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.

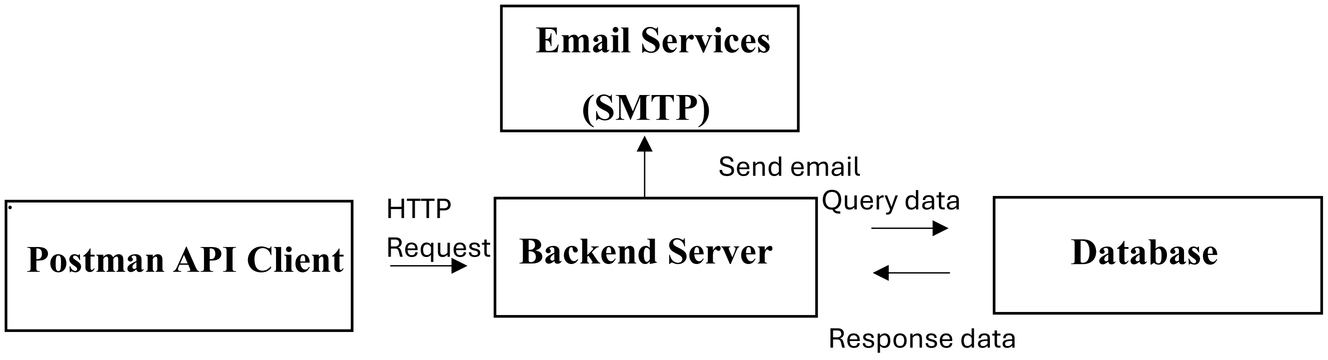
Technology Used: Postman, an API client, is used to send HTTP requests to the backend and test the application's endpoints to ensure they respond correctly.

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

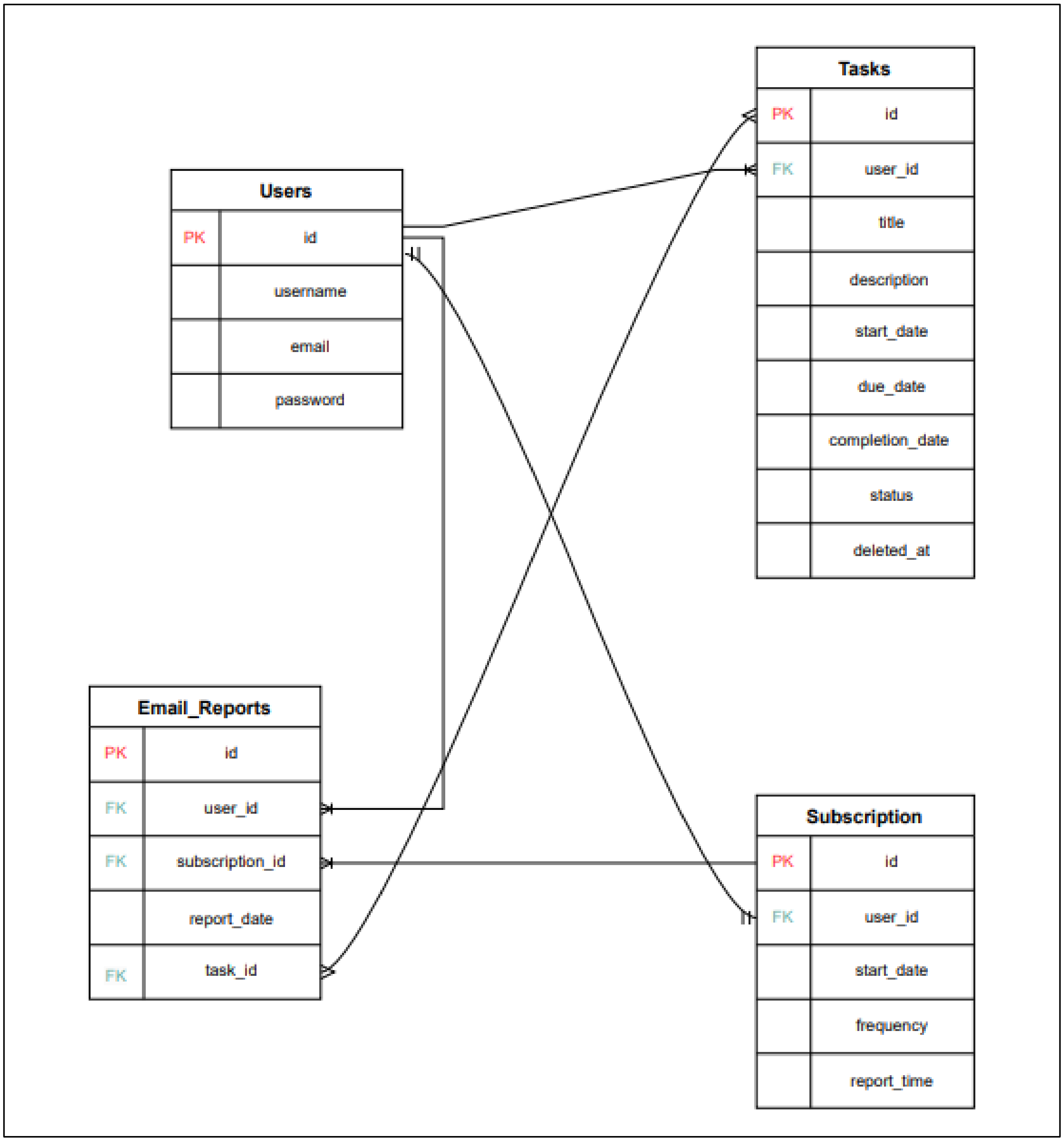
User -Tasks: One-to-many

User -Email_Reports: One-to-many.

Subscription – Email_Reports: One-to-many

Task – Email_Report: many-to-many

Database Diagram



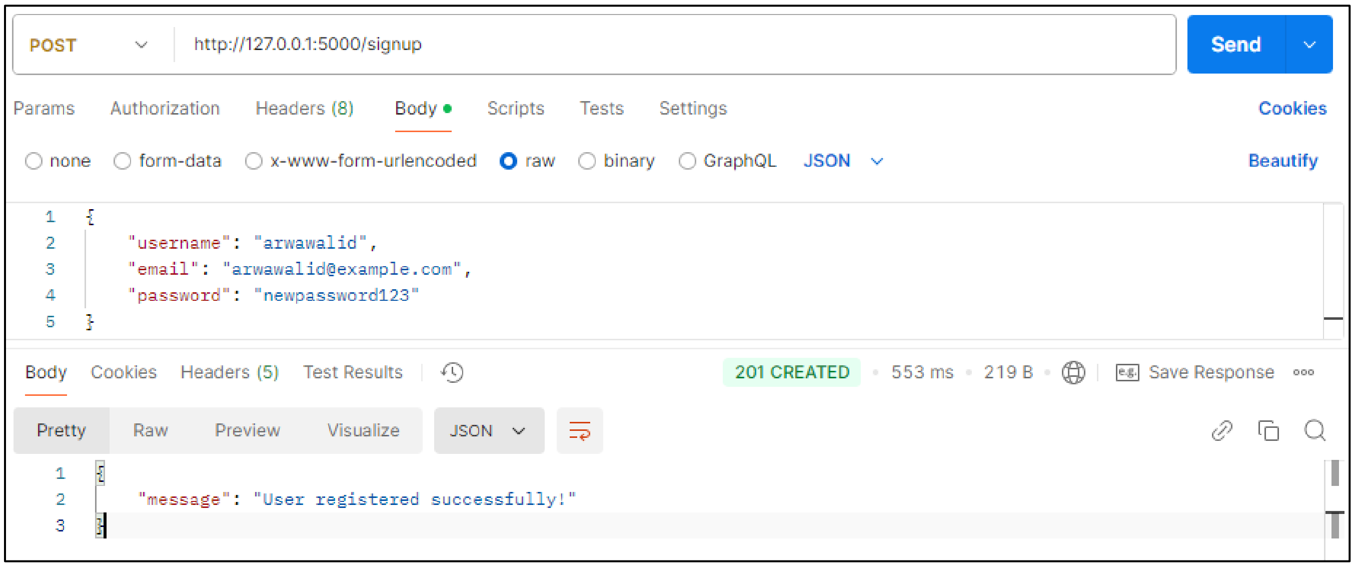
User Authentication

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:

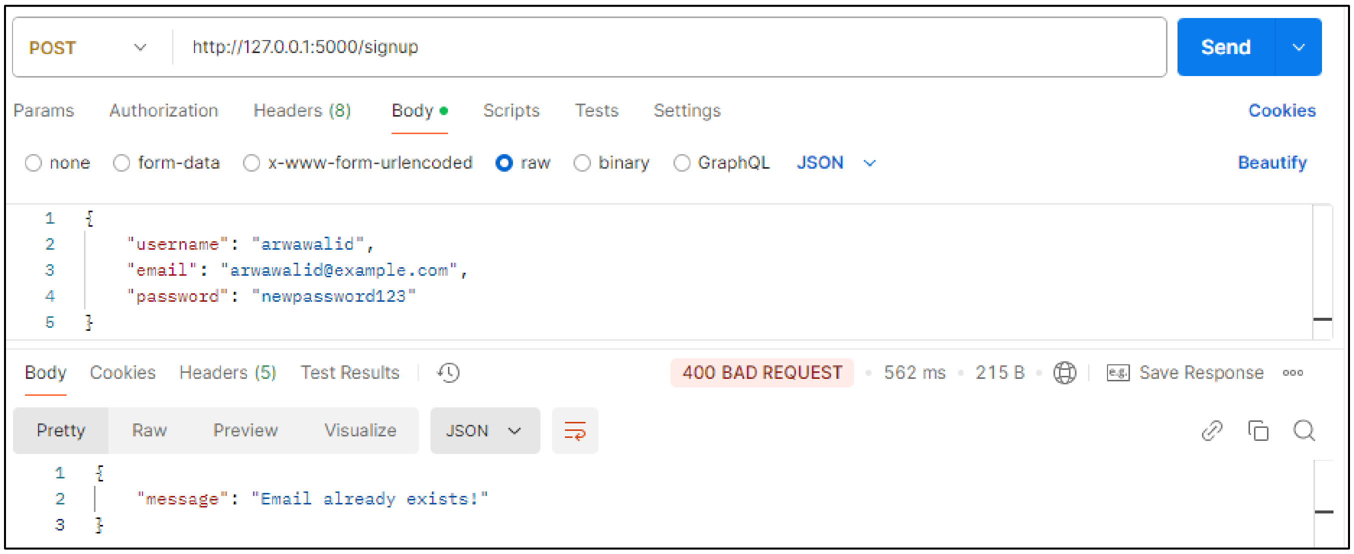
1. **For successful registration**



```
{
    "username": "arwawalid",
    "email": "arwawalid@example.com",
    "password": "newpassword123"
}
```

```
{
    "message": "User registered successfully!"
}
```

2. **If the user is already registered**



http://127.0.0.1:5000/signup

```
{
    "username": "arwawalid",
    "email": "arwawalid@example.com",
    "password": "newpassword123"
}
```

400 BAD REQUEST • 562 ms • 215 B

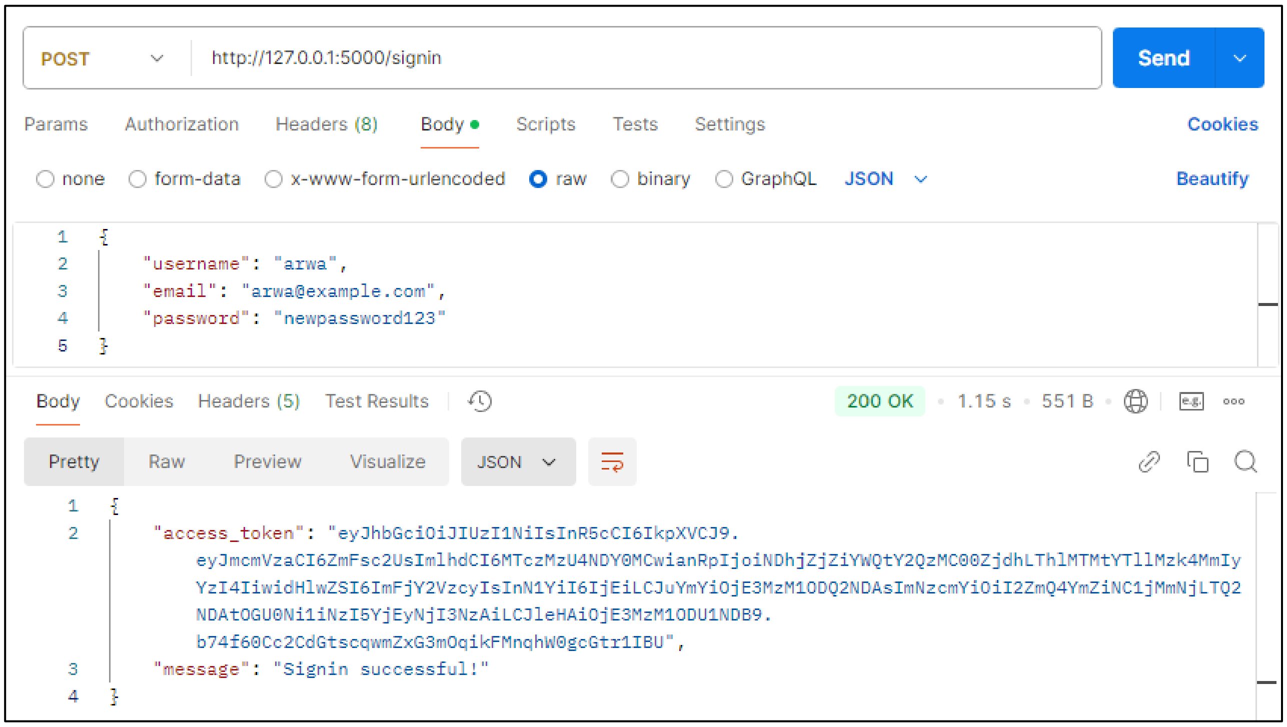
```
{
    "message": "Email already exists!"
}
```

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.

<u>Examples:</u>

1. **For successful sign-in**



```
POST http://127.0.0.1:5000/signin
```

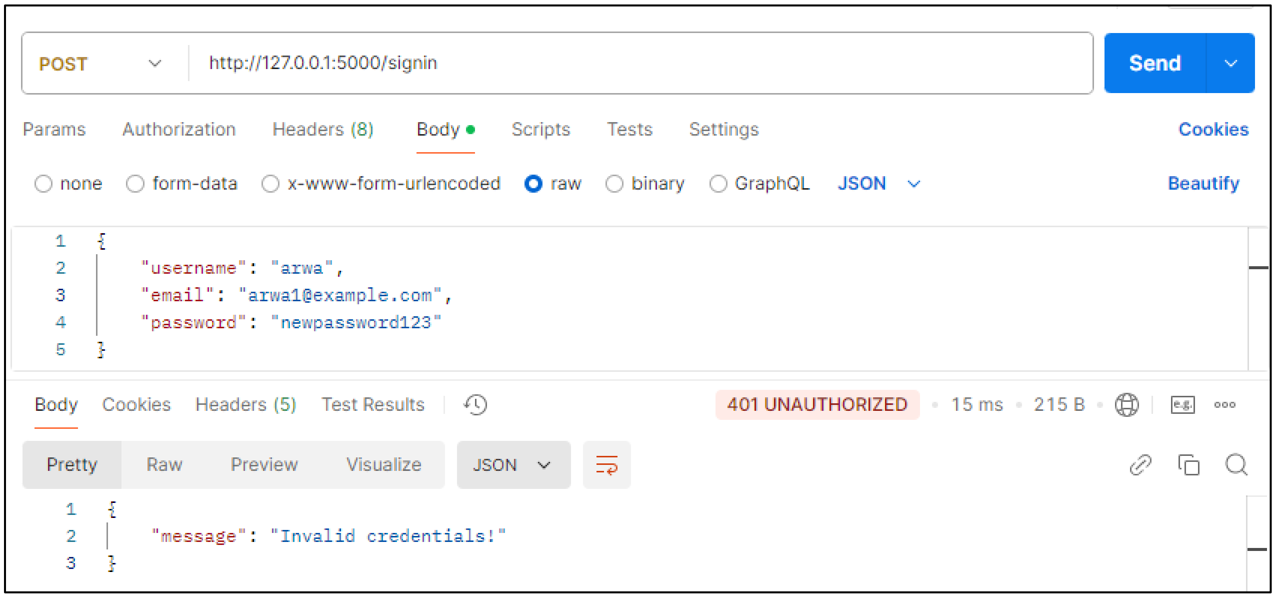
Body: raw, JSON

```
{
    "username": "arwa",
    "email": "arwa@example.com",
    "password": "newpassword123"
}
```

200 OK • 1.15 s • 551 B

```
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczMzU4NDY0MCwianRpIjoiNDhjZjZiYWQtY2QzMC00ZjdhLThlMTMtYTllMzk4MmIy
        YzI4IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjEiLCJuYmYiOjE3MzM1ODQ2NDAsImNzcmYiOiI2ZmQ4YmZiNC1jMmNjLTQ2
        NDAtOGU0Ni1iNzI5YjEyNjI3NzAiLCJleHAiOjE3MzM1ODU1NDB9.
        b74f60Cc2CdGtscqwmZxG3mOqikFMnqhW0gcGtr1IBU",
    "message": "Signin successful!"
}
```

2. **For invalid credentials**



```
POST http://127.0.0.1:5000/signin
```

Body: raw, JSON

```
{
    "username": "arwa",
    "email": "arwa1@example.com",
    "password": "newpassword123"
}
```

401 UNAUTHORIZED • 15 ms • 215 B

```
{
    "message": "Invalid credentials!"
}
```

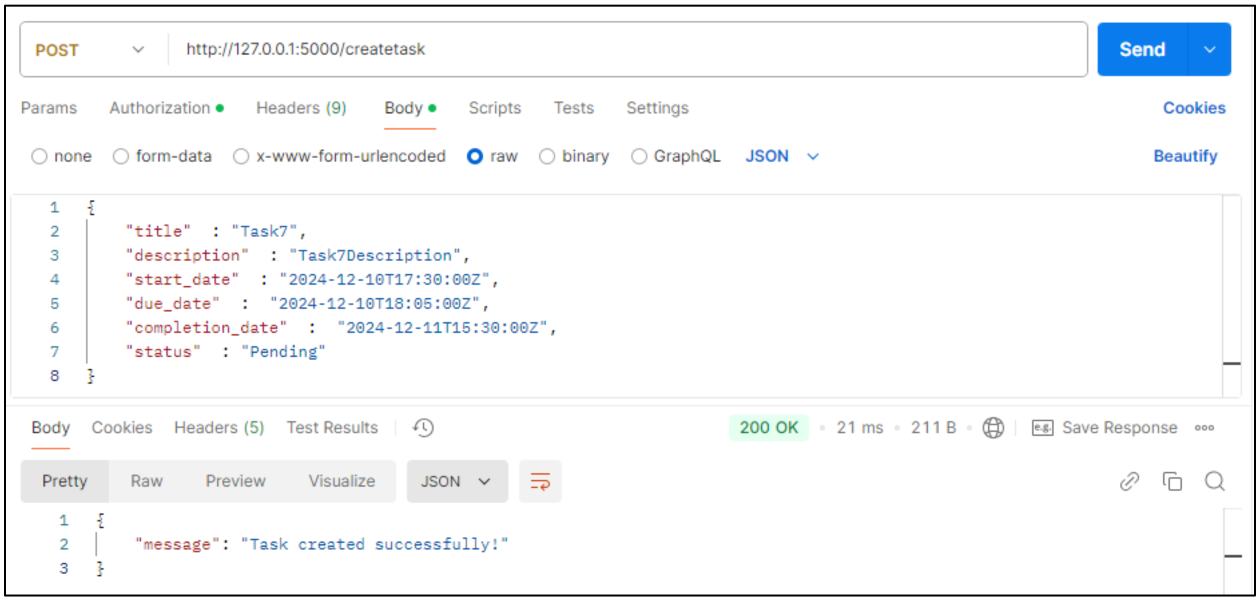
Task Management API

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**



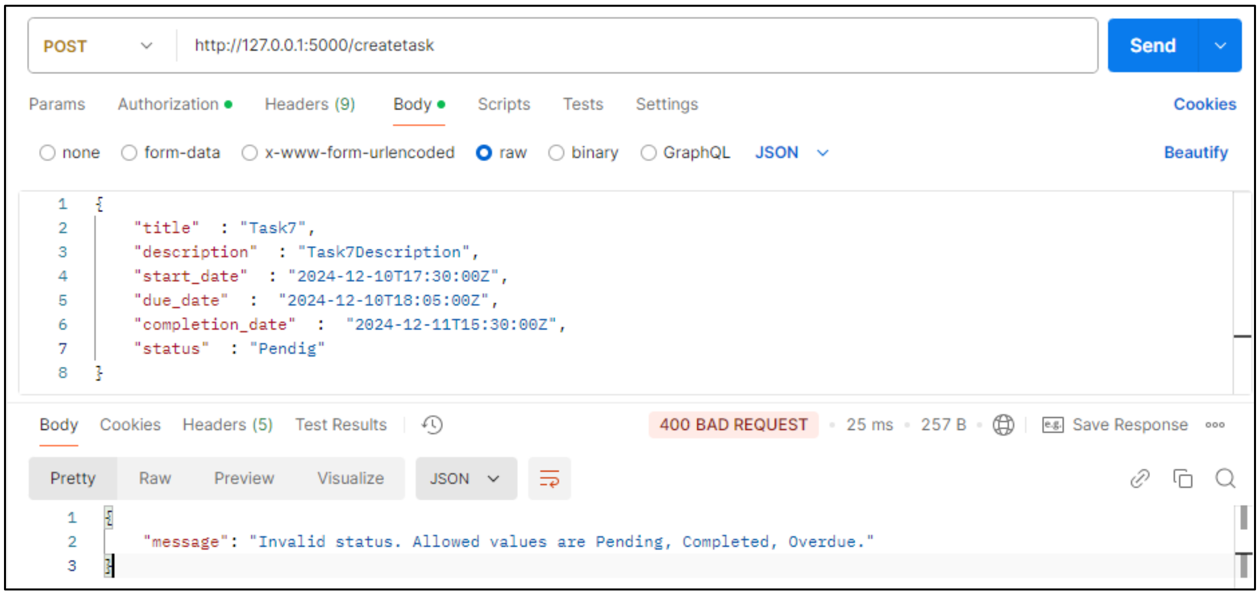
http://127.0.0.1:5000/createtask

```
{
    "title"  : "Task7",
    "description"  : "Task7Description",
    "start_date"  : "2024-12-10T17:30:00Z",
    "due_date"  :  "2024-12-10T18:05:00Z",
    "completion_date"  :  "2024-12-11T15:30:00Z",
    "status"  : "Pending"
}
```

```
{
    "message": "Task created successfully!"
}
```

2. **Validations**

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

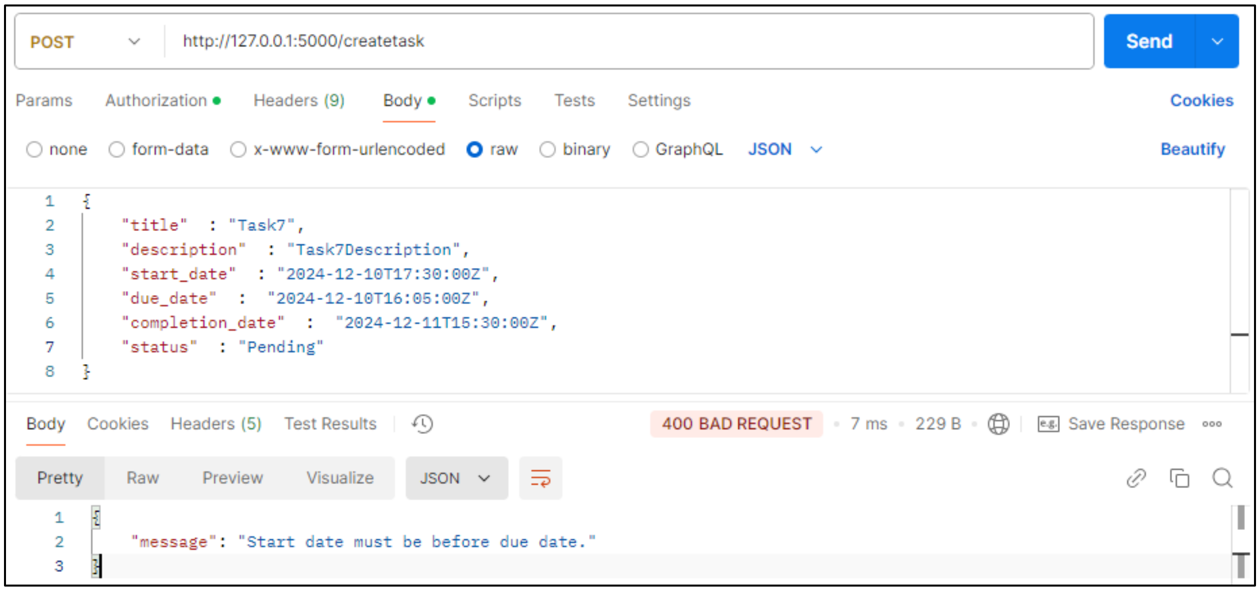


http://127.0.0.1:5000/createtask

```
{
    "title"  : "Task7",
    "description"  : "Task7Description",
    "start_date"  : "2024-12-10T17:30:00Z",
    "due_date"  :  "2024-12-10T18:05:00Z",
    "completion_date"  :  "2024-12-11T15:30:00Z",
    "status"  : "Pendig"
}
```

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

2. Start Date Must Be Earlier Than Due Date

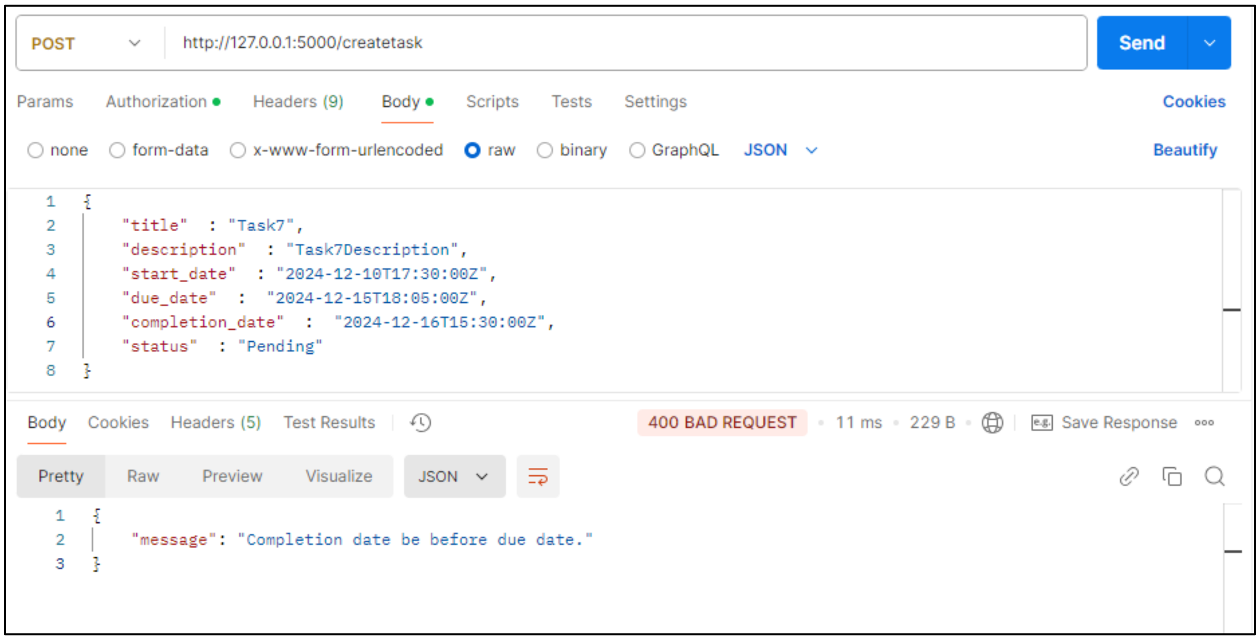


http://127.0.0.1:5000/createtask

```
{
    "title"  : "Task7",
    "description"  : "Task7Description",
    "start_date"  : "2024-12-10T17:30:00Z",
    "due_date"  :  "2024-12-10T16:05:00Z",
    "completion_date"  :  "2024-12-11T15:30:00Z",
    "status"  : "Pending"
}
```

```
{
    "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)

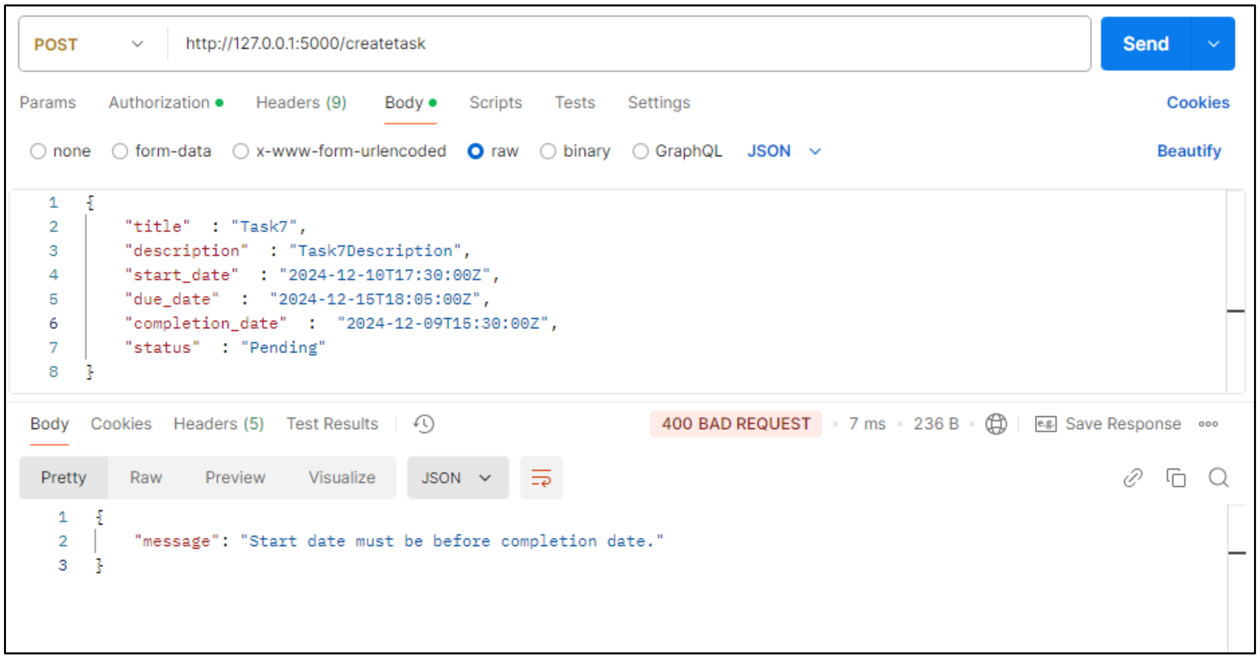


http://127.0.0.1:5000/createtask

```
{
    "title"  : "Task7",
    "description"  : "Task7Description",
    "start_date"  : "2024-12-10T17:30:00Z",
    "due_date"  :  "2024-12-15T18:05:00Z",
    "completion_date"  :  "2024-12-16T15:30:00Z",
    "status"  : "Pending"
}
```

400 BAD REQUEST · 11 ms · 229 B

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



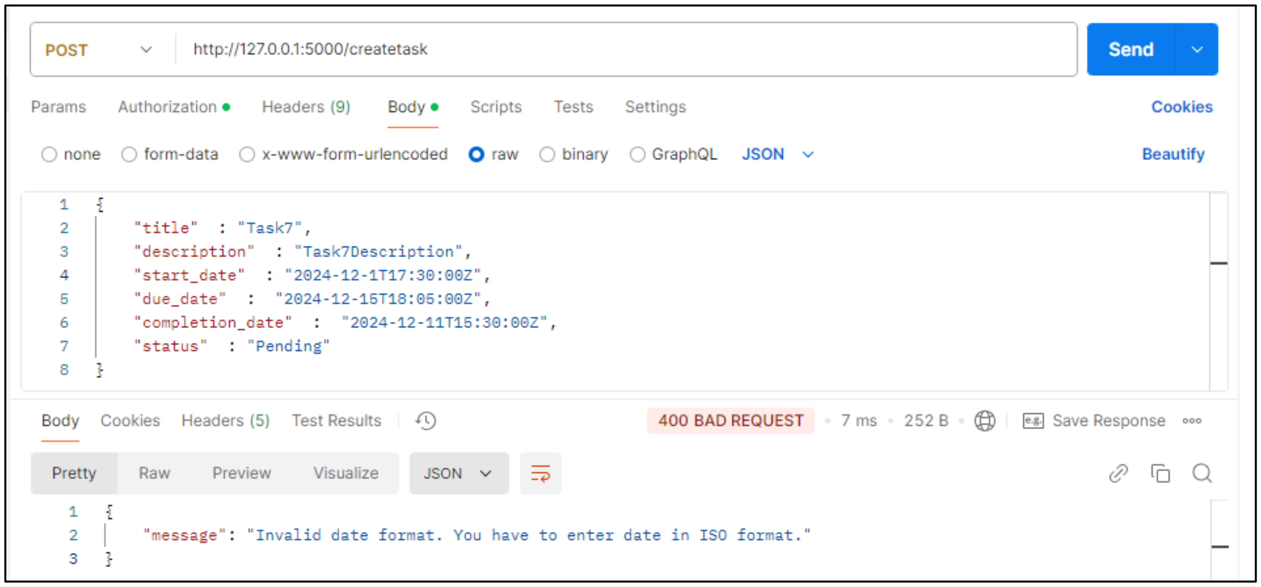
http://127.0.0.1:5000/createtask

```
{
    "title"  : "Task7",
    "description"  : "Task7Description",
    "start_date"  : "2024-12-10T17:30:00Z",
    "due_date"  :  "2024-12-15T18:05:00Z",
    "completion_date"  :  "2024-12-09T15:30:00Z",
    "status"  : "Pending"
}
```

400 BAD REQUEST · 7 ms · 236 B

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

4. Date Must Be in a Valid Format



http://127.0.0.1:5000/createtask

```
{
    "title"  : "Task7",
    "description"  : "Task7Description",
    "start_date"  : "2024-12-1T17:30:00Z",
    "due_date"  :  "2024-12-15T18:05:00Z",
    "completion_date"  :  "2024-12-11T15:30:00Z",
    "status"  : "Pending"
}
```

400 BAD REQUEST · 7 ms · 252 B

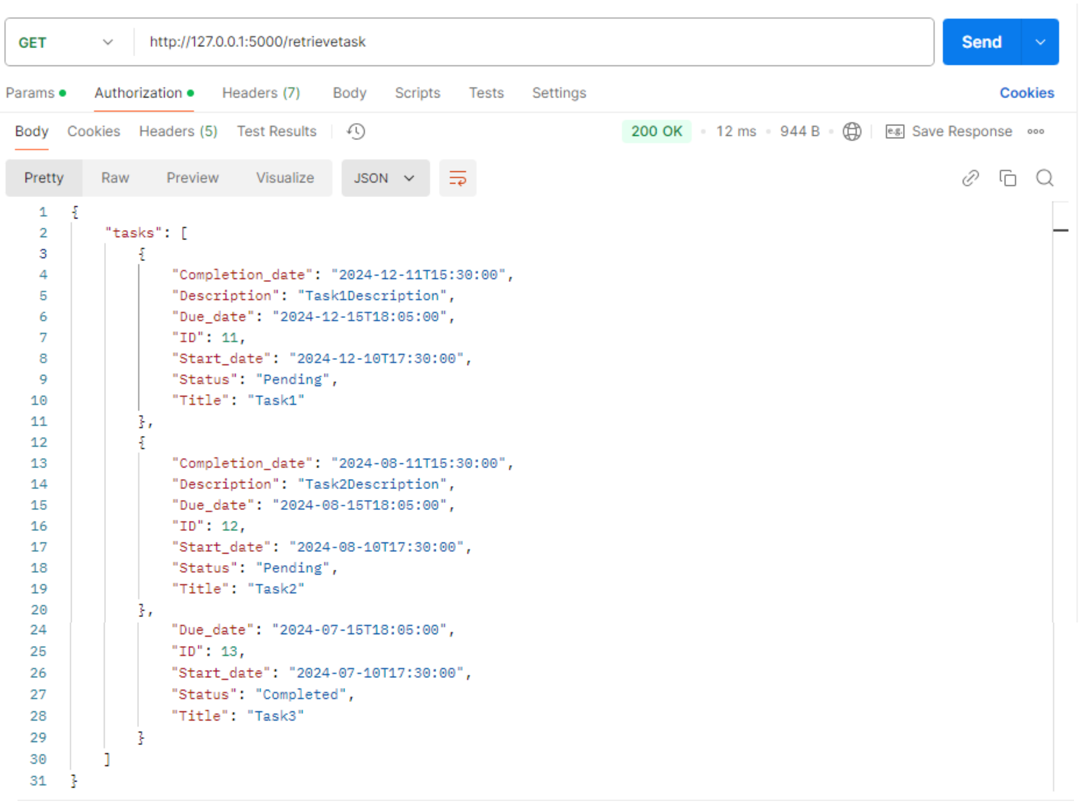
```
{
    "message": "Invalid date format. You have to enter date in ISO format."
}
```

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"

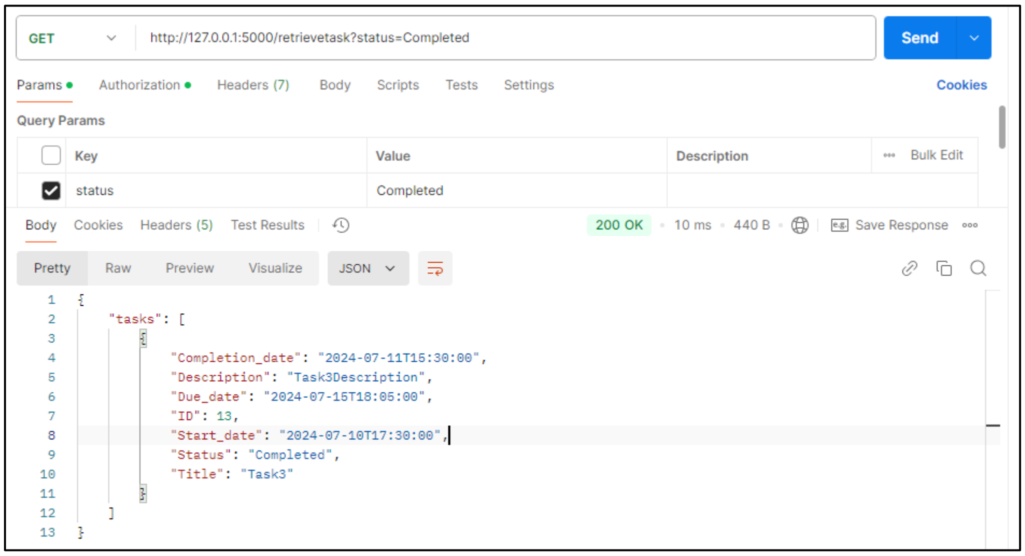
Examples:

1. **Retrieve All Tasks Without Applying Any Filters**



```
{
    "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"
        },
        {
            "Completion_date": "2024-08-11T15:30:00",
            "Description": "Task2Description",
            "Due_date": "2024-08-15T18:05:00",
            "ID": 12,
            "Start_date": "2024-08-10T17:30:00",
            "Status": "Pending",
            "Title": "Task2"
        },
            "Due_date": "2024-07-15T18:05:00",
            "ID": 13,
            "Start_date": "2024-07-10T17:30:00",
            "Status": "Completed",
            "Title": "Task3"
        }
    ]
}
```

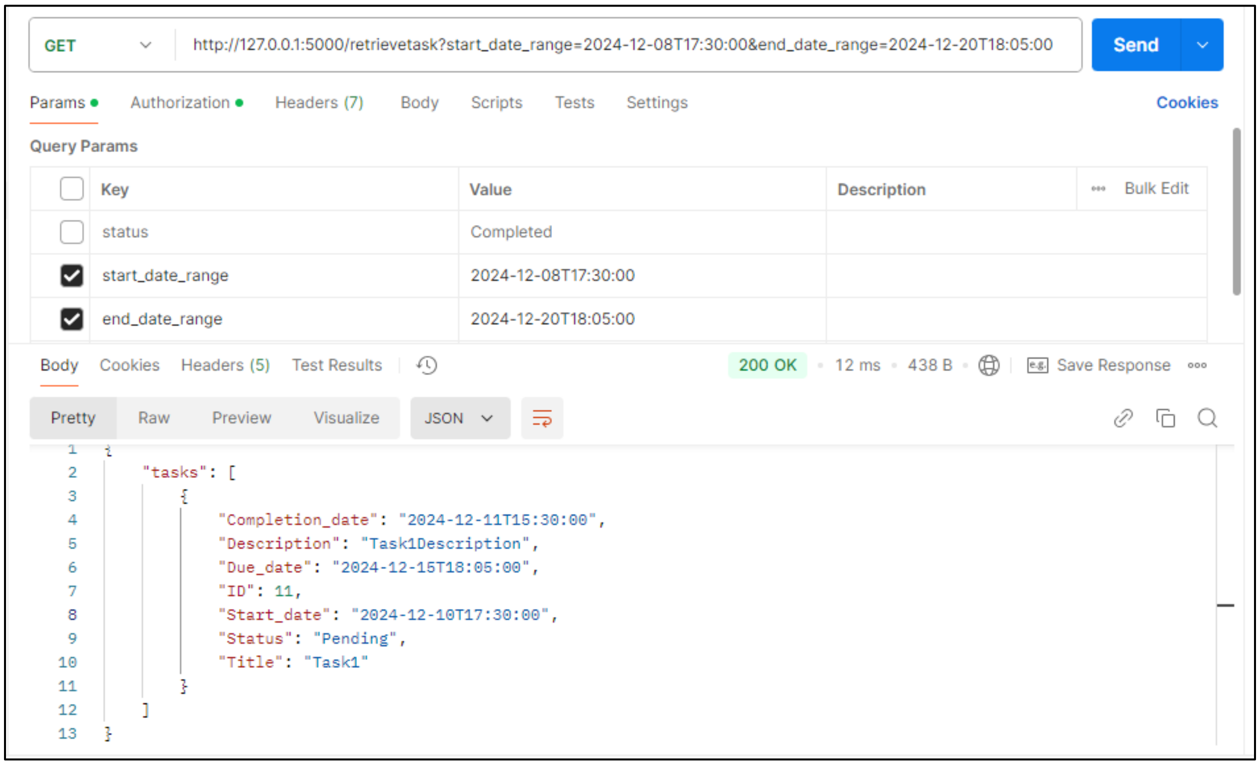
2. **Validations**
    1. Retrieve Tasks Filtered by Status (ex: Completed)



	Key	Value	Description
✓	status	Completed	

```
{
    "tasks": [
        {
            "Completion_date": "2024-07-11T15:30:00",
            "Description": "Task3Description",
            "Due_date": "2024-07-15T18:05:00",
            "ID": 13,
            "Start_date": "2024-07-10T17:30:00",
            "Status": "Completed",
            "Title": "Task3"
        }
    ]
}
```

2. Retrieve Tasks Within a Specific Time Range
   ex: start date : 2024-12-08T17:30:00 , end date : 2024-12-20T18:05:00


GET http://127.0.0.1:5000/retrievetask?start_date_range=2024-12-08T17:30:00&end_date_range=2024-12-20T18:05:00

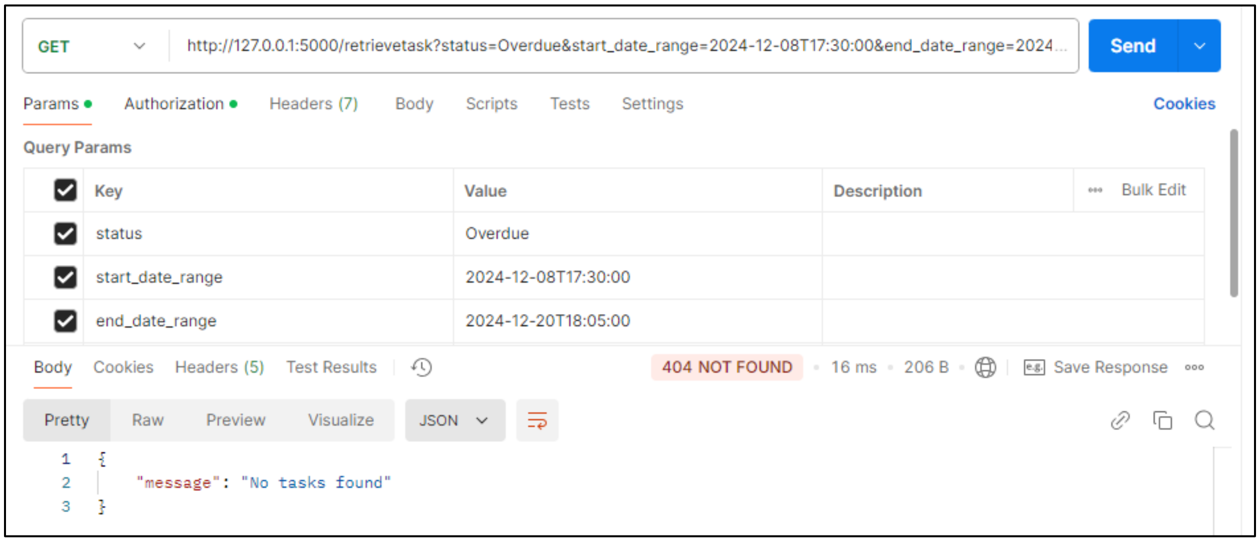
Query Params

	Key	Value	Description
☐	status	Completed	
✓	start_date_range	2024-12-08T17:30:00	
✓	end_date_range	2024-12-20T18:05:00	

200 OK · 12 ms · 438 B

```
{
    "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


GET http://127.0.0.1:5000/retrievetask?status=Overdue&start_date_range=2024-12-08T17:30:00&end_date_range=2024...

Query Params

	Key	Value	Description
✓	status	Overdue	
✓	start_date_range	2024-12-08T17:30:00	
✓	end_date_range	2024-12-20T18:05:00	

404 NOT FOUND · 16 ms · 206 B

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

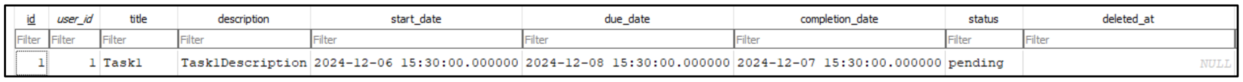
Update tasks API Testing Steps:

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:

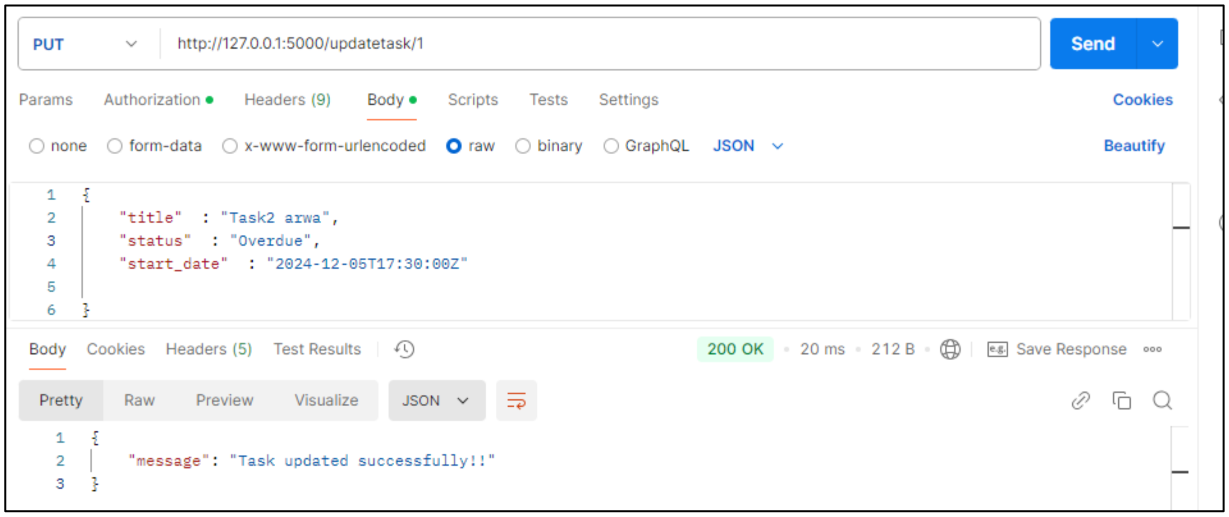
1. **Task is updated successfully**

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
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

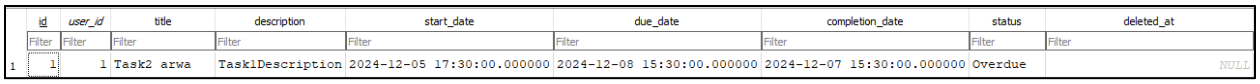


```
{
    "title"  : "Task2 arwa",
    "status"  : "Overdue",
    "start_date"  : "2024-12-05T17:30:00Z"

}
```

```
{
    "message": "Task updated successfully!!"
}
```

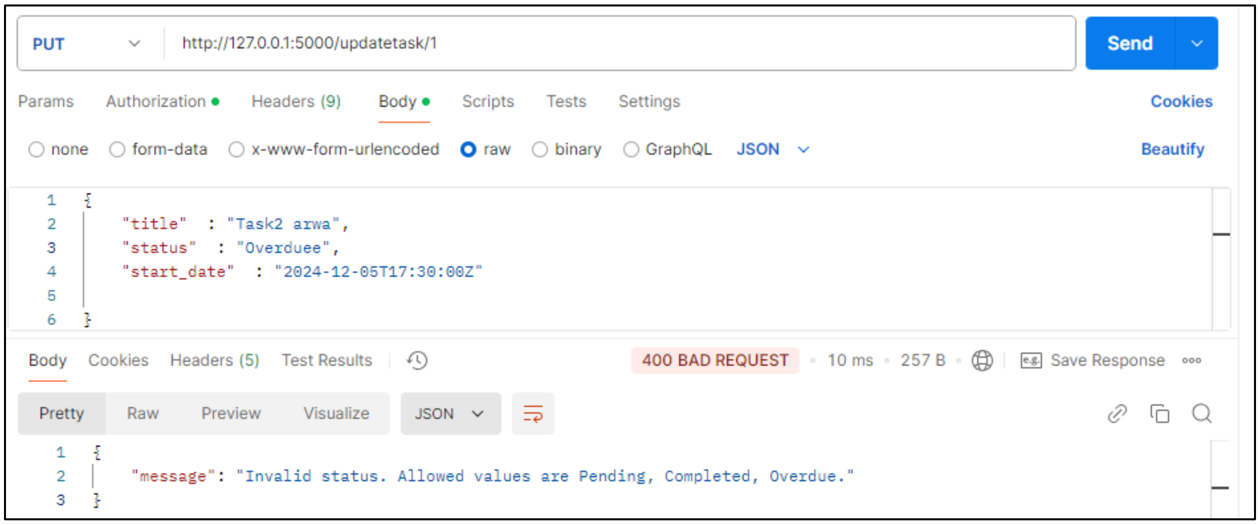
Task 1 Updated in the Database



	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	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. Validations

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



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

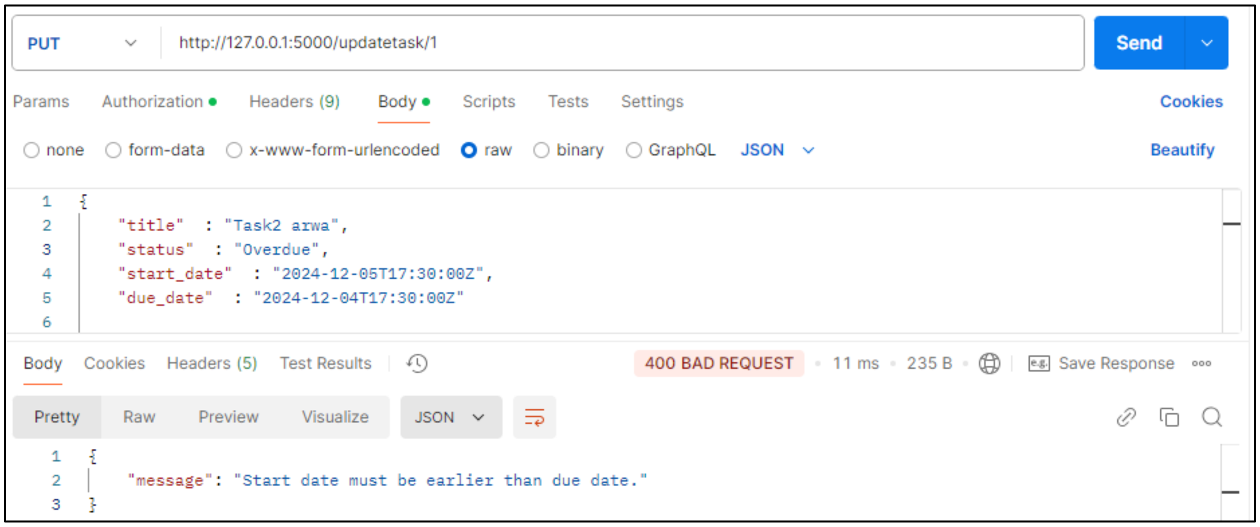
```
{
    "title"  : "Task2 arwa",
    "status"  : "Overduee",
    "start_date"  : "2024-12-05T17:30:00Z"

}
```

400 BAD REQUEST · 10 ms · 257 B

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

2. Start Date Must Be Earlier Than Due Date



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

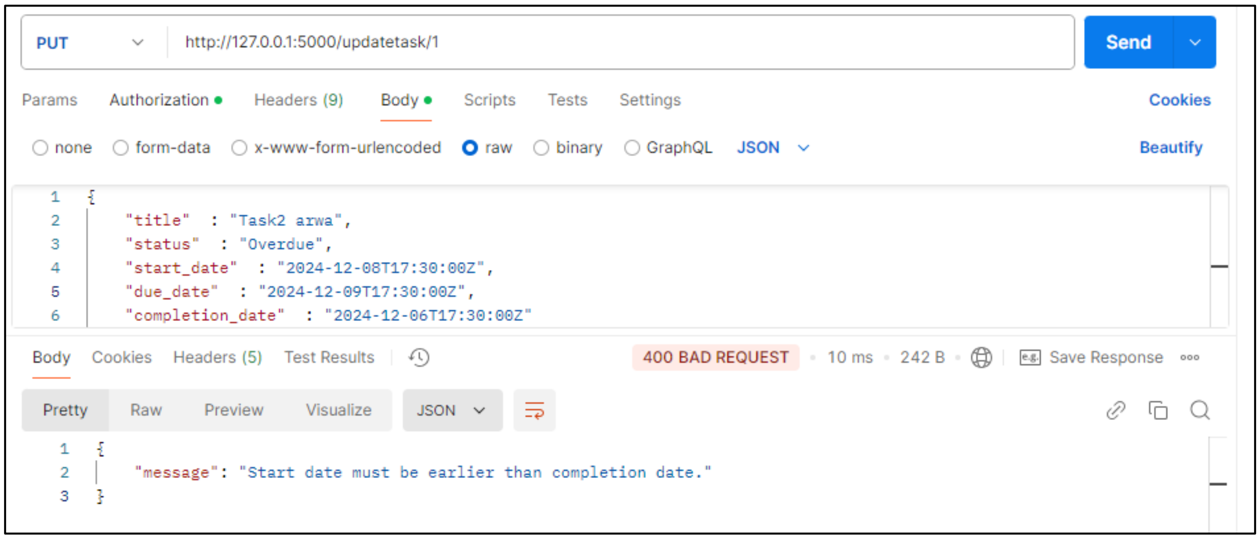
```
{
    "title"  : "Task2 arwa",
    "status"  : "Overdue",
    "start_date"  : "2024-12-05T17:30:00Z",
    "due_date"  : "2024-12-04T17:30:00Z"

```

400 BAD REQUEST · 11 ms · 235 B

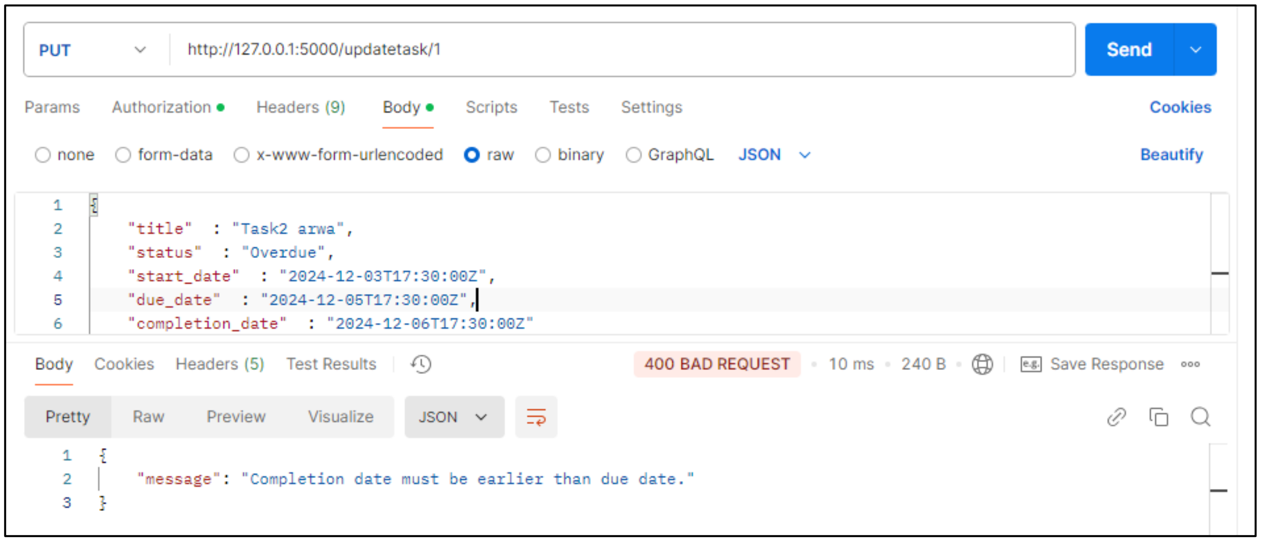
```
{
    "message": "Start date must be earlier than due date."
}
```

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)



```
{
    "title"  : "Task2 arwa",
    "status"  : "Overdue",
    "start_date"  : "2024-12-08T17:30:00Z",
    "due_date"  : "2024-12-09T17:30:00Z",
    "completion_date"  : "2024-12-06T17:30:00Z"
```

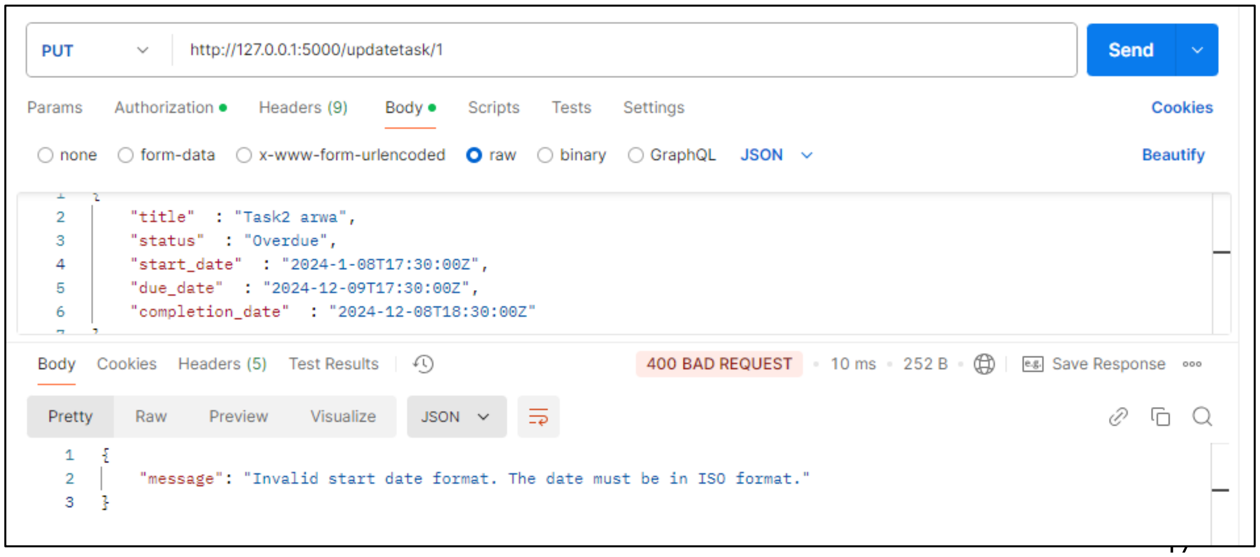
```
{
    "message": "Start date must be earlier than completion date."
}
```



```
{
    "title"  : "Task2 arwa",
    "status"  : "Overdue",
    "start_date"  : "2024-12-03T17:30:00Z",
    "due_date"  : "2024-12-05T17:30:00Z",
    "completion_date"  : "2024-12-06T17:30:00Z"
```

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

4. Date Must Be in a Valid Format



```
    "title"  : "Task2 arwa",
    "status"  : "Overdue",
    "start_date"  : "2024-1-08T17:30:00Z",
    "due_date"  : "2024-12-09T17:30:00Z",
    "completion_date"  : "2024-12-08T18:30:00Z"
```

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

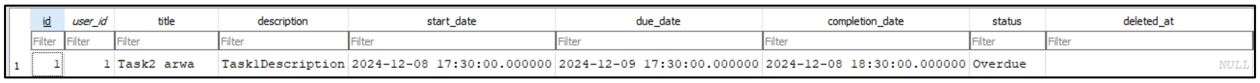
Delete tasks API Testing Steps:

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:

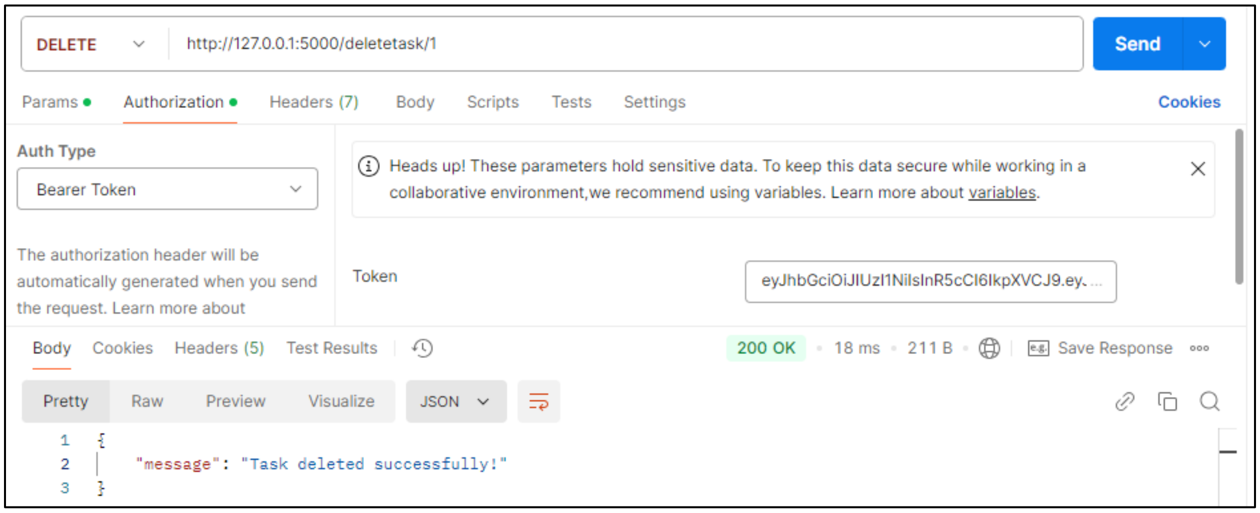
1. **task is deleted successfully**

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	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



DELETE http://127.0.0.1:5000/deletetask/1 Send

Params · Authorization · Headers (7) Body Scripts Tests Settings Cookies

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about

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.

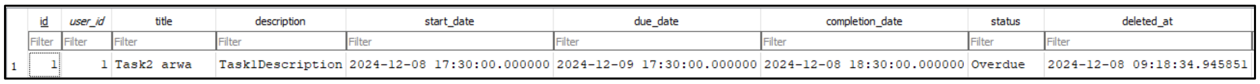
Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

Body Cookies Headers (5) Test Results 200 OK · 18 ms · 211 B Save Response

Pretty Raw Preview Visualize JSON

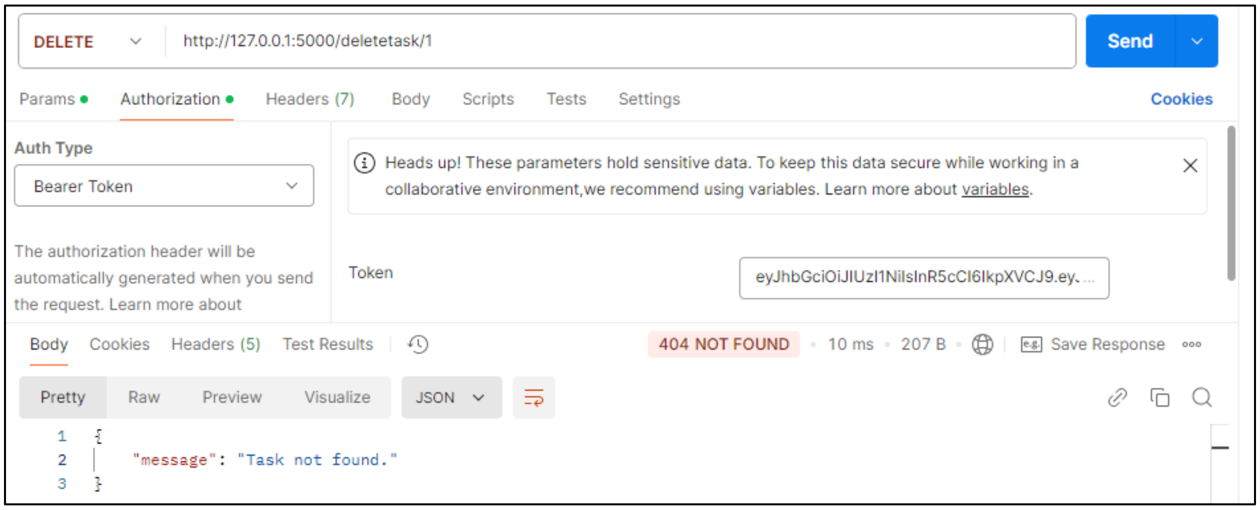
```
{
    "message": "Task deleted successfully!"
}
```

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	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)**



The authorization header will be automatically generated when you send the request. Learn more about

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.ey...

```
{
    "message": "Task not found."
}
```

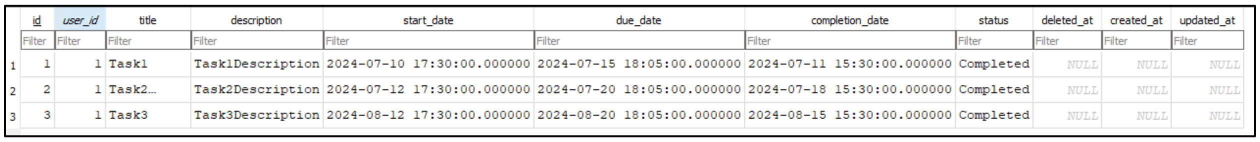
Batch delete tasks API Testing Steps:

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:

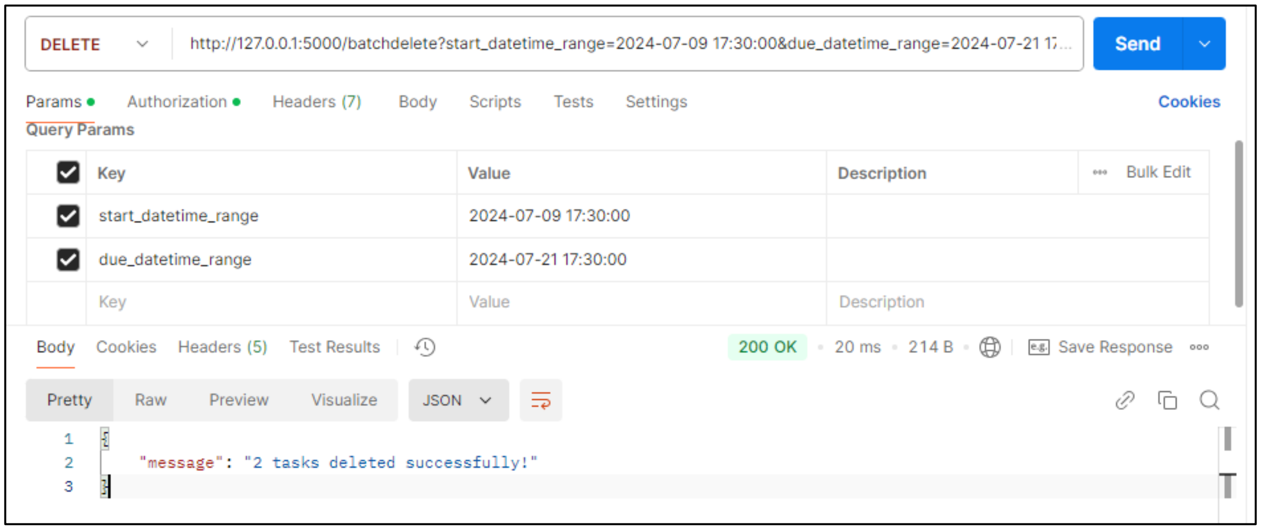
1. **Batch is deleted successfully**

   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	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	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	NULL	NULL	NULL
3	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	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...

Query Params

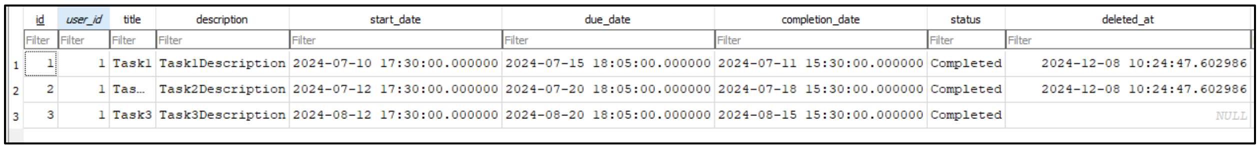
Key	Value	Description
start_datetime_range	2024-07-09 17:30:00	
due_datetime_range	2024-07-21 17:30:00	

200 OK · 20 ms · 214 B

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

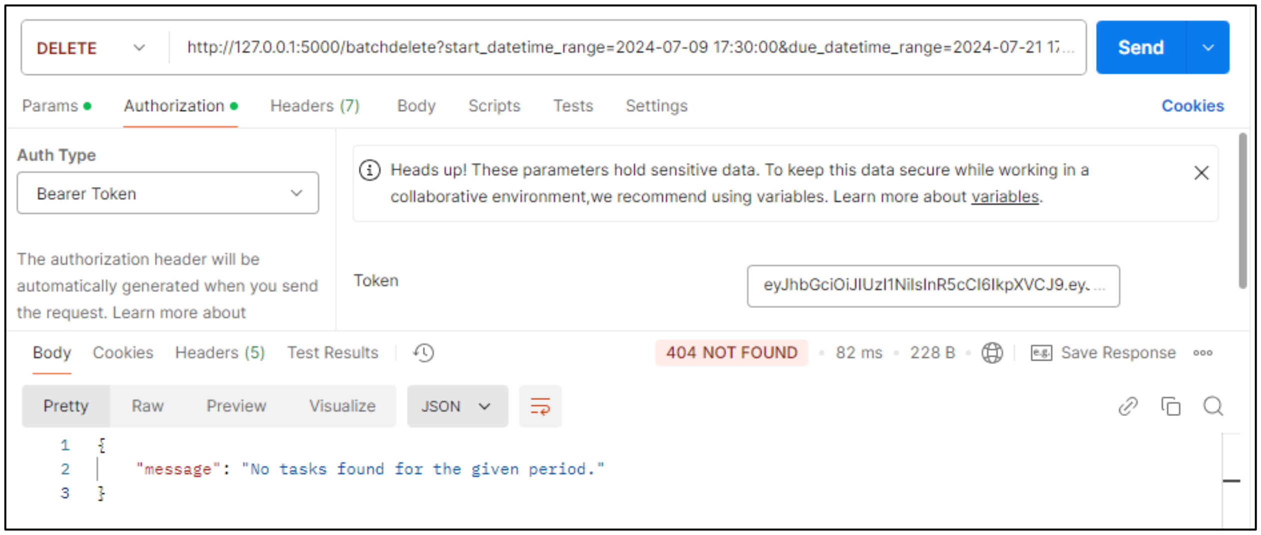
Date base after batchdelete

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	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	2	1	Tas…	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	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...

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.

The authorization header will be automatically generated when you send the request. Learn more about

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

404 NOT FOUND · 82 ms · 228 B

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

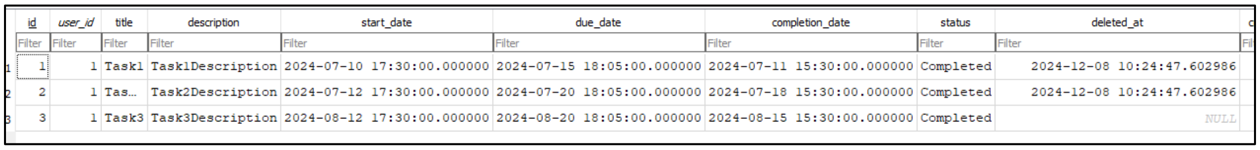
Restore tasks API Testing Steps:

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:

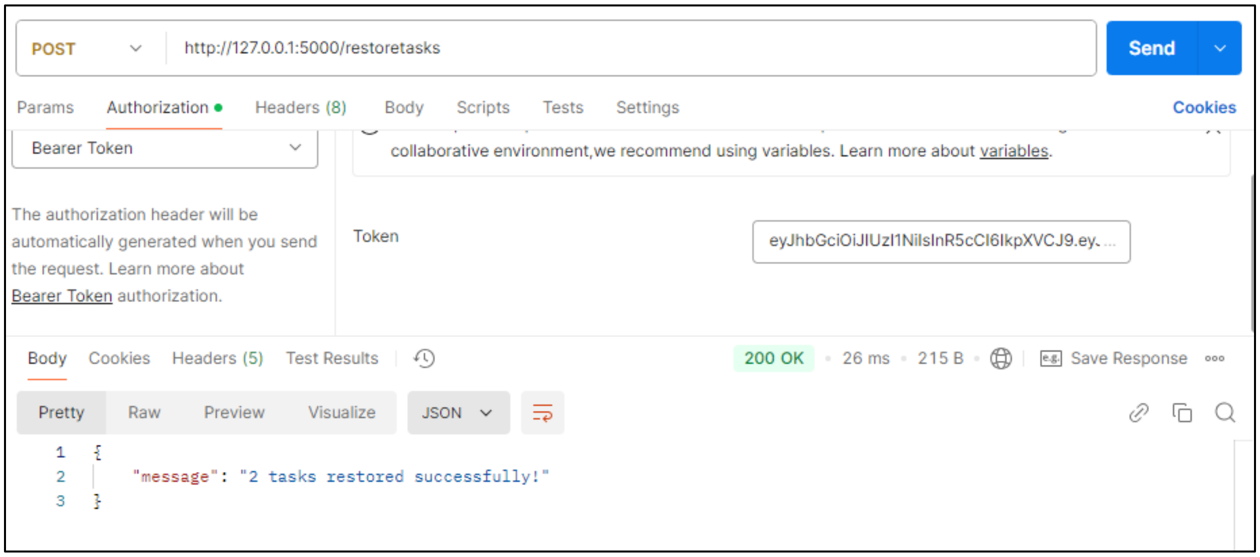
1. **Tasks are found and successfully restored**

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	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	2	1	Tas…	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	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



http://127.0.0.1:5000/restoretasks

Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

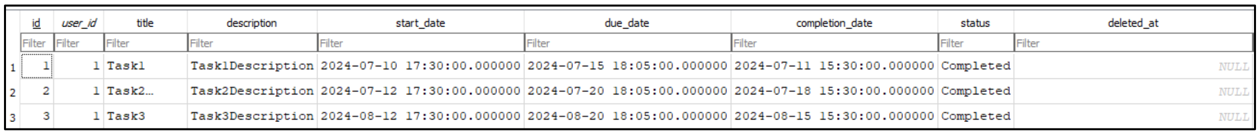
collaborative environment,we recommend using variables. Learn more about variables.

Token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

200 OK · 26 ms · 215 B

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

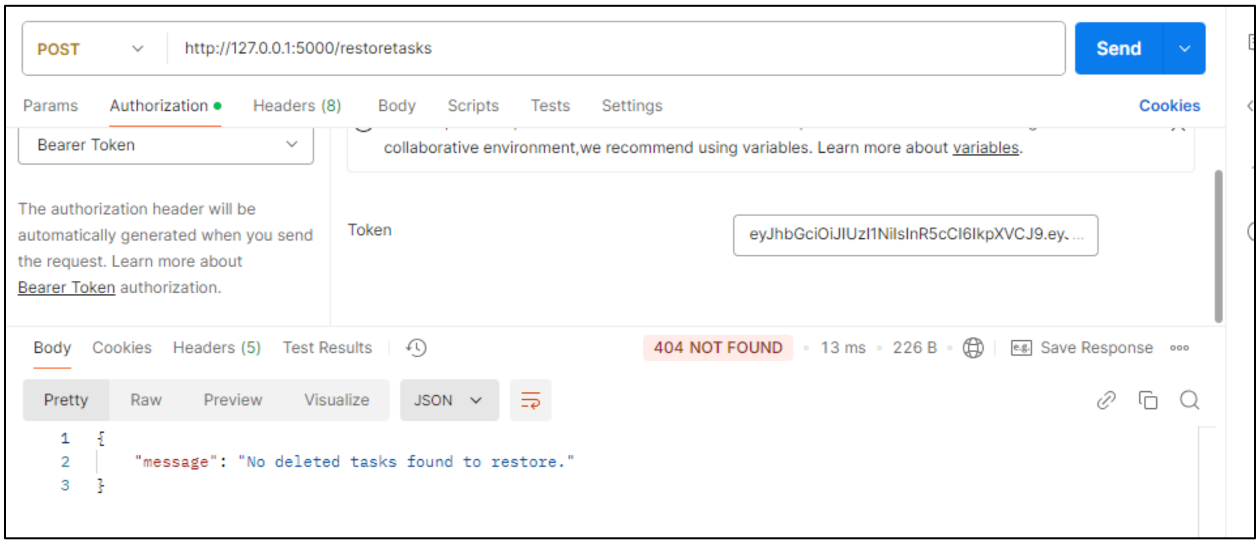
Database after restore tasks (Deleted at set to null)



	id	user_id	title	description	start_date	due_date	completion_date	status	deleted_at
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	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	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	NULL
3	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. **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

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

collaborative environment,we recommend using variables. Learn more about variables.

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

404 NOT FOUND · 13 ms · 226 B

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

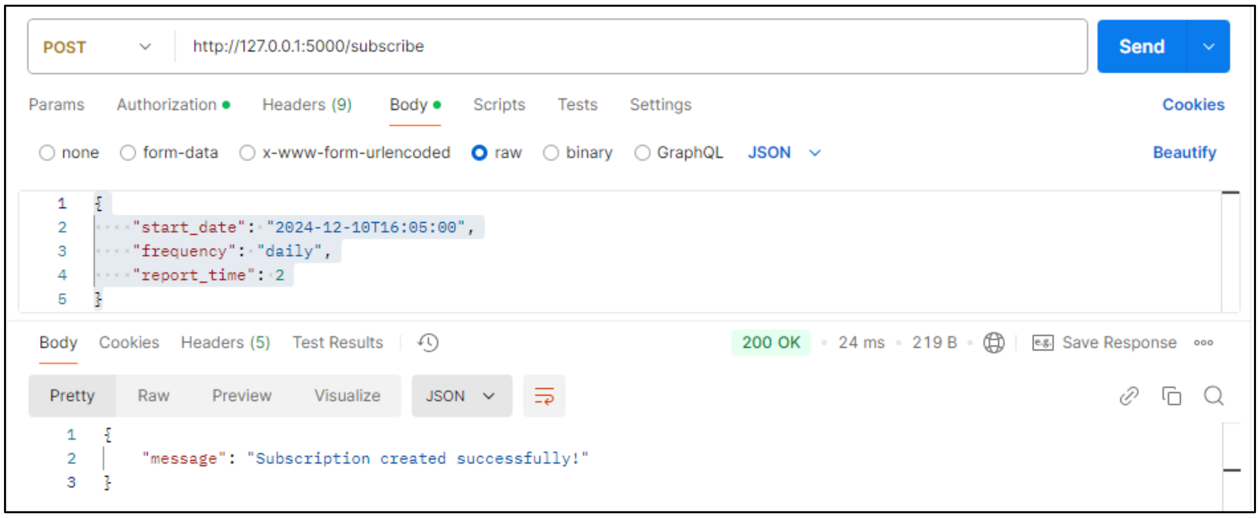
Subscription API

<u>Subscribe API Testing Steps:</u>

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**

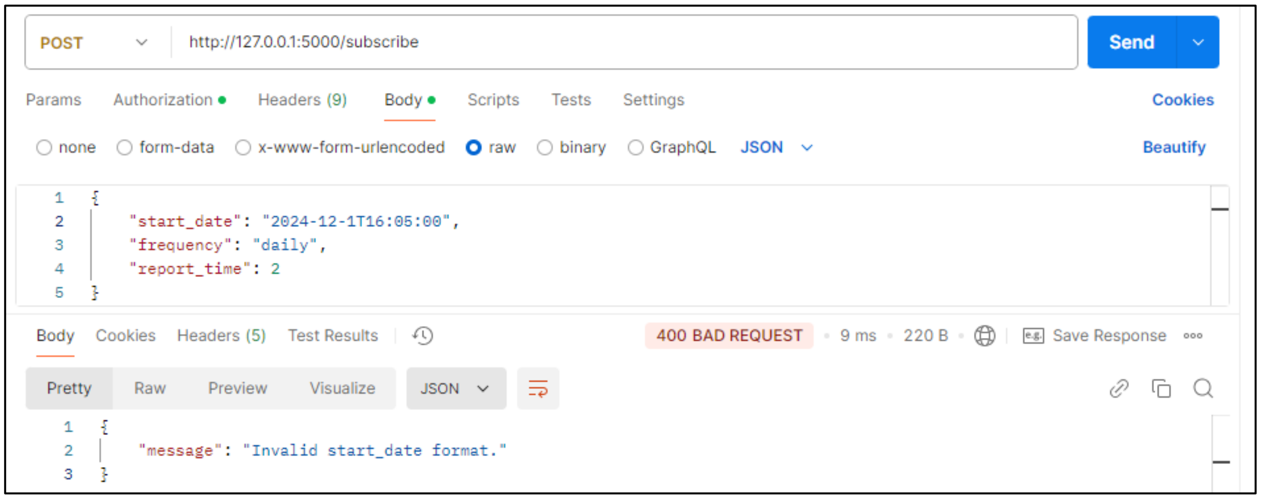

POST
http://127.0.0.1:5000/subscribe
Send
Params Authorization Headers (9) Body Scripts Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify

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

Body Cookies Headers (5) Test Results
200 OK 24 ms 219 B
Save Response
Pretty Raw Preview Visualize JSON

```
{
    "message": "Subscription created successfully!"
}
```

2. **Validations**
    1. Validate start_date in the correct format

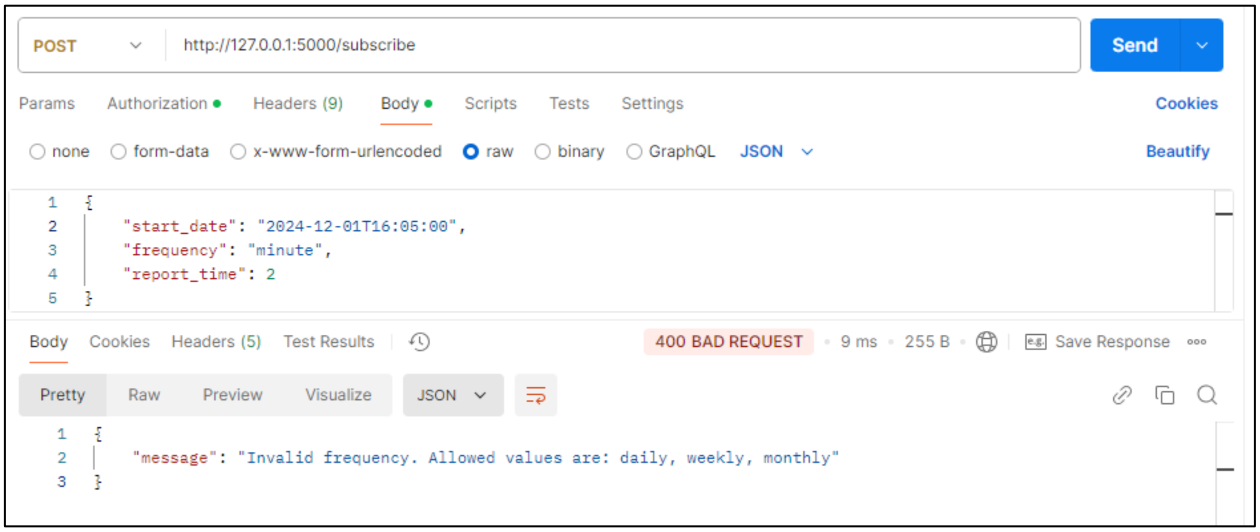

POST
http://127.0.0.1:5000/subscribe
Send
Params Authorization Headers (9) Body Scripts Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify

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

Body Cookies Headers (5) Test Results
400 BAD REQUEST 9 ms 220 B
Save Response
Pretty Raw Preview Visualize JSON

```
{
    "message": "Invalid start_date format."
}
```

    2. Validate frequency

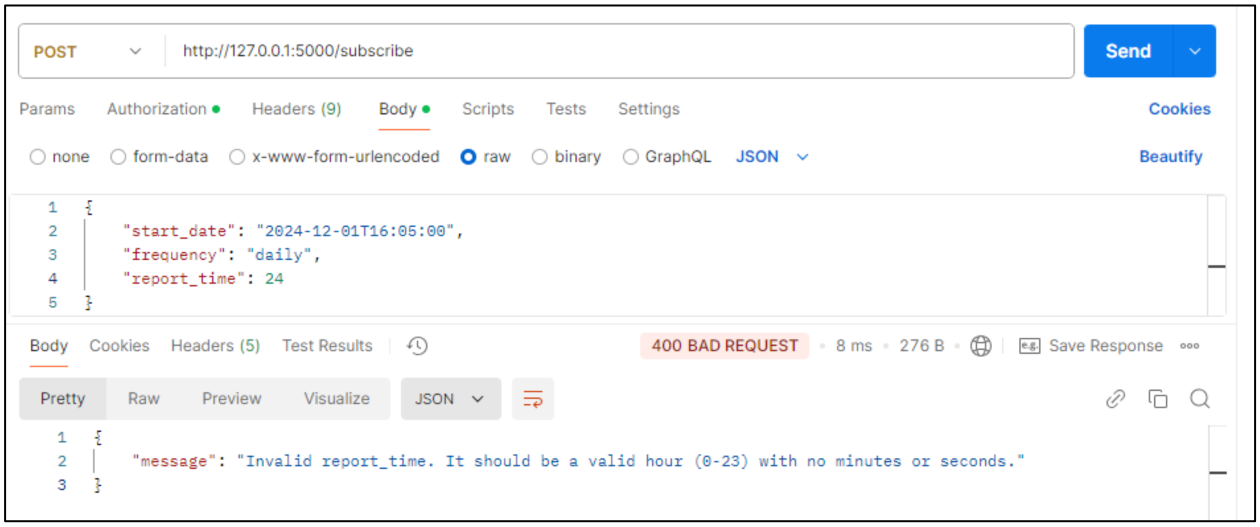

POST
http://127.0.0.1:5000/subscribe
Send
Params Authorization Headers (9) Body Scripts Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify

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

Body Cookies Headers (5) Test Results
400 BAD REQUEST 9 ms 255 B
Save Response
Pretty Raw Preview Visualize JSON

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

3. Validate report_time



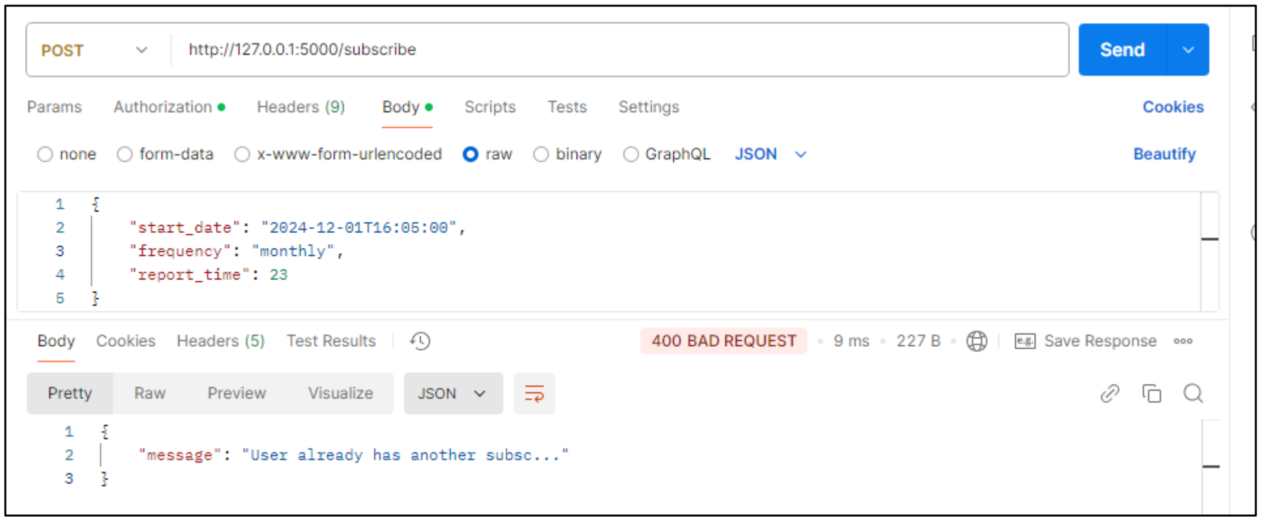
http://127.0.0.1:5000/subscribe

```
{
    "start_date": "2024-12-01T16:05:00",
    "frequency": "daily",
    "report_time": 24
}
```

400 BAD REQUEST · 8 ms · 276 B

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

4. User has only one subscription



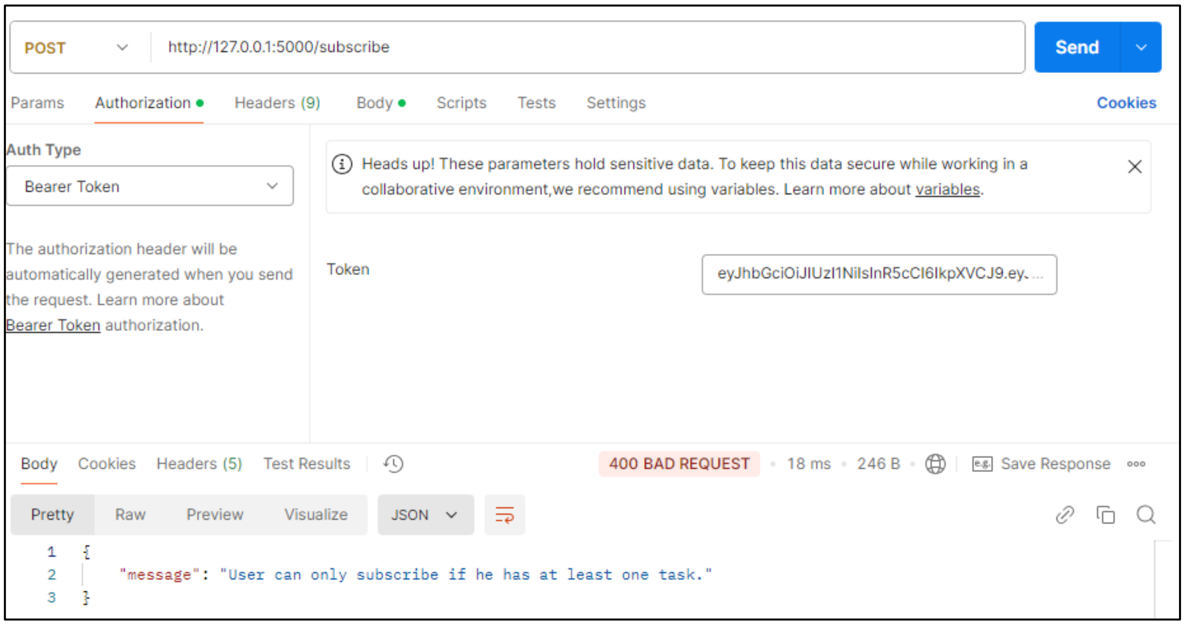
http://127.0.0.1:5000/subscribe

```
{
    "start_date": "2024-12-01T16:05:00",
    "frequency": "monthly",
    "report_time": 23
}
```

400 BAD REQUEST · 9 ms · 227 B

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

5. User has at least one task


POST
Send
Params
Authorization
Headers (9)
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.
The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.
Body
Cookies
Headers (5)
Test Results
18 ms
246 B
Save Response
Pretty
Raw
Preview
Visualize
JSON

http://127.0.0.1:5000/subscribe

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

400 BAD REQUEST

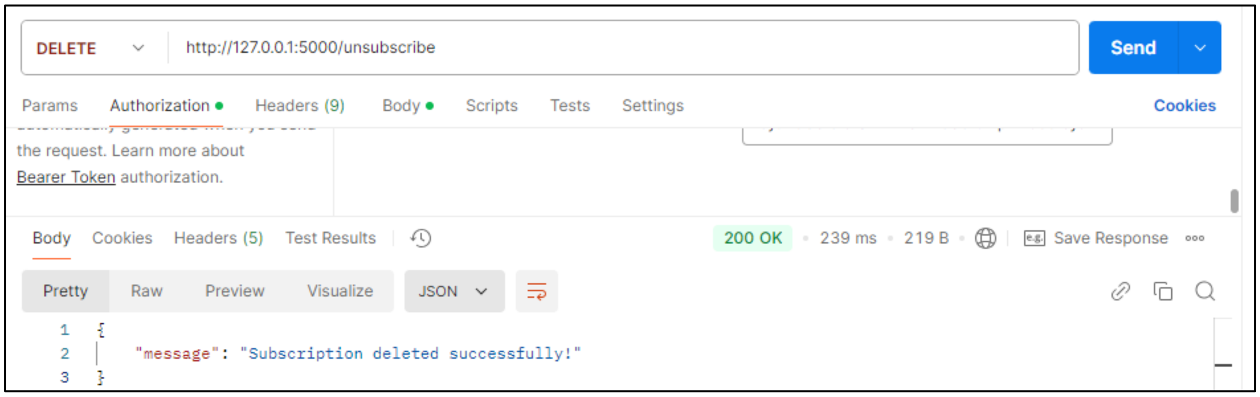
```
{
    "message": "User can only subscribe if he has at least one task."
}
```

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

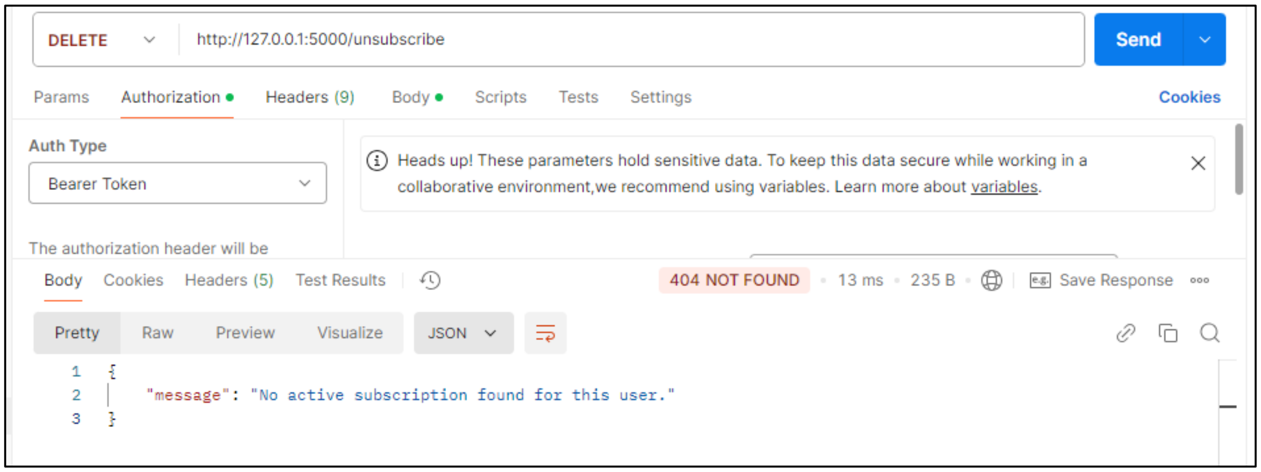


http://127.0.0.1:5000/unsubscribe

200 OK

```
{
    "message": "Subscription deleted successfully!"
}
```

2. If no subscription found


DELETE http://127.0.0.1:5000/unsubscribe

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.

The authorization header will be

404 NOT FOUND · 13 ms · 235 B

```
{
    "message": "No active subscription found for this user."
}
```

Report Generation

Automated Task Report Implementation

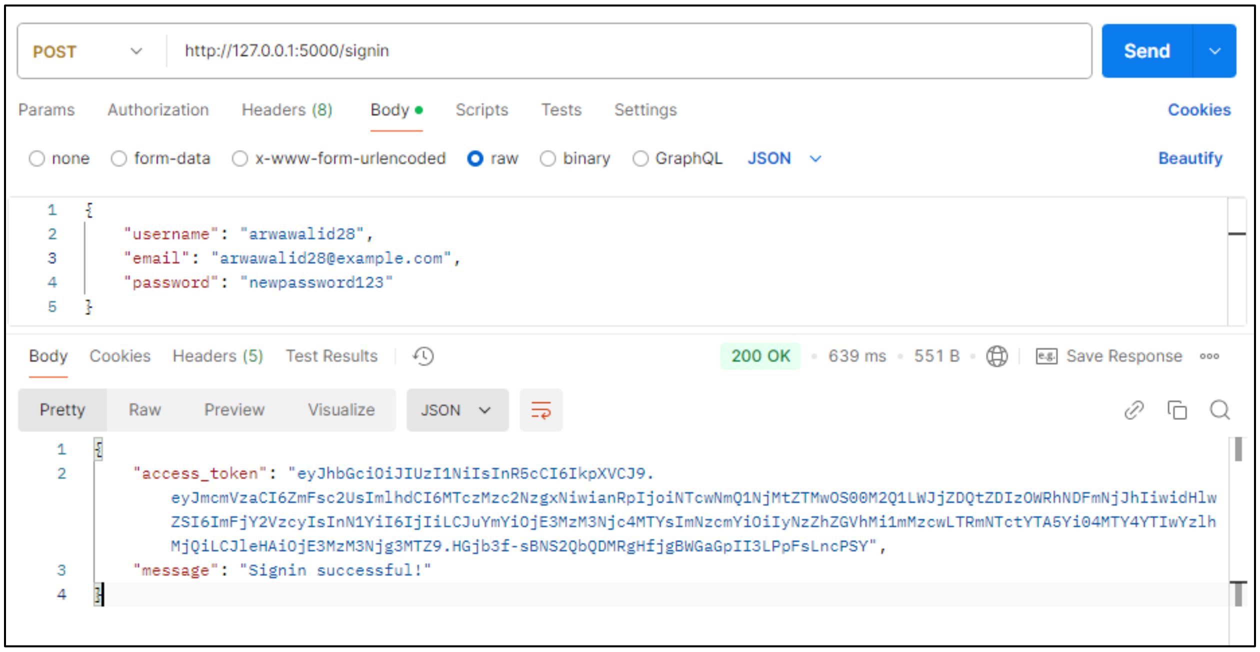
1. Flask App Context Setup
2. Retrieve all users from database
3. Check if user have a subscription. If yes extract user_id, frequency, report_time, next_send_time
   Note **: next_send_time** is next time to generate report and it initially set to be equal subscription start date
4. Check if it is the 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

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)



POST http://127.0.0.1:5000/signin

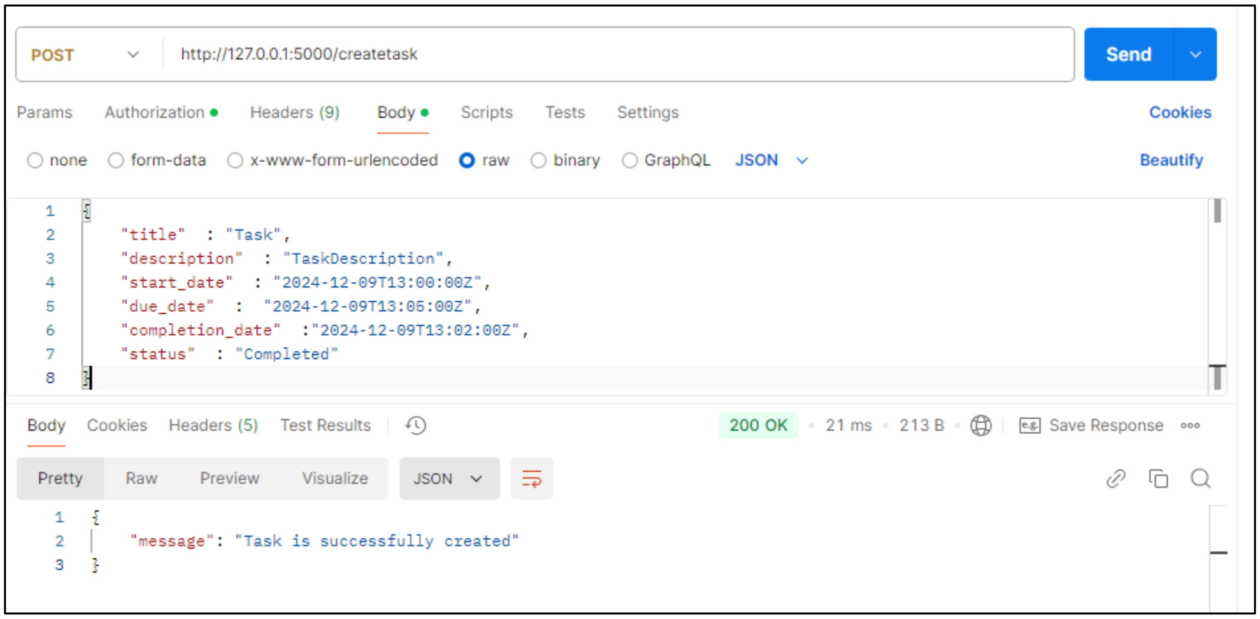
Body: raw JSON

```
{
    "username": "arwawalid28",
    "email": "arwawalid28@example.com",
    "password": "newpassword123"
}
```

200 OK · 639 ms · 551 B

```
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczMzc2NzgxNiwianRpIjoiNTcwNmQ1NjMtZTMwOS00M2Q1LWJjZDQtZDIzOWRhNDFmNjJhIiwidHlw
        ZSI6ImFjY2VzcyIsInN1YiI6IjIiLCJuYmYiOjE3MzM3Njc4MTYsImNzcmYiOiIyNzZhZGVhMi1mMzcwLTRmNTctYTA5Yi04MTY4YTIwYzlh
        MjQiLCJleHAiOjE3MzM3Njg3MTZ9.HGjb3f-sBNS2QbQDMRgHfjgBWGaGpII3LPpFsLncPSY",
    "message": "Signin successful!"
}
```

   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

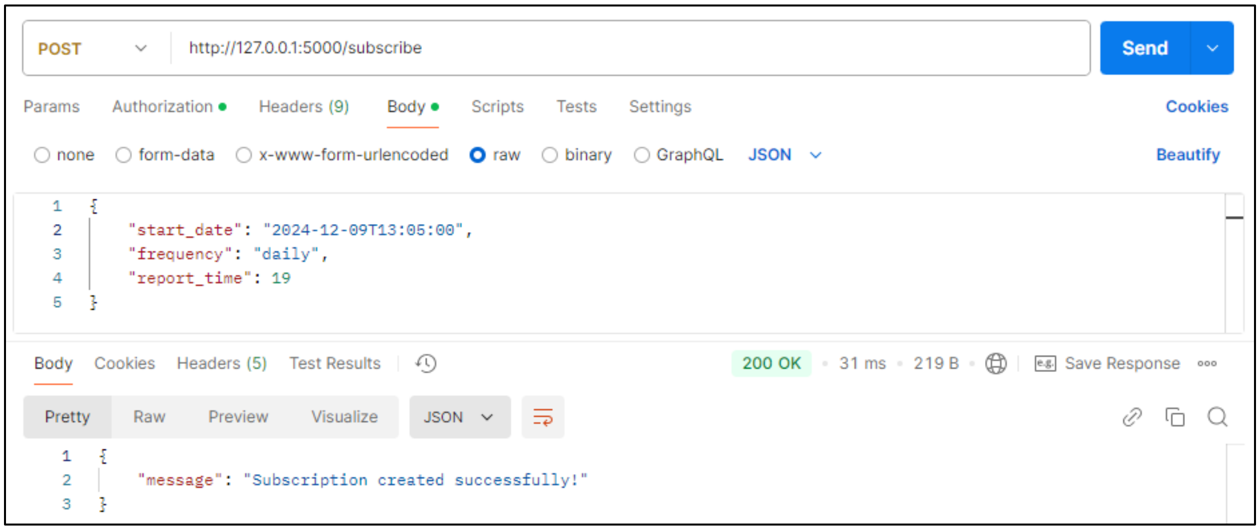
Body: raw JSON

```
{
    "title"  : "Task",
    "description"  : "TaskDescription",
    "start_date"  : "2024-12-09T13:00:00Z",
    "due_date"  :  "2024-12-09T13:05:00Z",
    "completion_date"  :"2024-12-09T13:02:00Z",
    "status"  : "Completed"
}
```

200 OK · 21 ms · 213 B

```
{
    "message": "Task is successfully created"
}
```

3. User subscription



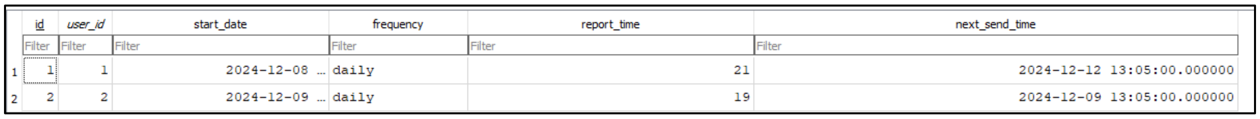
http://127.0.0.1:5000/subscribe

```
{
    "start_date": "2024-12-09T13:05:00",
    "frequency": "daily",
    "report_time": 19
}
```

200 OK · 31 ms · 219 B

```
{
    "message": "Subscription created successfully!"
}
```

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

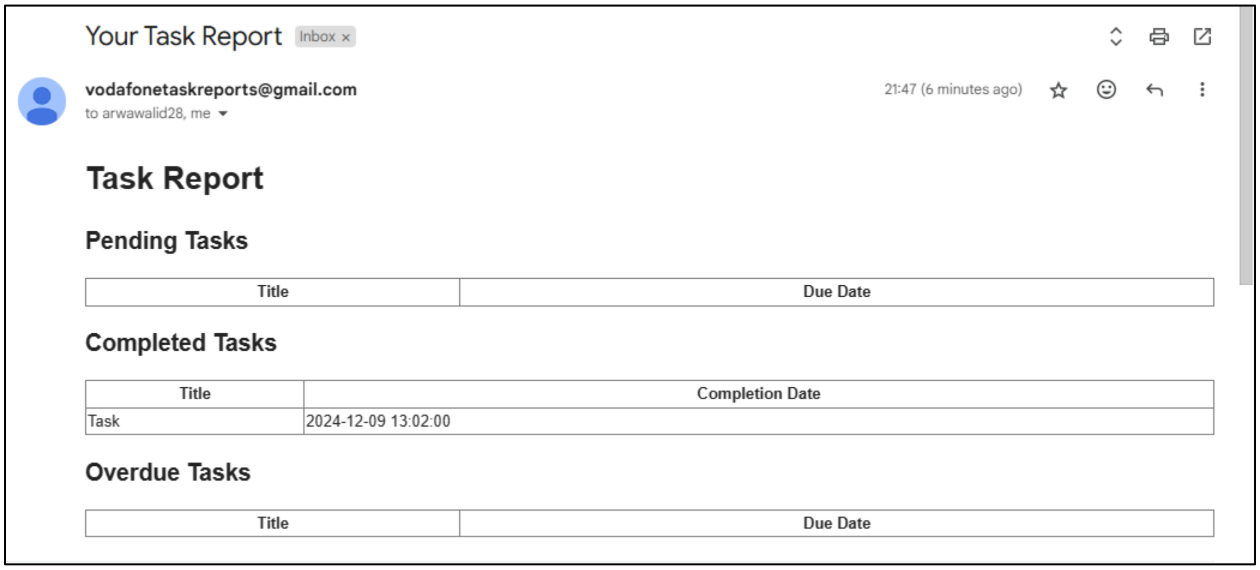


	id	user_id	start_date	frequency	report_time	next_send_time
1	1	1	2024-12-08 ...	daily	21	2024-12-12 13:05:00.000000
2	2	2	2024-12-09 ...	daily	19	2024-12-09 13:05:00.000000

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

vodafonetaskreports@gmail.com
to arwawalid28, me
21:47 (6 minutes ago)

Task Report

Pending Tasks

Title	Due Date

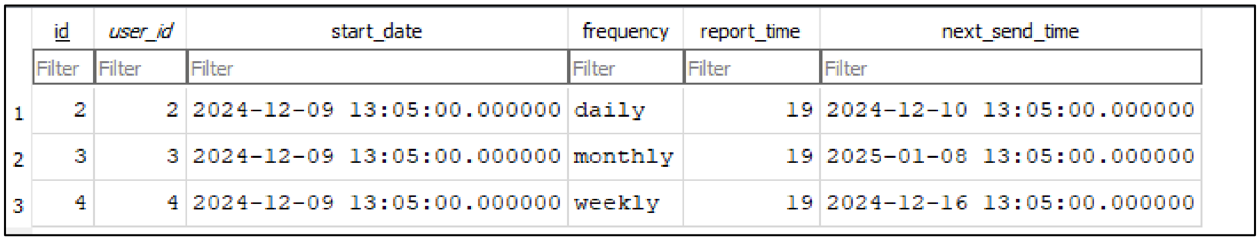
Completed Tasks

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

Overdue Tasks

Title	Due Date

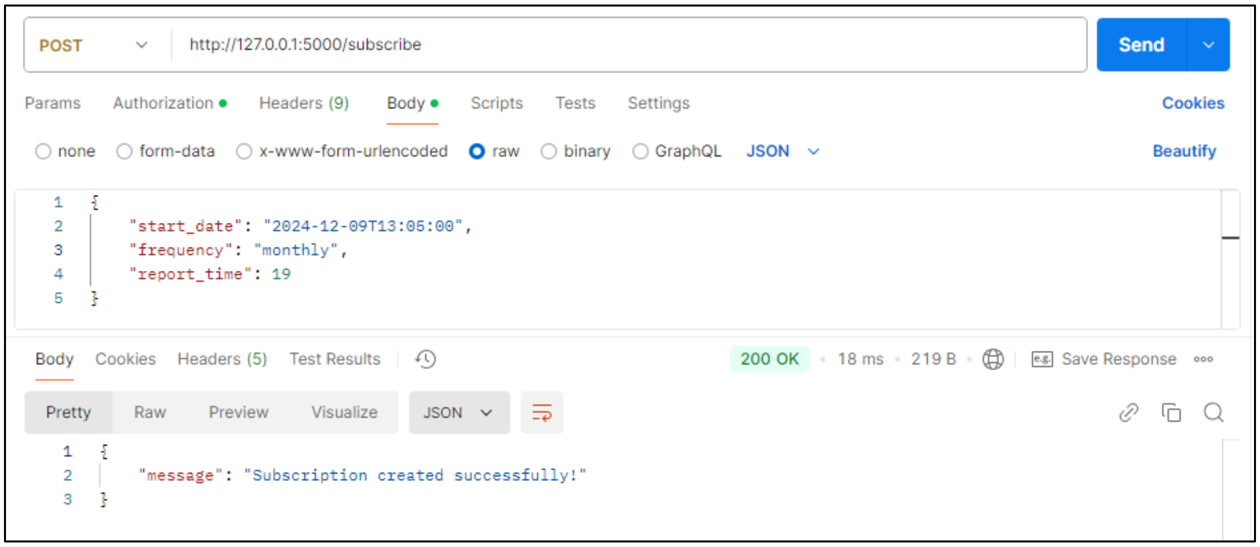
6. next_send_time = next_send_time + 24



	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 (user_id =3)
2. create tasks for the user during last month
3. User subscription



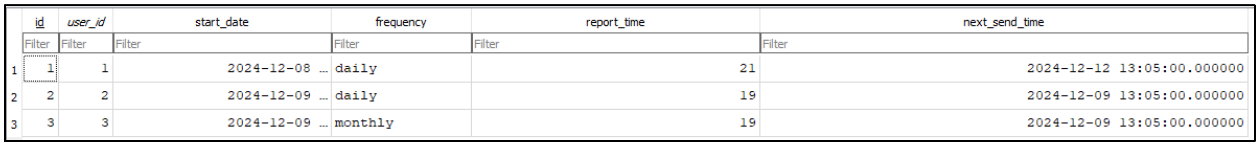
POST http://127.0.0.1:5000/subscribe

```
{
    "start_date": "2024-12-09T13:05:00",
    "frequency": "monthly",
    "report_time": 19
}
```

200 OK · 18 ms · 219 B

```
{
    "message": "Subscription created successfully!"
}
```

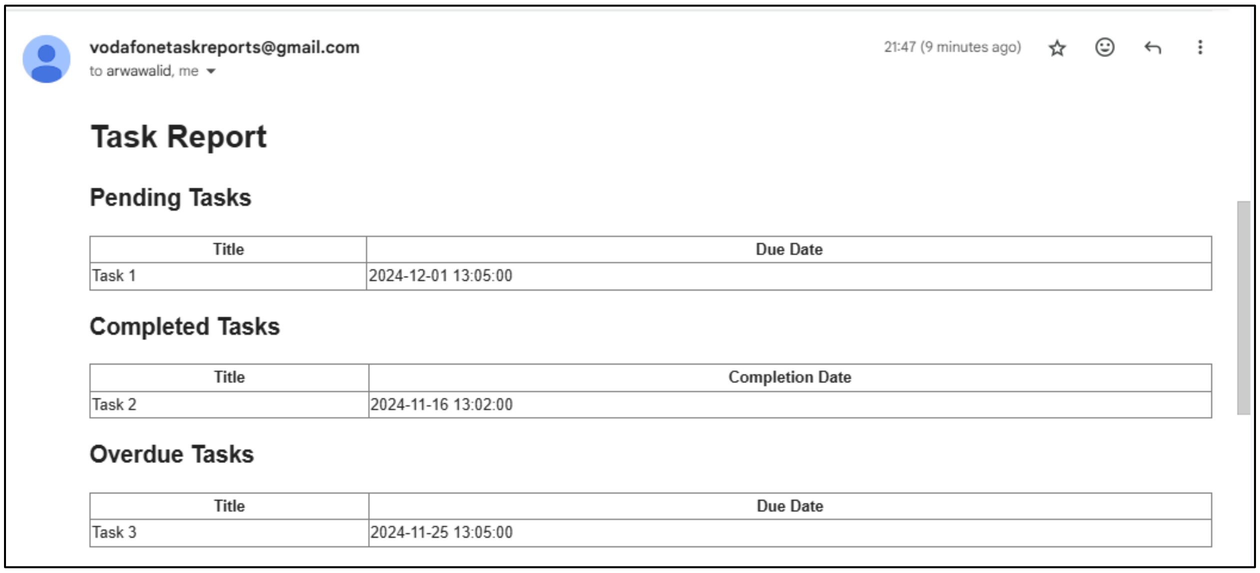
next_send_time = subscription_start_date = "2024-12-09T13:05:00"



	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

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)



vodafonetaskreports@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

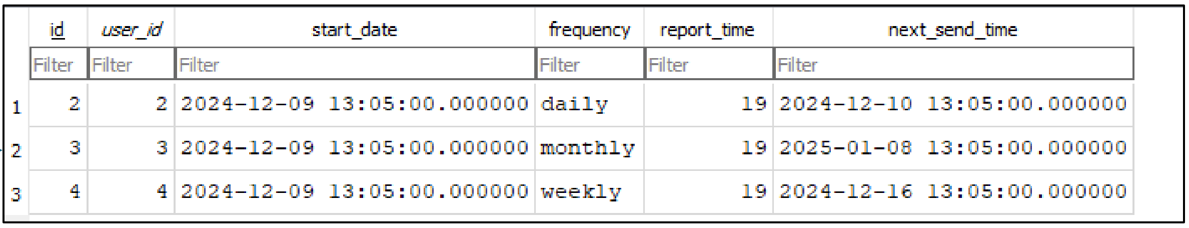
Completed Tasks

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

Overdue Tasks

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

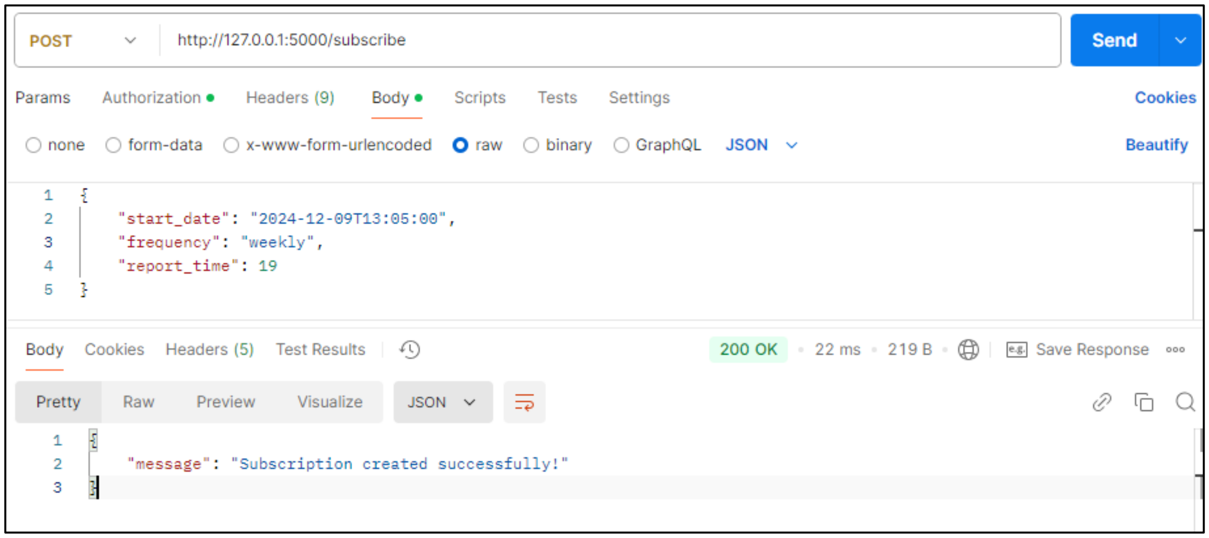
5. next_send_time = next_send_time + (24*30)



	id	user_id	start_date	frequency	report_time	next_send_time
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 wittasks 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



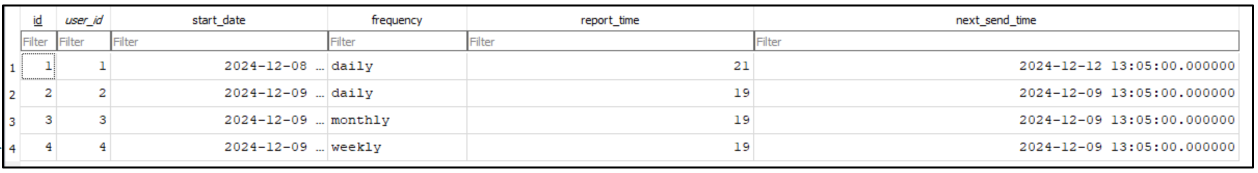
POST http://127.0.0.1:5000/subscribe

```
{
    "start_date": "2024-12-09T13:05:00",
    "frequency": "weekly",
    "report_time": 19
}
```

200 OK · 22 ms · 219 B

```
{
    "message": "Subscription created successfully!"
}
```

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

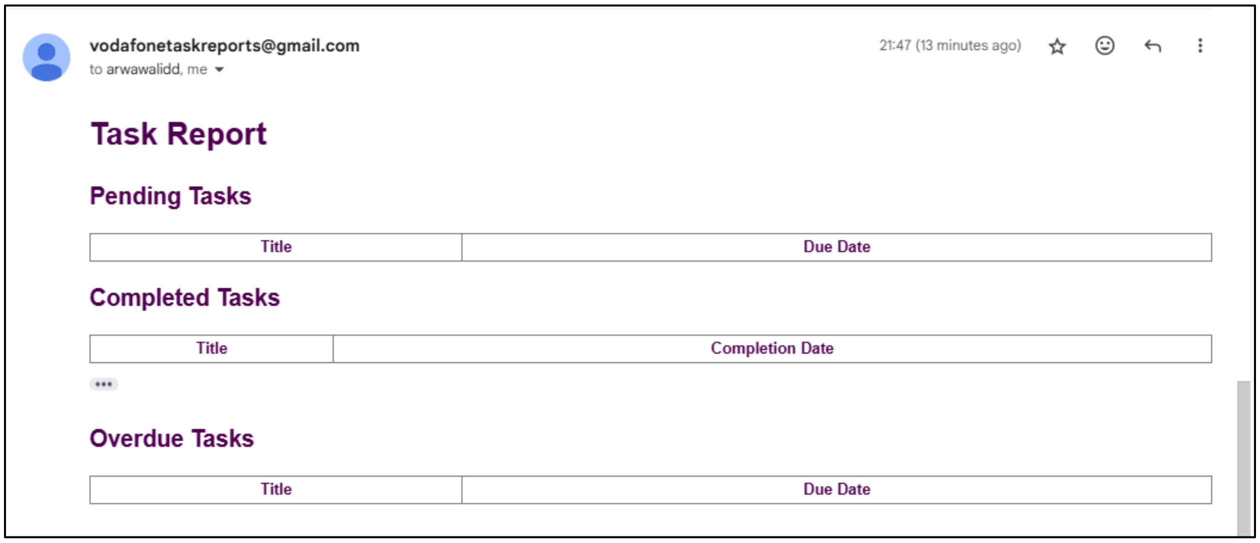
4. Report Time Check:

    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)



vodafonetaskreports@gmail.com
to arwawalidd, me

21:47 (13 minutes ago)

Task Report

Pending Tasks

Title	Due Date

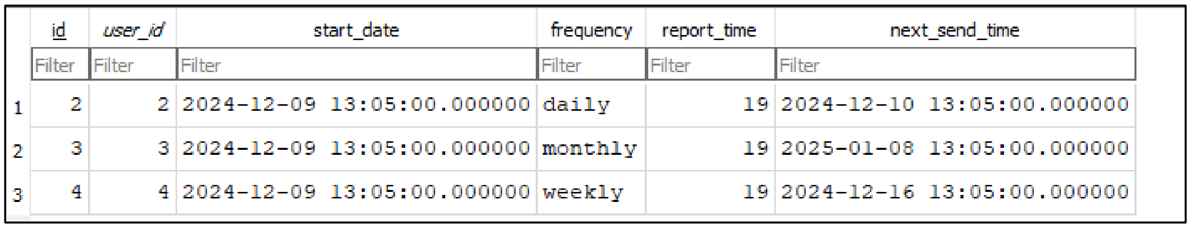
Completed Tasks

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