

You have **2 free member-only stories left** this month. [Sign up](#) for Medium and get an extra one.

◆ Member-only story

How to Build an App with React + Flask

Concise and Simple



Ran (Reine) · Follow

Published in The Startup

4 min read · Dec 26, 2020

Listen

Share

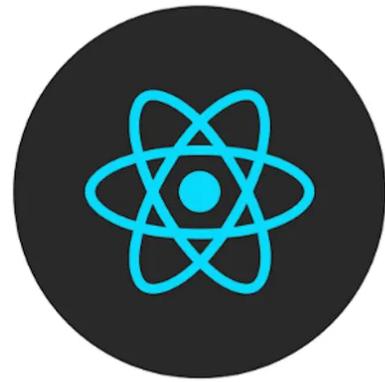


Image from [link](#)

Note: If you'd like to read about building + deploying your React-Flask project, do check out [this tutorial](#).

I mostly did AI related stuff back in my undergrad years but haven't had the chance to go into web development until my final semester of school. Even though AI was really interesting, I figured that the only way I can show others what I'm doing with it is to put it on the web.

With that thought, I learned Flask and React separately and these days I wanted to figure out how to use both Flask in conjunction with React so I can create applications with better looking interfaces.

The problem is, the first few tutorials I looked at were way too complicated (like, I just wanna make an API call from React front-end to Flask back-end that's it, do I really have to run eject, use another UI framework, etc?).

After a few tries, here's one that (I think) is really concise (probably will only take you 10 mins to complete).

[Open in app](#)

[Sign up](#)

[Sign In](#)



Search Medium



First things first, here's the GitHub Repo for easy reference — [link](#).

Back-end

```

react-flask-app - Visual Studio Code

File Edit Selection View Go Run Terminal Help
EXPLORER
> OPEN EDITORS
< REACT-FLASK-APP
  < backend / reactFlask
    > bin
    > include
    > lib
    > lib64
    > share
    > pyvenv.cfg
    > frontend
OUTPUT TERMINAL PROBLEMS DEBUG CONSOLE
1: bash
ran@ran-desktop:~/Documents/codes/react-flask-app$ mkdir frontend
ran@ran-desktop:~/Documents/codes/react-flask-app$ mkdir backend
ran@ran-desktop:~/Documents/codes/react-flask-app$ cd backend
ran@ran-desktop:~/Documents/codes/react-flask-app/backend$ python -m venv reactFlask
ran@ran-desktop:~/Documents/codes/react-flask-app/backend$ source reactFlask/bin/activate
(reactFlask) ran@ran-desktop:~/Documents/codes/react-flask-app/backend$ 

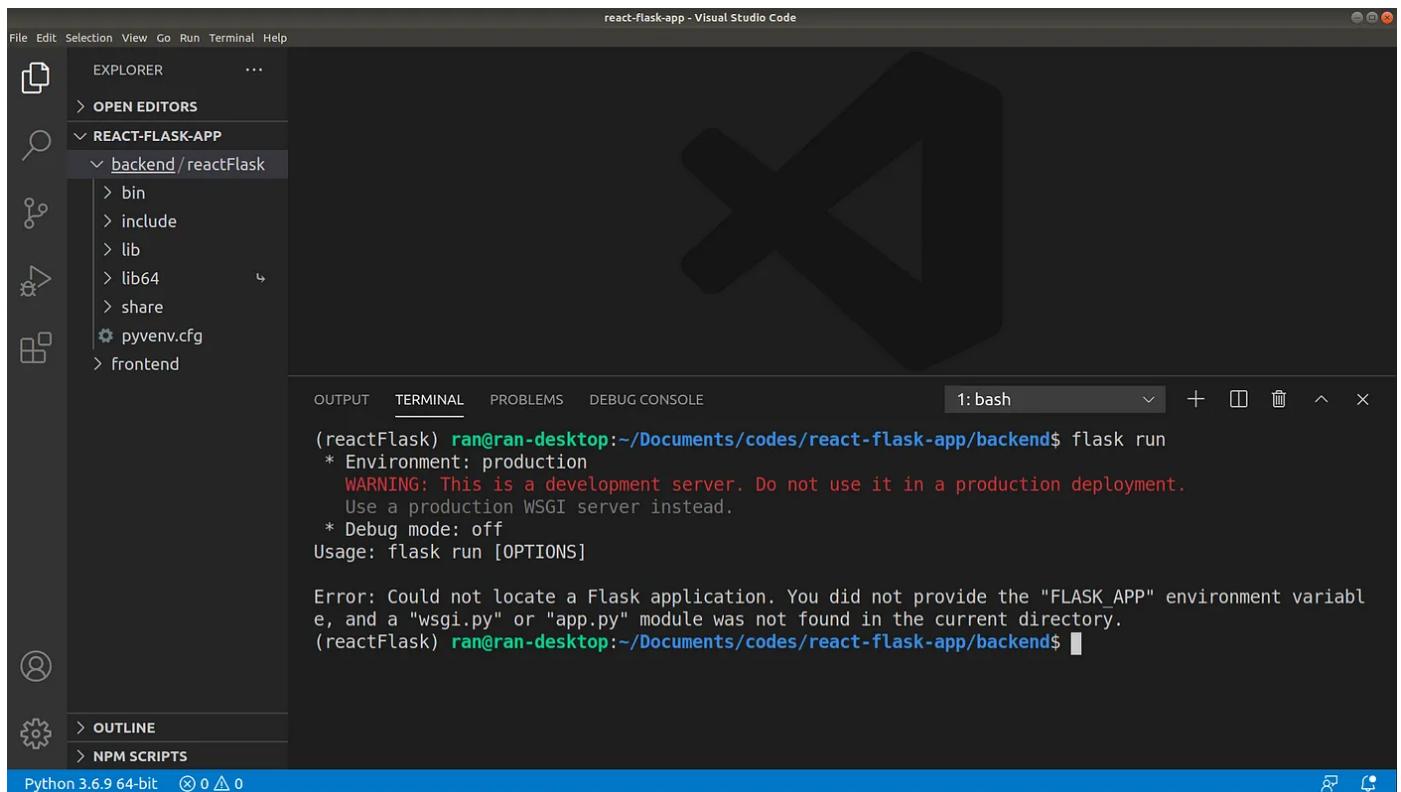
```

conda create -n reactFlask
 conda activate reactFlask
 pip install Flask
 pip install flask-restful

Project Set Up

We'll start off by creating two directories (one for back-end and one for front-end).

Although not strictly necessary, you're highly encouraged to use a virtual environment so that the versions of the python libraries and packages don't conflict with your other projects.



The screenshot shows a Visual Studio Code interface with a dark theme. The Explorer sidebar on the left shows a project structure under 'REACT-FLASK-APP': 'backend/reactFlask' (containing 'bin', 'include', 'lib', 'lib64', 'share', and 'pyvenv.cfg') and 'frontend'. The Terminal tab is active, displaying the command 'flask run' being run in a bash shell. The output shows a warning message: 'WARNING: This is a development server. Do not use it in a production deployment.' followed by usage instructions. Below this, an error message states: 'Error: Could not locate a Flask application. You did not provide the "FLASK_APP" environment variable, and a "wsgi.py" or "app.py" module was not found in the current directory.' The status bar at the bottom indicates Python 3.6.9 64-bit is in use.

```
(reactFlask) ran@ran-desktop:~/Documents/codes/react-flask-app/backend$ flask run
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
Usage: flask run [OPTIONS]

Error: Could not locate a Flask application. You did not provide the "FLASK_APP" environment variable, and a "wsgi.py" or "app.py" module was not found in the current directory.
(reactFlask) ran@ran-desktop:~/Documents/codes/react-flask-app/backend$
```

Here's what happens when you run flask without app.py

```
from flask import Flask
#An extension for Flask that adds support for quickly building REST APIs.
from flask_restful import Api, Resource, reqparse
app = Flask(__name__)
api = Api(app)
```

So we create a file `app.py` in the backend directory

We'll be using `flask_restful` (an extension for Flask that provides additional support for building REST APIs). `pip install flask-restful` (did above)

```
backend > core > HelloApiHandler.py
```

Writing our Hello Handler

In backend directory, create a core directory. This is where all our API handlers will reside in — we technically can just put this in the app.py file itself too but I'm just putting it here for a clean structure.

The screenshot shows a Visual Studio Code interface. The Explorer sidebar on the left displays a project structure under 'REACT-FLASK-APP'. It includes a 'backend' folder containing 'core' and 'HelloApiHandler.py'. The 'HelloApiHandler.py' file is open in the editor, showing Python code for a Flask API endpoint. The terminal at the bottom shows the command 'flask run' being executed, indicating the application is running on port 5000. The status bar at the bottom shows the file is 8 lines long, 2 spaces wide, in UTF-8 encoding, and is a Python file.

```
from flask_restful import Api, Resource, reqparse
class HelloApiHandler(Resource):
    def get(self):
        return {
            'resultStatus': 'SUCCESS',
            'message': "Hello API Handler"
        }
```

```
(reactFlask) ran@ran-desktop:~/Documents/codes/react-flask-app/backend$ flask run
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [27/Dec/2020 11:39:57] "GET /flask/hello HTTP/1.1" 200 -
```

We'll just write a very simple return when this API endpoint is called

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', 'Help', and various icons like '+ New', 'Import', 'Runner', 'My Workspace', 'Invite', and 'Sign In'. Below the navigation is a toolbar with several requests listed: 'POST r...', 'POST A...', 'GET log...', 'POST h...', 'GET IPO...', 'GET htt...', 'GET loc...', and 'GET loc...'. To the right of the toolbar is a dropdown for 'No Environment' and some other icons. The main workspace is titled 'Untitled Request' and shows a 'GET' request to 'localhost:5000/flask/hello'. Below the request details, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', 'Settings', 'Cookies', and 'Code'. Under 'Params', there's a table for 'Query Params' with one row: 'Key' (Value) and 'Description' (Description). The 'Body' tab is selected, showing a JSON response with four numbered lines: 1. {, 2. "resultStatus": "SUCCESS",, 3. "message": "Hello API Handler",, 4. } . Above the body content, status information is displayed: Status: 200 OK, Time: 21 ms, Size: 205 B, and Save Response. Below the body content, the text 'It works!' is displayed.

Okay now let's write something for a POST request in the same [HelloAPIHandler.py file](#)

```
1 def post(self):
2     parser = reqparse.RequestParser()
3     parser.add_argument('type', type=str)
4     parser.add_argument('message', type=str)
5
6     args = parser.parse_args()
7
8     #note, the post req from frontend needs to match the strings here (e.g. 'type' and 'mes
9
10    request_type = args['type']
11    request_json = args['message']
12    # process request_type and request_json here
13    # currently just returning directly for demo purposes
14    ret_status = request_type
15    ret_msg = request_json
16
17    if ret_msg:
18        message = "Your Message Requested: {}".format(ret_msg)
19    else:
20        message = "No Msg"
21
22    final_ret = {"resultStatus": "Success", "message": message}
23
24    return final_ret
```

HelloApiHandler.py hosted with ❤ by GitHub

[view raw](#)

Postman interface showing a successful POST request to `localhost:5000/flask/hello?type=posting&message=testing posting`. The response status is 200 OK, and the response body is:

```

1  {
2      "resultStatus": "Success",
3      "message": "Your Message Requested: testing posting"
4  }

```

Works for POST too! :)

Okay so now we're all done with the back-end codes!

Front-end

Visual Studio Code interface showing the `HelloApiHandler.py` file in the `app.py` editor. The code defines a `post` method that uses a `RequestParser` to parse arguments `'type'` and `'message'`. The terminal shows the command to run `npx create-react-app .`

```

def post(self):
    parser = reqparse.RequestParser()
    parser.add_argument('type', type=str)
    parser.add_argument('message', type=str)

    args = parser.parse_args()

    # note, the post req from frontend needs to match the strings here (e.g. 'typ
    request_type = args['type']
    request_json = args['message']

    # process request_type and request_json here
    # currently just returning directly for demo purposes
    ret_status = request_type

```

```
run create-react-app
```

Since we're using React, let's start by creating a React app. In order not to over-complicate this tutorial, we'll just display our GET and POST messages in the App.js page.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure under "REACT-FLASK-APP" with "backend" and "frontend" folders. Inside "frontend", there are "node_modules", "public", "src", "App.css", "App.js", "App.test.js", "index.css", "index.js", and "logo.svg".
- Code Editors:** "app.py", "HelloApiHandler.py", "App.js" (selected), and "package.json".
- Terminal:** Shows the command "npm install http-proxy-middleware@1.0.6" being run, followed by audit results: "added 2 packages from 6 contributors, removed 8 packages, updated 1 package and audited 1929 packages in 8.946s", "123 packages are looking for funding", and "found 0 vulnerabilities".
- Status Bar:** Shows "master*", "Python 3.6.9 64-bit", "Ln 1, Col 1", "Spaces: 2", "UTF-8", "LF", "JSON", "Prettier", and icons for file operations.

Install [http-proxy-middleware](#)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure for "REACT-FLASK-APP". The "src" folder contains "backend", "frontend", "node_modules", "public", and "src" subfolders. Inside "src", there are files: "# App.css", "App.js", "App.test.js", "# index.css", "index.js", "logo.svg", "reportWebVitals.js", and "setupProxy.js".
- Editor (Top Right):** The file "setupProxy.js" is open, displaying code to set up proxy middleware for a Flask application.
- Terminal (Bottom):** The terminal shows the command "npm start" being run, which outputs the message "You can now view frontend in the browser." It also provides local and network URLs for viewing the application. A note at the bottom says "Note that the development build is not optimized. To create a production build, use npm run build."

setupProxy.js

Create the setupProxy.js file in the src directory. What this piece of code above means is that all API calls from the front-end will be made to <http://localhost:5000/flask/{something}>.

The screenshot shows the Visual Studio Code interface with the following details:

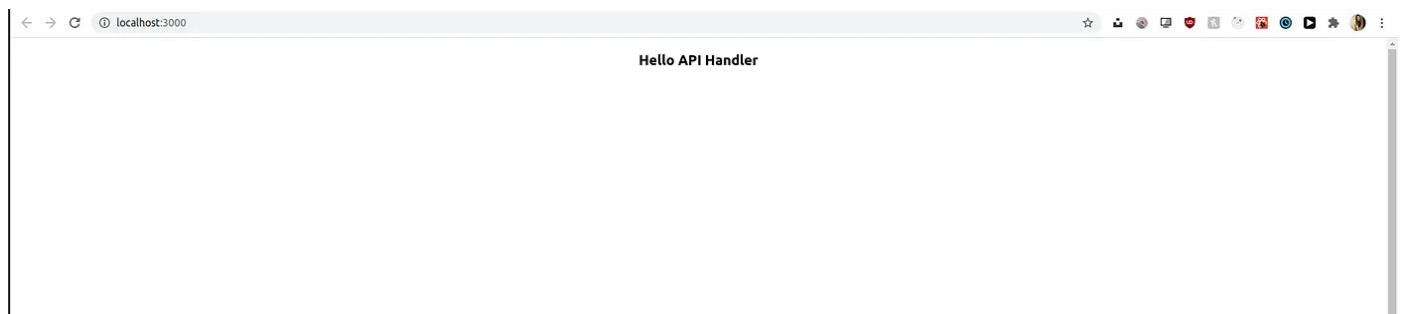
- File Explorer (Left):** Shows the project structure under "REACT-FLASK-APP". The "api" folder contains "ConnectServerGet.js", which is currently selected.
- Editor (Center):** Displays the code for "ConnectServerGet.js". The code defines a function "GetRestRequest" that uses the fetch API to make a GET request to " apiUrl ". It includes headers for Content-Type and Accept, and a promise chain to check the status and parse the JSON response.
- Terminal (Bottom):** Shows the output "You can now view frontend in the browser." followed by the local host URL "http://localhost:3000".
- Status Bar (Bottom):** Shows the current branch "master*", Python version "Python 3.6.9 64-bit", and file paths.

Get Function — click this to get the codes

This is one way of writing the GET request. Alternatively, you can also just write a simple axios GET request. If you're building a larger application, place this inside your Redux actions.

```
App.js - react-flask-app - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER ...
OPEN EDITORS
REACT-FLASK-APP
> backend
> frontend
  > node_modules
  > public
  > src
    > api
      JS ConnectServerGet.js
      # App.css
      JS App.js
      JS App.test.js
      # index.css
      JS index.js
      logo.svg
      JS reportWebVitals.js
      JS setupProxy.js
      JS setupTests.js
      .eslintcache
      .gitignore
      { } package-lock.json
      OUTLINE
      TIMELINE
      NPM SCRIPTS
JS setupProxy.js JS App.js X JS ConnectServerGet.js
frontend > src > JS App.js > App
1 import './App.css';
2 import React, { useEffect, useState } from 'react';
3
4 import GetRestObject from "../src/api/ConnectServerGet"
5
6 function App() {
7   const [getMessage, setGetMessage] = useState("")
8
9   useEffect(()=>{
10     GetRestObject.GetRestRequest('/flask/hello', getResultObj => {
11       console.log(getResultObj)
12       setGetMessage(getResultObj.message)
13     })
14   }, [])
15
16   return (
17     <div className="App">
18       <h3>{getMessage}</h3>
19     </div>
20   );
21 }
22
23 export default App;
24
```

Call the GET request function in a `useEffect` (with square bracket) so it gets called once `onLoad`



It works! ❤

As for the POST request function,

```

function PostRestRequest(apiUrl, postData, postResultObj) {
  var data = JSON.stringify(postData)
  return fetch(apiUrl, {
    mode: 'cors',
    method: 'POST',
    body: data,
    json: true,
    headers: new Headers({
      'Content-Type': 'application/json',
      'Accept': "application/json"
    })
  })
  .then(checkStatus)
  .then(parseJSON)
  .then(postResultObj);
}

function checkStatus(response) {
  if (response.status >= 200 && response.status < 300) {
    return response;
  }
  const error = new Error(`HTTP Error ${response.statusText}`);
  error.status = response.statusText;
  error.response = response;
  console.log(error);
  throw error;
}

```

Full codes please see — [link](#)

```

const [postMessage, setPostMessage] = useState("")
useEffect(()=>{
  GetRestObject.GetRestRequest('/flask/hello', getResultObj => {
    console.log(getResultObj)
    setGetMessage(getResultObj.message)
  })
  const postData = {
    type: "Api post req",
    message: "my message"
  }
  PostRestObject.PostRestRequest('/flask/hello', postData, postResultObj => {
    console.log(postResultObj)
    setPostMessage(postResultObj.message)
  })
}, [])
return (

```

Note that the development build is not optimized.
To create a production build, use `npm run build`.

We'll test with these two hard-coded messages

A screenshot of a web browser window. The address bar shows 'localhost:3000'. The main content area displays the text 'Hello API Handler' and 'Your Message Requested: my message'. Below this, a message from the author is displayed.

It returns it correctly! ❤️

Note that it's returning the same message we sent it because I assigned it that way (`ret_msg = args['message']`) in the back-end. I'm doing this just for demo purposes and if you're writing an actual application, remember to process the requests first before returning it :)

That's all for today's article! Thank you for reading this and please feel free to reach out by commenting or messaging me if you run into any issues/have any questions or comments! Referenced from — [link](#).

Happy ReFlasking! ♪♪♪?♪❤️

React

Python

Programming

Software Development

Flask



Follow



Written by Ran (Reine)

433 Followers · Writer for The Startup

I live for days when I can watch skies of blue, while enjoying the view. Most other days I'm a city rat who scuttles between Art and Coding. SG NTU CS Grad.

More from Ran (Reine) and The Startup



 Ran (Reine) in Towards Data Science

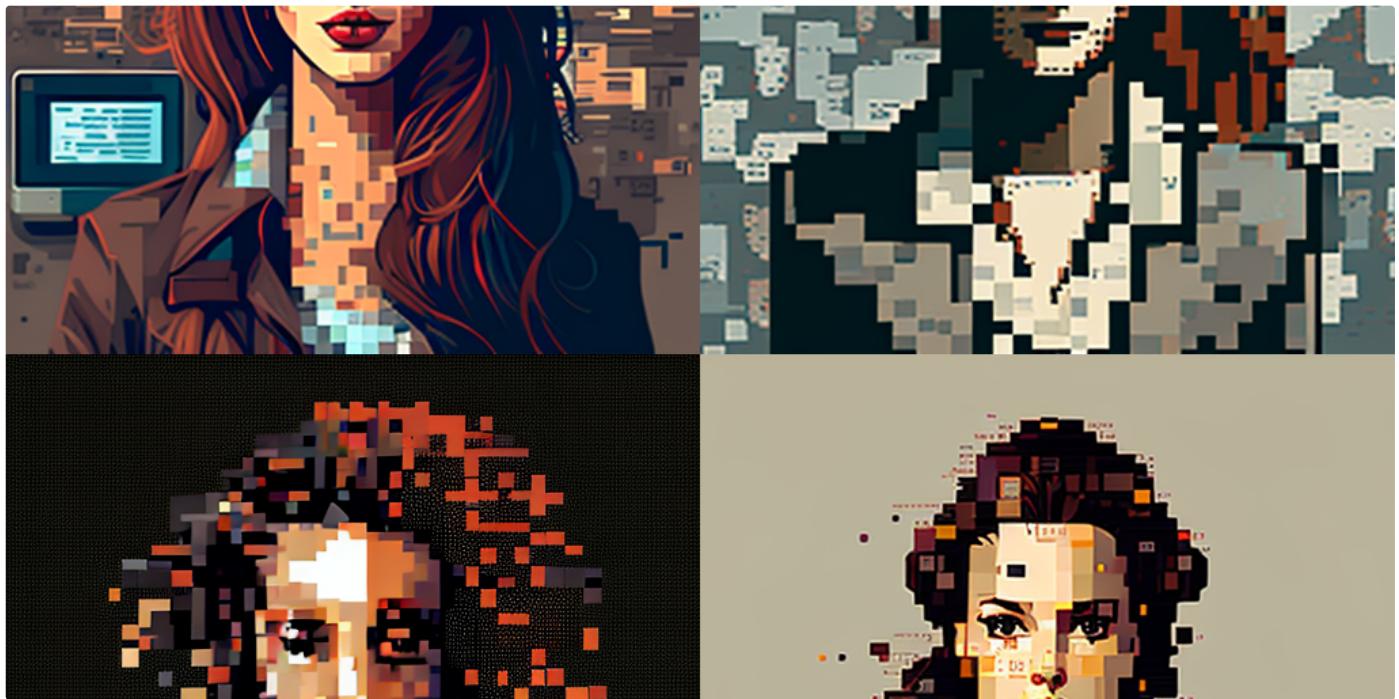
Build & Deploy a React + Flask App

Probably the most simple and concise guide

★ · 5 min read · Jan 15, 2021

 405  14





 Zulie Rane in The Startup

If You Want to Be a Creator, Delete All (But Two) Social Media Platforms

In October 2022, during the whole Elon Musk debacle, I finally deleted Twitter from my phone. Around the same time, I also logged out of...

◆ · 8 min read · Apr 18

 25K  502

+





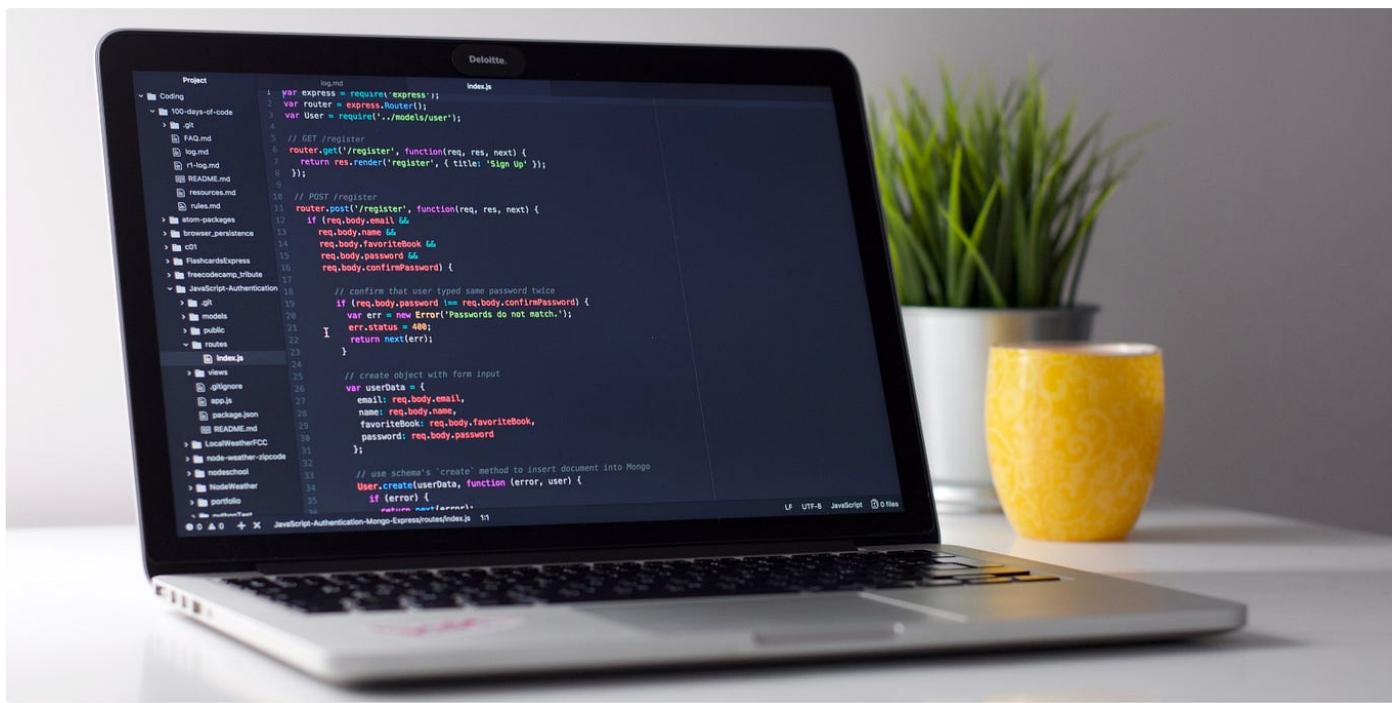
Nitin Sharma in The Startup

Goodbye ChatGPT: These (New) AI Tools Will Leave You Speechless

I bet that 99% of the readers are not familiar with any of these tools.

◆ · 7 min read · May 6

2.6K 41



Ran (Reine) in Analytics Vidhya

How to install OpenCV.js

(With detailed instructions for compiling from source)

◆ · 3 min read · Mar 15, 2020

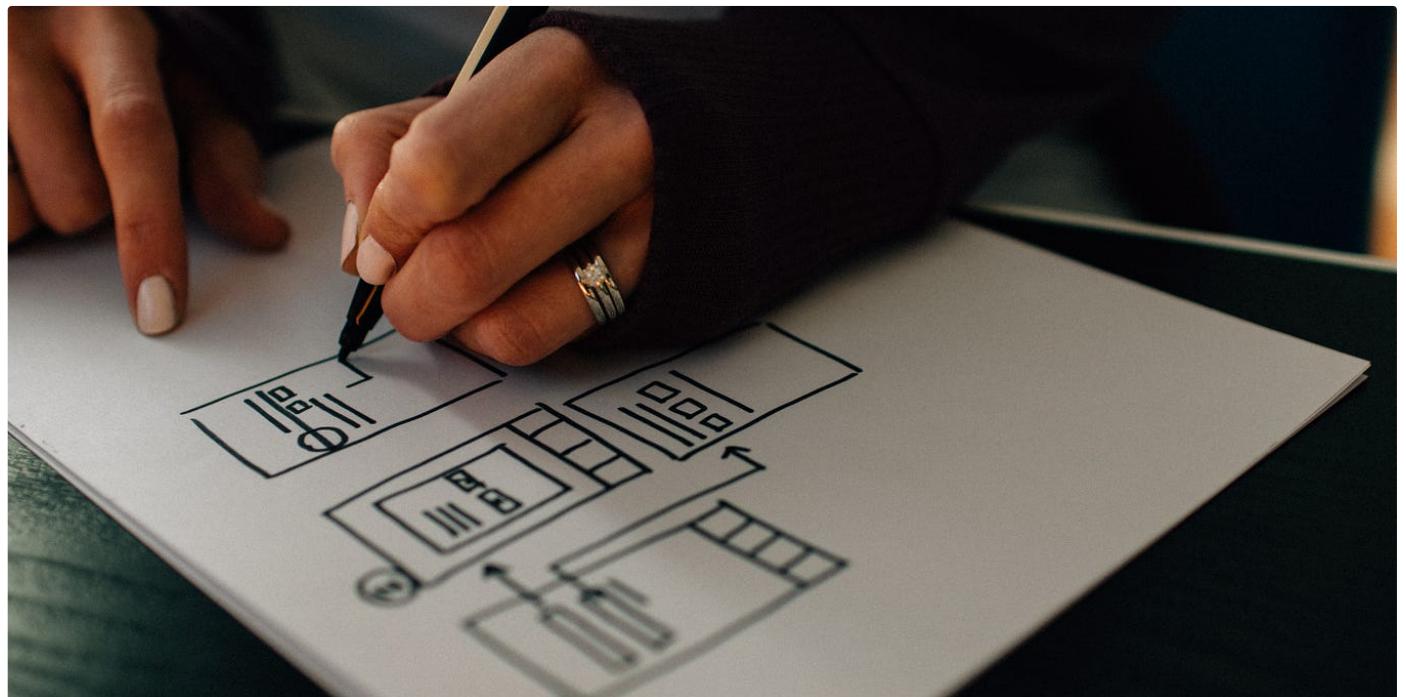
53 4



See all from Ran (Reine)

See all from The Startup

Recommended from Medium



Audhi Aprilliant in Geek Culture

Part 2—End to End Machine Learning Model Deployment Using Flask

How to build a user interface for loan approval prediction app and set up the backend using flask

★ · 10 min read · Nov 30, 2022

178

1

+

Python Firebase Authentication Integration with FastAPI



 Yujian Tang in Plain Simple Software

Create an API with User Management using FastAPI and Firebase

A quick start guide to creating an API with user authentication

◆ · 12 min read · Dec 13, 2022

 103



Lists



Stories to Help You Grow as a Software Developer

19 stories · 70 saves



Leadership

30 stories · 29 saves



Good Product Thinking

11 stories · 75 saves



Stories to Help You Level-Up at Work

19 stories · 53 saves



 Kumar Shubham in Towards Data Science

Build a Blog Website using Django Rest Framework—Users App (Part 2)

In the second part, we will deal with building user-related models and views and will test the user-related APIs.

◆ · 15 min read · Dec 14, 2022

 211 







Lynn Kwong in Towards Data Science

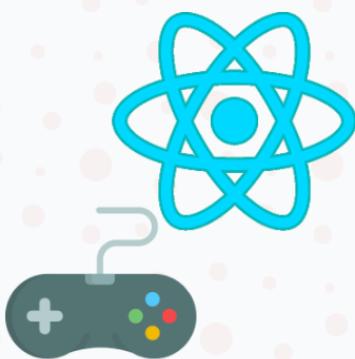
Build a WebSocket Application with FastAPI and Angular

Learn to build a two-way interactive communication application with the WebSocket protocol

◆ · 8 min read · Jan 30



77



7 React Projects for Beginners in 2023

Here are seven unique and fun React projects for you to make, all of which will teach you essential React concepts that you need to know in 2023.



Reed Barger in Web Dev Hero

7 React Projects for Beginners in 2023 (+ Code)

You're ready to start making simple projects with React, but you don't know what to make. Where should you start?

◆ · 6 min read · Jan 11



269



5



```
5 from django.core.exceptions import FieldError, ValidationError
6 from django.db import connections
7 from django.db.models.expressions import Exists, ExpressionList, F, OrderBy
8 from django.db.models.indexes import IndexExpression
9 from django.db.models.lookups import Exact
10 from django.db.models.query_utils import Q
11 from django.db.models.sql.query import Query
12 from django.db.utils import DEFAULT_DB_ALIAS
13 from django.utils.deprecation import RemovedInDjango60Warning
14 from django.utils.translation import gettext_lazy as _
15
16 __all__ = ["BaseConstraint", "Constraint", "IndexConstraint"]
17
18 class BaseConstraint:
19     defaultViolation_error_message = _("Constraint %(name)s is violated.")
20     violation_error_code = None
21     violation_error_message = None
22
23     def __init__(self, *args, name=None, violation_error_code=None, violation_error_message=None):
24         # RemovedInDjango60Warning.
25         if name is None and not args:
26             raise TypeError(
27                 f"{self.__class__.__name__}.__init__() missing 1 required keyword-only "
28                 f"argument: 'name'"
29             )
30         self.name = name
31         if violation_error_code is not None:
32             self.violation_error_code = violation_error_code
33         if violation_error_message is not None:
34             self.violation_error_message = violation_error_message
```

Django REST Framework



Viktor Nagornyy in Powered by Django

How To Create An API with Django REST Framework Quickly

Learn how to integrate Django Rest Framework API into your Django project with this beginner-friendly tutorial, including two example APIs.

◆ · 6 min read · Apr 4

👏 31

💬 1

+

See more recommendations