# bit-backend-assignment1

**Task Management API Documentation**

**Overview**

This API allows users to manage tasks through CRUD (Create, Read, Update, Delete) operations. Users must authenticate via token-based authentication to perform these operations. Additionally, permissions are implemented to ensure that users can only update or delete tasks they own.

**Base URL**

http://localhost:3000

**Task Model**

Fields

Title (CharField): Represents the title of the task.

Description (TextField): Describes the task in detail.

Due Date (DateField): Indicates the due date for the task.

Status (ChoiceField): Represents the status of the task. Options: Pending, In Progress, Completed.

**Authentication**

To use this API, users must authenticate through token-based authentication. This is achieved through the /register and /login endpoints.

**REQUESTS AND RESPONSES API’S**

**1. Register User**

Endpoint: POST /register/

**Request:**

POST http://localhost:3000/register/

Content-Type: application/json

{

"username": "your\_username",

"password": "your\_password"

}

**Response:**

200 OK: User created successfully.

400 Bad Request: User already exists or invalid password.

**2. Login**

Endpoint: POST /login/

**Request:**

POST http://localhost:3000/register/

Content-Type: application/json

{

"username": "your\_username",

"password": "your\_password"

}

**Response:**

200 OK: Successful login. Returns JWT token.

400 Bad Request: Invalid user or password.

**Endpoints**

**3. List All Tasks**

Endpoint: GET /tasks/

**Request:**

Include the JWT token in the Authorization header.

GET http://localhost:3000/tasks/

Authorization: Bearer <jwt-Token>

**Response:**

200 OK: List of all tasks.

**4. Retrieve a Single Task**

Endpoint: GET /tasks/:tasksId

**Request:**

Include the JWT token in the Authorization header.

GET http://localhost:3000/tasks/1

Authorization: Bearer <jwt-Token>

**Response:**

200 OK: Task details.

403 Forbidden: Permission denied. Task does not belong to the user.

**5. Create a New Task**

Endpoint: POST /tasks

**Request:**

POST http://localhost:3000/tasks

Content-Type: application/json

Authorization: Bearer <jwt-Token>

{

"title": "Task Title",

"description": "Task Description",

"due\_date": "YYYY-MM-DD",

"status": "Pending"

}

**Response:**

200 OK: Task inserted successfully.

**6. Update an Existing Task**

Endpoint: PUT /tasks/:tasksId

**Request:**

PUT http://localhost:3000/tasks/1

Content-Type: application/json

Authorization: Bearer <jwt-Token>

{

"title": "Updated Title",

"description": "Updated Description",

"due\_date": "YYYY-MM-DD",

"status": "Completed"

}

**Response:**

200 OK: Data updated successfully.

403 Forbidden: Permission denied. User does not own the task.

**7. Delete a Task**

Endpoint: DELETE /tasks/:tasksId

**Request:**

Include the JWT token in the Authorization header.

DELETE http://localhost:3000/tasks/2

Authorization: Bearer <jwt-Token>

**Response:**

200 OK: Task deleted successfully.

403 Forbidden: Permission denied. User does not own the task.

**Permissions**

Permissions are implemented to ensure that users can only update or delete tasks they own.

**Task Ownership Check**: Before updating or deleting a task, the API checks if the task belongs to the authenticated user.

**Error Handling**

400 Bad Request: Invalid request or user already exists.

401 Unauthorized: Invalid JWT token.

403 Forbidden: Permission denied or user not found.

500 Internal Server Error: Server error.

**Run the Application**

Use the following command to run your application:

Open a terminal in the root directory of your project and run the following command to install the required Node.js modules:

**In Terminal Type**

npm install express sqlite3 bcrypt jsonwebtoken nodejs

After Installation Of Above Modules

**In Terminal Type**

node app.js

Or, if you prefer nodemon for automatic server restarts during development:

**In Terminal Type**

npm install -g nodemon

nodemon app.js

Now, your server should be running on http://localhost:3000.

**Execute HTTP Requests**

Open the app.http file.

Place the cursor within a request block (e.g., "Register User").

Click the "Send Request" link that appears above the request to execute it.

Repeat these steps for each request in the app.http file

**Project Flow**

**Database Initialization:**

Created a SQLite database named todoTasks.db with tables users and tasks.

**USERS TABLE SCHEMA CREATION**

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT NOT NULL,

password TEXT NOT NULL

);

**TASKS TABLE SCHEMA CREATION**

CREATE TABLE IF NOT EXISTS tasks (

id INTEGER PRIMARY KEY AUTOINCREMENT,

title TEXT NOT NULL,

description TEXT,

due\_date DATE,

status TEXT,

user\_id INTEGER,

FOREIGN KEY (user\_id) REFERENCES users (id)

);

**User Registration:**

Register a user by making a POST request to /register/ with a unique username and password.Make a request to the server that I have mention above **1. Register User**

**User Login:**

Log in by making a POST request to /login/ with the registered username and password.

Make a request to the server that I have mention above **2. Login**

**Create New Task:**

Create a new task with a POST request to /tasks/ providing task details and JWT token.

Make a request to the server that I have mention above **5. Create a New Task**

**Task Listing:**

Get a list of all tasks by executing a GET request to /tasks/ with the JWT token.

Make a request to the server that I have mention above **3. List All Tasks**

**Retrieve Task:**

Retrieve a specific task by making a GET request to /tasks/:taskId with the JWT token.

Make a request to the server that I have mention above **4. Retrieve a Single Task**

**Update Task:**

Update an existing task via a PUT request to /tasks/:taskId with updated details and JWT token.Make a request to the server that I have mention above **6. Update an Existing Task**

**Delete Task:**

Delete a task with a DELETE request to /tasks/:taskId and JWT token.Make a request to the server that I have mention above **7. Delete a Task**

**Permissions:**

Ensure users can only modify tasks they own; permissions are automatically handled using user\_id in tasks tables that has linked with users table.

**Error Handling:**

Clear error messages for scenarios like invalid requests, authentication issues, permission denial, or internal server errors.

**Conclusion:**

**Follow these steps to interact seamlessly with the Task Management System, maintaining secure user authentication and authorization through token-based mechanisms.**