Issue Tracking System API Documentation

This document describes the API for the Issue Tracking System. It outlines the endpoints, their functionality, required parameters, and response formats.

Prerequisites

Before using the API, you need to register a super administrator user.

Register Super Admin User

- **Endpoint:** `POST /auth/register/superadmin`
- **Description:** Creates the first super administrator user in the system. This user has full access and can manage other users. This endpoint is typically used only once during initial setup.
- **Request Body:**

```
{
    "username": "string",  // Required
    "password": "string",  // Required
    "email": "string"     // Required
}
```

- **Response:**
  - Status Code: 201 Created
  - Body:

```
{
    "message": "Super admin user registered successfully."
}
```

- **Error Responses:**
  - Status Code: 400 Bad Request
    - Body: `{"error": "Username, password and email are required."}`
  - Status Code: 409 Conflict
    - Body: `{"error": "Username already exists."}`
  - Status Code: 500 Internal Server Error
    - Body: `{"error": "Failed to register super admin user."}`

Authentication

The API uses JWT (JSON Web Tokens) for authentication. After successful login, a token is returned, which must be included in the `Authorization` header of subsequent requests.

Login

- **Endpoint:** `POST /auth/login`
- **Description:** Authenticates a user and returns a JWT.
- **Request Body:**

```
{
    "username": "string",  // Required
    "password": "string"   // Required
}
```

- **Response:**
  - Status Code: 200 OK
  - Body:

```
{
    "message": "Login successful.",
    "token": "string"  // The JWT token
}
```

- **Error Responses:**
  - Status Code: 400 Bad Request
    - Body: `{"error": "Username and password are required."}`

    - Status Code: 401 Unauthorized
        - Body: `{"error": "Invalid username or password."}`
    - Status Code: 500 Internal Server Error
        - Body: `{"error": "Failed to login."}`

User Types

The system has three types of users, with roles and permissions managed in the database:

- **super_admin:** Has all privileges, including user management and role management.
- **admin:** Can manage certain aspects of the system, such as creating and managing issues, and creating normal users.
- **normal:** Basic user, typically can create and view issues.

User Management

These endpoints are for managing users in the system. Role-based access control is enforced.

Register Admin User

- **Endpoint:** `POST /auth/register/admin`
- **Description:** Creates a new administrator user. Requires a valid JWT from a super administrator.
- **Request Headers:**
    - `Authorization`: `Bearer <token>` (Replace `<token>` with a super admin's JWT)
- **Request Body:**

```
{
    "username": "string",  // Required
    "password": "string",  // Required
    "email": "string"     // Required
}
```

- **Response:**
    - Status Code: 201 Created
    - Body:

    ```
    {
        "message": "Admin user registered successfully."
    }
    ```

- **Error Responses:**
    - Status Code: 400 Bad Request
        - Body: `{"error": "Username, password and email are required."}`
    - Status Code: 401 Unauthorized
        - Body: `{"error": "No token provided."}` or `{"error": "Invalid or expired token."}`
    - Status Code: 403 Forbidden
        - Body: `{"error": "Unauthorized to perform this action."}`
    - Status Code: 409 Conflict
        - Body: `{"error": "Username already exists."}`
    - Status Code: 500 Internal Server Error
        - Body: `{"error": "Failed to register admin user."}`

Register Normal User

- **Endpoint:** `POST /auth/register/user`
- **Description:** Creates a new normal user. Requires a valid JWT from an admin or super administrator.
- **Request Headers:**
    - `Authorization`: `Bearer <token>` (Replace `<token>` with an admin or super admin's JWT)
- **Request Body:**

```
{
    "username": "string",  // Required
    "password": "string",  // Required
    "email": "string"     // Required
}
```

- **Response:**

  - Status Code: 201 Created
  - Body:

    ```
    {
        "message": "Normal user registered successfully."
    }
    ```

- **Error Responses:**
  - Status Code: 400 Bad Request
    - Body: `{"error": "Username, password and email are required."}`
  - Status Code: 401 Unauthorized
    - Body: `{"error": "No token provided."}` or `{"error": "Invalid or expired token."}`
  - Status Code: 403 Forbidden
    - Body: `{"error": "Unauthorized to perform this action."}`
  - Status Code: 409 Conflict
    - Body: `{"error": "Username already exists."}`
  - Status Code: 500 Internal Server Error
    - Body: `{"error": "Failed to register normal user."}`

Get All Users

- **Endpoint:** `GET /users`
- **Description:** Retrieves a list of all users. Requires a valid JWT from an admin or super administrator.
- **Request Headers:**
  - `Authorization`: `Bearer <token>` (Replace `<token>` with an admin or super admin's JWT)
- **Response:**
  - Status Code: 200 OK
  - Body:

    ```
    {
        "users": [
            {
                "id": "integer",
                "username": "string",
                "user_groud": "string",
                "email" : "string"
            },
            // ... more users
        ]
    }
    ```

- **Error Responses:**
  - Status Code: 401 Unauthorized
    - Body: `{"error": "No token provided."}` or `{"error": "Invalid or expired token."}`
  - Status Code: 403 Forbidden
    - Body: `{"error": "Unauthorized to perform this action."}`
  - Status Code: 500 Internal Server Error
    - Body: `{"error": "Failed to fetch users."}`

Get User by ID

- **Endpoint:** `GET /users/:id`
- **Description:** Retrieves a single user by their ID. Requires a valid JWT from an admin or super administrator.
- **Request Headers:**
  - `Authorization`: `Bearer <token>`
- **Path Parameters:**
  - `id`: The ID of the user to retrieve.
- **Response:**
  - Status Code: 200 OK
  - Body:

```
{
    "user": {
        "id": "integer",
        "username": "string",
        "role": "string"
    }
}
```

- **Error Responses:**
  - Status Code: 401 Unauthorized
    - Body: `{"error": "No token provided."}` or `{"error": "Invalid or expired token."}`
  - Status Code: 403 Forbidden
    - Body: `{"error": "Unauthorized to access this user information."}`
  - Status Code: 404 Not Found
    - Body: `{"error": "User with ID ${id} not found."}`
  - Status Code: 500 Internal Server Error
    - Body: `{"error": "Failed to fetch user."}`

Issue Management

Create Issue

- **Endpoint:** `POST /issues`
- **Description:** Creates a new issue. Requires a valid JWT from a normal user or an admin.
- **Request Headers:**
  - `Authorization` : `Bearer <token>` (Replace `<token>` with the actual JWT)
- **Request Body:**

```
{
   "title": "string",       // Required
   "description": "string"  // Required
}
```

- **Response:**
  - Status Code: 201 Created
  - Body:

    ```json { "title": "string", "description": "string" }

```

```

- **Error Responses:**
  - Status Code: 400 Bad Request
    - Body: `{"error": "Title and description are required."}`
  - Status Code: 401 Unauthorized
    - Body: `{"error": "No token provided."}` or `{"error": "Invalid or expired token."}`
  - Status Code: 500 Internal Server Error
    - Body: `{"error": "Failed to create issue."}`

### Get Issue

- **Endpoint:** `GET /issues/:id`
- **Description:** Retrieves a single issue by its ID. Requires a valid JWT from a normal user or an admin.
- **Request Headers:**
  - `Authorization` : `Bearer <token>` (Replace `<token>` with the actual JWT)
- **Path Parameters:**
  - `id` : The ID of the issue to retrieve.
- **Response:**
  - Status Code: 200 OK
  - Body:
```

```
{
    "issue":{
        "id": "integer",
        "title": "string",
        "description": "string",
        "userId": "integer",
        "created_at": "string",
        "created_by": "string",
        "updated_at": "string",
        "updated_by": "string"
    }
}
```

- **Error Responses:**
    - Status Code: 401 Unauthorized
        - Body: `{"error": "No token provided."}` or `{"error": "Invalid or expired token."}`
    - Status Code: 500 Internal Server Error
        - Body: `{"error": "Failed to get the issue"}`

Update Issue

- **Endpoint:** `PATCH /issues/:id`
- **Description:** Updates an existing issue. Requires a valid JWT from a normal user or an admin.
- **Request Headers:**
    - `Authorization`: `Bearer <token>`
- **Path Parameters:**
    - `id`: The ID of the issue to update.
- **Request Body:**

```
{
    "title": "string",
    "description": "string"
}
```

- **Response:**
    - Status Code: 200 OK
    - Body: The updated issue object (same structure as the Create Issue response).
- **Error Responses:**
    - Status Code: 400 Bad Request
        - Body: `{"error": "Invalid input."}`
    - Status Code: 401 Unauthorized
        - Body: `{"error": "No token provided."}` or `{"error": "Invalid or expired token."}`
    - Status Code: 404 Not Found
        - Body: `{"error": "Issue not found."}`
    - Status Code: 500 Internal Server Error
        - Body: `{"error": "Failed to update issue."}`

Issue Revisions

Each update to an issue creates a new revision, allowing you to track the history of changes.

Get Issue Revisions

- **Endpoint:** `GET /issues/:issueId/revisions`
- **Description:** Retrieves all revisions for a specific issue. Requires a valid JWT from a normal user or an admin.
- **Request Headers:**
    - `Authorization`: `Bearer <token>`
- **Path Parameters:**
    - `issueId`: The ID of the issue for which to retrieve revisions.
- **Response:**
    - Status Code: 200 OK
    - Body: An array of revision objects, ordered by `updatedAt` in descending order (most recent first).

```
[
    {
        "id": "integer",
        "issueId": "integer",
        "userId": "integer",        // ID of the user who made the revision
        "currentState": "object", // The complete state of the issue after this revision
        "changes": "object | null",   // The fields that were changed in this revision, or null for initial creation
        "updatedAt": "string",
        "user": {               // Information about the user who made the revision
            "id": "integer",
            "username": "string"
        }
    },
    // ... more revisions
]
```

- **Error Responses:**
  - Status Code: 401 Unauthorized
    - Body: `{"error": "No token provided."}` or `{"error": "Invalid or expired token."}`
  - Status Code: 500 Internal Server Error
    - Body: `{"error": "Failed to fetch issue revisions."}`

Phase II System Updates

- **Proper user management (role permissions in the database)**
- **Delete user functionality**
- **Issue assignee**
- **Issue release**
- **Issue status**
- **Archive Issues**
- **Dev swag for an issue**

Delete User

- **Endpoint:** `DELETE /users/:id`
- **Description:** Deletes a user. Requires a valid JWT from a super administrator.
- **Request Headers:**
  - `Authorization`: `Bearer <token>`
- **Path Parameters:**
  - `id`: The ID of the user to delete
- **Response:**
  - Status Code: 200 OK
  - Body:

```
{
    "message": "User deleted successfully"
}
```

Create Issue with New Attributes

- **Endpoint:** `POST /issues`
- **Description:** Creates a new issue. Requires a valid JWT from a normal user or an admin.
- **Request Headers:**
  - `Authorization`: `Bearer <token>` (Replace `<token>` with the actual JWT)
- **New Attributes:**
  - `assigneeId`
  - `releaseId`
  - `status`
  - `devSwag`
- **Request Body:**

```
{
    "title": "string",
    "description": "string",
    "assigneeId": "integer",
    "releaseId": "integer",
    "status": "string",  //(e.g., "open", "ready for development")
    "devSwag": "string"
}
```

- **Response:**
    - Status Code: 201 Created