Online Drinking Store

1. Introduction

This documentation covers a web-based beverage e-commerce platform called 'Online Drinking Store'. The system enables users to securely register, browse a wide range of beverages, place orders, and manage their proimages. It integrates Stripe for payments, React.js for the frontend, Node.js with Express for the backend, and MySQL for the database.

2. System Overview

2.1 Technology Stack

- Frontend\*\*: HTML5, Tailwind CSS, JavaScript

- Backend\*\*: Python Flask

- Database\*\*: MySQL

- Authentication\*\*: JWT (JSON Web Tokens)

2.2 Key Features

- User registration and authentication

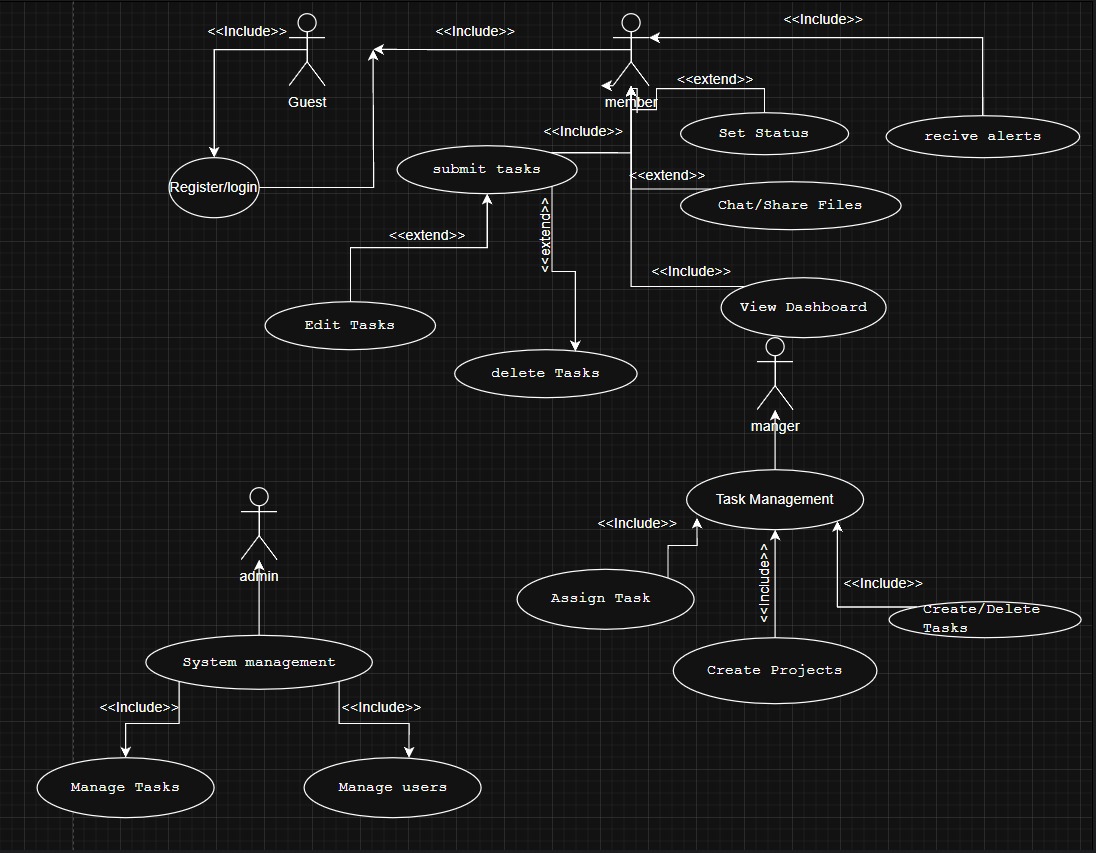
- Order creation and management

- Responsive design

- Secure session management

- RESTful API endpoints

3. Use Case Diagram



Use Case Descriptions

Actor Definitions

1. **Guest**
   * Unauthenticated user
   * Can register or log in to the system
2. Customer
   * Regular authenticated user
   * Can perform basic order operations
   * Can collaborate with admin staff customers
3. Admin
   * Privileged user with additional permissions
   * Can manage admin staff assignments and product categorys
   * Has administrative capabilities
4. **Admin** (implied from class diagram)
   * System administrator
   * Manages user accounts and system settings
   * Has full access to all features

Use Case Descriptions

Authentication Use Cases

1. **Register/Login** (Guest)
   * **Description**: Allows users to create accounts or authenticate
   * **Related Class**: User
   * **Preconditions**: None
   * **Postconditions**: User gains appropriate access level
2. View User Dashboard (Customer/Admin)
   * Description: Displays personalized order overview
   * Related Classes: User, Order, Product Category
   * **Preconditions**: Successful authentication
   * Postconditions: User Dashboard rendered with user-specific data

Order Management Use Cases

1. Submit Orders (Customer)
   * Description: Create new order items
   * Related Classes: Order, User
   * Preconditions: Customer logged in
   * Postconditions: New order appears in system
2. Delete Orders (Customer/Admin)
   * Description: Remove orders from system
   * Related Classes: Order, Notification
   * **Preconditions**: User has appropriate permissions
   * Postconditions: Order removed and notifications sent if assigned
3. Set Order Status (Customer/Admin)
   * Description: Update order progress state
   * Related Classes: Order, Notification
   * Preconditions: Order exists and is assigned to user
   * Postconditions: Order status updated, notifications triggered

Admin-Specific Use Cases

1. Assign Orders (Admin)
   * Description: Delegate orders to admin staff customers
   * Related Classes: Order, User, Notification
   * Preconditions: Admin logged in, order exists
   * Postconditions: Order assigned and notification sent
2. Create Product Categorys (Admin)
   * Description: Establish new product categories
   * Related Classes: Product Category, User
   * Preconditions: Admin logged in
   * Postconditions: New product category available for order organization

Collaboration Use Cases

1. Customer Support/Feedback (Customer/Admin)
   * **Description**: Team communication and file exchange
   * Related Classes: SupportMessage, ProductImage, User
   * **Preconditions**: User authenticated
   * Postconditions: SupportMessages/images stored in system

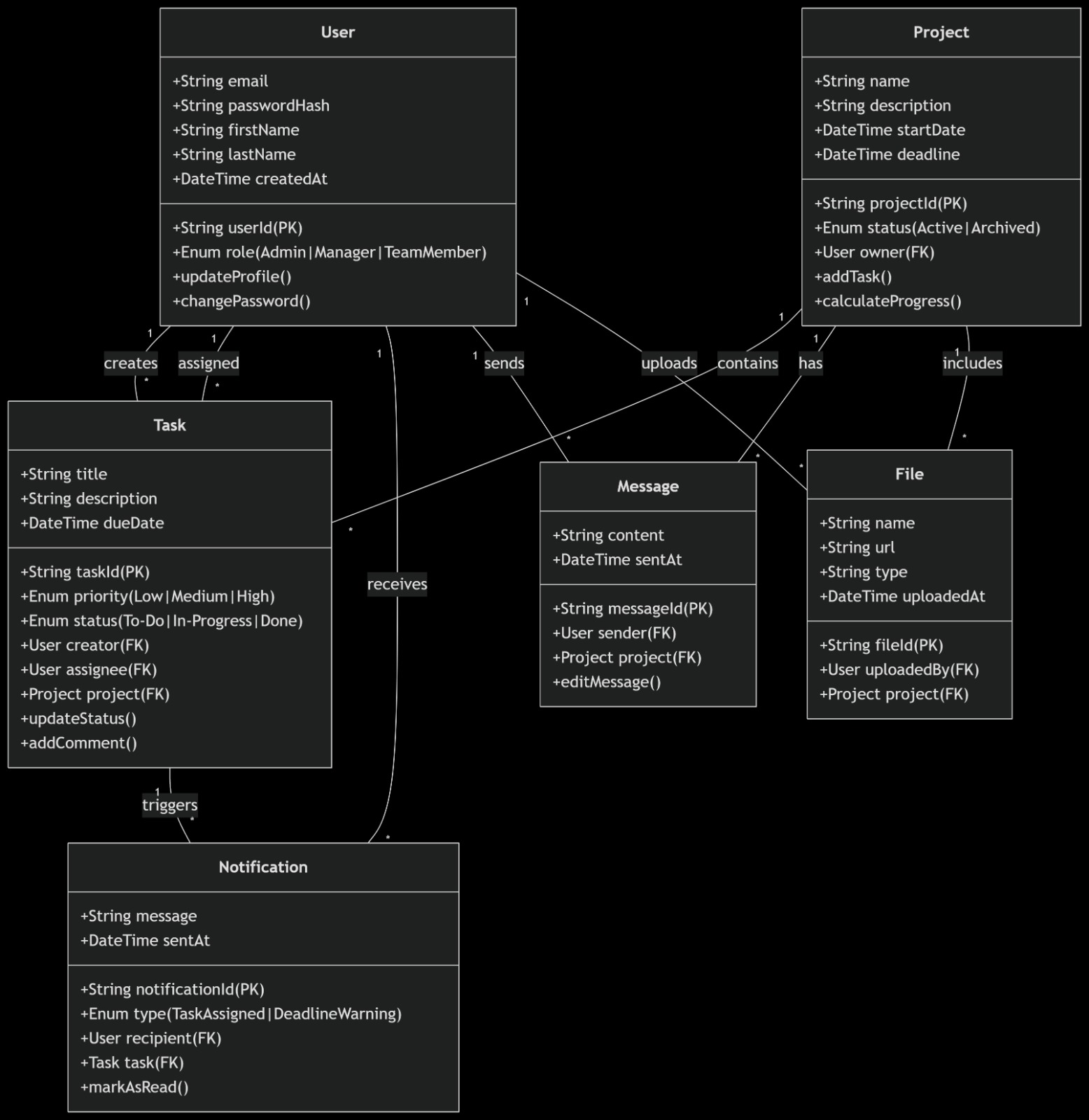
Admin Use Cases (implied)

1. **System Management** (Admin)
   * **Description**: Configure system settings and users
   * **Related Classes**: User
   * **Preconditions**: Admin privileges
   * **Postconditions**: System configuration updated

Relationships

1. **Inclusion Relationships** (<<include>>):
   * "Register/login" includes authentication verification
   * "Manage Orders" includes status updates and assignments
   * "Order Management" includes all order-related operations
2. **Extension Relationships** (<<extend>>):
   * "Delete Orders" extends from basic order management
   * "Set Status" extends from order operations
   * "System management" extends from user management

4. Class Diagram



1. User Class

**Attributes:**

* userId: Primary key (String)
* email: Unique user identifier (String)
* passwordHash: Securely stored password (String)
* firstName: User's first name (String)
* lastName: User's last name (String)
* createdAt: Account creation timestamp (DateTime)
* role: User role (Enum: Admin|Admin|TeamCustomer)

**Methods:**

* updateProfile(): Updates user information
* changePassword(): Changes user password

**Relationships:**

* Creates Orders (1-to-many)
* Receives Notifications (1-to-many)
* Owns Product Categorys (1-to-many)
* Uploads ProductImages (1-to-many)
* Sends SupportMessages (1-to-many)

2. Order Class

**Attributes:**

* orderId: Primary key (String)
* title: Order name (String)
* description: Order details (String)
* dueDate: Completion deadline (DateTime)
* priority: Importance level (Enum: Low|Medium|High)
* status: Completion state (Enum: To-Do|In-Progress|Done)

**Methods:**

* updateStatus(): Changes order status
* addComment(): Adds comments to order

**Relationships:**

* Belongs to Product Category (many-to-1)
* Created by User (many-to-1)
* Assigned to User (many-to-1)
* Triggers Notifications (1-to-many)

3. Notification Class

**Attributes:**

* notificationId: Primary key (String)
* message: Notification content (String)
* sentAt: Creation timestamp (DateTime)
* type: Notification category (Enum: OrderAssigned|DeadlineWarning)

**Methods:**

* markAsRead(): Updates read status

**Relationships:**

* Sent to User (many-to-1)
* Related to Order (many-to-1)

4. Product Category Class

**Attributes:**

* product categoryId: Primary key (String)
* name: Product Category title (String)
* description: Product Category details (String)
* startDate: Initiation date (DateTime)
* deadline: Completion date (DateTime)
* status: Current state (Enum: Active|Archived)

**Methods:**

* addOrder(): Creates new orders
* calculateProgress(): Computes completion percentage

**Relationships:**

* Owned by User (many-to-1)
* Contains Orders (1-to-many)
* Contains ProductImages (1-to-many)
* Contains SupportMessages (1-to-many)

5. ProductImage Class

**Attributes:**

* fileId: Primary key (String)
* name: ProductImage name (String)
* url: Storage location (String)
* type: ProductImage format (String)
* uploadedAt: Creation timestamp (DateTime)

**Relationships:**

* Uploaded by User (many-to-1)
* Belongs to Product Category (many-to-1)

6. SupportMessage Class

**Attributes:**

* messageId: Primary key (String)
* content: SupportMessage text (String)
* sentAt: Creation timestamp (DateTime)

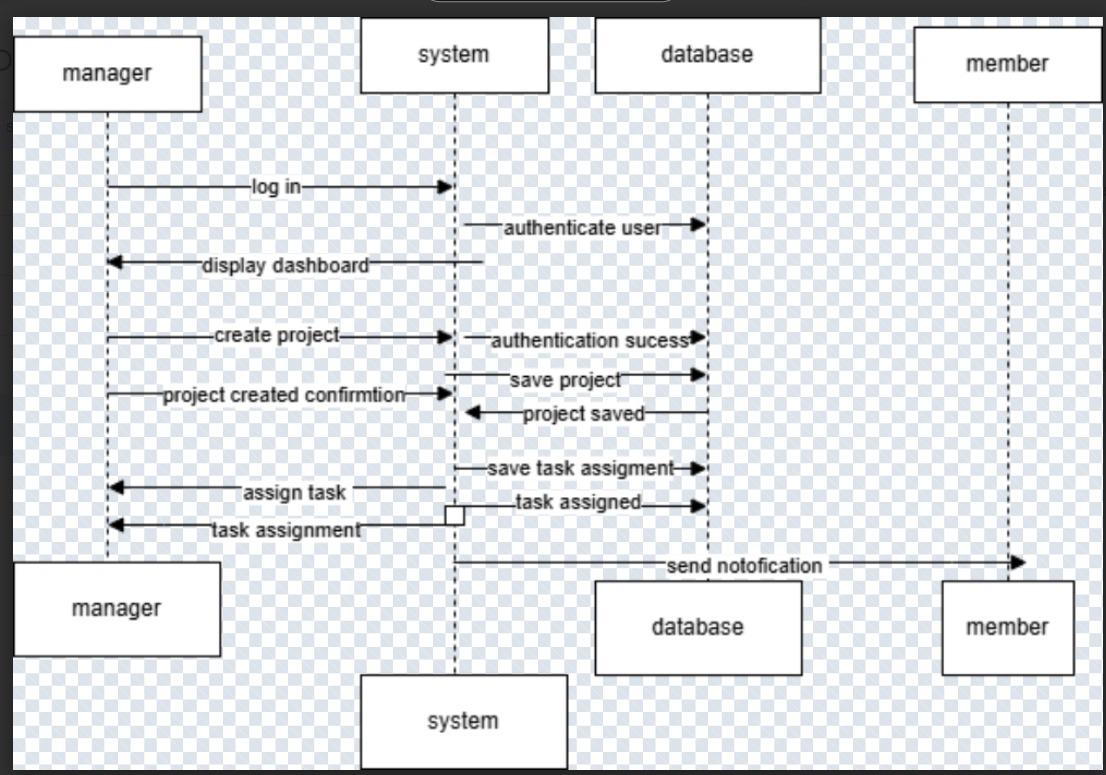
**Methods:**

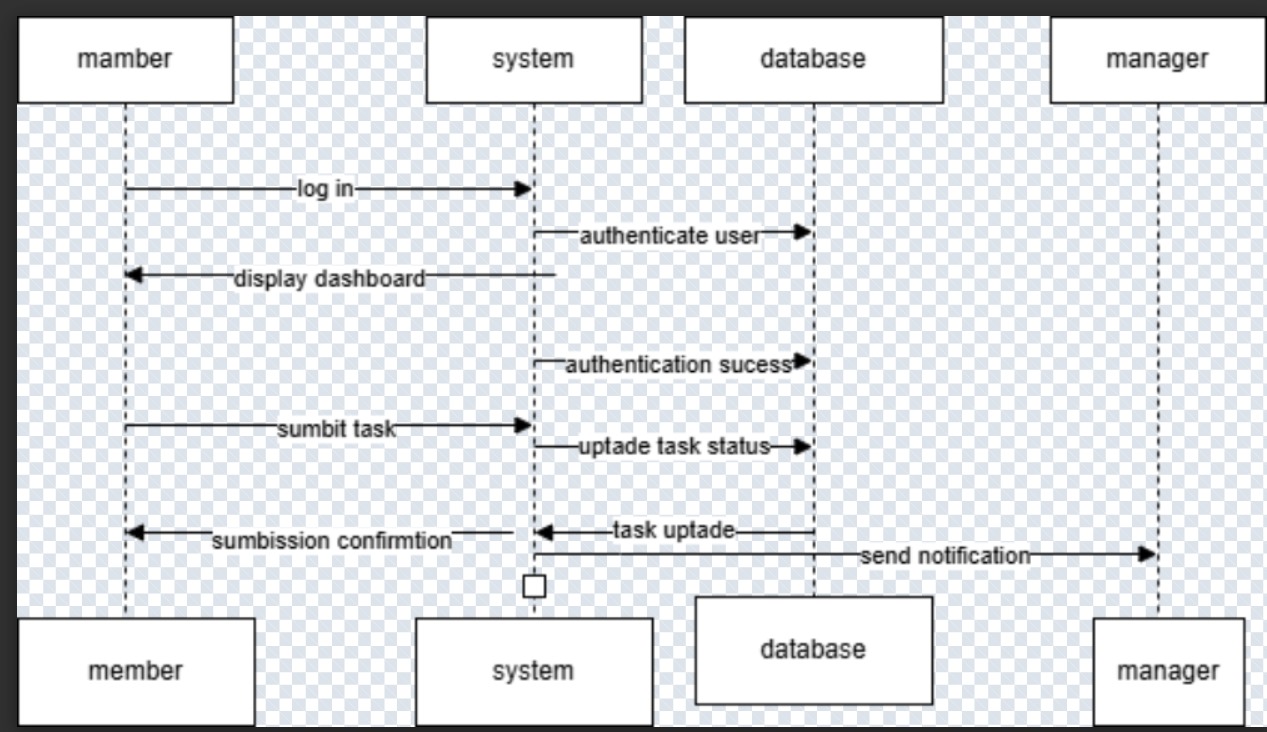
* editSupportMessage(): Modifies message content

**Relationships:**

* Sent by User (many-to-1)
* Belongs to Product Category (many-to-1)

5. Sequence Diagrams





5.1 User Registration

```plaintext

User -> Browser: Fills registration form

Browser -> Flask: POST /api/register (form data)

Flask -> Database: Create new user

Database -> Flask: User record

Flask -> Browser: 201 Created + JWT

Browser -> LocalStorage: Store token

Browser -> User: Redirect to dashboard

```

5.2 User Login

```plaintext

User -> Browser: Fills login form

Browser -> Flask: POST /api/login (credentials)

Flask -> Database: Validate credentials

Database -> Flask: User record

Flask -> Browser: 200 OK + JWT

Browser -> Storage: Store token (local/session)

Browser -> User: Redirect to dashboard

```

5.3 Order Management

6. Frontend Components

A screenshot of a contact form

AI-generated content may be incorrect.

A screenshot of a phone

AI-generated content may be incorrect.

1. Login Page

- Email/password input

- Recustomer me option

- Registration link

2. Registration Page

- Name, email, password fields

- Form validation

- Password requirements

7. Conclusion

This Online Drinking Store provides a secure, responsive web interface for personal order organization. The system follows modern web development practices with a clear separation between frontend and backend components, implemented with Flask and Tailwind CSS. The documentation covers all aspects of the system from use cases to technical implementation details.

**Chapter 6: Testing**

**6.1 Test Cases**

| **Test Case ID** | **Description** | **Expected Result** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- |
| TC01 | User registers with valid credentials | Redirect to dashboard with success message | As expected | ✅ Pass |
| TC02 | Login with incorrect password | Show error message, no login | As expected | ✅ Pass |
| TC03 | Create a new task with valid data | Task appears in dashboard | As expected | ✅ Pass |
| TC04 | Delete a task as a member | Task is removed from the system | As expected | ✅ Pass |
| TC05 (Negative) | Try submitting a task with blank title | Show form validation error | As expected | ✅ Pass |

**Chapter 7: Risk and Security Requirements**

**7.1 Risk Identification**

| **Risk ID** | **Category** | **Description** |
| --- | --- | --- |
| R1 | Security | Unauthorized access via token leakage |
| R2 | Performance | System slows down with 1000+ tasks |
| R3 | Usability | Users may forget to logout on shared devices |
| R4 | Data Integrity | Accidental deletion of tasks without undo |
| R5 | Hosting/Infra | Downtime due to server outage or database corruption |

**7.2 Risk Assessment**

| **Risk ID** | **Likelihood** | **Impact** | **Risk Level** |
| --- | --- | --- | --- |
| R1 | High | High | Critical |
| R2 | Medium | Medium | Moderate |
| R3 | Medium | Low | Low |
| R4 | Medium | High | High |
| R5 | Low | High | Moderate |

**7.3 Mitigation Techniques**

* **R1:** Implement token expiry and refresh mechanisms, secure HTTPS-only cookies.
* R2: Use order pagination or lazy loading.
* **R3:** Add auto-logout after inactivity.
* R4: Include a “Recycle Bin” or undo feature for deleted orders.
* **R5:** Use cloud-based hosting with automated daily backups.

**7.4 Requirement Refinement (SRS Updates)**

**Functional Requirements**

* FR9: System shall implement a "soft delete" mechanism for orders.
* FR10: System shall auto-logout users after 15 minutes of inactivity.

**Non-Functional Requirements**

* NFR5: System shall support up to 2000 concurrent orders with <500ms response time.
* NFR6: All JWT tokens must be encrypted and stored securely in session cookies.