**HRITHIK TIWARI**

**1905394@KIIT.AC.IN**

**Documentation on the Project demo of creation of APIs**

What is an API?

An API is a program that takes in some data and gives back some other data, usually after processing it.

We will be building such programs, so that our users can send us some data, we can process it, and then we can send them something else.

For example, Twitter Search. Twitter search accepts data (the search parameters), processes it (finds the tweets that match those parameters), and sends data back (the tweets themselves).

If you are a user of the Twitter Search API, you can send them the search parameters and you get back the tweets.

Security in REST APIs is extremely important, because often applications that use our REST APIs will be sending us all sorts of data about users: passwords, e-mail addresses, names, postal addresses, security questions, bank details, and more.

In order to prevent people from intercepting the data on the internet and being able to read it, we must use Secure Sockets Layer. This sits on top of HTTP and encrypts all communication between a server and a client.

What this means is that if someone does intercept traffic that is encrypted, they won't be able to understand it.

In this section we'll learn about enabling HTTPS (SSL) in our REST APIs.

Flask-JWT-Extended works in very much a similar way to Flask-JWT—but it's a little bit more complicated.

Because of that, it also adds a bunch more functionality that can be really useful. One of those is token refresh.

We'll be learning about the differences between Flask-JWT and Flask-JWT-Extended in this section, as well as learning about the various extra features it gives us.

Attached to this lecture as a Resource is a short guide on how to configure Flask-JWT—it includes things like:

* Changing the authentication endpoint (by default, /auth);
* Changing the token expiration time (by default, 5 minutes);
* Changing the authentication key name (by default, username);
* Changing the authentication response body (by default, only contains access\_token); and
* Changing the error handlers.

**API are used in Auth –**

POST Request

1.<http://127.0.0.1:5000/register>

{

    "username" : "1232",

    "password" : "sala121",

    "confirmPassword": "sa121"

}

2. <http://127.0.0.1:5000/auth>

{

    "username" : "salav",

    "password" : "salav124"

}

3. <http://127.0.0.1:5000/item/phone>

{

"price" : 15.99

}

PUT Request

4. <http://127.0.0.1:5000/register>

Authorization - JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NTQ3NDQyNTAsImlhdCI6MTY1NDc0Mzk1MCwibmJmIjoxNjU0NzQzOTUwLCJpZGVudGl0eSI6M30.lrIM2imMtkBUQ7H5UXZJS2K8EQptrC0JI-0i3rEcZGE

GET Request

5. <http://127.0.0.1:5000/item/test>

{

    "price" : 15.99,

    "store\_id" : 1

 }

DELETE Request

6. <http://127.0.0.1:5000/register>

I have made project setting up a development environment, creating endpoints, handling authentication, and deploying the API, along with CRUD functionalities.

It is divided into several sections, each covering a different topic related to building REST APIs with Flask and Python:

Introduction:.Development Environment: This section covers the process of setting up a development environment for building REST APIs with Flask and Python.

Creating Endpoints: This section covers how to create endpoints for the REST API,how to use Flask's routing system to define URL patterns and handle HTTP requests.

Handling Data: This section covers how to handle data in the REST API,how to use Flask's request and response objects to interact with client requests and return JSON data.

Handling Authentication: This section covers how to handle authentication in the REST API,how to use Flask-HTTPAuth to secure endpoints and restrict access to authenticated users.

CRUD Functionalities: This section covers how to implement CRUD (Create, Read, Update, Delete) operations in the REST API,how to create endpoints for each of these operations and interact with a database to perform these actions

Deploying the API: This section covers how to deploy the REST API to a production environment. Students will learn how to use Flask's built-in development server and deploy the API to a cloud-based platform.

Conclusion

This Project provides a comprehensive introduction to building REST APIs with Flask and Python, including CRUD functionalities. We will gain a solid understanding of the Flask web framework and the REST architecture, and will be able to create, deploy, and secure RESTful web services using these technologies. By the end of the course, we will have the skills and knowledge needed to develop REST APIs with CRUD functionalities for a wide range of applications.

**Documentation on Nudge creation page wireframe for events**

Introduction

This API documentation is for the nudge creation page wireframe for events, which is used to create and edit nudges that are triggered by specific events in a web application. The nudge creation page for events allows users to create and configure nudges that will be displayed to website visitors when a specific event occurs. This documentation describes the API endpoints that can be used to interact with the nudge creation page wireframe for events.

Authentication

All API endpoints require authentication with a valid access token. The access token must be passed in the Authorization header of each request. The format of the header is as follows:

makefile

Authorization: Bearer <access\_token>

Endpoints

The following endpoints are available for the nudge creation page wireframe for events API:

GET /nudge/create/event/{event\_id}

This endpoint retrieves the nudge creation page wireframe data for the specified event. The event\_id parameter specifies the ID of the event for which to create a nudge.

Example Request:

sql

Copy code

GET /nudge/create/event/123 HTTP/1.1

Authorization: Bearer <access\_token>

Example Response:

json

{

"title": "Create Nudge",

"form\_fields": [

{

"name": "nudge\_title",

"label": "Nudge Title",

"type": "text",

"required": true

},

{

"name": "nudge\_message",

"label": "Nudge Message",

"type": "textarea",

"required": true

},

{

"name": "nudge\_position",

"label": "Nudge Position",

"type": "select",

"options": [

{ "value": "top-left", "label": "Top Left" },

{ "value": "top-center", "label": "Top Center" },

{ "value": "top-right", "label": "Top Right" },

{ "value": "bottom-left", "label": "Bottom Left" },

{ "value": "bottom-center", "label": "Bottom Center" },

{ "value": "bottom-right", "label": "Bottom Right" }

],

"required": true

},

{

"name": "event\_trigger",

"label": "Event Trigger",

"type": "hidden",

"value": "123"

}

]

}

POST /nudge/create/event/{event\_id}

This endpoint creates a new nudge for the specified event based on the data provided in the request body. The event\_id parameter specifies the ID of the event for which to create a nudge. The request body should be a JSON object containing the nudge data.

Example Request:

bash

POST /nudge/create/event/123 HTTP/1.1

Authorization: Bearer <access\_token>

Content-Type: application/json

{

"nudge\_title": "Welcome Message",

"nudge\_message": "Welcome to our website!",

"nudge\_position": "top-right"

}

Example Response:

json

{

"success": true,

"message": "Nudge created successfully"

}

PUT /nudge/event/{event\_id}/{nudge\_id}

This endpoint updates an existing nudge for the specified event with the data provided in the request body. The event\_id parameter specifies the ID of the event for which to update a nudge, and the nudge\_id parameter specifies the ID of the nudge to update.

Example Request:

bash

PUT /n