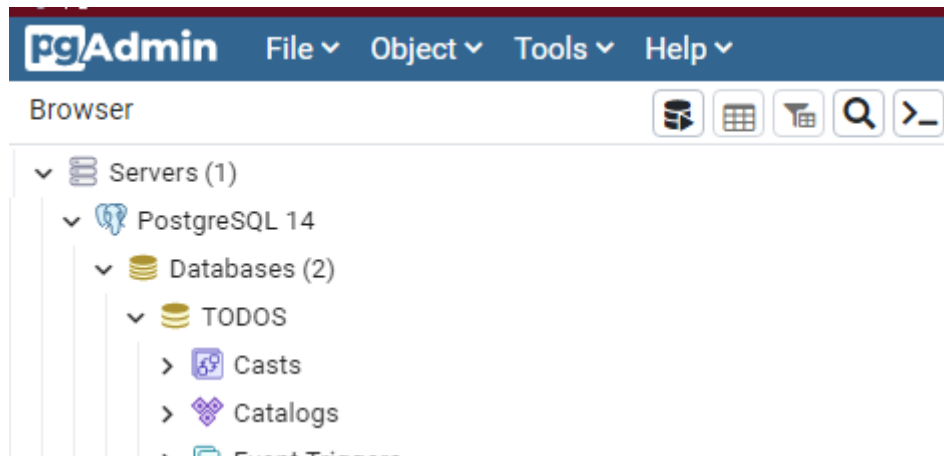


# Python assignment 1

-Faheem Ahmed

Install PostgreSQL, login to pgAdmin4 and set up a new database for the app.

Here it is named TODOS, for example.



Add your postgresql credentials in .env file

```
export DB_URI=postgresql://[Owner]:[Password]@[host]/[name of the database]
```

here it is configured as such:

```
export DB_URI=postgresql://postgres:onefour@localhost/TODOS
```

You need Python installed in your test system.

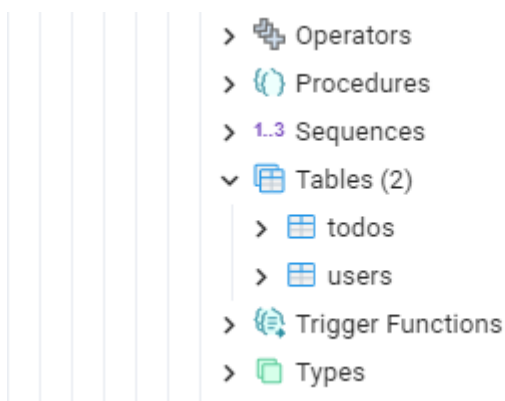
install required dependencies by running

```
$ pip install -r requirements.txt
```

Run the app

```
$ flask run
```

It will create two tables “users” and “todos” in the specified database. And start the server.



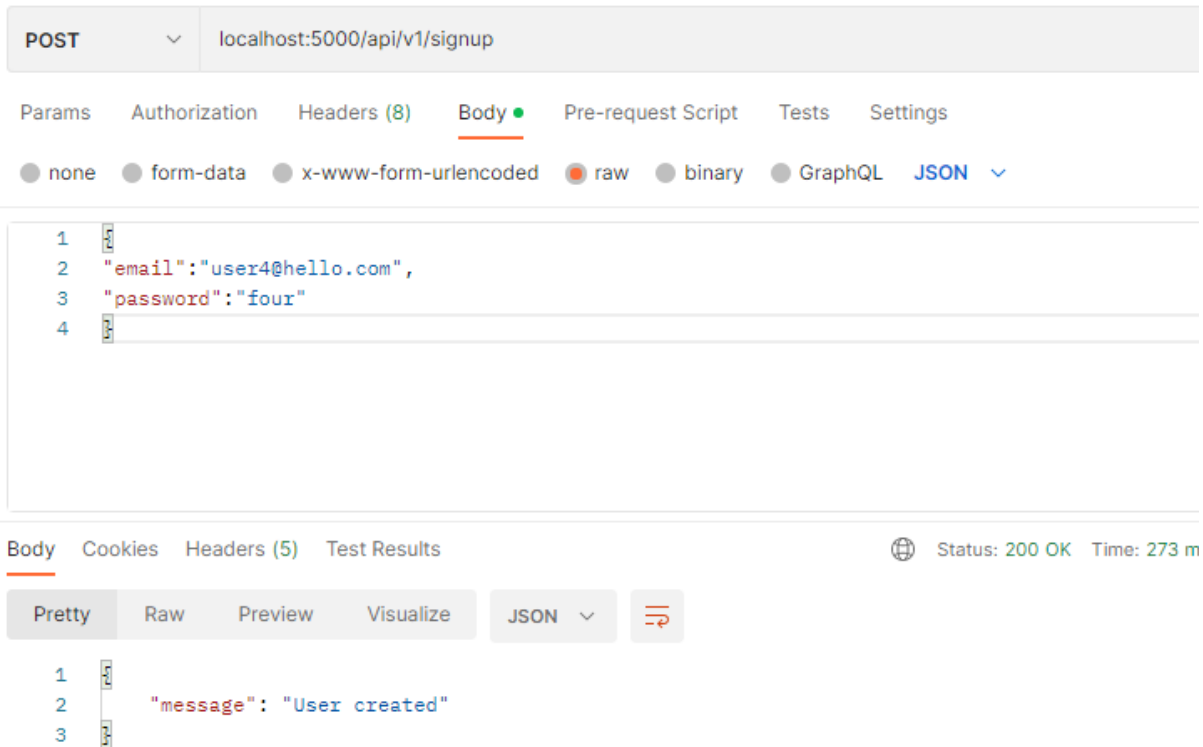
Using Postman send POST request to create user:

localhost:5000/api/v1/signup

```
{
  "email": "user1@hello.com",
  "password": "one"
}
```

Response you receive is:

```
{
  "message": "User created"
}
```



Database stores password hash, and time of any update.

The screenshot shows the pgAdmin 4 interface with the 'users' table selected. The table structure and data are as follows:

id	email	password	created_at	updated_at
1	user1@hello.com	pbkdf2:sha256:260000\$HLPjR7ODci6KAAdIOS2cfe6ed15c4d7cb0bc8fadda...	2022-09-11 23:17:34.307309	2022-09-11 23:17:34.307309
2	user2@hello.com	pbkdf2:sha256:260000\$42xUv3fvL8pKUIcmS996de2ebdef37a42cd5976d35...	2022-09-11 23:17:34.307309	2022-09-11 23:17:34.307309
3	user3@hello.com	pbkdf2:sha256:260000\$90Ps9Ql2rdGPGxg7\$05a4eb174aa48b0be2be653ae...	2022-09-11 23:17:34.307309	2022-09-11 23:17:34.307309
4	user4@hello.com	pbkdf2:sha256:260000\$Jz39SR6Tlg5dATN3\$38b124831f34b6ddf73924045f...	2022-09-12 18:17:23.885736	2022-09-12 18:17:23.885736

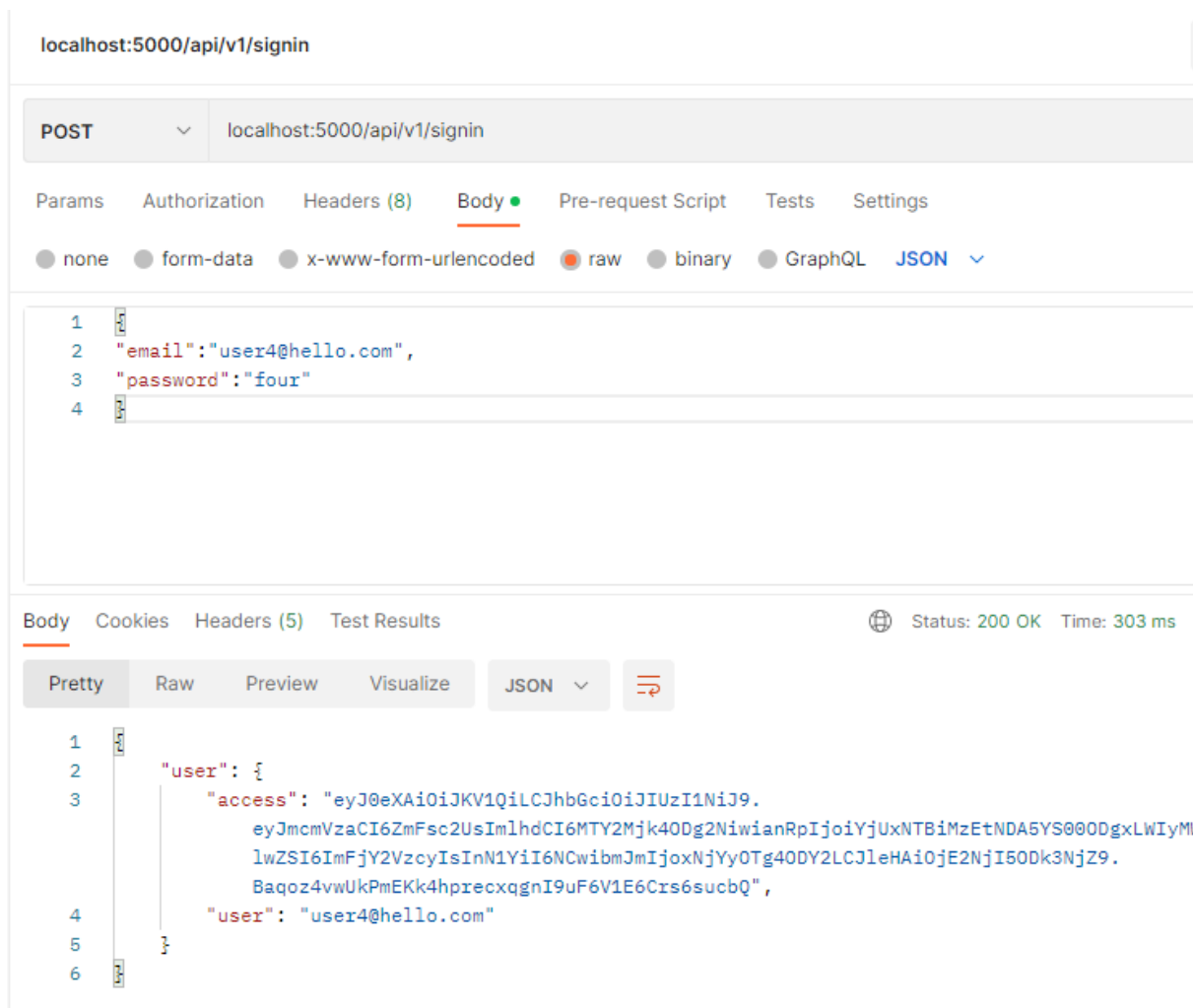
Send signin request POST:

localhost:5000/api/v1/signin

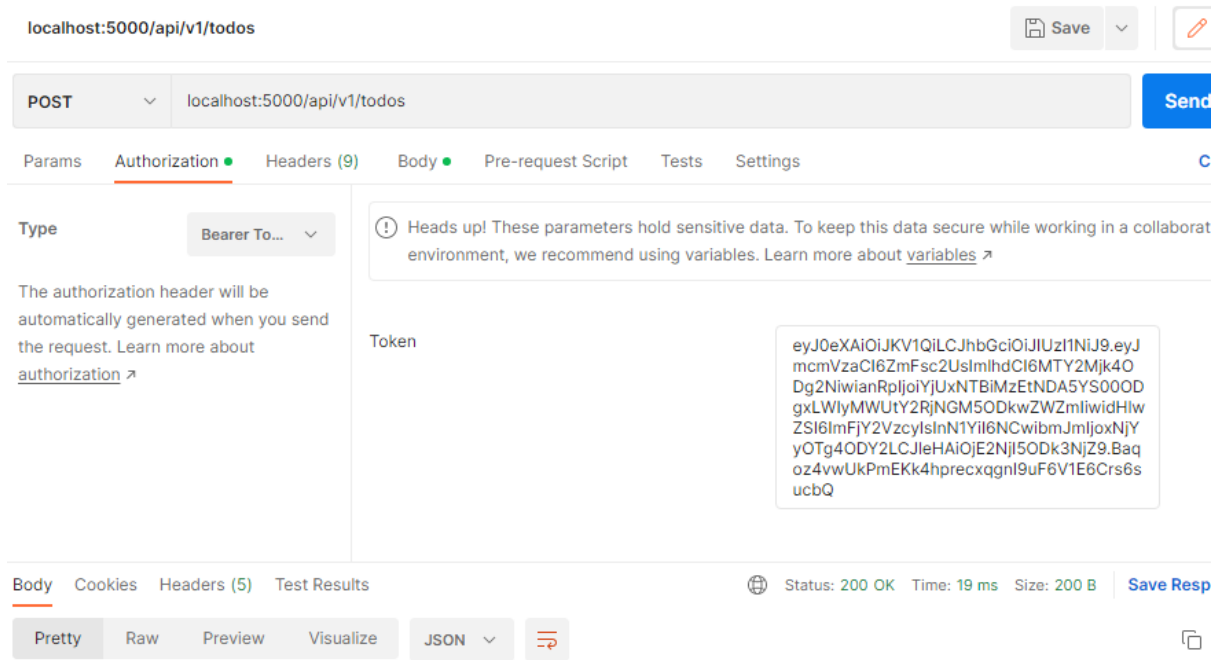
```
{
  "email": "user1@hello.com",
  "password": "one"
}
```

response received JWT access token for the user:

```
{
  "user": {
    "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImhhdCI6MTY2MjIxNTk4NSwianRpIjoiazQzMjhmMTktNmU3Ni00ODNiLTk0NmYtZmEwNzc5OTNmYTlhIiwidHlwZSI6ImFjY2VzcyIsInN1Yil6MSwibmJmljoxNjYyOTEOTg1LCJleHAiOiJlE2NjI5MTY4ODV9.QbqiNlkrT3hyQdGQSNkXxkDaL7SwhDOC1Sclmyj_MME",
    "user": "user1@hello.com"
  },
  "user_id": "1"
}
```



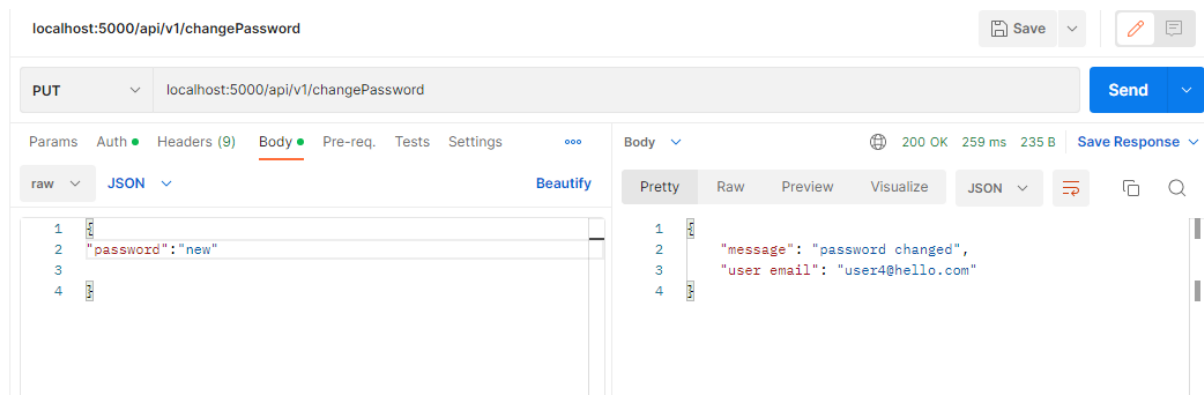
Add access token to the Auth Bearer Token field in the Postman app for all next functions



Send PUT request to change password

localhost:5000/api/v1/changePassword

```
{
  "password":"new"
}
```



Send POST request to create new todo item for current user

```
localhost:5000/api/v1/todos
{
  "name": "todo item1 user1",
  "description": "is optional"
}
```

POST localhost:5000/api/v1/todos

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

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

```
1 {
2   "name": "todo item user4",
3   "description": "new todo item for user number 4 with default status NotStarted"
4 }
5
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 13 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "todo item added"
3 }
```

Database also stores id of user who created the todo item, and time of any update.

pgAdmin 4

public.todos/TODOS/postgres@PostgreSQL 14

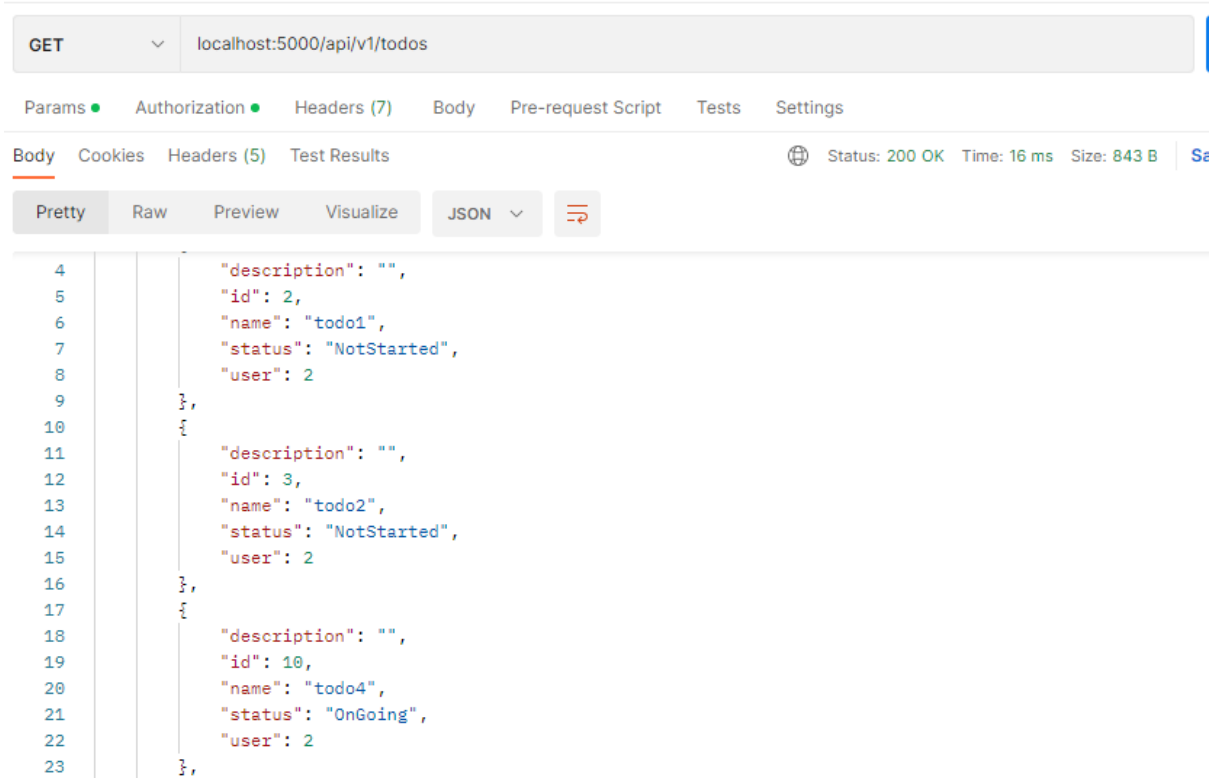
No limit

Data output Messages Notifications

	id [PK] integer	name character varying	description character varying	user_id integer	created_at timestamp without time zone	updated_at timestamp without time zone	status status
1	1	todo item user 1		1	2022-09-11 23:25:26.502044	2022-09-12 17:33:19.766712	OnGoing
2	2	todo1		2	2022-09-11 23:26:02.707573	2022-09-11 23:26:02.707573	NotStarted
3	3	todo2		2	2022-09-11 23:26:02.707573	2022-09-11 23:26:02.707573	NotStarted
4	4	todo2		3	2022-09-11 23:26:02.707573	2022-09-11 23:26:02.707573	NotStarted
5	5	todo1		3	2022-09-11 23:26:02.707573	2022-09-11 23:26:02.707573	NotStarted
6	6	todo3		3	2022-09-11 23:26:02.707573	2022-09-11 23:26:02.707573	NotStarted
7	7	todo 7	todo item for user 1 status upd...	1	2022-09-11 23:26:02.707573	2022-09-12 17:35:13.464824	Completed
8	8	todo4		1	2022-09-11 23:26:02.707573	2022-09-11 23:26:02.707573	NotStarted
9	9			1	2022-09-12 00:32:39.659721	2022-09-12 00:32:39.659721	NotStarted
10	10	todo4		2	2022-09-12 01:31:45.012051	2022-09-12 01:31:45.012051	OnGoing
11	11	todo5	new todo item for user 2	2	2022-09-12 14:35:46.484927	2022-09-12 14:35:46.484927	NotStarted
12	12	todo6	new todo item for user 2	2	2022-09-12 14:35:46.484927	2022-09-12 14:35:46.484927	Completed

Send GET request to receive list of all todo items

localhost:5000/api/v1/todos



GET localhost:5000/api/v1/todos

Params Authorization Headers (7) Body Pre-request Script Tests Settings

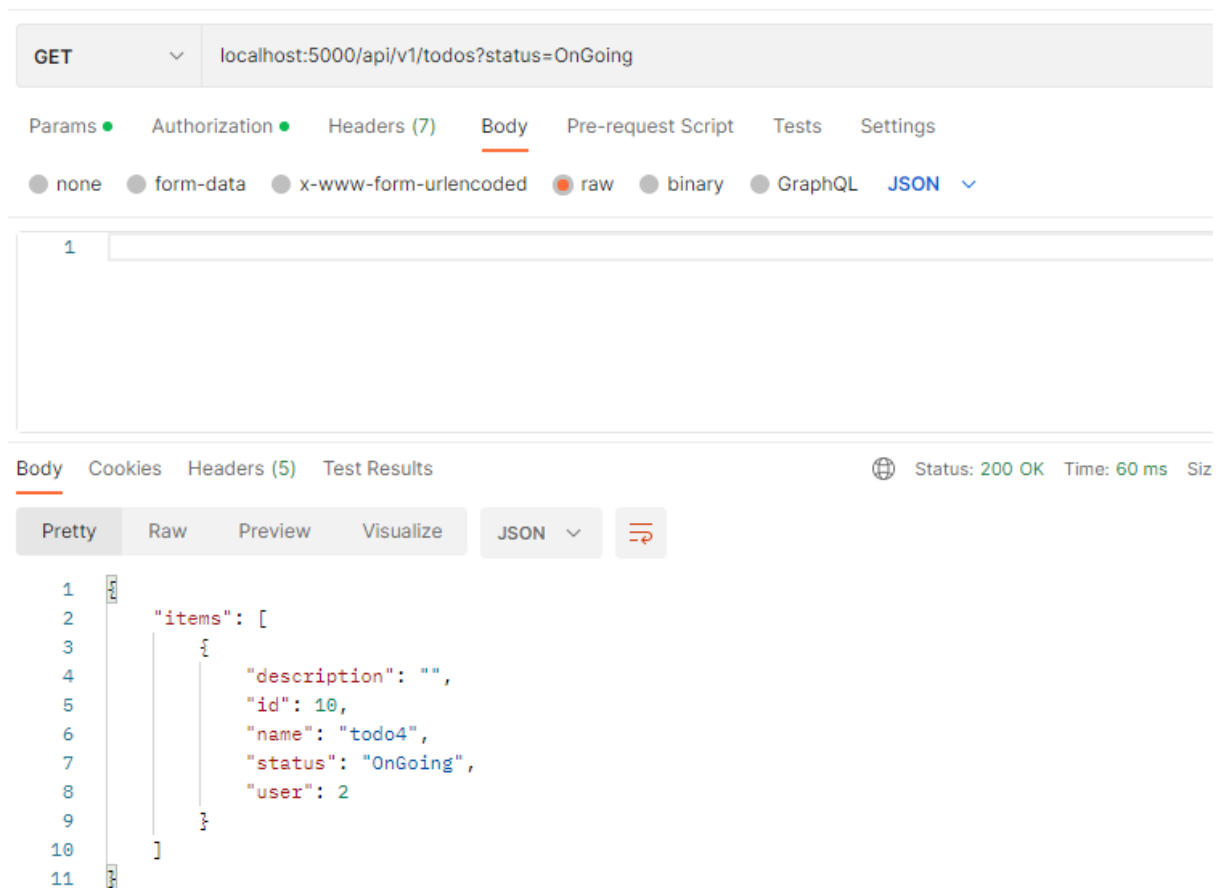
Body Cookies Headers (5) Test Results Status: 200 OK Time: 16 ms Size: 843 B

Pretty Raw Preview Visualize JSON

```
4      "description": "",
5      "id": 2,
6      "name": "todo1",
7      "status": "NotStarted",
8      "user": 2
9    },
10   {
11     "description": "",
12     "id": 3,
13     "name": "todo2",
14     "status": "NotStarted",
15     "user": 2
16   },
17   {
18     "description": "",
19     "id": 10,
20     "name": "todo4",
21     "status": "OnGoing",
22     "user": 2
23   }
24 ]
```

Or add status query parameter in the url to receive list of items with specific status

(example) localhost:5000/api/v1/todos?status=NotStarted



GET localhost:5000/api/v1/todos?status=OnGoing

Params Authorization Headers (7) Body Pre-request Script Tests Settings

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

1

Body Cookies Headers (5) Test Results Status: 200 OK Time: 60 ms Siz

Pretty Raw Preview Visualize JSON

```
1  [
2    {
3      "description": "",
4      "id": 10,
5      "name": "todo4",
6      "status": "OnGoing",
7      "user": 2
8    }
9  ]
```

Send PUT request to update existing todo item

by adding id of the todo item in the url, and access token of the user in the Auth Bearer token field.

(example) localhost:5000/api/v1/todos:1

```
{
  "name": "updated name of todo item1",
  "description": "updating status to OnGoing",
  "status": "OnGoing"
}
```

if item id is matched with the id of the bearer token sent, the item will be updated

The screenshot displays a REST client interface with a PUT request to `localhost:5000/api/v1/todos:15`. The request body is a JSON object: `{ "name": "todo 15", "description": "todo item 15 completed", "status": "Completed" }`. The response status is `200 OK` with a time of `26 ms` and size of `283`. The response body is a JSON object: `{ "description": "todo item 15 completed", "id": 15, "name": "todo 15", "status": "Completed", "user": 4 }`.

**PUT** `localhost:5000/api/v1/todos:15`

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

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

```
1 {
2   "name": "todo 15",
3   "description": "todo item 15 completed",
4   "status": "Completed"
5 }
6
```

**Body** Cookies Headers (5) Test Results Status: 200 OK Time: 26 ms Size: 283

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "description": "todo item 15 completed",
3   "id": 15,
4   "name": "todo 15",
5   "status": "Completed",
6   "user": 4
7 }
```

Send DELETE request to delete existing todo item

by adding id of the todo item in the url, and access token of the user in the Auth Bearer token field.

if item id is matched with the id of the bearer token sent, the item will be deleted

(example) localhost:5000/api/v1/todos:1

localhost:5000/api/v1/todos:15

DELETE

localhost:5000/api/v1/todos:15

Params

Authorization ●

Headers (7)

Body

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

This request does not have a body

Body

Cookies

Headers (5)

Test Results



Status: 200 OK

Time: 14 ms

Pretty

Raw

Preview

Visualize

JSON



1



2



3



"message": "requested item deleted"