Django+React Chatbot

Table of Contents

Overview

This document provides a comprehensive guide to the Django + React Chatbot application, detailing the project's setup, architecture, and key features. The project integrates a Django backend with a React frontend to deliver a robust and scalable chatbot solution.

Architecture

The application is built using a modern web stack. The key components and technologies used in the project are

Backend: Django (Python)
Frontend: React.js (JavaScript)
Database: SQLite
Communication: Axios for HTTP requests
Development Environment
Django Server: Runs on port 8001
React Server: Runs on port 3001
Environment Configuration: .env files are used to manage sensitive information like SECRET_KEY, AUTH_TOKEN, and API keys.

Setup Instructions

Backend (Django)
Clone the Repository: **https://dev.azure.com/abbvie-devops-lab/Abbvie%20BTS/_git/abbvie-ous-OAD-Automations**

bash

```
git clone https://dev.azure.com/abbvie-devops-lab/Abbvie%20BTS/_git/abbvie-ous-OAD-Automations cd
<project-directory>
```

Create a Virtual Environment:

bash

```
python3 -m venv env
source env/bin/activate  # On Windows: env\Scripts\activate
```

Install Dependencies:

bash

```
pip install -r requirements.txt
```

bash
Copy code

bash

```
python manage.py migrate
```

Start the Django Server:

bash
Copy code

bash

```
python manage.py runserver 8001
```

Frontend (React)

Navigate to the Frontend Directory:

bash

```
cd chatbot-app
```

Install Dependencies:

bash

```
npm install
```

Start the React Server:

bash

```
npm start
```

The React application will run on port 3001.

Key Features:

Chatbot Functionality:

Users can interact with the chatbot through the React frontend, with messages processed by the Django backend.

Source Management:

Users can create, update, and delete sources that the chatbot can use for information retrieval.

Document Upload:

Supports uploading documents to specific sources, with the ability to fetch and display them.

API Endpoints

/chatbot1/search/: Handles search queries.
/chatbot1/create-source/: Manages the creation of new sources.
/saml2/login/: Manages SAML-based authentication.
/upload-document/: Handles document uploads.

React Components
Chatbot: Main component for the chatbot interface.
CreateSource: Component to create new sources.
ListSources: Displays all available sources.
UploadDocument: Allows users to upload documents to a selected source.

Security:

Below is the breakdown of different backend components:

urls.py : This file maps URL paths to specific view functions that handle various operations within the application.

urls.py Overview
The urls.py file in your Django project is responsible for mapping URL patterns to specific views, enabling the application to handle various user requests. Below is a brief description of the URL patterns configured in the file:

Search Functionality:

search/: Maps to the api_search view, which handles the main search functionality within the application.
/: Maps to the search view, rendering the main search page.

Source Management:

create-source/: Maps to the create_source view, allowing users to create new sources within the application.
sync-source/<str:source>/: Maps to the sync_source view, enabling synchronization of data for a specified source.

Document Management:

upload-document/<str:source_name>/: Maps to the upload_document view, facilitating the upload of documents to a specific source.
get-documents/<str:source>/: Maps to the get_documents view, allowing retrieval of documents associated with a particular source.
list-documents/<str:source>/: Maps to the list_documents view, which lists documents under a specific source.

Source and Document Listing:

list-sources/: Maps to the list_sources view, which provides a list of all sources within the application.
list-documents/<str:source>/: Maps to the list_documents view, displaying documents related to a specific source.

Upload Status and Document Processing:

check-upload-status/<str:source>/<str:task_id>/: Maps to the check_upload_status view, checking the status of document uploads for a specific source.
process-documents/: Maps to the trigger_document_processing view, triggering the processing of documents within the application.
auto-upload/: Newly added, it also maps to the trigger_document_processing view, likely to automate the upload process.
start-scheduler/: Maps to the start_scheduler view, initiating a scheduled process, possibly for automation tasks.
get-upload-status/: Maps to the get_upload_status view, retrieving the current upload status within the application.

Usage

These URL patterns allow users to interact with various aspects of the application, from searching and managing sources to uploading and processing documents. Each view is linked to a specific functionality, ensuring that the application handles requests efficiently and effectively.

views.py
The views.py file contains the logic for each endpoint defined in urls.py. Here's an overview of the key functions:

list_sources: Retrieves a list of sources from an external API and returns them in JSON format.
list_documents: Fetches documents associated with a specific source from an external API and returns them in JSON format.
upload_document: Handles the upload of documents to a specific source and processes them.
check_upload_status: Checks the status of a document upload task and returns the result.
process_document: Simulates document processing and generates metadata for indexing.
update_source_documents: Updates the list of documents for a given source locally.
upload_to_api: Uploads the processed document to an external API.

5. **API Integration**

External API Calls: The application integrates with an external API to manage sources and documents. API keys and tokens are securely managed using encryption with the cryptography library.
6. Security
API Key Management: API keys are encrypted using Fernet and stored in environment variables, ensuring that sensitive information is protected.