

# 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

(Screenshots will be added here to visually demonstrate the core functionalities of the application, such as the login page, tickets dashboard, and ticket detail view.)

### 4.1 Login & Registration Page

Login

Email

Password

Login

Don't have an account? [Register](#)

Register

Username

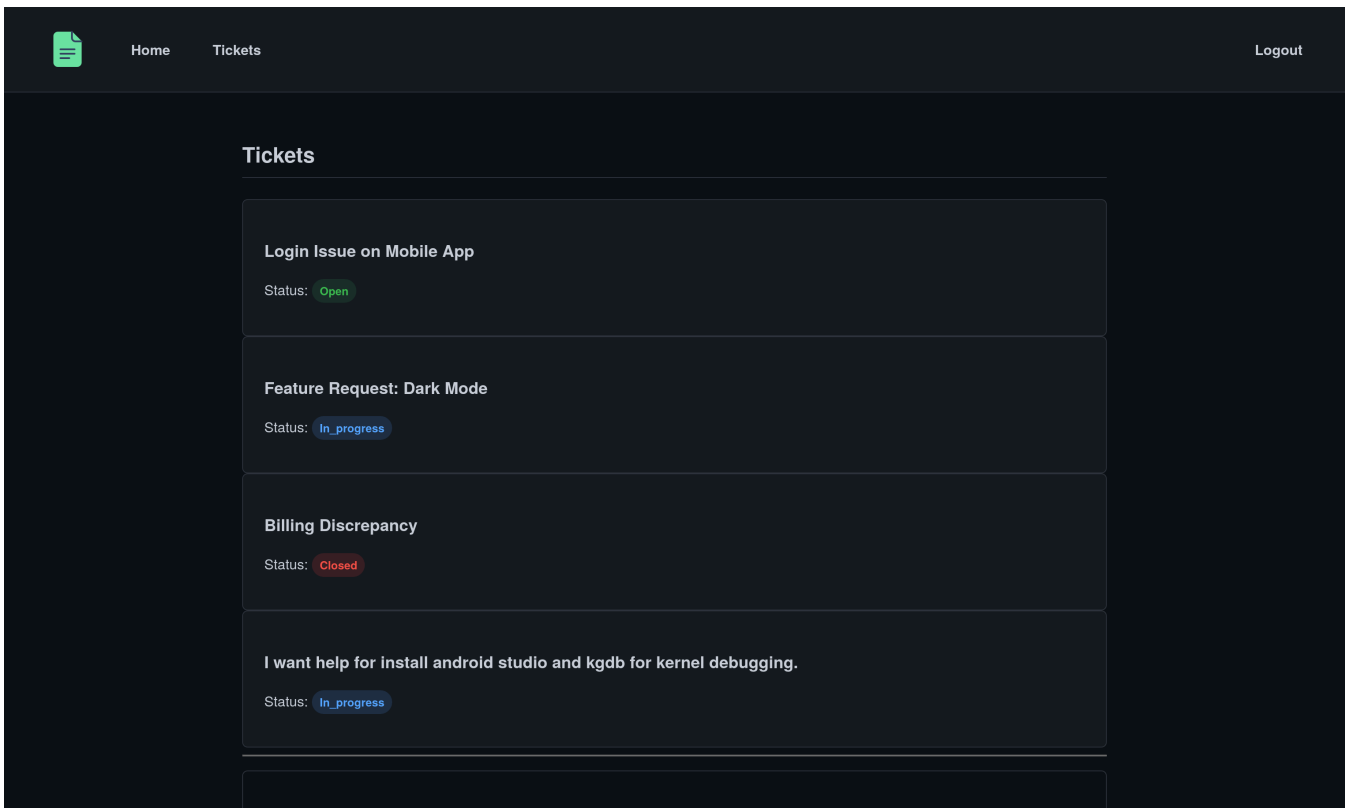
Email

Password

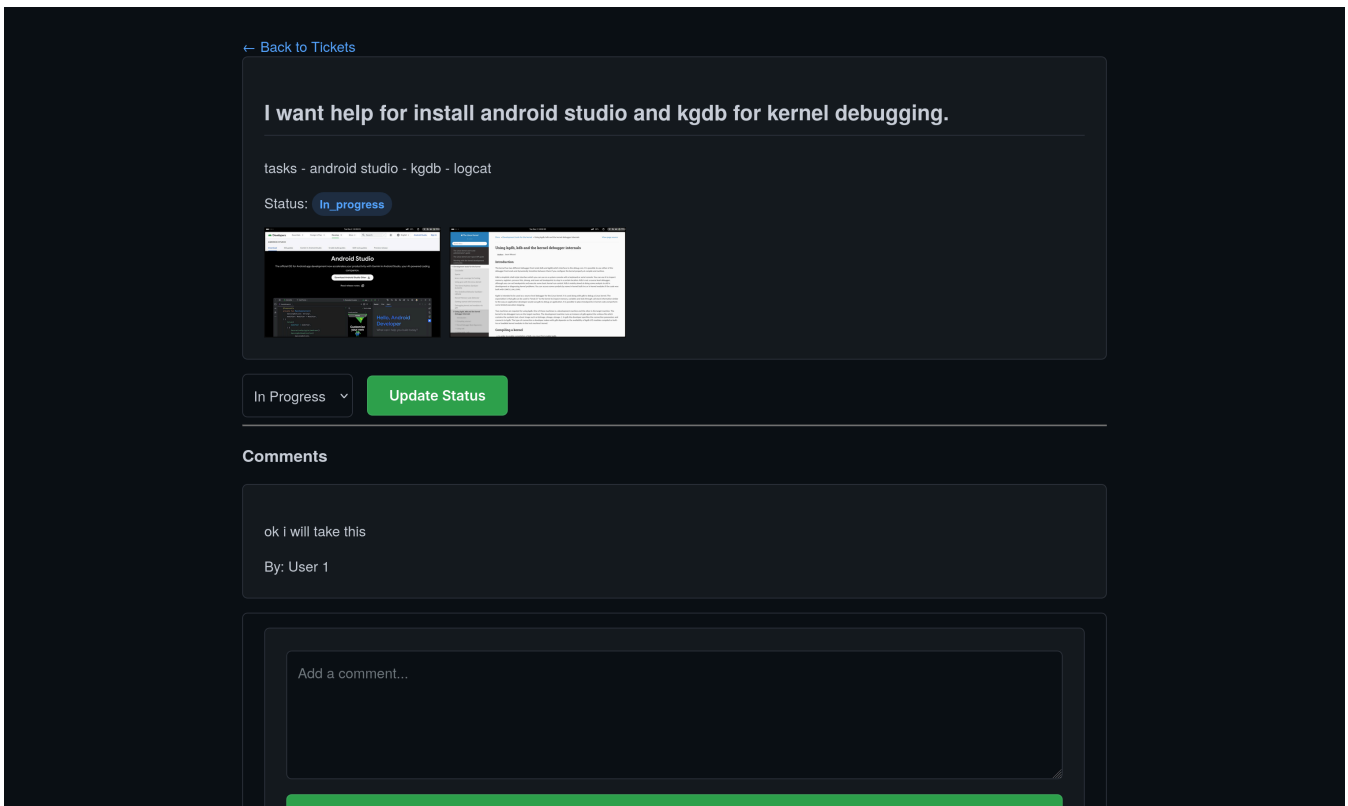
Register

Already have an account? [Login](#)

### 4.2 Tickets Dashboard



### 4.3 Ticket Detail View



## 5. Database Schema

### Users Table

```
CREATE TABLE users (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  username TEXT NOT NULL UNIQUE,  
  email TEXT NOT NULL UNIQUE,  
  password TEXT NOT NULL,
```

```
    role TEXT NOT NULL CHECK(role IN ('customer', 'agent', 'admin'))
);
```

## Tickets Table

```
CREATE TABLE tickets (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    description TEXT NOT NULL,
    status TEXT NOT NULL CHECK(status IN ('open', 'in_progress', 'closed')),
    author_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (author_id) REFERENCES users (id)
);
```

## Comments Table

```
CREATE TABLE comments (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    text TEXT NOT NULL,
    author_id INTEGER NOT NULL,
    ticket_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (author_id) REFERENCES users (id),
    FOREIGN KEY (ticket_id) REFERENCES tickets (id)
);
```

## Images Table

```
CREATE TABLE ticket_images (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    ticket_id INTEGER NOT NULL,
    image_data BLOB NOT NULL,
    FOREIGN KEY (ticket_id) REFERENCES tickets (id)
);
```