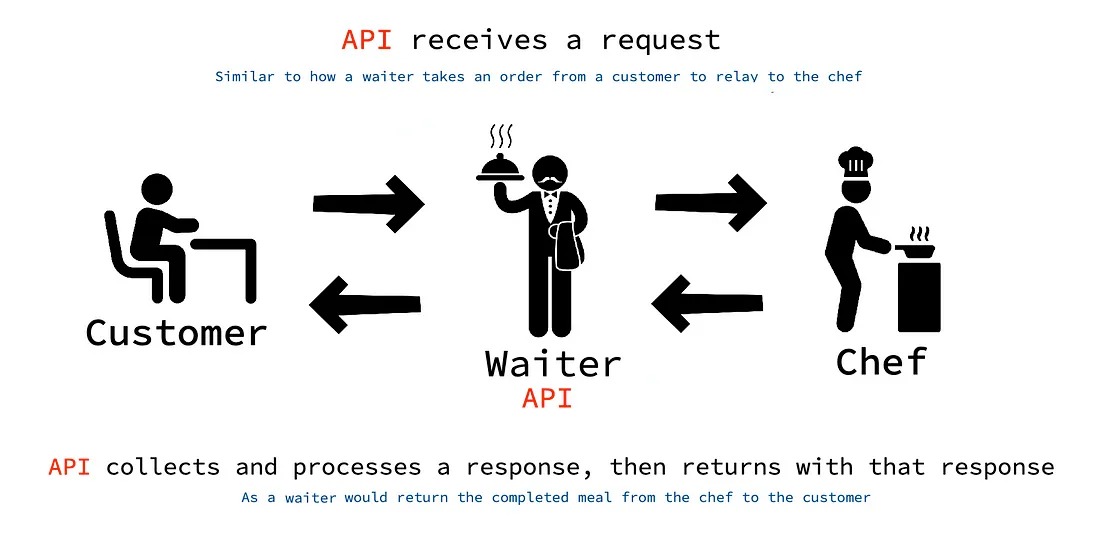
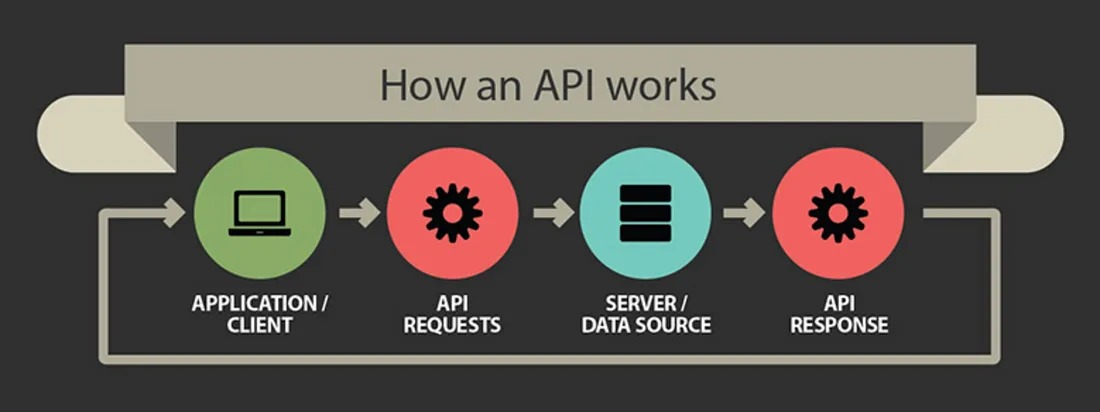
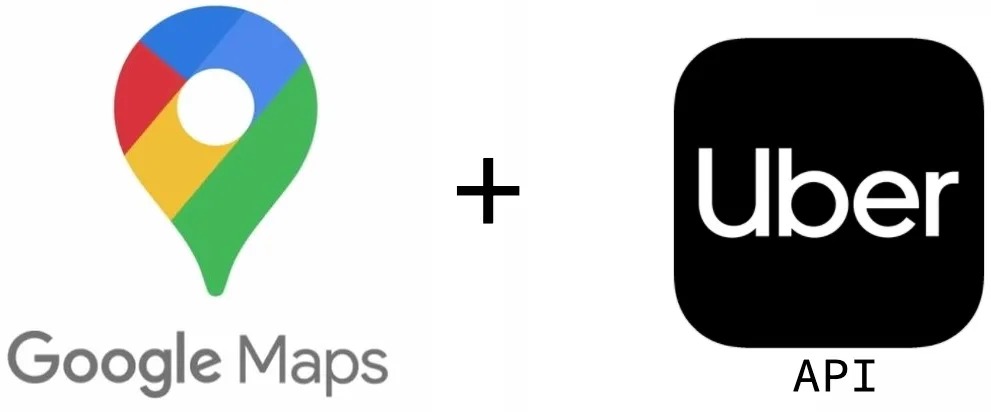
**What is API**

Application Programming Interface



**Real\_Time Example** :

**Let’s Build Our Own**

**REST API from Scratch using MONGO DB or Atlas ( cloud )**

We'll use **Python** + **Flask**, a beginner-friendly micro web framework.

**Step 1: Setup Environment**

pip install flask pymongo dnspython

**Create a Folder**

Simple\_api /

|--------app.py

STEPS :

1. Go to: <https://cloud.mongodb.com/>
2. Create an account → New Project → Create Cluster (Shared)
3. Create a database (e.g. collegeDB) and collection (e.g. students)
4. Get the **connection string**, like:  
   mongodb+srv://<username>:<password>@cluster0.xxxx.mongodb.net/collegeDB
5. Start MongoDB server locally: in CMD : mongod
6. Connection string for local: mongodb://localhost:27017

App.py

from flask import Flask, jsonify, request

from pymongo import MongoClient

from bson import ObjectId

import os

app = Flask(\_\_name\_\_)

# MongoDB setup (change this to local if needed)

client = MongoClient("mongodb://localhost:27017")  # or use MongoDB Atlas URI

db = client["collegeDB"]

students\_collection = db["students"]

# Convert ObjectId to string

def fix\_id(data):

    data["\_id"] = str(data["\_id"])

    return data

# Home

@app.route('/')

def home():

    return "🎓 Welcome to Student REST API with MongoDB!"

# GET all students

@app.route('/students', methods=['GET'])

def get\_students():

    students = list(students\_collection.find())

    return jsonify([fix\_id(s) for s in students])

# POST a new student

@app.route('/students', methods=['POST'])

def add\_student():

    data = request.get\_json()

    inserted = students\_collection.insert\_one(data)

    return jsonify({"id": str(inserted.inserted\_id)}), 201

# PUT - Update a student

@app.route('/students/<id>', methods=['PUT'])

def update\_student(id):

    data = request.get\_json()

    result = students\_collection.update\_one({"\_id": ObjectId(id)}, {"$set": data})

    if result.modified\_count > 0:

        return jsonify({"message": "Student updated"})

    return jsonify({"message": "No changes or student not found"}), 404

# DELETE - Remove a student

@app.route('/students/<id>', methods=['DELETE'])

def delete\_student(id):

    result = students\_collection.delete\_one({"\_id": ObjectId(id)})

    if result.deleted\_count > 0:

        return jsonify({"message": "Student deleted"})

    return jsonify({"message": "Student not found"}), 404

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

Operations Can be Done using Rest API :

Example JSON for POST (use in Postman or frontend):

{

  "name": "Kiran Kumar",

  "email": "kiran@example.com",

  "skills": ["Python", "AI", "ML"]

}

GET /students :

[

  {

    "\_id": "66aa55993c88f1fd88cc9003",

    "name": "Kiran Kumar",

    "email": "kiran@example.com",

    "skills": ["Python", "AI", "ML"]

  }

]

To get ALL Students :

Method: GET

URL: http://localhost:5000/students

No Body

**PUT (Update by ID):**

  Method: PUT

URL: http://localhost:5000/students/<id>

Body → raw → JSON:

{

  "name": "Kiran Updated",

  "email": "kiran@update.com"

}

**DELETE Student by ID :**

 Method: DELETE

URL: http://localhost:5000/students/<id>

example : http://localhost:5000/students/66b20e3403fc51b7b884e8e9

**CURL Commands :**

CURL Commands

  GET  : curl http://localhost:5000/students

  POST  :

  curl -X POST http://localhost:5000/students \

  -H "Content-Type: application/json" \

  -d '{"name": "Kiran", "email": "kiran@example.com", "skills": ["Python", "MongoDB"]}'

  PUT (Update) :

  curl -X PUT http://localhost:5000/students/<ID> \

  -H "Content-Type: application/json" \

  -d '{"name": "Updated Kiran", "email": "new@example.com"}'

  DELETE :

  curl -X DELETE http://localhost:5000/students/<ID>

  example :

  curl -X DELETE http://localhost:5000/students/66b20e3403fc51b7b884e8e9

**With Security JWT**

**Json web Tokens**

**Rest\_API\_JWT \**

**├── app.py**

**├── .env**

**├── .gitignore**

**App.py**

# pip install flask pymongo python-dotenv flask-jwt-extended bcrypt dnspython

from flask import Flask, jsonify, request

from flask\_jwt\_extended import (

    JWTManager, create\_access\_token,

    jwt\_required, get\_jwt\_identity

)

from pymongo import MongoClient

from bson import ObjectId

from dotenv import load\_dotenv

import os

import bcrypt

# Load .env

load\_dotenv()

# Flask app setup

app = Flask(\_\_name\_\_)

app.config['JWT\_SECRET\_KEY'] = os.getenv("JWT\_SECRET")

# JWT

jwt = JWTManager(app)

# MongoDB setup

MONGO\_URI = os.getenv("MONGO\_URI")

DB\_NAME = os.getenv("DB\_NAME")

STUDENT\_COLLECTION = os.getenv("STUDENT\_COLLECTION")

USER\_COLLECTION = os.getenv("USER\_COLLECTION")

client = MongoClient(MONGO\_URI)

db = client[DB\_NAME]

students = db[STUDENT\_COLLECTION]

users = db[USER\_COLLECTION]

# Helper to convert ObjectId

def fix\_id(data):

    data["\_id"] = str(data["\_id"])

    return data

# Register (signup)

@app.route('/register', methods=['POST'])

def register():

    data = request.get\_json()

    if users.find\_one({"email": data['email']}):

        return jsonify({"message": "User already exists"}), 409

    hashed = bcrypt.hashpw(data['password'].encode('utf-8'), bcrypt.gensalt())

    users.insert\_one({

        "email": data['email'],

        "password": hashed

    })

    return jsonify({"message": "User registered successfully"}), 201

# Login

@app.route('/login', methods=['POST'])

def login():

    data = request.get\_json()

    user = users.find\_one({"email": data['email']})

    if not user or not bcrypt.checkpw(data['password'].encode('utf-8'), user['password']):

        return jsonify({"message": "Invalid email or password"}), 401

    token = create\_access\_token(identity=user['email'])

    return jsonify({"access\_token": token})

# Protected route: Get all students

@app.route('/students', methods=['GET'])

@jwt\_required()

def get\_students():

    all\_students = list(students.find())

    return jsonify([fix\_id(s) for s in all\_students])

# Protected: Add student

@app.route('/students', methods=['POST'])

@jwt\_required()

def add\_student():

    data = request.get\_json()

    result = students.insert\_one(data)

    return jsonify({"message": "Student added", "id": str(result.inserted\_id)}), 201

# Protected: Update student

@app.route('/students/<id>', methods=['PUT'])

@jwt\_required()

def update\_student(id):

    data = request.get\_json()

    result = students.update\_one({"\_id": ObjectId(id)}, {"$set": data})

    if result.modified\_count:

        return jsonify({"message": "Student updated"})

    return jsonify({"message": "Not found or no changes"}), 404

# Protected: Delete student

@app.route('/students/<id>', methods=['DELETE'])

@jwt\_required()

def delete\_student(id):

    result = students.delete\_one({"\_id": ObjectId(id)})

    if result.deleted\_count:

        return jsonify({"message": "Student deleted"})

    return jsonify({"message": "Student not found"}), 404

# Home

@app.route('/')

def home():

    return "🎓 JWT-Secured Student API"

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**.env**

MONGO\_URI=mongodb://localhost:27017

DB\_NAME=collegeDB

STUDENT\_COLLECTION=students

USER\_COLLECTION=users

JWT\_SECRET=super-secret-key123

**.gitignore**

# ❌ Python cache files

\_\_pycache\_\_/

\*.pyc

# ❌ Environment variables and secrets

.env

# ❌ OS-specific files

.DS\_Store

Thumbs.db

# ❌ VSCode settings (optional, if using VSCode)

.vscode/

# ❌ Virtual environment folders (if any)

venv/

env/

**Commands :**

**\*Register\***

**POST http://localhost:5000/register**

**Body (raw JSON):**

**{ "email": "kiran@example.com", "password": "mypassword" }**

**\*Login\* (get JWT token)**

**POST http://localhost:5000/login**

**Body:**

**{ "email": "kiran@example.com", "password": "mypassword" }**

**Response:**

**{**

**"access\_token": "eyJ0eXAiOiJKV1QiLCJh..."**

**}**

**\*\*\*\* Set Up Postman First \*\*\*\***

1. **Global Step (for all secured routes)**
2. **After login, copy the access\_token**

**In Postman → Go to Headers**

**Add:**

**Key: Authorization**

**Value: Bearer <your\_access\_token\_here>**

**3. Get All Students (Protected)**

**Method: GET**

**URL: http://localhost:5000/students**

**Headers:**

**Authorization: Bearer <your\_token\_here>**

**4. Add a Student (Protected)**

**Method: POST**

**URL: http://localhost:5000/students**

**Headers:**

**Authorization: Bearer <your\_token\_here>**

**Content-Type: application/json**

**Body:**

**{ "name": "Kiran Kumar", "email": "kiran@example.com", "skills": ["Python", "MongoDB"]**

**}**

**5. Update a Student (Protected)**

**Method: PUT**

**URL: http://localhost:5000/students/<student\_id>**

**Headers:**

**Authorization: Bearer <your\_token\_here>**

**Content-Type: application/json**

**{**

**"name": "Kiran Updated",**

**"email": "updated@example.com"**

**}**

**6. Delete a Student (Protected)**

**Method: DELETE**

**URL: http://localhost:5000/students/<student\_id>**

**Headers:**

**Authorization: Bearer <your\_token\_here>**