**Technical Design Document for**

**WorkFlow**

**(Project Management Tool)**

**1. INTRODUCTION**

This document outlines the technical design for a web-based project management tool. It details the system's architecture, database design, services, technology stack, and user interface components.

**Title:** WorkFlow

**Description:**

The project management tool is a web-based application designed to streamline project workflows and enhance team collaboration. It provides a centralized platform for managing project-related information, tasks, communication, and progress. The tool caters to teams with distinct user roles, ensuring that each member has the appropriate level of access and functionality.

**Technology Stack**

**Frontend:**

* JavaScript Framework: Angular
* UI Library: Shadcn/ui
* Styling: Tailwind CSS

**Backend:**

* Language: Java
* Framework: SpringBoot
* ORM: Spring Data JPA

**Database:**

* Relational Database: MySQL

**Authentication & Authorization**

* JSON Web Tokens (JWT)
* Spring Security

**Testing:**

* Unit Testing: JUnit & Mockito

**Deployment:**

* Cloud Platform: AWS

**Containerization:**

* Docker

**Version Control:**

* Git & GitHub

**API Documentation**

* Swagger

**2. SYSTEM ARCHITECTURE**

The system will be designed using a multi-layered architecture to ensure scalability, maintainability, and security. The architecture will consist of the following layers:

* **Presentation Layer:** This layer will handle the user interface (UI) and user experience (UX) of the application. It will be responsible for displaying information to the user and capturing user input.
* **Application Layer (Services):** This layer will contain the business logic of the application. It will process user requests, interact with the data access layer, and enforce business rules.
* **Data Access Layer:** This layer will be responsible for interacting with the database. It will handle data retrieval, storage, and manipulation.
* **Database Layer:** This layer will store the application's data.

**3. DATABASE SCHEMA**

**Users Table**

Stores user authentication and profile information.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the user (e.g., UUID) |
| first\_name | VARCHAR(255) | NOT NULL | User's first name |
| last\_name | VARCHAR(255) | NOT NULL | User's last name |
| email | VARCHAR(255) | NOT NULL, UNIQUE | User's email address, used for login |
| password\_hash | VARCHAR(255) | NOT NULL | Hashed and salted password |
| role | VARCHAR(50) | NOT NULL, ENUM('Product Owner', 'Scrum Master', 'SDET Engineer') | User's role within the system |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of user creation |
| updated\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Timestamp of last update to user record |

**Projects Table**

Stores information about each project created by Product Owners.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the project |
| project\_name | VARCHAR(255) | NOT NULL | Name of the project |
| description | TEXT | NULLABLE | Detailed description of the project |
| product\_owner\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID of the Product Owner who created the project |
| unique\_project\_code | VARCHAR(50) | NOT NULL, UNIQUE | A unique code for Scrum Masters and SDET Engineers to join |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of project creation |
| updated\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Timestamp of last update to project record |

**Project\_members Table**

Links users (Scrum Masters, SDET Engineers) to specific projects they are assigned to.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the project member entry |
| project\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES projects(id) | ID of the project |
| user\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | ID of the user (Scrum Master or SDET Engineer) |
| joined\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp when the user joined the project |
| UNIQUE(project\_id, user\_id) |  | Ensures a user can only be a member of a project once |  |

**Epics Table**

Stores Epics, which are large bodies of work for a project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the epic |
| project\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES projects(id) | ID of the project the epic belongs to |
| title | VARCHAR(255) | NOT NULL | Title of the epic |
| description | TEXT | NULLABLE | Detailed description of the epic |
| created\_by\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID of the Product Owner who created the epic |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of epic creation |
| updated\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Timestamp of last update to epic record |

**User\_stories Table**

Stores User Stories, which are smaller units of work within an Epic or Project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the user story |
| project\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES projects(id) | ID of the project the user story belongs to |
| epic\_id | VARCHAR(36) | NULLABLE, FOREIGN KEY REFERENCES epics(id) | ID of the associated epic (if any) |
| title | VARCHAR(255) | NOT NULL | Title of the user story |
| description | TEXT | NULLABLE | Detailed description of the user story |
| acceptance\_criteria | TEXT | NULLABLE | Criteria for when the user story is considered done |
| priority | VARCHAR(50) | NOT NULL, ENUM('Low', 'Medium', 'High') | Priority level of the user story |
| created\_by\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID of the Product Owner who created the user story |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of user story creation |
| updated\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Timestamp of last update to user story record |

**Sprints Table**

Stores details about each sprint.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the sprint |
| project\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES projects(id) | ID of the project the sprint belongs to |
| scrum\_master\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID of the Scrum Master managing this sprint |
| sprint\_goal | TEXT | NULLABLE | Goal for the sprint |
| start\_date | DATE | NOT NULL | Start date of the sprint |
| end\_date | DATE | NOT NULL | End date of the sprint |
| status | VARCHAR(50) | NOT NULL, ENUM('Planning', 'Active', 'Completed', 'Canceled'), DEFAULT 'Planning' | Current status of the sprint |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of sprint creation |
| updated\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Timestamp of last update to sprint record |

**Tasks Table**

Stores individual tasks within user stories or sprints.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the task |
| user\_story\_id | VARCHAR(36) | NULLABLE, FOREIGN KEY REFERENCES user\_stories(id) | ID of the user story this task belongs to (if any) |
| sprint\_id | VARCHAR(36) | NULLABLE, FOREIGN KEY REFERENCES sprints(id) | ID of the sprint this task is part of (if any) |
| title | VARCHAR(255) | NOT NULL | Title of the task |
| description | TEXT | NULLABLE | Detailed description of the task |
| due\_date | DATETIME | NULLABLE | Deadline for the task |
| priority | VARCHAR(50) | NOT NULL, ENUM('Low', 'Medium', 'High') | Priority level of the task |
| assigned\_to\_id | VARCHAR(36) | NULLABLE, FOREIGN KEY REFERENCES users(id) | User ID (SDET Engineer) assigned to the task |
| created\_by\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID (Scrum Master/SDET) who created the task |
| status | VARCHAR(50) | NOT NULL, ENUM('To Do', 'In Progress', 'Done', 'Blocked'), DEFAULT 'To Do' | Current status of the task |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of task creation |
| updated\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Timestamp of last update to task record |

**Subtasks Table**

For checklists or smaller steps within a task.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the subtask |
| task\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES tasks(id) | ID of the parent task |
| description | TEXT | NOT NULL | Description of the subtask |
| is\_completed | BOOLEAN | NOT NULL, DEFAULT FALSE | Status indicating if the subtask is completed |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of subtask creation |
| updated\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Timestamp of last update to subtask record |

**File\_attachments**

Table Stores information about files attached to tasks.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the attachment |
| task\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES tasks(id) | ID of the task the file is attached to |
| file\_name | VARCHAR(255) | NOT NULL | Original name of the file |
| file\_url | VARCHAR(500) | NOT NULL | URL or path where the file is stored (e.g., cloud storage) |
| uploaded\_by\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID who uploaded the file |
| uploaded\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of file upload |

**Time\_tracking**

Table Records time spent by SDET Engineers on tasks.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the time entry |
| task\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES tasks(id) | ID of the task the time was spent on |
| user\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID (SDET Engineer) who tracked the time |
| start\_time | DATETIME | NOT NULL | Start timestamp of the work session |
| end\_time | DATETIME | NULLABLE | End timestamp of the work session (NULL if ongoing) |
| duration\_minutes | INT | NULLABLE | Duration in minutes (can be calculated or stored for performance) |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of time entry creation |

**Bug\_reports**

Table Records bugs reported by SDET Engineers.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the bug report |
| task\_id | VARCHAR(36) | NULLABLE, FOREIGN KEY REFERENCES tasks(id) | ID of the task related to the bug (if any) |
| user\_story\_id | VARCHAR(36) | NULLABLE, FOREIGN KEY REFERENCES user\_stories(id) | ID of the user story related to the bug (if any) |
| reported\_by\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID (SDET Engineer) who reported the bug |
| title | VARCHAR(255) | NOT NULL | Title/summary of the bug |
| description | TEXT | NOT NULL | Detailed description of the bug |
| severity | VARCHAR(50) | NOT NULL, ENUM('Critical', 'High', 'Medium', 'Low') | Severity of the bug |
| status | VARCHAR(50) | NOT NULL, ENUM('Open', 'In Progress', 'Resolved', 'Closed'), DEFAULT 'Open' | Current status of the bug |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of bug report creation |
| updated\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Timestamp of last update to bug record |

**Chat\_messages**

Table Stores chat messages, supporting both project-specific channels and direct messages.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the message |
| project\_id | VARCHAR(36) | NULLABLE, FOREIGN KEY REFERENCES projects(id) | ID of the project chat channel (NULL for direct messages) |
| sender\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID who sent the message |
| receiver\_id | VARCHAR(36) | NULLABLE, FOREIGN KEY REFERENCES users(id) | User ID of the recipient (NULL for project chat) |
| message\_text | TEXT | NOT NULL | Content of the message |
| sent\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp when the message was sent |
| is\_read | BOOLEAN | NOT NULL, DEFAULT FALSE | Indicates if a direct message has been read |

**Notifications Table**

Stores notification details for users.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| id | VARCHAR(36) | PRIMARY KEY, NOT NULL | Unique identifier for the notification |
| user\_id | VARCHAR(36) | NOT NULL, FOREIGN KEY REFERENCES users(id) | User ID who is the recipient of the notification |
| type | VARCHAR(100) | NOT NULL | Type of notification (e.g., 'task\_assignment', 'chat\_message') |
| entity\_type | VARCHAR(100) | NULLABLE | Type of entity related to the notification (e.g., 'task', 'sprint') |
| entity\_id | VARCHAR(36) | NULLABLE | ID of the entity related to the notification |
| message | TEXT | NOT NULL | Content of the notification message |
| is\_read | BOOLEAN | NOT NULL, DEFAULT FALSE | Indicates if the notification has been read |
| created\_at | DATETIME | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of notification creation |

**4. API SERVICES FOR PROJECT MANAGEMENT TOOL**

This section outlines the RESTful API services for the Agile Project Management Tool, detailing the endpoints, expected inputs, and outputs for each core functionality.

**1. Authentication Service**

Handles user registration, authentication, and password management.

**1.1 registerUser**

* **Endpoint:** POST /api/auth/register
* **Description:** Registers a new user account.
  + **Product Owner Role:** Requires firstName, lastName, email, password, confirmPassword, and role set to "Product Owner".
  + **Scrum Master/SDET Engineer Role:** Requires firstName, lastName, email, password, confirmPassword, role (either "Scrum Master" or "SDET Engineer"), and uniqueProjectCode.
  + **Technical Detail:** Validates data (email format, password complexity). Hashes the password using bcrypt before storing in the users table. For Scrum Masters/SDET Engineers, it verifies the uniqueProjectCode against the projects table.

**Request Body:**

{

"firstName": "string",

"lastName": "string",

"email": "user@example.com",

"password": "strongPassword123!",

"confirmPassword": "strongPassword123!",

"role": "Product Owner | Scrum Master | SDET Engineer",

"uniqueProjectCode": "string" // Required for Scrum Master / SDET Engineer

}

**Response Body (Success 201 Created):**

{

"message": "User registered successfully",

"userId": "uuid-string",

"token": "jwt.token.string" // Optional: may include JWT token on registration

}

**Response Body (Error 400 Bad Request / 409 Conflict):**

{

"message": "Error details (e.g., Email already exists, Invalid project code, Password mismatch)"

}

**1.2 authenticateUser**

* **Endpoint:** POST /api/auth/login
* **Description:** Authenticates a user and returns a JWT token.
* **Technical Detail:** Retrieves user record, compares provided password with hashed password using bcrypt. If correct, generates a JWT token containing userId, email, and role.

**Request Body:**

{

"email": "user@example.com",

"password": "strongPassword123!"

}

**Response Body (Success 200 OK):**

{

"message": "Login successful",

"userId": "uuid-string",

"email": "user@example.com",

"role": "Product Owner | Scrum Master | SDET Engineer",

"token": "jwt.token.string"

}

**Response Body (Error 401 Unauthorized):**

{

"message": "Invalid credentials"

}

**1.3 resetPasswordRequest**

* **Endpoint:** POST /api/auth/reset-password-request
* **Description:** Sends a password reset link to the user's email.
* **Technical Detail**: Checks if email exists. Generates a unique reset token (UUID), stores it with expiry in Redis Cache. Sends an email with the reset link using Spring Boot's JavaMailSender.

**Request Body:**

{

"email": [user@example.com](mailto:user@example.com)

}

**Response Body (Success 200 OK):**

{

"message": "Password reset link sent to your email"

}

**Response Body (Error 404 Not Found):**

{

"message": "Email not found"

}

**1.4 verifyResetToken**

* **Endpoint:** GET /api/auth/verify-reset-token/{token}
* **Description:** Verifies the validity of a password reset token.
* **Technical Detail:** Retrieves the stored token from Redis Cache and checks if it exists and has not expired.
* **Path Parameters:**
  + **token:** The unique reset token.

**Response Body (Success 200 OK):**

{

"message": "Token is valid",

"userId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 404 Not Found):**

{

"message": "Invalid or expired token"

}

**1.5 updatePassword**

* **Endpoint:** POST /api/auth/update-password
* **Description:** Updates the user's password using a valid reset token.
* **Technical Detail:** Verifies the reset token. Hashes the new password using bcrypt. Updates the password in the users table. Invalidates (deletes) the token from Redis Cache.

**Request Body:**

{

"token": "reset-token-string",

"newPassword": "newStrongPassword123!",

"confirmNewPassword": "newStrongPassword123!"

}

**Response Body (Success 200 OK):**

{

"message": "Password updated successfully"

}

**Response Body (Error 400 Bad Request / 401 Unauthorized):**

{

"message": "Error details (e.g., Invalid token, Passwords do not match, Password complexity not met)"

}

**2. User Service**

Manages user information and roles. Requires authentication and authorization.

**2.1 getUser**

* **Endpoint:** GET /api/users/{userId}
* **Description:** Retrieves user information for a given userId.
* **Technical Detail:** Queries the users table. Does NOT include the password.
* **Path Parameters:**
  + **userId:** The ID of the user to retrieve.

**Response Body (Success 200 OK):**

{

"id": "uuid-string",

"firstName": "string",

"lastName": "string",

"email": "user@example.com",

"role": "Product Owner | Scrum Master | SDET Engineer",

"isActive": true

}

**Response Body (Error 404 Not Found / 403 Forbidden):**

{

"message": "User not found or unauthorized access"

}

**2.2 updateUser**

* **Endpoint:** PUT /api/users/{userId}
* **Description:** Updates user information.
* **Technical Detail:** Validates the userData. Updates the user's record in the users table.
* **Path Parameters:**
  + **userId:** The ID of the user to update.

**Request Body:**

{

"firstName": "string",

"lastName": "string",

"email": "new\_email@example.com",

// Password update should use dedicated password reset/update flow

}

**Response Body (Success 200 OK):**

{

"message": "User information updated successfully",

"userId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to update this user"

}

**2.3 createUser (Product Owner Only)**

* **Endpoint:** POST /api/users
* **Description:** Creates a new user account. Typically called by a Product Owner.
* **Technical Detail:** Validates data. Hashes the password using bcrypt. Creates a new user record in the users table.

**Request Body:**

{

"firstName": "string",

"lastName": "string",

"email": "new\_user@example.com",

"password": "initialPassword123!",

"role": "Scrum Master | SDET Engineer" // Product Owner creates other roles

}

**Response Body (Success 201 Created):**

{

"message": "User created successfully",

"userId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden / 409 Conflict):**

{

"message": "Invalid input, unauthorized, or user with email already exists"

}

**2.4 deactivateUser (Product Owner Only)**

* **Endpoint:** PUT /api/users/{userId}/deactivate
* **Description:** Deactivates a user account.
* **Technical Detail**: Updates the is\_active column in the users table to false (soft delete).
* **Path Parameters:**
  + **userId:** The ID of the user to deactivate.

**Response Body (Success 200 OK):**

{

"message": "User deactivated successfully",

"userId": "uuid-string"

}

**Response Body (Error 404 Not Found / 403 Forbidden):**

{

"message": "User not found or unauthorized to deactivate"

}

**2.5 getUserRole**

* **Endpoint:** GET /api/users/{userId}/role
* **Description:** Retrieves the role of a user.
* **Technical Detail:** Queries the users table to retrieve the role field.
* **Path Parameters:**
  + **userId:** The ID of the user.

**Response Body (Success 200 OK):**

{

"userId": "uuid-string",

"role": "Product Owner | Scrum Master | SDET Engineer"

}

**Response Body (Error 404 Not Found):**

{

"message": "User not found"

}

**3. Project Service**

Manages project creation, retrieval, updates, and user assignments. Requires authentication and authorization.

**3.1 createProject (Product Owner Only)**

* **Endpoint:** POST /api/projects
* **Description:** Creates a new project.
* **Technical Detail:** Receives projectData (name, description) and the product\_owner\_id (from authenticated user context). Validates data. Creates a new record in the projects table.

**Request Body:**

{

"projectName": "New Project Title",

"description": "A detailed description of the project.",

"keyStakeholders": ["stakeholder1@example.com", "stakeholder2@example.com"]

}

**Response Body (Success 201 Created):**

{

"message": "Project created successfully",

"projectId": "uuid-string",

"uniqueProjectCode": "generated-code-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to create project"

}

**3.2 getProject**

* **Endpoint:** GET /api/projects/{projectId}
* **Description:** Retrieves project information.
* **Technical Detail:** Queries the projects table based on projectId.
* **Path Parameters:**
  + **projectId:** The ID of the project.

**Response Body (Success 200 OK):**

{

"id": "uuid-string",

"projectName": "Project Title",

"description": "Detailed project description.",

"productOwnerId": "uuid-string",

"uniqueProjectCode": "code-string",

"createdAt": "datetime-string",

"updatedAt": "datetime-string",

"members": [{

"userId": "uuid-member1",

"firstName": "Member",

"lastName": "One",

"role": "Scrum Master"

}]}

**Response Body (Error 404 Not Found / 403 Forbidden):**

{

"message": "Project not found or unauthorized access"

}

**3.3 updateProject (Product Owner Only)**

* **Endpoint:** PUT /api/projects/{projectId}
* **Description:** Updates project information.
* **Technical Detail:** Validates the projectData. Updates the project's record in the projects table.
* **Path Parameters:**
  + **projectId:** The ID of the project to update.

**Request Body:**

{

"projectName": "Updated Project Name",

"description": "Updated project description."

}

**Response Body (Success 200 OK):**

{

"message": "Project updated successfully",

"projectId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to update project"

}

**3.4 getAllProjects (Product Owner Filtered)**

* **Endpoint:** GET /api/projects
* **Description:** Retrieves all projects created by the authenticated Product Owner.
* **Technical Detail:** Retrieves all projects from the projects table, filtering results by the product\_owner\_id matching the authenticated user's ID.

**Response Body (Success 200 OK):**

[

{

"id": "uuid-project1",

"projectName": "Project Alpha",

"description": "...",

"productOwnerId": "uuid-current-user",

"uniqueProjectCode": "code-alpha"

},

{

"id": "uuid-project2",

"projectName": "Project Beta",

"description": "...",

"productOwnerId": "uuid-current-user",

"uniqueProjectCode": "code-beta"

}

]

**Response Body (Error 403 Forbidden):**

{

"message": "Unauthorized to access this resource"

}

**3.5 assignUserToProject (Product Owner Only)**

* **Endpoint:** POST /api/projects/{projectId}/assign-user
* **Description:** Assigns a user (Scrum Master or SDET Engineer) to a project.
* **Technical Detail:** Validates userId, projectId, and assignmentCode. If assignmentCode is provided, verifies it against the projects table's unique\_project\_code. Creates a new record in the project\_members table.
* **Path Parameters:**
  + **projectId:** The ID of the project.

**Request Body:**

{

"userId": "uuid-user-to-assign",

"assignmentCode": "optional-project-code"

// Used during registration, or can be passed by Product Owner

}

**Response Body (Success 200 OK):**

{

"message": "User assigned to project successfully",

"projectId": "uuid-string",

"userId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden / 404 Not Found):**

{

"message": "Invalid input, unauthorized, or project/user not found"

}

**4. Task Service**

Manages tasks, subtasks, file attachments, time tracking, and bug reporting. Requires authentication and authorization.

**4.1 createTask (Scrum Master, SDET Engineer)**

* **Endpoint:** POST /api/tasks
* **Description:** Creates a new task.
* **Technical Detail:** Receives taskData (title, description, due date, priority, status, optional userStoryId, optional sprintId, optional assignedToId). Validates data. Creates a new record in the tasks table.

**Request Body:**

{

"title": "Task Title",

"description": "Task description.",

"dueDate": "YYYY-MM-DDTHH:MM:SSZ",

"priority": "Low | Medium | High",

"status": "To Do | In Progress | Done | Blocked",

"userStoryId": "uuid-optional",

"sprintId": "uuid-optional",

"assignedToId": "uuid-optional-sdet-engineer"

}

**Response Body (Success 201 Created):**

{

"message": "Task created successfully",

"taskId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to create task"

}

**4.2 getTask**

* **Endpoint:** GET /api/tasks/{taskId}
* **Description:** Retrieves task information.
* **Technical Detail:** Queries the tasks table based on taskId. Includes related subtasks, attachments, and comments.
* **Path Parameters:**
  + **taskId:** The ID of the task.

**Response Body (Success 200 OK):**

{

"id": "uuid-string",

"title": "Task Title",

"description": "Task description.",

"dueDate": "YYYY-MM-DDTHH:MM:SSZ",

"priority": "Low | Medium | High",

"status": "To Do | In Progress | Done | Blocked",

"userStoryId": "uuid-optional",

"sprintId": "uuid-optional",

"assignedTo": {

"userId": "uuid-assigned-user",

"firstName": "Assigned",

"lastName": "User"

},

"createdBy": {

"userId": "uuid-creator",

"firstName": "Task",

"lastName": "Creator"

},

"subtasks": [

{"id": "uuid-subtask1", "description": "Subtask 1", "isCompleted": false}

],

"attachments": [

{"id": "uuid-attach1", "fileName": "doc.pdf", "fileUrl": "url-to-file"}

],

"timeLogs": [

{"id": "uuid-log1", "userId": "uuid-user", "durationMinutes": 60, "date": "YYYY-MM-DD"}

],

"bugs": [

{"id": "uuid-bug1", "title": "UI Glitch", "status": "Open"}

],

"createdAt": "datetime-string",

"updatedAt": "datetime-string"

}

**Response Body (Error 404 Not Found / 403 Forbidden):**

{

"message": "Task not found or unauthorized access"

}

**4.3 updateTask**

* **Endpoint:** PUT /api/tasks/{taskId}
* **Description: Updates task information.**
* **Technical Detail:** Validates the taskData. Updates the task's record in the tasks table.
* **Path Parameters:**
  + **taskId**: The ID of the task to update.

**Request Body:**

{

"title": "Updated Task Title",

"description": "Updated description.",

"dueDate": "YYYY-MM-DDTHH:MM:SSZ",

"priority": "High",

"status": "In Progress",

"assignedToId": "uuid-new-assignee"

}

**Response Body (Success 200 OK):**

{

"message": "Task updated successfully",

"taskId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to update task"

}

**4.4 deleteTask**

* **Endpoint:** DELETE /api/tasks/{taskId}
* **Description:** Deletes a task.
* **Technical Detail:** Performs a soft delete (e.g., setting an is\_deleted flag to true in tasks table).
* **Path Parameters:**
  + **taskId:** The ID of the task to delete.

**Response Body (Success 200 OK):**

{

"message": "Task deleted successfully",

"taskId": "uuid-string"

}

**Response Body (Error 404 Not Found / 403 Forbidden):**

{

"message": "Task not found or unauthorized to delete task"

}

**4.5 assignUserToTask (Scrum Master)**

* **Endpoint:** PUT /api/tasks/{taskId}/assign
* **Description:** Assigns a user (SDET Engineer) to a task.
* **Technical Detail:** Updates the assigned\_to\_id field in the tasks table.
* **Path Parameters:**
  + **taskId:** The ID of the task.

**Request Body:**

{

"userId": "uuid-sdet-engineer"

}

**Response Body (Success 200 OK):**

{

"message": "User assigned to task successfully",

"taskId": "uuid-string",

"userId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to assign user"

}

**4.6 getProjectTasks**

* **Endpoint:** GET /api/projects/{projectId}/tasks
* **Description:** Retrieves all tasks for a specific project.
* **Technical Detail:** Queries the tasks table to retrieve all tasks associated with the given projectId.
* **Path Parameters:**
  + **projectId:** The ID of the project.

**Response Body (Success 200 OK):**

[

{

"id": "uuid-task1",

"title": "Task 1 for Project A",

"status": "In Progress",

"assignedTo": {"userId": "uuid-sdet1", "firstName": "Alice"}

},

{

"id": "uuid-task2",

"title": "Task 2 for Project A",

"status": "To Do",

"assignedTo": null

}

]

**Response Body (Error 404 Not Found / 403 Forbidden):**

{

"message": "Project not found or unauthorized access"

}

**4.7 createSubtask**

* **Endpoint:** POST /api/tasks/{taskId}/subtasks
* **Description:** Creates a subtask for a given task.
* **Technical Detail:** Stores the subtask in the subtasks table, linked to its parent task\_id.
* **Path Parameters:**
  + **taskId:** The ID of the parent task.

**Request Body:**

{

"description": "Description of the new subtask."

}

**Response Body (Success 201 Created):**

{

"message": "Subtask created successfully",

"subtaskId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to create subtask"

}

**4.8 updateSubtaskStatus**

* **Endpoint:** PUT /api/subtasks/{subtaskId}/status
* **Description:** Updates the completion status of a subtask.
* **Technical Detail:** Updates the is\_completed field in the subtasks table.
* **Path Parameters:**
  + **subtaskId:** The ID of the subtask.

**Request Body:**

{

"isCompleted": true

}

**Response Body (Success 200 OK):**

{

"message": "Subtask status updated successfully",

"subtaskId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden / 404 Not Found):**

{

"message": "Invalid input, unauthorized, or subtask not found"

}

**4.9 uploadAttachment**

* **Endpoint:** POST /api/tasks/{taskId}/attachments
* **Description:** Attaches a file to a task.
* **Technical Detail:** Stores file metadata in the file\_attachments table. The actual file would be uploaded to a separate file storage service (e.g., S3, Google Cloud Storage), and fileUrl would be the link to that file**.**
* **Path Parameters:**
  + **taskId:** The ID of the task.

**Request Body (Multipart Form Data):**

Content-Type: multipart/form-data

{

"file": "binary-file-data"

}

**Response Body (Success 201 Created):**

{

"message": "File attached successfully",

"attachmentId": "uuid-string",

"fileName": "uploaded\_file.png",

"fileUrl": "url-to-uploaded-file"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Error uploading file or unauthorized"

}

**4.10 logTime (SDET Engineer)**

* **Endpoint:** POST /api/tasks/{taskId}/time-logs
* **Description:** Logs time spent by an SDET Engineer on a task.
* **Technical Detail:** Stores the time entry in the time\_tracking table. durationMinutes can be calculated from start\_time and end\_time if both are provided, or provided directly.
* **Path Parameters:**
  + **taskId:** The ID of the task.

**Request Body:**

{

"startTime": "YYYY-MM-DDTHH:MM:SSZ",

"endTime": "YYYY-MM-DDTHH:MM:SSZ",

"durationMinutes": 60 // Alternative to start/end time, or derived from them

}

**Response Body (Success 201 Created):**

{

"message": "Time logged successfully",

"timeLogId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to log time"

}

**4.11 reportBug (SDET Engineer)**

* **Endpoint:** POST /api/bugs
* **Description:** Reports a bug associated with a task or user story.
* **Technical Detail:** Stores the bug report in the bug\_reports table.

**Request Body:**

{

"title": "UI Glitch on Login Page",

"description": "When entering incorrect credentials, the error message flickers.",

"severity": "High | Medium | Low | Critical",

"status": "Open",

"taskId": "uuid-optional-task",

"userStoryId": "uuid-optional-user-story"

}

**Response Body (Success 201 Created):**

{

"message": "Bug reported successfully",

"bugId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to report bug"

}

**5. Chat Service**

Handles real-time and direct messaging. Requires authentication.

**5.1 createChatMessage**

* **Endpoint:** POST /api/chats
* **Description:** Creates a new chat message, either for a project channel or a direct message.
* **Technical Detail:** Receives chatData (project ID or recipient ID, sender ID, message content). Validates data. Creates a new record in the chat\_messages table. Handles file attachments by creating records in file\_attachments linked to the chat message.

**Request Body (Project Chat):**

{

"projectId": "uuid-project",

"messageText": "Hello team, let's discuss this task.",

"attachmentUrls": ["url-to-file1.jpg"] // Optional

}

**Request Body (Direct Message):**

{

"receiverId": "uuid-recipient",

"messageText": "Checking in on task X.",

"attachmentUrls": ["url-to-file1.jpg"] // Optional

}

**Response Body (Success 201 Created):**

{

"message": "Message sent successfully",

"chatMessageId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to send message"

}

**5.2 getProjectChats**

* **Endpoint:** GET /api/projects/{projectId}/chats
* **Description:** Retrieves all chat messages for a specific project.
* **Technical Detail:** Queries the chat\_messages table for messages associated with the given projectId, optionally retrieving associated file attachments.
* **Path Parameters:**
  + **projectId:** The ID of the project.
* **Query Parameters:**
  + **includeAttachments:** boolean (default: false)

**Response Body (Success 200 OK):**

[

{

"id": "uuid-chat1",

"projectId": "uuid-project",

"senderId": "uuid-sender1",

"senderName": "John Doe",

"messageText": "Initial project message.",

"sentAt": "datetime-string",

"attachments": []

},

{

"id": "uuid-chat2",

"projectId": "uuid-project",

"senderId": "uuid-sender2",

"senderName": "Jane Smith",

"messageText": "Attaching the updated design doc.",

"sentAt": "datetime-string",

"attachments": [

{"id": "uuid-attach2", "fileName": "design.pdf", "fileUrl": "url-to-design.pdf"}

]

}

]

**Response Body (Error 404 Not Found / 403 Forbidden):**

{

"message": "Project not found or unauthorized access"

}

**5.3 getDirectMessages**

* **Endpoint:** GET /api/users/{userId}/direct-messages
* **Description:** Retrieves all direct messages for the authenticated user, either sent or received.
* **Technical Detail:** Queries the chat\_messages table where sender\_id or receiver\_id matches the authenticated userId.
* **Path Parameters:**
  + **userId:** The ID of the authenticated user.

**Response Body (Success 200 OK):**

[

{

"id": "uuid-dm1",

"senderId": "uuid-other-user",

"senderName": "Mike",

"receiverId": "uuid-current-user",

"messageText": "Are you free to chat?",

"sentAt": "datetime-string",

"isRead": false

},

{

"id": "uuid-dm2",

"senderId": "uuid-current-user",

"receiverId": "uuid-other-user",

"messageText": "Yes, I am.",

"sentAt": "datetime-string",

"isRead": true

}

]

**Response Body (Error 403 Forbidden):**

{

"message": "Unauthorized access"

}

**6. Notification Service**

Manages in-app and email notifications. Requires authentication.

**6.1 createNotification**

* **Endpoint:** POST /api/notifications
* **Description:** Creates a new notification for a user.
* **Technical Detail:** Receives notificationData (userId, type, content). Validates data. Creates a new record in the notifications table. Uses a queuing system (e.g., RabbitMQ, Kafka) for asynchronous handling to support different channels (email, push notifications).

**Request Body:**

{

"userId": "uuid-recipient",

"type": "task\_assignment | deadline\_alert | sprint\_update | new\_chat\_message",

"entityType": "task | sprint | chat", // Optional, for linking

"entityId": "uuid-entity-id", // Optional, for linking

"message": "You have been assigned to 'Task X'.",

"emailRequired": true // Optional, if email notification is also desired

}

**Response Body (Success 201 Created):**

{

"message": "Notification created successfully",

"notificationId": "uuid-string"

}

**Response Body (Error 400 Bad Request / 403 Forbidden):**

{

"message": "Invalid input or unauthorized to create notification"

}

**6.2 getUserNotifications**

* **Endpoint:** GET /api/users/{userId}/notifications
* **Description:** Retrieves a user's notifications.
* **Technical Detail:** Queries the notifications table for all notifications for the given userId.
* **Path Parameters:**
  + **userId:** The ID of the user.
* **Query Parameters:**
  + **unreadOnly:** boolean (default: false)

**Response Body (Success 200 OK):**

[

{

"id": "uuid-notif1",

"userId": "uuid-user",

"type": "task\_assignment",

"message": "You've been assigned to 'Implement Feature Y'.",

"isRead": false,

"createdAt": "datetime-string"

},

{

"id": "uuid-notif2",

"userId": "uuid-user",

"type": "new\_chat\_message",

"message": "John sent you a message in 'Project Alpha' chat.",

"isRead": true,

"createdAt": "datetime-string"

}

]

**Response Body (Error 403 Forbidden):**

{

"message": "Unauthorized to access these notifications"

}

**6.3 markNotificationAsRead**

* **Endpoint:** PUT /api/notifications/{notificationId}/read
* **Description:** Marks a notification as read.
* **Technical Detail:** Updates the is\_read column in the notifications table to true**.**
* **Path Parameters:**
  + **notificationId:** The ID of the notification.

**Response Body (Success 200 OK):**

{

"message": "Notification marked as read",

"notificationId": "uuid-string"

}

**Response Body (Error 404 Not Found / 403 Forbidden):**

{

"message": "Notification not found or unauthorized"

}

**7. Report Service**

Generates various project reports. Requires authentication and authorization.

**7.1 generateProjectReport**

* **Endpoint:** GET /api/reports/projects/{projectId}
* **Description:** Generates a project report based on reportType.
* **Technical Detail:** Receives projectId and reportType. Queries relevant tables (projects, tasks, users, project\_members, time\_tracking, bug\_reports, sprints, user\_stories) to gather necessary data. Performs calculations and formatting.
* **Path Parameters:**
  + projectId: The ID of the project**.**
* **Query Parameters:**
  + reportType: overall\_progress | individual\_progress | team\_progress | sprint\_burn\_down | sprint\_burn\_up | cumulative\_flow
  + startDate: Optional, YYYY-MM-DD
  + endDate: Optional, YYYY-MM-DD
  + userId: Optional (for individual reports), uuid-string
  + sprintId: Optional (for sprint reports), uuid-string

**Response Body (Success 200 OK, Varies by reportType):**

{

"reportType": "overall\_progress",

"projectId": "uuid-string",

"reportData": {

"totalTasks": 100,

"completedTasks": 70,

"completionPercentage": 70.0,

"openBugs": 5,

"activeSprints": 2,

"teamMembers": 5

}

}

// OR for sprint\_burn\_down:

{

"reportType": "sprint\_burn\_down",

"sprintId": "uuid-string",

"chartData": [

{"date": "2025-06-01", "remainingWork": 100},

{"date": "2025-06-02", "remainingWork": 90},

// ... more data points

]

}

**Response Body (Error 400 Bad Request / 403 Forbidden / 404 Not Found):**

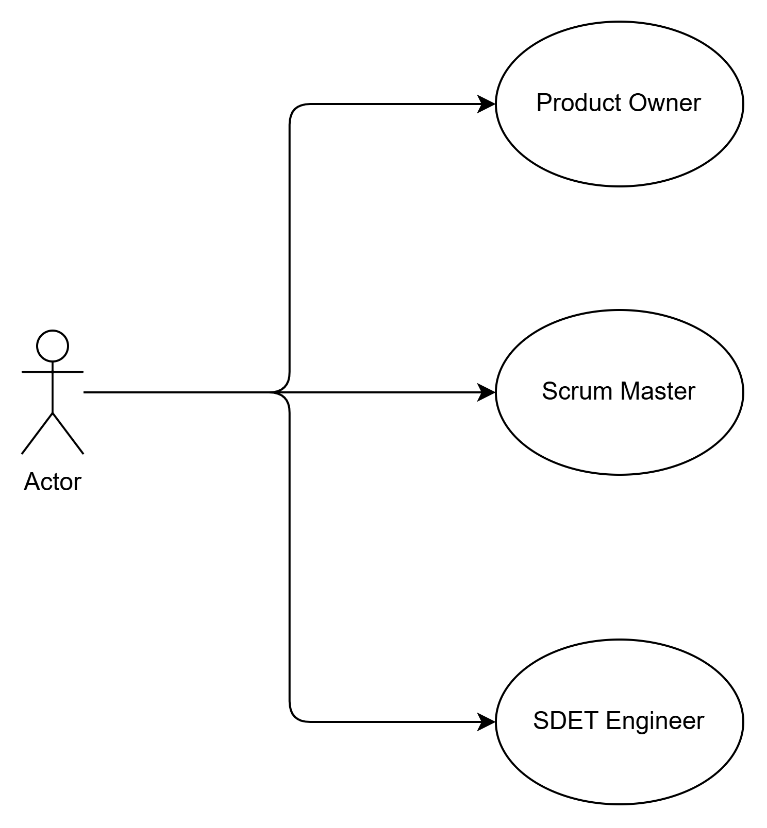
{

"message": "Invalid report type, project not found, or unauthorized"

}

**PROJECT FLOW DIAGRAMS**

**Roles**

****

**A screenshot of a computer screen

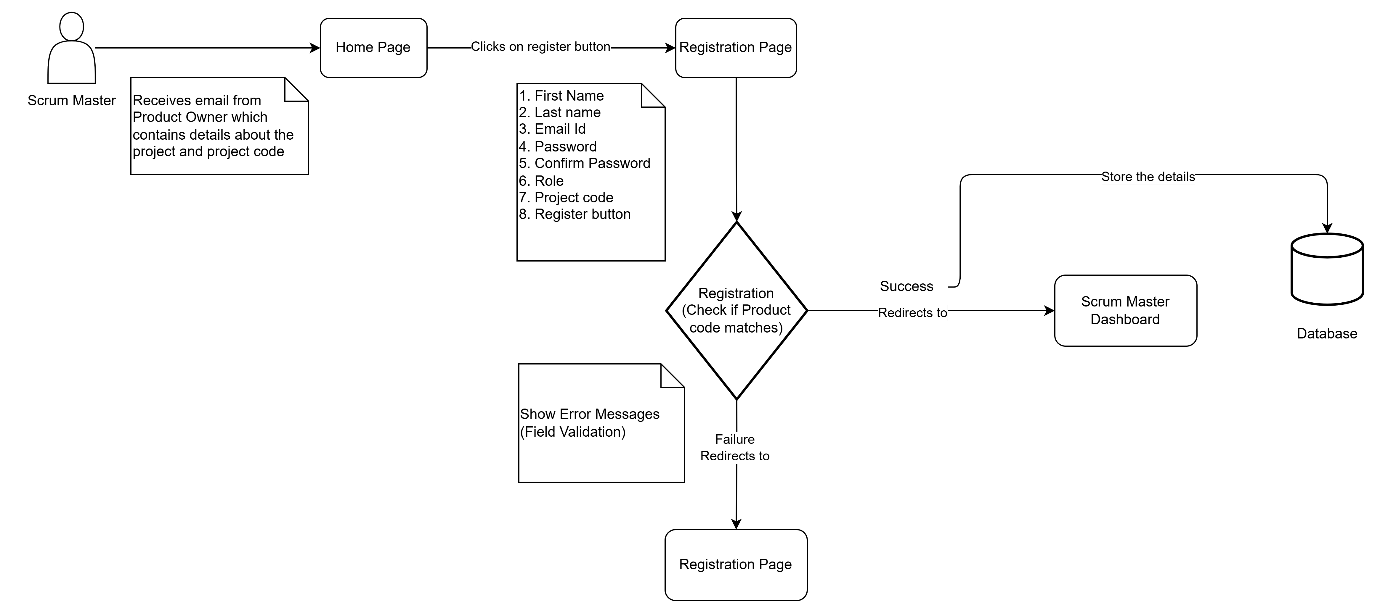
AI-generated content may be incorrect.Product Owner Registration**

**Create and Assign Project**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Scrum Master Registration**

****

**SDET Engineer Registration**

**A screenshot of a computer

AI-generated content may be incorrect.**

**EPIC and User Story Creation**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Sprint Creation**

**A screenshot of a black screen

AI-generated content may be incorrect.**

**Task Creation**

**A screenshot of a black screen

AI-generated content may be incorrect.**

**Report a Bug**

**A screenshot of a computer

AI-generated content may be incorrect.**