# Inventory and POS System - Developer Documentation

**Overview**

Welcome to the developer documentation for Inventory and POS management system. This documentation is designed to guide developers through the software's architecture, codebase, and implementation processes, ensuring a smooth and efficient development experience.

**System Architecture**

**Technology Stack**

* **Frontend: HTML, CSS, JavaScript, Bootstrap 4**
* **Backend: Python (Django)**
* **Database: MySQL**
* **Authentication: JSON Web Tokens (JWT)**
* **API Documentation: Swagger**
* **Deployment:**

**High-Level Architecture**

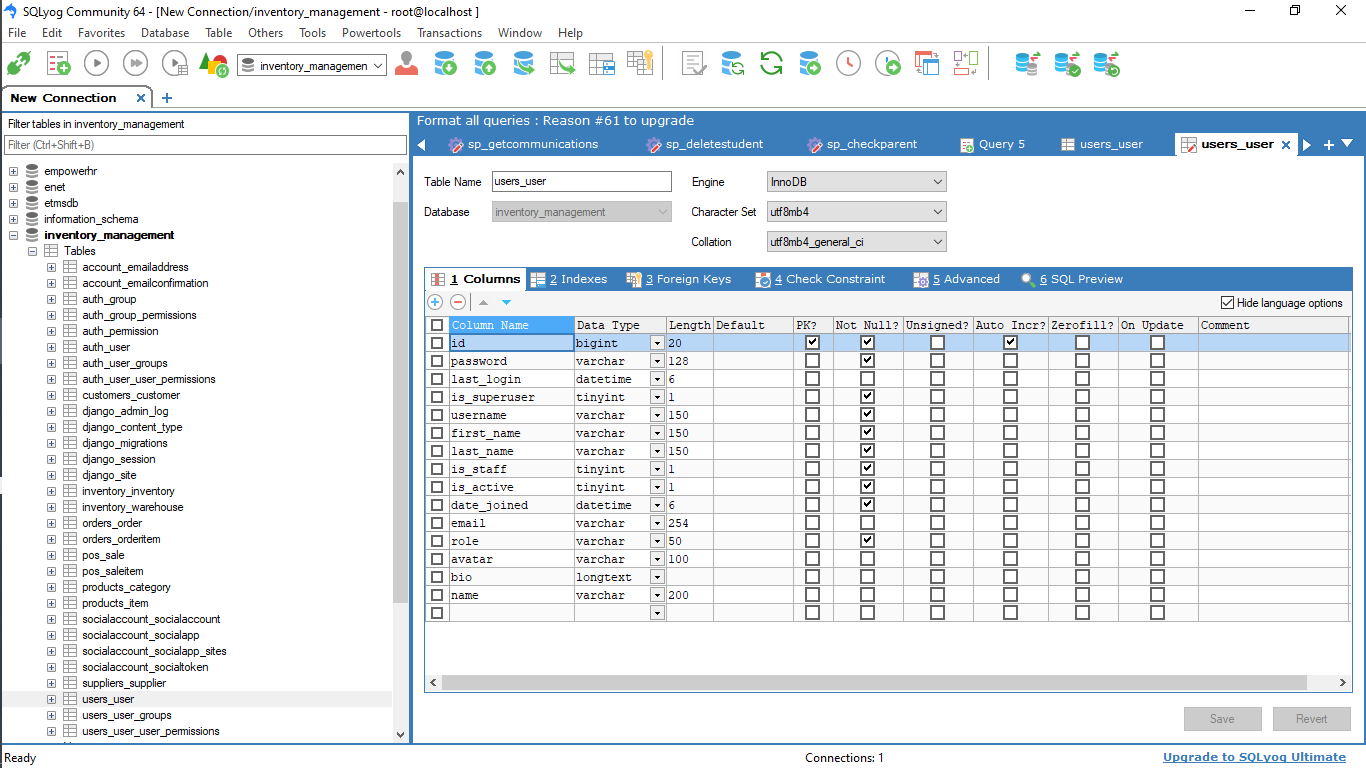
**The system is designed with a modular architecture, separating the frontend, backend, and database layers.**

* **Frontend: The frontend communicates with the Django backend via RESTful APIs.**
* **Backend: The Django backend handles API requests, processes business logic, and interacts with the MySQL database.**
* **Database: MySQL is used for persistent storage of inventory and sales data. The database is normalized to reduce redundancy.**

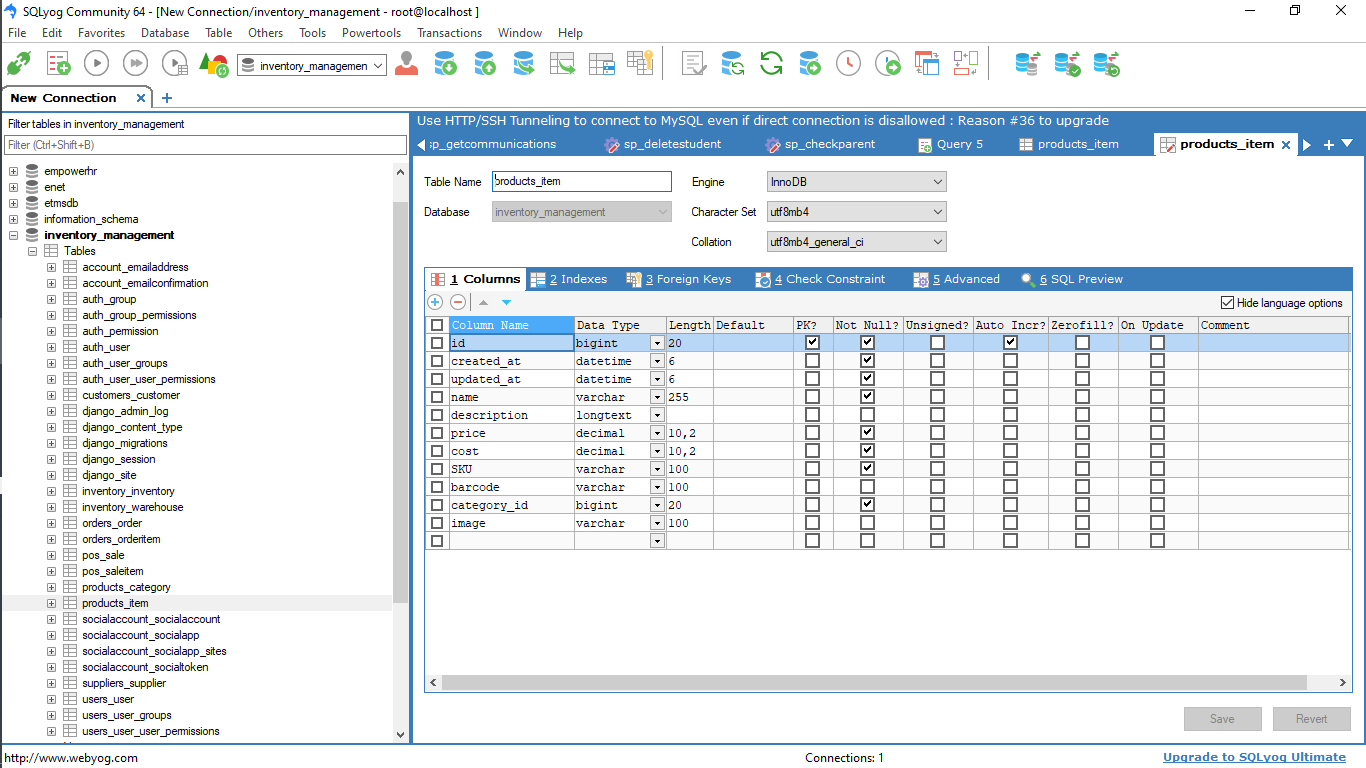
**Database Design**

**The database consists of the following key tables:**

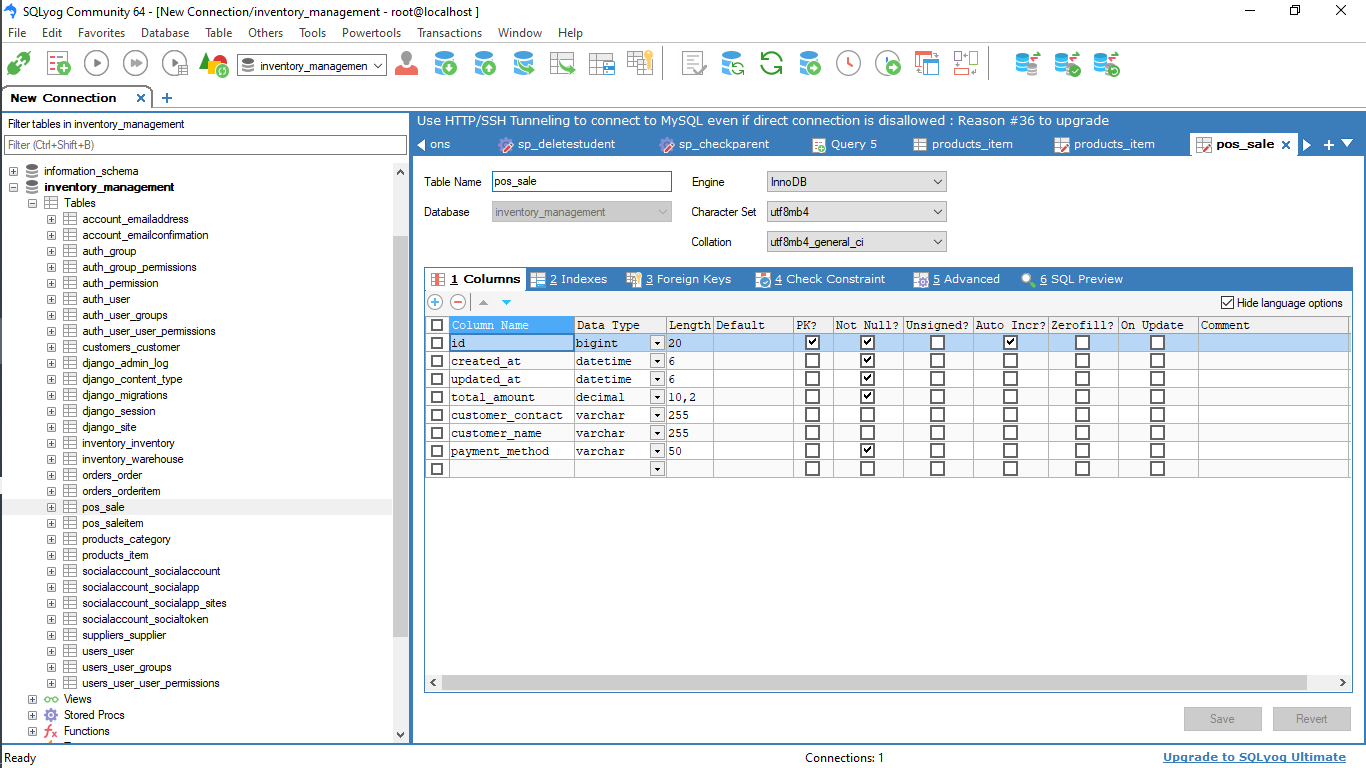
* **Users: Stores user information, including hashed passwords and roles.**



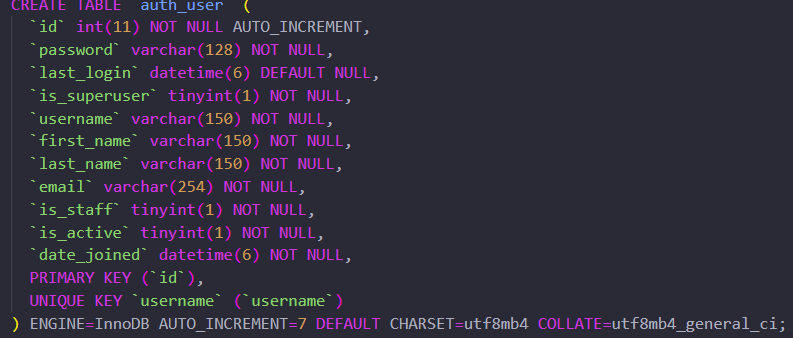
* **Products: Contains details of the products in the inventory.**

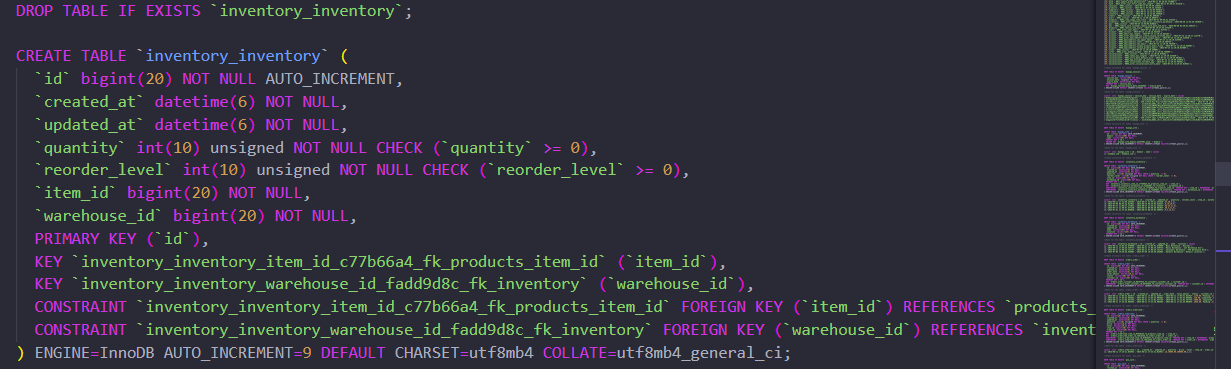


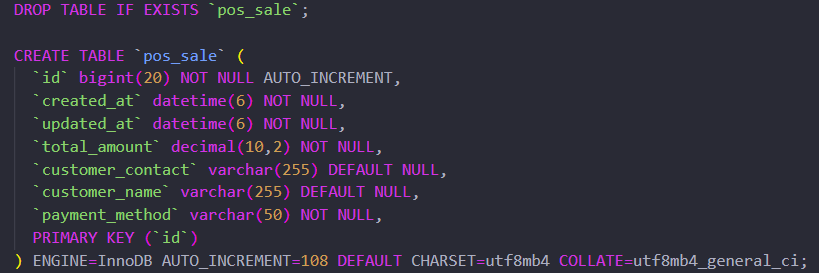
* **Sales: Records individual sales transactions.**

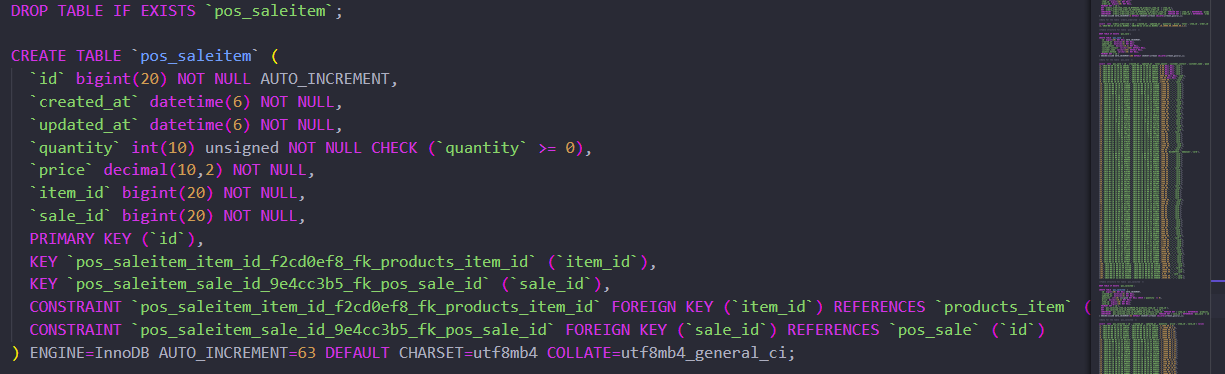


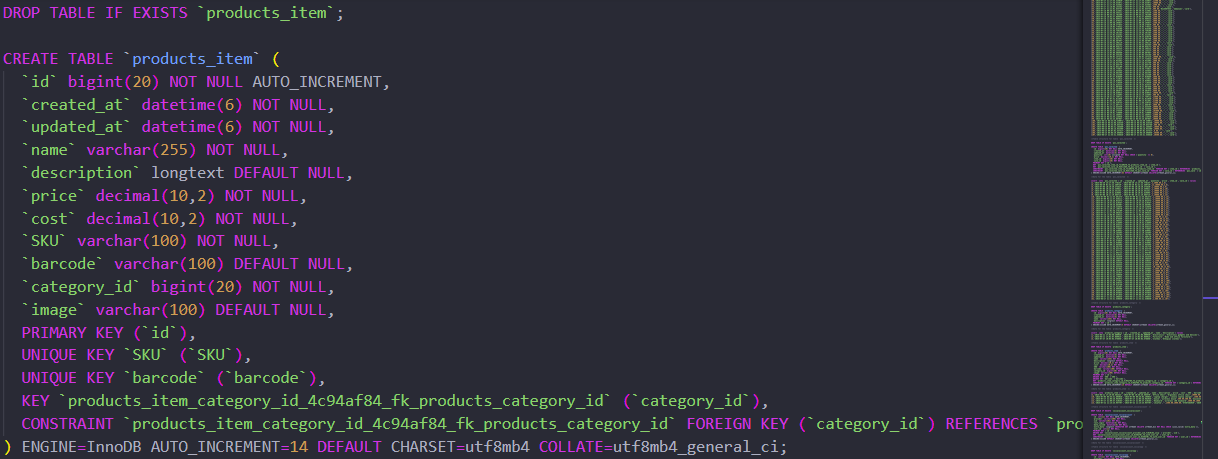
**Sample Schema:**











**Setup and Installation**

**Prerequisites**

* **Python: v3.8 or higher**
* **Django: v5.1 or higher**
* **MySQL: v8.x or higher**
* **Docker: Optional, for containerized deployment**

**Local Development Environment**

1. **Clone the Repository:**

**git clone https://github.com/emmanuelokoth2002/inventory\_management.git**

1. **Create and Activate a Virtual Environment:**

**mkdir virtualenv**

**virtialenv/Scripts/activate**

1. **Install Dependencies:**

**pip install -r requirements.txt**

1. **Configure Environment Variables:**

**Create a .env file in the root directory and add the following:**

**makefile**

1. **Run Migrations:**

**python manage.py makemigrations**

**python manage.py migrate**

1. **Create a Superuser:**

**python manage.py createsuperuser “name”**

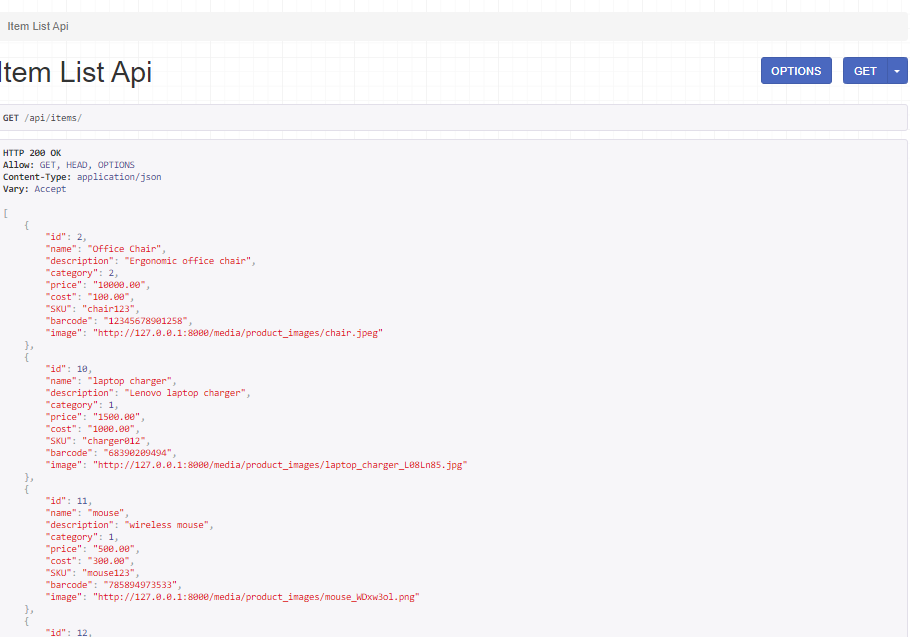
1. **Start the Development Server:**

**python manage.py runserver**

**Production Environment**

* **Docker Deployment: Use the provided Dockerfile and docker-compose.yml to set up the production environment.**
* **Nginx Configuration: Configure Nginx as a reverse proxy to manage requests to the backend.**

**API Documentation**



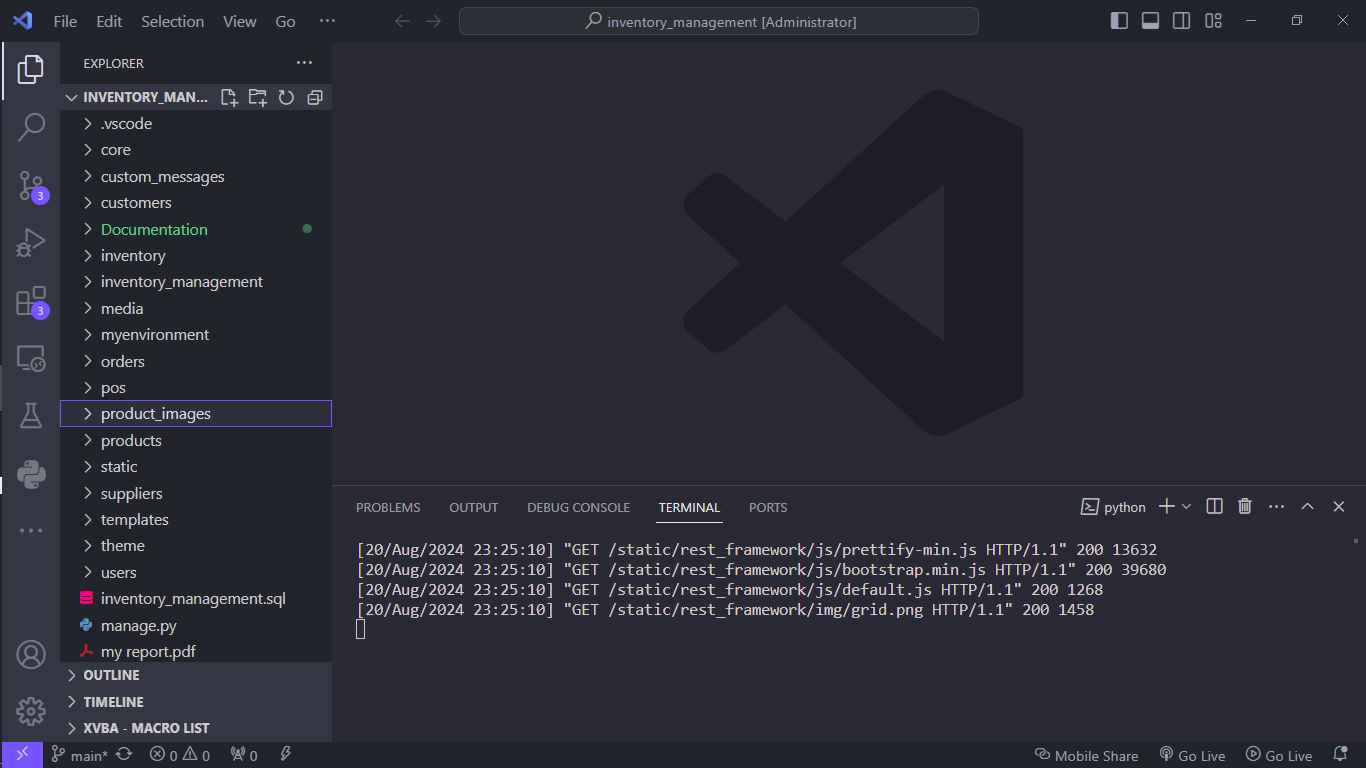
**Frontend Development**

**UI/UX Framework**

* **The system uses Bootstrap 4 for responsive design and styling. Custom styles are added in the assets/css/custom.css file.**

**Component Structure**

* **Django Templates:**
  + **base.html: Main template that includes the header, footer, and common elements.**
  + **dashboard.html: Displays the overview of sales, inventory levels, and alerts.**
  + **product\_list.html: Manages the display and CRUD operations for products.**
  + **pos.html: Handles the Point of Sale functionality.**
  + **inventory.html: Manages stock levels and reorder points.**



**State Management**

* **Django Context Processors: Used to pass common data to all templates.**
* **JavaScript: Custom JavaScript files manage dynamic elements and interactions within the frontend.**

**Backend Development**

**Business Logic**

* **The business logic is implemented in the services/ directory. Each service corresponds to a major component of the system (e.g., ProductService, SaleService).**

**Error Handling**

* **Centralized error handling is implemented using Django's middleware. Custom error classes are defined in utils/exceptions.py.**

**Logging**

* **Logging is handled using Django’s built-in logging framework. Logs are stored in the /logs/ directory and are rotated weekly.**

**Testing**

**Unit Testing**

* **Framework: Django’s built-in test framework**
* **Location: Unit tests are located in the tests/unit/ directory.**
* **Running Tests:**

**bash**

**Copy code**

**python manage.py test**

**Integration Testing**

* **Integration tests are located in the tests/integration/ directory.**
* **Use Django’s test client to test API endpoints.**

**End-to-End Testing**

* **Tool: Cypress (for frontend testing)**
* **Location: E2E tests are located in the cypress/integration/ directory.**
* **Running Tests:**

**bash**

**Copy code**

**npm run cypress:open**

**Deployment and CI/CD**

**Deployment Process**

* **Staging Environment: Push to the staging branch to trigger deployment to the staging server.**
* **Production Environment: Merge to the main branch to trigger deployment to the production server.**

**CI/CD Pipeline**

* **Tool: GitHub Actions**
* **Pipeline: The pipeline is defined in .github/workflows/deploy.yml.**
* **Stages:**
  + **Build: Installs dependencies and runs tests.**
  + **Deploy: Deploys the application using Docker to the target environment.**

**Security Considerations**

**Data Encryption**

* **All sensitive data, including passwords, are encrypted using Django’s PBKDF2PasswordHasher before storage.**
* **TLS/SSL is enforced for all network communication.**

**User Authentication and Authorization**

* **JWT tokens are used for user authentication.**
* **Role-based access control (RBAC) is implemented using Django’s built-in Groups and Permissions system to restrict access to certain features.**

**Version Control and Branching Strategy**

* **Main Branches:**
  + **main: Production-ready code.**
* **Pull Requests: All changes must be submitted via pull requests, which require code reviews before merging.**

**Troubleshooting and Debugging**

**Common Issues:**

* **Database Connection Errors: Ensure environment variables are correctly set and the MySQL server is running.**
* **API Errors: Review logs in the /logs/ directory for detailed error messages.**

**Debugging:**

* **Use Django’s DEBUG setting in settings.py to enable verbose logging and detailed error pages during development.**

**Contributing to the Project**

**Contribution Guidelines:**

* **Follow the guidelines outlined in CONTRIBUTING.md.**
* **Use consistent code formatting and adhere to PEP 8 standards.**
* **Write meaningful commit messages using the conventional commit format.**

**Appendix**

