****

**S.P. Mandali’s**

**Ramnarain Ruia Autonomous College**

**Matunga, Mumbai-400019**

**Project Report on**

**AI Branding Slogan Generator**

**Project Guide**

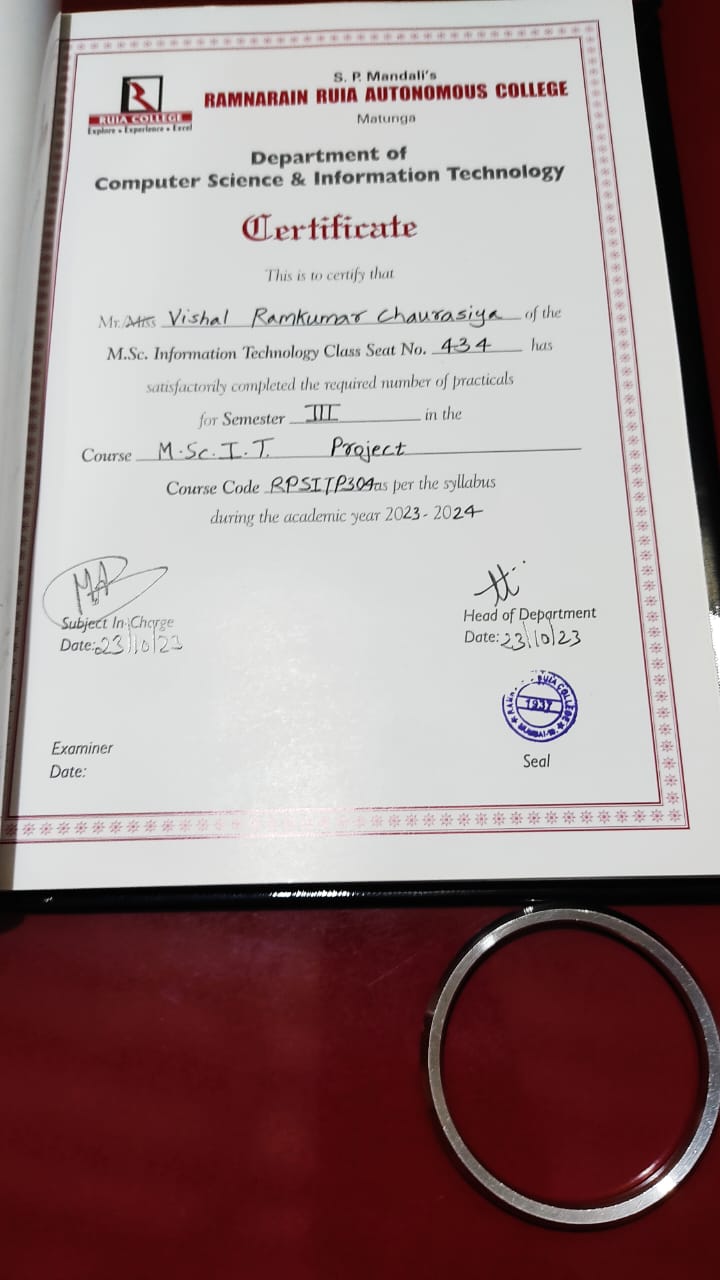
**Prof. Mahavir Advaya**

**Project By**

**Vishal Ramkumar Chaurasiya (434)**

**Department of Computer Science & Information Technology**

**2023-24**



ACKNOWLEDGEMENT

This document is prepared by Mr. Vishal Chaurasiya, student of M.Sc. IT Part-II Ramnarain Ruia Autonomous College Matunga, Mumbai.

I express sincere gratitude to Prof. Mahavir Advaya, HOD & Assistant Professor, Department of Computer Science, Ramnarain Ruia Autonomous College Matunga, Mumbai, as the mentor for my project. It is my great fortune that I got opportunity to carry out this project work under your supervision. I express my sincere thanks for the encouragement, support and the guidance. I would further like to thank the entire faculty member for their cooperation and extended support in undergoing the project work. I also would thank the department of Computer Science to provide the infrastructure and facility in carrying the projects like these.

Secondly I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

INDEX

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **PARTICULARS** | Signature |
| 1 | Abstract   * Problem statement |  |
| 2 | Introduction   * Detailed explanation of study area |  |
| 3 | Literature Review   * Research Paper Based |
| 4 | Research Methodology   * Project/Files Structure * Program Logics |  |
| 5 | Experimental setup   * Tools * Architecture/Framework * Software Language |  |
| 6 | Results   * Screen Layout * Testing report |  |
| 7 | Conclusion   * Findings of Project work * Future enhancements |  |
| 8 | References   * References (Conference Papers/ Journal Papers etc.) * Websites |
| 9 | Presentation of working project and allocating internal marks |  |
| 10. | Blackbook signing |  |

**AI Branding Slogan Generator**

**Abstract:**

Slogans are a key element of a brand's identity, and contribute to a brand's equity. In today's marketplace, almost all brands employ slogans; they enhance a brand's image, aid in its recognition and recall, and help create brand differentiation in consumers' minds. While there is general consensus on the importance of the device itself, little agreement exists as to what constitutes a successful slogan.

As such, although marketing managers use slogans extensively, they are often at a loss when it comes to creating them. In turn, this leads to ineffective use of slogans and, ultimately, the possibility of a surprisingly poor linkage between a brand and its slogan, even among the most well-known brands.

To solve this problem I have created AI based branding slogan generator. I this project user need to just time there work name and application will give you ‘slogan’ along with the ‘hashtags’

**Introduction:**

In the age of digital marketing and rapidly evolving consumer preferences, crafting a compelling and memorable branding slogan is of paramount importance for businesses seeking to establish their unique identity and connect with their target audience. To meet this demand and harness the power of artificial intelligence, we present our innovative project - the "AI-Based Branding Slogan Generator."

This project leverages cutting-edge technologies such as AWS Lambda, Python, Docker, Vercel, Node.js, and more to create a dynamic and intelligent system that generates brand slogans tailored to a company's vision, values, and mission. The traditional approach to branding slogans often involves extensive brainstorming sessions and subjective decision-making, making it a time-consuming and sometimes ineffective process. Our AI-based solution aims to streamline and enhance this critical aspect of brand identity.

In this project back-end is developed using Python programming language. Front-end is developed using NextJS. AWS cloud infrastructure is developed using TypeScript programming language.

AWS Lambda serves as the backbone of our AI system, providing a scalable and serverless computing platform to run our slogan generation algorithms. This ensures high availability and rapid response times, even as the system handles a growing number of requests.

The core of our AI model is built using Python, a versatile and widely-used programming language for machine learning and natural language processing. Through extensive training and data analysis, our Python-based system can generate creative and context-aware branding slogans.

Docker containers are used to encapsulate and deploy our AI model, making it highly portable and consistent across various environments. Dockerization ensures that our system can be easily deployed on diverse platforms and integrates seamlessly into the project's architecture.

Vercel, a cloud platform optimized for frontend and serverless deployment, is employed to create a user-friendly web interface for our branding slogan generator. This allows businesses and individuals to access the AI-driven slogan generator conveniently via the web.

Node.js is used to build the interactive frontend that communicates with our AI backend hosted on Vercel. It provides an intuitive user experience, allowing users to input their brand information and receive tailored branding slogans instantly.

OpenAI is an American artificial intelligence research laboratory consisting of the non-profit OpenAI, Inc. and its for-profit subsidiary corporation OpenAI, L.P. registered in Delaware.

GPT-3: GPT-3 is a large scale natural language processing system based on deep learning and developer by OpenAI. It is now generally available for developers and curious people.

GPT-3 is the kind of AI that works exclusively on text. Text is both input and output for GPT-3. One can provide questions and instructions in plain English and receive fairly coherent responses.

The engine is the sub-model of GPT-3. There are 4 base engines available (ordered from most to least powerful):

Davinci

Curie

Babbage

Ada

Using a more powerful engine cost more time and credits than using a less powerful one. Base engines are meant for general-purpose tasks.

Instruct engines are made to understand and follow instructions. At this point, there are 3 of them:

**Davinci-instruct-beta-v3**

Davinci-instruct-beta

Curie-instruct-beta

The objective of our project is to empower businesses and marketers with a powerful tool that simplifies and accelerates the process of crafting effective branding slogans. By harnessing the capabilities of AWS Lambda, Python, Docker, Vercel, Node.js, and AI, we aim to revolutionize how brands connect with their audiences, leaving a lasting impact in the competitive landscape of modern marketing. Our AI-Based Branding Slogan Generator represents the future of brand identity creation, combining the best of technology, creativity, and efficiency to help businesses stand out and thrive.

**Literature Review:**

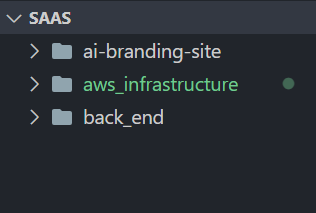
1. Research paper on use of AWS lambda function taken from researchgate website.

<https://www.researchgate.net/publication/338391132_Case_Study_Use_of_AWS_Lambda_for_Building_a_Serverless_Chat_Application>

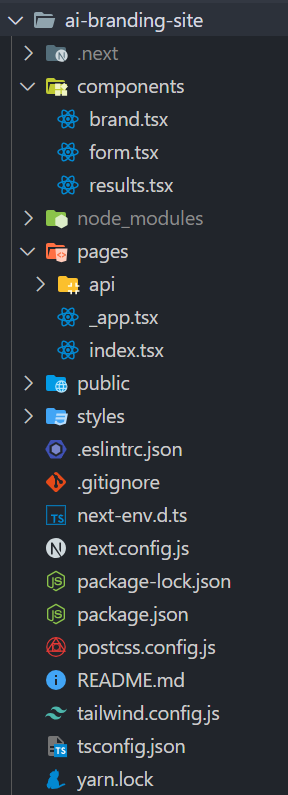
**Research Methodology:**

**Project/Files Structure:**

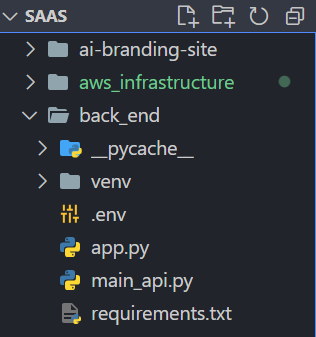
1. There are mainly 3 directory



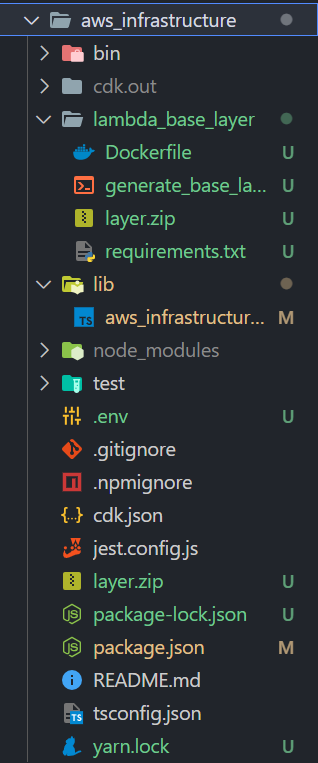
1. The “ai-brandin-site” directory contains front-end logic.



1. The “back-end” directory contain back-end logic, which is written in Python programming language. FastAPI is used to create API.



1. The “aws\_infrastructure” directory contains logic of deployments on AWS Lambda Function. Logic is written in TypeScript.



**Program Logics:**

1. Back-end logic:

import os

import openai

from typing import List

import argparse

import re

from dotenv import load\_dotenv

load\_dotenv()

MAX\_INPUT\_LENGTH = 32

def main():

parser = argparse.ArgumentParser()

parser.add\_argument("--input", "-i", type=str, required=True)

args = parser.parse\_args()

user\_input = args.input

print(f"User input: {user\_input}")

if validate\_length(user\_input):

generate\_branding\_snippet(user\_input)

generate\_keywords(user\_input)

else:

raise ValueError(

f"Input length is too long. Must be under {MAX\_INPUT\_LENGTH}. Submitted input is {user\_input}"

)

def validate\_length(prompt: str) -> bool:

return len(prompt) <= MAX\_INPUT\_LENGTH

def generate\_keywords(prompt: str) -> List[str]:

*# Load your API key from an environment variable or secret management service*

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

enriched\_prompt = f"Generate related branding keywords for {prompt}: "

print(enriched\_prompt)

'''You make an API call to generate text completion by calling openai.Completion.create(). This method sends a request to the OpenAI GPT-3.5 model, which generates a response based on the input parameters you provide.'''

response = openai.Completion.create(

engine="davinci-instruct-beta-v3", prompt=enriched\_prompt, max\_tokens=32

)

*# Extract output text.*

keywords\_text: str = response["choices"][0]["text"]

*# Strip whitespace.*

keywords\_text = keywords\_text.strip()

keywords\_array = re.split(",|\n|;|-", keywords\_text)

keywords\_array = [k.lower().strip() for k in keywords\_array]

keywords\_array = [k for k in keywords\_array if len(k) > 0]

print(f"Keywords: {keywords\_array}")

return keywords\_array

def generate\_branding\_snippet(prompt: str) -> str:

*# Load your API key from an environment variable or secret management service*

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

enriched\_prompt = f"Generate upbeat branding snippet for {prompt}: "

print(enriched\_prompt)

response = openai.Completion.create(

engine="davinci-instruct-beta-v3", prompt=enriched\_prompt, max\_tokens=32

)

*# Extract output text.*

branding\_text: str = response["choices"][0]["text"]

*# Strip whitespace.*

branding\_text = branding\_text.strip()

*# Add ... to truncated statements.*

last\_char = branding\_text[-1]

if last\_char not in {".", "!", "?"}:

branding\_text += "..."

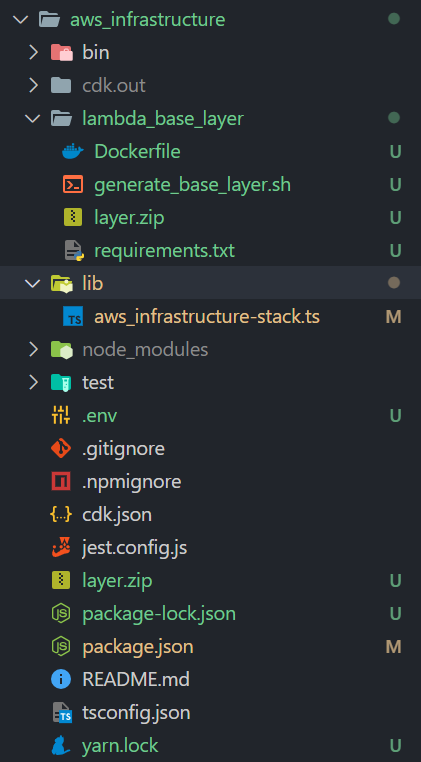
print(f"Snippet: {branding\_text}")

return branding\_text

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. AWS Cloud Infrastructure system:



2.1. The “aws\_infrastructure-stack.ts” file ?

→ This file is back-bone of the project’s deployment.

import { Stack, StackProps } from "aws-cdk-lib";

import { Construct } from "constructs";

import \* as lambda from "aws-cdk-lib/aws-lambda";

import \* as apiGateway from "aws-cdk-lib/aws-apigateway";

import \* as dotenv from "dotenv";

dotenv.config();

export class AwsInfrastructureStack extends Stack {

constructor(scope: Construct, id: string, props?: StackProps) {

super(scope, id, props);

const layer = new lambda.LayerVersion(this, "BaseLayer", {

code: lambda.Code.fromAsset("lambda\_base\_layer/layer.zip"),

compatibleRuntimes: [lambda.Runtime.PYTHON\_3\_9],

});

const apiLambda = new lambda.Function(this, "ApiFunction", {

runtime: lambda.Runtime.PYTHON\_3\_9,

code: lambda.Code.fromAsset("../back\_end/"),

handler: "main\_api.handler",

layers: [layer],

environment: {

OPENAI\_API\_KEY: process.env.OPENAI\_API\_KEY ?? "",

},

});

const mainApi = new apiGateway.RestApi(this, "RestApi", {

restApiName: "My API",

});

mainApi.root.addProxy({

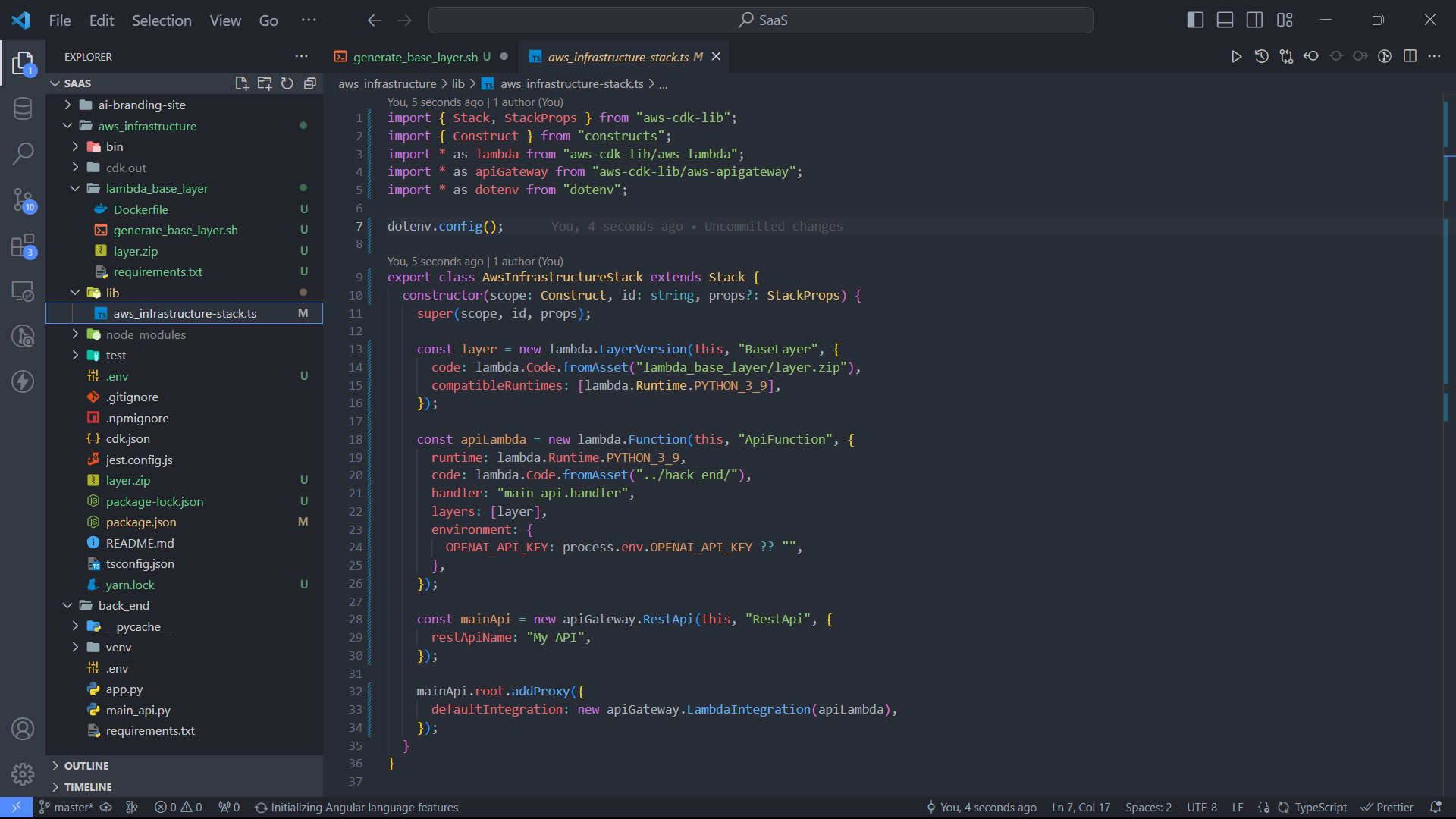
defaultIntegration: new apiGateway.LambdaIntegration(apiLambda),

});

}

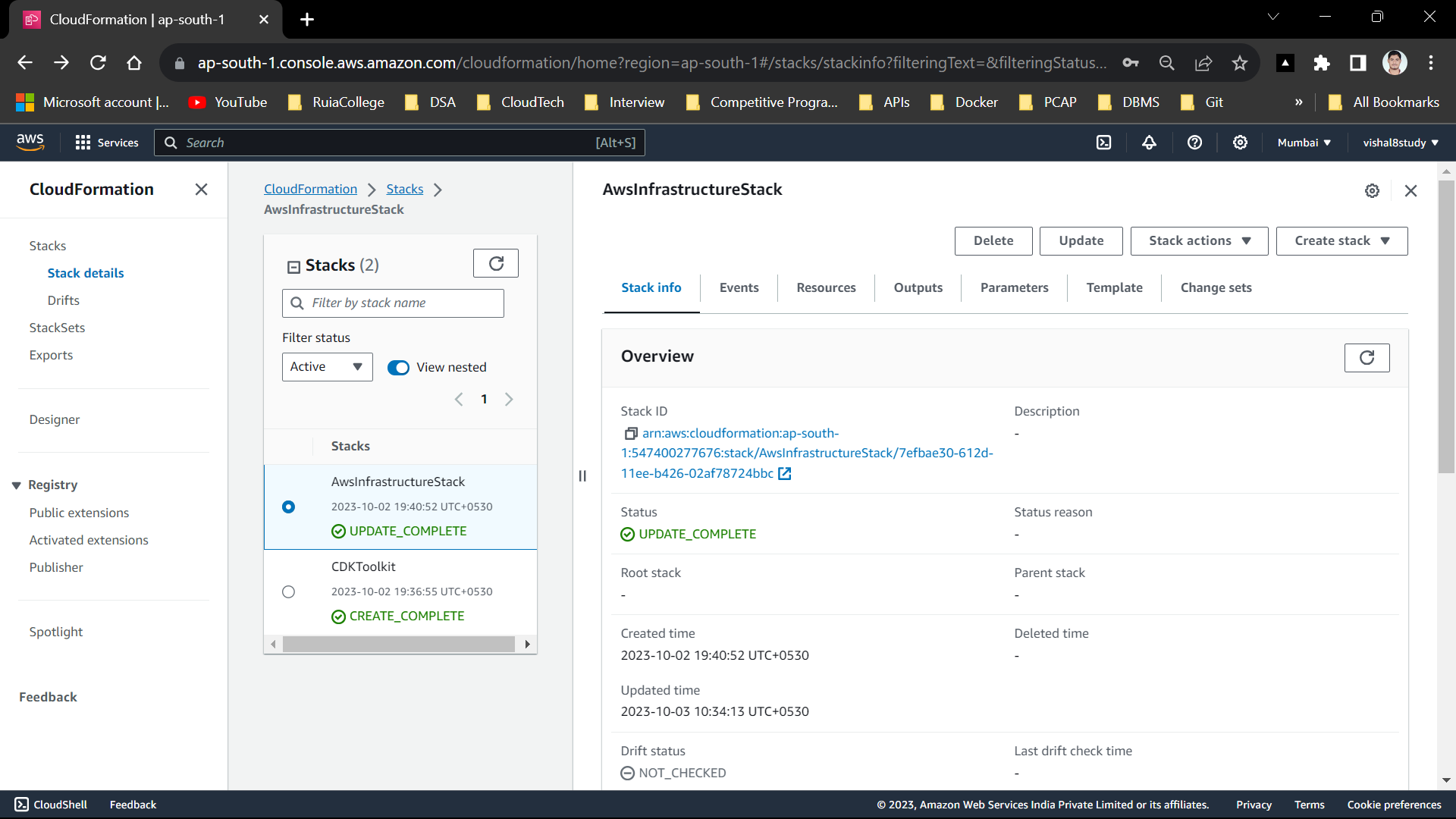
}

→ AWS’s Cloud Development Kit is used for deployment. The command for testing AWS Labda Function is “cdk bootstrap aws://5474-0027-7676/ap-south-1”.



→ In short, AWS CDK bootstrap process creates resources - S3 bucket (to host cloud assembly assets, application artificats), IAM roles/policies, SSM parameters etc. to facilitate deploying AWS resources using CDK.

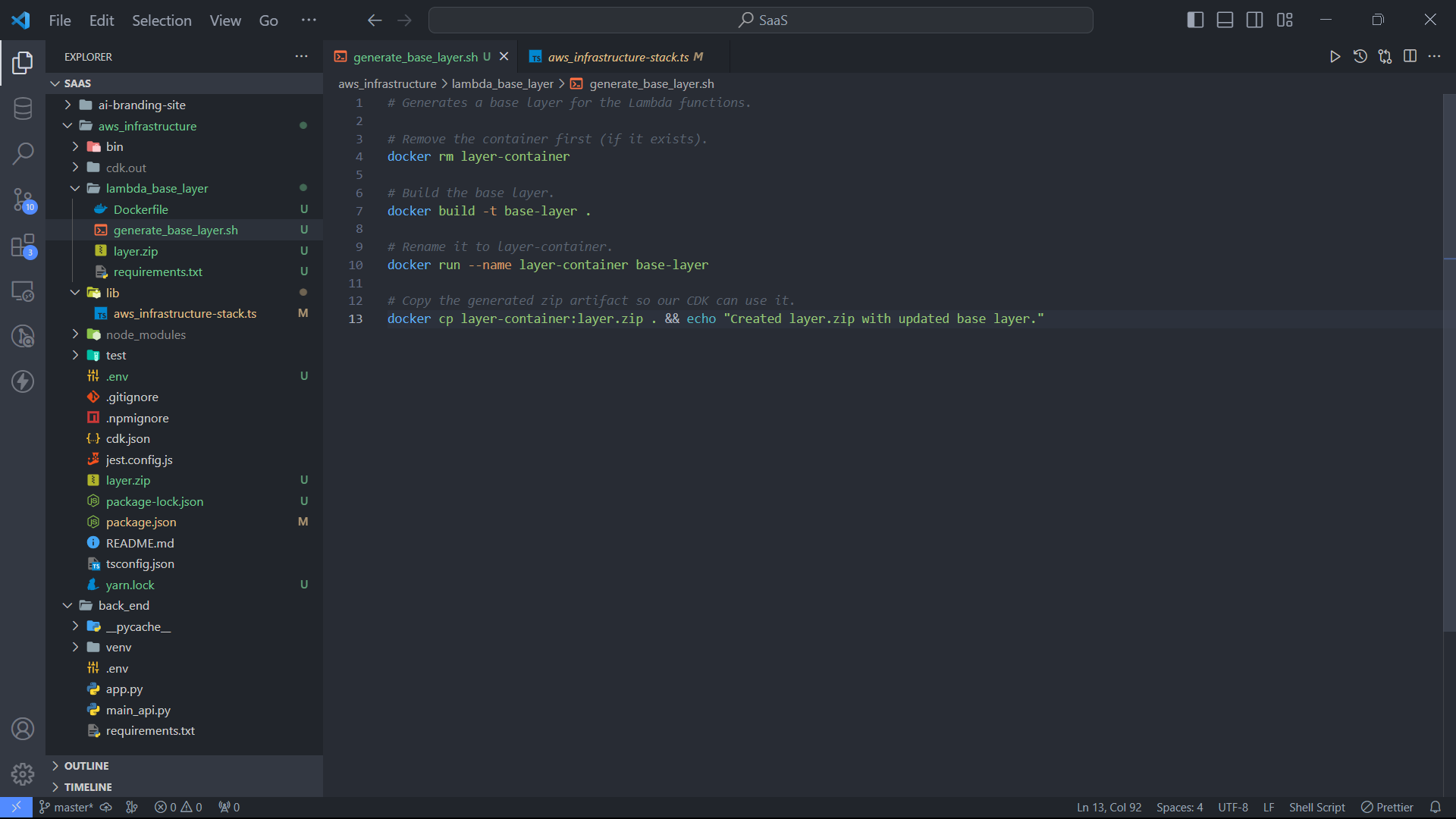
→ For deployment “cdk deploy”.



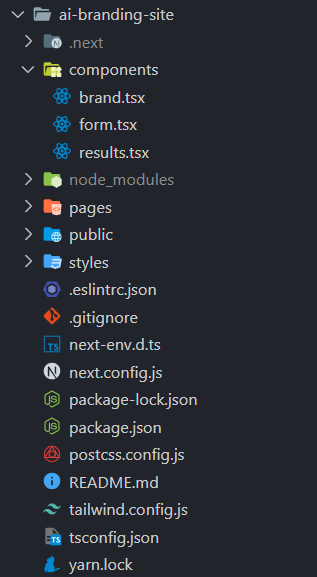
1. Docker container:

3.1. I have used docker for creating container, so that I can bunddle the entire back-end program.

3.2. After putting program into the docker container now it is ready to deploy.



3.3. The above program shows how to archive the containerization.

1. Front-end:
   1. 
   2. Components:
      1. brand.tsx: It contains main connection part for the AWS.

import React from "react";

import Form from "./form";

import Results from "./results";

import Image from "next/image";

import logo from "../public/BrandLogo.svg";

const Brand: React.FC = () => {

const CHARACTER\_LIMIT: number = 32;

const ENDPOINT: string =

"https://7xv7tnekh7.execute-api.ap-south-1.amazonaws.com/prod/generate\_snippet\_and\_keywords";

const [prompt, setPrompt] = React.useState("");

const [snippet, setSnippet] = React.useState("");

const [keywords, setKeywords] = React.useState([]);

const [hasResult, setHasResult] = React.useState(false);

const [isLoading, setIsLoading] = React.useState(false);

const onSubmit = () => {

console.log("Submitting: " + prompt);

setIsLoading(true);

fetch(`${ENDPOINT}?prompt=${prompt}`)

.then((res) => res.json())

.then(onResult);

};

const onResult = (data: any) => {

setSnippet(data.snippet);

setKeywords(data.keywords);

setHasResult(true);

setIsLoading(false);

};

const onReset = () => {

setPrompt("");

setHasResult(false);

setIsLoading(false);

};

let displayedElement = null;

if (hasResult) {

displayedElement = (

<Results

snippet={snippet}

keywords={keywords}

onBack={onReset}

prompt={prompt}

/>

);

} else {

displayedElement = (

<Form

prompt={prompt}

setPrompt={setPrompt}

onSubmit={onSubmit}

isLoading={isLoading}

characterLimit={CHARACTER\_LIMIT}

/>

);

}

const gradientTextStyle =

"text-white text-transparent bg-clip-text bg-gradient-to-r from-lime-400 to-orange-500 font-light w-fit mx-auto";

return (

<div className="h-screen flex">

<div className="max-w-md m-auto p-2">

<div className="bg-rose-900 p-9 rounded-md text-white">

<div className="text-center my-10">

<Image src={logo} width={42} height={42} />

<h1 className={gradientTextStyle + " text-3xl text-lime-300 font-light"}>

Branding Slogans

</h1>

<div className={gradientTextStyle}>Your AI branding assistant</div>

</div>

{displayedElement}

</div>

</div>

</div>

);

};

export default Brand;

* + 1. form.tsx: This file contains form and all. Including buttons and text area.

interface FormProps {

prompt: string;

setPrompt: any;

onSubmit: any;

isLoading: boolean;

characterLimit: number;

}

const Form: React.FC<FormProps> = (props) => {

const isPromptValid = props.prompt.length < props.characterLimit;

const updatePromptValue = (text: string) => {

if (text.length <= props.characterLimit) {

props.setPrompt(text);

}

};

let statusColor = "text-yellow-500";

let statusText = null;

if (!isPromptValid) {

statusColor = "text-red-400";

statusText = `Input must be less than ${props.characterLimit} characters.`;

}

return (

<>

<div className="mb-8 text-lime-300">

<p>

Tell me what your brand is about and I will generate slogan and keywords

for you.

</p>

</div>

<input

className="p-2 w-full rounded-md focus:outline-teal-400 focus:outline text-slate-700"

type="text"

placeholder="Enter your interest"

value={props.prompt}

onChange={(e) => updatePromptValue(e.currentTarget.value)}

></input>

<div className={statusColor + " flex justify-between my-2 mb-6 text-sm"}>

<div>{statusText}</div>

<div>

{props.prompt.length}/{props.characterLimit}

</div>

</div>

<button

className="bg-gradient-to-r from-teal-400

to-amber-500 disabled:opacity-50 w-full p-2 rounded-md text-lg"

onClick={props.onSubmit}

disabled={props.isLoading || !isPromptValid}

>

Submit

</button>

</>

);

};

export default Form;

* + 1. result.tsx: This file contains final result/response part. After user click the button response will be generated and it will be shown to the user.

interface ResultsProps {

prompt: string;

snippet: string;

keywords: string[];

onBack: any;

}

const Results: React.FC<ResultsProps> = (props) => {

const keywordElements = [];

for (let i = 0; i < props.keywords.length; i++) {

const element = (

<div

key={i}

className="bg-teal-200 p-1 text-teal-700 px-2 text-sm rounded-md"

>

#{props.keywords[i]}

</div>

);

keywordElements.push(element);

}

const keywordElementsHolder = (

<div className="flex flex-wrap gap-2">{keywordElements}</div>

);

const resultSection = (label: string, body: any) => {

return (

<div className="bg-rose-700 p-4 my-3 rounded-md">

<div className="text-slate-400 text-sm font-bold mb-4">{label}</div>

<div>{body}</div>

</div>

);

};

return (

<>

<div className="mb-6">

{resultSection(

"Prompt",

<div className="text-lg font-bold">{props.prompt}</div>

)}

{resultSection("Branding Snippet", props.snippet)}

{resultSection("Keywords", keywordElementsHolder)}

</div>

<button

className="bg-gradient-to-r from-teal-400

to-amber-500 disabled:opacity-50 w-full p-2 rounded-md text-lg"

onClick={props.onBack}

>

Back

</button>

</>

);

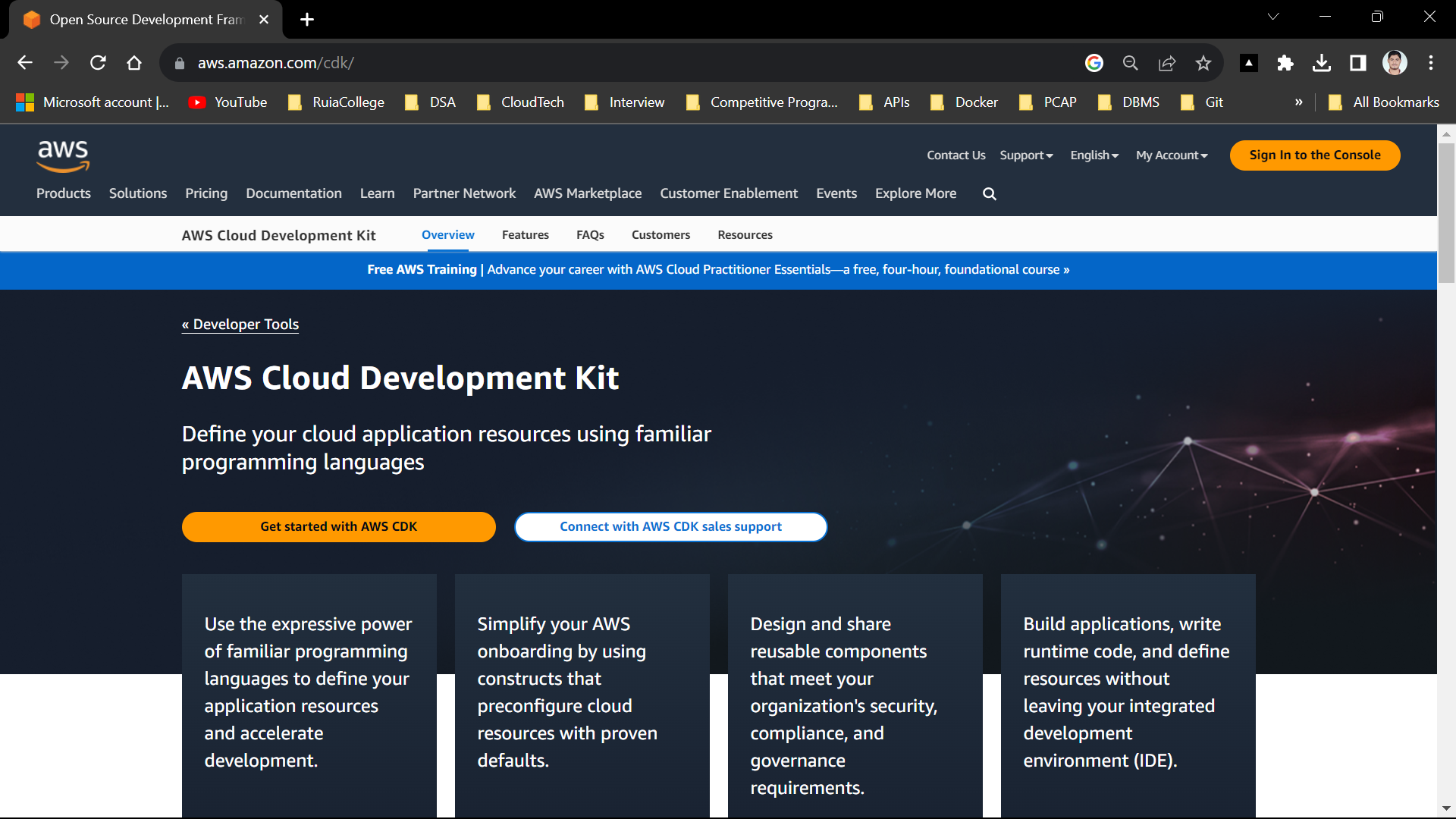
};

export default Results;

**Experimental setup:**

**Tools:**

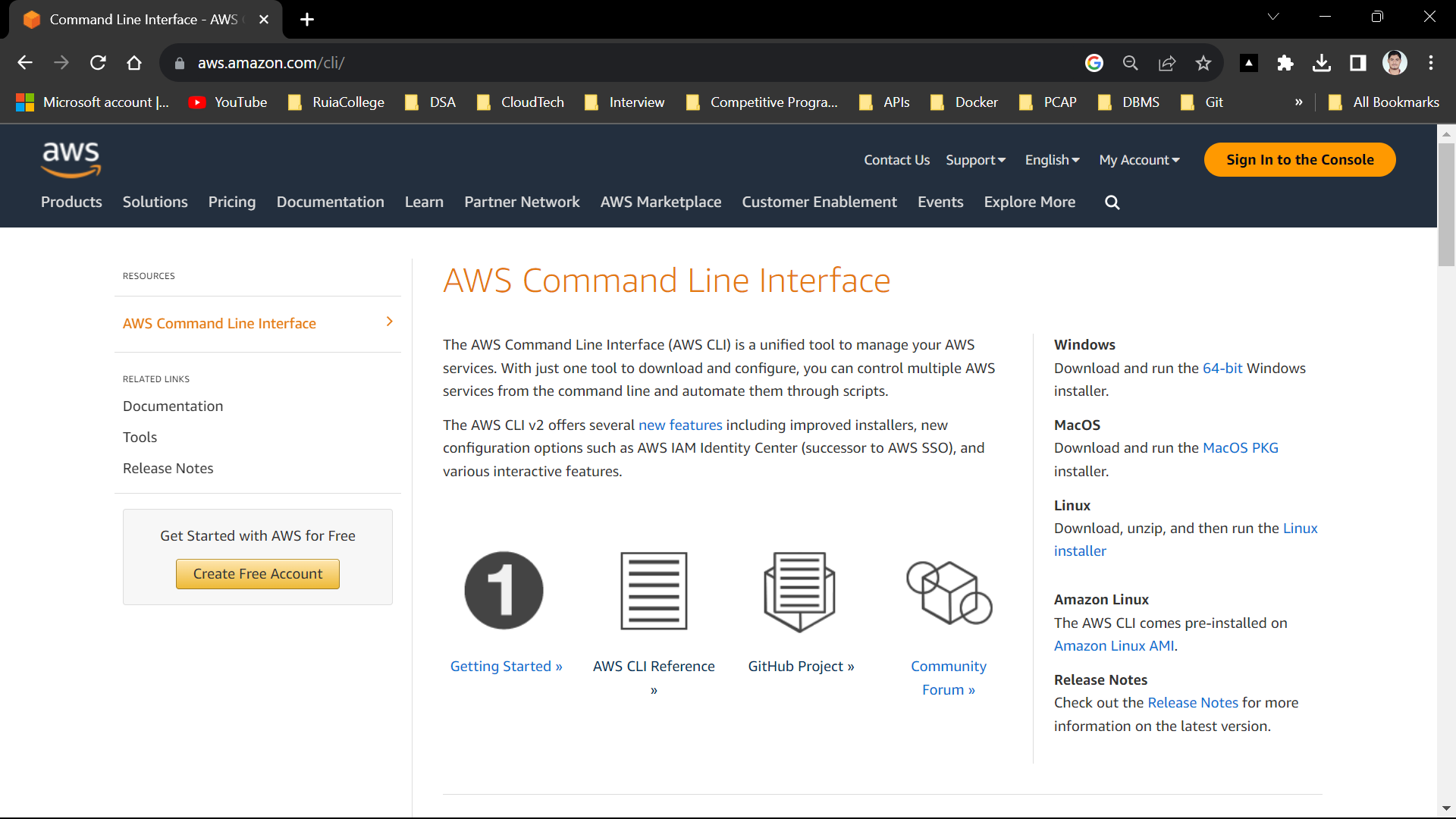
**CDK:**

****

The AWS Cloud Development Kit is an open-source software development framework developed by Amazon Web Services for defining and provisioning cloud infrastructure resources using familiar programming languages.

AWS CDK is an infrastructure-as-code tool that allows developers to define AWS resources in code. On the other hand, AWS SDK, or Software Development Kit, provides a set of tools for developers to interact with AWS services, such as creating API calls and Lambda functions.

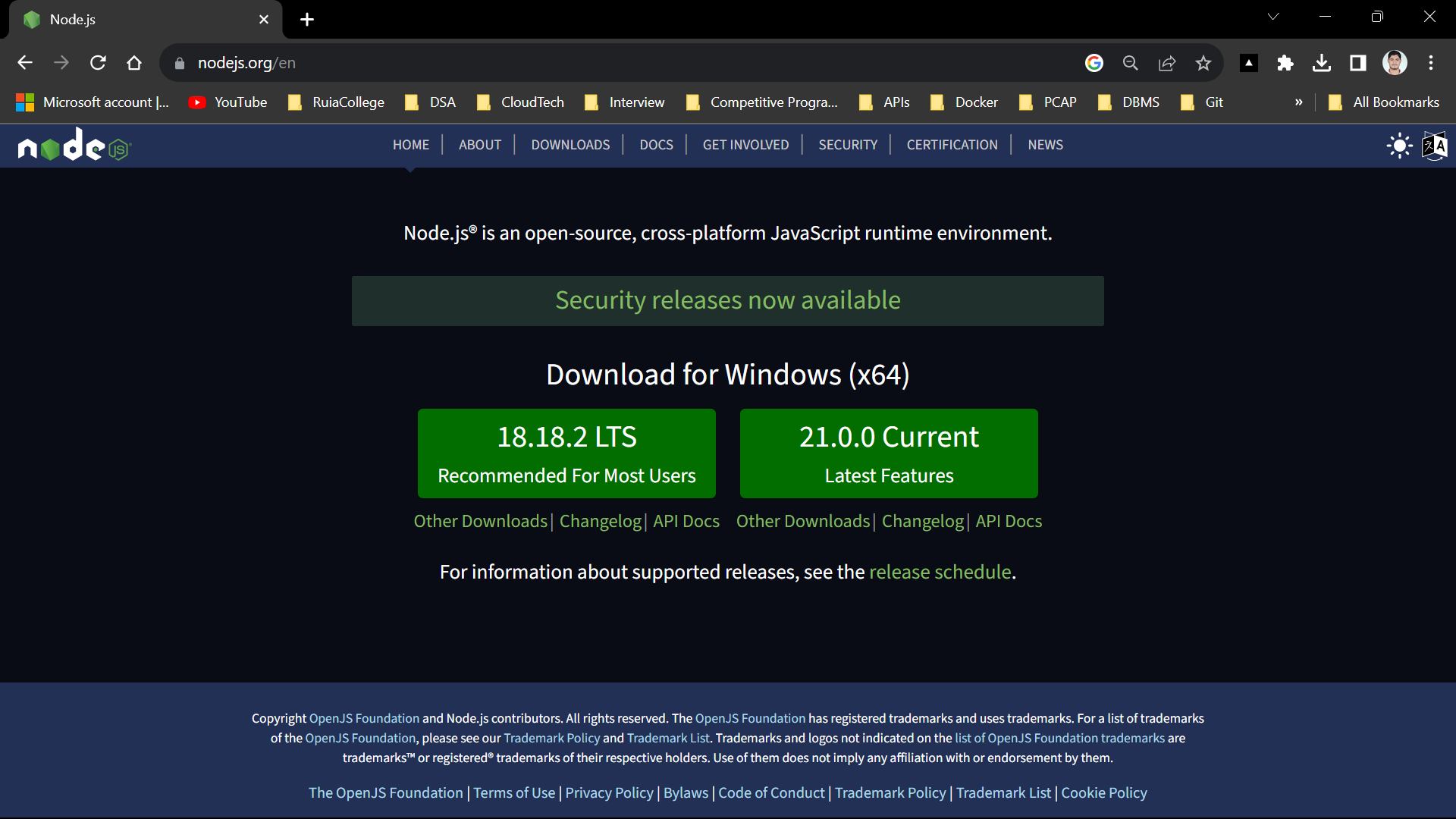
**AWS CLI:**

****

The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

**Architecture/Framework:**

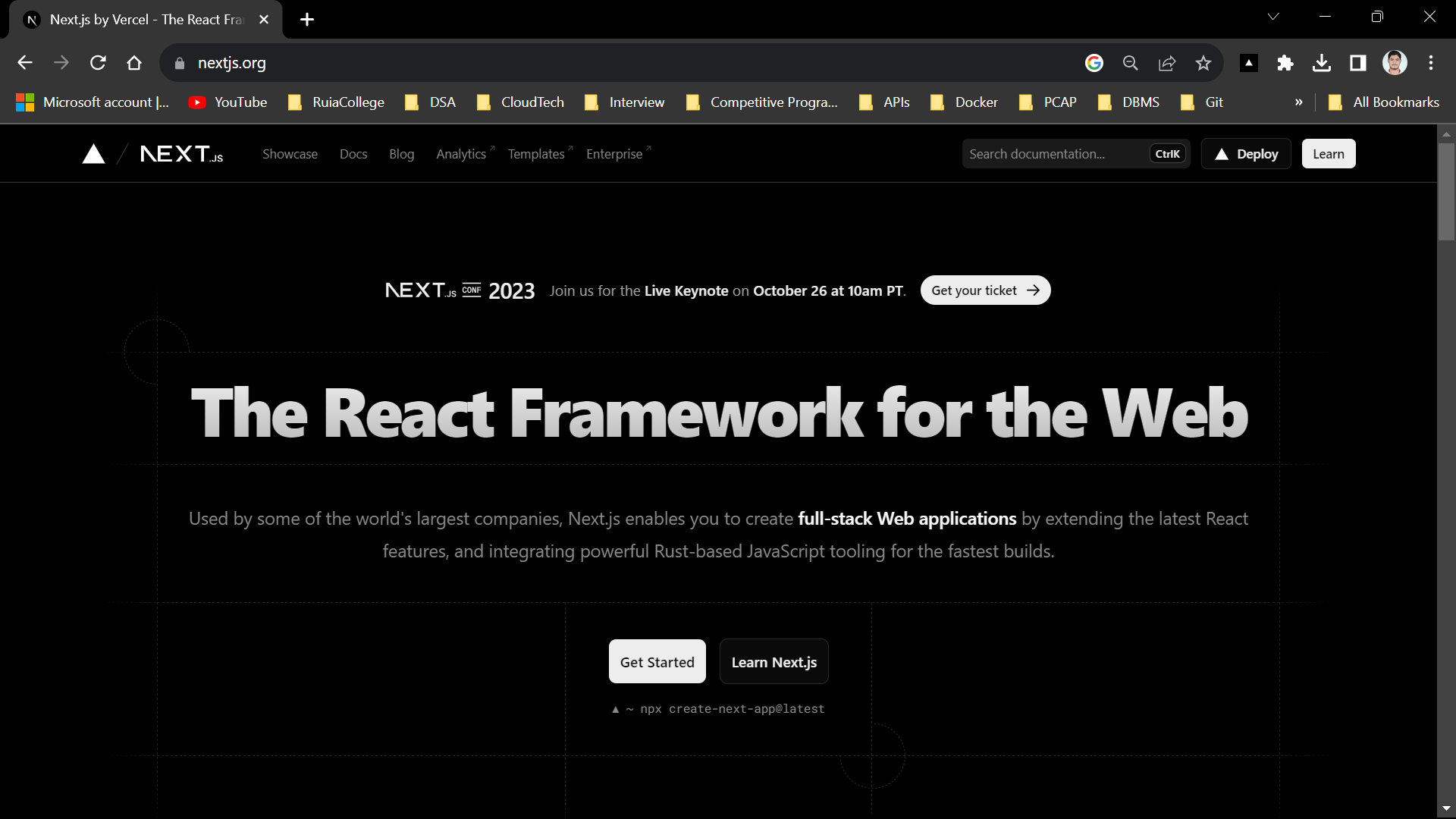
**NodeJS:**

****

Node.js is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more.

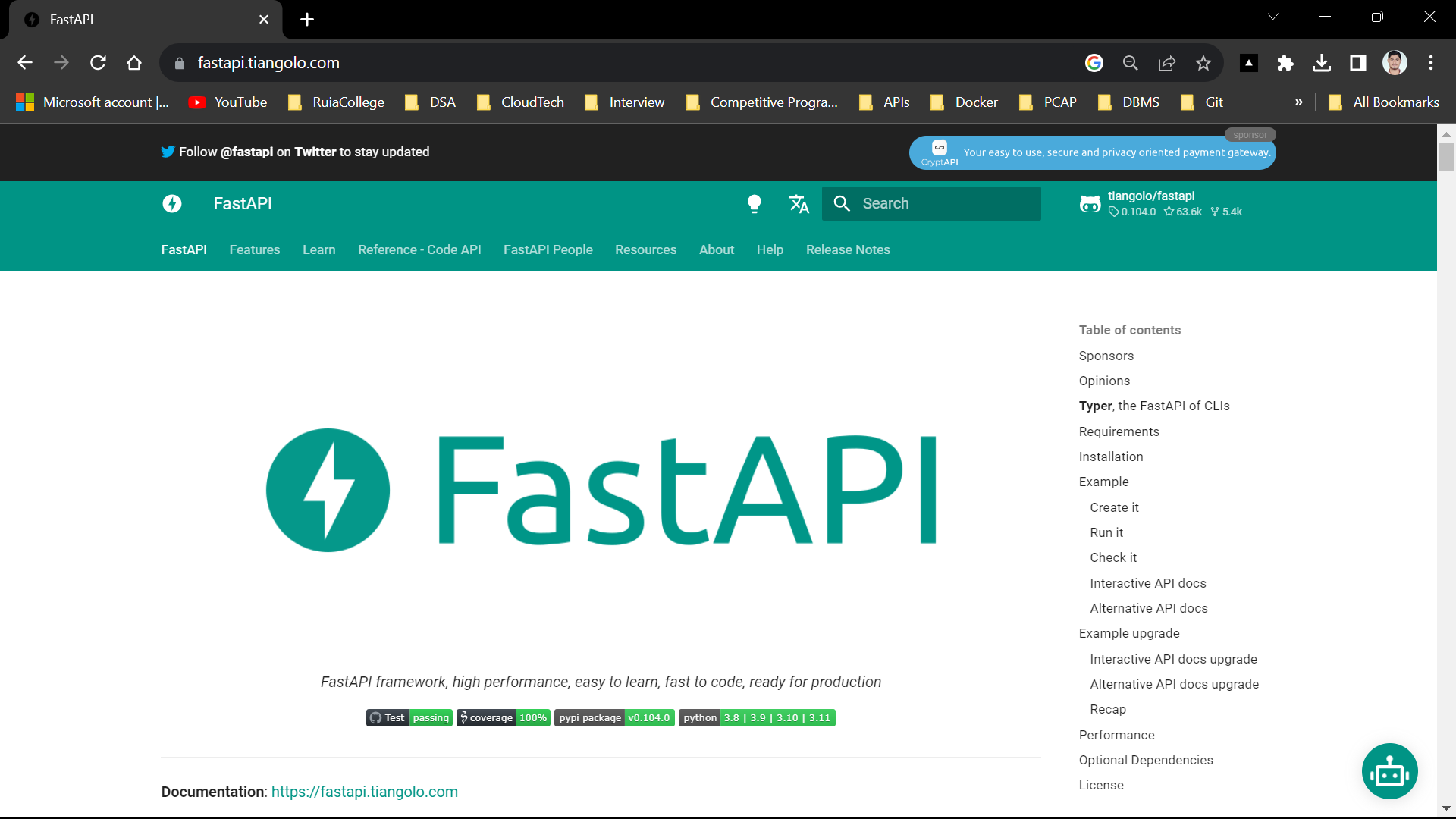
Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript engine, and executes JavaScript code outside a web browser.

**NextJS:**

****

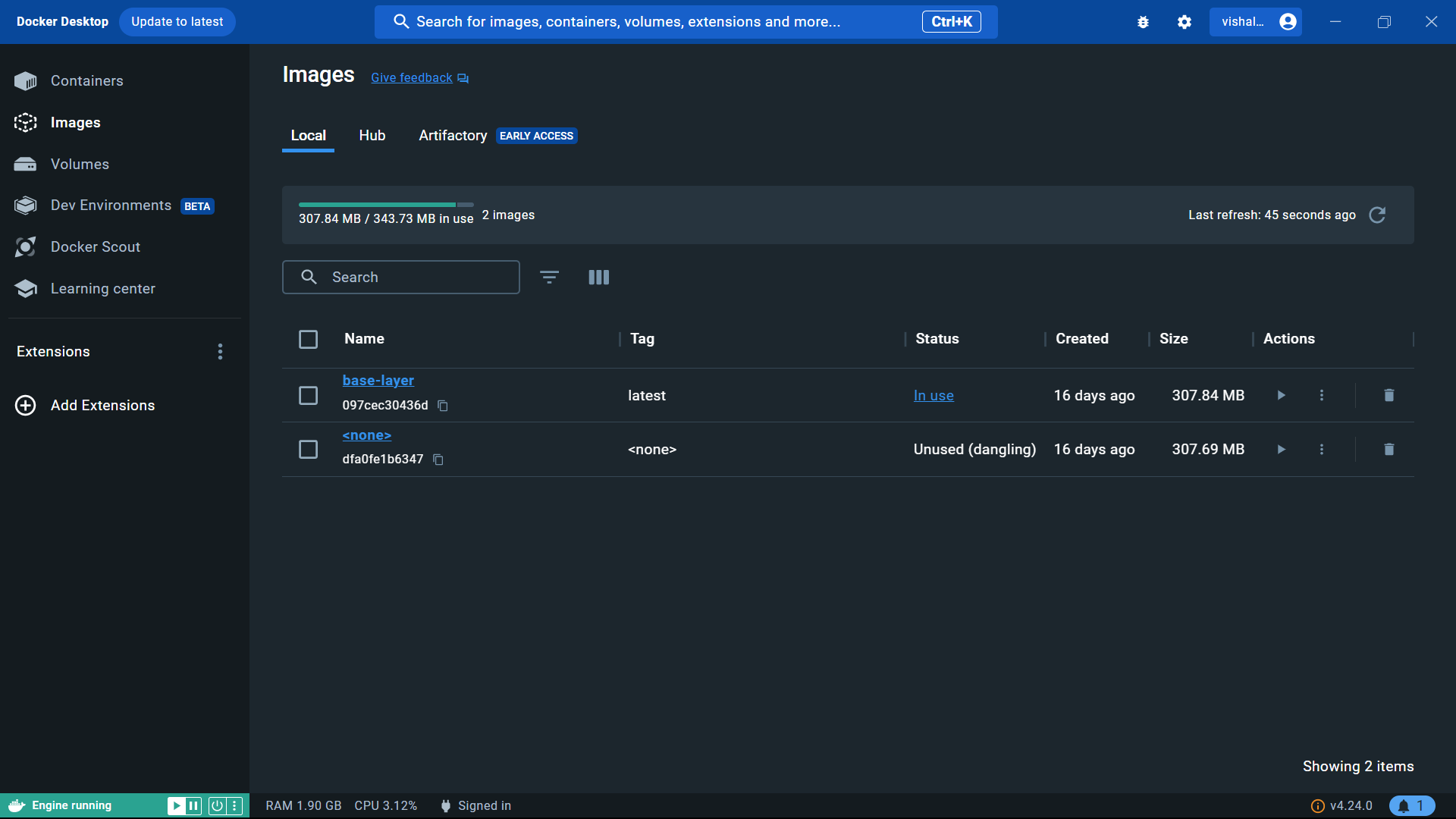
Next.js is an open-source web development framework created by the private company Vercel providing React-based web applications with server-side rendering and static website generation. In this project I have used NextJS for developing the attractive front-end.

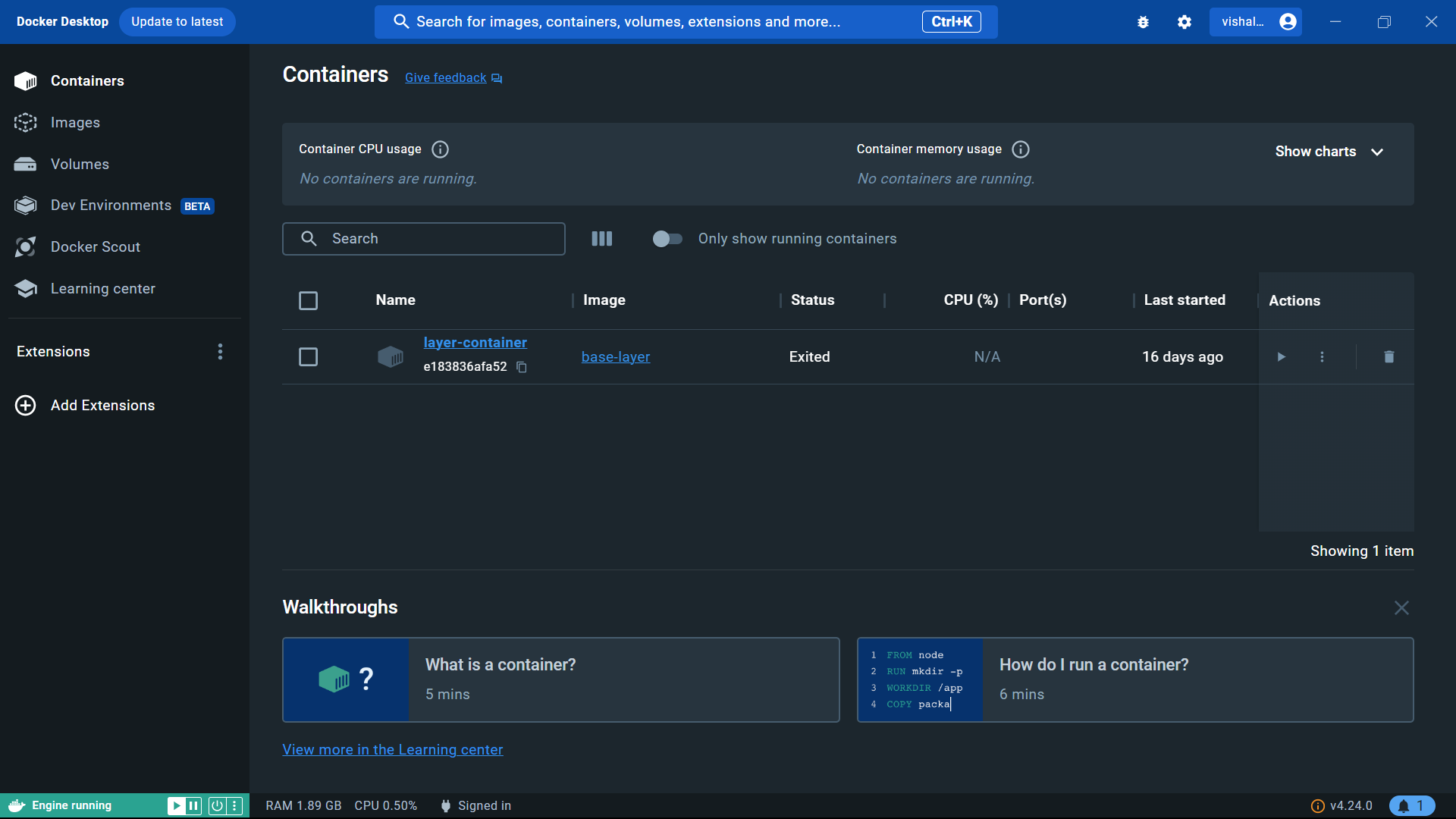
**FastAPI:**

****

FastAPI is a modern web framework for building RESTful APIs in Python. It was first released in 2018 and has quickly gained popularity among developers due to its ease of use, speed and robustness. FastAPI is based on Pydantic and uses type hints to validate, serialize and deserialize data.

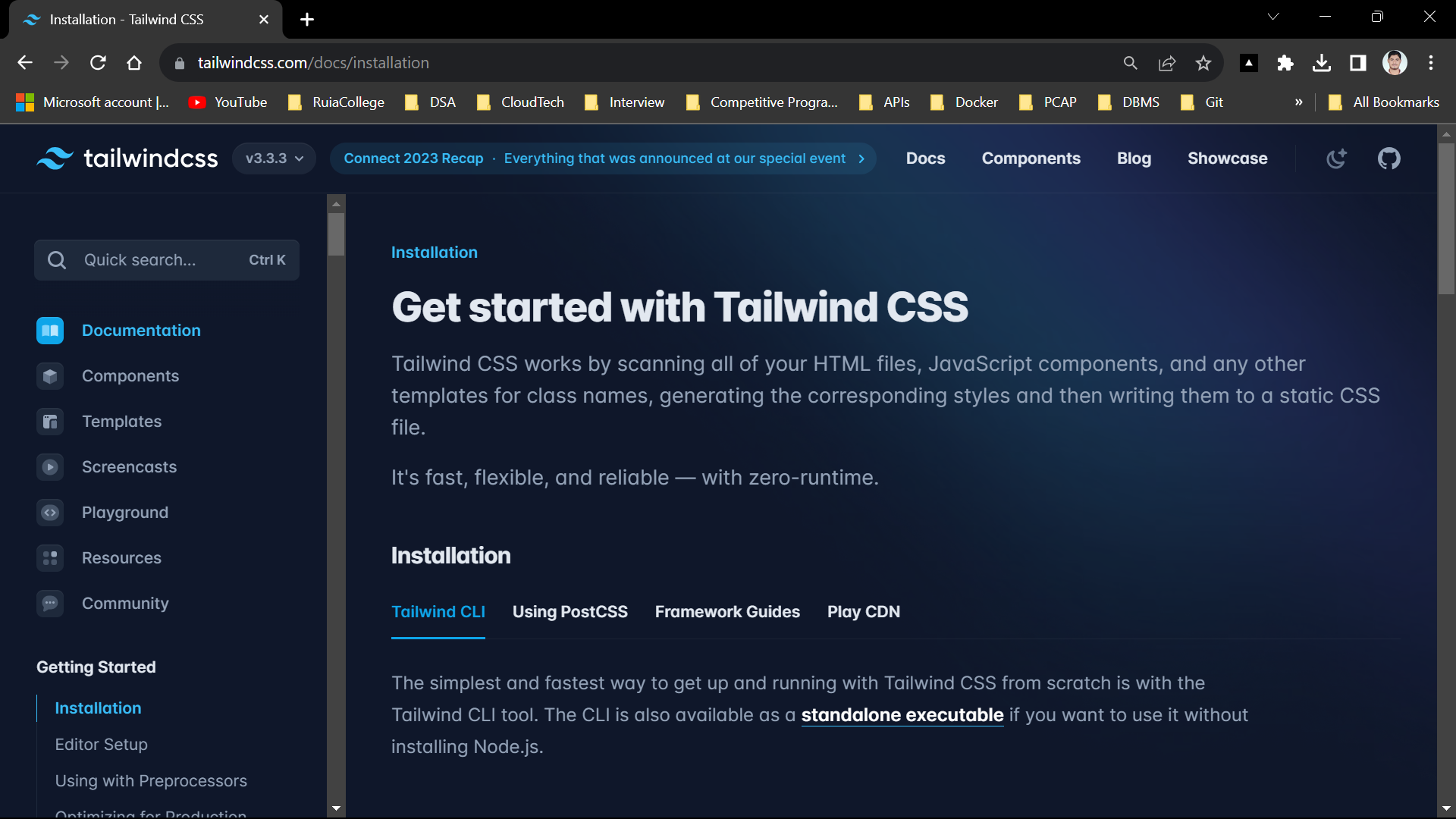
**Docker:**

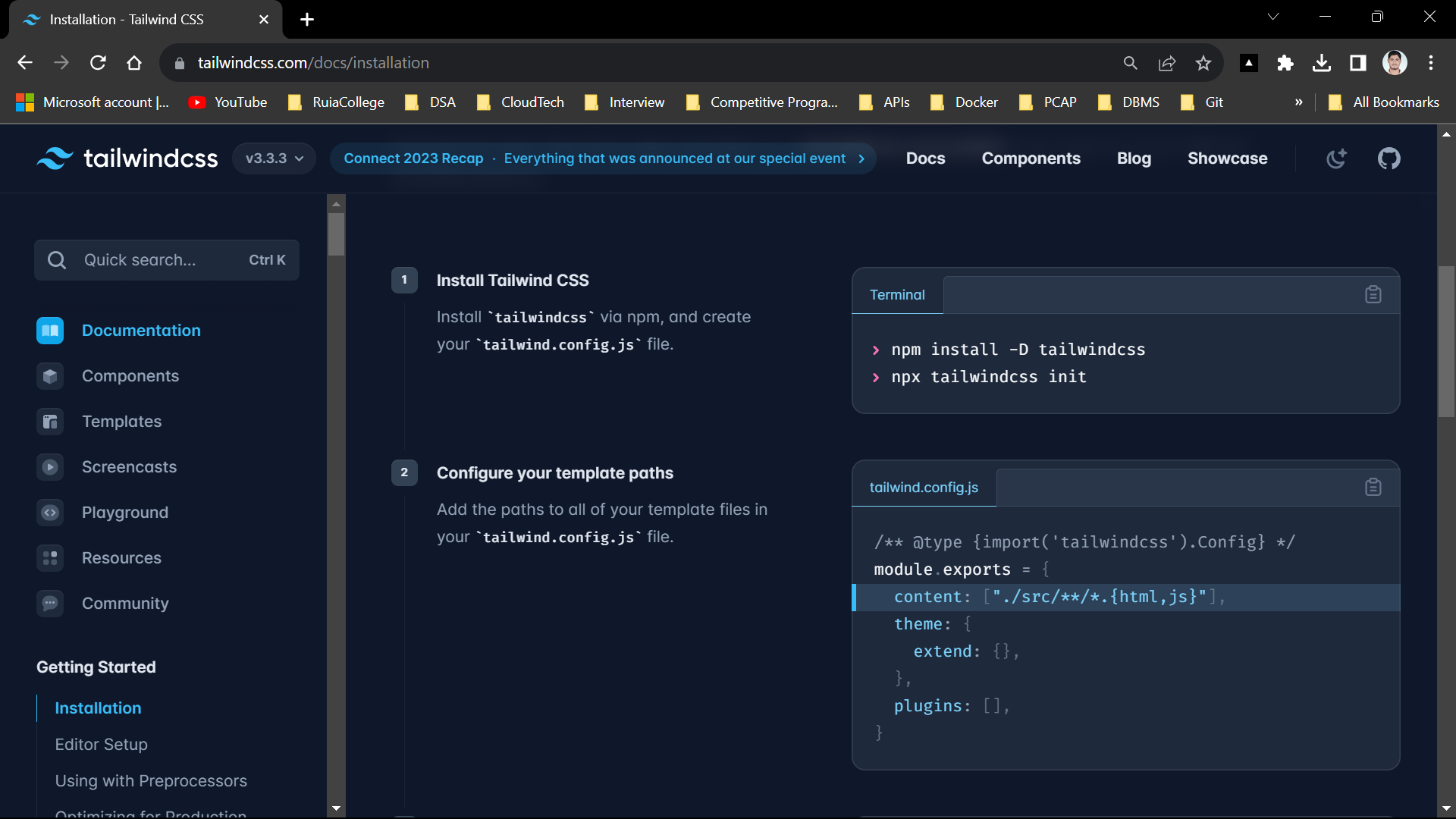
****

****

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first released in 2013 and is developed by Docker, Inc.

**Tailwind CSS:**

****

****

Tailwind CSS is an open source CSS framework. The main feature of this library is that, unlike other CSS frameworks like Bootstrap, it does not provide a series of predefined classes for elements such as buttons or tables.

**Software Language:**

**Python:**

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented and functional programming.

**TypeScript:**

TypeScript is a free and open-source high-level programming language developed by Microsoft that adds static typing with optional type annotations to JavaScript. It is designed for the development of large applications and transpiles to JavaScript.

**HTML:**

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It defines the meaning and structure of web content.

**CSS:**

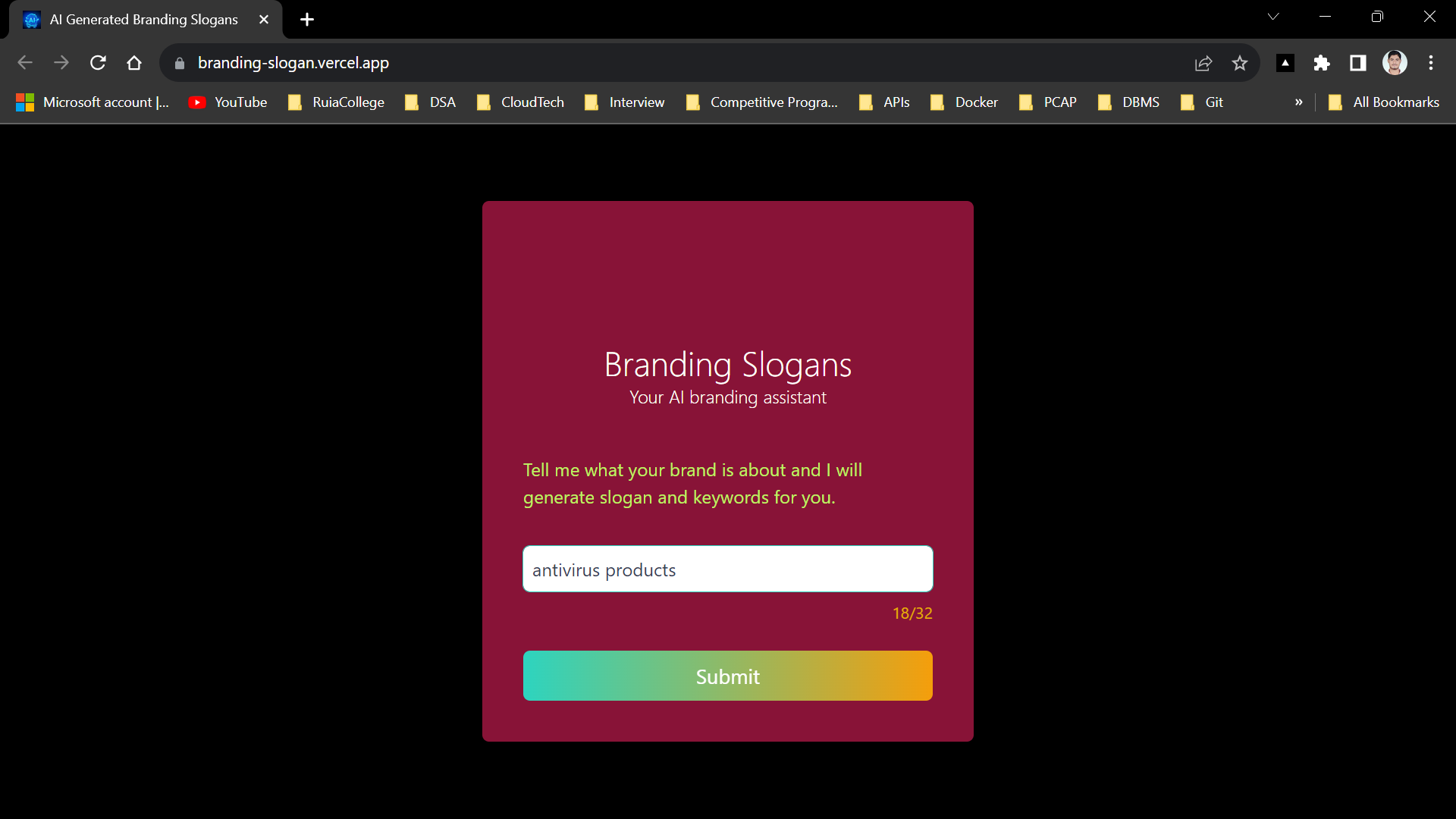
Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation and layout of web pages written in HTML and XML.

**Results:**

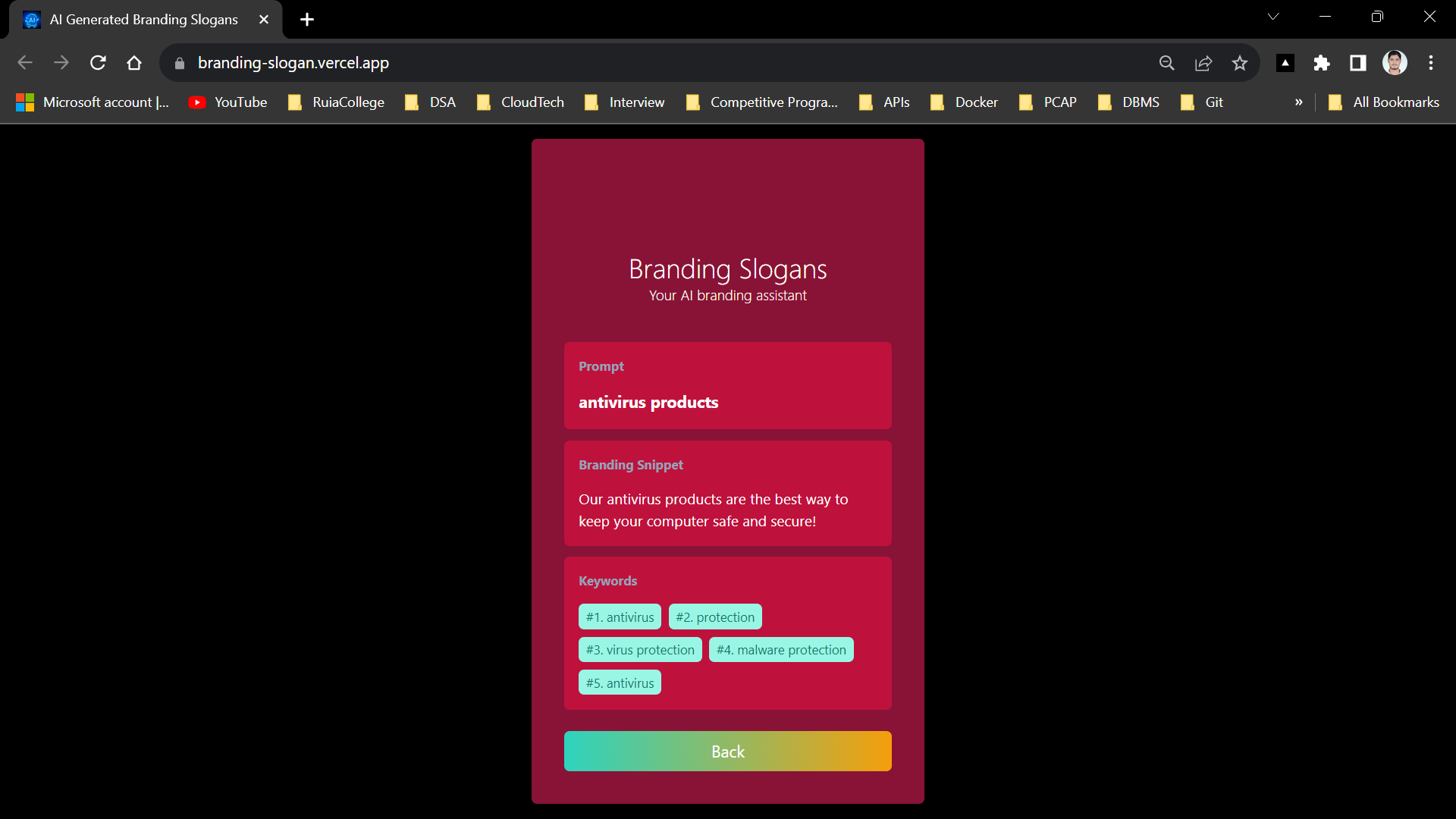
1. **Website Link:**

[**https://branding-slogan.vercel.app/**](https://branding-slogan.vercel.app/)

1. **Form:**

****

1. **Response/Result:**

****

1. **AI-Powered Slogan Generation:**
   1. The AI-based branding slogan generator successfully generates creative and context-aware branding slogans based on user-provided brand information. The slogans are generated with high accuracy, taking into account the brand's vision, values, and mission, ensuring that they resonate with the intended audience.
2. **Scalability and Speed:**
   1. Utilizing AWS Lambda for serverless computing, our system demonstrates remarkable scalability and speed. It can handle a high volume of simultaneous requests without any noticeable degradation in performance. Users can generate slogans rapidly, making it a practical tool for businesses of all sizes.
3. **User-Friendly Web Interface:**
   1. The integration of Vercel and Node.js provides an intuitive and user-friendly web interface. Users can easily input their brand details and receive customized branding slogans in real-time. The combination of a powerful backend and an engaging frontend ensures a seamless user experience.
4. **Portability and Consistency:**
   1. Docker containers are used to encapsulate the AI model, making the system highly portable and consistent across different environments. This feature simplifies deployment and guarantees consistent performance, regardless of the hosting platform.
5. **Enhanced Branding Efforts:**
   1. By automating the slogan generation process, businesses can streamline their branding efforts and save valuable time. The AI-generated slogans offer a fresh perspective and creativity, enhancing the overall brand identity and messaging.

**Conclusion:**

1. **Efficiency and Creativity in Branding**

The project successfully demonstrates that AI can play a pivotal role in streamlining and enhancing the branding process. It automates the generation of branding slogans, which traditionally involved time-consuming brainstorming sessions, and adds a creative touch that can be hard to achieve through traditional methods.

1. **Scalability and Accessibility**

Utilizing AWS Lambda for serverless computing, the system showcases exceptional scalability, making it accessible to businesses of all sizes. Whether a startup or a well-established corporation, organizations can tap into the power of AI-driven branding without concerns about infrastructure limitations.

1. **User-Centric Design**

The integration of Vercel and Node.js in the project ensures a user-friendly web interface. This design element prioritizes user experience, allowing individuals and businesses to access and utilize the branding slogan generator with ease. A seamless interface makes the AI technology more approachable.

1. **Consistency and Portability**

The use of Docker containers ensures that the AI model can be consistently deployed across various environments. This enhances reliability and simplifies deployment processes, irrespective of the hosting platform or infrastructure.

1. **Enhancement of Brand Identity**

Automated branding slogan generation can enhance brand identity and messaging. The slogans generated by the AI model offer a fresh and innovative perspective, helping businesses to better resonate with their target audience and stand out in a competitive market.

In conclusion, the AI-Based Branding Slogan Generator project not only meets its primary objective of providing creative and context-aware branding slogans but also highlights the transformative potential of AI in the field of marketing and branding. By making use of AWS Lambda, Python, Docker, Vercel, Node.js, and other technologies, this project serves as a forward-thinking solution that empowers businesses to elevate their brand identity and streamline their marketing efforts in the digital age. As technology continues to evolve, it is clear that AI-driven branding will play an increasingly vital role in helping businesses connect with their audience and establish a strong and memorable brand presence.

**References:**

* Amazon Web Services, Inc. (2021). "AWS Lambda Documentation." <https://aws.amazon.com/lambda/documentation/>
* Python Software Foundation. (2021). "Python Documentation." <https://docs.python.org/3/>
* Docker, Inc. (2021). "Docker Documentation." <https://docs.docker.com/>
* Vercel Inc. (2021). "Vercel Documentation."

<https://vercel.com/docs>

* Node.js Foundation. (2021). "Node.js Documentation." <https://nodejs.org/en/docs/>