



Unlocking the potential : **REST APIs**





APIs have become an integral part of the modern digital landscape, driving innovation and enabling seamless integration between applications and services.

In this presentation, we will delve deeper into the world of APIs and explore their significance, working principles, and practical applications. We will gain insights into how APIs enable developers to leverage existing functionalities, accelerate development, and create seamless user experiences.



Hello, I'm Rwik Dey

I'm a first year undergraduate student from the department of Electrical Engineering, and I would like to present on the topic of REST APIs and their growing influence in our world.

Contents

- **What is an API ?**
- **Understanding Rest APIs**
- **Rest API Methods**
- **API Consumption**
- **Creating an basic API**
- **Real Life Application of APIs**
- **Conclusion**

What is an API ?

- API stands for Application Programming Interface.
- It serves as a set of rules and protocols that allows different software applications to communicate and interact with each other.
- APIs define the methods and data formats that applications should use to request and exchange information.



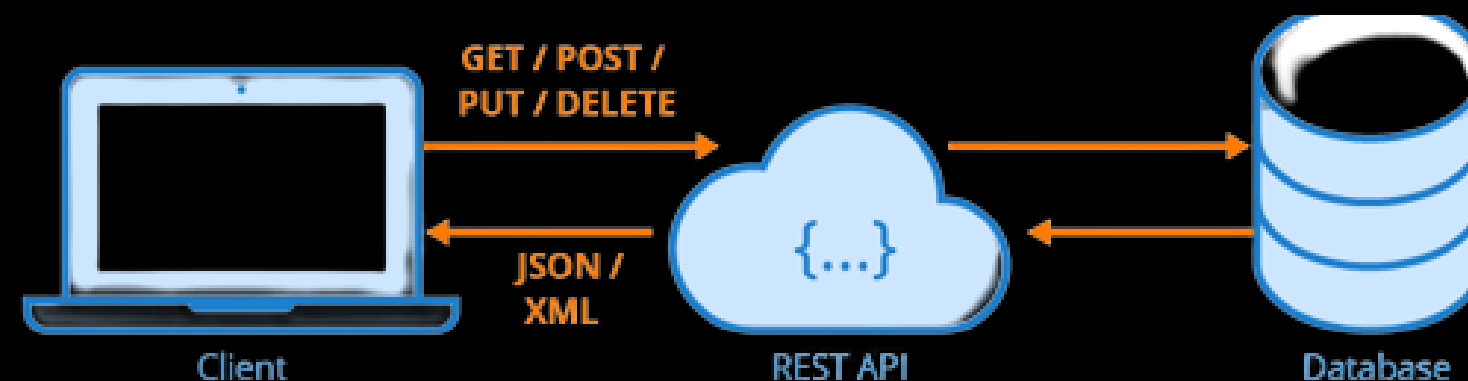
- APIs can be categorized into different types, such as web APIs (e.g., REST, SOAP) that use HTTP for communication, library APIs that provide pre-defined functions or classes for specific programming languages, and operating system APIs that expose system-level functionalities to developers.
- Overall, APIs play a crucial role in enabling interoperability, modularity, and extensibility in software development, allowing applications to work together, share data, and provide enhanced functionality through integration with external services or systems.

Understanding REST APIs

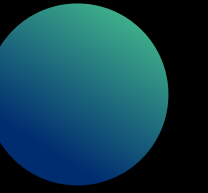
REST APIs, or **Representational State Transfer APIs, are a type of web API that follows the principles and constraints of the REST architectural style. REST is an architectural style for designing networked applications that leverage the existing capabilities of the World Wide Web.**

REST API Methods


REST API methods, also known as HTTP methods or HTTP verbs, form the foundation of building and interacting with RESTful APIs. They define the actions that can be performed on resources within an API, allowing clients to **read, create, update, or delete data**. Understanding REST API methods is essential for designing and consuming APIs that adhere to the principles of the REST architectural style.



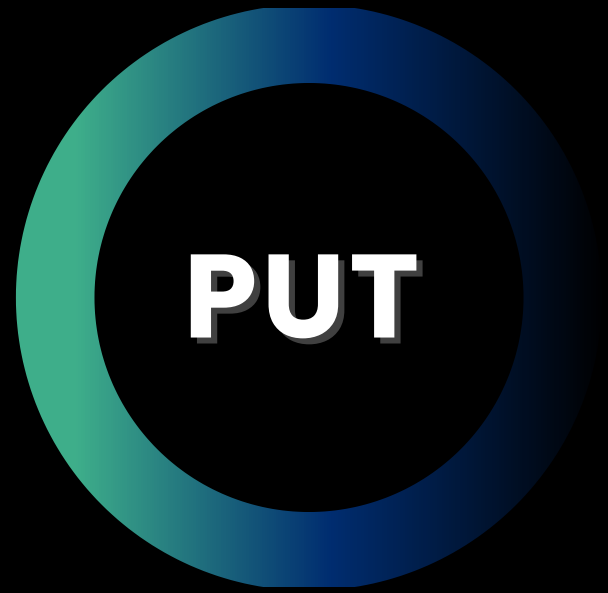
The common REST API Methods are :




- The GET method is an HTTP method used in REST APIs to retrieve data from a specified resource.
- It is safe and idempotent, meaning it does not modify data on the server and can be called multiple times without altering the resource's state.
- Clients send a GET request to a specific URL endpoint provided by the API to fetch the desired resource.
- GET requests do not have a request body and rely on the server to provide the requested resource in the response body.
- The response to a GET request includes the requested resource's representation along with metadata such as headers and status codes.

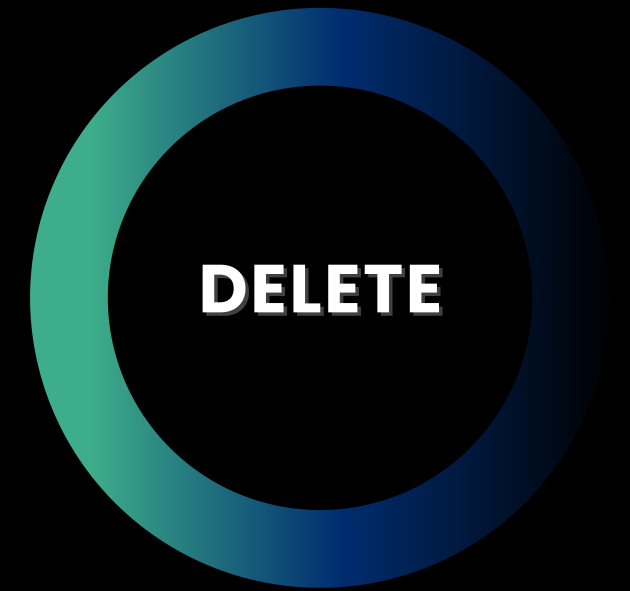
- 
- The POST method is used to submit data to the server and create new resources.
 - Clients send a POST request to a specific URL endpoint, including the data in the request body.
 - POST requests are non-idempotent, meaning calling the request multiple times can result in the creation of multiple resources.
 - The server processes the data and creates a new resource based on the provided information.
 - The server responds with a status code indicating the success or failure of the operation, along with any relevant details about the newly created resource.





- The PUT method is used to update or replace an existing resource in RESTful APIs.
- Clients send a PUT request to a specific URL endpoint, providing a complete representation of the updated resource in the request body.
- Unlike POST, PUT requests are idempotent, meaning calling the request multiple times produces the same result.
- The server processes the PUT request and either updates the existing resource with the provided data or creates a new resource if it doesn't exist.
- The response to a PUT request typically includes a status code indicating the success or failure of the operation.

- 
- The DELETE method is used to remove a specific resource identified by its URL in RESTful APIs.
 - Clients send a DELETE request to a specific URL endpoint to initiate the deletion of the resource.
 - The server processes the DELETE request and permanently removes the specified resource from the server.
 - DELETE requests are idempotent, meaning calling the request multiple times has the same outcome of removing the resource.
 - The server responds with a status code indicating the success or failure of the deletion operation.



API Consumption

- Identifying a suitable public API for your project. Popular APIs like **GITHub** can be used.
- Understanding the API documentation to familiarize yourself with its endpoints and data.
- Choosing a programming environment suitable for building a **frontend** or **CLI application**.
- Setting up the development environment, including necessary tools and libraries.

- Utilizing appropriate HTTP methods (**GET, POST, PUT, DELETE**) to make API requests and access the required data
- Implementing authentication if required by the API, following the specified authentication process
- Parsing and processing API responses to extract relevant information for your application.
- Testing and refining your application, ensuring proper functionality and a seamless user experience.

Creating a basic API

Flask is a lightweight and flexible web framework for building APIs in Python. It provides essential features like request parsing, response formatting, and error handling, allowing you to create robust APIs. Flask's simplicity makes it a popular choice for creating basic APIs, as it requires minimal setup and configuration. Whether you're building a small project or a larger application, Flask's versatility and extensive ecosystem make it a great tool for quickly developing APIs with Python.

Here I'm going to demonstrate a basic API, I created using Flask

```
#Creating Basic API using Flask and REST API Methods - GET and POST
from flask import Flask, render_template, redirect, url_for, request

app=Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/score/<int:score>')
def success(score):
    res=""
    if score>=50:
        res="PASSED"
    else:
        res="FAILED"

    return(render_template('result.html',result=res))

@app.route('/fail/<int:score>')
def fail(score):
    res=""
    if score>=50:
        res="PASSED"
    else:
        res="FAILED"

    return(render_template('result.html',result=res))
```

```
@app.route('/result/<int:marks>')
def result(marks):
    result=''
    if marks<30:
        result='Fail'
    else:
        result='Pass'
    return(redirect(url_for(result,score=marks)))

@app.route('/submit',methods=["GET","POST"])
def submit():
    total_score=0
    if request.method=="POST":
        science=float(request.form['science'])
        maths=float(request.form['maths'])
        chemistry=float(request.form['chemistry'])
        comp_science=float(request.form['computerscience'])
        total_score=(science+maths+chemistry+comp_science)/4
        res=""
        if total_score>=50:
            res="success"
        else:
            res="fail"
        return(redirect(url_for(res,score=total_score)))

if __name__=="__main__":
    app.run(debug=True)
```

This code demonstrates the usage of **Flask** for creating a basic API with routes that handle **GET and POST** requests and generate appropriate responses based on the provided input. Here's some info on the code :

- It defines several routes using the **@app.route** decorator to handle different URLs.
- The **'/'** route renders an **'index.html'** template when accessed.
- The **/score/<int:score>** and **/fail/<int:score>** routes handle URLs with an integer parameter called score.
- The **/result/<int:marks>** route redirects to either the 'success' or 'fail' route based on the value of marks.

- The **/submit** route handles both **GET and POST** requests and calculates the total score based on form input. It then redirects to either the 'success' or 'fail' route based on the calculated score.
- The **'success'** and **'fail'** routes render a **'result.html'** template, displaying the result based on the score.
- The Flask application is run with **app.run(debug=True)** to start the API server

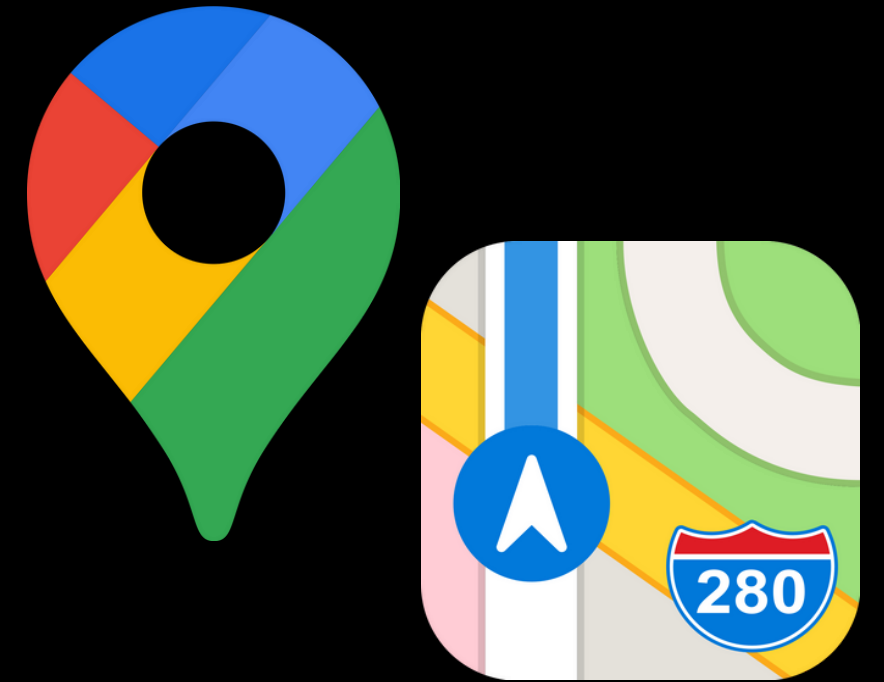
Real Life Applications of APIs

- **Social Media Integration:** APIs are commonly used to integrate social media platforms, allowing users to sign in, share content, retrieve user data, and interact with social features within applications.



- **Payment Gateways:** Payment service providers offer APIs that allow developers to integrate secure payment processing into their applications, enabling users to make online transactions.


- **Mapping and Geolocation Services:** APIs such as Google Maps provide developers with access to mapping and geolocation data, enabling the integration of location-based features like directions, geocoding, and real-time tracking..



- **Weather Data:** Weather APIs provide access to real-time and historical weather information, allowing developers to incorporate weather forecasts and conditions into their applications, such as weather apps, travel planners, or event organizers.



Conclusion

- In conclusion, REST APIs have become the de facto standard for building web services due to their simplicity, scalability, and interoperability. They provide a structured and uniform approach to designing and interacting with APIs.
 - By adhering to REST principles, developers can create robust and scalable APIs that can be easily consumed by clients, fostering seamless integration and enabling the development of diverse and interconnected applications.
- 



Thank You