



TASK-API DOCUMENTATION

Eat Ziffy Python Internship Task

This Documentation displays the usage and testing of API

Prajwol Chhetri

Table of Contents

1. Configuring Postgres as Default Database.....	2
2. Authentication using JWT Token	2
3. Creating Custom User Model.....	3
4. Creating Custom User Manager	4
5. Login Using Email and Password	5
6. Register a User.....	6
7. Access Task endpoint without authentication	7
8. Admin Permissions	8
9. User Permissions.....	10
10. Unit testing for each Individual Endpoints.....	13

1. Configuring Postgres as Default Database

The default database used in Django is MySQL. Postgres was configured as the default database as per requirement.

```
97 DATABASES = {
98     'default': {
99         'ENGINE': 'django.db.backends.postgresql_psycopg2',
100         'NAME': 'postgres',
101         'USER': 'postgres',
102         'PASSWORD': '12345678',
103         'HOST': 'localhost',
104         'PORT': '5432',
105     }
106 }
```

Figure 1 Configuring POSTGRES as default db

2. Authentication using JWT Token

The authentication in the API was done using JWT token. Lifetime of access token was increased through the settings and configured it to be 60 minutes.

```
79 REST_FRAMEWORK = {
80     'DEFAULT_AUTHENTICATION_CLASSES': (
81         'rest_framework_simplejwt.authentication.JWTAuthentication',
82     )
83 }
84
85
86 SIMPLE_JWT = {
87     'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
88     'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
89
90     'AUTH_HEADER_TYPES': ('Bearer',),
91 }
```

Figure 2 Using JWT for authentication

3. Creating Custom User Model

A custom user model to register the user using their email was created.

```
class User(AbstractBaseUser, PermissionsMixin):
    """Database model for users in the system"""
    email = models.EmailField(max_length=255, unique=True)
    first_name = models.CharField(max_length=125)
    last_name = models.CharField(max_length=125)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)

    objects = UserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name']

    def get_full_name(self):
        """Retrieve full name of user"""
        return self.first_name+" "+self.last_name

    def get_short_name(self):
        """Retrieve short name of user"""
        return self.first_name

    def __str__(self):
        """Return string representation of our user"""
        return self.email
```

Figure 3 Custom User Model

4. Creating Custom User Manager

To register the user model a manager was created.

```
class UserManager(BaseUserManager):
    """Manager for Users. Creating a manager to handle the model because as default Django
    requires Username and Password Field to create a user but we've changed this to Email field"""

    def create_user(self, email, first_name, last_name, password=None):
        """Create a new user"""
        if not email:
            raise ValueError('User must have an email address')

        # normalizing email for standarization
        email = self.normalize_email(email)
        # creating user model that user manager is representing
        user = self.model(email=email, first_name=first_name, last_name=last_name)
        # Encrypting password using method of AbstractBaseUserClass
        user.set_password(password)
        # self._db to save to any database
        user.save(using=self._db)

        return user

    def create_superuser(self, email, first_name, last_name, password):
        """create and save new superuser with given details"""
        user = self.create_user(email, first_name, last_name, password)

        user.is_superuser = True
        user.is_staff = True
        user.save(using=self._db)

        return user
```

Figure 4 User manager for User Model

5. Login Using Email and Password

The user can login using their email and password. After the login is successful user receives an access token which is valid for 60 minutes and a refresh token which is valid for 1 day. The access token received by the user is used to authenticate various endpoints.

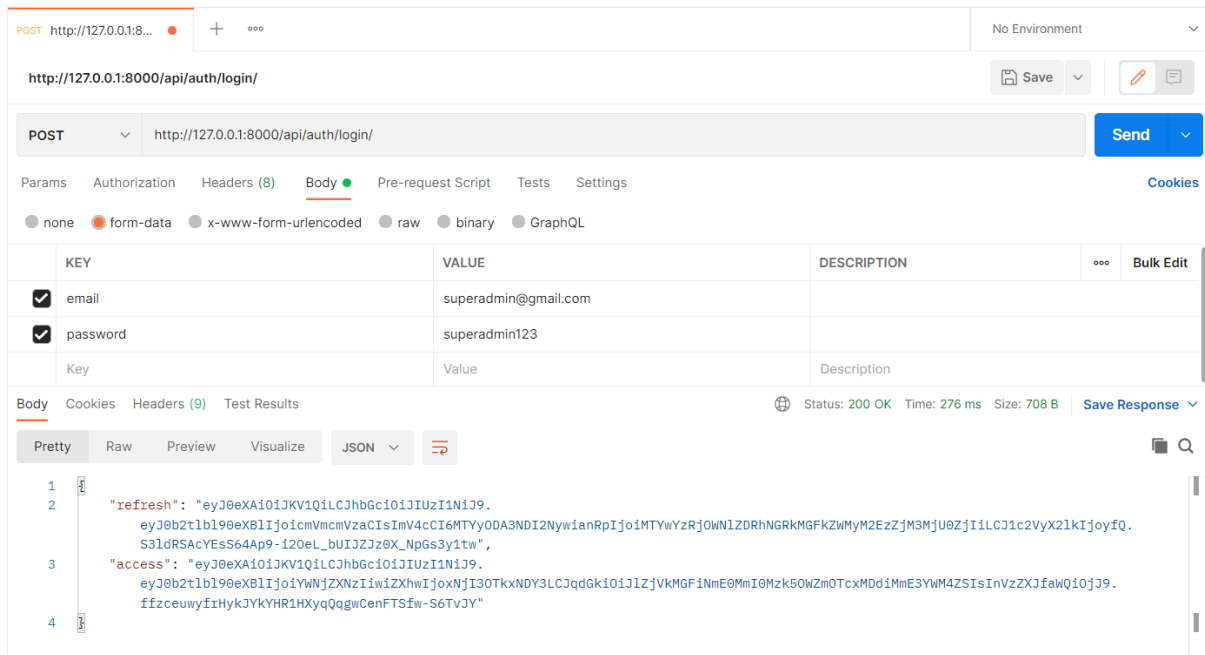
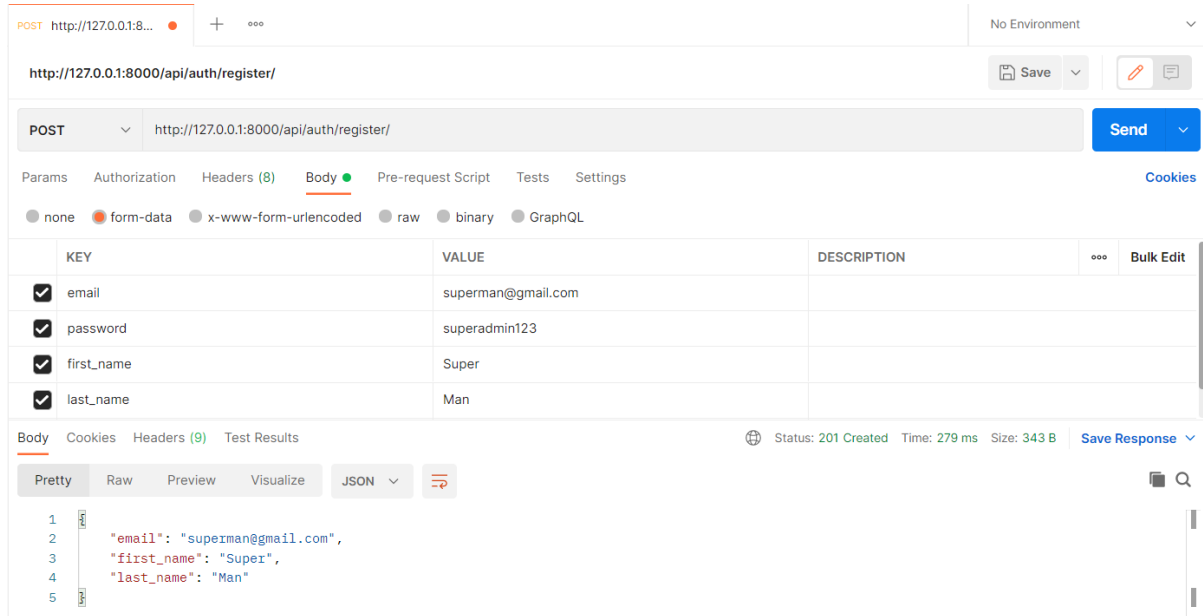


Figure 6 User Login

6. Register a User

The users can be registered in the system if they provide their first name, last name, email and valid password.



The screenshot displays a REST client interface with the following details:

- Request Method:** POST
- URL:** http://127.0.0.1:8000/api/auth/register/
- Form Data:**

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> email	superman@gmail.com	
<input checked="" type="checkbox"/> password	superadmin123	
<input checked="" type="checkbox"/> first_name	Super	
<input checked="" type="checkbox"/> last_name	Man	
- Response Body (JSON):**

```
1 {  
2   "email": "superman@gmail.com",  
3   "first_name": "Super",  
4   "last_name": "Man"  
5 }
```
- Status:** 201 Created
- Time:** 279 ms
- Size:** 343 B

Figure 7 User Registration

7. Access Task endpoint without authentication

User can't access task endpoint without providing authorization token which was provided during login before.

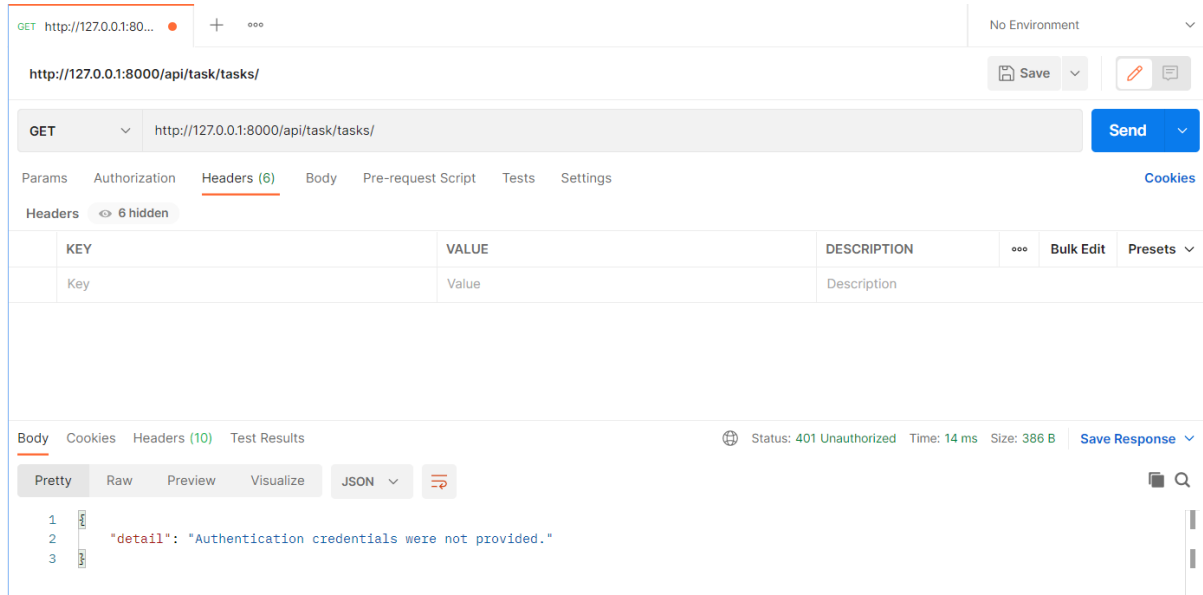


Figure 9 Task endpoint before authorization

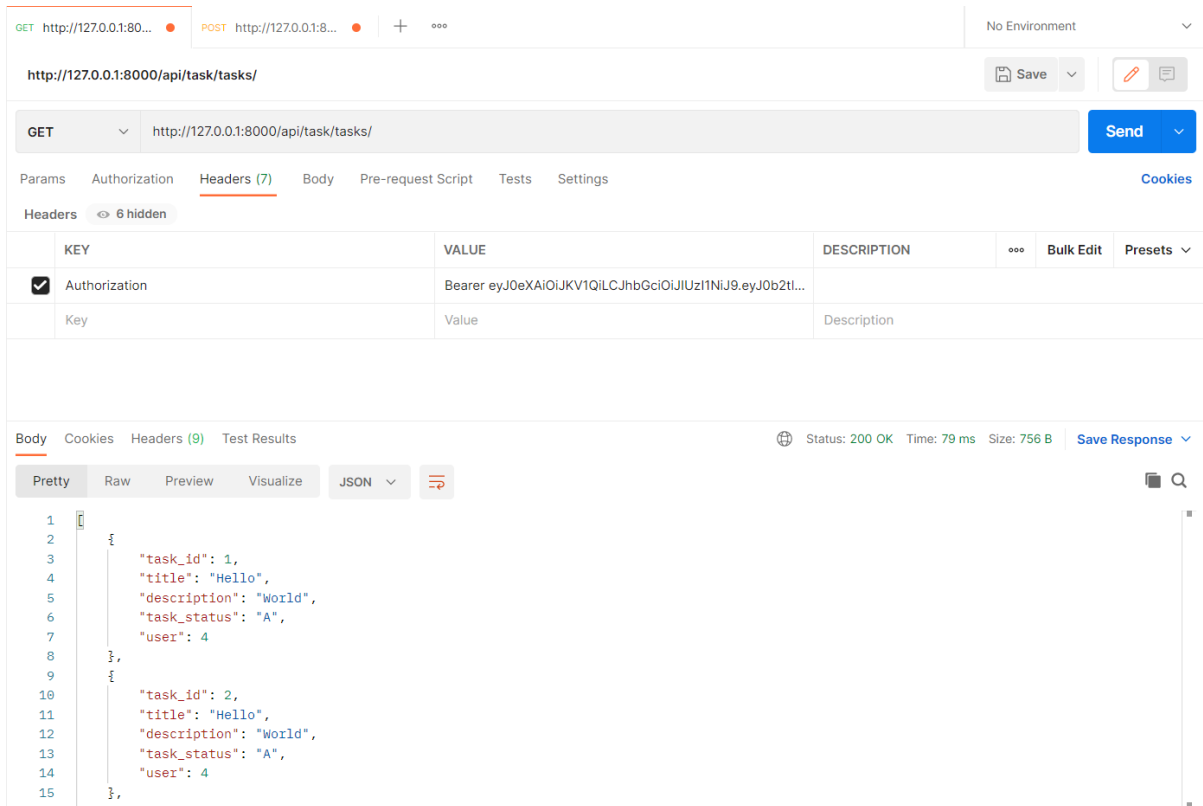


Figure 8 Task endpoint after authorization

8. Admin Permissions

The admin can see the list of all tasks.

GET http://127.0.0.1:8000/api/task/tasks/

Headers (9)

KEY	VALUE	DESCRIPTION
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t...	

Body

```
8  },
9  {
10     "task_id": 2,
11     "title": "Hello",
12     "description": "World",
13     "task_status": "A",
14     "user": 4
15  },
16  {
17     "task_id": 3,
18     "title": "Hello",
19     "description": "World",
20     "task_status": "A",
21     "user": 4
22  },
23  {
24     "task_id": 5,
25     "title": "Hello",
26     "description": "World",
27     "task_status": "A",
28     "user": 4
29  }
```

Status: 200 OK Time: 176 ms Size: 756 B

The admin can create task for others as well as him/herself.

POST http://127.0.0.1:8000/api/task/tasks/

Body

```
1  {
2     "task_id": 8,
3     "title": "Test Task",
4     "description": "lorem ipsum",
5     "task_status": "A",
6     "user": 2
7  }
```

Status: 201 Created Time: 78 ms Size: 373 B

Figure 10 Admin Creating task for himself

POST http://127.0.0.1:8000/api/task/tasks/ No Environment

Save Send

POST http://127.0.0.1:8000/api/task/tasks/ Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

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

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	task_id	9			
<input checked="" type="checkbox"/>	title	Test Task			
<input checked="" type="checkbox"/>	description	lorem ipsum			
<input checked="" type="checkbox"/>	task_status	A			
<input checked="" type="checkbox"/>	user	7			

Body Cookies Headers (9) Test Results Status: 201 Created Time: 110 ms Size: 373 B Save Response

Pretty Raw Preview Visualize JSON

```
1  {
2    "task_id": 9,
3    "title": "Test Task",
4    "description": "lorem ipsum",
5    "task_status": "A",
6    "user": 7
7  }
```

Figure 11 Admin Creating Task for Others

9. User Permissions

User can only see tasks that are assigned to that particular user.

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8000/api/task/tasks/`. The request includes an Authorization header with a Bearer token. The response is a JSON array of three tasks, all assigned to user 4.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tl...	
Key	Value	Description

Body: Cookies Headers (9) Test Results Status: 200 OK Time: 78 ms Size: 677 B Save Response

```
10  {
11    "task_id": 2,
12    "title": "Hello",
13    "description": "World",
14    "task_status": "A",
15    "user": 4
16  },
17  {
18    "task_id": 3,
19    "title": "Hello",
20    "description": "World",
21    "task_status": "A",
22    "user": 4
23  },
24  {
25    "task_id": 5,
26    "title": "Hello",
27    "description": "World",
28    "task_status": "A",
29    "user": 4
30  }
```

Figure 12 Getting List of Tasks by user

User can create tasks. The created tasks are assigned to particular user.

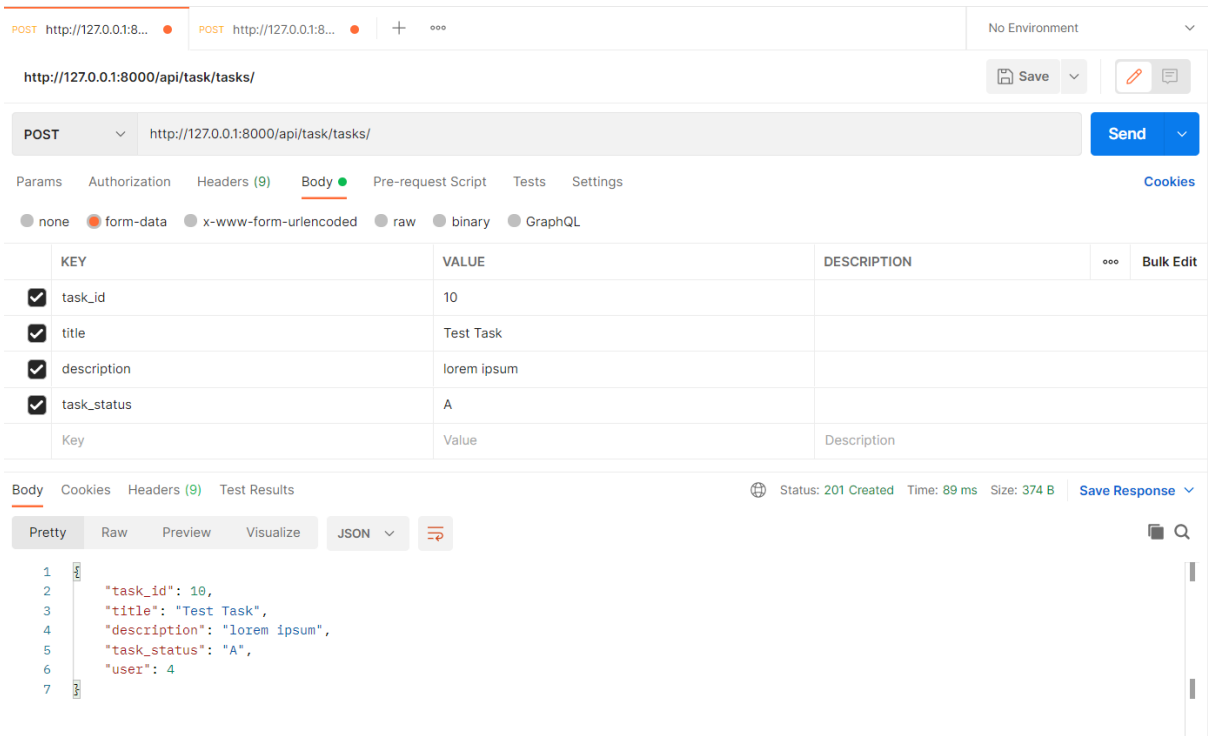


Figure 13 User Creating task

The user or admin can also update a task.

PUT http://127.0.0.1:8000/

POST http://127.0.0.1:8000/

+

...

No Environment

http://127.0.0.1:8000/api/task/tasks/10/

Save

PUT

http://127.0.0.1:8000/api/task/tasks/10/

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	task_id	10			
<input checked="" type="checkbox"/>	title	Changed Task			
<input checked="" type="checkbox"/>	description	lorem ipsum			
<input checked="" type="checkbox"/>	task_status	A			
	Key	Value	Description		

Body

Cookies

Headers (9)

Test Results

Status: 200 OK Time: 85 ms Size: 386 B Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

task_id": 10,

title": "Changed Task",

description": "lorem ipsum",

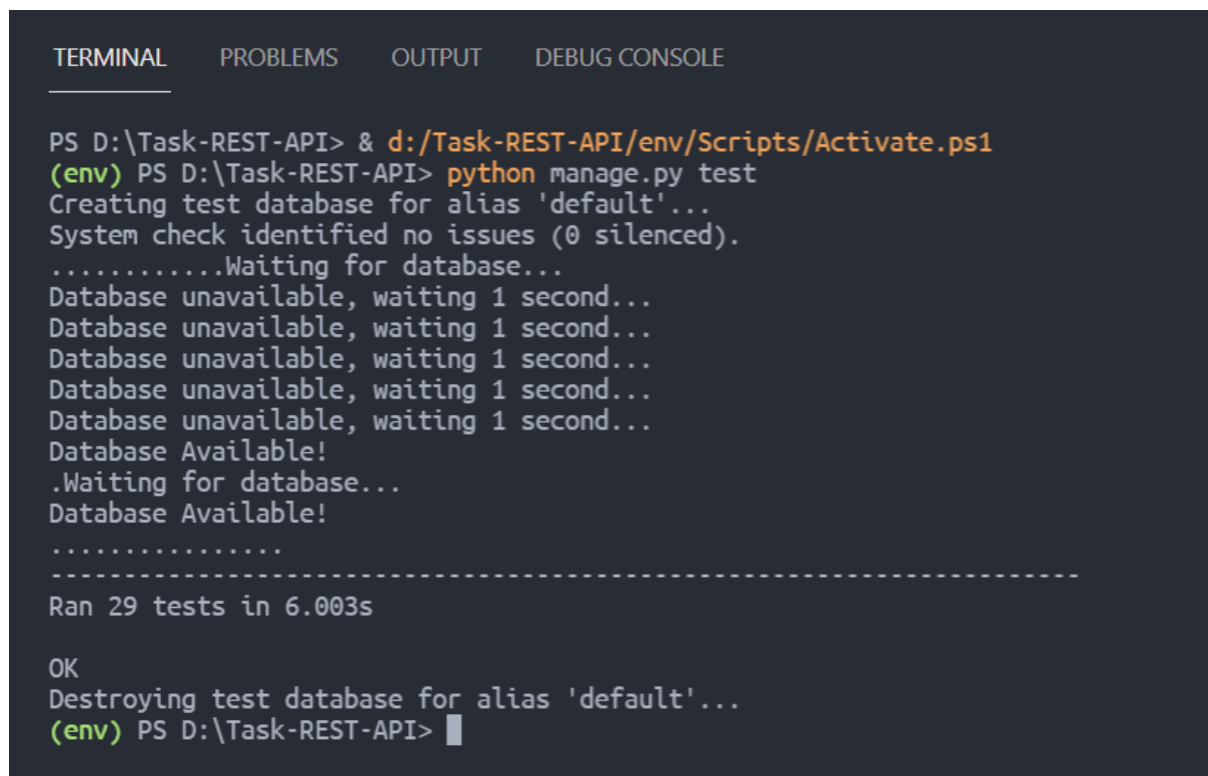
task_status": "A",

user": 4

Figure 14 Update Task

10. Unit testing for each Individual Endpoints

Unit For each individual endpoint and test case was done. Ensuring minimization of errors in the API. The unit test for endpoints of each app are in their respective app folders in the project.



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

PS D:\Task-REST-API> & d:/Task-REST-API/env/Scripts/Activate.ps1
(env) PS D:\Task-REST-API> python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....Waiting for database...
Database unavailable, waiting 1 second...
Database unavailable, waiting 1 second...
Database unavailable, waiting 1 second...
Database unavailable, waiting 1 second...
Database unavailable, waiting 1 second...
Database Available!
.Waiting for database...
Database Available!
.....
-----
Ran 29 tests in 6.003s

OK
Destroying test database for alias 'default'...
(env) PS D:\Task-REST-API> █
```

Figure 15 Unit Test for Each Individual Endpoints