# Lab 1

## Introduction

In this lab, you will explore the fundamentals of RESTful APIs, including how to develop your own API, effectively call external APIs, and utilize OAuth for secure authentication. Additionally, the lab will cover the practical use of webhooks, focusing particularly on their application with the GitHub API.

## FastAPI Introduction

### what is FastAPI?

FastAPI is a Python framework used for creating web applications and APIs (Application Programming Interfaces). It's called "FastAPI" because it allows you to build APIs quickly and efficiently.

### Why Use FastAPI?

- **Speed:** FastAPI is built on Starlette for the web part and Pydantic for the data part, making it very fast.
- **Easy to Use:** It has an intuitive design, which makes it easy to learn, especially for those already familiar with Python.
- **Automatic Documentation:** One of the coolest features of FastAPI is that it automatically generates documentation for your API. This means you can see and test your API endpoints easily.
- **Data Validation and Serialization:** FastAPI simplifies data validation and serialization. It uses modern Python type hints, making your code more robust and clear.

### How Does FastAPI Work?

- You define path operations for your API. These are the different requests you can make, like GET, POST, PUT, and DELETE.
  Path parameters and query parameters are used to define variable parts of your routes and queries.
- For data coming into your API, you define Pydantic models, which help with data validation and serialization.

# REST API Overview

REST, short for Representational State Transfer, is an architectural style for creating web services that operate via the HTTP protocol. Its principles were established in 2000 by computer scientist Roy Fielding, and it has since become a preferred method due to its scalability and flexibility, outpacing older techniques of machine-to-machine communication. To this day, it remains the de facto standard for public APIs.

# REST API Concepts

At the heart of the REST API design are the following elements:

- **Requests and Responses**: To access a resource, the client sends an HTTP request. The server then responds with an HTTP response, containing the encoded data about the resource. Both the request and response are self-descriptive, carrying the necessary details to parse and handle them.

### Key Components of a REST Request:

Every REST request is composed of four critical elements:

1. **HTTP Method**: This specifies the action to be performed on the resource. The four basic methods, also known as CRUD operations, are:
   - `GET`: Retrieve a resource.
   - `POST`: Create a new resource.
   - `PUT`: Update an existing resource.
   - `DELETE`: Delete a resource.
2. **Endpoint**: The endpoint includes a Uniform Resource Identifier (URI), which specifies where the resource can be found on the Internet. The most common URI form is a Uniform Resource Locator (URL), which is the full address used to access the web resource.
3. **Headers**: These contain metadata for the HTTP request and response. Headers are used for various purposes, such as providing authentication (e.g., API keys), specifying the host (server name or IP address), and indicating the desired response format.
4. **Body**: The body of the request carries additional information to the server. This could be data that you wish to create or update.

### Step 0: Setting Up Your Environment

1. **Install Python**: Make sure Python 3.6+ is installed on your system. You can download it from the official Python website.
2. **Install FastAPI**: Run `pip install fastapi` in your terminal.
3. **Install Uvicorn**: Uvicorn is an ASGI server. Install it using `pip install uvicorn`.

## Step 1 : Project Setup

1. **Create a New Project Folder**:

   Create a new folder named `Repo_management_app` for your project.

   ```
   mkdir Repo_management_app
   cd Repo_management_app
   ```

2. **Create the Main Application File**:

   Inside the `Repo_management_app` folder, create a file named `main.py`. This file will serve as the entry point for our FastAPI application.

   ```
   from fastapi import FastAPI
   import uvicorn

   app = FastAPI()

   @app.get("/")
   def read_root():
       return {"Message": "Repo Management Tool API"}

   if __name__ == "__main__":
   uvicorn.run(app, host="0.0.0.0", port=8080)
   ```

3. **Navigate to Your Project Directory**:

   Open your terminal and navigate to the `Repo_management_app` directory.

4. **Start the FastAPI Application**:

   Run the application using Uvicorn with the following command:

```
uvicorn main:app --reload
```

The `--reload` flag is particularly useful as it enables auto-reload. This means that the server will automatically restart whenever you make changes to the code. **You won't need to manually stop and restart the server each time you update your code**, which greatly facilitates the development process.

5. **Verify the Application**:

   Open your browser and go to [http://127.0.0.1:8000/](http://127.0.0.1:8000/). You should see the JSON response `{"Message": "Repo Management Tool API"}`.

## Step 2: Repo Management App

We are going to create a FastAPI application, named "Repo Management App", which is designed to simulate the management of software repositories and user accounts in a straightforward and efficient manner. The application features a series of endpoints to handle user registrations, repository additions, data retrievals, and updates.

### 2.1. Update `main.py`:

```
from fastapi import FastAPI, HTTPException, Query, Path
from typing import Optional
import uvicorn

app = FastAPI()

# In-memory storage for demonstration purposes (we coud have used
Database for storing data but for simplicty we will use in-memory)

repo_data = {}
user_data = {}

if __name__ == "__main__":
  uvicorn.run(app, host="0.0.0.0", port=8000)
```

🔴 **Very Important Note:**

When working with the `main.py` file in our FastAPI application, it's crucial to be aware of the following:

Even though we've made updates to the content of `main.py`, there's no need to manually re-run the command:

```
uvicorn main:app --reload
```

This is because we're using the `--reload` option with `uvicorn`, which enables automatic reloading of the server whenever changes are detected in the code.

However, it's important to remember that our current setup uses in-memory storage for data. This means that **every time the server automatically restarts due to code changes, all stored data will be lost**. Therefore, any modifications to `main.py` that trigger a server restart will result in a reset of our in-memory data.

### 2.2. Models Creation:

Pydantic is a data validation and settings management library that is integral to FastAPI. In FastAPI, Pydantic models are used to define the structure, types, and validation rules for data that your API receives and sends.

When you define your API endpoints in FastAPI, you typically use Pydantic models to specify the expected request body and the format of the response data. FastAPI uses these models to:

- Validate incoming JSON requests to ensure they match the schema defined by the Pydantic model.
- Serialize model instances to JSON responses automatically.
- Generate OpenAPI schemas for your API documentation, which is one of FastAPI's standout features.

In this App we will be using the following

- `**User**` **Model**: This Pydantic model represents a user with `username` and `email` fields. It's used in the user registration process to validate and store user information.
- `**RepoData**` **Model**: Another Pydantic model that encapsulates repository information, including `name` and an optional `description`. This model is used for updating and storing repository data.

1. **Add the following Models to your** `**main.py**`:

```
from pydantic import BaseModel

class User(BaseModel):
    username: str
    email: str

class RepoData(BaseModel):
    name: str
    description: Optional[str] = None
```

### 2.3 Creating Endpoints

These endpoints cover the basic operations of user registration, repository management (addition, retrieval, update, and deletion), and provide a simple root message. They use HTTP status codes to indicate the success or failure of the requests, following standard RESTful practices. Add the following endpoints to the a :

1. **Root Endpoint**
    - **HTTP Method:** GET
    - **Endpoint:** `/`
    - **Description:** This endpoint provides a root message. It doesn't require any parameters or a request body.

    - **Code Snippet:**

      ```
      @app.get("/")
      def read_root():
          return {"Message": "GitHub Analytics Tool API"}
      ```

    - **Status Codes:**
        - 200 OK: Successfully retrieved the root message.
2. **Register User**
    - **HTTP Method:** POST
    - **Endpoint:** `/register`
    - **Path Parameters:** None
    - **Query Parameters:** None
    - **Expected Body:** JSON with `username` and `email`.

   - **Description:** Registers a new user.
   - **Status Codes:**
      - 201 Created: User registered successfully.
      - 400 Bad Request: Username already exists.

      - **Code Snippet:**

```
@app.post("/register", status_code=201)
def register_user(user: User):
  if user.username in user_data:
      raise HTTPException(status_code=400,
detail="Username already exists")
  user_data[user.username] = {"email":
user.email, "repos": []}
  return {"Message": "User registered
successfully"}
```

3. **Add Repository**
   - **HTTP Method:** POST
   - **Endpoint:** `/user/{username}/add_repo`
   - **Path Parameters:**
      - `username`: The username to which the repository is to be added.
   - **Query Parameters:** None
   - **Expected Body:** JSON with repository `name` and optional `description`.
   - **Description:** Adds a new repository to a specified user.
   - **Status Codes:**
      - 200 OK: Repository added successfully.
      - 404 Not Found: User not found.
      - 400 Bad Request: Repository name already exists.

   - **Code Snippet:**

```
@app.post("/user/{username}/add_repo")
def add_repository(username: str, repo: RepoData):
    if username not in user_data:
        raise HTTPException(status_code=404,
detail="User not found")
```

```
    if repo.name in repo_data:
        raise HTTPException(status_code=400,
detail="Repository name already exists")
    repo_data[repo.name] = repo.dict()
    user_data[username]["repos"].append(repo.name)
    return {"Message": "Repository added
successfully"}
```

4. **Get Repository**
    - **HTTP Method:** GET
    - **Endpoint:** `/repo/{repo_name}`
    - **Path Parameters:**
        - `repo_name`: The name of the repository to retrieve.
    - **Query Parameters:** None
    - **Expected Body:** None
    - **Description:** Retrieves information about a specific repository.
    - **Status Codes:**
        - 200 OK: Successfully retrieved the repository data.
        - 404 Not Found: Repository not found.

    - **Code Snippet:**

```
@app.get("/repo/{repo_name}")
def get_repository(repo_name: str = Path(...)):
    if repo_name not in repo_data:
        raise HTTPException(status_code=404,
detail="Repository not found")
    return repo_data[repo_name]
```

5. **Update Repository**
    - **HTTP Method:** PUT
    - **Endpoint:** `/update_repo/{repo_name}`
    - **Path Parameters:**
        - `repo_name`: The name of the repository to update.
    - **Query Parameters:** None
    - **Expected Body:** JSON with updated repository `name` and optional `description`.
    - **Description:** Updates the data of an existing repository.

- **Status Codes:**
    - 200 OK: Repository data updated successfully.
    - 404 Not Found: Repository not found.

- **Code Snippet:**

```
@app.put("/update_repo/{repo_name}")
def update_repository(repo_name: str, data:
RepoData):
    if repo_name not in repo_data:
        raise HTTPException(status_code=404,
detail="Repository not found")
    repo_data[repo_name] = data.dict()
    return {"Message": "Repository data updated
successfully"}
```

6. **Delete Repository**
    - **HTTP Method:** DELETE
    - **Endpoint:** `/user/{username}/delete_repo`
    - **Path Parameters:**
        - `username`: The username from whose account the repository is to be deleted.
    - **Query Parameters:**
        - `repo_name`: The name of the repository to delete.
    - **Expected Body:** None
    - **Description:** Deletes a repository from a specified user's account.
    - **Status Codes:**
        - 200 OK: Repository deleted successfully.
        - 404 Not Found: User not found or repository not found in user account.

    - **Code Snippet:**

```
@app.delete("/user/{username}/delete_repo")
def delete_repository(username: str, repo_name: str =
Query(...)):
    if username not in user_data:
        raise HTTPException(status_code=404,
detail="User not found")
    if repo_name not in user_data[username]["repos"]:
        raise HTTPException(status_code=404,
detail="Repository not found in user account")
    user_data[username]["repos"].remove(repo_name)
    return {"Message": "Repository deleted
successfully"}
```

Hints:

- Path parameters are part of the URL path of the request. They are used to identify a specific resource or resources.
- Path parameters are included in the route definition. For example, in `/user/{username}`, `username` is a path parameter.
- Query parameters, on the other hand, are appended to the URL with a '?' and can be used to filter or sort the data, or specify certain options.For example, in `/items?sort=asc&page=2`, `sort` and `page` are query parameters.

### 2.4 Testing the Endpoints

- Use a tool like Postman https://www.postman.com/downloads/ to test these endpoints and make sure each endpoint handles requests and responses correctly.
- **( Bonus )** Add two endpoints to display list of all users and to update a user email . Explain your endpoints **( Bonus )**.

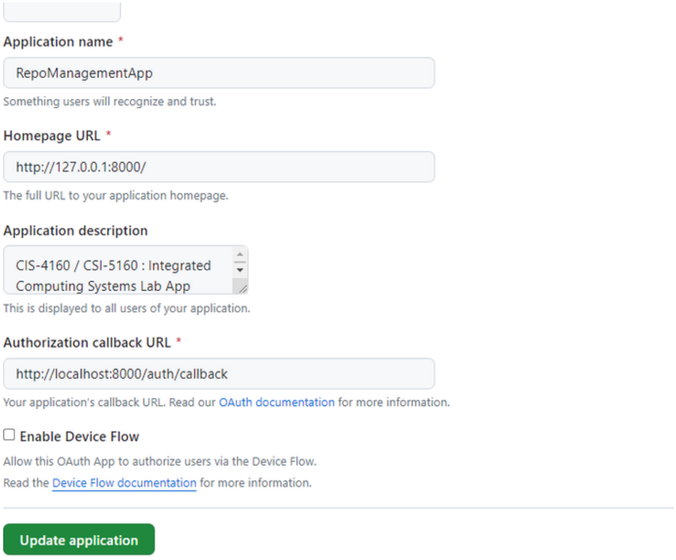
## Step 3: Github Repo Management App using Github API App

Moving beyond the limitations of managing fictional repositories internally, our application has now evolved to integrate directly with GitHub's powerful API. This enhancement allows users to directly manage their GitHub repositories through our application.

### 3.1 Github Repo Management App Registration

To utilize the GitHub API, it's essential to first register our application on GitHub. This registration process grants us a unique Client ID and Client Secret.

please go to https://github.com/settings/applications/new fill the app info (you need to have a github account to accomplish this) and make sure to fill all the values same as the screenshot below ( callback url and homepage url are very important values )



Application name *

RepoManagementApp

Something users will recognize and trust.

Homepage URL *

http://127.0.0.1:8000/

The full URL to your application homepage.

Application description

CIS-4160 / CSI-5160 : Integrated Computing Systems Lab App

This is displayed to all users of your application.

Authorization callback URL *

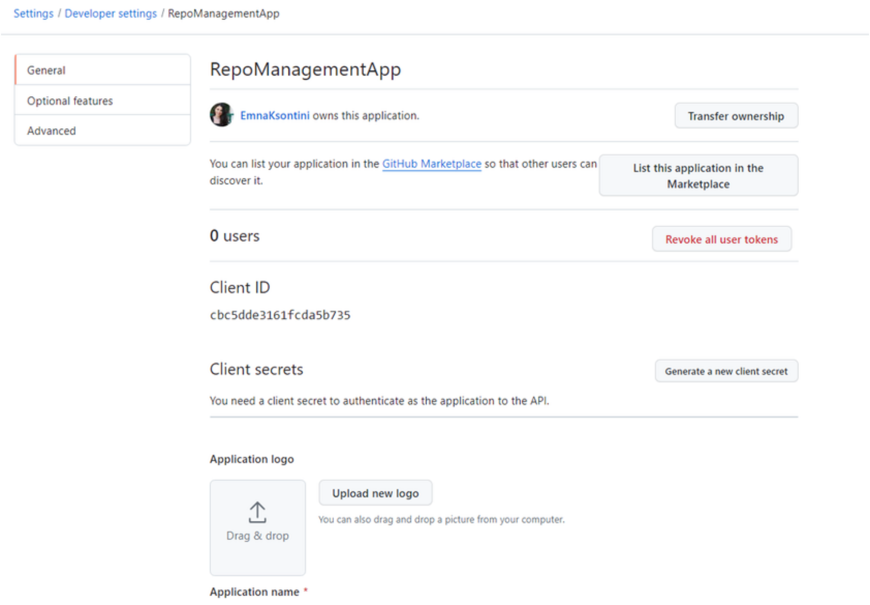
http://localhost:8000/auth/callback

Your application's callback URL. Read our OAuth documentation for more information.

□ Enable Device Flow

Allow this OAuth App to authorize users via the Device Flow.
Read the Device Flow documentation for more information.

Generate a client secret and copy them somwhere along with the client ID and save them somewhere (we will use them later on the code).



Settings / Developer settings / RepoManagementApp

RepoManagementApp

EmnaKsontini owns this application.

You can list your application in the GitHub Marketplace so that other users can discover it.

0 users

Client ID

cbc5dde3161fcda5b735

Client secrets

You need a client secret to authenticate as the application to the API.

Application logo

You can also drag and drop a picture from your computer.

Application name *

Transfer ownership
List this application in the Marketplace
Revoke all user tokens
Generate a new client secret
Drag & drop
Upload new logo

### 3.2 Repo Management OAuth and endpoints implementation

1- `httpx` is an HTTP client library for Python, supporting both synchronous and asynchronous requests, we will need it for this step, please install it with pip using the command `pip install` httpx.

2- Delete the content of main.py and replace it with the following also make sure to update the CLIENT_ID and CLIENT_SECRET with the values generated in step 3.1:

```
from fastapi import FastAPI, HTTPException, Query, Request
from pydantic import BaseModel
from typing import Optional
import httpx
from fastapi.responses import RedirectResponse

app = FastAPI()

repo_data = {}  # Stores repository data indexed by repo ID
user_data = {}  # Stores user data indexed by username

class User(BaseModel):
    username: str
    email: str

class RepoData(BaseModel):
    name: str
    description: Optional[str] = None
    homepage: Optional[str] = None
    private: bool
    has_issues: Optional[bool] = False
    has_projects: Optional[bool] = False
    has_wiki: Optional[bool] = False
    is_template: Optional[bool] = True

async def github_api_request(url: str, token: str):
    headers = {"Authorization": f"Bearer {token}"}
    async with httpx.AsyncClient() as client:
        response = await client.get(url, headers=headers)
    return response.json()

CLIENT_ID = "your_client_id"
CLIENT_SECRET = "your_client_secret"
```

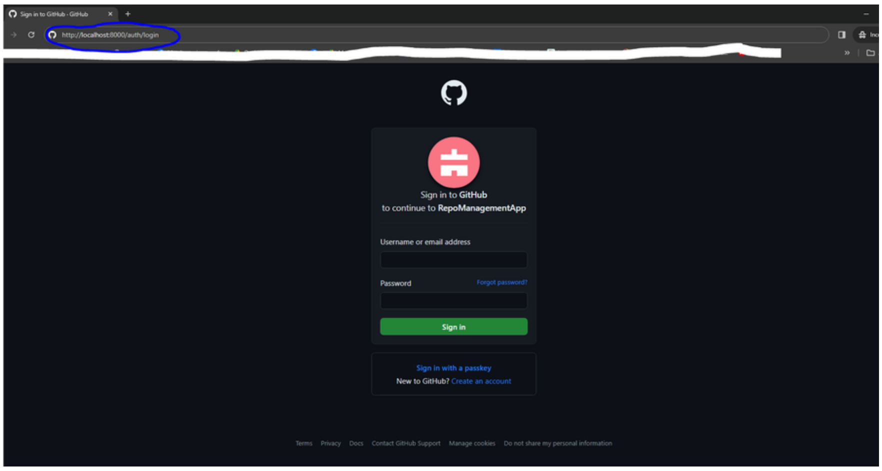
```
@app.get("/")
def read_root():
    return {"Message": "Github Repo Management App"}

@app.get("/auth/login")
def login():
    github_auth_url = (
        f"https://github.com/login/oauth/authorize"
        f"?client_id={CLIENT_ID}"
        f"&redirect_uri=http://localhost:8000/auth/callback"
        f"&scope=repo"
    )
    return RedirectResponse(url=github_auth_url)

@app.get("/auth/callback")
async def auth_callback(code: str):
    token_url = "https://github.com/login/oauth/access_token"
    async with httpx.AsyncClient() as client:
        token_response = await client.post(
            token_url,
            headers={"Accept": "application/json"},
            data={
                "client_id": CLIENT_ID,
                "client_secret": CLIENT_SECRET,
                "code": code
            }
        )
    access_token = token_response.json().get("access_token")

    user_data_response = await
github_api_request("https://api.github.com/user", access_token)
    username = user_data_response.get("login")

    if username in user_data:
        raise HTTPException(status_code=400, detail="Username already
exists")
```

```
    user_data[username] = {
        "email": user_data_response.get("email"),
        "repos": [],
        "token": access_token
    }
    return {"Message": "User authenticated and data retrieved successfully", "username": username}

@app.post("/user/{username}/add_repo")
async def add_repository(username: str, repo_data: RepoData):
    if username not in user_data or "token" not in user_data[username]:
        raise HTTPException(status_code=401, detail="User not authenticated or token missing")

    token = user_data[username]["token"]
    headers = {
        "Authorization": f"Bearer {token}",
        "Accept": "application/vnd.github.v3+json"
    }
    data = repo_data.dict()
    async with httpx.AsyncClient() as client:
        response = await client.post(
            "https://api.github.com/user/repos",
            headers=headers,
            json=data
        )

    if response.status_code != 201:
        raise HTTPException(status_code=response.status_code, detail=response.text)

    response_data = response.json()
    repo_id = response_data.get("id")
    user_data[username]["repos"].append(repo_id)
    return {"Message": "Repository added successfully", "repo_info": response_data}
```

```
@app.get("/repo/{repo_id}")
async def get_repository(repo_id: str, username: str = Query(...)):
    if username not in user_data or "token" not in
user_data[username]:
        raise HTTPException(status_code=401, detail="User not
authenticated or token missing")

    token = user_data[username]["token"]
    repo_info = await
github_api_request(f"https://api.github.com/repos/{username}/{repo_id}",
token)

    return repo_info
```

### 3.3 Explain the purpose of the two provided endpoints /user/{username}/add_repo and /repo/{repo_id}

### 3.4 Call the Login Endpoint from the Browser

- Open your preferred web browser.
- Enter the URL for the login endpoint provided by your instructor or course material. This URL will look something like `http://localhost:8000/auth/login`
- Make sure to enter your Github account credentials.
- 

[redacted]

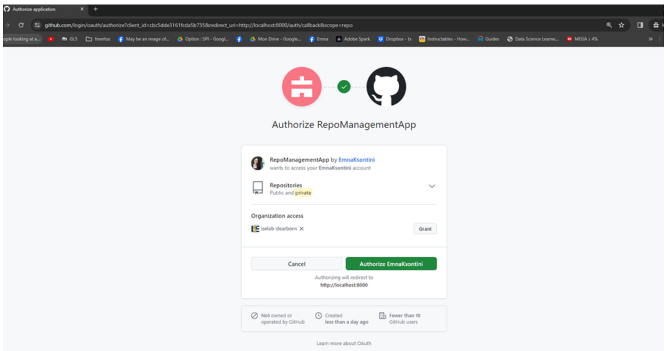
Sign in to GitHub
to continue to RepoManagementApp

Sign in to GitHub · GitHub
http://localhost:8000/auth/login
Username or email address
Password
Forgot password?
Sign in
Sign in with a passkey
New to GitHub? Create an account
Terms
Privacy
Docs
Contact GitHub Support
Manage cookies
Do not share my personal information

### 3.5 Accept the Grants Given to the App

- After navigating to the login endpoint, you will be redirected to a GitHub authorization page.
- Read the permissions requested by the application carefully.
- If you agree to grant these permissions to the application, click on the "Authorize" button to proceed.

- Remember, this step is crucial for the application to interact with your GitHub account.



github.com/login/oauth/authorize?client_id=cbc5dde3161fcda5b735&redirect_uri=http://localhost:8000/auth/callback&scope=repo

Authorize RepoManagementApp

RepoManagementApp by EmnaKsontini
wants to access your EmnaKsontini account

Repositories
Public and private

Organization access

iselab-dearborn

Authorizing will redirect to
http://localhost:8000

Not owned or operated by GitHub

Created less than a day ago

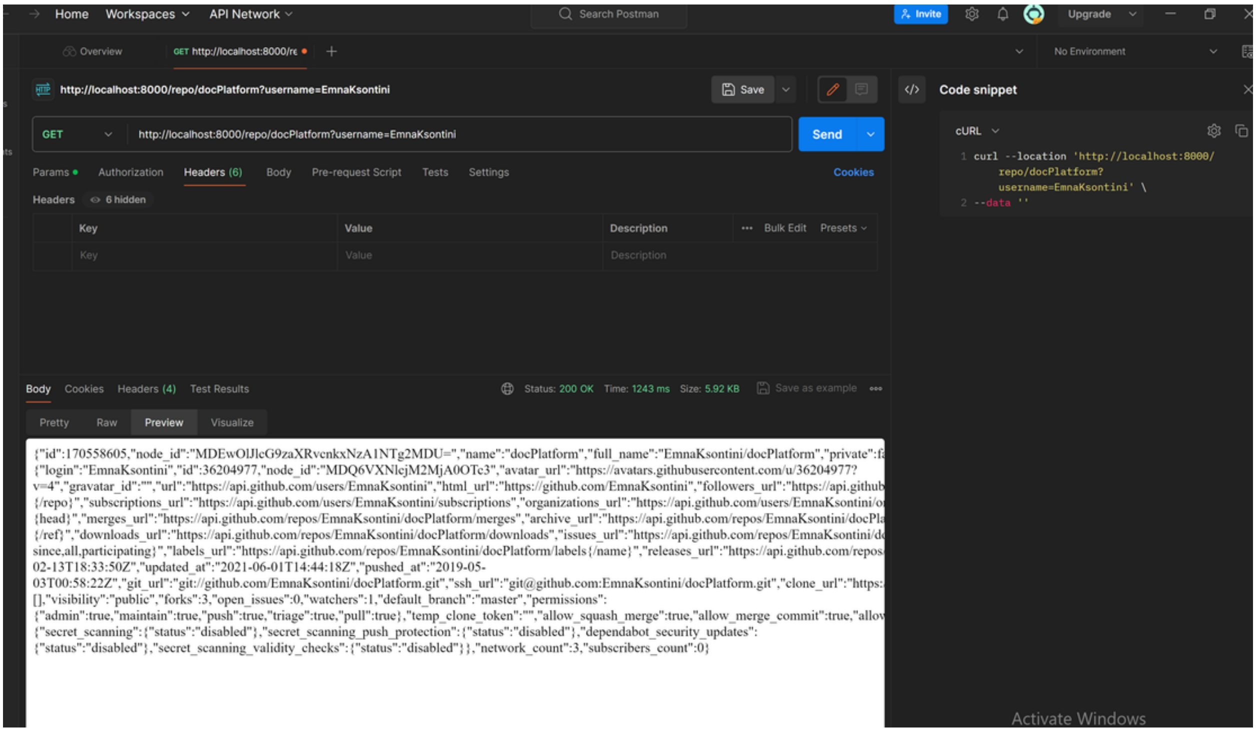
Fewer than 10 GitHub users

### 3.6 Describe the steps involved in the OAuth authentication process as implemented in the provided code (draw the flow).

### 3.7 Use Postman to Call the Get Repository Endpoint

- Ensure you have a public repository in your GitHub account. If you don't have one, create it on github.com.
- Open Postman, a popular API client that you can use to send requests to the API.
- Set up a `GET` request in Postman to the `get repo` endpoint. This endpoint URL will be something like `http://localhost:8000/repo/{repo_id}`, where `{repo_id}` is the ID of your existing public repository on GitHub.
- Don't forget to include your GitHub username as a query parameter in the URL, like `http://localhost:8000/repo/{repo_id}?username=your_username`.

- Send the request and you should receive details of your specified repository in response.



http://localhost:8000/repo/docPlatform?username=EmnaKsontini

GET http://localhost:8000/repo/docPlatform?username=EmnaKsontini

```
curl --location 'http://localhost:8000/
    repo/docPlatform?
    username=EmnaKsontini' \
--data ''
```

Status: 200 OK Time: 1243 ms Size: 5.92 KB

```
{"id":170558605,"node_id":"MDEwOlJlcG9zaXRvcnkxNzA1NTg2MDU=","name":"docPlatform","full_name":"EmnaKsontini/docPlatform","private":fa
{"login":"EmnaKsontini","id":36204977,"node_id":"MDQ6VXNlcjM2MjA0OTc3","avatar_url":"https://avatars.githubusercontent.com/u/36204977?
v=4","gravatar_id":"","url":"https://api.github.com/users/EmnaKsontini","html_url":"https://github.com/EmnaKsontini","followers_url":"https://api.github
{/repo}","subscriptions_url":"https://api.github.com/users/EmnaKsontini/subscriptions","organizations_url":"https://api.github.com/users/EmnaKsontini/o
{head}","merges_url":"https://api.github.com/repos/EmnaKsontini/docPlatform/merges","archive_url":"https://api.github.com/repos/EmnaKsontini/docPla
{/ref}","downloads_url":"https://api.github.com/repos/EmnaKsontini/docPlatform/downloads","issues_url":"https://api.github.com/repos/EmnaKsontini/d
since,all,participating}","labels_url":"https://api.github.com/repos/EmnaKsontini/docPlatform/labels{/name}","releases_url":"https://api.github.com/repos
02-13T18:33:50Z","updated_at":"2021-06-01T14:44:18Z","pushed_at":"2019-05-
03T00:58:22Z","git_url":"git://github.com/EmnaKsontini/docPlatform.git","ssh_url":"git@github.com:EmnaKsontini/docPlatform.git","clone_url":"https:
[],"visibility":"public","forks":3,"open_issues":0,"watchers":1,"default_branch":"master","permissions":
{"admin":true,"maintain":true,"push":true,"triage":true,"pull":true},"temp_clone_token":"","allow_squash_merge":true,"allow_merge_commit":true,"allow
{"secret_scanning":{"status":"disabled"},"secret_scanning_push_protection":{"status":"disabled"},"dependabot_security_updates":
{"status":"disabled"},"secret_scanning_validity_checks":{"status":"disabled"}},"network_count":3,"subscribers_count":0}
```

### 3.8 Make a POST Request to the Add Repository Endpoint

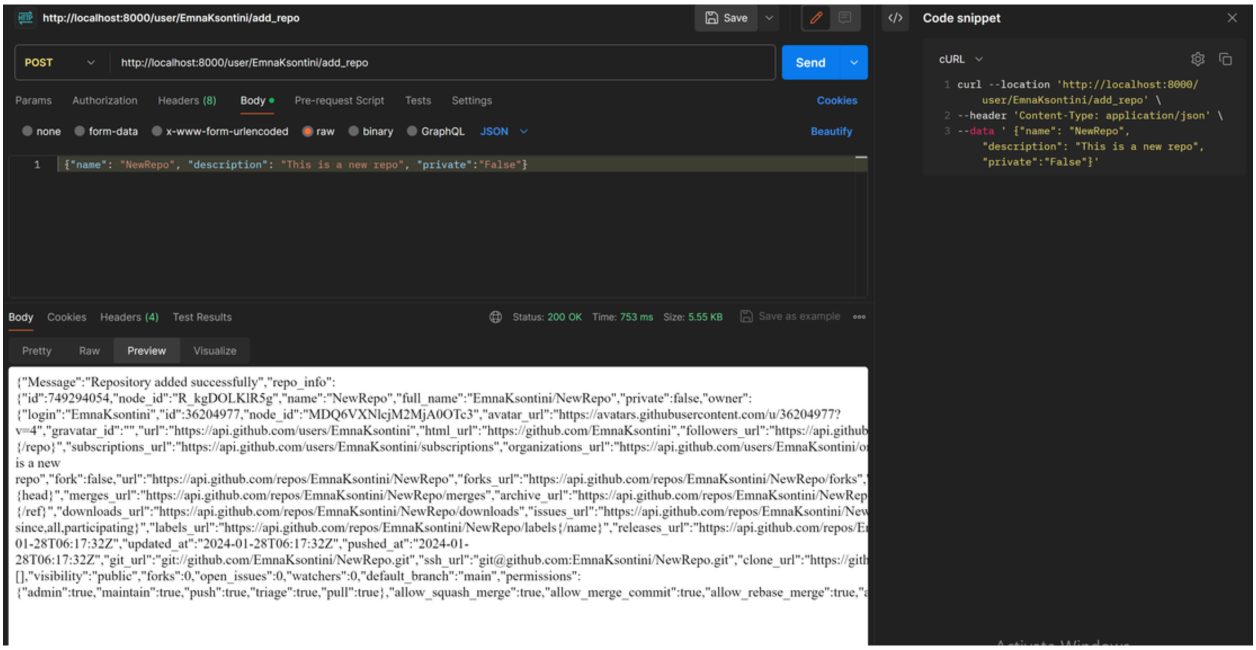
- In Postman, set up a new `POST` request to the `add repo` endpoint. This will be at a URL like `http://localhost:8000/user/{username}/add_repo`, replacing `{username}` with your GitHub username.
- You will need to send data about the new repository you wish to add. This data should be in JSON format and include details like the repository's name, description, and visibility (public or private).

- An example of the JSON data you might send is:

```
{
  "name": "NewRepo",
  "description": "This is a new repository",
  "private": false
}
```

- After setting up the request with the correct URL and JSON body, send the request.
- If successful, you will receive a confirmation response with details of the newly created repository.

- Go to your Github account to check if the repo was created.



POST http://localhost:8000/user/EmnaKsontini/add_repo

Send

```
{"name": "NewRepo", "description": "This is a new repo", "private":"False"}
```

```
curl --location 'http://localhost:8000/
    user/EmnaKsontini/add_repo' \
--header 'Content-Type: application/json' \
--data ' {"name": "NewRepo",
    "description": "This is a new repo",
    "private":"False"}'
```

Status: 200 OK Time: 753 ms Size: 5.55 KB

```
{"Message":"Repository added successfully","repo_info":
{"id":749294054,"node_id":"R_kgDOLKlR5g","name":"NewRepo","full_name":"EmnaKsontini/NewRepo","private":false,"owner":
{"login":"EmnaKsontini","id":36204977,"node_id":"MDQ6VXNlcjM2MjA0OTc3","avatar_url":"https://avatars.githubusercontent.com/u/36204977?
v=4","gravatar_id":"","url":"https://api.github.com/users/EmnaKsontini","html_url":"https://github.com/EmnaKsontini","followers_url":"https://api.github
{/repo}","subscriptions_url":"https://api.github.com/users/EmnaKsontini/subscriptions","organizations_url":"https://api.github.com/users/EmnaKsontini/o
is a new
repo","fork":false,"url":"https://api.github.com/repos/EmnaKsontini/NewRepo","forks_url":"https://api.github.com/repos/EmnaKsontini/NewRepo/forks",
{head}","merges_url":"https://api.github.com/repos/EmnaKsontini/NewRepo/merges","archive_url":"https://api.github.com/repos/EmnaKsontini/NewRep
{/ref}","downloads_url":"https://api.github.com/repos/EmnaKsontini/NewRepo/downloads","issues_url":"https://api.github.com/repos/EmnaKsontini/New
since,all,participating}","labels_url":"https://api.github.com/repos/EmnaKsontini/NewRepo/labels{/name}","releases_url":"https://api.github.com/repos/E
01-28T06:17:32Z","updated_at":"2024-01-28T06:17:32Z","pushed_at":"2024-01-
28T06:17:32Z","git_url":"git://github.com/EmnaKsontini/NewRepo.git","ssh_url":"git@github.com:EmnaKsontini/NewRepo.git","clone_url":"https://gith
[],"visibility":"public","forks":0,"open_issues":0,"watchers":0,"default_branch":"main","permissions":
{"admin":true,"maintain":true,"push":true,"triage":true,"pull":true},"allow_squash_merge":true,"allow_merge_commit":true,"allow_rebase_merge":true,"a
```

## Step 4: Github Repo Management App webhooks

### 4.1 Webhooks: A Brief Explanation

Webhooks can be thought of as a "reverse API." In a standard API setup, your application actively makes requests to an external service for data or to perform an action. In contrast, webhooks work the other way around – the external service sends data to your application automatically when a specified event occurs.

With webhooks, you set up a URL in your application to receive and handle incoming data. When the event you've subscribed to happens (like a commit or push in a GitHub repository), the external service makes an HTTP request (usually POST) to this URL, delivering real-time data related to that event. This allows your application to react to events as they occur, without the need for polling the external service for updates.

### 4.2 Application in This Lab

In this lab, we will utilize webhooks to enhance our application's interactivity with GitHub. This will involve:

1. **Setting up a Webhook in GitHub**: We will configure a webhook in a GitHub repository to notify our application of specific events like commits or push requests.
2. **Handling Incoming Data**: Our application will have an endpoint prepared to receive these webhook notifications. When GitHub triggers an event, it will send a POST request to our webhook URL in our app with details about the event.

3. **Responding to Events**: Our application will parse and use the data from these webhook notifications and display it.

### 4.3 What is Ngrok?

In a development environment, especially when working locally, your application is not usually accessible from the internet. This poses a challenge for using webhooks, as the external service (like GitHub) needs a publicly accessible URL to send its notifications.

Ngrok is a tool that creates a secure tunnel to your localhost environment. It allows you to expose a local server to the internet, making it accessible via a public URL. This is particularly useful for calling webhooks.

### 4.4 Instructions for Installing Ngrok

#### Download Ngrok

1. Go to Ngrok's official website.
2. Sign up for an account if you haven't already, as you'll need your Ngrok auth token.
3. Once logged in, go to the download section.
4. Choose the appropriate version for your operating system (Windows, macOS, or Linux).
5. Download the Ngrok executable file.

#### Installation

For Windows:

1. Unzip the downloaded file.
2. You can place the `ngrok.exe` file in any directory. Optionally, add the directory to your system's PATH environment variable for easier access from the command line.

For macOS/Linux:

1. After downloading, open your terminal.
2. Navigate to the directory containing the downloaded `ngrok` file.
3. Unzip the file using the command: `unzip /path/to/ngrok.zip`.
4. Optionally, move `ngrok` to a directory in your PATH for easy execution.

#### Connect your account

1. Open your command line interface (CLI).

2. Navigate to the directory where `ngrok` is located, if it's not in your PATH.

3. Connect your Ngrok account by running the command:

   ```
   ./ngrok authtoken [YOUR_AUTH_TOKEN]
   ```

   Replace `[YOUR_AUTH_TOKEN]` with the auth token from your Ngrok dashboard.

**Start Ngrok**

1. To start a tunnel for our app, use the command:

   ```
   ./ngrok http 8000
   ```

2. After running this command, you'll see the Ngrok interface in your CLI, which includes the public URL (both HTTP and HTTPS) that forwards to your local server ( or go to o http://localhost:4040/inspect/http to see the new url of your app.)

## 4.5 webhook implementation

These functions should be integrated into your `main.py`:

1- Function 1: `add_webhook`

This function adds a webhook to a user's GitHub repository (change NGROK_URL with yourpublic URL )

```
@app.post("/user/{username}/add_webhook")
async def add_webhook(username: str, repo_name: str = Query(...)):

    # Check if the user is authenticated
    if username not in user_data or "token" not in
user_data[username]:
        raise HTTPException(status_code=401, detail="User not
authenticated or token missing")

    # Retrieve the user's token
    token = user_data[username]["token"]
```

```
    # Define the URL where your application will receive webhook
events
    webhook_url = "{NGROK_URL}/webhook_receiver"  # Replace with your
server's webhook URL

    # Set up the configuration for the webhook
    webhook_config = {
        "url": webhook_url,
        "content_type": "json"
    }

    # Data to be sent to GitHub API for creating the webhook
    data = {
        "name": "web",
        "active": True,
        "events": ["push", "pull_request", "issue_comment"],
        "config": webhook_config
    }

    # Set headers for the request
    headers = {
        "Authorization": f"Bearer {token}",
        "Accept": "application/vnd.github.v3+json"
    }

    # Make a POST request to the GitHub API to create the webhook
    async with httpx.AsyncClient() as client:
        response = await client.post(

f"https://api.github.com/repos/{username}/{repo_name}/hooks",
            headers=headers,
            json=data
        )

    # Check the response status
    if response.status_code != 201:
        raise HTTPException(status_code=response.status_code,
detail=response.text)

    # Return a success message and webhook details
    return {"message": "Webhook added successfully", "webhook_info":
response.json()}
```

**Explanation:**

- **Webhook Configuration**: Sets up the webhook URL and the types of events (`push`, `pull_request`, `issue_comment`) it should listen to.
- **API Request**: Sends a POST request to the GitHub API to create the webhook.
- **Response Handling**: Checks the response status code. If it's 201 (Created), it returns the details of the created webhook.
- Function 2: `webhook_receiver`

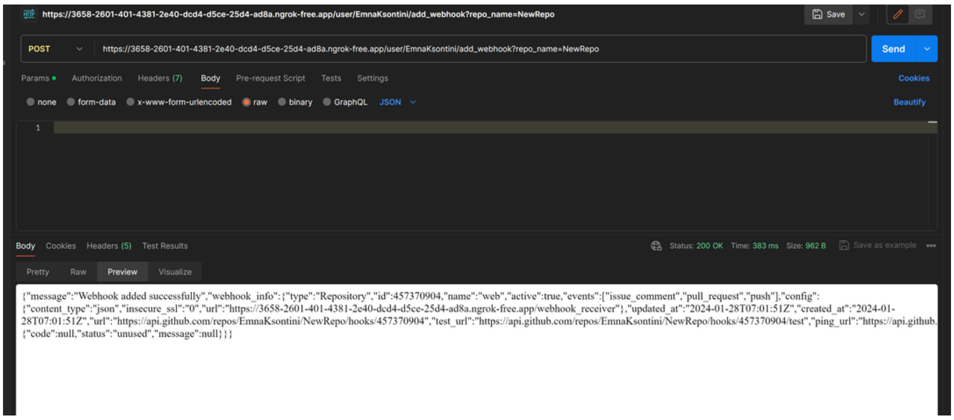
This function acts as an endpoint for GitHub to send webhook events.

```
@app.post("/webhook_receiver")
async def webhook_receiver(request: Request):
    # Extract the JSON payload from the request
    payload = await request.json()

    # Process the webhook payload as needed
    print("Webhook event received:", payload)

    # Return a response
    return {"message": "Webhook event received"}
```

**Explanation:**

- **Payload Processing**: Extracts the payload from the incoming request, which contains the information about the GitHub event.
- **Response**: Prints the payload for demonstration purposes and sends a confirmation response.

### 5.5 Using postman make an HTTP request to add a webhook to the NewRepo create in step 3.8.

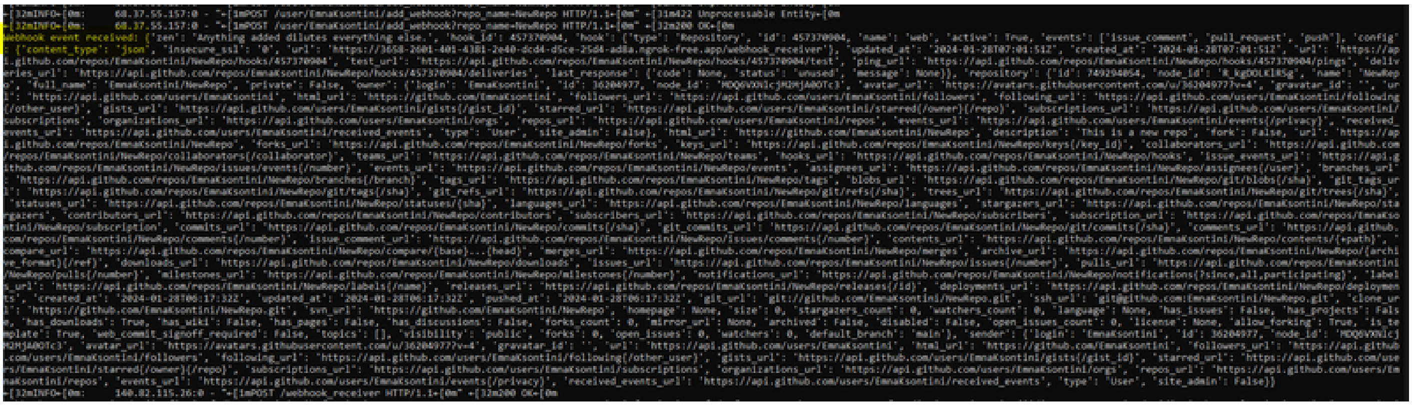


POST https://3658-2601-401-4381-2e40-dcd4-d5ce-25d4-ad8a.ngrok-free.app/user/EmnaKsontini/add_webhook?repo_name=NewRepo

Status: 200 OK Time: 383 ms Size: 962 B

```
{"message":"Webhook added successfully","webhook_info":{"type":"Repository","id":457370904,"name":"web","active":true,"events":["issue_comment","pull_request","push"],"config":
{"content_type":"json","insecure_ssl":"0","url":"https://3658-2601-401-4381-2e40-dcd4-d5ce-25d4-ad8a.ngrok-free.app/webhook_receiver"},"updated_at":"2024-01-28T07:01:51Z","created_at":"2024-01-
28T07:01:51Z","url":"https://api.github.com/repos/EmnaKsontini/NewRepo/hooks/457370904","test_url":"https://api.github.com/repos/EmnaKsontini/NewRepo/hooks/457370904/test","ping_url":"https://api.github.
{"code":null,"status":"unused","message":null}}}
```

## 5.6 Go to your Github account and create a commit by adding a any file to the NewRepo and describe what happens in the console of the app.



```
←[32mINFO←[0m:     68.37.55.157:0 - "←[1mPOST /user/EmnaKsontini/add_webhook?repo_name=NewRepo HTTP/1.1←[0m" ←[31m422 Unprocessable Entity←[0m
←[32mINFO←[0m:     68.37.55.157:0 - "←[1mPOST /user/EmnaKsontini/add_webhook?repo_name=NewRepo HTTP/1.1←[0m" ←[32m200 OK←[0m
Webhook event received: {'zen': 'Anything added dilutes everything else.', 'hook_id': 457370904, 'hook': {'type': 'Repository', 'id': 457370904, 'name': 'web', 'active': True, 'events': ['issue_comment', 'pull_request', 'push'], 'config
': {'content_type': 'json', 'insecure_ssl': '0', 'url': 'https://3658-2601-401-4381-2e40-dcd4-d5ce-25d4-ad8a.ngrok-free.app/webhook_receiver'}, 'updated_at': '2024-01-28T07:01:51Z', 'created_at': '2024-01-28T07:01:51Z', 'url': 'https://ap
i.github.com/repos/EmnaKsontini/NewRepo/hooks/457370904', 'test_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/hooks/457370904/test', 'ping_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/hooks/457370904/pings', 'deliv
eries_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/hooks/457370904/deliveries', 'last_response': {'code': None, 'status': 'unused', 'message': None}}, 'repository': {'id': 749294054, 'node_id': 'R_kgDOLKl85g', 'name': 'NewRep
o', 'full_name': 'EmnaKsontini/NewRepo', 'private': False, 'owner': {'login': 'EmnaKsontini', 'id': 36204977, 'node_id': 'MDQ6VXNlcjM2MjA0OTc3', 'avatar_url': 'https://avatars.githubusercontent.com/u/36204977?v=4', 'gravatar_id': '', 'ur
l': 'https://api.github.com/users/EmnaKsontini', 'html_url': 'https://github.com/EmnaKsontini', 'followers_url': 'https://api.github.com/users/EmnaKsontini/followers', 'following_url': 'https://api.github.com/users/EmnaKsontini/following
{/other_user}', 'gists_url': 'https://api.github.com/users/EmnaKsontini/gists{/gist_id}', 'starred_url': 'https://api.github.com/users/EmnaKsontini/starred{/owner}{/repo}', 'subscriptions_url': 'https://api.github.com/users/EmnaKsontini/
subscriptions', 'organizations_url': 'https://api.github.com/users/EmnaKsontini/orgs', 'repos_url': 'https://api.github.com/users/EmnaKsontini/repos', 'events_url': 'https://api.github.com/users/EmnaKsontini/events{/privacy}', 'received_
events_url': 'https://api.github.com/users/EmnaKsontini/received_events', 'type': 'User', 'site_admin': False}, 'html_url': 'https://github.com/EmnaKsontini/NewRepo', 'description': 'This is a new repo', 'fork': False, 'url': 'https://ap
i.github.com/repos/EmnaKsontini/NewRepo', 'forks_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/forks', 'keys_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/keys{/key_id}', 'collaborators_url': 'https://api.github.com
/repos/EmnaKsontini/NewRepo/collaborators{/collaborator}', 'teams_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/teams', 'hooks_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/hooks', 'issue_events_url': 'https://api.g
ithub.com/repos/EmnaKsontini/NewRepo/issues/events{/number}', 'events_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/events', 'assignees_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/assignees{/user}', 'branches_url'
: 'https://api.github.com/repos/EmnaKsontini/NewRepo/branches{/branch}', 'tags_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/tags', 'blobs_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/git/blobs{/sha}', 'git_tags_ur
l': 'https://api.github.com/repos/EmnaKsontini/NewRepo/git/tags{/sha}', 'git_refs_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/git/refs{/sha}', 'trees_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/git/trees{/sha}',
 'statuses_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/statuses/{sha}', 'languages_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/languages', 'stargazers_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/sta
rgazers', 'contributors_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/contributors', 'subscribers_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/subscribers', 'subscription_url': 'https://api.github.com/repos/EmnaKso
ntini/NewRepo/subscription', 'commits_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/commits{/sha}', 'git_commits_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/git/commits{/sha}', 'comments_url': 'https://api.github.
com/repos/EmnaKsontini/NewRepo/comments{/number}', 'issue_comment_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/issues/comments{/number}', 'contents_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/contents/{+path}', '
compare_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/compare/{base}...{head}', 'merges_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/merges', 'archive_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/{archi
ve_format}{/ref}', 'downloads_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/downloads', 'issues_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/issues{/number}', 'pulls_url': 'https://api.github.com/repos/EmnaKsontini
/NewRepo/pulls{/number}', 'milestones_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/milestones{/number}', 'notifications_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/notifications{?since,all,participating}', 'label
s_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/labels{/name}', 'releases_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/releases{/id}', 'deployments_url': 'https://api.github.com/repos/EmnaKsontini/NewRepo/deploymen
ts', 'created_at': '2024-01-28T06:17:32Z', 'updated_at': '2024-01-28T06:17:32Z', 'pushed_at': '2024-01-28T06:17:32Z', 'git_url': 'git://github.com/EmnaKsontini/NewRepo.git', 'ssh_url': 'git@github.com:EmnaKsontini/NewRepo.git', 'clone_ur
l': 'https://github.com/EmnaKsontini/NewRepo.git', 'svn_url': 'https://github.com/EmnaKsontini/NewRepo', 'homepage': None, 'size': 0, 'stargazers_count': 0, 'watchers_count': 0, 'language': None, 'has_issues': False, 'has_projects': Fals
e, 'has_downloads': True, 'has_wiki': False, 'has_pages': False, 'has_discussions': False, 'forks_count': 0, 'mirror_url': None, 'archived': False, 'disabled': False, 'open_issues_count': 0, 'license': None, 'allow_forking': True, 'is_te
mplate': True, 'web_commit_signoff_required': False, 'topics': [], 'visibility': 'public', 'forks': 0, 'open_issues': 0, 'watchers': 0, 'default_branch': 'main'}, 'sender': {'login': 'EmnaKsontini', 'id': 36204977, 'node_id': 'MDQ6VXNlcj
M2MjA0OTc3', 'avatar_url': 'https://avatars.githubusercontent.com/u/36204977?v=4', 'gravatar_id': '', 'url': 'https://api.github.com/users/EmnaKsontini', 'html_url': 'https://github.com/EmnaKsontini', 'followers_url': 'https://api.github
.com/users/EmnaKsontini/followers', 'following_url': 'https://api.github.com/users/EmnaKsontini/following{/other_user}', 'gists_url': 'https://api.github.com/users/EmnaKsontini/gists{/gist_id}', 'starred_url': 'https://api.github.com/use
rs/EmnaKsontini/starred{/owner}{/repo}', 'subscriptions_url': 'https://api.github.com/users/EmnaKsontini/subscriptions', 'organizations_url': 'https://api.github.com/users/EmnaKsontini/orgs', 'repos_url': 'https://api.github.com/users/Em
naKsontini/repos', 'events_url': 'https://api.github.com/users/EmnaKsontini/events{/privacy}', 'received_events_url': 'https://api.github.com/users/EmnaKsontini/received_events', 'type': 'User', 'site_admin': False}}
←[32mINFO←[0m:     140.82.115.26:0 - "←[1mPOST /webhook_receiver HTTP/1.1←[0m" ←[32m200 OK←[0m
```