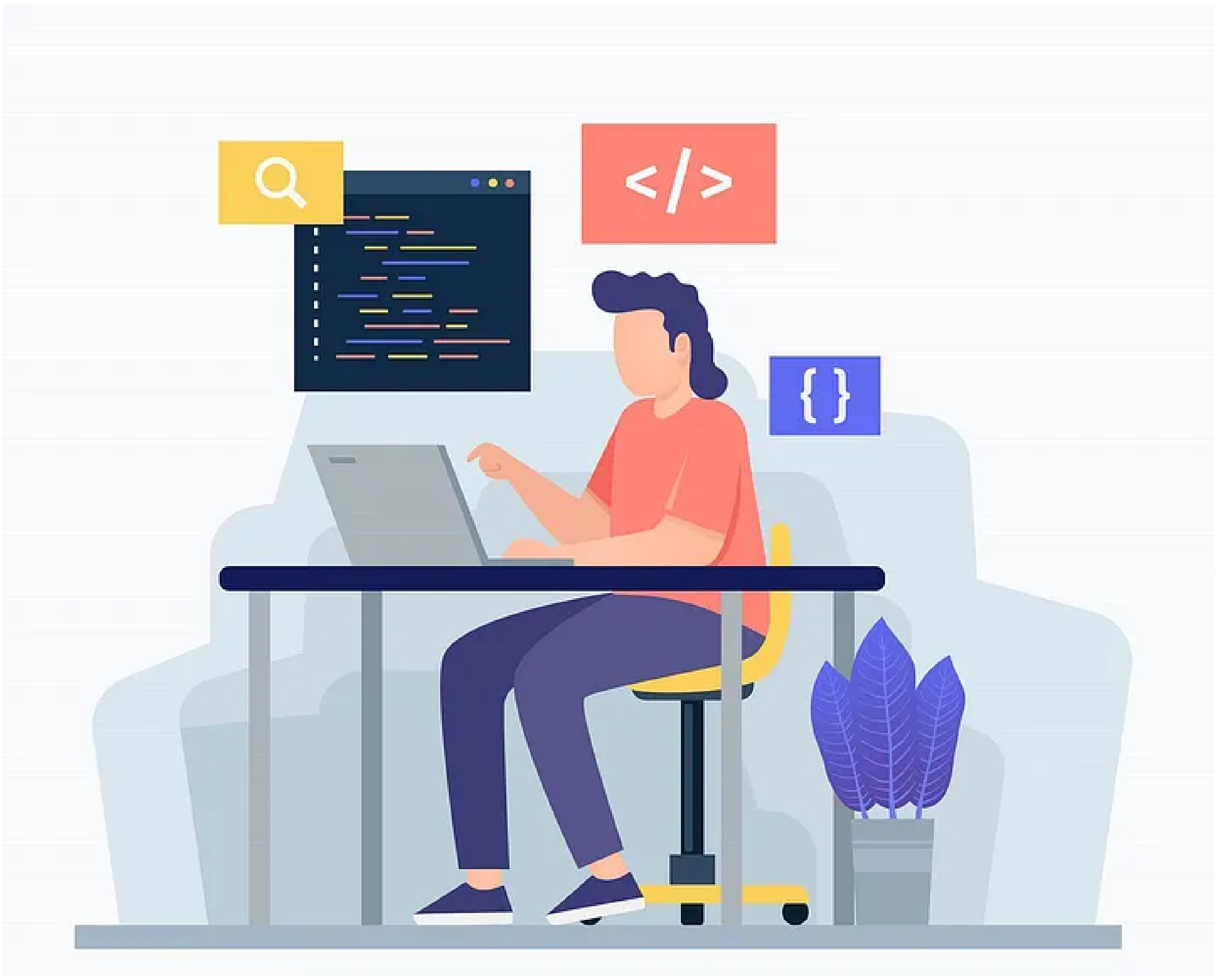


Business in the Front, Party in the Back— Combining Flask and React for Full-Stack Web Development

Sofia Katzman · Follow
7 min read · Aug 4, 2023

In the world of web development, Flask and React stand out as powerful tools for backend and frontend development. Combining these technologies allows developers to create sophisticated full-stack web applications that are both efficient and dynamic. When approaching my first deployment, I was a bit unsure and didn't quite understand exactly how how Flask and React came together seamlessly to build a cohesive and robust web application. By the end of this blog, I hope you will have a clear understanding of how to structure your project, handle API routes, set up front-end navigation, and even deploy your application directly from Flask!

Setting up the Project

Before we begin integrating Flask and React, let's first outline the necessary files and their roles in the project. Ideally, your structure should look something like what is below.

```
- server
  - app.py
  - config.py

- client
  - src
    - components
      - NavigationBar.js
      - Home.js
      - About.js
      - Contact.js
    - App.js
    - index.js
  - public
    - index.html
    - favicon.ico
  - package.json
  - README.md

- Pipfile
- README.md
```

> Server Directory :
> `server` *(directory): Contains the backend code for the Flask application.*
> `app.py` *: The main Flask application file that defines the core functionality of the server, handles requests, and manages the integration with React.*
> `config.py` *: Stores configuration variables that the Flask app may require, such as database credentials and environment settings.*
>
> Client Directory :
> `client` *(directory): Contains the frontend code for the React.js application.*
> `src` *(directory): Contains the source code for the React application.*
> `components` *(directory): Holds individual React components for the application.*
> `NavigationBar.js` *: A sample navigation bar component with navigation links.*
> `Home.js` *,* `About.js` *,* `Contact.js` *: Sample components representing different pages of the application.*
> `App.js` *: The main component that handles the routing and rendering of different pages/components.*
> `index.js` *: The entry point of the React application.*
> `public` *(directory): Contains static assets for the React application.*
> `index.html` *: The HTML template that serves as the entry point for the React application.*
> `favicon.ico` *: An example favicon for the application.*
> `package.json` *: The package configuration file for managing dependencies and scripts for the React application.*
>
> Root Directory :
> `Pipfile` *: The Pipfile is used with Pipenv, a package manager for Python projects. It specifies the dependencies and Python version required for the backend application.*
> `README.md` *: The main README file for the entire project. It typically contains general project information, installation instructions, and any other relevant details for both the frontend and backend components.*

This structure separates the frontend and backend code, making it easy to manage and deploy the full-stack web application. The `client` directory holds the React frontend, while the `server` directory contains the Flask backend, and both are organized alongside the necessary configuration files (`Pipfile` and root `README.md`).

Creating a Flask Database

To get started with the backend, we need to set up a database using Flask-SQLAlchemy. Flask-SQLAlchemy is a powerful ORM (Object-Relational Mapping) tool that simplifies database interactions in Flask applications. By using Flask-SQLAlchemy, we can define database models using Python classes, allowing us to interact with the database using Python objects. This enhances the efficiency and maintainability of our project.

Here's an example of how to set up a table with the name *'users'*, and columns for *id*, *username*, and *email*:

```
# in models.py: NEED TO REWRITE THIS CODE #
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
class User(db.Model):
  __tablename__='users'

  id = db.Column(db.Integer, primary_key=True)
  username = db.Column(db.String(80), unique=True, nullable=Fals
  email = db.Column(db.String(120), unique=True, nullable=False)

# In app.py
from flask import Flask, jsonify
from models import db

app = Flask(__name__)
db.init_app(app)
```

Handling API Routes

In order to communicate between the frontend and backend, we'll set up API routes using Flask-RESTful. These routes will act as endpoints that our React frontend can call to request and send data to the server. By defining different HTTP methods (GET, POST, etc.) for each route, we can handle various types of requests and perform corresponding actions in the backend.

Here's an example of how to use *api.add_resource* to define API routes:

```
# In app.py
from flask_restful import Api, Resource
api = Api(app)
class UserResource(Resource):
  def get(self):
        users = [user.to_dict() for user in User.query.all()]

        return jsonify(users)

  def post(self):
        form_json = request.get_json()
        new_user = User(username=form_json['username'], email=fo

        db.session.add(new_user)
        db.session.commit()

        response = make_response(
            new_user.to_dict(),
            201
        )
        return response

api.add_resource(UserResource, '/api/users')
```

- Flask-RESTful allows us to define functions as get, post, patch, and delete requests, so we don't have to write custom routes for each of them. Using the data sent from the front end, the request type determines which function will fire. Since a post request involves data from the client, we need to retrieve that data using another Flask-RESTful method, called *request.get_json()*, which extracts values from the client request.
- It is standard convention to create routes using the *'/api/'* prefix — more on this a little later. *

Business in the Front

Setting up Front-End Navigation

React's front-end capabilities come into play here. We can use React Router to handle navigation on the client-side. In your React component, you can use the Link component to create navigation links:

```
// In your React component
import React from 'react'
import { Link } from 'react-router-dom'

function NavigationBar(){
  return (
    <nav>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/about">About</Link></li>
        <li><Link to="/contact">Contact</Link></li>
      </ul>
    </nav>
  )
}

export default NavigationBar
```

- Clicking on any of these links will result in a redirect to its respective url path, causing the component which belongs to that path to render.

Party in the Back

Connecting Front-End React Routes to Flask Backend

With our React front-end and Flask backend set up, we need to connect them to work together seamlessly. We can do this by using *@app.route('/')* in Flask to serve the React build files directly:

```
# In app.py
from flask import Flask, render_template

@app.route('/')
def index():
# Render the React app built files
  return render_template('index.html')
```

In this code snippet, we have a Flask route defined using the *@app.route('/')* decorator. This decorator tells Flask that whenever a request is made to the root URL of the application (i.e., the domain name or IP address without any additional paths), the function *index()* should be executed.

Here's a breakdown of what everything really means under the hood:

- *from flask import Flask, render_template*: We import the necessary modules from Flask. Flask is the core Flask class used to create the application, and render_template is a function used to render HTML templates.
- *@app.route('/')*: This is a decorator that associates the function below it (index()) with a specific URL route. In this case, the route is '/', which represents the root URL of the application.
- *def index():* : This is a function called index(). When a request is made to the root URL of the application, this function will be executed.
- *return render_template('index.html')*: Inside the *index()* function, we use *render_template()* to render an HTML template called *'index.html'*. This means that when a user accesses the root URL of the application, Flask will return the content of the *'index.html'* template as the response.

What is render_template()?

render_template() is a Flask function that takes the name of an HTML template file as an argument and returns the HTML content of that template. In this example, render_template*('index.html')* will look for an *'index.html'* file in the templates folder and render its content. The *'index.html'* file is typically the entry point of your React front-end application, which will then be loaded and displayed in the user's web browser.

Adding the rest of your react routes

```
@app.route('/')
@app.route("/about")
@app.route("/contact")
def index(id=0):
    return render_template("index.html")
```

In order for all of your react routes to work from your backend server, you will need to add an *'@app.route()'* for each of your front end routes. When a user visits the root URL *(/),* Flask renders the index.html template and sends it as the response. Similarly, when the user visits */about* or */contact*, Flask renders the respective templates and serves them accordingly.

Note: Remember when I said it is proper convention to use *'/api/'* on any routes that you define as api use? The reason why this is particularly important here is due to the fact that when we bring in our React front end, for which we've already defined routes, if any of those routes overlap with our back-end routes, an error will occur since the routes are being over written! By using *'/api/'* as a prefix, you prevent that from occurring since the routes are different, and it keeps your codebase organized, too!

This setup is common in projects where Flask serves as the backend and React serves as the frontend. By connecting the root URL to the React entry point, you can seamlessly integrate the frontend and backend into a single application. This way, users can access your React application by simply navigating to the root URL of your Flask server, and there's no need to run a separate React development server. Everything is served from the Flask server itself.

Despite your front-end being hosted on your Flask Server, for deployment, you will still need to deploy your backend databases separately from your backend server that hosts your front end.

Deployment

Once our web application is ready, we'll need to deploy it to make it accessible to users worldwide. There are various deployment options available, like Heroku and Netlify, that offer straightforward deployment processes that streamline the deployment of both frontend and backend components. For a step-by-step deployment guide using Render, check out this step-by-step check-list provided by FlatIron School!

By now we've covered the essentials of integrating Flask and React together to build a cohesive web application. By understanding how to structure the project, set up API routes, handle front-end navigation, and connect both technologies, you can create dynamic, efficient, and user-friendly applications with ease. These steps are all that are between you and creating powerful full-stack web applications! Remember to follow best practices and explore the vast potential of Flask and React to build captivating and feature-rich applications that amaze users worldwide.

Happy coding!

Flask Python React Full Stack Deployment

S

Follow

Written by Sofia Katzman

1 Follower

More from Sofia Katzman

```
 _____ _____ _____ _____
|     |   __|   | |  |  |
| | | |   __| | | |  |  |
|_|_|_|_____|_|___|_____|


What module would you like to enter?
--------------------------------------
    1 : STAFF

    2 : POSITIONS

    3 : EVENTS

    4 : REPORTS

    5 : EXIT
```

Sofia Katzman

The Power of Organization: Building a Comprehensive Command-Line Interface

The Power of Organization:

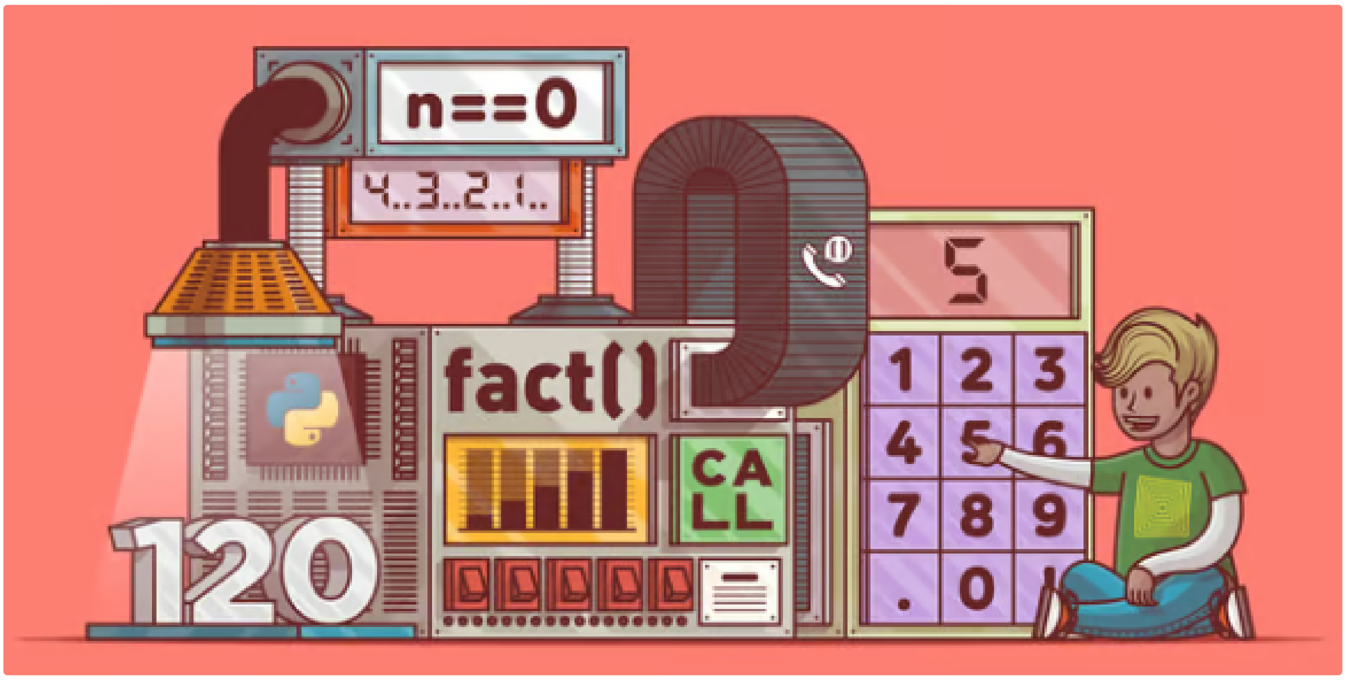
9 min read · Jun 13, 2023

Sofia Katzman

Be the MASTER of the DOM

DOM manipulation is something that seems very daunting at first but once you understand what is occurring under the hood you realize that...

2 min read · Apr 25, 2023



Sofia Katzman

Understanding Recursion as an Error: Pitfalls and Solutions in Python

Recursion is a powerful programming technique that allows functions to call themselves, offering an elegant approach to problem-solving in...

4 min read · Jul 19, 2023

Sofia Katzman

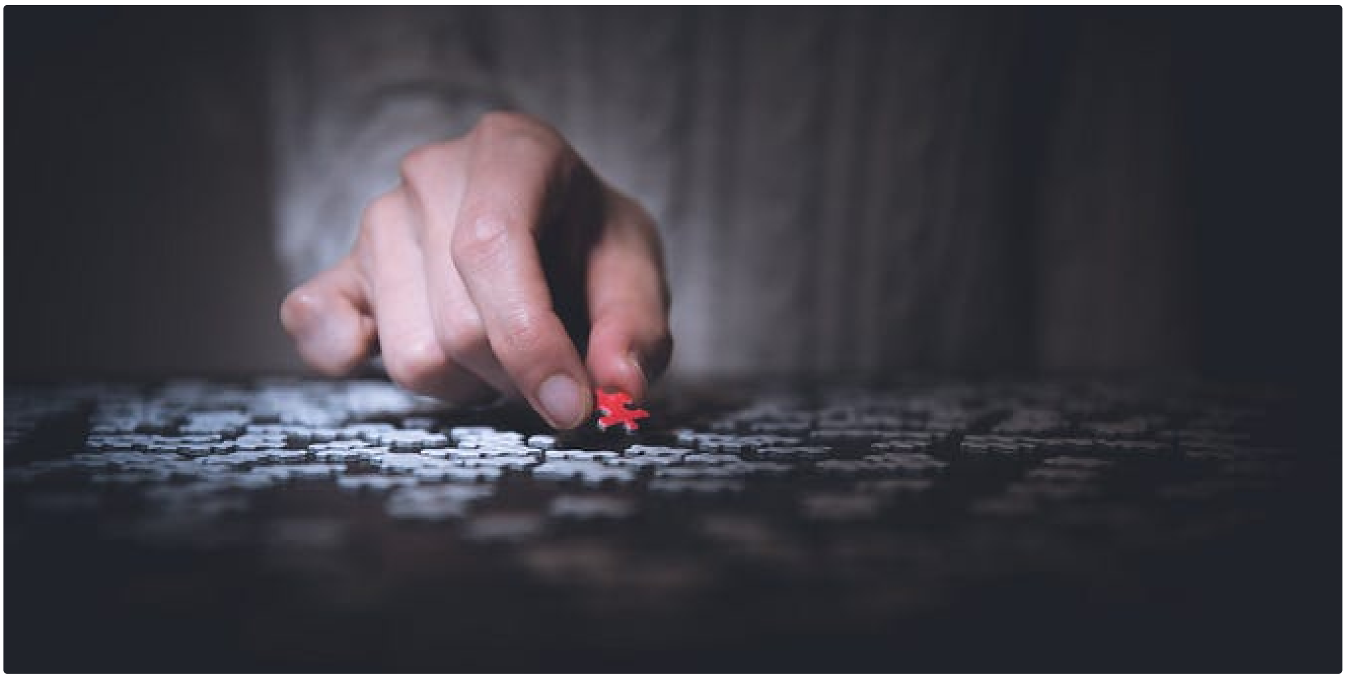
Event Listeners? Can you hear me?

An event listener is a function that is triggered when a certain event occurs, such as a button click, mouse movement, or key press.

2 min read · Apr 25, 2023

See all from Sofia Katzman

Recommended from Medium



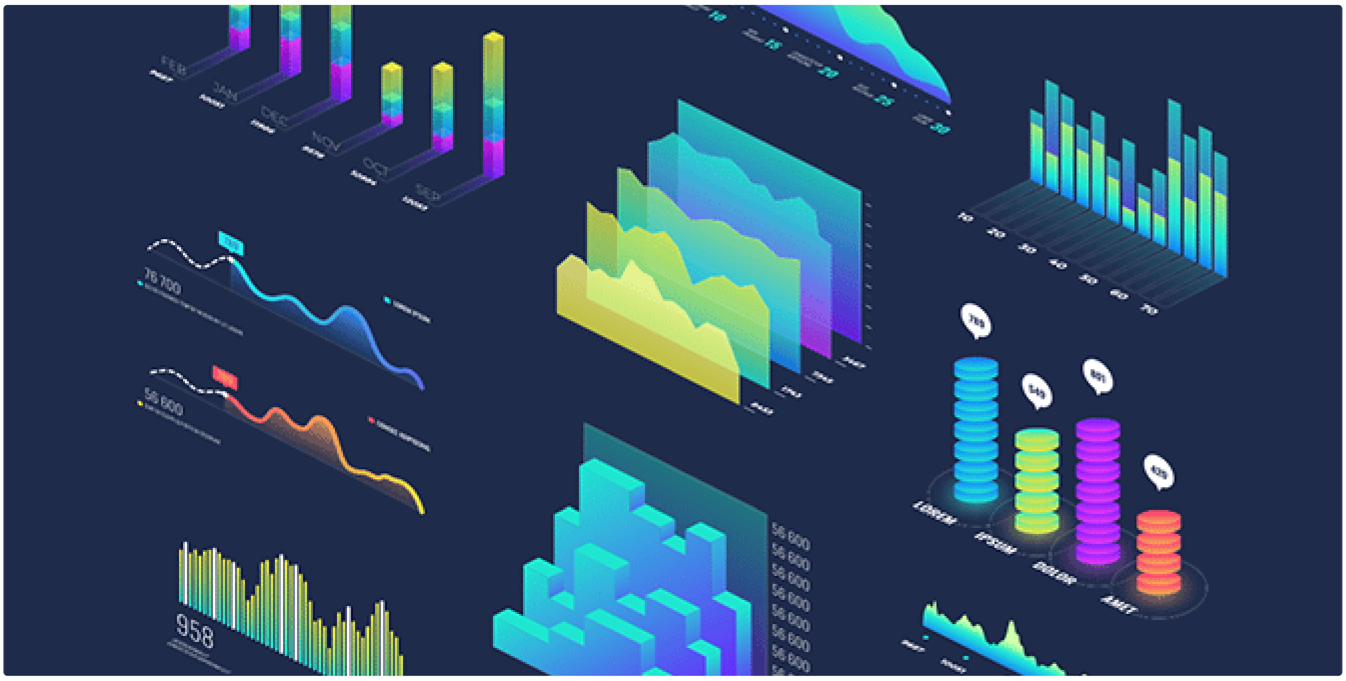
Mehdi Mohammadi

Building and Deploying an End-to-End Machine Learning Application with Flask, Angular and Docker

Introduction

9 min read · Nov 1, 2023

54



Lotfi Habbiche in Stackademic

Enhancing a Real-Time Data Visualization Dashboard with React.js and Python

Integrating React.js with Python for a real-time data visualization dashboard offers a robust, efficient, and scalable solution. React's...

✦ · 4 min read · Nov 23, 2023

84

Lists

Coding & Development
11 stories · 384 saves

Predictive Modeling w/ Python
20 stories · 791 saves

Practical Guides to Machine Learning
10 stories · 915 saves

ChatGPT
23 stories · 395 saves

Kanaecoder

Creating a Full Stack Website with React Frontend and Python Flask Backend

Step 1: Setting up the Flask Backend

3 min read · Sep 22, 2023

3

Eylem Aytas

Building a Full-Stack Website with React and Flask: A Beginner's Guide

If you're new to web development, building a full-stack website may sound intimidating, but don't worry! I'll try my best to explain...

3 min read · Jul 19, 2023

10

Sandyjtech

Building a Full Stack App with Flask, React, MySQL

Creating a full-stack app is a daunting task and I remember feeling overwhelmed and lost during my first attempt. However, after putting in...

6 min read · Sep 2, 2023

21

Mariia Ingersoll

Creating RESTful APIs with Flask

REST, or Representational State Transfer, serve as the bridge between different applications, allowing them to communicate and exchange…

4 min read · Aug 30, 2023

1

See more recommendations