Using JWT for user authentication in Flask

Last Updated : 09 Feb, 2023

Pre-requisite: Basic knowledge about JSON Web Token (JWT)
I will be assuming you have the basic knowledge of JWT and how JWT works. If not, then I suggest reading the linked Geeksforgeeks article.

Let's jump right into the setup. Ofcourse, you need **python3** installed on your system. Now, follow along with me. I will be using a **virtual environment** where I will install the libraries which is undoubtedly the best way of doing any kind of development.

- First create a folder named **flask project** and change directory to it. If you are on linux then type the following in your terminal.

```
mkdir "flask project" && cd "flask project"
```

- Now, create a virtual environment. If you are on linux then type the following in your terminal.

```
python3 -m venv env
```

Note: If you get any error then that means **venv** isn't installed in your system. To install it, type *sudo apt install python3-venv* in your terminal and then you are good to go. If you are on windows then use something like *virtualenv* to make a virtual environment.

This will create a folder named **venv** in the **flask project** which will contain the project specific libraries.

- Now create a file named *requirements.txt* and add the following lines in it.

```
Flask-RESTful==0.3.8
PyJWT==1.7.1
```

```
Flask-SQLAlchemy==2.4.1
```

- Now, lets install these libraries for this project. To do so, first we need to activate the virtual environment. To do so, type the following in your terminal.

```
source env/bin/activate
```

Note: If you are on windows then it would be *Scripts* instead of *bin*
Now, its time to install the libraries. To do so, again type the following in your terminal.

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

Now, we are done with the setup part. Lets now start writing the actual code. Before beginning with the code, I would like to make something clear. I would be writing the entire code in a single file, i.e. the database models and the routes all together, which is not a good practice and definitely not manageable for larger projects. Try keeping creating separate python files or modules for routes and database models.
With that cleared out, lets directly jump into the writing the actual code. I will be adding inline comments explaining every part of the code.

Create a python file called *app.py* and type the following code in it.

Python3

```
# flask imports
from flask import Flask, request, jsonify, make_response
from flask_sqlalchemy import SQLAlchemy
import uuid # for public id
from  werkzeug.security import generate_password_hash, check_password_hash
# imports for PyJWT authentication
import jwt
from datetime import datetime, timedelta
from functools import wraps

# creates Flask object
app = Flask(__name__)
# configuration
# NEVER HARDCODE YOUR CONFIGURATION IN YOUR CODE
# INSTEAD CREATE A .env FILE AND STORE IN IT
app.config['SECRET_KEY'] = 'your secret key'
# database name
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///Database.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = True
# creates SQLALCHEMY object
db = SQLAlchemy(app)

# Database ORMs
class User(db.Model):
    id = db.Column(db.Integer, primary_key = True)
    public_id = db.Column(db.String(50), unique = True)
    name = db.Column(db.String(100))
    email = db.Column(db.String(70), unique = True)
    password = db.Column(db.String(80))

# decorator for verifying the JWT
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None
        # jwt is passed in the request header
        if 'x-access-token' in request.headers:
            token = request.headers['x-access-token']
        # return 401 if token is not passed
        if not token:
            return jsonify({'message' : 'Token is missing !!'}), 401

        try:
            # decoding the payload to fetch the stored details
            data = jwt.decode(token, app.config['SECRET_KEY'])
            current_user = User.query\
                .filter_by(public_id = data['public_id'])\
                .first()
        except:
            return jsonify({
                'message' : 'Token is invalid !!'
            }), 401
        # returns the current logged in users context to the routes
        return  f(current_user, *args, **kwargs)

    return decorated

# User Database Route
# this route sends back list of users
@app.route('/user', methods =['GET'])
@token_required
def get_all_users(current_user):
    # querying the database
```

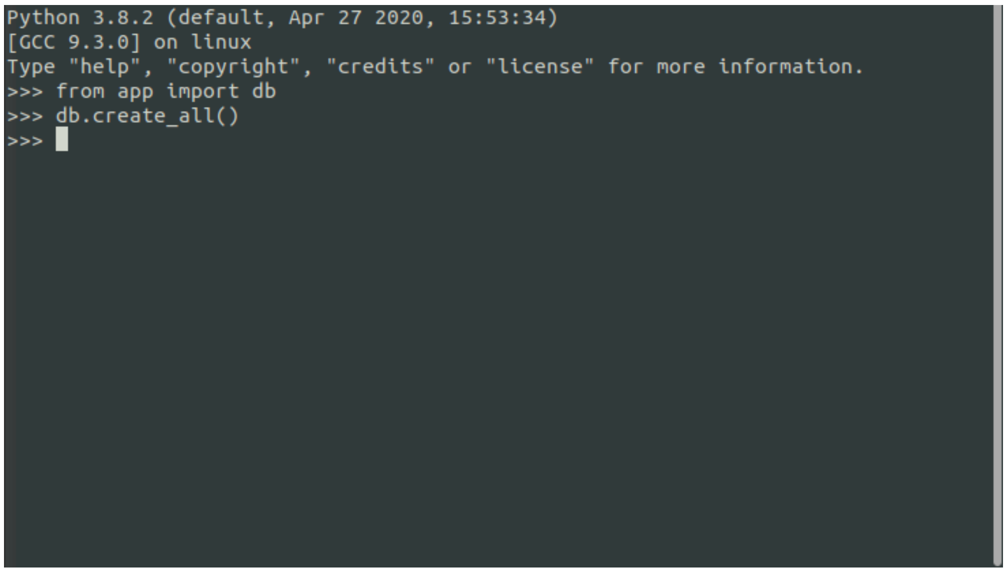
```
        # for all the entries in it
        users = User.query.all()
        # converting the query objects
        # to list of jsons
        output = []
        for user in users:
            # appending the user data json
            # to the response list
            output.append({
                'public_id': user.public_id,
                'name' : user.name,
                'email' : user.email
            })

        return jsonify({'users': output})

    # route for logging user in
    @app.route('/login', methods =['POST'])
    def login():
        # creates dictionary of form data
        auth = request.form

        if not auth or not auth.get('email') or not auth.get('password'):
            # returns 401 if any email or / and password is missing
            return make_response(
                'Could not verify',
                401,
                {'WWW-Authenticate' : 'Basic realm ="Login required !!"'}
            )

        user = User.query\
            .filter_by(email = auth.get('email'))\
            .first()

        if not user:
            # returns 401 if user does not exist
            return make_response(
                'Could not verify',
                401,
                {'WWW-Authenticate' : 'Basic realm ="User does not exist !!"'}
            )

        if check_password_hash(user.password, auth.get('password')):
            # generates the JWT Token
            token = jwt.encode({
                'public_id': user.public_id,
                'exp' : datetime.utcnow() + timedelta(minutes = 30)
            }, app.config['SECRET_KEY'])
            return make_response(jsonify({'token' : token.decode('UTF-8')}), 201)
        # returns 403 if password is wrong
        return make_response(
            'Could not verify',
            403,
            {'WWW-Authenticate' : 'Basic realm ="Wrong Password !!"'}
        )

    # signup route
    @app.route('/signup', methods =['POST'])
    def signup():
        # creates a dictionary of the form data
        data = request.form

        # gets name, email and password
        name, email = data.get('name'), data.get('email')
        password = data.get('password')

        # checking for existing user
        user = User.query\
            .filter_by(email = email)\
            .first()
        if not user:
            # database ORM object
            user = User(
                public_id = str(uuid.uuid4()),
                name = name,
                email = email,
                password = generate_password_hash(password)
            )
            # insert user
            db.session.add(user)
            db.session.commit()

            return make_response('Successfully registered.', 201)
        else:
            # returns 202 if user already exists
            return make_response('User already exists. Please Log in.', 202)

    if __name__ == "__main__":
        # setting debug to True enables hot reload
        # and also provides a debugger shell
        # if you hit an error while running the server
        app.run(debug = True)
```

Now, our code is ready. We now need to create the database first and then the table *User* from the ORM (Object Relational Mapping). To do so, first start the python3 interpreter in your terminal. You can do that by typing *python3* in your terminal and that should do the trick for you.

Next you need to type the following in your python3 interpreter:

```
from app import db
db.create_all()
```

So, what this does is first it imports the database object and then calls the *create_all()* function to create all the tables from the ORM. It should look something like this.



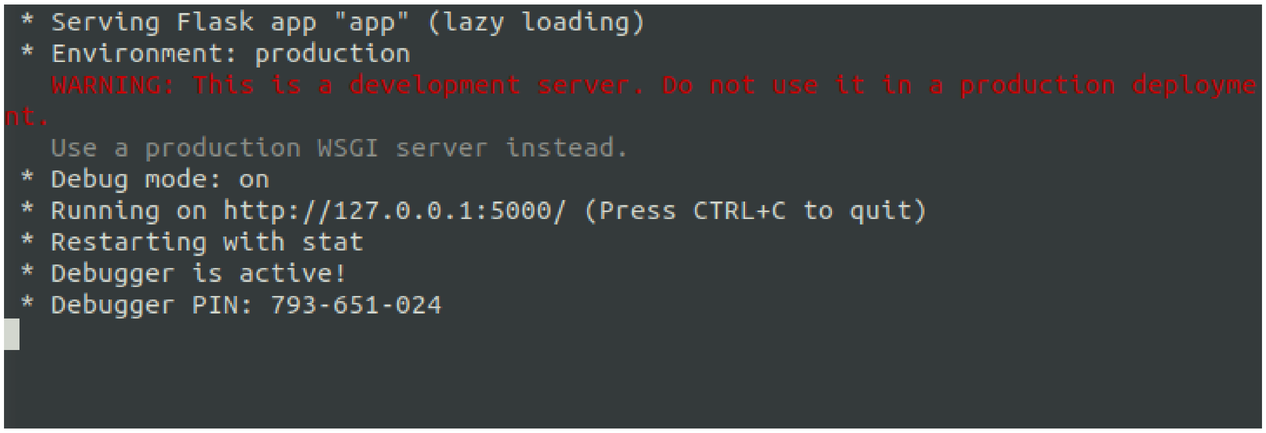
```
Python 3.8.2 (default, Apr 27 2020, 15:53:34)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from app import db
>>> db.create_all()
>>> 
```

Now that our actual code is ready, let's test it out. I recommend using postman for testing out the APIs. You can use something like CURL but I will be using postman for this tutorial.

To start testing our api, first we need to run our API. To do so, open up a terminal window and type the following in it.

```
python app.py
```

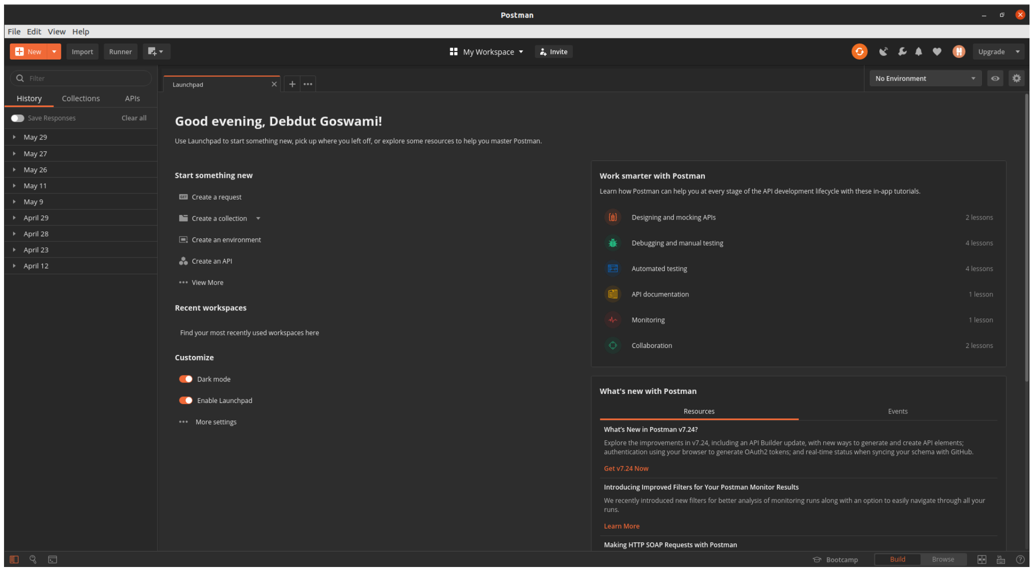
You should see an output like this



```
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 793-651-024
```

If you get any error then make sure all your syntax and indentation are correct. You can see that our api is running on http://localhost:5000/. Copy this url. We will use this urlalong with the routes to test the api.

Now, open up *Postman*. You should be greated with the following screen.



Good evening, Debdut Goswami!

Use Launchpad to start something new, pick up where you left off, or explore some resources to help you master Postman.

Start something new

- Create a request
- Create a collection
- Create an environment
- Create an API
- View More

Recent workspaces

Find your most recently used workspaces here

Customize

- Dark mode
- Enable Launchpad
- More settings

Work smarter with Postman

Learn how Postman can help you at every stage of the API development lifecycle with these in-app tutorials.

Designing and mocking APIs	2 lessons
Debugging and manual testing	4 lessons
Automated testing	4 lessons
API documentation	1 lesson
Monitoring	1 lesson
Collaboration	2 lessons

What's new with Postman

Resources Events

What's New in Postman v7.24?

Explore the improvements in v7.24, including an API Builder update, with new ways to generate and create API elements; authentication using your browser to generate OAuth2 tokens; and real-time status when syncing your schema with GitHub.

Get v7.24 Now

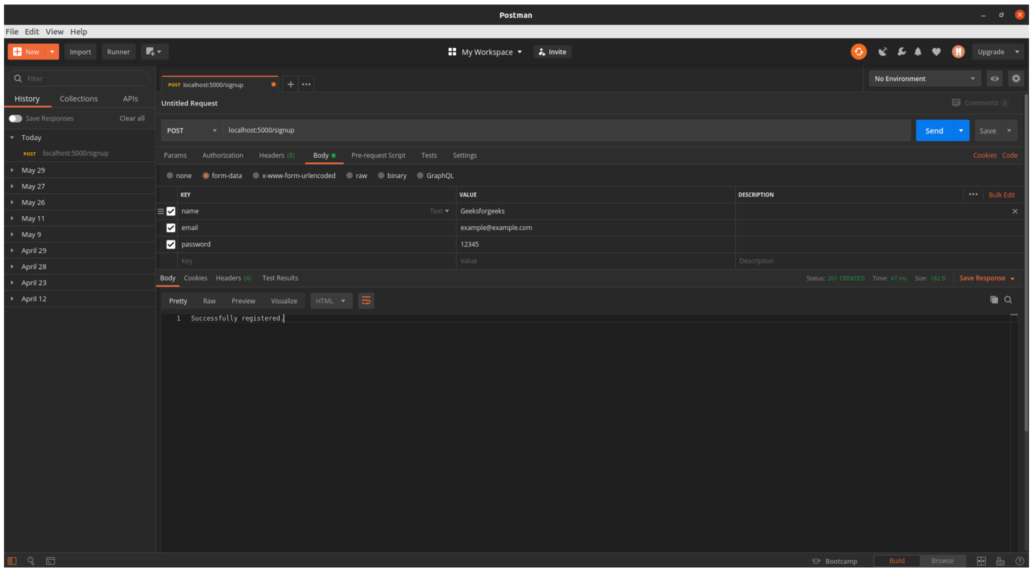
Introducing Improved Filters for Your Postman Monitor Results

We recently introduced new filters for better analysis of monitoring runs along with an option to easily navigate through all your runs.

Learn More

Making HTTP SOAP Requests with Postman

Now, click on the + sign and enter the url *localhost:5000/signup* change request type to *POST*, then select *Body* and then *form-data* and enter the data as key-value pair and then click on *Send* and you should get a response. It should look something like this.



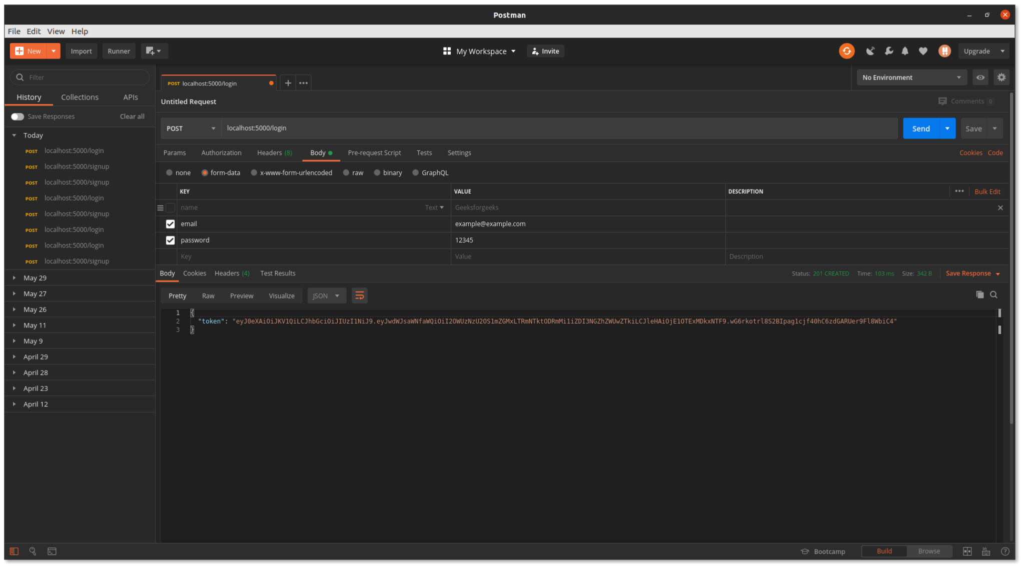
POST localhost:5000/signup

KEY	VALUE	DESCRIPTION
✓ name	Geeksforgeeks	
✓ email	example@example.com	
✓ password	12345	

Status: 201 CREATED Time: 47 ms Size: 182 B

```
Successfully registered
```

So, we are registered. Now lets login. To do that just change the endpoint to */login* and untick the *Name* field and click on *Send*. You should get a JWT as a response. Note down that JWT. That will be our token and we will need to send that token along with every subsequent requests. This token will identify us as logged in.



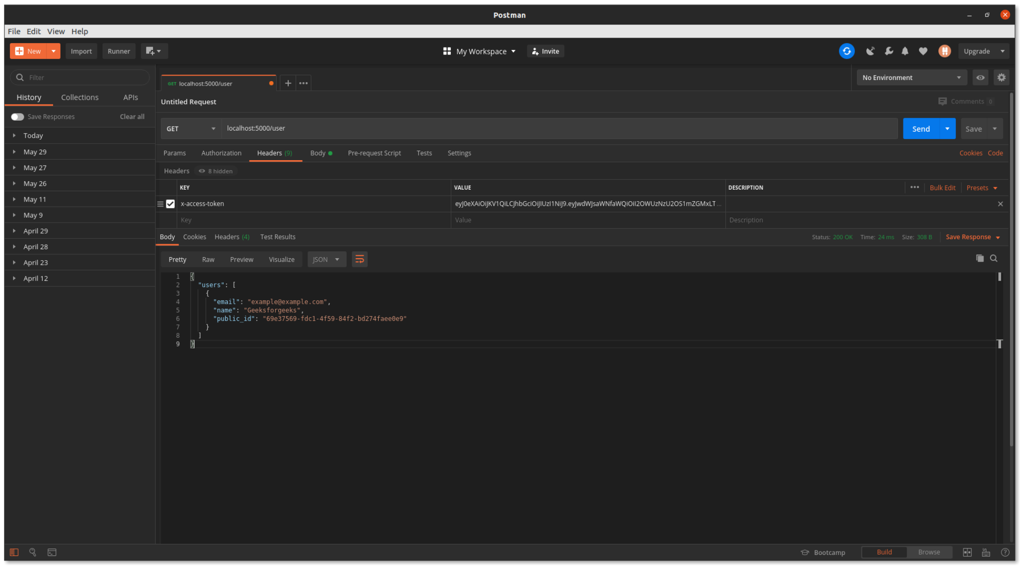
POST localhost:5000/login

KEY	VALUE	DESCRIPTION
name	Geeksforgeeks	
✓ email	example@example.com	
✓ password	12345	

Status: 201 CREATED Time: 103 ms Size: 342 B

```
{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfaWQiOiI2OWUzNzU2OS1mZGMxLTRmNTktODRmMi1iZDI3NGZhZWUwZTkiLCJleHAiOjE1OTExMDkxNTF9.wG6rkotrl8S2BIpag1cjf40hC6zdGARUer9Fl8WbiC4"
}
```

The JSON contains the token. Note it down. Next try to fetch the list of users. To do that, change the endpoint to */user* and then in the headers section, add a field as *x-access-token* and add the JWT token in the value and click on *Send*. You will get the list of users as JSON.


Postman
My Workspace
Invite
Upgrade
No Environment
Untitled Request
Send
Save
Params Authorization Headers (9) Body Pre-request Script Tests Settings
Body Cookies Headers (4) Test Results
Pretty Raw Preview Visualize JSON

GET localhost:5000/user

KEY	VALUE	DESCRIPTION
x-access-token	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfaWQiOiI2OWUzNzU2OS1mZGMxLT...	

Status: 200 OK Time: 24 ms Size: 308 B

```
{
  "users": [
    {
      "email": "example@example.com",
      "name": "Geeksforgeeks",
      "public_id": "69e37569-fdc1-4f59-84f2-bd274faee0e9"
    }
  ]
}
```

So, this is how you can perform authentication with JWT in Flask. I recommend you to practice more with JWTs and user authentication to get your concepts more clear.

debd...

10

Previous Article

How to Run a Flask Application

Next Article

Similar Reads

JWT Authentication with Django REST Framework

JSON Web Token is an open standard for securely transferring data within parties using a JSON object. JWT is used for stateless authentication mechanisms for...

2 min read

Documenting Flask Endpoint using Flask-Autodoc

Documentation of endpoints is an essential task in web development and being able to apply it in different frameworks is always a utility. This article discusses...

4 min read

Minify HTML in Flask using Flask-Minify

Flask offers HTML rendering as output, it is usually desired that the output HTML should be concise and it serves the purpose as well. In this article, we would...

12 min read

How to use Flask-Session in Python Flask ?

Flask Session - Flask-Session is an extension for Flask that supports Server-side Session to your application.The Session is the time between the client logs in to...

4 min read

How to Integrate Flask-Admin and Flask-Login

In order to merge the admin and login pages, we can utilize a short form or any other login method that only requires the username and password. This is know...

8 min read

Flask URL Helper Function - Flask url_for()

In this article, we are going to learn about the flask url_for() function of the flask URL helper in Python. Flask is a straightforward, speedy, scalable library, used f...

11 min read

OAuth Authentication with Flask - Connect to Google, Twitter, and Facebook

In this article, we are going to build a flask application that will use the OAuth protocol to get user information. First, we need to understand the OAuth protoc...

8 min read

How To Add Authentication to Your App with Flask-Login

Whether it is building a simple blog or a social media site, ensuring user sessions are working correctly can be tricky. Fortunately, Flask-Login provides a simplifie...

9 min read

Flask API Authentication with JSON Web Tokens

Authentication is the process of verifying the identity of the user. It checks whether the user is real or not. It is used to provide access to resources only to...

7 min read

Setup API for GeeksforGeeks user data using WebScraping and Flask

Prerequisite: WebScraping in Python, Introduction to Flask In this post, we will discuss how to get information about a GeeksforGeeks user using web scraping...

3 min read

Article Tags : Python Python Flask

Practice Tags : python

Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial

Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Inteview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects