

PROTOTYPE DOCUMENTATION

1. Multiple Repository Insertion from Azure DevOps

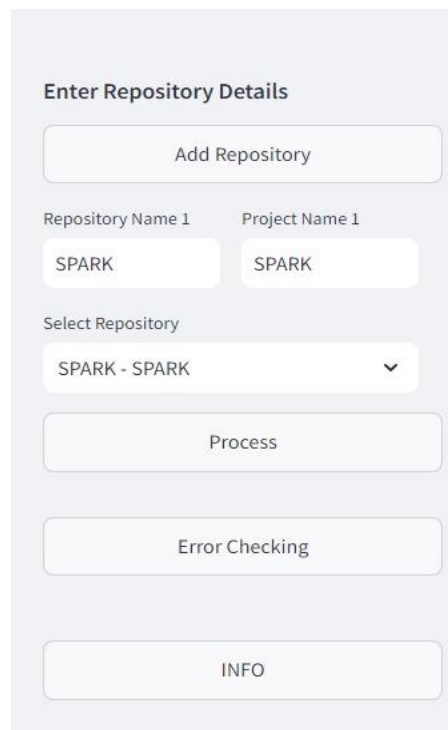
Description:

This feature allows developers to insert multiple repositories from Azure DevOps. Users can add the project and repository names through a form interface, enabling the bot to access and search across various repositories.

Steps:

1. Open the chatbot interface.
2. Navigate to the "Add Repository" form.
3. Enter the project name and repository name.
4. Click "Add Repository."
5. Repeat the process to insert multiple repositories as needed.

Image:



The image shows a mobile application interface for adding repository details. The form is titled "Enter Repository Details" and contains the following elements:

- An "Add Repository" button at the top.
- Two input fields: "Repository Name 1" with the value "SPARK" and "Project Name 1" with the value "SPARK".
- A "Select Repository" dropdown menu showing "SPARK - SPARK" with a downward arrow.
- A "Process" button.
- An "Error Checking" button.
- An "INFO" button at the bottom.

2. Generating Code Snippets from Selected Repo

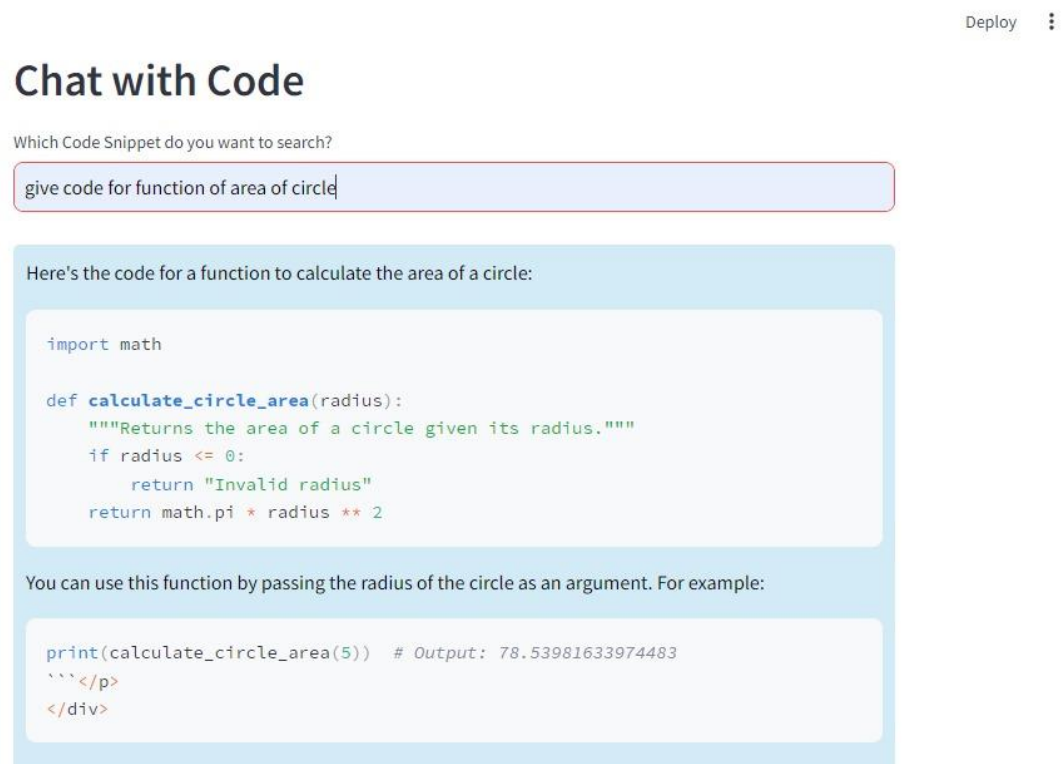
Description:

This feature enables the chatbot to retrieve relevant code snippets from a selected repository based on queries, keywords, or specific functions. The user can ask for particular features, and the bot will return relevant code.

Steps:

1. Select the repository from which you want to generate code snippets.
2. Enter your query, keywords, or function names related to the required code.
3. Submit the query to the bot.
4. Review the code snippets returned by the bot from the selected repository.

Image:



3. Company's Custom Wrappers and Standard Practices

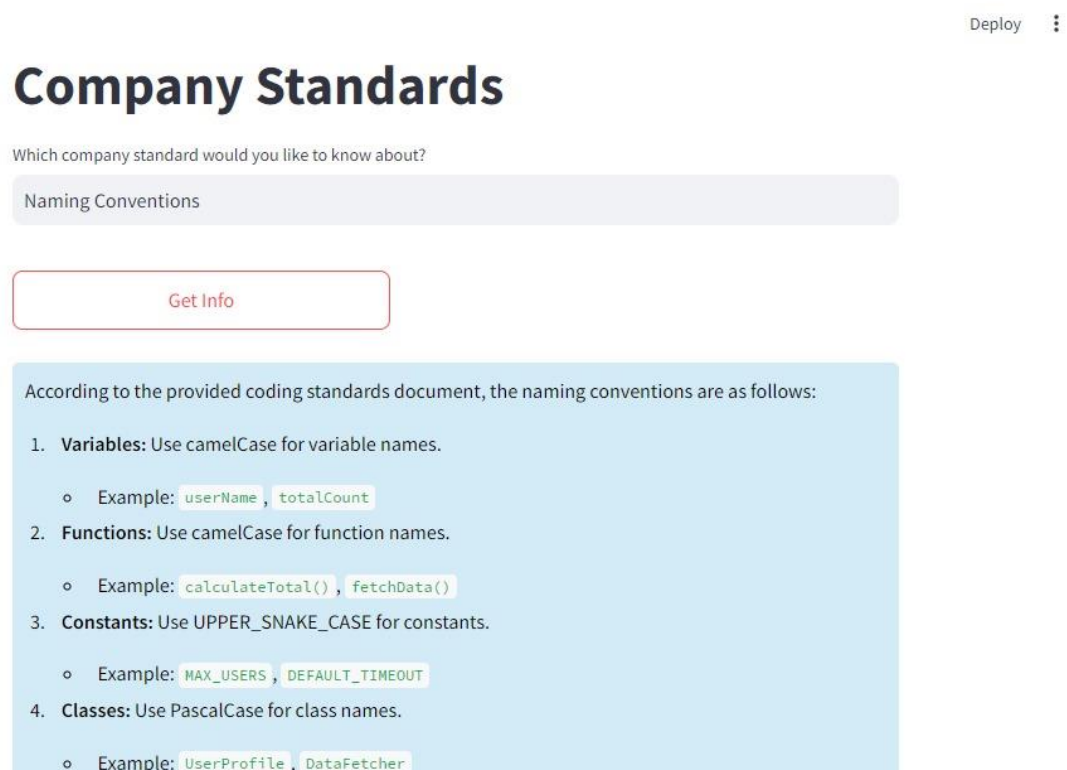
Description:

The bot retrieves code snippets that follow the company's specific wrappers and coding standards. This ensures that all code recommendations are in line with internal guidelines.

Steps:

1. Submit a query related to a function or code feature.
2. The bot will automatically apply the company's custom wrappers and coding standards when retrieving code snippets.
3. Review the code, which will adhere to the established standards and wrappers.

Image:



4. Error-Solving Abilities

Description:

This feature provides the capability to solve errors by returning error-resolved code snippets or guidance based on the developer's query.

Steps:

1. Enter the error or issue you're facing in the query input.
2. Submit the query to the bot.
3. The bot will return error-resolved code snippets or explanations that address the issue.

Image:

Error Checking

Enter your code here to check for errors:

```
def add_numbers(a, b):  
    return a + b  
  
# Calling the function with missing argument
```

Check for Errors

There are two potential errors in this code:

1. The function `add_numbers` is called with only one argument, but it requires two arguments. This will raise a `TypeError` with the message "add_numbers() missing 1 required positional argument: 'b'".
2. Even if the function call was correct, the `print` statement would still raise a `TypeError` because it's trying to concatenate a string with an integer. The `result` variable is an integer, and you can't concatenate integers with strings using the `+` operator.

Here's how you can fix these errors:

5. Feedback and User Interaction

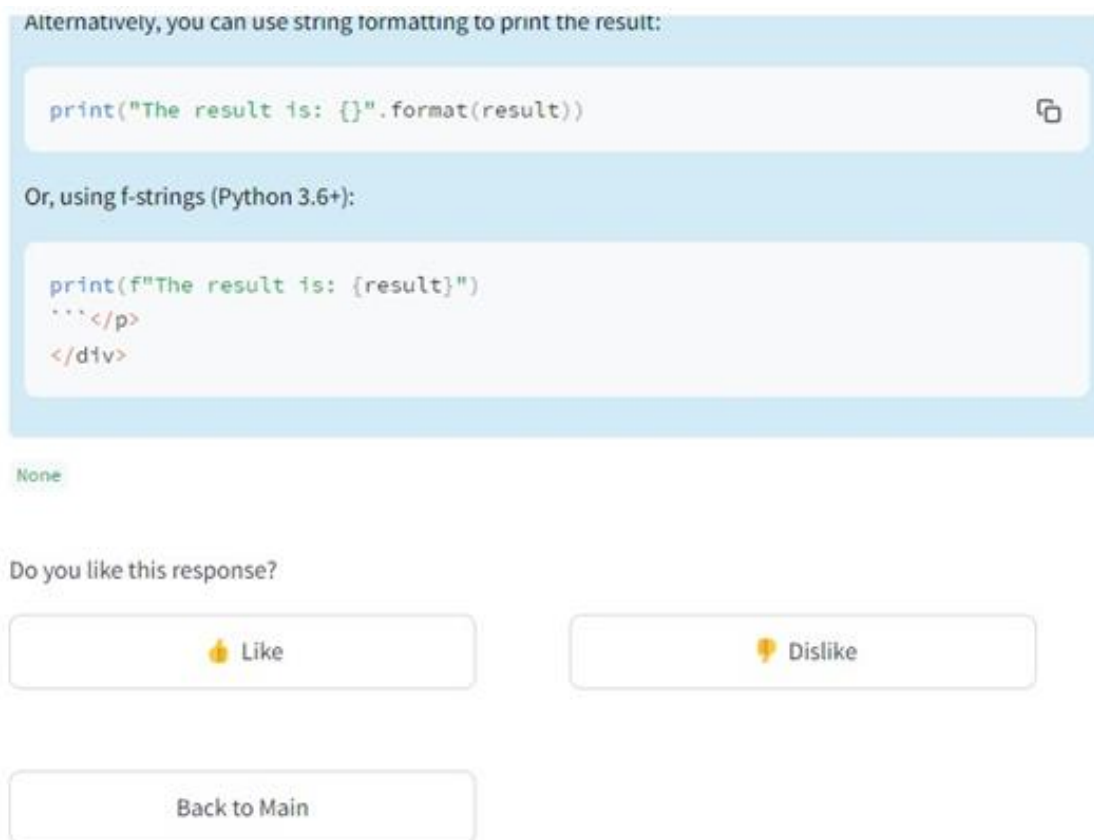
Description:

Developers can provide feedback on the retrieved code snippets. Users can like or dislike the responses, which helps improve future accuracy and relevance based on the feedback.

Steps:

1. Review the code snippet provided by the bot.
2. If the snippet is useful, click the "Like" button. If not, click "Dislike."
3. Feedback will be used to enhance future responses, improving the bot's learning and search algorithms.

Image:



6. Personalized Code Assistance

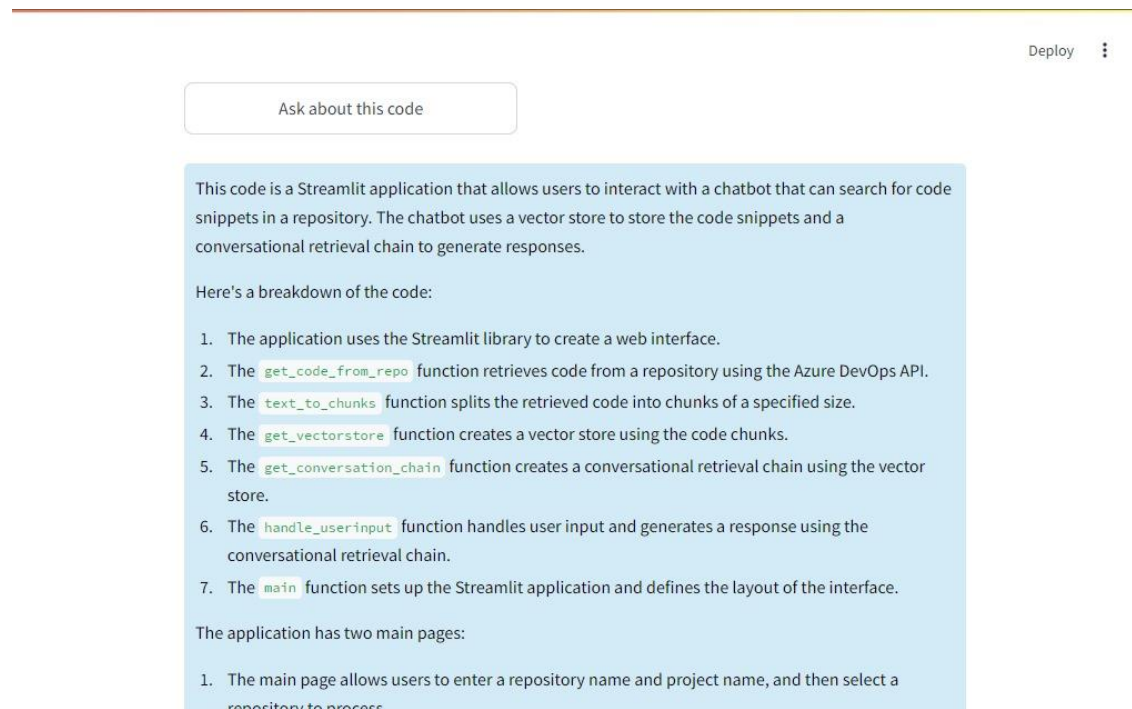
Description:

Developers can also add files from their local device and can analyze, display and generate about it's descriptions.

Steps:

1. Uploading Code File from Local Device
2. Analyzing the Uploaded Code
3. Displaying Uploaded Code
4. Generating Descriptions for Code Snippets

Image:



[Ask about this code](#)

This code is a Streamlit application that allows users to interact with a chatbot that can search for code snippets in a repository. The chatbot uses a vector store to store the code snippets and a conversational retrieval chain to generate responses.

Here's a breakdown of the code:

1. The application uses the Streamlit library to create a web interface.
2. The `get_code_from_repo` function retrieves code from a repository using the Azure DevOps API.
3. The `text_to_chunks` function splits the retrieved code into chunks of a specified size.
4. The `get_vectorstore` function creates a vector store using the code chunks.
5. The `get_conversation_chain` function creates a conversational retrieval chain using the vector store.
6. The `handle_userinput` function handles user input and generates a response using the conversational retrieval chain.
7. The `main` function sets up the Streamlit application and defines the layout of the interface.

The application has two main pages:

1. The main page allows users to enter a repository name and project name, and then select a repository to process