API Documentation

Contents

1 Authentication APIs

1.1 Login

- **Method:** POST
- **URL:** `http://localhost:8080/auth/login`
- **Description:** Authenticates the user and provides a JWT token upon successful login.
- **Request Body:**

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

- **Response:**

```
{
  "token": "string"
}
```

- **Error:** Throws `BadRequestException` if the username or password is blank, or `AuthenticationException` for invalid credentials.

1.2 Register

- **Method:** POST
- **URL:** `http://localhost:8080/auth/register`
- **Description:** Registers a new user in the system.
- **Request Body:**

```
{
  "username": "string",
  "password": "string",
  "email": "string",
  "name": "string",
  "surname": "string"
}
```

- **Validation Rules:**
  - **username**: Must be at least 3 characters long, not blank, and unique.
  - **password**: Must be at least 6 characters long and not blank.
  - **email**: Must not be blank, must match a valid email format (e.g., `example@domain.com`), and must be unique.
  - **name**: Must be between 2 and 50 characters long and not blank.
  - **surname**: Must be between 2 and 50 characters long and not blank.

- **Response:**

```
{
  "id": "integer",
  "username": "string",
  "email": "string",
  "name": "string",
  "surname": "string"
}
```

- **Error:**
  - Throws `ValidationException` if any field does not meet the validation rules.
  - Throws `BadRequestException` for invalid input data.
  - Throws `ConflictException` if the username or email already exists in the system.

1.3 Validate Token

- **Method:** GET
- **URL:** `http://localhost:8080/auth/validate`
- **Description:** Validates the provided JWT token to ensure it is active and unexpired.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Response:**

```
{
  "username": "string",
  "role": "string",
  "expiresAt": "string"
}
```

- **Error:** Throws `AuthenticationException` if the token is invalid or expired.

2 Priority Management APIs

2.1 Get All Priorities

- **Method:** GET
- **URL:** `http://localhost:8080/priorities`
- **Description:** Retrieves a list of all priorities.
- **Response:**

```
[
  {
    "id": 1,
    "name": "string"
  }
]
```

- **Error:** None.

2.2 Get Priority by ID

- **Method:** GET
- **URL:** `http://localhost:8080/priorities/{id}`
- **Description:** Retrieves a specific priority by its ID.
- **Path Parameter:**

```
id: integer (ID of the priority)
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `BadRequestException` if the ID is invalid.
  - Throws `NotFoundException` if the priority is not found.

2.3 Create a Priority

- **Method:** POST
- **URL:** `http://localhost:8080/priorities`
- **Description:** Creates a new priority. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Request Body:**

```
{
  "name": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.

2.4 Update a Priority

- **Method:** PUT
- **URL:** `http://localhost:8080/priorities/{id}`
- **Description:** Updates an existing priority by its ID. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the priority)
```

- **Request Body:**

```
{
  "name": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.
  - Throws `BadRequestException` if the ID is invalid.

2.5 Delete a Priority

- **Method:** DELETE
- **URL:** `http://localhost:8080/priorities/{id}`
- **Description:** Deletes a priority by its ID. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the priority)
```

- **Response:** No content.
- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.
  - Throws `BadRequestException` if the ID is invalid.

3 Role Management APIs

3.1 Get All Roles

- **Method:** GET
- **URL:** `http://localhost:8080/roles`
- **Description:** Retrieves a list of all roles.
- **Response:**

```
[
  {
    "id": 1,
    "name": "string"
  }
]
```

- **Error:** None.

3.2 Get Role by ID

- **Method:** GET
- **URL:** `http://localhost:8080/roles/{id}`
- **Description:** Retrieves a specific role by its ID.
- **Path Parameter:**

```
id: integer (ID of the role)
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `BadRequestException` if the ID is invalid.
  - Throws `NotFoundException` if the role is not found.

3.3 Create a Role

- **Method:** POST
- **URL:** `http://localhost:8080/roles`
- **Description:** Creates a new role. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Request Body:**

```
{
  "name": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.

3.4 Update a Role

- **Method:** PUT
- **URL:** `http://localhost:8080/roles/{id}`
- **Description:** Updates an existing role by its ID. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the role)
```

- **Request Body:**

```
{
  "name": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.
  - Throws `BadRequestException` if the ID is invalid.

3.5 Delete a Role

- **Method:** DELETE
- **URL:** `http://localhost:8080/roles/{id}`
- **Description:** Deletes a role by its ID. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the role)
```

- **Response:** No content.
- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.
  - Throws `BadRequestException` if the ID is invalid.

4 Status Management APIs

4.1 Get All Statuses

- **Method:** GET
- **URL:** `http://localhost:8080/statuses`
- **Description:** Retrieves a list of all statuses.
- **Response:**

```
[
  {
    "id": 1,
    "name": "string"
  }
]
```

- **Error:** None.

4.2 Get Status by ID

- **Method:** GET
- **URL:** `http://localhost:8080/statuses/{id}`
- **Description:** Retrieves a specific status by its ID.
- **Path Parameter:**

```
id: integer (ID of the status)
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `BadRequestException` if the ID is invalid.
  - Throws `NotFoundException` if the status is not found.

4.3 Create a Status

- **Method:** POST
- **URL:** `http://localhost:8080/statuses`
- **Description:** Creates a new status. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Request Body:**

```
{
  "name": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.

4.4 Update a Status

- **Method:** PUT
- **URL:** `http://localhost:8080/statuses/{id}`
- **Description:** Updates an existing status by its ID. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the status)
```

- **Request Body:**

```
{
  "name": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "name": "string"
}
```

- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.
  - Throws `BadRequestException` if the ID is invalid.

4.5 Delete a Status

- **Method:** DELETE
- **URL:** `http://localhost:8080/statuses/{id}`
- **Description:** Deletes a status by its ID. Only accessible to admins.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the status)
```

- **Response:** No content.
- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.
  - Throws `BadRequestException` if the ID is invalid.

5 Task Management APIs

5.1 Get All Tasks (Paginated)

- **Method:** GET
- **URL:** `http://localhost:8080/tasks`
- **Description:** Retrieves a paginated list of tasks based on the role and user ID of the requester.
- **Query Parameters:**

```
page: integer (optional, default = 1)
limit: integer (optional, default = 10)
filter: string (optional)
```

- **Response:**

```
{
  "data": [
    {
      "id": 1,
      "title": "string",
      "description": "string",
      "assignedTo": {
        "userId": 1,
        "teamId": 2
      },
      "assignedBy": {
        "userId": 3,
        "teamId": 4
      },
      "priority": {
        "id": 1,
        "name": "string"
      },
      "status": {
        "id": 1,
        "name": "string"
      },
      "assignedDate": "2025-01-06",
      "dueDate": "2025-01-10"
    }
  ],
  "currentPage": 1,
  "pageSize": 10,
  "totalItems": 100,
  "totalPages": 10
}
```

- **Error:**
  - Throws `BadRequestException` if ‘page‘ or ‘limit‘ is invalid.

5.2 Get Task by ID

- **Method:** GET
- **URL:** `http://localhost:8080/tasks/{id}`
- **Description:** Retrieves the details of a specific task by its ID.
- **Path Parameter:**

```
id: integer (ID of the task)
```

- **Response:**

```
{
  "id": 1,
  "title": "string",
  "description": "string",
  "assignedTo": {
    "userId": 1,
    "teamId": 2
  },
  "assignedBy": {
    "userId": 3,
    "teamId": 4
  },
    "priority": {
      "id": 1,
      "name": "string"
    },
    "status": {
      "id": 1,
      "name": "string"
    },
    "assignedDate": "2025-01-06",
    "dueDate": "2025-01-10"
  }
```

- **Error:**
  - Throws `BadRequestException` if the ID is invalid.
  - Throws `NotFoundException` if the task is not found.

5.3 Create a Task

- **Method:** POST
- **URL:** `http://localhost:8080/tasks`
- **Description:** Creates a new task and assigns it to a user or team.
- **Headers:**

```
  Authorization: Bearer <JWT_TOKEN>
```

- **Request Body:**

```
  {
    "title": "string",
    "description": "string",
    "assignedTo": {
      "userId": 1,
      "teamId": 2
    },
    "priorityId": 1,
    "statusId": 1,
    "dueDate": "2025-01-10"
  }
```

- **Response:**

```
  {
    "id": 1,
    "title": "string",
    "description": "string",
    "assignedTo": {
      "userId": 1,
      "teamId": 2
    },
    "assignedBy": {
      "userId": 3,
      "teamId": 4
    },
    "priority": {
      "id": 1,
      "name": "string"
    },
    "status": {
      "id": 1,
      "name": "string"
    },
    "assignedDate": "2025-01-06",
    "dueDate": "2025-01-10"
  }
```

- **Error:** None.

5.4 Update a Task

- **Method:** PUT
- **URL:** `http://localhost:8080/tasks/{id}`
- **Description:** Updates an existing task by its ID.
- **Headers:**

```
  Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
  id: integer (ID of the task)
```

- **Request Body:**

```
  {
    "title": "string",
    "description": "string",
    "assignedTo": {
      "userId": 1,
      "teamId": 2
    },
    "priorityId": 1,
    "statusId": 1,
    "dueDate": "2025-01-10"
  }
```

- **Response:**

```
  {
    "id": 1,
    "title": "string",
    "description": "string",
    "assignedTo": {
      "userId": 1,
      "teamId": 2
    },
    "assignedBy": {
      "userId": 3,
      "teamId": 4
    },
    "priority": {
      "id": 1,
      "name": "string"
    },
    "status": {
      "id": 1,
      "name": "string"
    },
    "assignedDate": "2025-01-06",
    "dueDate": "2025-01-10"
  }
```

- **Error:**
  - Throws `BadRequestException` if the task ID is invalid.
  - Throws `NotFoundException` if the task is not found.

5.5 Delete a Task

- **Method:** DELETE
- **URL:** `http://localhost:8080/tasks/{id}`
- **Description:** Deletes a task by its ID.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the task)
```

- **Response:** No content.
- **Error:**
  - Throws `BadRequestException` if the task ID is invalid.
  - Throws `NotFoundException` if the task is not found.

6 Team Management APIs

6.1 Get All Teams (Paginated)

- **Method:** GET
- **URL:** `http://localhost:8080/teams`
- **Description:** Retrieves a paginated list of teams available for the user based on their role.
- **Query Parameters:**

```
page: integer (optional, default = 1)
limit: integer (optional, default = 10)
filter: string (optional)
```

- **Response:**

```
{
  "data": [
    {
      "id": 1,
      "title": "string",
      "createdDate": "2025-01-06",
      "createdBy": {
        "id": 1,
        "username": "string",
        "name": "string",
        "surname": "string",
        "email": "string",
        "authRole": "string"
      },
      "members": [
        {
          "user": {
            "id": 2,
            "username": "string",
            "name": "string",
            "surname": "string",
            "email": "string",
            "authRole": "string"
          },
          "role": {
            "id": 1,
            "name": "string"
          },
          "joinedDate": "2025-01-01"
        }
      ]
    }
  ],
  "currentPage": 1,
  "pageSize": 10,
  "totalItems": 100,
  "totalPages": 10
}
```

- **Error:** None.

6.2 Get Team by ID

- **Method:** GET
- **URL:** `http://localhost:8080/teams/{id}`
- **Description:** Retrieves the details of a specific team by its ID.
- **Path Parameter:**

```
id: integer (ID of the team)
```

- **Response:**

```
{
  "id": 1,
  "title": "string",
  "createdDate": "2025-01-06",
  "createdBy": {
    "id": 1,
    "username": "string",
    "name": "string",
    "surname": "string",
    "email": "string",
    "authRole": "string"
  },
  "members": [
    {
      "user": {
        "id": 2,
        "username": "string",
        "name": "string",
        "surname": "string",
        "email": "string",
        "authRole": "string"
      },
      "role": {
        "id": 1,
        "name": "string"
      },
      "joinedDate": "2025-01-01"
    }
  ]
}
```

- **Error:**
  - Throws `BadRequestException` if the team ID is invalid.
  - Throws `NotFoundException` if the team is not found.

6.3 Create a Team

- **Method:** POST
- **URL:** `http://localhost:8080/teams`
- **Description:** Creates a new team.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Request Body:**

```
{
  "title": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "title": "string",
  "createdDate": "2025-01-06",
  "createdBy": {
    "id": 1,
    "username": "string",
    "name": "string",
    "surname": "string",
    "email": "string",
    "authRole": "string"
  },
  "members": []
}
```

- **Error:** None.

6.4 Update a Team

- **Method:** PUT
- **URL:** `http://localhost:8080/teams/{id}`
- **Description:** Updates the details of an existing team by its ID.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the team)
```

- **Request Body:**

```
{
  "title": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "title": "string",
  "createdDate": "2025-01-06",
  "createdBy": {
    "id": 1,
    "username": "string",
    "name": "string",
    "surname": "string",
    "email": "string",
    "authRole": "string"
  },
  "members": []
}
```

- **Error:** None.

6.5 Delete a Team

- **Method:** DELETE
- **URL:** `http://localhost:8080/teams/{id}`
- **Description:** Deletes a team by its ID.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the team)
```

- **Response:** No content.
- **Error:**
  - Throws `BadRequestException` if the team ID is invalid.
  - Throws `NotFoundException` if the team is not found.

6.6 Add a Member to a Team

- **Method:** POST
- **URL:** `http://localhost:8080/teams/{id}/members`
- **Description:** Adds a new member to a specific team.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Request Body:**

```
    {
      "userId": 2,
      "roleId": 1
    }
```

- **Response:** HTTP 200 OK.
- **Error:** Throws `BadRequestException` if the team ID is invalid.

7 User Management APIs

7.1 Get All Users (Paginated)

- **Method:** GET
- **URL:** `http://localhost:8080/users`
- **Description:** Retrieves a paginated list of users. Requires authentication.
- **Query Parameters:**

```
page: integer (optional, default = 1)
limit: integer (optional, default = 10)
filter: string (optional)
```

- **Response:**

```
{
  "data": [
    {
      "id": 1,
      "username": "string",
      "name": "string",
      "surname": "string",
      "email": "string",
      "authRole": "string"
    }
  ],
  "currentPage": 1,
  "pageSize": 10,
  "totalItems": 100,
  "totalPages": 10
}
```

- **Error:** None.

7.2 Get User by ID

- **Method:** GET
- **URL:** `http://localhost:8080/users/{id}`
- **Description:** Retrieves a specific user by ID.
- **Path Parameter:**

```
id: integer (ID of the user)
```

- **Response:**

```
{
  "id": 1,
  "username": "string",
  "name": "string",
  "surname": "string",
  "email": "string",
  "authRole": "string"
}
```

- **Error:**
  - Throws `BadRequestException` if the user ID is invalid.

7.3 Delete User

- **Method:** DELETE
- **URL:** `http://localhost:8080/users/{id}`
- **Description:** Deletes a user by ID. Requires admin privileges.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the user)
```

- **Response:** No content.
- **Error:**
  - Throws `AuthorizationException` if the user is not an admin.
  - Throws `BadRequestException` if the user ID is invalid.

7.4 Get User's Teams

- **Method:** GET
- **URL:** `http://localhost:8080/users/{id}/teams`
- **Description:** Retrieves the teams that the user belongs to. Users can access their own teams or require admin privileges.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the user)
```

- **Response:**

```
[
  {
    "id": 1,
    "title": "string",
    "createdDate": "2025-01-06",
    "createdBy": {
      "id": 1,
      "username": "string",
      "name": "string",
      "surname": "string",
      "email": "string",
      "authRole": "string"
    },
    "members": [
      {
        "user": {
          "id": 2,
          "username": "string",
          "name": "string",
          "surname": "string",
          "email": "string",
          "authRole": "string"
        },
        "role": {
          "id": 1,
          "name": "string"
        },
        "joinedDate": "2025-01-01"
      }
    ]
  }
]
```

- **Error:**
  - Throws `Forbidden` if the user is not authorized to access the teams.
  - Throws `BadRequestException` if the user ID is invalid.

7.5 Get Current User Profile

- **Method:** GET
- **URL:** `http://localhost:8080/users/me`
- **Description:** Retrieves the profile information of the currently authenticated user.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Response:**

```
{
  "id": 1,
  "username": "string",
  "name": "string",
  "surname": "string",
  "email": "string",
  "authRole": "string"
}
```

- **Error:** None.

7.6 Update Current User Profile

- **Method:** PUT
- **URL:** `http://localhost:8080/users/me`
- **Description:** Updates the profile information of the currently authenticated user.

- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Request Body (Optional Fields):**

```
{
  "name": "string",
  "surname": "string",
  "email": "string"
}
```

- **Response:**

```
{
  "id": 1,
  "username": "string",
  "name": "string",
  "surname": "string",
  "email": "string",
  "authRole": "string"
}
```

- **Error:** None.

7.7 Change Current User Password

- **Method:** PUT
- **URL:** `http://localhost:8080/users/me/password`
- **Description:** Changes the password of the currently authenticated user.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Request Body:**

```
{
  "oldPassword": "string",
  "newPassword": "string"
}
```

- **Response:**

```
{
  "message": "Password updated successfully"
}
```

- **Error:** None.

7.8 Delete Current User Account

- **Method:** DELETE
- **URL:** `http://localhost:8080/users/me`
- **Description:** Deletes the account of the currently authenticated user.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Response:**

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

- **Error:** None.

7.9 Update User Role

- **Method:** PUT
- **URL:** `http://localhost:8080/users/{id}/role`
- **Description:** Updates the role of a user. Requires admin privileges.
- **Headers:**

```
Authorization: Bearer <JWT_TOKEN>
```

- **Path Parameter:**

```
id: integer (ID of the user)
```

- **Request Body:**

```
{
  "role": "string" // New role (e.g., "ADMIN", "USER")
}
```

- **Response:**

```
{
  "message": "User role updated successfully"
}
```

- **Error:**
  - Throws `Forbidden` if the user is not an admin.
  - Throws `BadRequestException` if the user ID is invalid.