



Task Management and Report Subscription System Report

Name: Arwa Walid Shaker Hanafy

Table of Contents

1. System Components.....	(3)
2. System Architecture Diagram.....	(4)
3. Project Setup.....	(4)
4. Project Structure	(6)
5. Relational Database.....	(6)
6. Use case diagram.....	(8)
7. User Authentication	
a. Sign up API testing steps and examples.....	(9)
b. Sign in API testing steps and examples.....	(10)
8. Task Management API	
a. Create task API Testing Steps and examples.....	(12)
b. Retrieve task API Testing Steps and examples.....	(15)
c. Update task API Testing Steps and examples.....	(18)
d. Delete task API Testing Steps and examples.....	(21)
e. Batch delete tasks API Testing Steps and examples.....	(22)
f. Restore task API Testing Steps and examples.....	(24)
9. Subscription API	
a. Subscribe API Testing Steps and examples.....	(25)
b. Unsubscribe API Testing Steps and examples.....	(28)
10. Report Generation	
a. Automated Task Report	
Implementation.....	(29)
b. Report Generation test cases.....	(30)

System Components

1. Backend (API Server):

Description: The backend serves as the core logic of the project, responsible for processing incoming requests from clients and sending appropriate responses. It handles essential business functionalities, including user authentication, task management, and subscription handling.

Technology Used: Flask, a lightweight web framework, is utilized for routing and implementing the API logic, making it efficient for handling HTTP requests and responses.

2. Hosting:

Description: The hosting environment is where the application and database are deployed, allowing the system to run and be accessed. During development, the application is hosted locally for testing purposes.

Technology Used: Local hosting, using PyCharm as the integrated development environment (IDE) to run the app on the local machine. In this setup, the server is accessible via the IP address 127.0.0.1 and port 5000.

3. Database:

Description: The database is used to store structured data such as user details, task information, and subscription preferences. It ensures that the data is securely stored and easily retrievable when needed.

Technology Used: SQLite, a lightweight and file-based database, is chosen for local development, providing an efficient and easy-to-set-up solution for handling data.

Justification for ORM: SQLAlchemy is used to simplify database operations by abstracting raw SQL queries into Python code. It enhances maintainability, ensures secure interactions with the database, and provides flexibility to switch databases for production use.

4. Client:

Description: The client serves as the interface that allows users to interact with the application. It is responsible for sending requests to the backend and receiving responses, enabling users to access the application's functionalities.

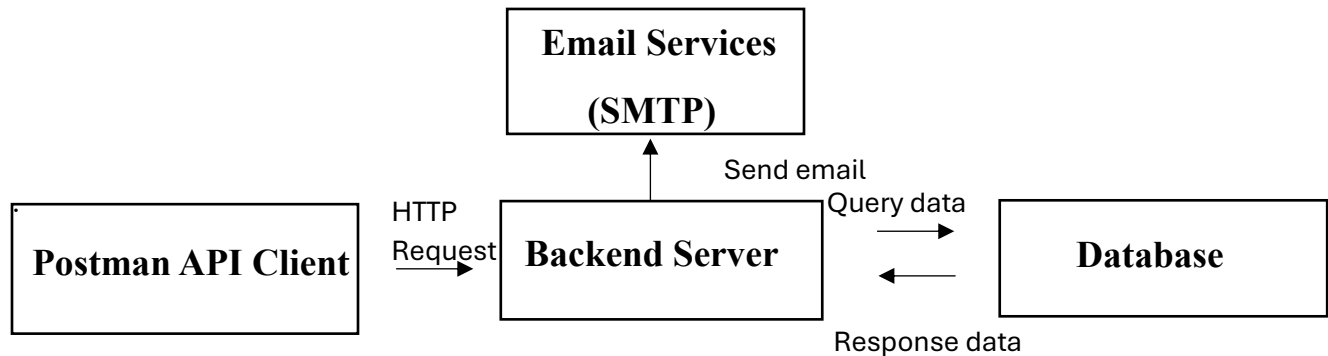
Technology Used: Postman, an API client, is used to send HTTP requests to the backend and test the application's endpoints to ensure they respond correctly.

5. Email Service:

Description: The email service handles the functionality of sending email reports to users based on their subscription settings. It automates the delivery of periodic task summaries (daily, weekly, monthly) as formatted HTML emails.

Technology Used: SMTP (The Simple Mail Transfer Protocol) is used to configure the application for email sending and App-Specific Password uses a secure app-specific password setup via the email provider's settings to authenticate the email-sending process securely.

System Architecture Diagram:



Project Setup

1. Download PyCharm:

- PyCharm was downloaded from [here](#) and installed following the standard installation process.

2. Project Folder and Files:

- A new folder was created to store all project files, and the necessary task files were established.

3. Install Required Libraries:

- The following libraries were installed to run the project:
 - pip install Flask: The main backend framework for building the web application.
 - pip install Flask-JWT-Extended: Used for handling **JWT (JSON Web Tokens)** for user authentication.
 - pip install Flask-SQLAlchemy: Facilitates interaction with the **SQLite** database, simplifying database operations.
 - pip install Flask-Bcrypt: Provides password hashing to securely store user passwords.

- pip install Flask-Mail: Allows sending email reports to users based on their subscription settings.
- pip install Flask-SQLAlchemy: Facilitates interaction with the SQLite database by abstracting raw SQL queries into Python code, simplifying database operations and enhancing maintainability.

4. **Create Virtual Environment:**

- PyCharm was used to create a virtual environment, isolating the project's dependencies and ensuring the application functions correctly in the development environment. The application runs on a local server using Flask, with all dependencies managed within the virtual environment.

5. **Write and Run Code:**

- The code was written and tested using Flask. The application was run locally on the Flask server.

6. **Database Setup:**

- SQLite was downloaded and connected to the project. PyCharm's database integration tools allowed easy access to the database to check and modify data directly within the IDE.

7. **Database Initialization:**

- The database schema was defined in the model.py file, and the db.py file was used to initialize database-related extensions such as SQLAlchemy, Bcrypt, and JWTManager.
- A config.py file was created to store the database URI (database.db) and other settings like the JWT secret key.
- The int_db.py script was created to run the db.create_all() command, which initializes the database and creates the tables defined in the models.

8. **API Testing:**

- Postman was downloaded and used for API testing to ensure all endpoints were functioning correctly.

9. **Email Service Setup:**

The application uses Flask-Mail for managing email functionality. The SMTP server details and credentials are configured in the config.py file. An app-specific password is generated and used to secure the email-sending process.

Steps Taken:

1. Set up an email account specifically for the application
2. Enabled SMTP access for the account.

3. Generated an app-specific password for authentication.
4. Added the following settings in config.py

```
app.config.update(  
    MAIL_SERVER='smtp.gmail.com',  
    MAIL_PORT=587,  
    MAIL_USE_TLS=True,  
    MAIL_USERNAME='vodafonetaskreports@gmail.com',  
    MAIL_PASSWORD='cspj tkab dsie xjgk'
```

Project Structure

1. app.py => Main application file (entry point)
2. config.py => Configuration Settings
3. db.py => Initializes SQLAlchemy, Bcrypt, and JWT
4. model.py => Defines database models
5. int_db.py => Initializes the databas
6. check_user.py => Check if a specific user exists in the database
7. auth.py => Handles user authentication and authorization

Relational Database

Relationship between tables:

User - Subscription: One-to-one

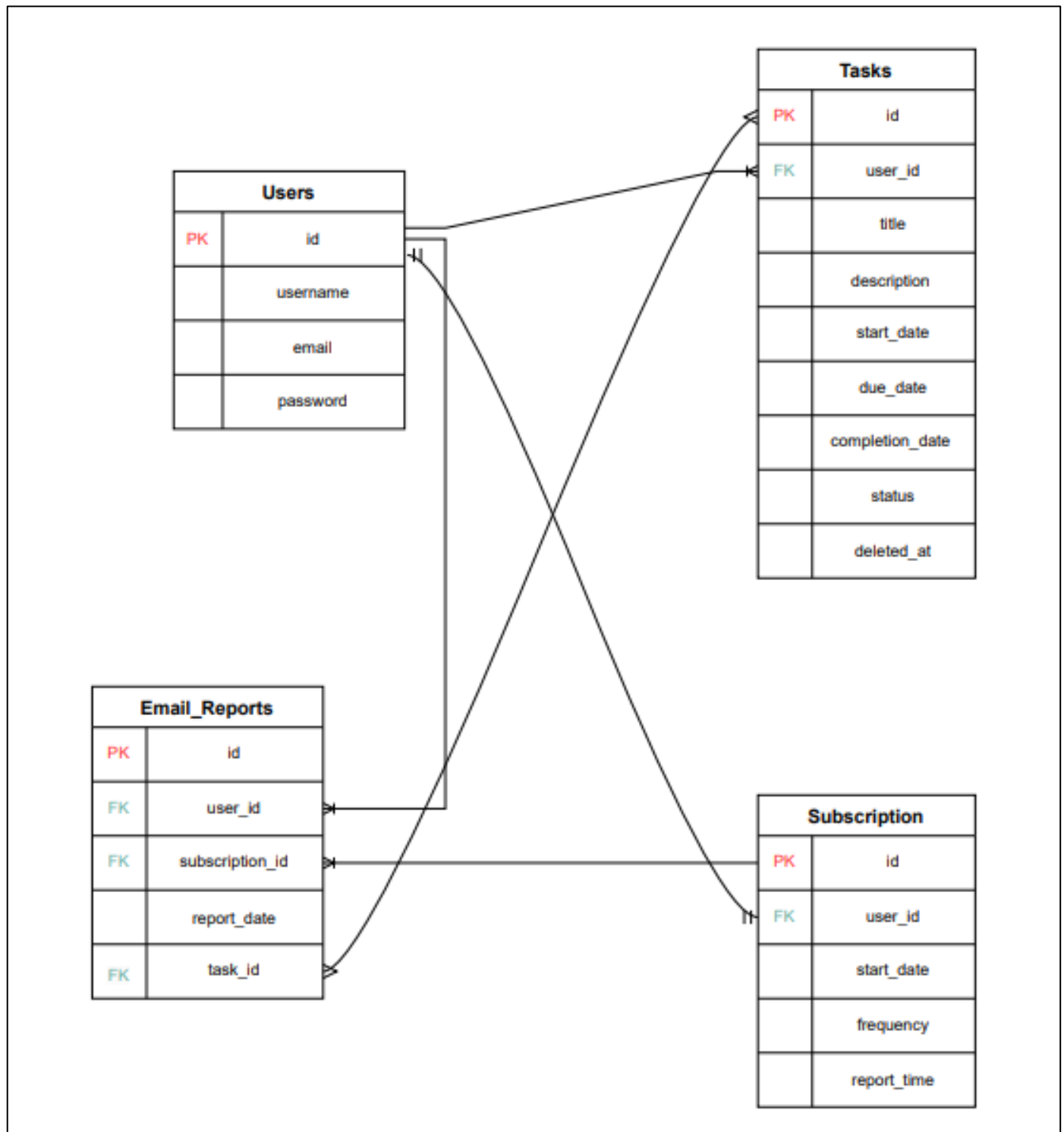
User -Tasks: One-to-many

User -Email_Reports: One-to-many.

Subscription – Email_Reports: One-to-many

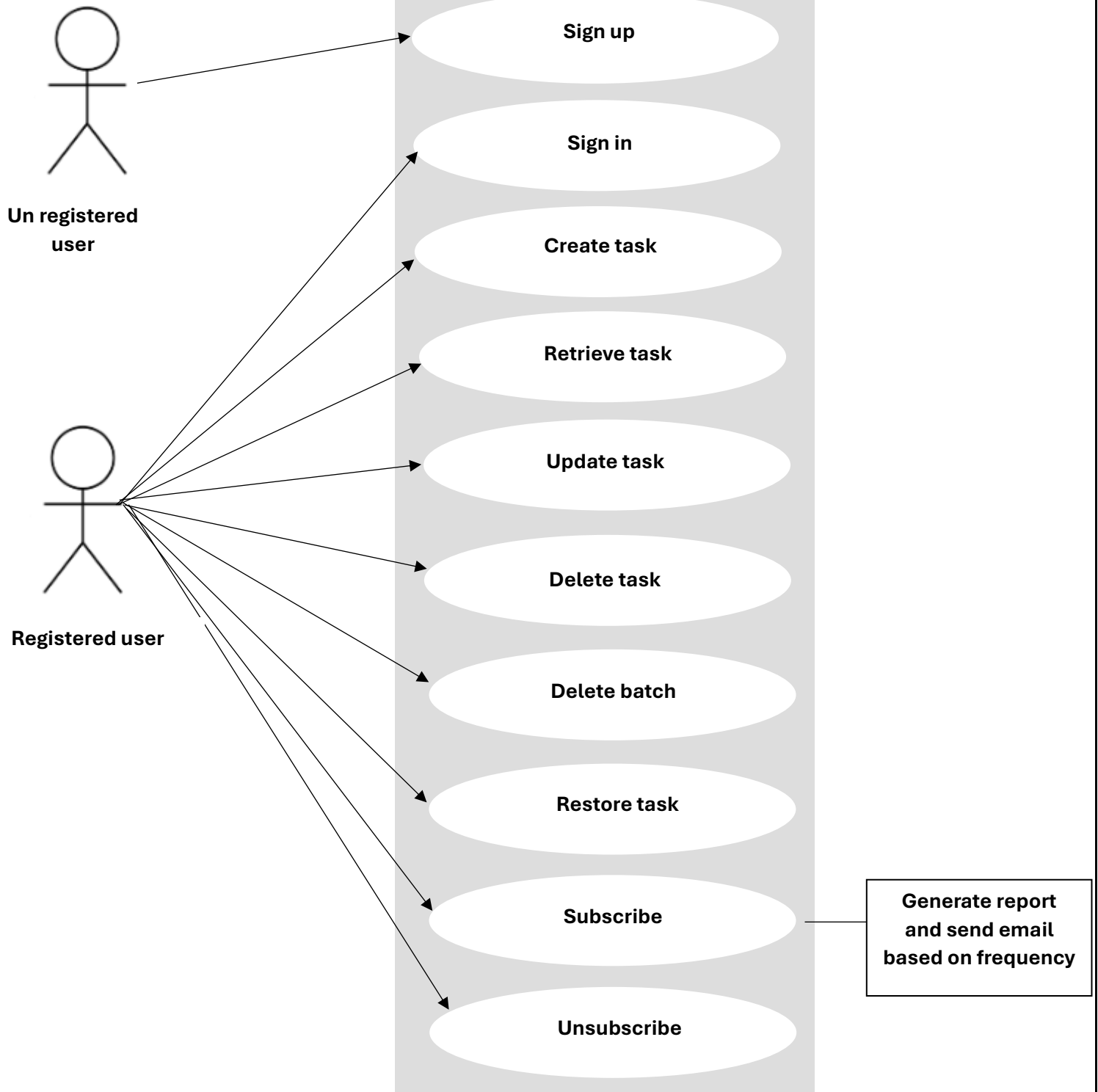
Task – Email_Report: many-to-many

Database Diagram:



Use Case Diagram

Task Management and Report Subscription System



User Authentication

Sign-up API Testing Steps:

1. Open Postman
2. Select POST request type
3. Enter endpoint url: <http://localhost:5000/signup>
4. Navigate to the Body tab then select raw data type and choose JSON format
5. Enter JSON data in the body

```
{  
  "username": "user",  
  "email": "user@example.com",  
  "password": "Password"  
}
```

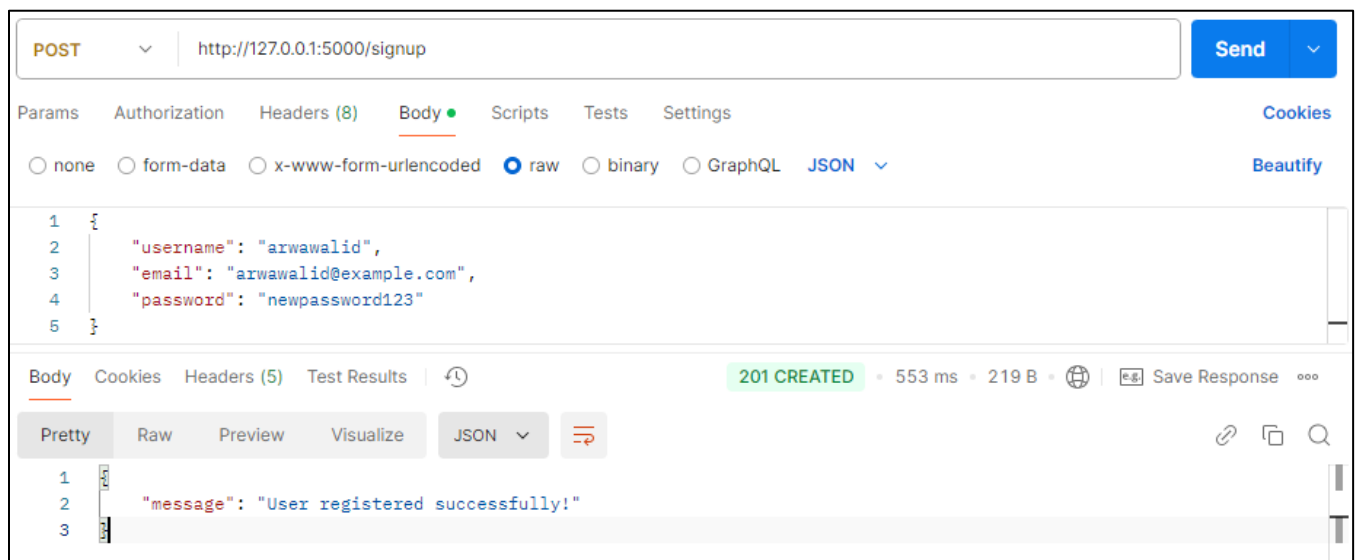
6. Send the Request
7. Verify the Response

For successful registration, the response should include the message "User registered successfully!" with a 201 status code.

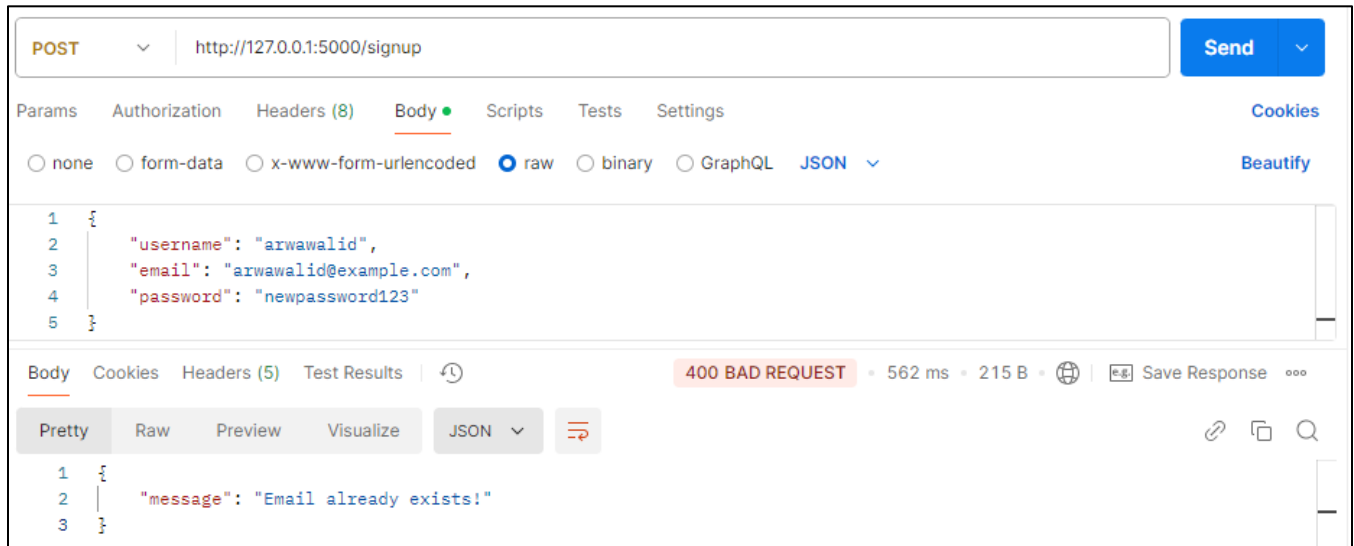
If the user is already registered, the response should return a message like "Email already exists!" with a 400 status code.

Examples:

1. For successful registration



2. If the user is already registered



Sign-in API Testing Steps:

1. Open Postman
2. Select POST request type
3. Enter endpoint url: <http://127.0.0.1:5000/signin>
4. Navigate to the Body tab then select raw data type and choose JSON format
5. Enter JSON data in the body

```
{
  "username": "user",
  "email": "user@example.com",
  "password": "Password"
}
```

6. Send the Request
7. Verify the Response

For successful sign-in, the response should include the message "Signin successful!" and an `access_token` (JWT) generated for the user with a 200 status code.

For invalid credentials, the response should return the message "Invalid credentials!" with a 401 status code.

Examples:

1. For successful sign-in

POST http://127.0.0.1:5000/signin

Params Authorization Headers (8) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "username": "arwa",
3   "email": "arwa@example.com",
4   "password": "newpassword123"
5 }
```

Body Cookies Headers (5) Test Results 200 OK • 1.15 s • 551 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsIm1hdCI6MTczMzU4NDY0MCwianRpIjoibDhjZjZiYWQtY2QzMzU0ODQ2NDAsImNzcmYiOiI2ZmQ4YmZiNC1jMmNjLTQ2NDAtOGU0Ni1iNzI6YjE5YjE3NzAiLCJleHAiOiE3MzM1ODU1NDB9.b74f60Cc2CdGtscqwmZxG3mOqikFMnqhW0gcGtr1IBU",
3   "message": "Signin successful!"
4 }
```

2. For invalid credentials

POST http://127.0.0.1:5000/signin

Params Authorization Headers (8) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "username": "arwa",
3   "email": "arwa1@example.com",
4   "password": "newpassword123"
5 }
```

Body Cookies Headers (5) Test Results 401 UNAUTHORIZED • 15 ms • 215 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Invalid credentials!"
3 }
```

Task Management API

Create task API Testing Steps:

1. Open Postman
2. Select POST request type
3. Enter endpoint url: <http://127.0.0.1:5000/createtask>
4. **Navigate to the Authorization tab** and select the "Bearer Token" type
5. **Enter the Access Token** in the "Token" field
6. Navigate to the Body tab then select raw data type and choose JSON format
7. Enter JSON data in the body

```
{
  "title" : "Task",
  "description" : "TaskDescription",
  "start_date" : "2024-12-10T17:30:00Z",
  "due_date" : "2024-12-10T18:05:00Z",
  "completion_date" : "2024-12-11T15:30:00Z",
  "status": "Pending"
}
```

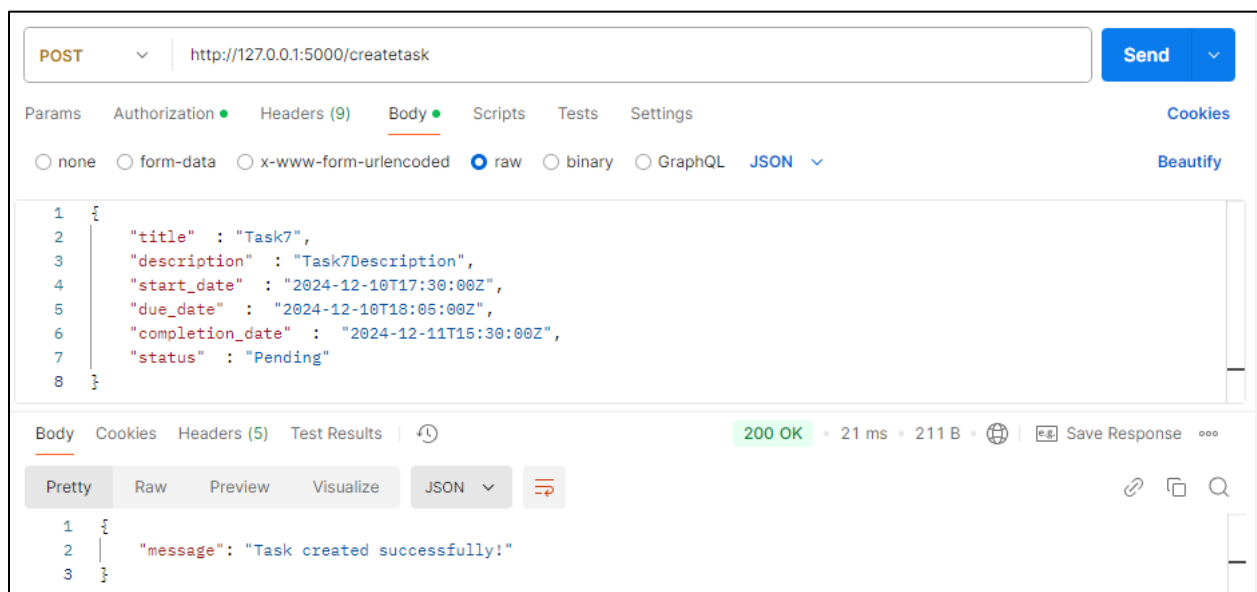
8. Send the Request
9. Verify the Response

For successful task creation, the response should include the message "Task is successfully created" with a 200 status code.

If there's any issue (ex: one of the validations isn't satisfied.), the response should return appropriate error message

Examples:

1. For successful task creation



2. Validations

1. Status is in the allowed statuses ('Pending', 'Completed', 'Overdue')

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:5000/createtask`. The request body is a JSON object:

```
1 {
2   "title" : "Task7",
3   "description" : "Task7Description",
4   "start_date" : "2024-12-10T17:30:00Z",
5   "due_date" : "2024-12-10T18:05:00Z",
6   "completion_date" : "2024-12-11T15:30:00Z",
7   "status" : "Pendig"
8 }
```

The response is a 400 BAD REQUEST with a message: `"message": "Invalid status. Allowed values are Pending, Completed, Overdue."`

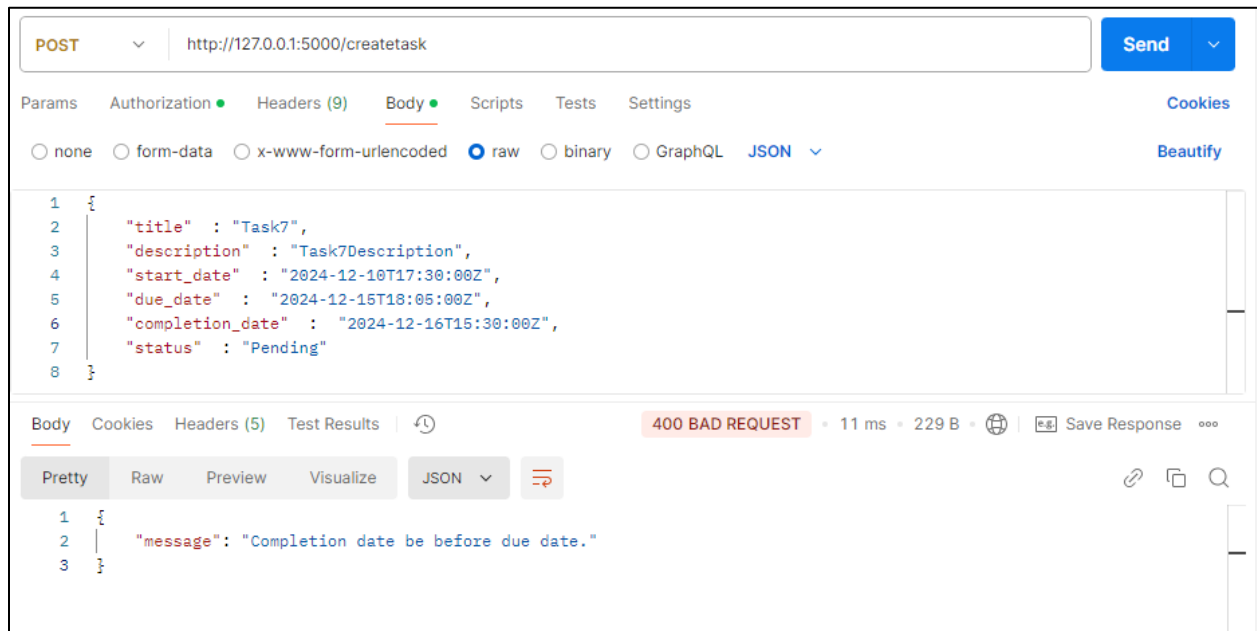
2. Start Date Must Be Earlier Than Due Date

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:5000/createtask`. The request body is a JSON object:

```
1 {
2   "title" : "Task7",
3   "description" : "Task7Description",
4   "start_date" : "2024-12-10T17:30:00Z",
5   "due_date" : "2024-12-10T16:05:00Z",
6   "completion_date" : "2024-12-11T15:30:00Z",
7   "status" : "Pending"
8 }
```

The response is a 400 BAD REQUEST with a message: `"message": "Start date must be before due date."`

3. Start Date Must Be Earlier Than Completion Date, and Completion Date Must Be Earlier Than Due Date (Validate that start date < Completion date < due date)



POST http://127.0.0.1:5000/createtask

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

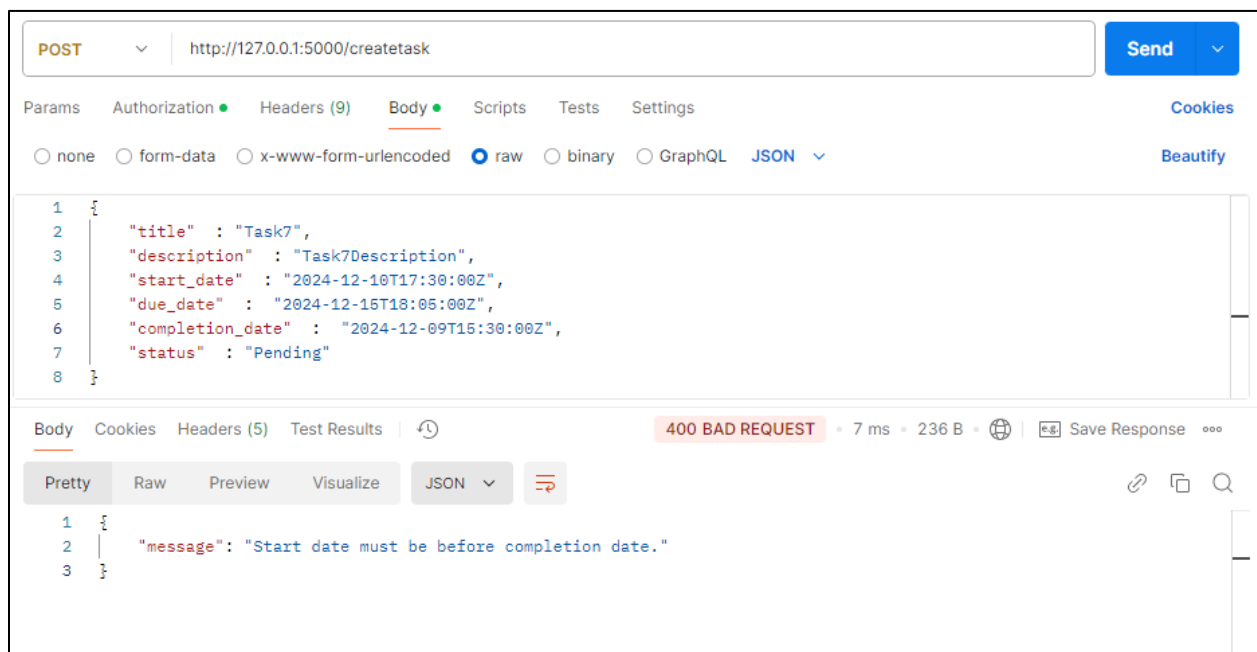
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "title" : "Task7",
3   "description" : "Task7Description",
4   "start_date" : "2024-12-10T17:30:00Z",
5   "due_date" : "2024-12-15T18:05:00Z",
6   "completion_date" : "2024-12-16T15:30:00Z",
7   "status" : "Pending"
8 }
```

Body Cookies Headers (5) Test Results 400 BAD REQUEST 11 ms 229 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Completion date be before due date."
3 }
```



POST http://127.0.0.1:5000/createtask

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

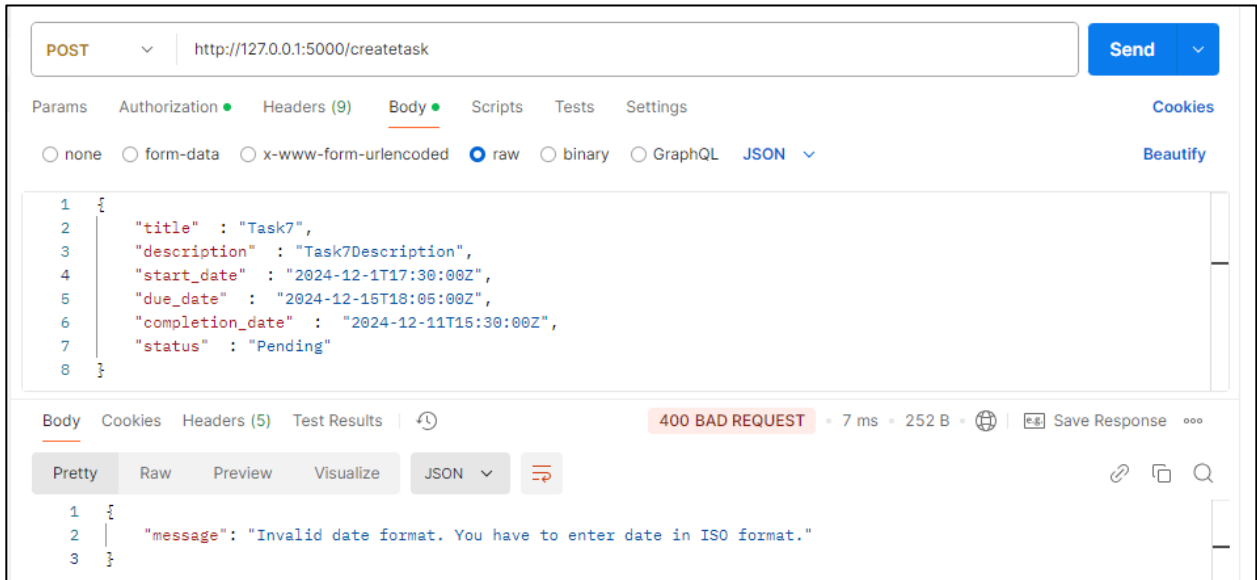
```
1 {
2   "title" : "Task7",
3   "description" : "Task7Description",
4   "start_date" : "2024-12-10T17:30:00Z",
5   "due_date" : "2024-12-15T18:05:00Z",
6   "completion_date" : "2024-12-09T15:30:00Z",
7   "status" : "Pending"
8 }
```

Body Cookies Headers (5) Test Results 400 BAD REQUEST 7 ms 236 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Start date must be before completion date."
3 }
```

4. Date Must Be in a Valid Format



Retrieve tasks API Testing Steps:

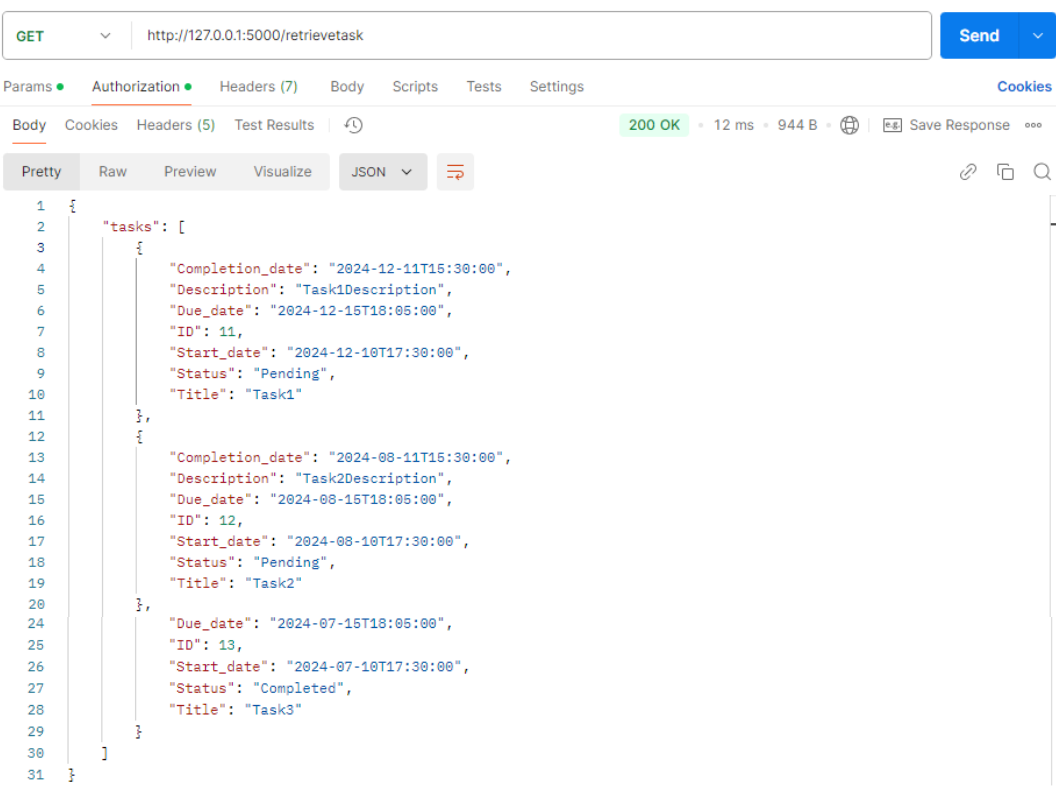
1. Open Postman
2. Select GET request type
3. Enter endpoint url: <http://127.0.0.1:5000/retrievetask>
4. Navigate to the Authorization tab and select the "Bearer Token" type
5. Enter the Access Token in the "Token" field
6. If you want to filter tasks based on status, start_date_range, or end_date_range, go to the Params tab and add the following parameters:
status: Pending
start_date_range: 2024-01-01T00:00:00
end_date_range: 2024-12-31T23:59:59
7. Send the Request
8. Verify the Response

Retrieve tasks successfully (With or without filters), the response should return a list of tasks with a 200 status code (OK).

If no tasks are found or no tasks match the criteria, you should receive a 404 status code (Not Found) with a message: "No tasks found"

Examples:

1. Retrieve All Tasks Without Applying Any Filters



GET http://127.0.0.1:5000/retrievetask

Send

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

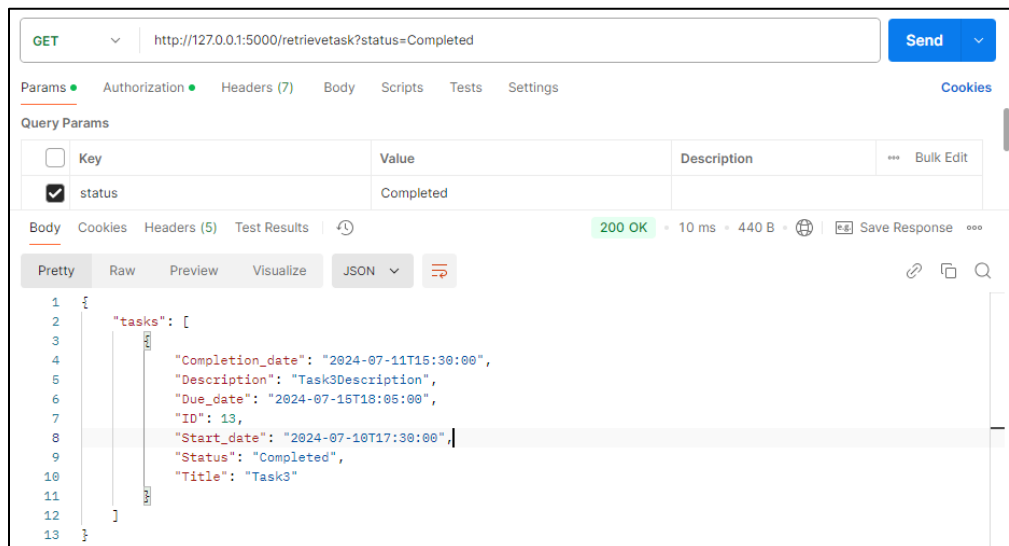
Body Cookies Headers (5) Test Results 200 OK 12 ms 944 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "tasks": [
3     {
4       "Completion_date": "2024-12-11T15:30:00",
5       "Description": "Task1Description",
6       "Due_date": "2024-12-15T18:05:00",
7       "ID": 11,
8       "Start_date": "2024-12-10T17:30:00",
9       "Status": "Pending",
10      "Title": "Task1"
11    },
12    {
13      "Completion_date": "2024-08-11T15:30:00",
14      "Description": "Task2Description",
15      "Due_date": "2024-08-15T18:05:00",
16      "ID": 12,
17      "Start_date": "2024-08-10T17:30:00",
18      "Status": "Pending",
19      "Title": "Task2"
20    },
21    {
22      "Due_date": "2024-07-15T18:05:00",
23      "ID": 13,
24      "Start_date": "2024-07-10T17:30:00",
25      "Status": "Completed",
26      "Title": "Task3"
27    }
28  ]
29 }
30
31 }
```

2. Validations

1. Retrieve Tasks Filtered by Status (ex: Completed)



GET http://127.0.0.1:5000/retrievetask?status=Completed

Send

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> status	Completed		

Body Cookies Headers (5) Test Results 200 OK 10 ms 440 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "tasks": [
3     {
4       "Completion_date": "2024-07-11T15:30:00",
5       "Description": "Task3Description",
6       "Due_date": "2024-07-15T18:05:00",
7       "ID": 13,
8       "Start_date": "2024-07-10T17:30:00",
9       "Status": "Completed",
10      "Title": "Task3"
11    }
12  ]
13 }
```


2. Retrieve Tasks Within a Specific Time Range

ex: start date : 2024-12-08T17:30:00 , end date : 2024-12-20T18:05:00

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:5000/retrievetask?start_date_range=2024-12-08T17:30:00&end_date_range=2024-12-20T18:05:00`. The response is a 200 OK status with a JSON body containing a single task.

Query Params

Key	Value	Description
<input type="checkbox"/> status	Completed	
<input checked="" type="checkbox"/> start_date_range	2024-12-08T17:30:00	
<input checked="" type="checkbox"/> end_date_range	2024-12-20T18:05:00	

Body (JSON)

```
{
  "tasks": [
    {
      "Completion_date": "2024-12-11T15:30:00",
      "Description": "Task1Description",
      "Due_date": "2024-12-15T18:05:00",
      "ID": 11,
      "Start_date": "2024-12-10T17:30:00",
      "Status": "Pending",
      "Title": "Task1"
    }
  ]
}
```

3. No tasks found or no tasks matches criteria

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:5000/retrievetask?status=Overdue&start_date_range=2024-12-08T17:30:00&end_date_range=2024-12-20T18:05:00`. The response is a 404 NOT FOUND status with a JSON body containing a message.

Query Params

Key	Value	Description
<input checked="" type="checkbox"/> status	Overdue	
<input checked="" type="checkbox"/> start_date_range	2024-12-08T17:30:00	
<input checked="" type="checkbox"/> end_date_range	2024-12-20T18:05:00	

Body (JSON)

```
{
  "message": "No tasks found"
}
```

Update tasks API Testing Steps:

1. Open Postman
2. Select PUT request type
3. Enter endpoint url: <http://127.0.0.1:5000/updatetask/13>
*note: we can replace 13 with the task we want to update
4. Navigate to the Authorization tab and select the "Bearer Token" type
5. Enter the Access Token in the "Token" field
6. Navigate to the Body tab then select raw data type and choose JSON format
7. Enter JSON data in the body

```
{
  "title" : "updated Task",
  "description" : "updated TaskDescription",
  "start_date" : "2024-11-10T17:30:00Z",
  "due_date" : "2024-11-10T18:05:00Z",
  "completion_date" : "2024-11-11T15:30:00Z",
  "status": "Pending"
} # if any field not given will remain same as in database
```

8. Send the Request

9. Verify the Response

If the task is updated successfully, you should receive a response with a 200 status code

If the task does not exist, you should receive a 404 status code

If there's any issue (ex: one of the validations isn't satisfied.), the response should return appropriate error message

Examples:

1. Task is updated successfully

Task 1 for user with user id = 1 before any updates:

id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Task1	Task1Description	2024-12-06 15:30:00.000000	2024-12-08 15:30:00.000000	2024-12-07 15:30:00.000000	pending	NULL

Send update request

The screenshot shows the Postman interface for a PUT request. The URL is `http://127.0.0.1:5000/updatetask/1`. The request body is a JSON object: `{ "title": "Task2 axwa", "status": "Overdue", "start_date": "2024-12-05T17:30:00Z" }`. The response is a 200 OK status with a message: `{ "message": "Task updated successfully!!" }`.

Task 1 Updated in the Database

id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Task2 arwa	Task1Description	2024-12-05 17:30:00.000000	2024-12-08 15:30:00.000000	2024-12-07 15:30:00.000000	Overdue	NULL

2. Task not found

PUT ▼ http://127.0.0.1:5000/updatesetask/91 Send ▼

Params Authorization ● Headers (9) Body ● Scripts Tests Settings Cookies

Body Cookies Headers (5) Test Results ↺ 404 NOT FOUND • 10 ms • 207 B • 🌐 📄 Save Response ⋮

Pretty Raw Preview Visualize JSON ▼ ↻

```
1 {
2   "message": "Task not found."
3 }
```

3. Validations

1. Updated status is in the allowed statuses ('Pending', 'Completed', 'Overdue')

PUT ▼ http://127.0.0.1:5000/updatesetask/1 Send ▼

Params Authorization ● Headers (9) Body ● Scripts Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼ Beautify

```
1 {
2   "title" : "Task2 arwa",
3   "status" : "Overdue",
4   "start_date" : "2024-12-05T17:30:00Z"
5 }
6 }
```

Body Cookies Headers (5) Test Results ↺ 400 BAD REQUEST • 10 ms • 257 B • 🌐 📄 Save Response ⋮

Pretty Raw Preview Visualize JSON ▼ ↻

```
1 {
2   "message": "Invalid status. Allowed values are Pending, Completed, Overdue."
3 }
```

2. Start Date Must Be Earlier Than Due Date

PUT ▼ http://127.0.0.1:5000/updatesetask/1 Send ▼

Params Authorization ● Headers (9) Body ● Scripts Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼ Beautify

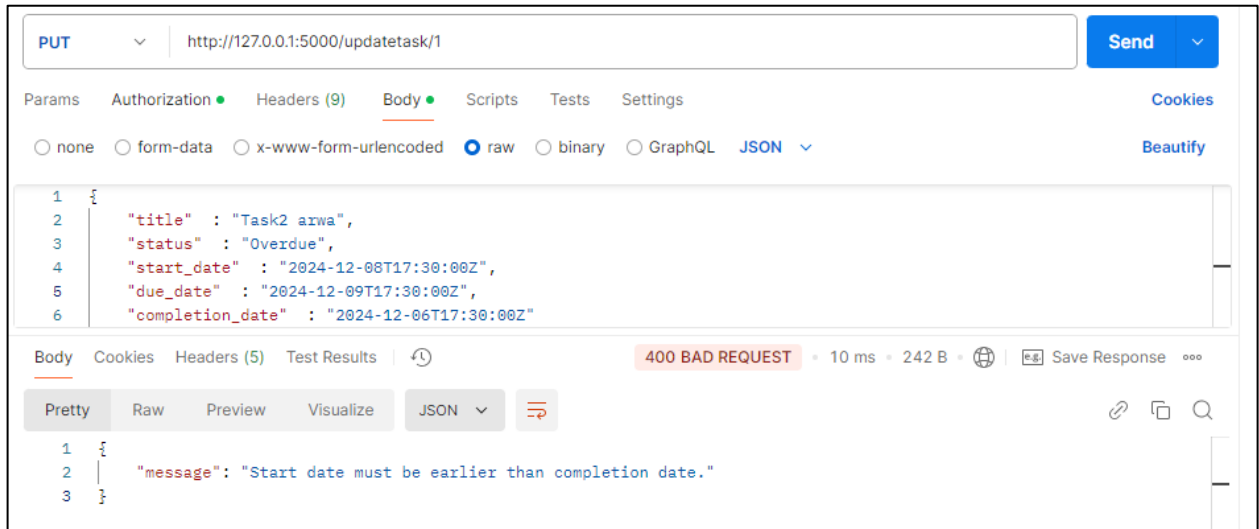
```
1 {
2   "title" : "Task2 arwa",
3   "status" : "Overdue",
4   "start_date" : "2024-12-05T17:30:00Z",
5   "due_date" : "2024-12-04T17:30:00Z"
6 }
```

Body Cookies Headers (5) Test Results ↺ 400 BAD REQUEST • 11 ms • 235 B • 🌐 📄 Save Response ⋮

Pretty Raw Preview Visualize JSON ▼ ↻

```
1 {
2   "message": "Start date must be earlier than due date."
3 }
```

3. Start Date Must Be Earlier Than Completion Date, and Completion Date Must Be Earlier Than Due Date (Validate that start date < Completion date < due date)



PUT ▼ http://127.0.0.1:5000/updatetask/1 Send ▼

Params Authorization ● Headers (9) **Body** ● Scripts Tests Settings Cookies Beautify

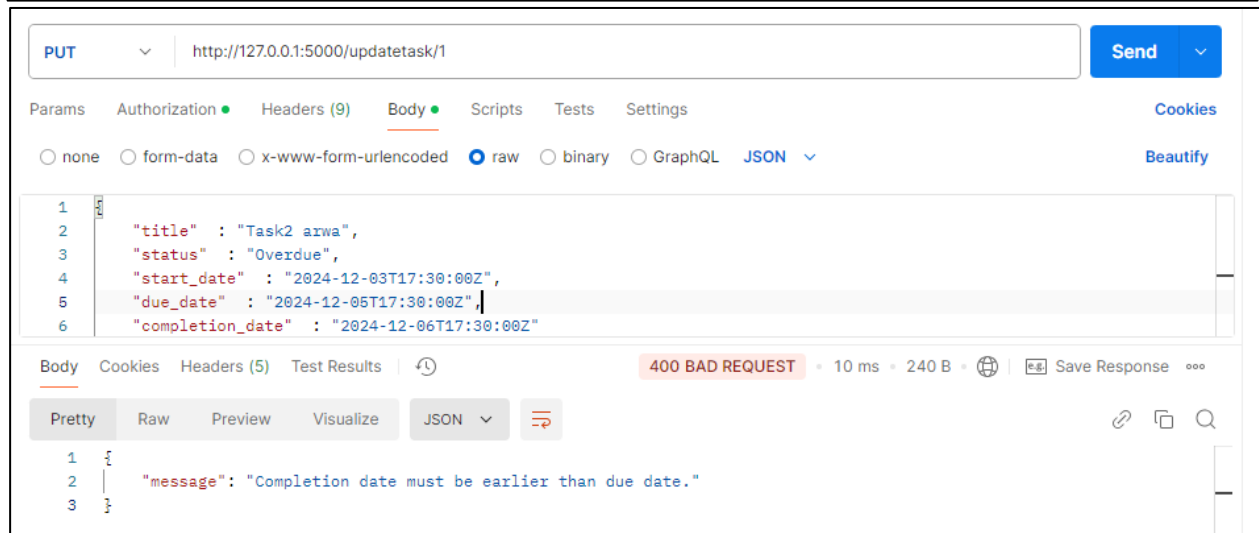
☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

```
1 {
2   "title" : "Task2 arwa",
3   "status" : "Overdue",
4   "start_date" : "2024-12-08T17:30:00Z",
5   "due_date" : "2024-12-09T17:30:00Z",
6   "completion_date" : "2024-12-06T17:30:00Z"
```

Body Cookies Headers (5) Test Results 🕒 **400 BAD REQUEST** • 10 ms • 242 B • 🌐 💾 Save Response ⋮

Pretty Raw Preview Visualize JSON ▼ 🔗 📄 🔍

```
1 {
2   "message": "Start date must be earlier than completion date."
3 }
```



PUT ▼ http://127.0.0.1:5000/updatetask/1 Send ▼

Params Authorization ● Headers (9) **Body** ● Scripts Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

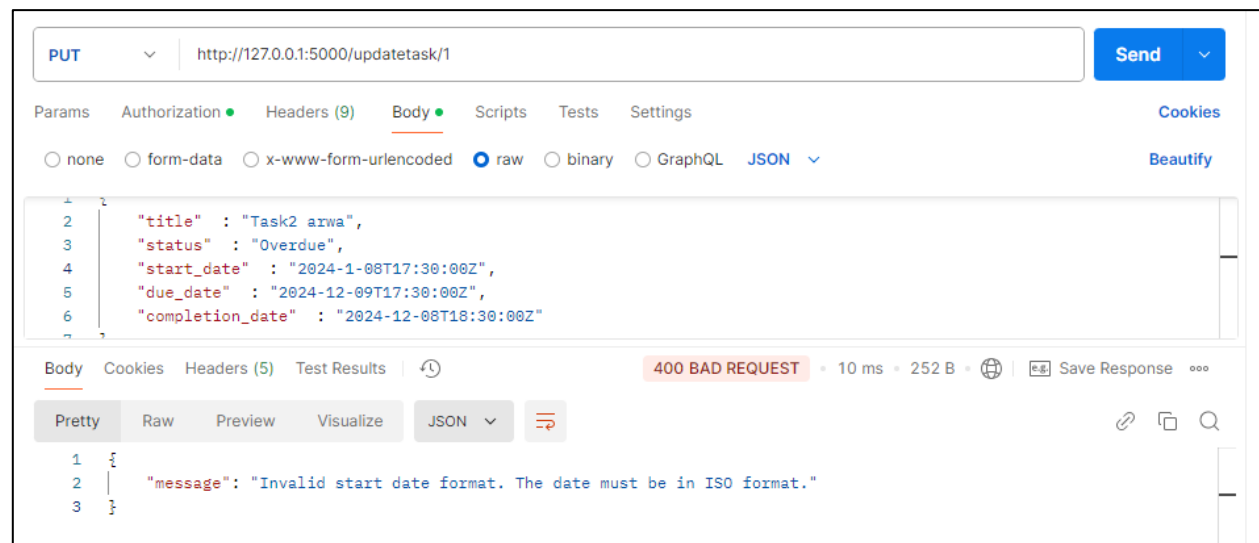
```
1 {
2   "title" : "Task2 arwa",
3   "status" : "Overdue",
4   "start_date" : "2024-12-03T17:30:00Z",
5   "due_date" : "2024-12-05T17:30:00Z",
6   "completion_date" : "2024-12-06T17:30:00Z"
```

Body Cookies Headers (5) Test Results 🕒 **400 BAD REQUEST** • 10 ms • 240 B • 🌐 💾 Save Response ⋮

Pretty Raw Preview Visualize JSON ▼ 🔗 📄 🔍

```
1 {
2   "message": "Completion date must be earlier than due date."
3 }
```

4. Date Must Be in a Valid Format



PUT ▼ http://127.0.0.1:5000/updatetask/1 Send ▼

Params Authorization ● Headers (9) **Body** ● Scripts Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

```
1 {
2   "title" : "Task2 arwa",
3   "status" : "Overdue",
4   "start_date" : "2024-1-08T17:30:00Z",
5   "due_date" : "2024-12-09T17:30:00Z",
6   "completion_date" : "2024-12-08T18:30:00Z"
```

Body Cookies Headers (5) Test Results 🕒 **400 BAD REQUEST** • 10 ms • 252 B • 🌐 💾 Save Response ⋮

Pretty Raw Preview Visualize JSON ▼ 🔗 📄 🔍

```
1 {
2   "message": "Invalid start date format. The date must be in ISO format."
3 }
```

Delete tasks API Testing Steps:

1. Open Postman
2. Select DELETE request type
3. Enter endpoint url: <http://127.0.0.1:5000/deletetask/1>
*note: we can replace 1 with the task we want to delete
4. Navigate to the Authorization tab and select the "Bearer Token" type
5. Enter the Access Token in the "Token" field
6. Send the Request
7. Verify the Response

If the task is deleted successfully, you should receive a response with a 200 OK status code

If the task does not found, you should receive a 404 Not found status code

Examples:

1. task is deleted successfully

Task 1 for user id 1 in the database where deleted at is null

id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Task2 arwa	Task1Description	2024-12-08 17:30:00.000000	2024-12-09 17:30:00.000000	2024-12-08 18:30:00.000000	Overdue	NULL

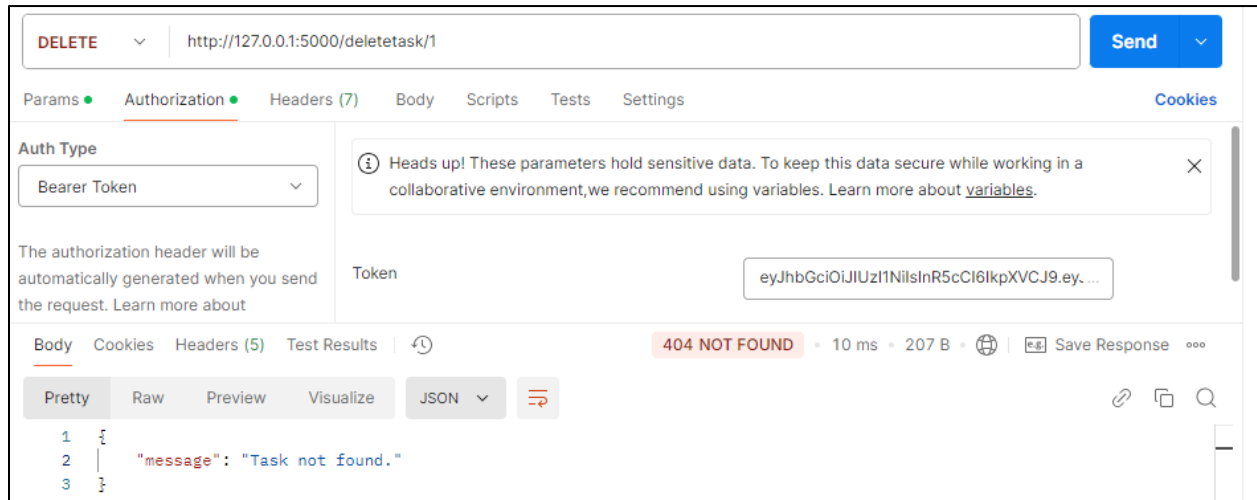
Send delete request

The screenshot shows the Postman interface for a DELETE request. The URL is `http://127.0.0.1:5000/deletetask/1`. The Authorization tab is selected, showing 'Bearer Token' as the auth type and a token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...`. The response is `200 OK` with a status bar indicating `18 ms` and `211 B`. The response body is shown in JSON format: `{ "message": "Task deleted successfully!" }`.

Task 1 for user id 1 in the database where deleted at is set to the current time

id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Task2 arwa	Task1Description	2024-12-08 17:30:00.000000	2024-12-09 17:30:00.000000	2024-12-08 18:30:00.000000	Overdue	2024-12-08 09:18:34.945851

2. Task does not found where it is Already Deleted (Deleted At is not null)



Batch delete tasks API Testing Steps:

1. Open Postman
2. Select DELETE request type
3. Enter endpoint url: http://127.0.0.1:5000/batchdelete?start_datetime_range=2024-07-09T17:30:00Z&due_datetime_range=2024-07-21T17:30:00Z

*note: we can change start and due date range.

4. Navigate to the Authorization tab and select the "Bearer Token" type
5. Enter the Access Token in the "Token" field
6. Enter the Query Parameters: Go to the "Params" tab in Postman and add the query parameters for the start and due date-time ranges
7. Send the Request
8. Verify the Response

If the batch is deleted successfully, you should receive a response with a 200 OK status code

If the task is not found, you should receive a 404 Not found status code

If invalid Date-Time Format given (400 Bad Request)

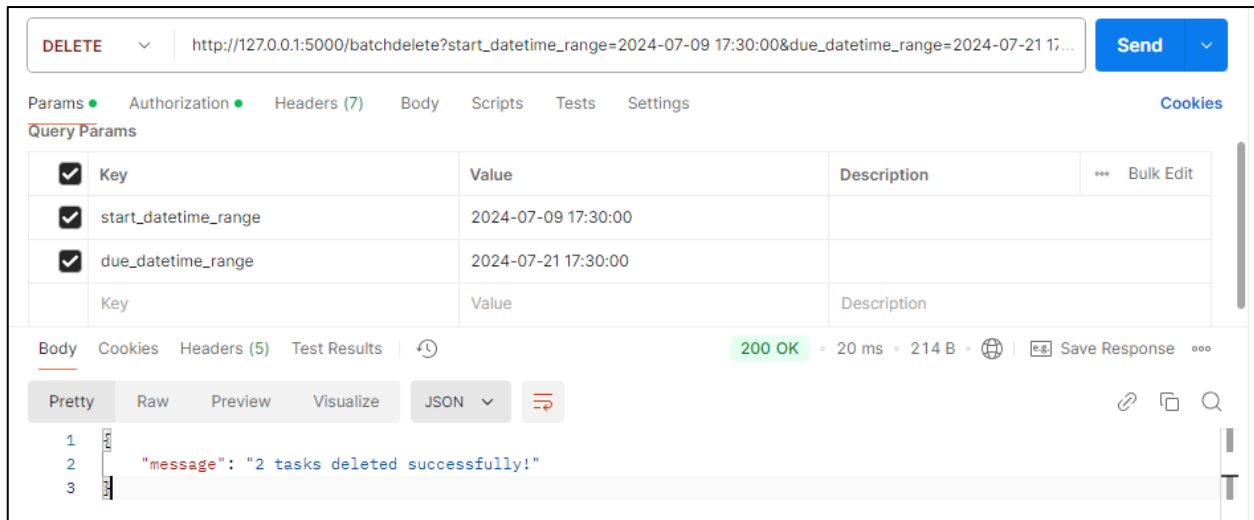
Examples:

1. Batch is deleted successfully

Tasks table in the database

id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at	created_at	updated_at
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Task1	Task1Description	2024-07-10 17:30:00.000000	2024-07-15 18:05:00.000000	2024-07-11 15:30:00.000000	Completed	NULL	NULL	NULL
2	2	Task2...	Task2Description	2024-07-12 17:30:00.000000	2024-07-20 18:05:00.000000	2024-07-18 15:30:00.000000	Completed	NULL	NULL	NULL
3	3	Task3	Task3Description	2024-08-12 17:30:00.000000	2024-08-20 18:05:00.000000	2024-08-15 15:30:00.000000	Completed	NULL	NULL	NULL

Delete batch send request from start_datetime_range=2024-07-09 17:30:00 & due_datetime_range=2024-07-21 17:30:00



DELETE http://127.0.0.1:5000/batchdelete?start_datetime_range=2024-07-09 17:30:00&due_datetime_range=2024-07-21 17:30:00 Send

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description
start_datetime_range	2024-07-09 17:30:00	
due_datetime_range	2024-07-21 17:30:00	

Body Cookies Headers (5) Test Results 200 OK • 20 ms • 214 B Save Response

Pretty Raw Preview Visualize JSON

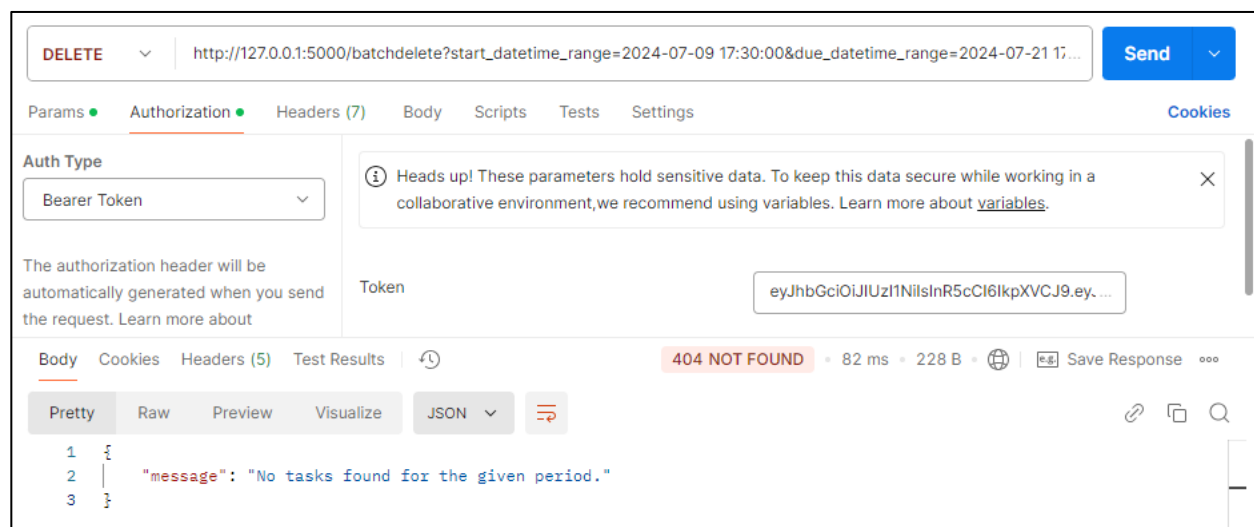
```
1 {
2   "message": "2 tasks deleted successfully!"
3 }
```

Date base after batchdelete

The first 2 tasks within the given time range are deleted and deleted_at is changed to the current time

id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at
1	1	Task1	Task1Description	2024-07-10 17:30:00.000000	2024-07-15 18:05:00.000000	2024-07-11 15:30:00.000000	Completed	2024-12-08 10:24:47.602986
2	1	Task2	Task2Description	2024-07-12 17:30:00.000000	2024-07-20 18:05:00.000000	2024-07-18 15:30:00.000000	Completed	2024-12-08 10:24:47.602986
3	1	Task3	Task3Description	2024-08-12 17:30:00.000000	2024-08-20 18:05:00.000000	2024-08-15 15:30:00.000000	Completed	NULL

2. Task is not found (Tasks already deleted or no tasks found)



DELETE http://127.0.0.1:5000/batchdelete?start_datetime_range=2024-07-09 17:30:00&due_datetime_range=2024-07-21 17:30:00 Send

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Auth Type: Bearer Token

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about [variables](#).

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...

Body Cookies Headers (5) Test Results 404 NOT FOUND • 82 ms • 228 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "No tasks found for the given period."
3 }
```

Restore tasks API Testing Steps:

1. Open Postman
2. Select POST request type
3. Enter endpoint url: `http://127.0.0.1:5000/restoretasks`
4. Navigate to the Authorization tab and select the "Bearer Token" type
5. Enter the Access Token in the "Token" field
6. Send the Request
7. Verify the Response
 - If tasks are found and successfully restored, there is a response message "# tasks restored successfully!" and status code will be 200 OK
 - If no deleted tasks are found, there is a response message "No deleted tasks found to restore"

Examples:

1. Tasks are found and successfully restored

Tasks table in Data base

id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Task1	Task1Description	2024-07-10 17:30:00.000000	2024-07-15 18:05:00.000000	2024-07-11 15:30:00.000000	Completed	2024-12-08 10:24:47.602986
2	1	Task2	Task2Description	2024-07-12 17:30:00.000000	2024-07-20 18:05:00.000000	2024-07-18 15:30:00.000000	Completed	2024-12-08 10:24:47.602986
3	1	Task3	Task3Description	2024-08-12 17:30:00.000000	2024-08-20 18:05:00.000000	2024-08-15 15:30:00.000000	Completed	NULL

Request restore tasks

The screenshot shows a Postman interface for a POST request to `http://127.0.0.1:5000/restoretasks`. The Authorization tab is selected, showing a Bearer Token. The response tab is also selected, displaying a 200 OK status with a response time of 26 ms and a body size of 215 B. The response body is shown in JSON format:

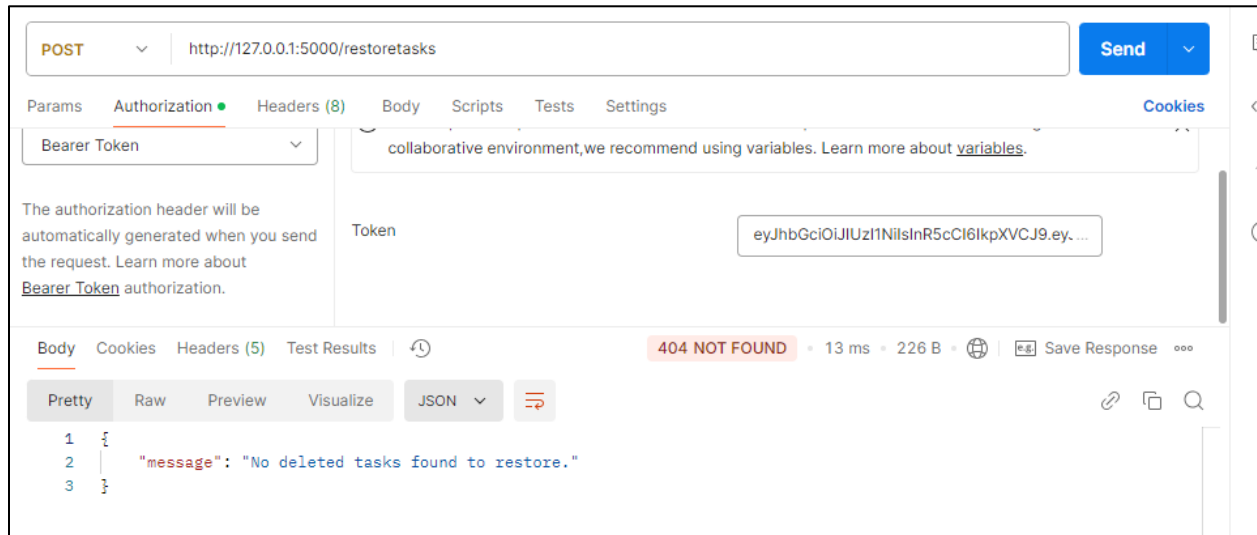
```
{  "message": "2 tasks restored successfully!"}
```


Database after restore tasks (Deleted at set to null)

id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at
1	1	Task1	Task1Description	2024-07-10 17:30:00.000000	2024-07-15 18:05:00.000000	2024-07-11 15:30:00.000000	Completed	NULL
2	2	Task2...	Task2Description	2024-07-12 17:30:00.000000	2024-07-20 18:05:00.000000	2024-07-18 15:30:00.000000	Completed	NULL
3	3	Task3	Task3Description	2024-08-12 17:30:00.000000	2024-08-20 18:05:00.000000	2024-08-15 15:30:00.000000	Completed	NULL

2. No deleted tasks are found

For user id =2 there is no tasks found



POST http://127.0.0.1:5000/restoretasks

Authorization: Bearer Token

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...

404 NOT FOUND • 13 ms • 226 B

Body: {"message": "No deleted tasks found to restore."}

Subscription API

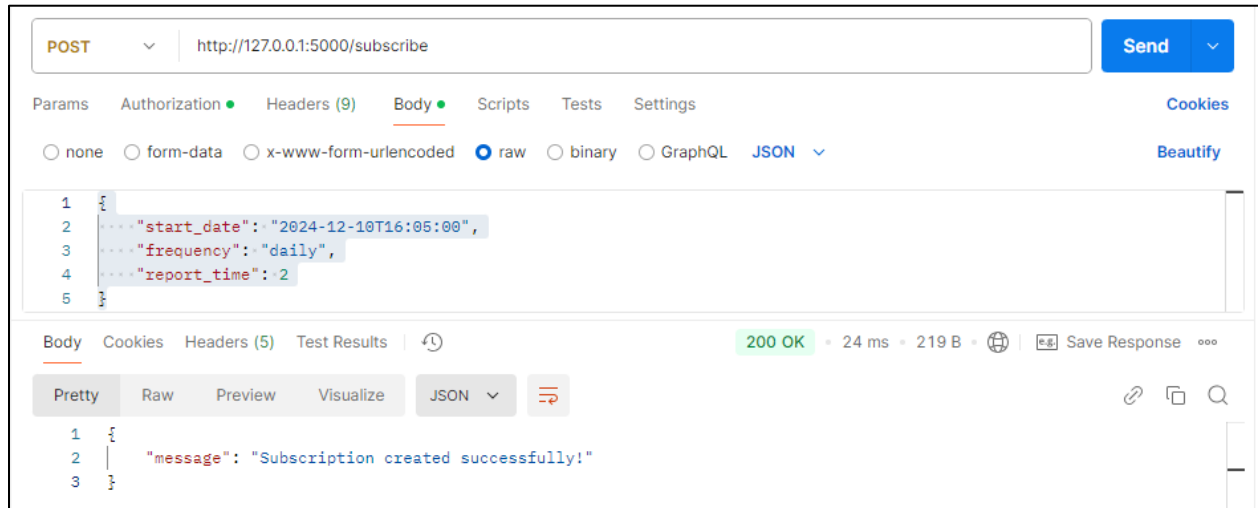
Subscribe API Testing Steps:

1. Open Postman
2. Select POST request type
3. Enter endpoint url: <http://127.0.0.1:5000/subscribe>
4. Navigate to the Authorization tab and select the "Bearer Token" type
5. Enter the Access Token in the "Token" field
6. Navigate to the Body tab then select raw data type and choose JSON format
7. Enter JSON data in the body

```
{  "start_date": "2024-12-10T16:05:00",  "frequency": "daily",  "report_time": 2}
```
8. Send the Request
9. Verify the Response
 - If the subscription is created successfully, there is a response message and status code will be 200 OK
 - If there's any issue (ex: one of the validations isn't satisfied.), the response should return appropriate error message

Examples:

1. subscription is created successfully



POST http://127.0.0.1:5000/subscribe

Params Authorization Headers (9) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "start_date": "2024-12-10T16:05:00",
3   "frequency": "daily",
4   "report_time": 2
5 }
```

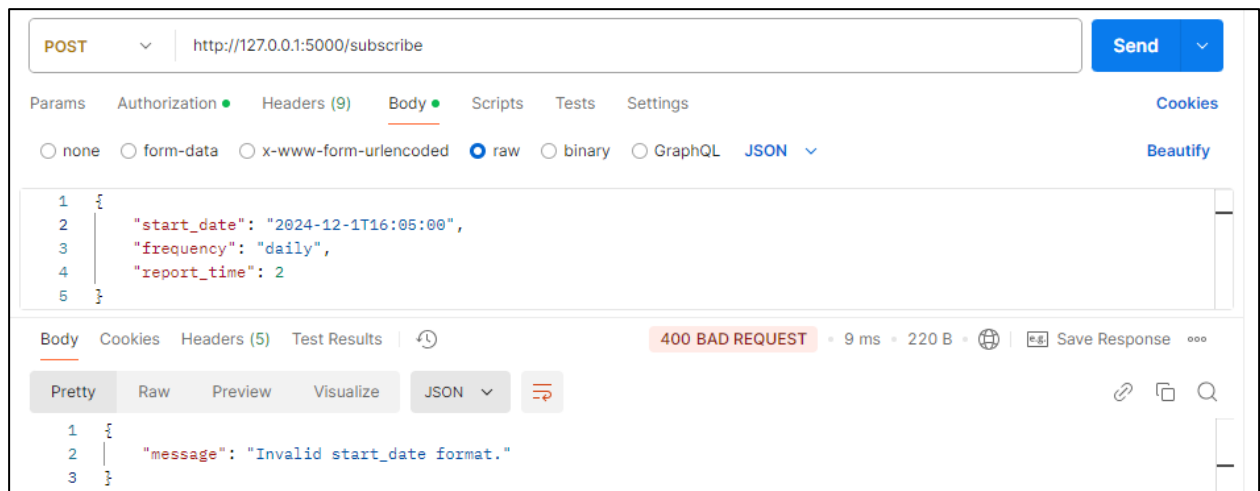
Body Cookies Headers (5) Test Results 200 OK • 24 ms • 219 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Subscription created successfully!"
3 }
```

2. Validations

1. Validate start_date in the correct format



POST http://127.0.0.1:5000/subscribe

Params Authorization Headers (9) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

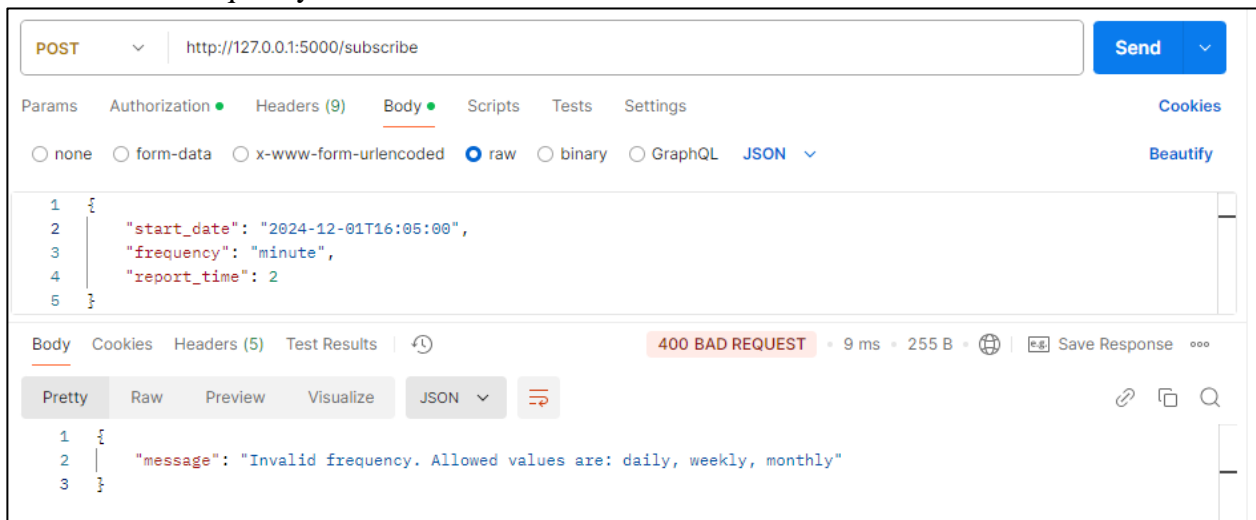
```
1 {
2   "start_date": "2024-12-1T16:05:00",
3   "frequency": "daily",
4   "report_time": 2
5 }
```

Body Cookies Headers (5) Test Results 400 BAD REQUEST • 9 ms • 220 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Invalid start_date format."
3 }
```

2. Validate frequency



POST http://127.0.0.1:5000/subscribe

Params Authorization Headers (9) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

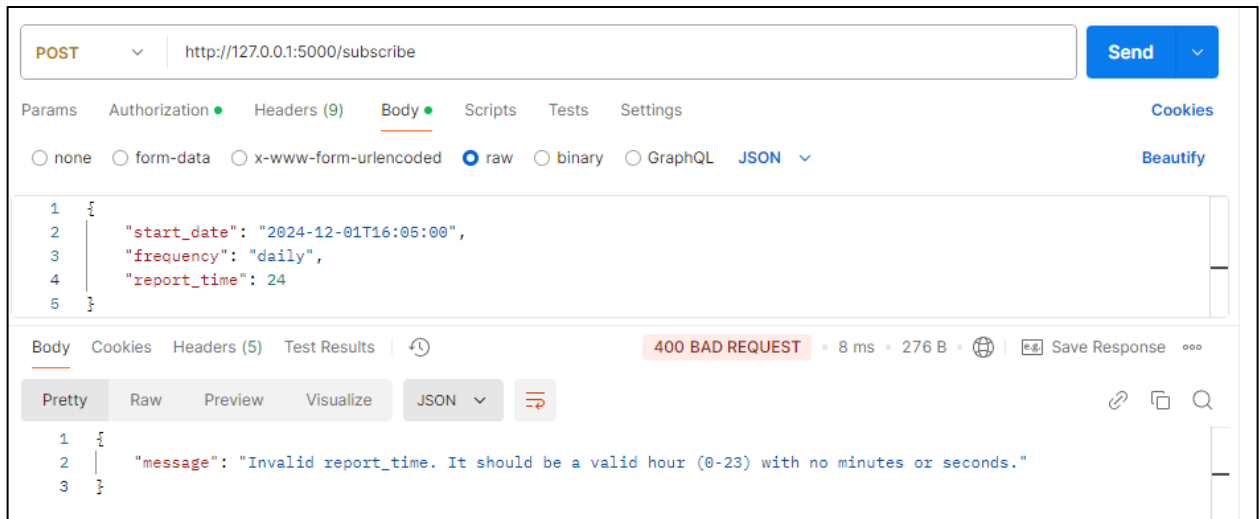
```
1 {
2   "start_date": "2024-12-01T16:05:00",
3   "frequency": "minute",
4   "report_time": 2
5 }
```

Body Cookies Headers (5) Test Results 400 BAD REQUEST • 9 ms • 255 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Invalid frequency. Allowed values are: daily, weekly, monthly"
3 }
```

3. Validate report_time



POST http://127.0.0.1:5000/subscribe

Params Authorization Headers (9) **Body** Scripts Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

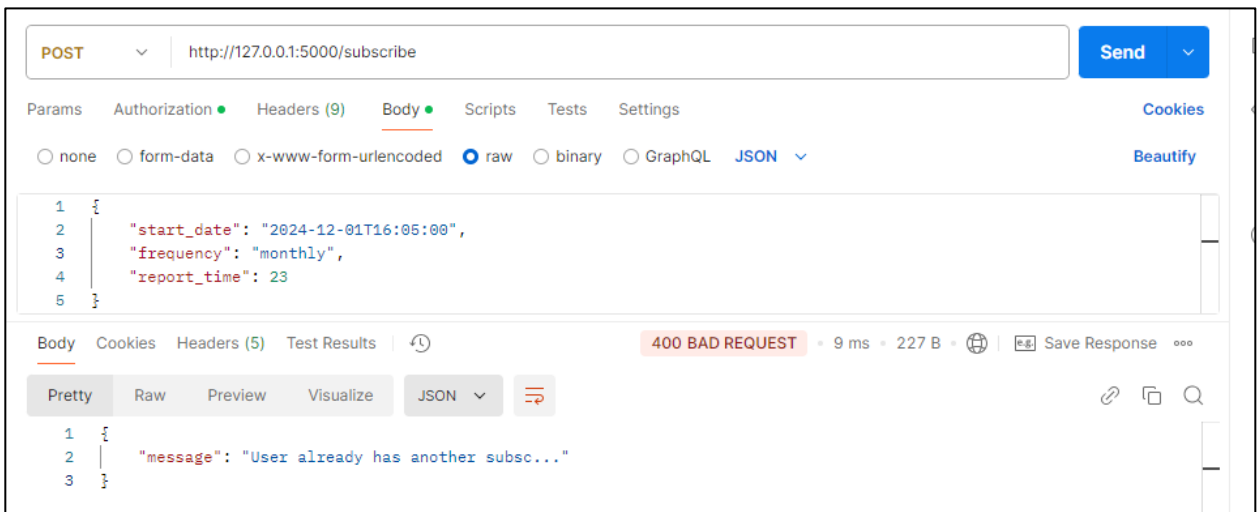
```
1 {
2   "start_date": "2024-12-01T16:05:00",
3   "frequency": "daily",
4   "report_time": 24
5 }
```

Body Cookies Headers (5) Test Results 400 BAD REQUEST • 8 ms • 276 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Invalid report_time. It should be a valid hour (0-23) with no minutes or seconds."
3 }
```

4. User has only one subscription



POST http://127.0.0.1:5000/subscribe

Params Authorization Headers (5) **Body** Scripts Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

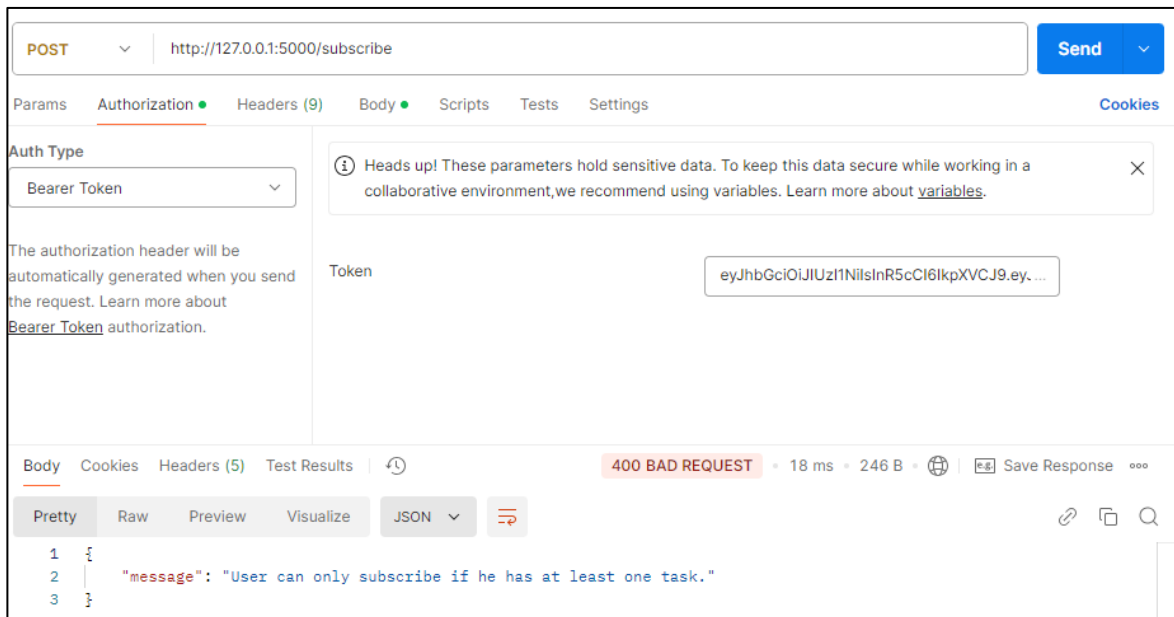
```
1 {
2   "start_date": "2024-12-01T16:05:00",
3   "frequency": "monthly",
4   "report_time": 23
5 }
```

Body Cookies Headers (5) Test Results 400 BAD REQUEST • 9 ms • 227 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "User already has another subsc..."
3 }
```

5. User has at least one task to subscribe

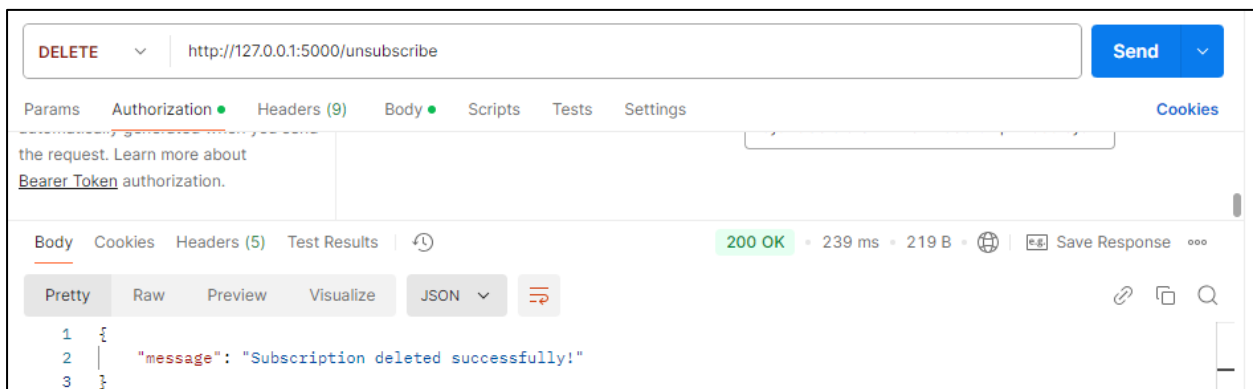


Unsubscribe API Testing Steps:

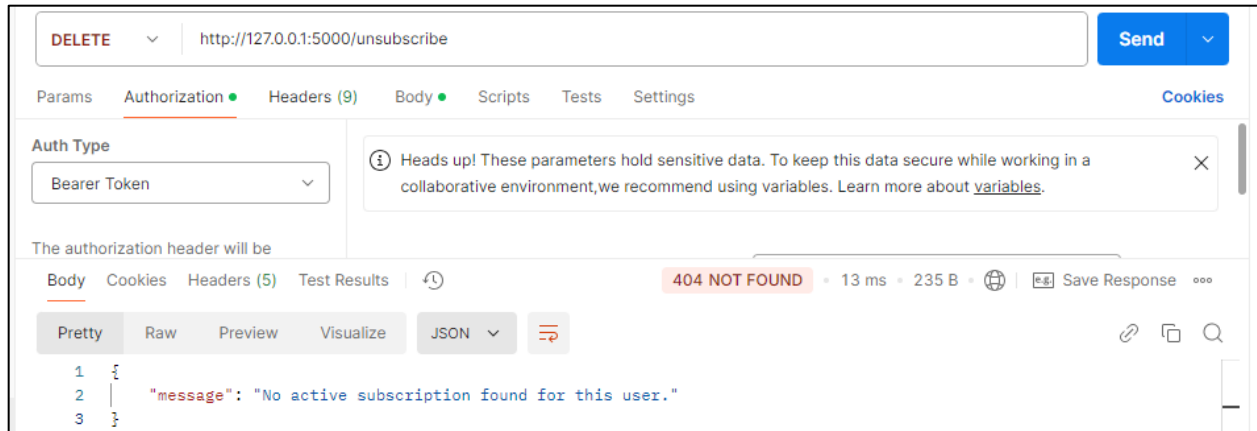
1. Open Postman
2. Select DELETE request type
3. Enter endpoint url: <http://127.0.0.1:5000/unsubscribe>
4. Navigate to the Authorization tab and select the "Bearer Token" type
5. Enter the Access Token in the "Token" field
6. Send the Request
7. Verify the Response
 - If the unsubscribe is created successfully, there is a response message and status code will be 200 OK
 - If there's no subscription found, the response should return appropriate error message

Examples:

1. Unsubscribe is created successfully



2. If no subscription found



Report Generation

Automated Task Report Implementation

1. Flask App Context Setup
2. Retrieve all users from database
3. Check if user have a subscription. If yes extract user_id, frequency, report_time, next_send_time
Note : **next_send_time** is next time to generate report and it initially set to be equal subscription start date
4. Check if it is the time to send report (next send time > current time). If no: return
If yes:
 1. Check subscription frequency. According to frequency the function calculates the time range for which the tasks will be retrieved by making variable time_limit and updates next_send_time based on the subscription frequency
For daily subscription: next_send_time = next_send_time +timedelta(days=1)
For weekly subscription: next_send_time = next_send_time +timedelta(weeks=1)
For monthly subscription: next_send_time = next_send_time +timedelta(days=30)
(Note: without frequency check email will be send every 1 minute)
 2. Check if report time = current time in hours. If yes retrieve tasks within this time range (note in checking on range I **assumed** that task due_date > time_limit)
 3. Categorize tasks to “Pending”, “Overdue” and “Completed”
 4. Generate the HTML Email layout
 5. Send the Email
 6. scheduler is used to automate the execution of the generate_report function at regular intervals this is done by:
 1. Initialize scheduler
 2. Schedule the task to run every 1 minute

Note: To optimize the task execution and avoid redundant responses, it would be better to schedule the task to run once every hour. In this case, the server must be executed at the start of each hour ex 9:00

Report Generation test cases:

1. Successful report generation with tasks of the last day: (Only one task)

Steps:

1. User Sign-In (user_id=2)

[illegible]

2. Create task for the user

(to test the task is created that the due date during the last 24 hr)

POST

▼

http://127.0.0.1:5000/createtask

Send ▼

Params

Authorization ●

Headers (9)

Body ●

Scripts

Tests

Settings

Cookies

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▼

Beautify

```
1 {
2   "title" : "Task",
3   "description" : "TaskDescription",
4   "start_date" : "2024-12-09T13:00:00Z",
5   "due_date" : "2024-12-09T13:05:00Z",
6   "completion_date" : "2024-12-09T13:02:00Z",
7   "status" : "Completed"
8 }
```

Body

Cookies

Headers (5)

Test Results

🔄

200 OK • 21 ms • 213 B • 🌐

📄 Save Response ⋮

Pretty

Raw

Preview

Visualize

JSON ▼

🔗

```
1 {
2   "message": "Task is successfully created"
3 }
```

3. User subscription

POST http://127.0.0.1:5000/subscribe

Params Authorization Headers (9) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "start_date": "2024-12-09T13:05:00",
3   "frequency": "daily",
4   "report_time": 19
5 }
```

Body Cookies Headers (5) Test Results 200 OK 31 ms 219 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Subscription created successfully!"
3 }
```

next_send_time = subscription_start_date = "2024-12-09T13:05:00"

	id	user_id	start_date	frequency	report_time	next_send_time
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	2024-12-08 ...	daily	21	2024-12-12 13:05:00.000000
2	2	2	2024-12-09 ...	daily	19	2024-12-09 13:05:00.000000

4. Report Time Check:

1. next_send_time is not greater than now, so the code proceeds.
2. The system then checks if the report_time matches the current hour (now).

5. An email is sent to user with task details

(only one task is ended to the user in the last 24 hours)

Your Task Report Inbox x

vodafonetaskreports@gmail.com 21:47 (6 minutes ago)

to arwawaid28, me

Task Report

Pending Tasks

Title	Due Date
-------	----------

Completed Tasks

Title	Completion Date
Task	2024-12-09 13:02:00

Overdue Tasks

Title	Due Date
-------	----------

6. $\text{next_send_time} = \text{next_send_time} + 24$

	id	user_id	start_date	frequency	report_time	next_send_time
	Filter	Filter	Filter	Filter	Filter	Filter
1	2	2	2024-12-09 13:05:00.000000	daily	19	2024-12-10 13:05:00.000000
2	3	3	2024-12-09 13:05:00.000000	monthly	19	2025-01-08 13:05:00.000000
3	4	4	2024-12-09 13:05:00.000000	weekly	19	2024-12-16 13:05:00.000000

2. Successful report generation with tasks of the last month: (Many tasks)

1. User sign in ($\text{user_id}=3$)
2. create tasks for the user during last month
3. User subscription

POST

http://127.0.0.1:5000/subscribe

Send

ParamsAuthorizationHeaders (9)BodyScriptsTestsSettingsCookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

1

{

2

"start_date": "2024-12-09T13:05:00",

3

"frequency": "monthly",

4

"report_time": 19

5

}

BodyCookiesHeaders (5)Test Results200 OK · 18 ms · 219 B · Save Response

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"message": "Subscription created successfully!"

3

}


$\text{next_send_time} = \text{subscription_start_date} = \text{"2024-12-09T13:05:00"}$

	id	user_id	start_date	frequency	report_time	next_send_time
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	2024-12-08 ...	daily	21	2024-12-12 13:05:00.000000
2	2	2	2024-12-09 ...	daily	19	2024-12-09 13:05:00.000000
3	3	3	2024-12-09 ...	monthly	19	2024-12-09 13:05:00.000000

3. Report Time Check:

1. next_send_time is not greater than now, so the code proceeds.
2. The system then checks if the report_time matches the current hour (now).

4. An email is sent to user with task details
(3 task is ended by the user in the last 1 month)



vodafone task reports@gmail.com
to arwawalid, me

21:47 (9 minutes ago)

☆
😊
↩
⋮

Task Report

Pending Tasks

Title	Due Date
Task 1	2024-12-01 13:05:00

Completed Tasks

Title	Completion Date
Task 2	2024-11-16 13:02:00

Overdue Tasks

Title	Due Date
Task 3	2024-11-25 13:05:00

5. $\text{next_send_time} = \text{next_send_time} + (24 \times 30)$

	id	user_id	start_date	frequency	report_time	next_send_time
	Filter	Filter	Filter	Filter	Filter	Filter
1	2	2	2024-12-09 13:05:00.000000	daily	19	2024-12-10 13:05:00.000000
2	3	3	2024-12-09 13:05:00.000000	monthly	19	2025-01-08 13:05:00.000000
3	4	4	2024-12-09 13:05:00.000000	weekly	19	2024-12-16 13:05:00.000000

3. Successful report generation with tasks of the last week: (no tasks last week)

1. User sign in (user_id =4)
2. create 0 tasks for the user during last week
3. User subscription

POST

http://127.0.0.1:5000/subscribe

Send

Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings

none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON

1 {
2 "start_date": "2024-12-09T13:05:00",
3 "frequency": "weekly",
4 "report_time": 19
5 }

Body
Cookies
Headers (5)
Test Results

200 OK
22 ms
219 B
Save Response

Pretty
Raw
Preview
Visualize
JSON

1 {
2 "message": "Subscription created successfully!"
3 }

next_send_time = subscription_start_date = "2024-12-09T13:05:00"

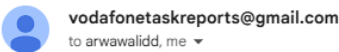
	id	user_id	start_date	frequency	report_time	next_send_time
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	2024-12-08 ...	daily	21	2024-12-12 13:05:00.000000
2	2	2	2024-12-09 ...	daily	19	2024-12-09 13:05:00.000000
3	3	3	2024-12-09 ...	monthly	19	2024-12-09 13:05:00.000000
4	4	4	2024-12-09 ...	weekly	19	2024-12-09 13:05:00.000000

4. Report Time Check:

1. next_send_time is not greater than now, so the code proceeds.
2. The system then checks if the report_time matches the current hour (now).

5. An email is sent to user with task details

(0 task is ended by the user in the last 1 week)

21:47 (13 minutes ago) ☆ 😊 ↩ ⋮

Task Report

Pending Tasks

Title	Due Date
-------	----------

Completed Tasks

Title	Completion Date
-------	-----------------

Overdue Tasks

Title	Due Date
-------	----------

6. next_send_time = next_send_time + (24*7)

	id	user_id	start_date	frequency	report_time	next_send_time
	Filter	Filter	Filter	Filter	Filter	Filter
1	2	2	2024-12-09 13:05:00.000000	daily	19	2024-12-10 13:05:00.000000
2	3	3	2024-12-09 13:05:00.000000	monthly	19	2025-01-08 13:05:00.000000
3	4	4	2024-12-09 13:05:00.000000	weekly	19	2024-12-16 13:05:00.000000