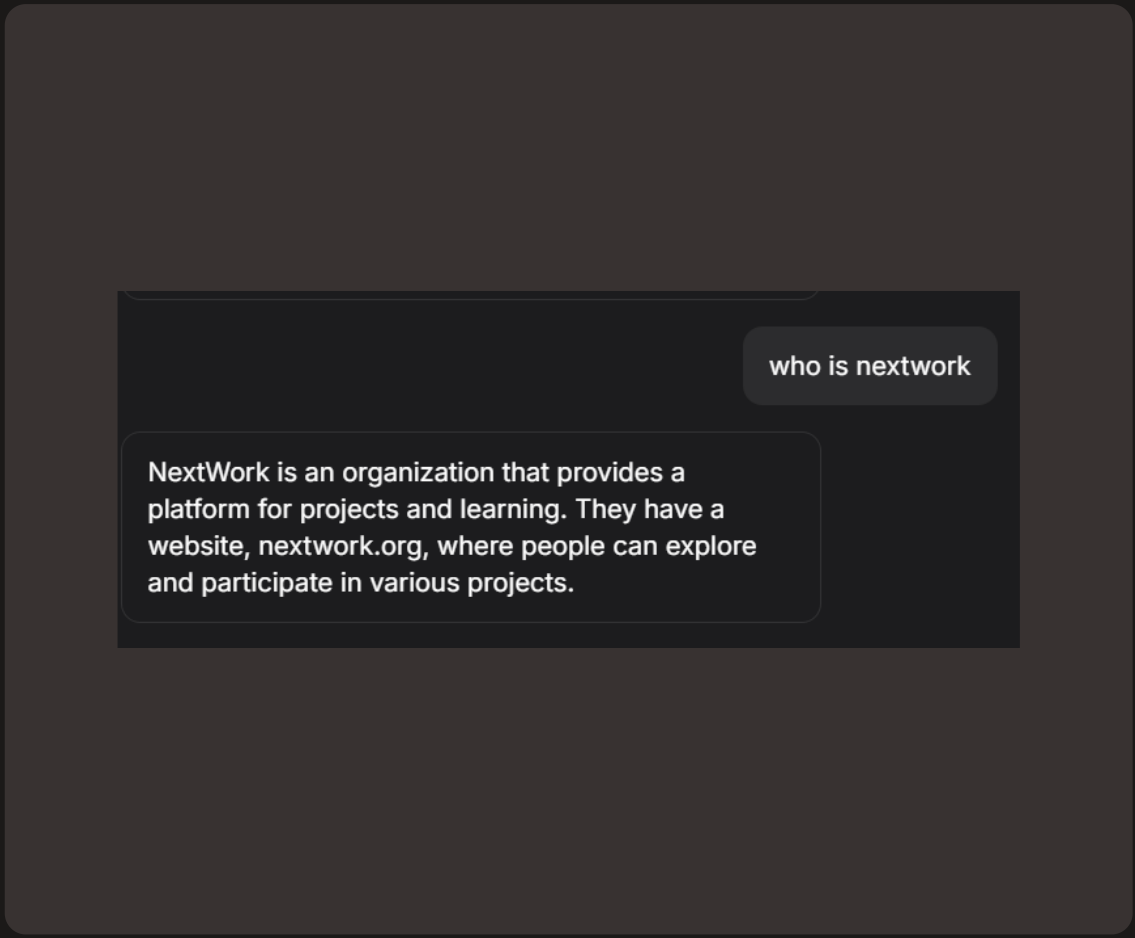
nextwork.org

Building a RAG Chatbot with a Web Interface

DI Dineshraj Dhanapathy



DI
Dineshraj Dhanapathy
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to build a full-stack AI chatbot application using Amazon Bedrock and S3 for knowledge storage, a pre-built web app for the frontend and backend, and FastAPI for API customization. I'm doing this project to learn how to integrate a Knowledge Base with a chatbot, connect it to a web app, and test both backend and frontend interactions. This hands-on setup will help me gain practical skills in deploying intelligent apps using AWS tools.

Tools and concepts

Services I used were Amazon Bedrock, S3, and FastAPI. Key concepts I learnt include how to create and use a Knowledge Base for Retrieval-Augmented Generation (RAG), how to connect frontend and backend components using API routes, and how to securely manage configuration with environment variables. I also learned how to build and customize a complete chat interface, integrate it with AWS services, and handle real-time interactions between a user and an AI model.

Project reflection

This project took me approximately 2 hours to complete. The most challenging part was debugging the API integration and ensuring the environment variables were correctly set for both Knowledge Base and direct model access. It was most rewarding to see the customized frontend working seamlessly with the backend, successfully switching between RAG and direct AI responses in real-time.

DI

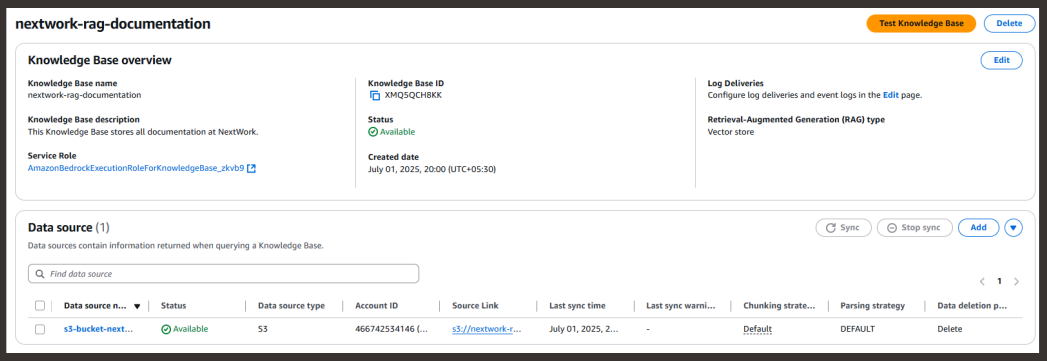
Dineshraj Dhanapathy
NextWork Student

I did this project today to deepen my practical skills in building AI-powered web apps using Amazon Bedrock and FastAPI. It helped me connect backend APIs with a custom frontend and understand how to work with both general AI models and knowledge-based responses. This project met my goals by giving me hands-on experience in API integration, frontend customization, and deploying intelligent chat interfaces.

DI
Dineshraj Dhanapathy
NextWork Student

Chatbot setup

I created a Knowledge Base to link my uploaded documents in S3 with Amazon Bedrock, enabling the chatbot to retrieve relevant information from them. This is important because it allows the AI to generate accurate, context-specific responses based on my own content, rather than relying only on general knowledge. It enhances the chatbot's usefulness for personalized or domain-specific queries.



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 fastapi helps build the API quickly, uvicorn runs the server, boto3 lets the app connect to AWS services like Bedrock and S3, and python-dotenv loads environment variables securely. Other packages like pydantic and starlette support data validation and request handling, making the app stable and production-ready.

A virtual environment helps me by creating an isolated space for this project's dependencies, so the Python packages I install won't affect other projects on my computer. This is especially useful when working with specific versions of libraries or frameworks. We need it for this project to ensure that the chatbot app runs consistently, without conflicts from global packages or other environments, making development and troubleshooting much easier.

```
(venv) dd@DD:/mnt/c/Users/dines/Downloads/RAG/nextwork-rag-webapp$ pip3 list

Package           Version
----------------- -----------
annotated-types   0.7.0
anyio             4.8.0
boto3             1.36.20
botocore          1.36.20
click             8.1.8
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
typing_extensions 4.12.2
urllib3           2.3.0
uvicorn           0.34.0
```

Breaking down the API

In our web app, we need an API because it acts as a bridge between the frontend and the Amazon Bedrock chatbot. The API takes the user's question from the web interface, sends it to the chatbot, and then returns the chatbot's response back to the user. Without the API, the frontend wouldn't be able to communicate with the AI model or the Knowledge Base. It ensures smooth, secure, and structured interaction between the user and the chatbot.

Digging deeper into the API code, the main tools we need to import are FastAPI, boto3, os, and load_dotenv. These tools help us by simplifying backend development and securely connecting our web app to Amazon Bedrock. FastAPI builds and runs the API efficiently, boto3 lets us access AWS services like Bedrock and S3, os retrieves environment variables securely, and load_dotenv loads those variables from a .env file—making our app safer, more organized, and easier to manage.

The environment variables we need are AWS_REGION, KNOWLEDGE_BASE_ID, and MODEL_ARN. We store them separately because hardcoding them into our script is a security risk—especially if the code is pushed to GitHub. Using environment variables keeps sensitive information like AWS credentials and model IDs secure and makes it easier to manage and update settings without changing the code itself.

placeholder

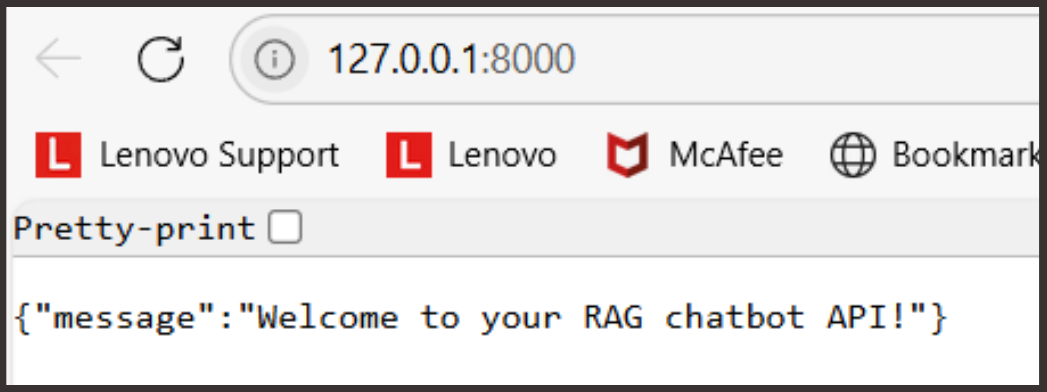
My API uses the Python AWS SDK (boto3) to connect to Amazon Bedrock and handle chatbot interactions programmatically.

The difference between the AWS CLI and an SDK is that the CLI is a command-line tool used for quick, manual tasks in the terminal, while an SDK is a library you import into your code to let applications interact with AWS services automatically. The SDK is ideal for building apps, while the CLI is great for testing or running one-off commands.

Our API has two main routes: /, which is the root route that simply confirms the API is running, and /bedrock/query, which sends user questions to the chatbot via the Knowledge Base. In general, routes help us organize different functions in our API and define how the app should respond to specific requests. They allow us to handle multiple tasks, like sending queries, switching models, or managing different chatbot configurations, all through clear and structured endpoints.

Testing the API

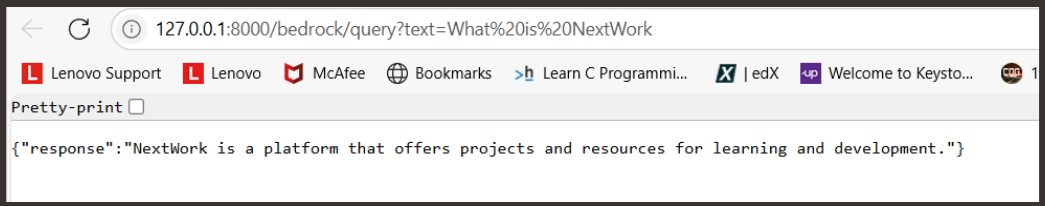
When I visited the root endpoint, I saw {"message":"Welcome to your RAG chatbot API!"}. This confirms that the FastAPI server is running successfully and the backend is correctly set up. It also shows that the application is responding to requests and ready to handle further API calls for both the Knowledge Base and direct model interactions.



Troubleshooting the API

My second endpoint had errors!

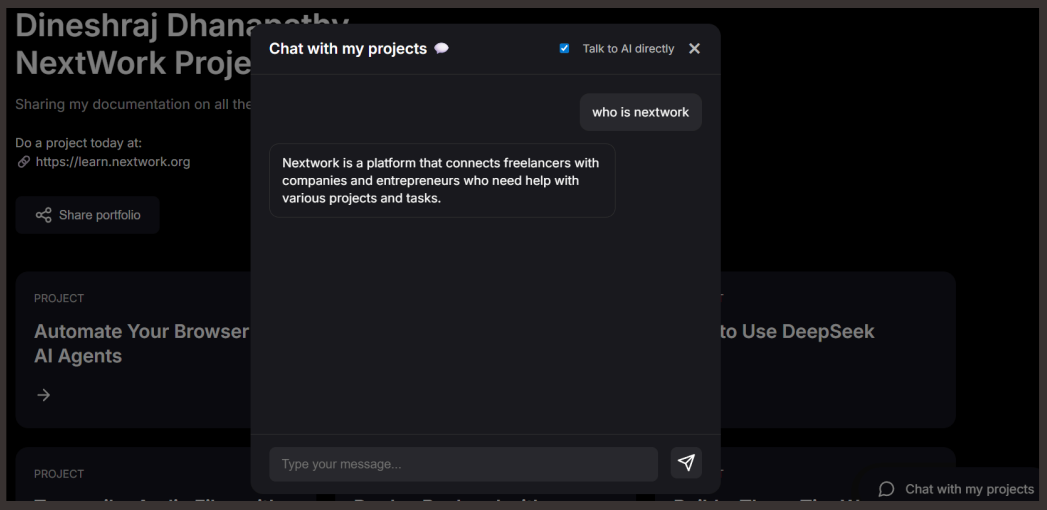
To fix the Parameter Failed error, I had to set the missing environment variables KNOWLEDGE_BASE_ID and MODEL_ARN in the .env file. These values were initially None, causing the API to fail validation. Doing this helps the API pass the correct parameters to Amazon Bedrock, allowing it to connect with the Knowledge Base and generate accurate chatbot responses.



Running the Web App

The difference between the API and the web app for the user is that the API is a backend service that processes requests and returns responses in JSON, usually accessed by developers or other applications. The web app, on the other hand, provides a user-friendly interface where users can type messages and see responses visually. While the API handles the logic, the web app delivers the full chat experience by connecting the frontend to the backend.

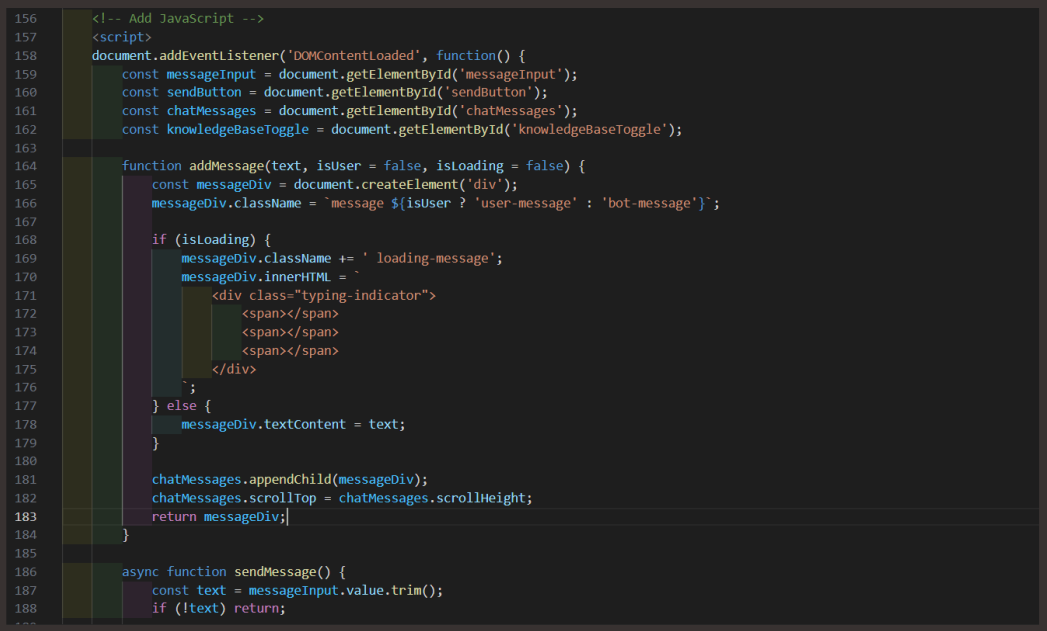
The web app has the ability to switch between using the Knowledge Base (RAG) and talking to the AI directly via the model. When I asked the same question through both modes, the RAG-enabled response pulled specific details from my uploaded documents, while the direct AI response relied on general knowledge. This functionality lets users choose between personalized, document-based answers and broader, general-purpose responses—all within a simple interface.


Chat with my projects
Talk to AI directly
who is nextwork
Nextwork is a platform that connects freelancers with companies and entrepreneurs who need help with various projects and tasks.
Type your message...
Share portfolio
Do a project today at:
https://learn.nextwork.org
PROJECT
Automate Your Browser
AI Agents
to Use DeepSeek
Chat with my projects

Breaking down the Web App

When a user sends a chat message through the web app, the frontend captures the input and sends it to the backend API using a request. The backend receives the message, either sends it to Amazon Bedrock directly or through the Knowledge Base, and gets a response from the AI model. It then returns that response to the frontend, which displays it to the user. This flow connects the UI, API routes, and AWS services to create a smooth chat experience.

web_app.py extends the API by adding a frontend layer using HTML templates and static files, allowing users to interact with the chatbot through a web interface. Unlike main.py, which only handles API logic, web_app.py serves both the API and the UI. It includes template rendering with Jinja2, static file handling, and a home route (/) that loads index.html. This makes web_app.py a full-stack app, combining backend logic with a user-friendly frontend experience.



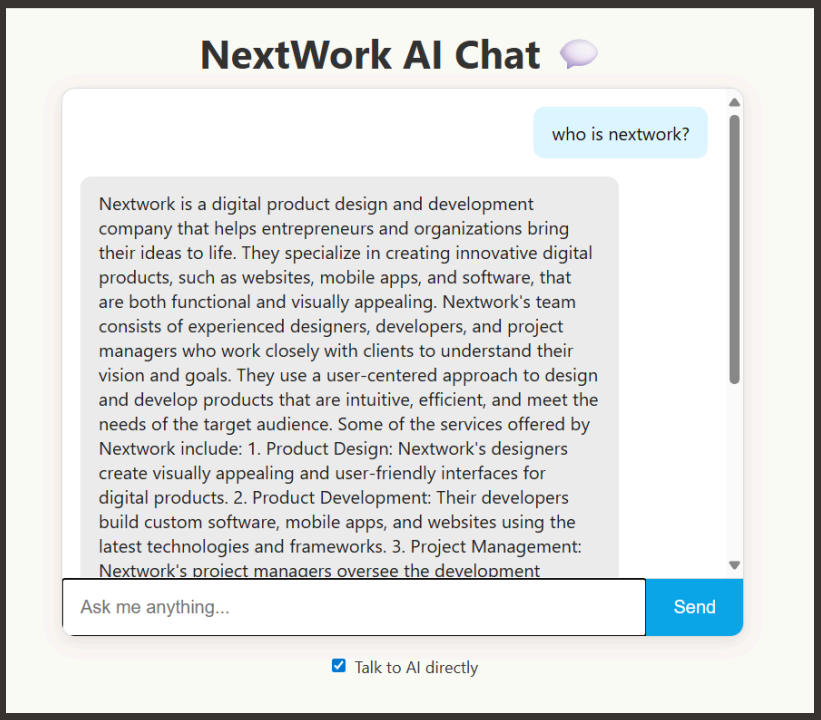
```
<!-- Add JavaScript -->
<script>
document.addEventListener('DOMContentLoaded', function() {
    const messageInput = document.getElementById('messageInput');
    const sendButton = document.getElementById('sendButton');
    const chatMessages = document.getElementById('chatMessages');
    const knowledgeBaseToggle = document.getElementById('knowledgeBaseToggle');

    function addMessage(text, isUser = false, isLoading = false) {
        const messageDiv = document.createElement('div');
        messageDiv.className = `message ${isUser ? 'user-message' : 'bot-message'}`;

        if (isLoading) {
            messageDiv.className += ' loading-message';
            messageDiv.innerHTML = `
                <div class="typing-indicator">
                    <span></span>
                    <span></span>
                    <span></span>
                </div>
            `;
        } else {
            messageDiv.textContent = text;
        }

        chatMessages.appendChild(messageDiv);
        chatMessages.scrollTop = chatMessages.scrollHeight;
        return messageDiv;
    }

    async function sendMessage() {
        const text = messageInput.value.trim();
        if (!text) return;
```

DI
Dineshraj Dhanapathy
NextWork Student

Customizing the Frontend

I want to experiment with customizing the frontend because it allows me to improve the user experience, match the design to my project's branding, and add features like input hints or styling. This is possible because as long as the API endpoints stay the same, the frontend can be updated freely without breaking the connection to the backend. It gives me flexibility to enhance usability while keeping the chatbot functionality intact.

My customized interface now features a clean, pastel-colored design with a card-style chatbox centered on the screen. It replaces the original dark-themed layout with a light, modern look. The chat messages are displayed in soft bubbles, with user messages on the right and bot responses on the left. A simple toggle lets users switch between talking to the Knowledge Base or the AI model directly. The layout is minimalist, responsive, and designed to enhance usability without changing the API functionality.


NextWork AI Chat
who is nextwork?
Nextwork is a digital product design and development company that helps entrepreneurs and organizations bring their ideas to life. They specialize in creating innovative digital products, such as websites, mobile apps, and software, that are both functional and visually appealing. Nextwork's team consists of experienced designers, developers, and project managers who work closely with clients to understand their vision and goals. They use a user-centered approach to design and develop products that are intuitive, efficient, and meet the needs of the target audience. Some of the services offered by Nextwork include: 1. Product Design: Nextwork's designers create visually appealing and user-friendly interfaces for digital products. 2. Product Development: Their developers build custom software, mobile apps, and websites using the latest technologies and frameworks. 3. Project Management: Nextwork's project managers oversee the development
Ask me anything...
Send
Talk to AI directly