Retail Granite Wiki Chatbot Setup Guide

This document provides step-by-step instructions on how to set up the Retail Granite Wiki Chatbot project from the GitHub repository. Follow the steps below to install and run both the front-end and back-end of the application.

# Clone the Repository

To get started with the project, clone the Git repository from the following URL:  
  
Git Repository URL: [https://github.com/jit4ibm/ChatBot.git](https://github.com/jit4ibm/ChatBot.git)  
  
To clone the repository, use the following command in your terminal:  
  
git clone https://github.com/jit4ibm/ChatBot.git  
cd ChatBot

# Install Dependencies

Navigate to the project folder and install all the required dependencies:

### Step 1. Set up your environment

We first need to set up our environment by fulfilling some prerequisites.

1. Install the latest version of [Ollama](https://ollama.com/) to run locally.
2. Pull the latest Granite 3.1 model by running the following command.
3. Install and import the necessary libraries and modules.  
     
    npm install

npm install web-vitals ollama axios

ollama pull granite3.1-dense:8b granite3.1-dense:2b etc.

ollama pull nomic-embed-text

# Run the Backend Server

Start the backend server:

node Retail\_bee\_wiki\_granite\_server.js  
This will run the Express server, which will be accessible on port 4000 by default.

# Start the Frontend Development Server

To start the React development server for the frontend, run:  
  
 npm start  
This will launch the React app, typically accessible on http://localhost:3000.

# Verify the Communication

Ensure that the front-end React app is correctly communicating with the back-end Express server. You can test the interaction by opening the front-end in your browser and checking if the API calls are being correctly made to the back-end.

# Check the Backend Port

Ensure the back-end is running on an appropriate port, like 4000. If necessary, adjust the port number in the backend code (e.g., in Retail\_bee\_wiki\_granite\_server.js). The default backend port is 4000.

# Environment Setup (Optional)

If the environment requires additional configurations (such as API keys, environment variables, etc.), ensure to create a .env file in the root directory. Example .env file:  
  
  
REACT\_APP\_API\_URL=http://localhost:4000  
OLLAMA\_API\_KEY=your\_api\_key  
 These can be accessed in your React app using process.env.REACT\_APP\_API\_URL.

# Test the Frontend and Backend

After setting up both the front-end and back-end servers, test if both are working. Open the React app and ensure it is making API calls to the back-end server.

# Production Build (Optional)

Once everything works, you can create a production build of the React app by running:  
  
npm run build  
This will create a static build of your front-end code, which can be served through your back-end or a different static file server.

# Troubleshooting

If you encounter any errors during the setup, ensure all dependencies are installed correctly. Check the back-end server's port and ensure no port conflicts with the front-end. If something is missing, it will likely appear as an error during npm install. To resolve missing packages, run:  
  
 npm install <missing-package-name>

To build a document-based question-answering system using Docling with Granite 3.1 and run it from a React frontend, follow these steps:

### Backend with Node.js

1. **Install Required Packages**  
   Install Docling and Granite 3.1 along with any additional dependencies:

npm install docling granite@3.1 express body-parser cors

1. **Set Up Granite and Docling**  
   Import and initialize Docling and Granite in your Node.js backend:

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const { Granite } = require('granite');

const { Docling } = require('docling');

const app = express();

app.use(cors());

app.use(bodyParser.json());

// Initialize Granite and Docling

const granite = new Granite();

const docling = new Docling();

1. **Upload and Process Documents**  
   Create endpoints for uploading documents and processing questions:

app.post('/upload', async (req, res) => {

const { documentContent } = req.body;

try {

await docling.addDocument(documentContent);

res.status(200).send('Document uploaded successfully.');

} catch (error) {

res.status(500).send('Error uploading document: ' + error.message);

}

});

app.post('/ask', async (req, res) => {

const { question } = req.body;

try {

const answer = await docling.query(question);

res.status(200).send({ answer });

} catch (error) {

res.status(500).send('Error processing question: ' + error.message);

}

});

app.listen(4000, () => {

console.log('Server running on port 4000');

});

### Frontend with React

1. **Set Up React App**  
   If you don’t have an existing React app, create one:

npx create-react-app document-qa-chatbot

cd document-qa-chatbot

npm install axios

1. **Create Components for Uploading and Asking**  
   Add components for file upload and question answering:

import React, { useState } from 'react';

import axios from 'axios';

const App = () => {

const [documentContent, setDocumentContent] = useState('');

const [question, setQuestion] = useState('');

const [answer, setAnswer] = useState('');

const uploadDocument = async () => {

try {

await axios.post('http://localhost:5000/upload', { documentContent });

alert('Document uploaded successfully!');

} catch (error) {

alert('Error uploading document: ' + error.message);

}

};

const askQuestion = async () => {

try {

const response = await axios.post('http://localhost:5000/ask', { question });

setAnswer(response.data.answer);

} catch (error) {

alert('Error asking question: ' + error.message);

}

};

return (

<div>

<h1>Document-Based QA</h1>

<textarea

placeholder="Paste your document here"

value={documentContent}

onChange={(e) => setDocumentContent(e.target.value)}

rows={10}

cols={50}

></textarea>

<button onClick={uploadDocument}>Upload Document</button>

<br />

<input

type="text"

placeholder="Ask a question"

value={question}

onChange={(e) => setQuestion(e.target.value)}

/>

<button onClick={askQuestion}>Ask</button>

<h3>Answer: {answer}</h3>

</div>

);

};

export default App;

1. **Run Both Backend and Frontend**

Start the Node.js server:

node server.js

Start the React app:

npm start

### Optional Enhancements

* **Add Authentication** for secure access.
* **Use File Uploads** for handling larger documents.
* **Integrate AI Models** if advanced question-answering capabilities are required. For instance, integrate a pretrained model compatible with Granite for semantic understanding.

Let me know if you need further guidance on specific aspects!

**Set Up the Django Project**

1. **Install Python and Django**
   * Ensure Python is installed (>=3.8). Check using:

python –version

* + Install Django and required libraries:

pip install django djangorestframework django-cors-headers granite

docling

pip install python-magic

pip install flask flask-cors docling langchain\_community langchain faiss-cpu

pip install pypdfium2 docling langchain\_community langchain faiss-cpu

1. **Create a Django Project**
   * Open VS Code and create a new folder for your project (e.g., Retail-Chatbot).
   * Open the terminal in VS Code and run:

django-admin startproject Retail\_Chatbot.

python manage.py startapp QA\_Chatbot

1. **Install VS Code Extensions**
   * Install the **Python** extension for VS Code for syntax highlighting and debugging.
   * Install the **Django** extension for project-specific features.

### ****Configure the Django Project****

1. **Update settings.py**
   * Add the following to INSTALLED\_APPS:

INSTALLED\_APPS = [

...,

'rest\_framework',

'corsheaders',

'qa',

]

* + Add corsheaders.middleware.CorsMiddleware to the top of the MIDDLEWARE list:

MIDDLEWARE = [

'corsheaders.middleware.CorsMiddleware',

...,

]

* + Allow all origins for development:

CORS\_ALLOW\_ALL\_ORIGINS = True

1. Open VS Code and create a new folder for your project
   * ff