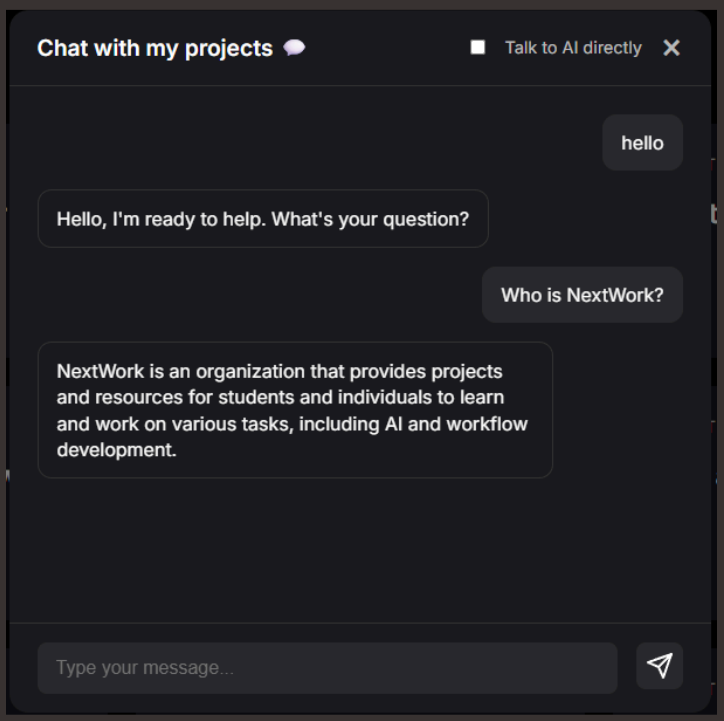
nextwork.org

Building a RAG Chatbot with a Web Interface

ME Melvin J Bonner



ME
Melvin J Bonner
NextWork Student

Introducing Today's Project!

In this project, I will build a fully functional web app where users can chat with my chatbot and get answers based on my documents. I'm doing this project to gain hands-on experience building complete web applications that integrate AI capabilities.

Tools and concepts

Services I used were Amazon Bedrock, S3, Amazon OpenSearch Service, and IAM. Key concepts I learned included Knowledge Base, RAG chatbot, Query Endpoint, and vector store.

Project reflection

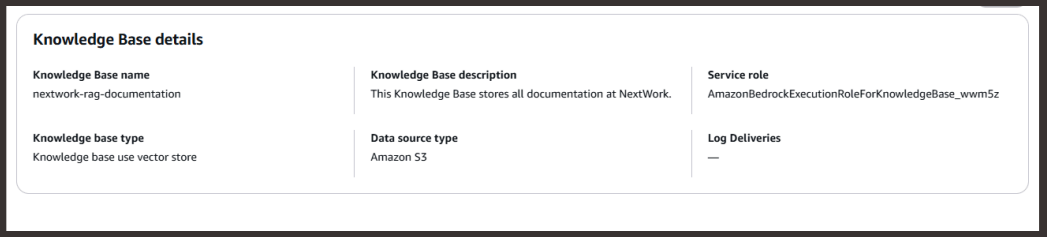
This project took me approximately 3 hours. the most challenging part was troubleshooting the parameter failed error. It was the most rewarding to have a fully functioning chatbot.

I wanted to gain more experience with AI and a better understanding of Amazon Bedrock. This project met my golas.

ME
Melvin J Bonner
NextWork Student

Chatbot setup

I created a Knowledge Base to create the knowledge management system for my chatbot. This is important because when I ask my chatbot a question, the Knowledge Base is responsible for quickly finding the information that will answer my question.



Setting Up the API

To run the API, I had to install requirements like fastapi, uvicorn, boto3, and python-dotenv. These packages are important because they are the required packages needed for Python to run the API.

A virtual environment helps me by telling my computer to start using the virtual environment I created to allow all the Python packages I installed to only exist in this environment. A virtual environment will not allow my project to interfere with other Python projects on my computer.

Breaking down the API

In our web app, we need an API because the API lets our web app talk to the Amazon Bedrock chatbot.

Digging deeper into the API code, the main tools I need to import are FastAPI, AWS SDK, environment variables, and load. These tools help by giving the tools I need for the API to work. Instead of coding everything from scratch, I'm grabbing pre-built tools. The tools make it way faster than making every piece myself.

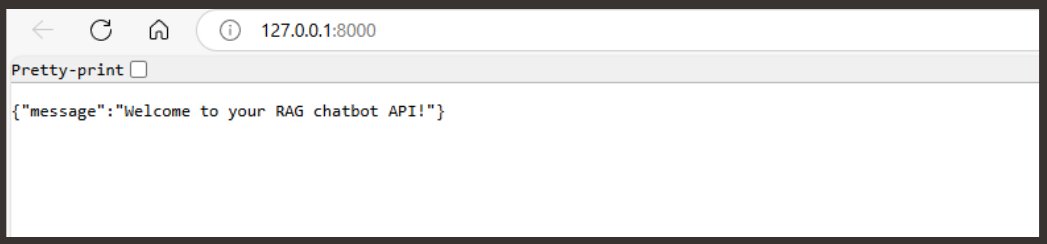
The environment variables we need are AWS_REGION, KNOWLEDGE_BASE_ID, and MODEL_ARN. I store them separately because putting these variables directly in the code is a security risk. If you accidentally push your code to GitHub with AWS credentials or Knowledge Base ID in it, anyone could use them. Environment variables keep sensitive stuff separate from your code.

My API uses the Python AWS SDK to to talk to Bedrock. The difference between the AWS CLI an SDK is you use the CLI when you need to do something quickly yourself in the terminal, and you use an SDK when you're building an application that needs to work with AWS.

Our API has two main routes: / is the root route, which is like the default route for my API. If the user doesn't specify a route, my API will use this route. /bedrock/query: is a route that lets the user query my chatbot. In general, an API route is like a path to a specific part or function in my API.

Testing the API

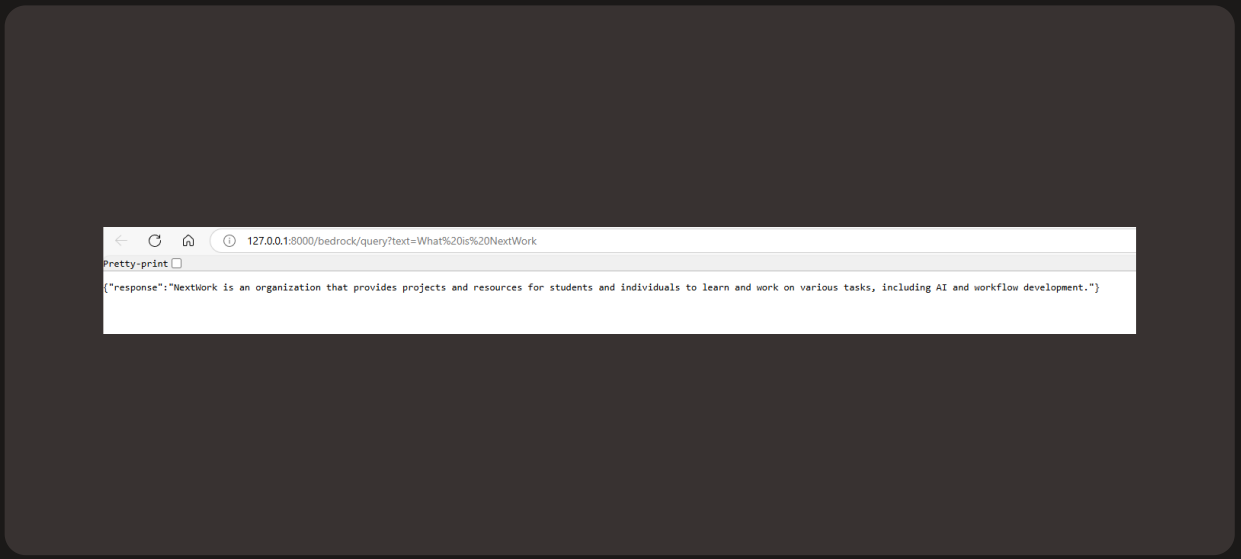
When I visited the root endpoint, I saw the message {"message": "Welcome to your RAG chatbot API"} in the browser. This confirms that the API is running.



Troubleshooting the API

My second endpoint had errors!

To fix the Parameter Failed error, I had to update the environment variables. Doing this helps to fix my error and get the API's query endpoint running.

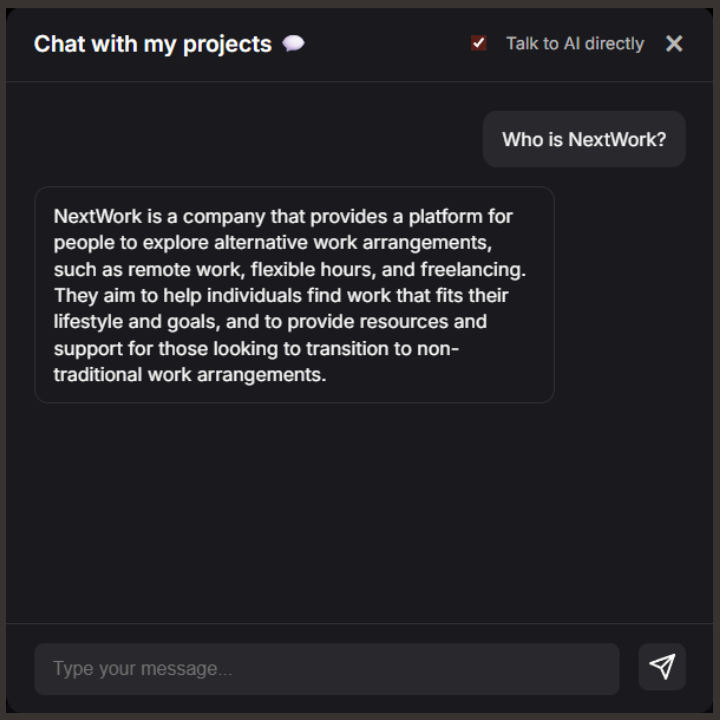


Running the Web App

The difference between the API and the web app for the user is how they react inside the app after receiving a question. The web app sends your question to the API endpoint, the API receives the question, and uses the Bedrock command to query my Knowledge Base with the question, the api send Bedrock's response back to the web app, and the web app displays the response.

The web app has the ability to switch between my knowledge base and general knowledge. When RAG is disabled, I ask the same question and get a lot less accurate response, because the chatbot is now using the AI model directly, without using my Knowledge Base.

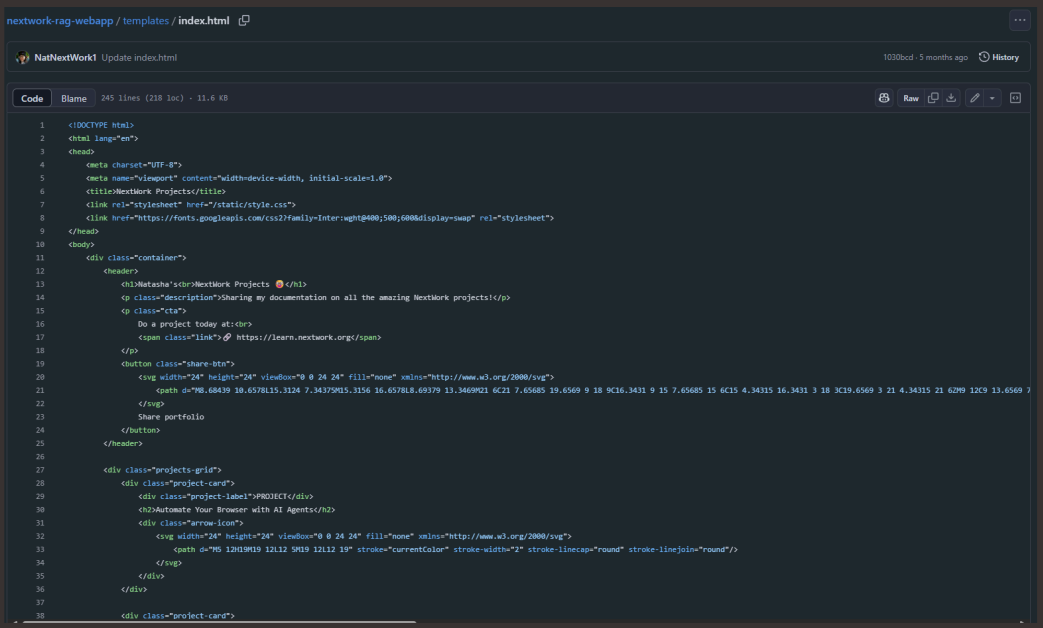
ME
Melvin J Bonner
NextWork Student



Breaking down the Web App

When a user sneds a chat message JavaScript makes an API call to /bedrock/query, the API (powered by web_app.py) processes your message using Bedrock, the response comes back and JavaScript displays it in the chat window.

web_app.apy extends the API by including features needed to serve a web interface.

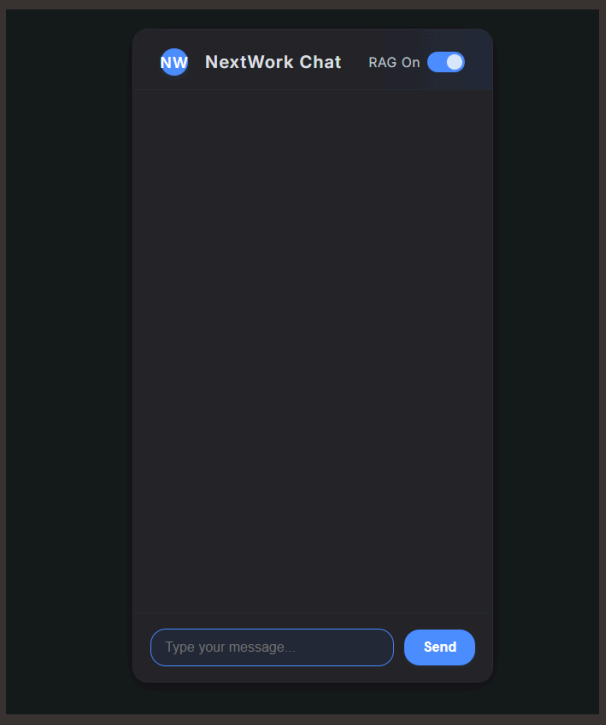


ME
Melvin J Bonner
NextWork Student

Customizing the Frontend

I want to experiment with customizing the frontend because I want to have an app that is unique and has a more professional look, tailored for sophisticated users. This is possible because as long as the API endpoints stay the same the application will continue to work.

My customized interface now features a standalone chatbot that has an uncluttered UI.



nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects