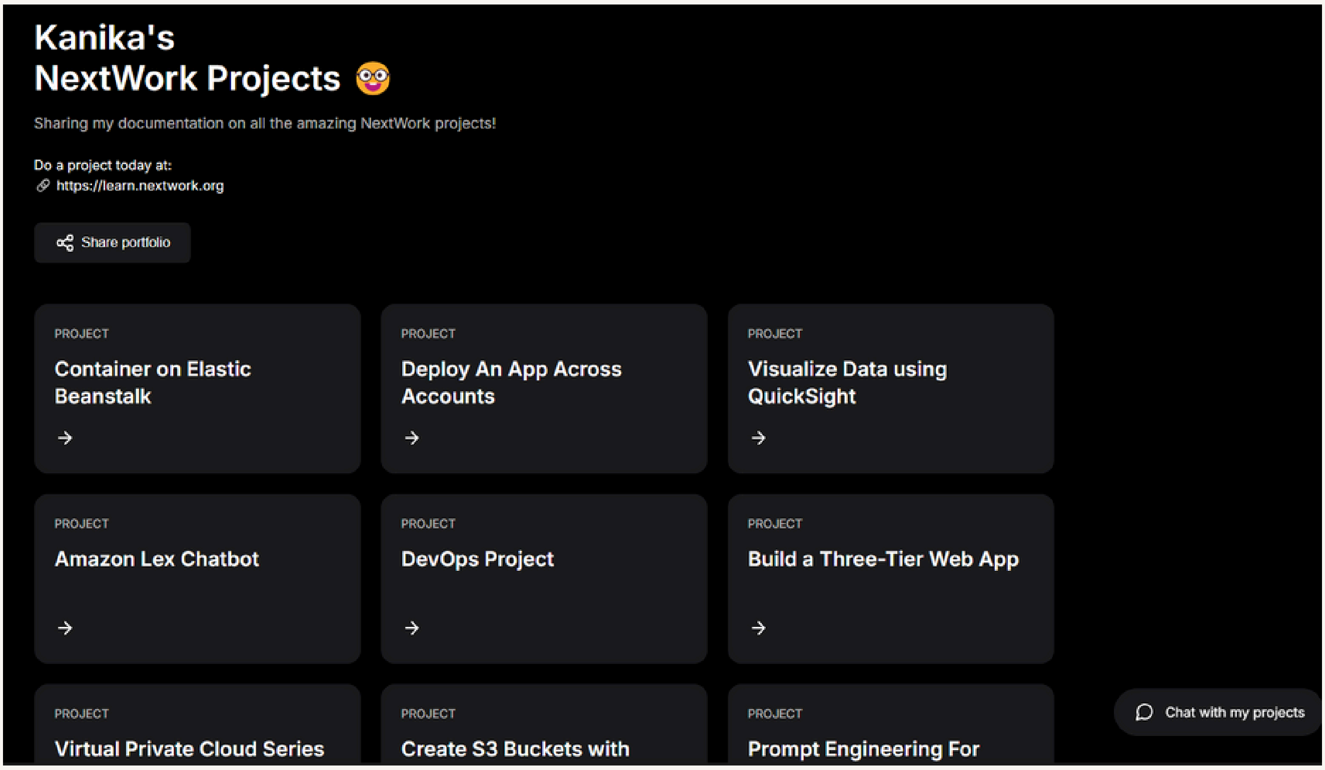
nextwork.org

# Building a RAG Chatbot with a Web Interface

Kanika Mathur
github.com/KanikaGenesis



Kanika's
NextWork Projects 🤓

Sharing my documentation on all the amazing NextWork projects!

Do a project today at:
https://learn.nextwork.org

PROJECT
Container on Elastic Beanstalk

PROJECT
Deploy An App Across Accounts

PROJECT
Visualize Data using QuickSight

PROJECT
Amazon Lex Chatbot

PROJECT
DevOps Project

PROJECT
Build a Three-Tier Web App

PROJECT
Virtual Private Cloud Series

PROJECT
Create S3 Buckets with

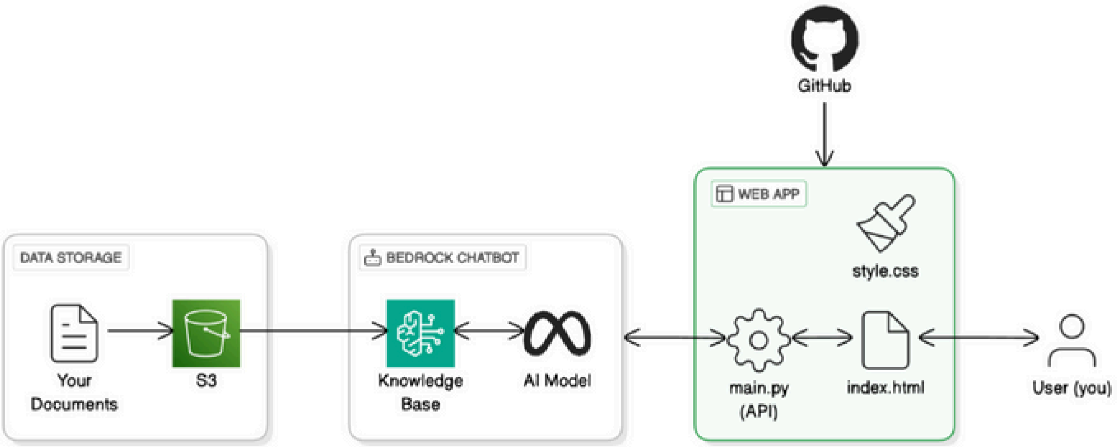
PROJECT
Prompt Engineering For

# Introducing Today's Project!

Retrieval-augmented generation (RAG) is a powerful technique that enhances chatbot performance by allowing it to fetch relevant data dynamically.
In this project, I built a RAG chatbot for my portfolio Web App. By doing this project I learnt how to not only create AI tools or resources (like a RAG chatbot) but also integrate those tools into web apps that users can use.

## Tools and Technologies Used

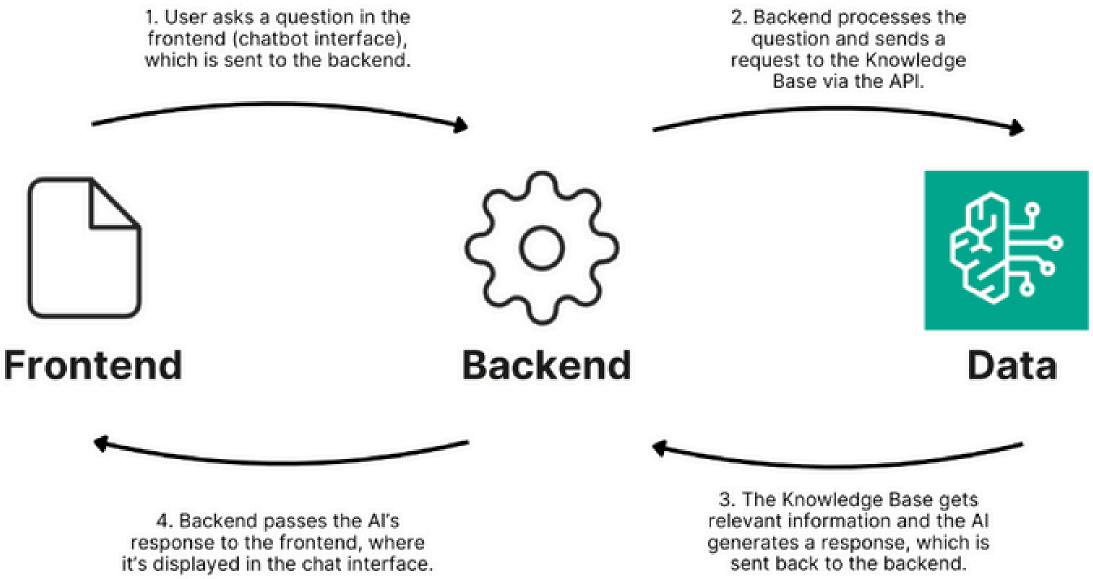
- Amazon Bedrock – For AI-based response generation.
- Amazon S3 – To store chatbot data and project files.
- OpenSearch Serverless – For real-time document search.
- FastAPI & Uvicorn – To build and run the backend API.
- GitHub – For version control and project management.
- IAM (Identity and Access Management) – For secure access to AWS services.
- HTML, JavaScript, and CSS – To build the frontend web interface.



# How It Works!

A web app is often made up of three layers.

- **Layer 1**: The data is the information our app needs. For our RAG chatbot, this is the knowledge we're storing in Bedrock that our chatbot will use to answer questions.
- **Layer 2**: The backend handles the processing and logic. For our RAG chatbot, this is the code that passes your users' messages to Amazon Bedrock (to generate a response behind the scenes) and passes back the response.
- **Layer 3**: The frontend is what users see and interact with. For our RAG chatbot, this is the chat interface that lets users send a message.



1. User asks a question in the frontend (chatbot interface), which is sent to the backend.

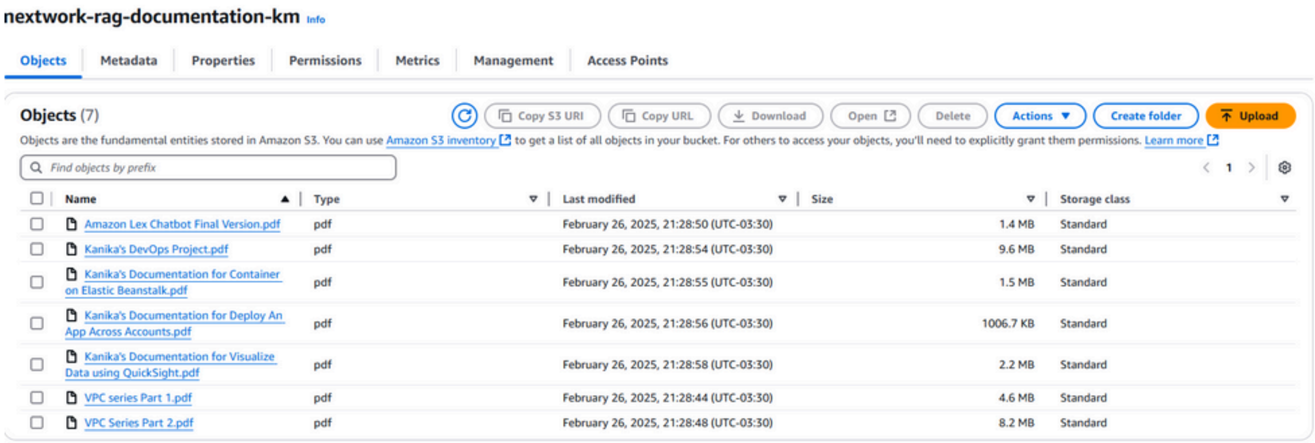
2. Backend processes the question and sends a request to the Knowledge Base via the API.

3. The Knowledge Base gets relevant information and the AI generates a response, which is sent back to the backend.

4. Backend passes the AI's response to the frontend, where it's displayed in the chat interface.

# Uploading Project Files to Amazon S3

For creating the RAG chatbot’s Knowledge Base, I uploaded my project files to an Amazon S3 bucket. This data will be the source of knowledge for my chatbot.
This ensured that my chatbot had easy access to structured data, that would allow it to retrieve accurate responses efficiently. I will connect this S3 to a Knowledge Base in Bedrock.

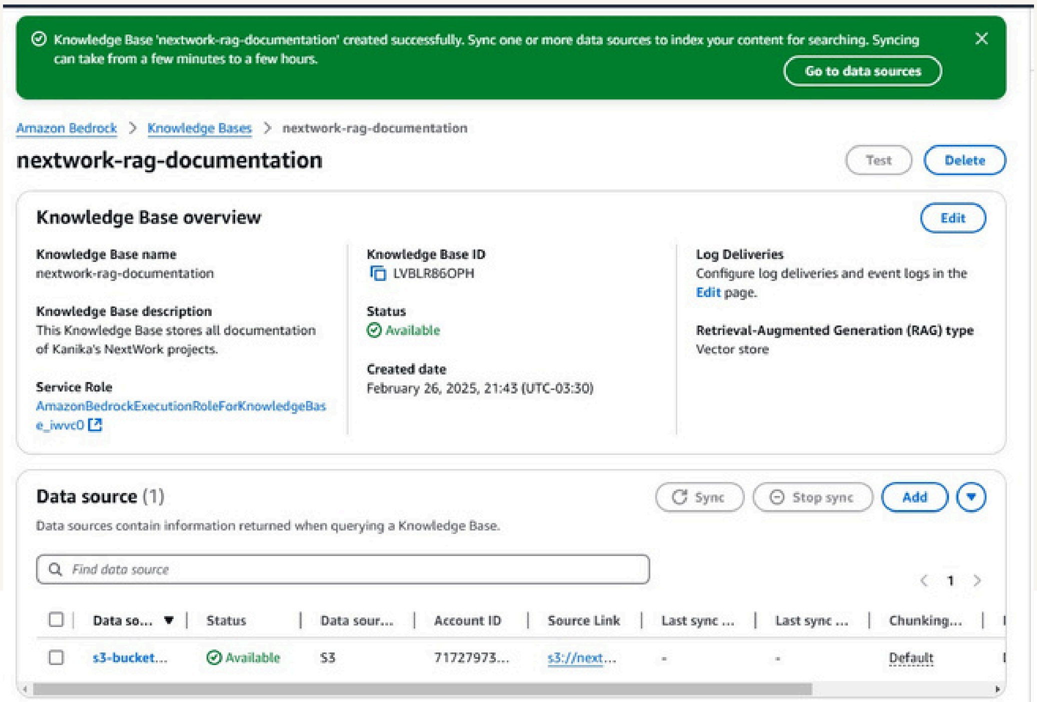


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

| ☐ | Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|---|
| ☐ | Amazon Lex Chatbot Final Version.pdf | pdf | February 26, 2025, 21:28:50 (UTC-03:30) | 1.4 MB | Standard |
| ☐ | Kanika's DevOps Project.pdf | pdf | February 26, 2025, 21:28:54 (UTC-03:30) | 9.6 MB | Standard |
| ☐ | Kanika's Documentation for Container on Elastic Beanstalk.pdf | pdf | February 26, 2025, 21:28:55 (UTC-03:30) | 1.5 MB | Standard |
| ☐ | Kanika's Documentation for Deploy An App Across Accounts.pdf | pdf | February 26, 2025, 21:28:56 (UTC-03:30) | 1006.7 KB | Standard |
| ☐ | Kanika's Documentation for Visualize Data using QuickSight.pdf | pdf | February 26, 2025, 21:28:58 (UTC-03:30) | 2.2 MB | Standard |
| ☐ | VPC series Part 1.pdf | pdf | February 26, 2025, 21:28:44 (UTC-03:30) | 4.6 MB | Standard |
| ☐ | VPC Series Part 2.pdf | pdf | February 26, 2025, 21:28:48 (UTC-03:30) | 8.2 MB | Standard |

# Knowledge Base Setup in Bedrock

I created a Knowledge Base to store all the information that my chatbot needs to know. This is important because it sets up the DATA layer of my entire web app. Without the knowledge base, my chatbot wouldn't know how to answer my tailored questions. I synced the Knowledge Base to the S3 bucket, which will make the training materials on S3 available to my chatbot. The KBase uses a vector database and embeddings model, so my data is searchable by meaning (not just keywords).



Knowledge Base 'nextwork-rag-documentation' created successfully. Sync one or more data sources to index your content for searching. Syncing can take from a few minutes to a few hours.

Go to data sources
Amazon Bedrock > Knowledge Bases > nextwork-rag-documentation
Test
Delete
Edit

## nextwork-rag-documentation

### Knowledge Base overview

**Knowledge Base name**
nextwork-rag-documentation

**Knowledge Base description**
This Knowledge Base stores all documentation of Kanika's NextWork projects.

**Service Role**
AmazonBedrockExecutionRoleForKnowledgeBase_iwvc0

**Knowledge Base ID**
LVBLR86OPH

**Status**
Available

**Created date**
February 26, 2025, 21:43 (UTC-03:30)

**Log Deliveries**
Configure log deliveries and event logs in the Edit page.

**Retrieval-Augmented Generation (RAG) type**
Vector store

### Data source (1)

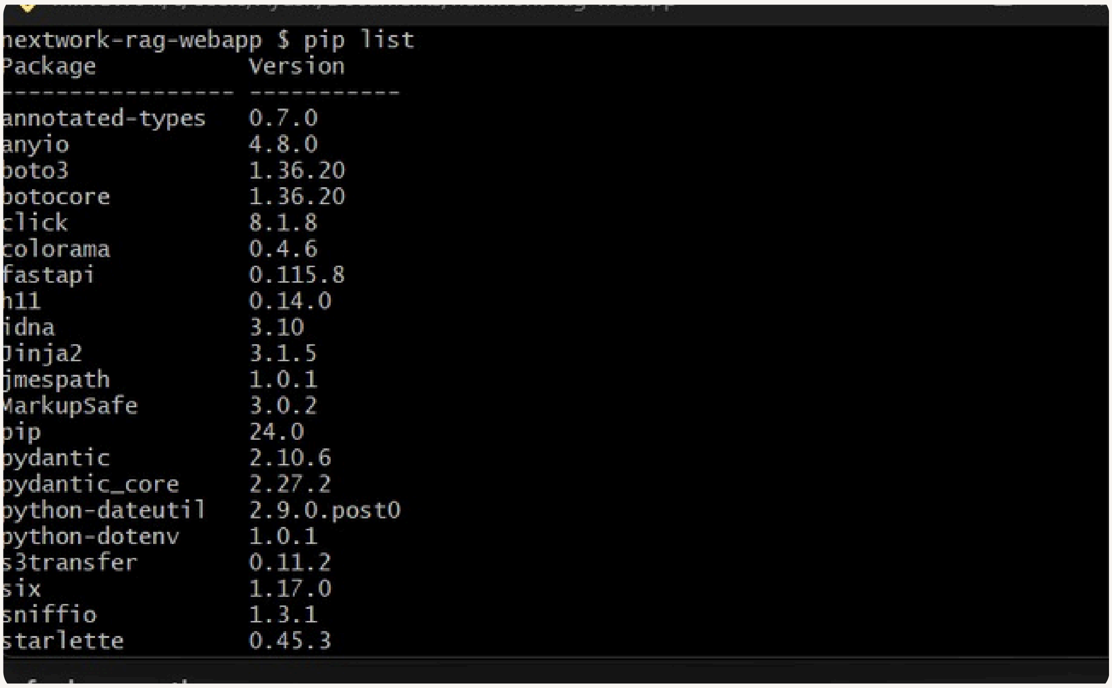
Data sources contain information returned when querying a Knowledge Base.

| ☐ | Data so... | Status | Data sour... | Account ID | Source Link | Last sync ... | Last sync ... | Chunking... |
|---|---|---|---|---|---|---|---|---|
| ☐ | s3-bucket... | Available | S3 | 71727973... | s3://next... | - | - | Default |

# Setting Up the API

To run the API, I had to install requirements like **Fast API, uvicorn** and **Boto3**. These packages are important because my web app's frontend and backend code need them to run certain features e.g, the connection to Bedrock, running of the API.

A **virtual environment** helps me by keeping all of the packages & tools we install for this web app in one place. We need a virtual environment because we have a lot of tools to install and it is a lot easier to delete them later if they are all in a single environment.



```
nextwork-rag-webapp $ pip list
Package            Version
------------------ -----------
annotated-types    0.7.0
anyio              4.8.0
boto3              1.36.20
botocore           1.36.20
click              8.1.8
colorama           0.4.6
fastapi            0.115.8
h11                0.14.0
idna               3.10
Jinja2             3.1.5
jmespath           1.0.1
MarkupSafe         3.0.2
pip                24.0
pydantic           2.10.6
pydantic_core      2.27.2
python-dateutil    2.9.0.post0
python-dotenv      1.0.1
s3transfer         0.11.2
six                1.17.0
sniffio            1.3.1
starlette          0.45.3
```

# Breaking down the API

In the web app, we need an API because APIs are software tools that help us connect different applications together, aka get them to talk to each other or share information with each other. I will use my API to connect my app with Amazon Bedrock.

Let us dig deeper into the API code - the main tools we need to import are **FastAPI, Boto3, os, and load_env**. These tools help us connect with AWS, load environment variables, and use the Python framework to build APIs.

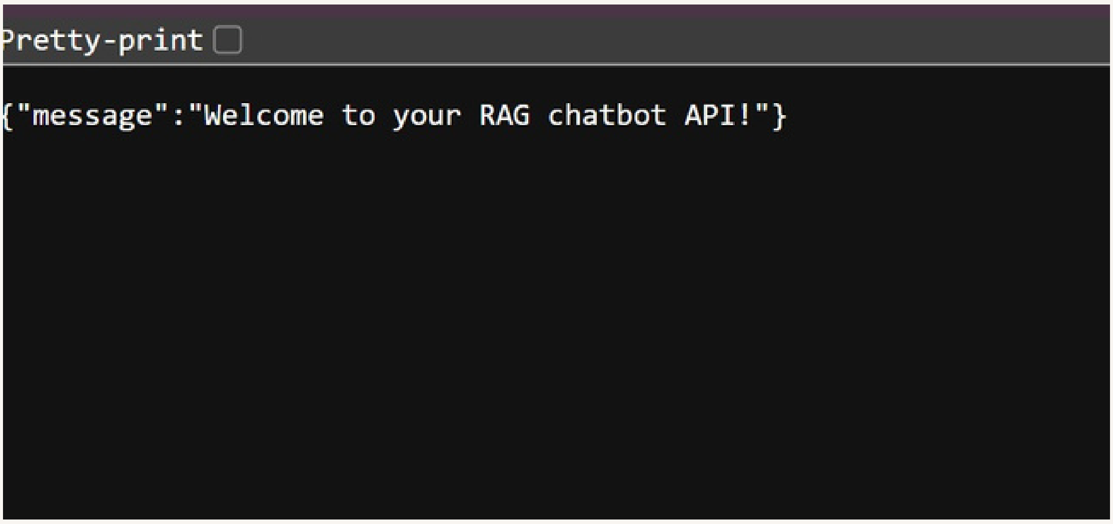
The environment variables we need include **AWS region, Knowledge Base ID and Model ARN**. We store them separately because potential attackers can use them to steal data. We are protecting our information by avoiding hard coding them in the web app.

My API uses Python AWS SDK to set up a connection between my app and AWS. The difference between the AWS CLI and an SDK is that the CLI is for running quick commands in the terminal for direct connection while the SDK is for integrating the connection into the app.

The API has two main routes. One is a root route, it just returns the message. The other is a special route for querying our KBase, i.e. ask chatbot a question. In general, routes help us to separate our APIs functionalities into different paths.

# Testing the API

When I visited the root endpoint, I saw a message from my API. This confirms that my backend code is working in my local environment. My API is running.

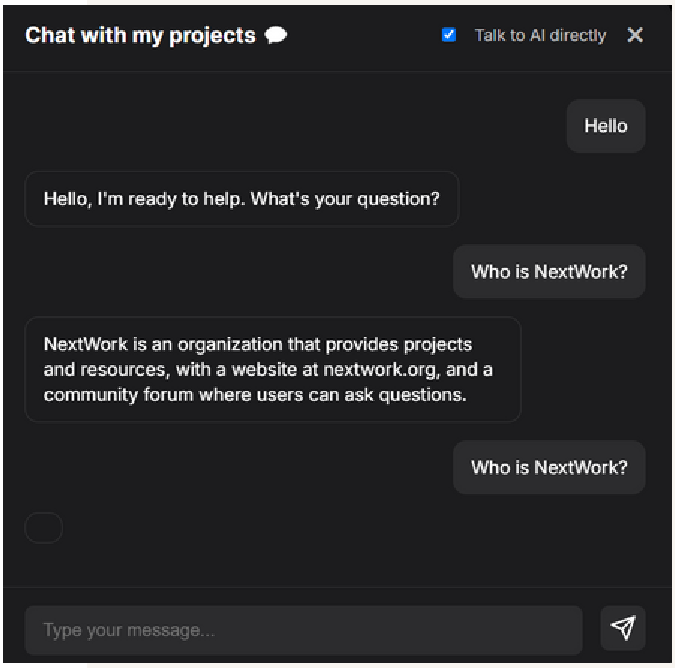


```
{"message":"Welcome to your RAG chatbot API!"}
```

# Running the Web App

The difference between the API and the web app for the user is the visual experience. Instead of having to type in a direct API endpoint URL in the browser, the user simply opens a web app link and sees a nice interface and chat modal.

The web app can switch between using RAG and going directly to the AI model. When I asked the same question, "Who is NextWork?" directly to the AI model, it no longer used the Knowledge Base and failed to give me a proper answer.



**Chat with my projects** 💬

☑ Talk to AI directly

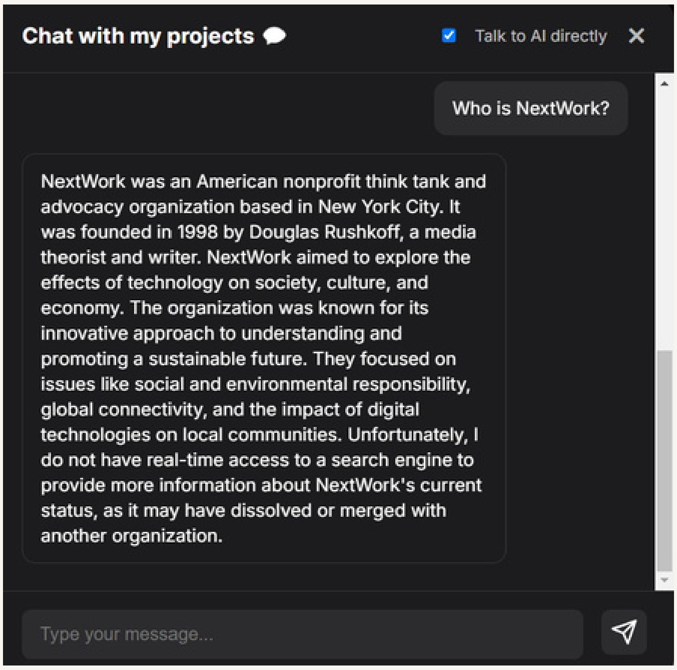
Hello

Hello, I'm ready to help. What's your question?

Who is NextWork?

NextWork is an organization that provides projects and resources, with a website at nextwork.org, and a community forum where users can ask questions.

Who is NextWork?



**Chat with my projects** 💬

☑ Talk to AI directly

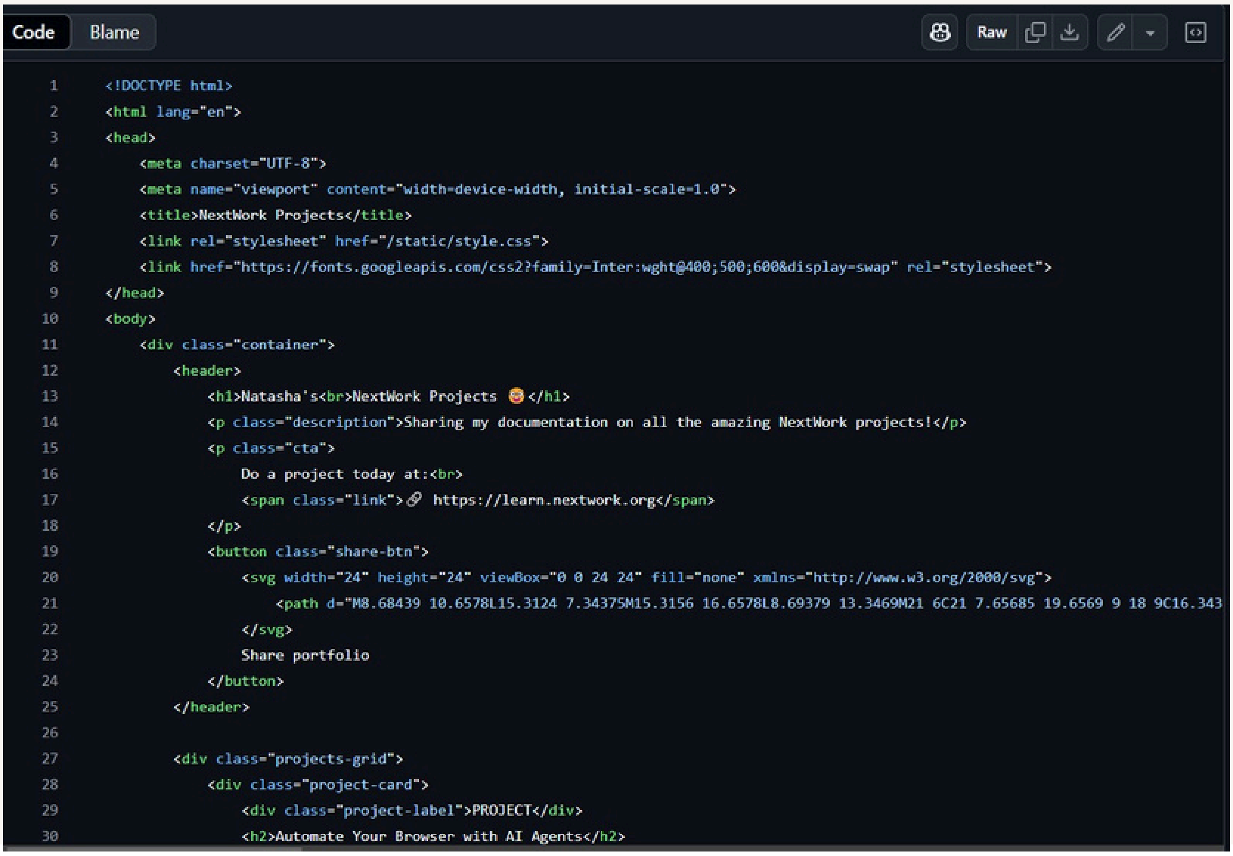
Who is NextWork?

NextWork was an American nonprofit think tank and advocacy organization based in New York City. It was founded in 1998 by Douglas Rushkoff, a media theorist and writer. NextWork aimed to explore the effects of technology on society, culture, and economy. The organization was known for its innovative approach to understanding and promoting a sustainable future. They focused on issues like social and environmental responsibility, global connectivity, and the impact of digital technologies on local communities. Unfortunately, I do not have real-time access to a search engine to provide more information about NextWork's current status, as it may have dissolved or merged with another organization.

# Breaking down the Web App

When a user sends a chat message, the javaScript code in index.html is responsible for packaging the message and sending it to our backend endpoint in web_app.py, which is the backend file. The endpoint fetches a response and then passes it back to the frontend for rendering.

web_app.py extends the API by adding the ability to also load the app's frontend, which includes the HTML file and the CSS file. To load these files, it also needed to import more tools and load a new environment variable - **model ID.**



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>NextWork Projects</title>
    <link rel="stylesheet" href="/static/style.css">
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600&display=swap" rel="stylesheet">
</head>
<body>
    <div class="container">
        <header>
            <h1>Natasha's<br>NextWork Projects 🤓</h1>
            <p class="description">Sharing my documentation on all the amazing NextWork projects!</p>
            <p class="cta">
                Do a project today at:<br>
                <span class="link">🔗 https://learn.nextwork.org</span>
            </p>
            <button class="share-btn">
                <svg width="24" height="24" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
                    <path d="M8.68439 10.6578L15.3124 7.34375M15.3156 16.6578L8.69379 13.3469M21 6C21 7.65685 19.6569 9 18 9C16.343
                </svg>
                Share portfolio
            </button>
        </header>

        <div class="projects-grid">
            <div class="project-card">
                <div class="project-label">PROJECT</div>
                <h2>Automate Your Browser with AI Agents</h2>
```

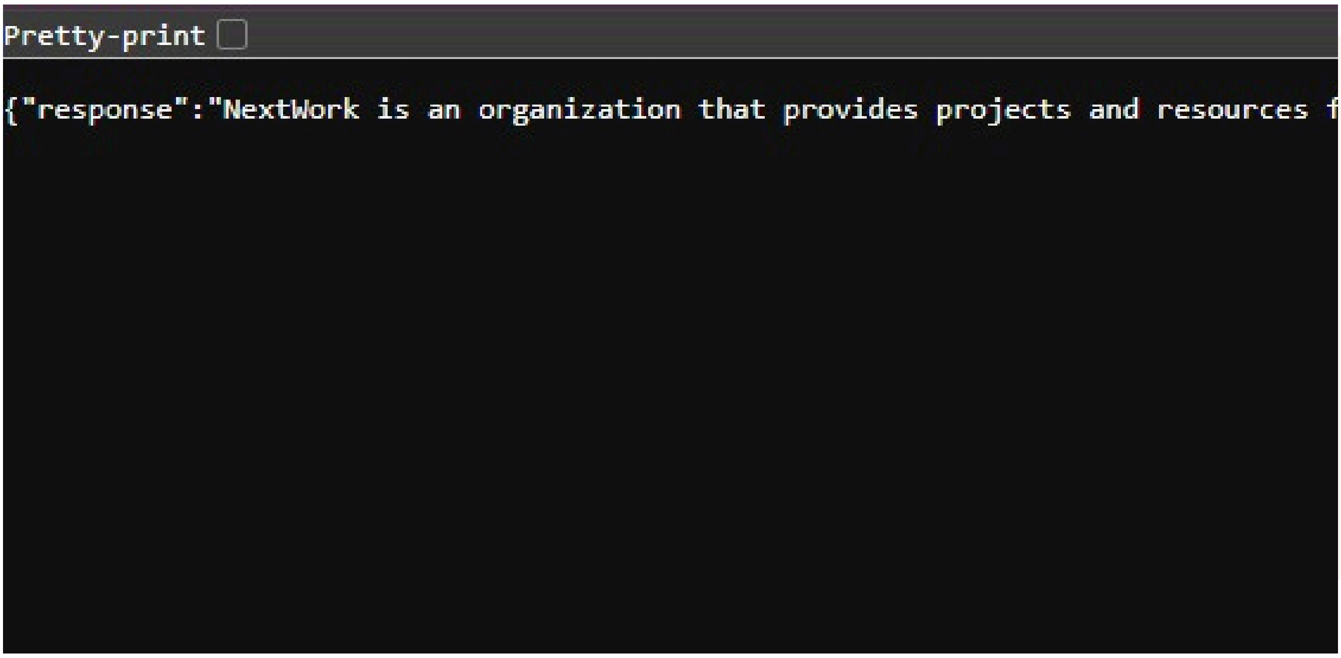
# Troubleshooting and Reflection

During the project, I encountered and resolved the following issues:

To fix the Parameter Failed error that occurred when I tested the API, I had to store the environment variables in .env file within my project directory. Doing this helps my backend to configure and use Knowledge Base and Model ARN using environment variables that have been set.

I fixed the credential error by changing my AWS region configured from us-east-1 to us-east-2.

My chatbot initially failed to generate responses, so I went back into my Knowledge Base setup to sync my Knowledge Base with the raw data stored in S3. This fixed my chatbot's failure because my Knowledge Base has all the information it needs to answer my questions.



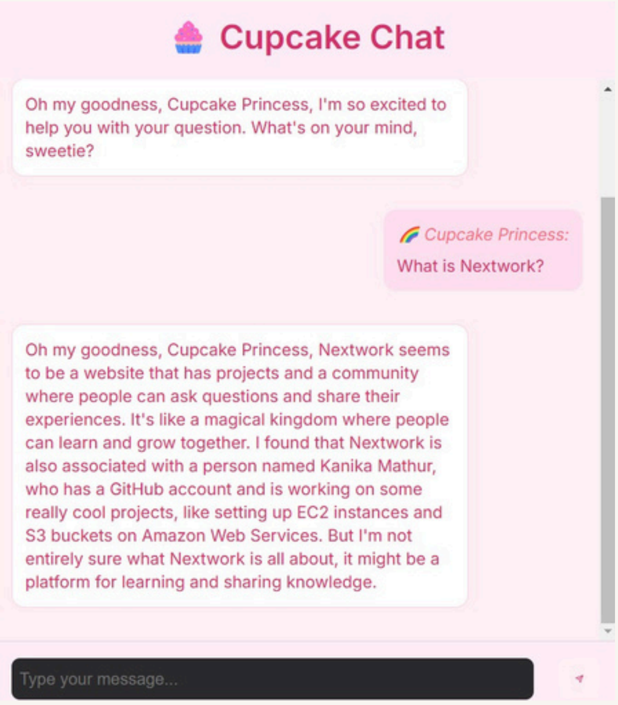
Pretty-print ☐

```
{"response":"NextWork is an organization that provides projects and resources f
```

# Customizing the Frontend

I want to experiment with customizing the frontend because I want to personalize the web app. This is possible because as long as the API endpoints stay the same in index.html, the connection to the backend will still work, i.e., chat responses.

My customized interface now features a focused chat app without a portfolio displayed/ It uses a different colour scheme, but I only had to change the index.html file. Every other file in the web app remains the same.



🧁 Cupcake Chat

Oh my goodness, Cupcake Princess, I'm so excited to help you with your question. What's on your mind, sweetie?

🌈 *Cupcake Princess:*
What is Nextwork?

Oh my goodness, Cupcake Princess, Nextwork seems to be a website that has projects and a community where people can ask questions and share their experiences. It's like a magical kingdom where people can learn and grow together. I found that Nextwork is also associated with a person named Kanika Mathur, who has a GitHub account and is working on some really cool projects, like setting up EC2 instances and S3 buckets on Amazon Web Services. But I'm not entirely sure what Nextwork is all about, it might be a platform for learning and sharing knowledge.

# Everyone should be in a job they love.

Check out nextwork.org for more projects