Task Management API Documentation

This documentation provides details on how to use the Task Management API, including endpoint descriptions, request and response formats, and authentication requirements.

Base URL

http://localhost:5000/api

Authentication

The API uses JWT (JSON Web Tokens) for authentication. To access protected endpoints, you need to include a valid JWT token in the Authorization header of your requests.

Obtaining a JWT Token

You can obtain a JWT token by logging in with valid user credentials.

Endpoints

1. User Authentication

Register

- **URL**: /user/registration
- **Method**: POST
- **Description**: Registers a new user.

Request:

```
{
  "user_name": "isra_shafique16",
  "email": "isra@gmail.com",
  "password": "isra134"
}
```

Response:

```
{
  "success": true,
  "message": "User registered successfully",
  "status": 200
}
```

- **Authentication**: Not required

Login

- **URL**: /user/login
- **Method**: POST
- **Description**: Authenticates a user and returns a JWT token.

Request:


```
{
  "email": "isra@gmail.com",
  "password": "isra134"
}
```


Response:


```
{
  "success": true,
  "result": {
    "access_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjMxIiwiZW1haWwiOiJpc3JhQGdtYWlsLmNvbSIsImlhdCI
6MTcxNzQzNTAzMSwiZXhwIjoxNzE3NDM4NjMxfQ.kkHZW000oemm1hhbbuHxfNZI-Q92UyLxZecFSglt6i4",
    "token_type": "Bearer",
    "expires_in": 3600
  },
  "message": "user login sucessfull",
  "status": 200
}
```


- **Authentication**: Not required

2. Task Management

Get Tasks

- **URL**: /tasks
- **Method**: GET
- **Description**: Retrieves a list of tasks

Response:


```
{
  "success": true,
```

```
  "result": [
    {
      "title": "Complete Project Report",
      "description": "Finalize and submit the project report by the end of the week.",
      "due_date": "2024-06-03T16:41:04.742Z",
      "priority": "Low",
      "status": "COMPLETED",
      "user_id": "31",
      "id": "338"
    },
    {
      "title": "Complete Project Report",
      "description": "Finalize and submit the project report by the end of the week.",
      "due_date": "2024-06-03T17:07:52.176Z",
      "priority": "High",
      "status": "PENDING",
      "user_id": "31",
      "id": "339"
    }
  ],
  "message": "getting all tasks",
  "status": 200
}
```


- **Authentication**: Required

Example In header "Authorization: Bearer your.jwt.token.here

Get One Task

- **URL**: /task/:id
- **Method**: GET
- **Description**: Retrieves a single task by its ID.

Response:


```
{
  "success": true,
  "result": [
    {
      "title": "Complete Project Report",
```

```
        "description": "Finalize and submit the project report by week.",
        "due_date": "2024-06-03T16:38:17.732Z",
        "priority": "High",
        "status": "PENDING",
        "user_id": "31",
        "id": "337"
      }
  ],
  "message": "getting the task",
  "status": 200
}
```


- **Authentication**: Required

Create Task

- **URL**: /tasks
- **Method**: POST
- **Description**: Creates a new task.

Request:


```
[{
  "title": "Complete Project Report",
  "description": "Finalize and submit the project report by the end of the week.",
  "priority": "High",
  "status": "PENDING"
}]
```


Response:


```
{
  "success": true,
  "result": [
    {
      "title": "Complete Project Report",
      "description": "Finalize and submit the project report by the end",
      "priority": "High",
      "status": "PENDING",
      "user_id": 1,
      "id": "357"
    }
```

```
  ],
  "message": "tasks Added",
  "status": 200
}
```


- **Authentication**: Required

Update Task

- **URL**: /task/:id
- **Method**: Patch
- **Description**: Updates an existing task.

Request:


```
{
  "title": "A very new Project",
  "description": "Finalize and submit the project report by the end of the week.",
  "priority": "High",
  "status": "COMPLETED"
}
```


Response:


```
{
  "success": true,
  "result": {
    "title": "A very new Project",
    "description": "Finalize and submit the project report by the end of the week.",
    "priority": "High",
    "status": "COMPLETED",
    "due_date": "2024-06-03T16:38:17.732Z",
    "user_id": "31",
    "id": "337"
  },
  "message": "tasks updated",
  "status": 200
}
```


- **Authentication**: Required

Delete Task

- **URL**: /task/:id
- **Method**: DELETE
- **Description**: Deletes an existing task.

Response:

```
{
  "success": true,
  "result": 1,
  "message": "tasks successfully deleted",
  "status": 200
}
```

Authentication: Required

Error Handling Details

- **Joi Validation Errors**: If the error originates from Joi (used for input validation), a 400 Bad Request status is returned with a specific message indicating a client error.
- **Syntax or Database Violations**: Handles errors such as unique constraint violations (e.g., email already in use) and other database-related issues. It returns a 400 status with an appropriate error message.
- **Unauthorized Access**: For login errors or missing authorization, a 401 Unauthorized status is returned with a descriptive message.
- **URL Not Found**: If the requested URL does not exist, a 404 Not Found status is returned with a specific message.
- **Forbidden Access**: For errors related to forbidden access, a 403 Forbidden status is returned with a descriptive message in case teh jwt token is expired or false.
- **Internal Server Errors**: For all other unhandled errors, a 500 Internal Server Error status is returned with a general error message.

Unit Tests for Task Management API

The unit tests for the Task Management API are implemented using Jest and Supertest. These tests verify the functionality of various API endpoints, ensuring that they work as expected and handle errors properly. Below is an explanation of the key components and the tests themselves.

Test Setup

The test suite begins by setting up the necessary environment.

- Separate testing data base is used for the purpose of unit testing, not to disrupt the production DB.
- Switch the env from “development” to “testing” in knexfile.js. It will connect to the test DB through configurations

Each test suite uses beforeEach to handle any prerequisite setup and afterEach to clean up after each test. This includes:

- Logging in to get a JWT token for authenticated requests.
- Closing the server if it's already running.
- Inserting and deleting dummy tasks in a test database.

Test Suite

Here are the specific test cases and what they validate:

1. **Login and Authorization:**
    - Logs in a user to obtain a JWT token for authenticated requests.
2. **Post /api/tasks:**
    - Tests inserting a new task.
    - Verifies the status code and the response content.
3. **Patch /api/task/:id:**
    - Tests updating an existing task.
    - Verifies the status code and the updated task details.
4. **Delete /api/task/:id:**
    - Tests deleting a specific task.
    - Verifies the status code and the response content.
5. **Get /api/tasks:**
    - Tests retrieving all tasks for a user.
    - Verifies the status code and the retrieved tasks.
6. **Get /api/task/:id:**
    - Tests retrieving a specific task by its ID.
    - Verifies the status code and the retrieved task details.
7. **Error Handling - Authentication:**
    - Tests the behavior when a request is made without an authorization token.
    - Verifies the status code and the error message.
8. **Error Handling - URL Not Found:**
    - Tests the behavior when a request is made to an invalid URL.
    - Verifies the status code.

Security Precautions

To protect against common security threats such as SQL injection and cross-site scripting (XSS), the following precautions are implemented:

1. **SQL Injection:**
    - Using parameterized queries with Objection.js, an ORM for Node.js, which ensures that user input is safely incorporated into SQL queries.
2. **Cross-Site Scripting (XSS):**
    - Sanitizing user input using libraries such as `xss-clean` to remove any potentially malicious scripts.
3. **Authentication and Authorization:**
    - Using JSON Web Tokens (JWT) for authentication and ensuring secure transmission of tokens.
    - Validating JWT tokens on protected routes to ensure that only authorized users can access them.
4. **Input Validation:**
    - Validating incoming requests using Joi schemas to ensure the data is in the expected format and within the allowed constraints.
5. **Error Handling:**
    - Implementing a comprehensive error handler to handle different types of errors gracefully and securely without exposing sensitive information.

Minor Details about Database Configuration

The Task Management API uses PostgreSQL as the database system and is managed through DBeaver. The database schema is designed to work seamlessly with Objection.js, an ORM (Object-Relational Mapping) for Node.js applications.

Database Configuration:

- **Database System:** PostgreSQL
- **Database Management Tool:** DBeaver
- **ORM:** Objection.js
- **Directory Structure:** Models are organized in directories to maintain a clean and modular codebase.

Starting the Application:

Production Mode (Start):

- Use npm start command to start the application in production mode.
- This command typically runs the application using the entry point specified in the app field of your package.json file

Development Mode (Debug):

- Use npm run debug command to start the application in debug mode.
- This command uses nodemon to watch for file changes and automatically restarts the server when changes are detected.
- Debug output is enabled using the DEBUG environment variable, specifying the namespace for debug messages (app:task-management).

Testing the Application:

Unit Testing:

- Use npm run test command to run your unit tests.
- Jest is configured to run tests located in the specified test directory (Backend/src/test/*).
- You can use the --watchAll flag to watch for changes and re-run tests automatically during development.
- Jest provides verbose output (--verbose) to display detailed test results.

Note: For each condition, start by npm i, to install all the necessary packages