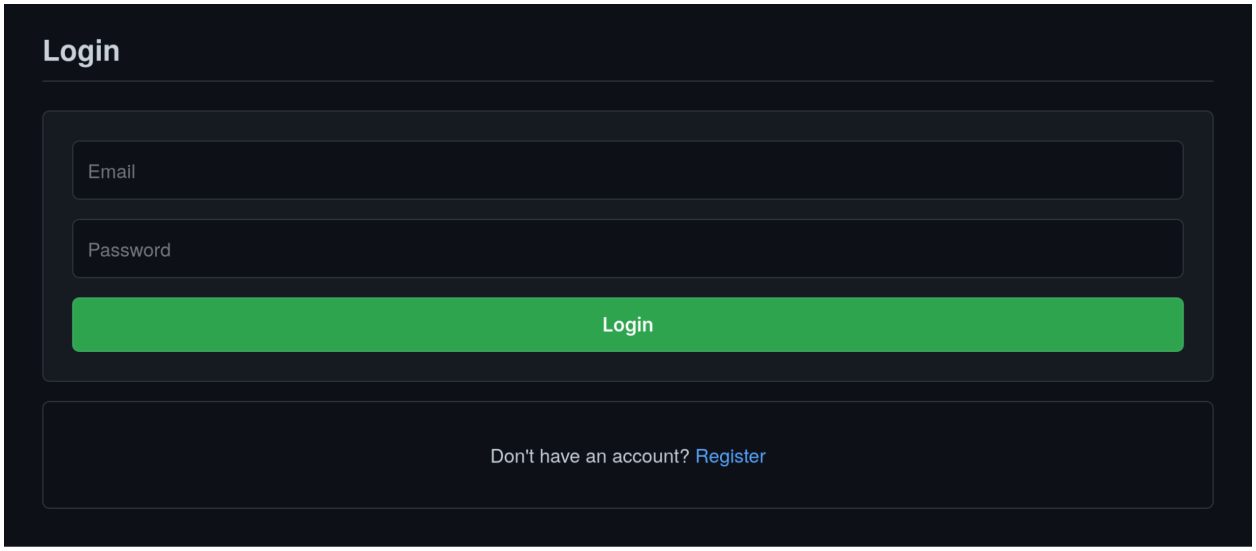
- **`app.py`** The main Flask application file, responsible for setting up the Flask app, registering blueprints, and defining API routes.
- **`models.py`:** Defines the SQLAlchemy ORM models for **`Users`**, **`Tickets`**, and **`Comments`** tables, representing the database schema.
- **`setup_db.py`:** Script for initializing the database and applying migrations.
- **`insert_dummy_data.py`:** Script to populate the database with sample data for testing and development.
- **`openapi.yml`:** OpenAPI specification for the backend API endpoints.

3.2 Frontend Modules

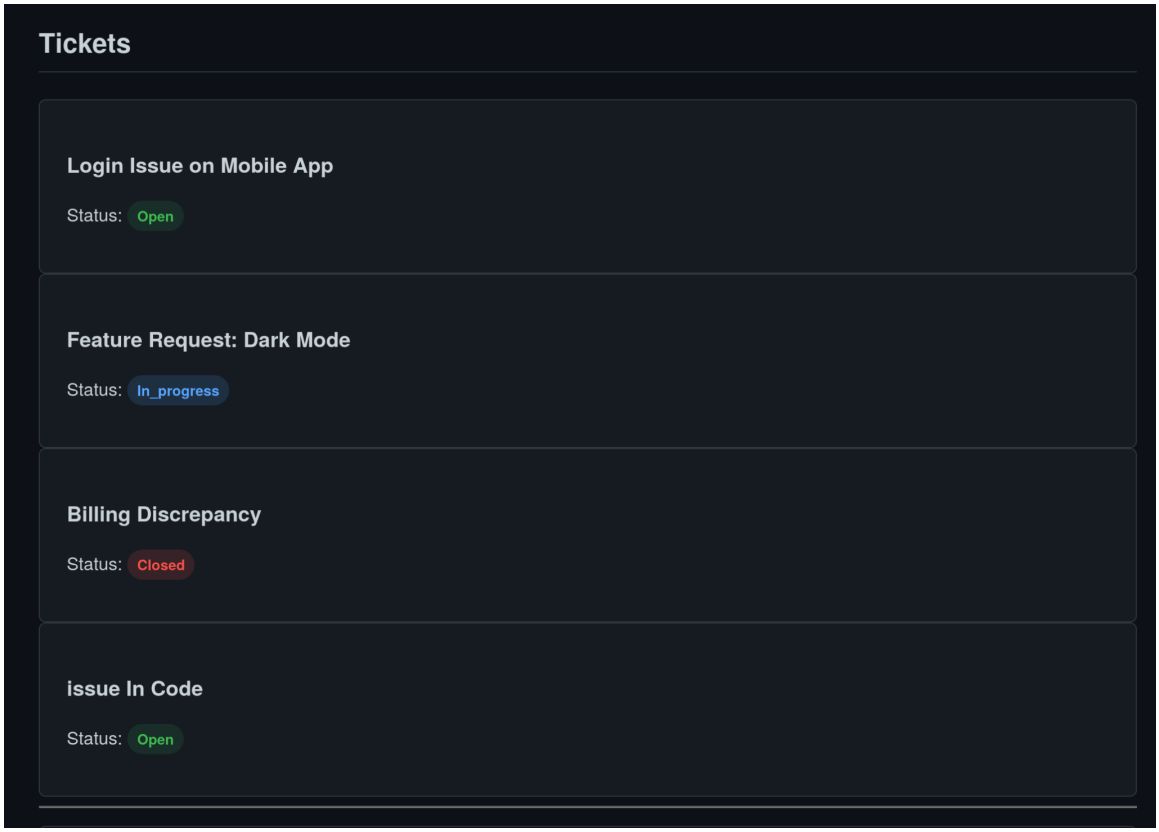
- **`main.jsx`:** The entry point of the React application, responsible for rendering the root component.
- **`router.jsx`:** Defines the application's routing using React Router, mapping URLs to specific views.
- **`api.jsx`:** Contains functions for interacting with the backend API, abstracting HTTP requests.
- **`components/Header.jsx`:** Reusable React component for the application's header/navigation.
- **`views/` (e.g., `Home.jsx`, `Login.jsx`, `Tickets.jsx`, `TicketDetail.jsx`):** React components representing different pages or views of the application.

4. Screenshots of Core Modules

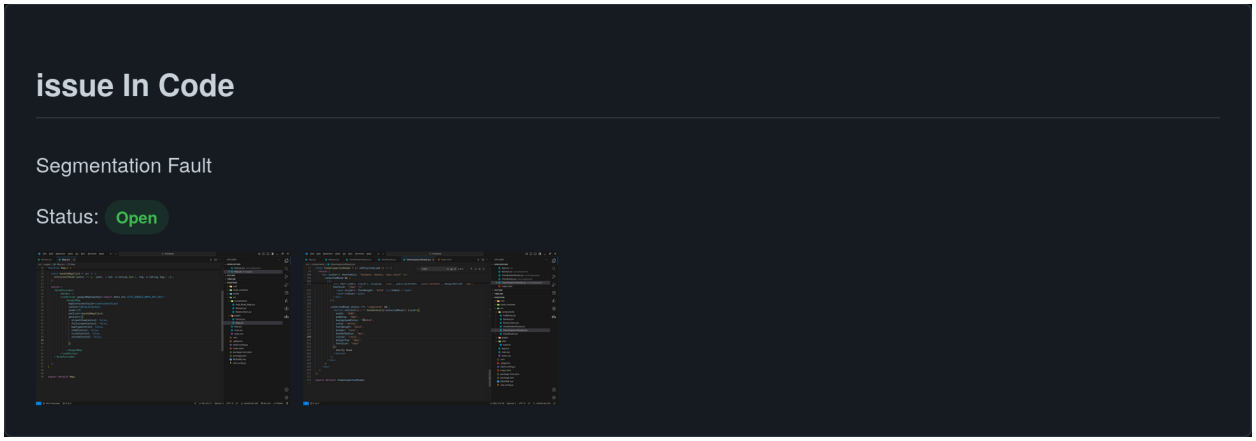
4.1 Login Page



4.2 Tickets Dashboard



4.3 Ticket Detail View



5. Database Schema

Users Table

Column	Data Type	Constraints	Description
id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique user ID
username	TEXT	NOT NULL, UNIQUE	Username of the user
email	TEXT	NOT NULL, UNIQUE	User’s email address
password	TEXT	NOT NULL	Hashed user password
role	TEXT	NOT NULL, CHECK(role IN ('customer', 'agent', 'admin'))	Defines the user’s role

Tickets Table

Column	Data Type	Constraints	Description
id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique ticket ID
title	TEXT	NOT NULL	Title of the ticket
description	TEXT	NOT NULL	Description of the issue
status	TEXT	NOT NULL, CHECK(status IN ('open', 'in_progress', 'closed'))	Current status of the ticket

author_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES users(id)	The user who created the ticket
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp when ticket was created

Comments Table

Column	Data Type	Constraints	Description
id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique comment ID
text	TEXT	NOT NULL	Comment text content
author_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES users(id)	User who posted the comment
ticket_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES tickets(id)	Ticket the comment belongs to
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp when comment was created

Ticket_images Table

Column	Data Type	Constraints	Description
id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique image ID
ticket_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES tickets(id)	Associated ticket ID
image_data	BLOB	NOT NULL	Binary image data (stored as BLOB)

User Manual

1. Introduction

This manual provides a guide for end-users on how to set up and use the Complaint / Helpdesk Management System. It covers the steps to get the application running, how to use its core features, and discusses potential future enhancements.

2. Execution Steps

To run the application, you need to start both the backend and frontend servers.

2.1 Prerequisites

- Python 3.x
- Node.js and npm
- A cloned copy of the project repository.

2.2 Running the Backend Server

Open a terminal and navigate to the **`backend`** directory of the project.
`cd path/to/Complaint / Helpdesk-management/backend.`

1. Activate the Python virtual environment.
   `source .venv/bin/activate`
2. Start the Flask server.
   `flask run`
3. The backend API will now be running at **`http://127.0.0.1:5000`**. Keep this terminal window open.

2.3 Running the Frontend Application

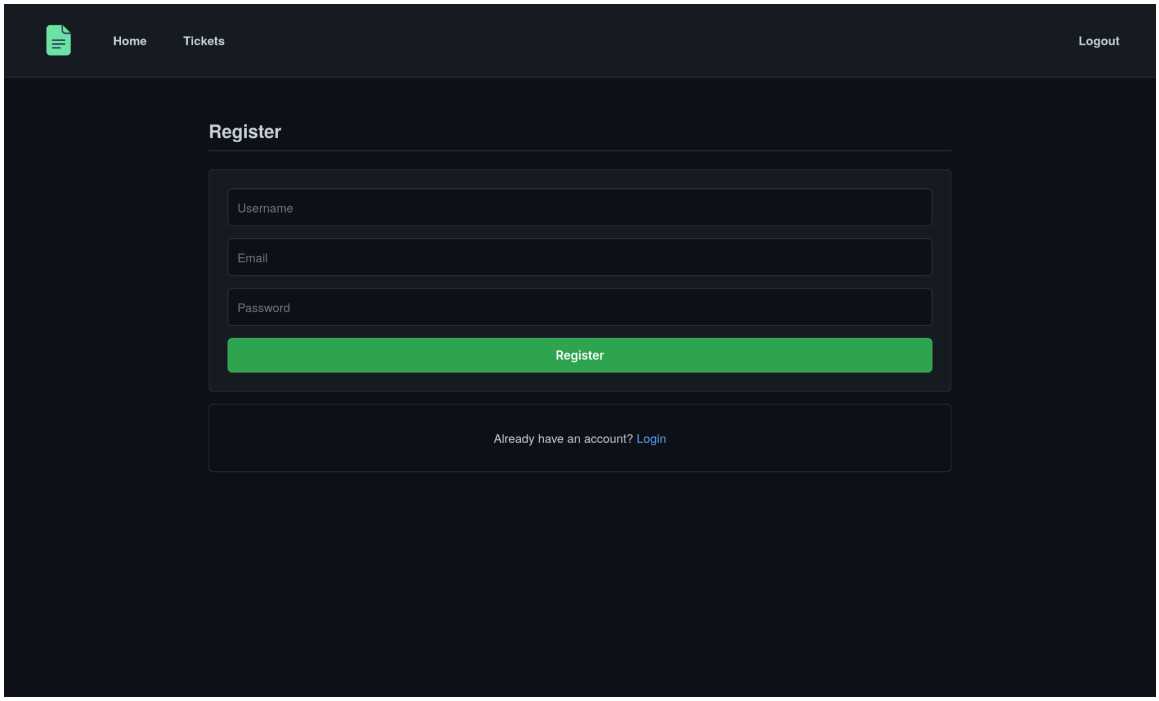
Open a **new** terminal window and navigate to the **`frontend`** directory.
`cd path/to/Complaint / Helpdesk-management/frontend`

1. Install the necessary Node.js packages (only required for the first time).
   `npm install`
2. Start the frontend development server.
3. **`npm run dev`**
4. The application will be accessible in your web browser at **`http://localhost:5173`**.

3. Using the Application: A Step-by-Step Guide

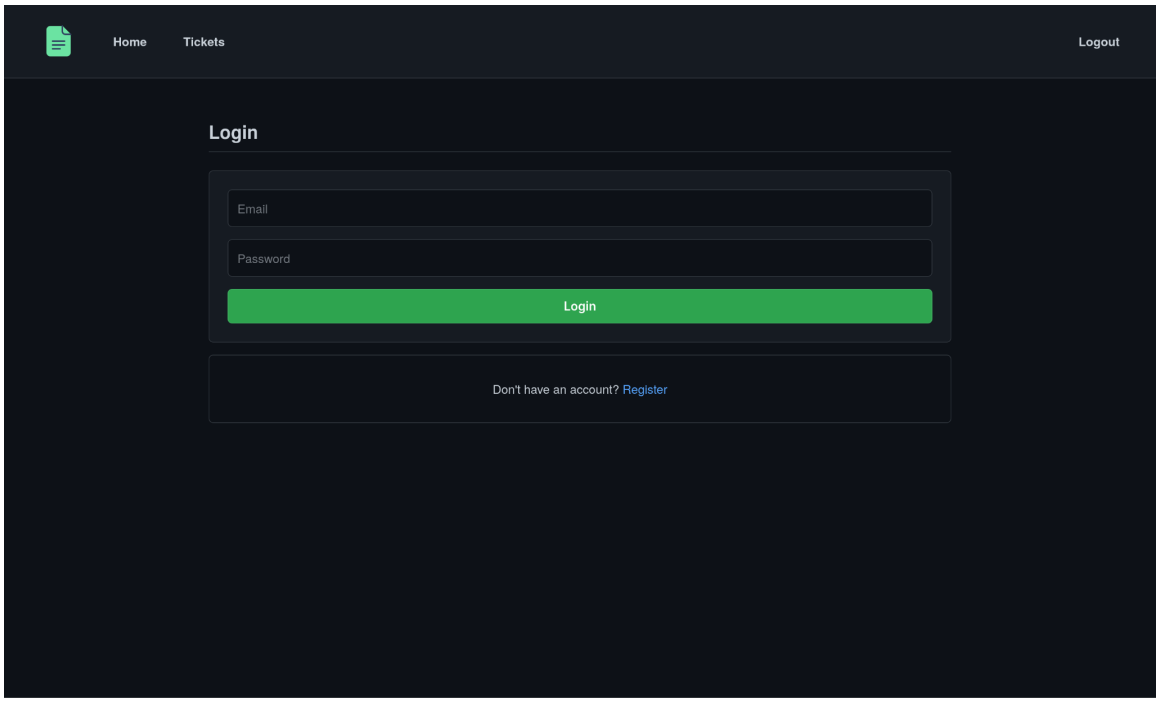
3.1 Registration (**`http://localhost:5173`**).

1. Navigate to the application URL Click on the "Register" link in the navigation bar.
2. Fill in the registration form with your desired username, email, and password.
3. Click the "Register" button to create your account.



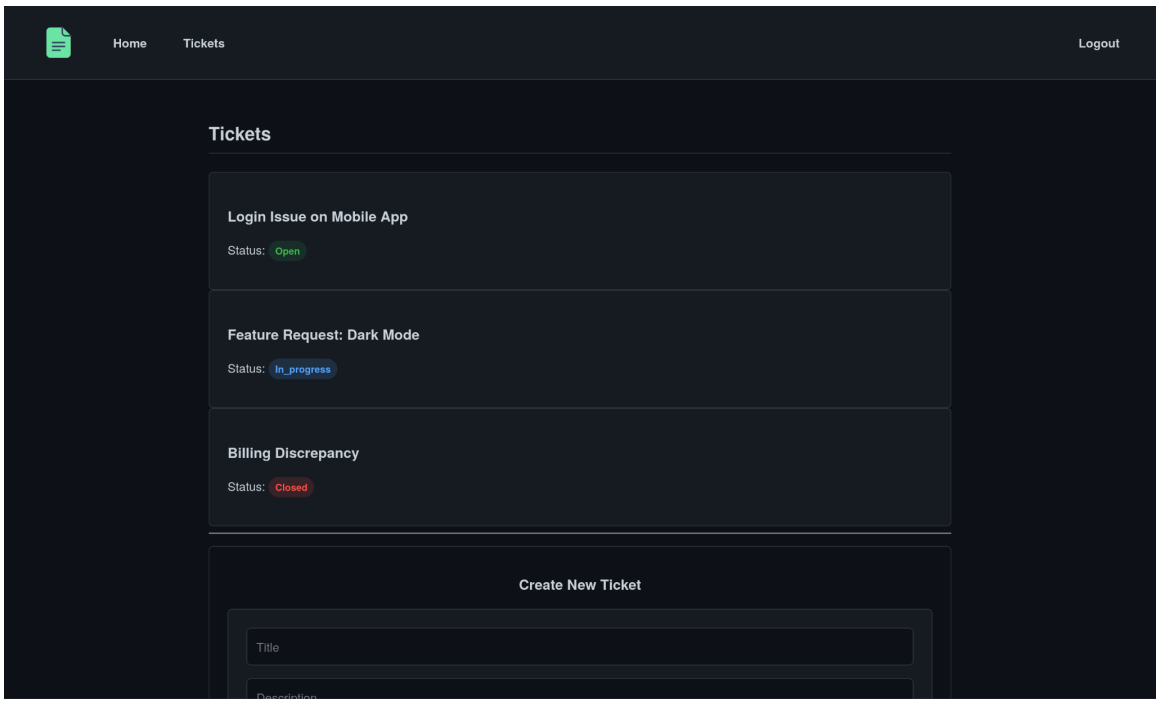
3.2 Login

1. After registering, or if you already have an account, go to the "Login" page.
2. Enter your username and password.
3. Click the "Login" button to access your dashboard.



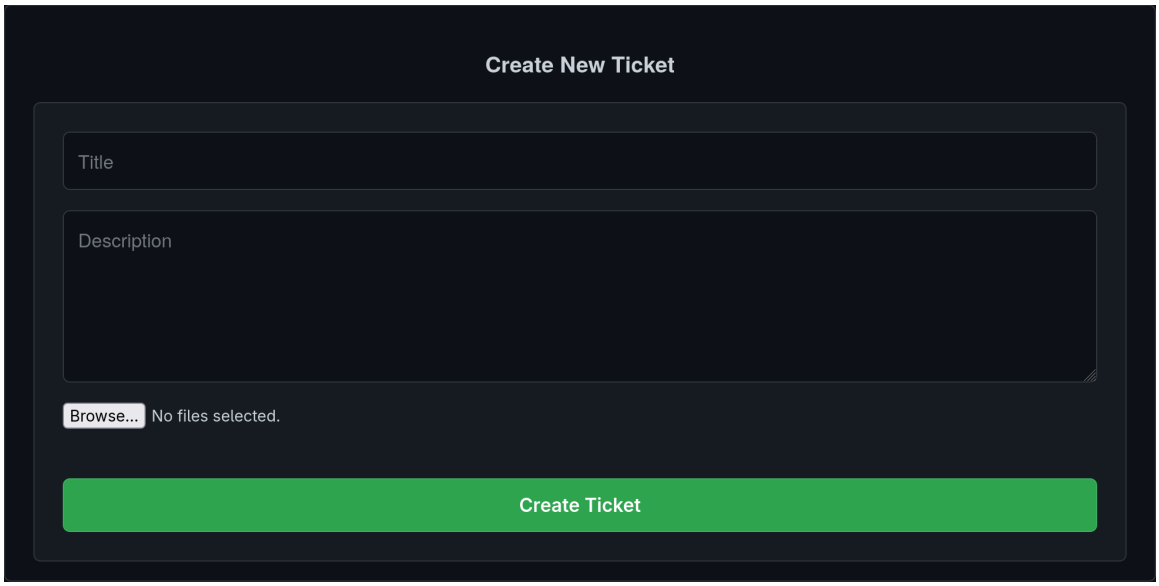
3.3 Viewing the Tickets Dashboard

1. Upon logging in, you will be redirected to the tickets dashboard.
2. This page displays a list of tickets.
    - **Customers** will see a list of tickets they have created.
    - **Agents and Admins** will see all tickets in the system.



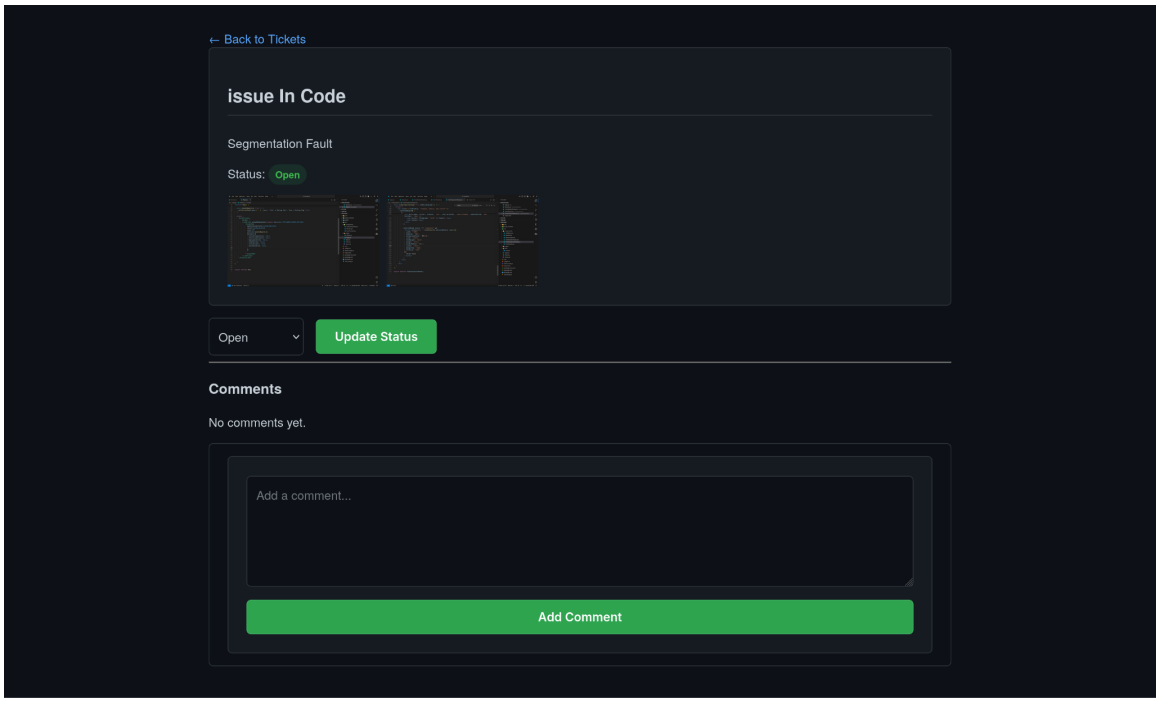
3.4 Creating a New Ticket

1. From the dashboard, click on the "Create New Ticket" button.
2. You will be taken to a form where you can enter the ticket's title and a detailed description of the issue.
3. Click "Submit" to create the ticket. You will be redirected back to your dashboard where you can see the newly created ticket.



3.5 Viewing a Ticket and Adding Comments

1. From the dashboard, click on the title of any ticket to view its details.
2. The ticket detail page shows the full description, status, and a history of comments.
3. To add a new comment, type your message in the comment box at the bottom and click "Add Comment".



4. Future Scope

The Complaint / Helpdesk Management System is a foundational application with significant potential for expansion. Future enhancements could include:

- **Enhanced User Roles and Permissions:** More granular permissions for different user roles (e.g., team leads, department managers).
- **Email Notifications:** Automatic email notifications to users when a ticket is created, updated, or commented on.
- **Ticket Assignment:** Functionality for admins or agents to assign specific tickets to individual agents.
- **Reporting and Analytics:** A dashboard for admins to view key metrics, such as ticket resolution times, agent performance, and common issue categories.
- **Full-Text Search:** A powerful search functionality to quickly find tickets based on keywords.
- **Knowledge Base Integration:** A section where users can find answers to frequently asked questions, potentially reducing the number of new tickets.