PUNCHBIZ

CHATBOT

TEAM MEMEBERS:

RAVINDRA RATHICK S
SUBHASRI V
ABIRAMI D
DHARSHINI R

1.INTRODUCTION:

Our rule-based eLearning chatbot, powered by Rasa, is designed to provide a comprehensive list of available courses. The chatbot offers immediate access to a categorized list of domains and the respective courses within each domain, making it easy to explore and find courses that suit your interests. It includes interactive buttons for easy navigation, ensuring a seamless user experience.

2.RASA MODEL:

Rasa is an open-source framework for building conversational AI, offering powerful tools for creating chatbots that understand and respond to user inputs. In this project, we use Rasa's rule-based approach, which relies on predefined rules to manage dialogue and responses. This method ensures consistent and reliable interactions by following explicit guidelines for handling different user intents. By employing a rule-based model, we can effectively control the conversation flow and provide information tailored to our eLearning platform.

3.ENVIRONMENT SETUP:

3.1 Creating a Virtual Environment:

A virtual environment is crucial for isolating the project's dependencies from the global Python environment. Use Anaconda Prompt to create and activate a new environment, ensuring Python version 3.8.13 is used.

- **Create a New Environment:**

Use Anaconda Prompt to create a new environment with Python version 3.8.13.

```
conda create -n myenv python=3.8.13
```

- **Activate the Environment:**

After creating the environment, activate it using Anaconda Prompt.

```
conda activate myenv
```

3.2 Installing Project Dependencies

With the virtual environment activated in Anaconda Prompt, install the necessary Python packages:

- **Install Rasa:**

Rasa is essential for building and managing the chatbot. Install it using pip by using the following command.

```
pip install rasa
```

- **Install Flask:**

Flask is used to create the web interface for user interaction with the chatbot. This also can be installed by using pip.

```
pip install flask
```

4. COMPONENTS:

- **NLU Model:**

  Trained to recognize intents and entities from user messages. This model is crucial for understanding and classifying user inputs accurately.

- **Core Model:**

  Handles dialogue management using stories and rule policies. It determines the chatbot’s responses and actions based on the context and predefined rules.

- **Domain File:**

  Defines the intents, entities, slots, responses, and actions used by the chatbot. It serves as the central configuration for the chatbot’s behaviour.

- **Actions:**

  Include both standard and custom actions. Custom actions are used for complex logic and integrations, such as querying databases or interacting with external systems.

- **Interactive Buttons:**

  Enhance user experience by providing easy navigation and quick access to relevant information, improving overall engagement with the chatbot.

5. WORKFLOW OF CHATBOT

5.1 Intents and Entities

- **Intents:**

  Represent the purpose behind a user's message, such as "ask_course_details", "greet". They help the chatbot understand what the user wants to achieve.

- **Entities:**

  Specific pieces of information extracted from the user's message. For example, "course_name" in the query "Tell me about the Python course" is an entity that specifies which course the user is interested in.

5.2 Training Data

- **NLU Training Data:**

  Contains example sentences for each intent, with annotations for entities. This data helps the chatbot learn how to recognize and classify user inputs accurately.

  The NLU training data is typically stored in a file named “nlu.yml”. This file contains various intents and associated example sentences that help the NLU model learn to classify user inputs correctly and extract relevant information.

```yaml
version: "3.1"

nlu:

- intent: greet
  examples: |
    - Hi
    - Hello
    - Hey there
    - Greetings
    - Good morning
    - Good afternoon
    - What's up

- intent: goodbye
  examples: |
    - Bye
    - Thank you
    - TQ
    - Sure
    - Goodbye
    - See you later
    - Have a great day
    - Farewell
    - Take care
    - Until next time

- intent: course_info
  examples: |
    - List of available courses
    - What are the courses do you offer?
    - Tell me about your course offerings
    - Available courses
    - Courses provided
    - Courses offered
    - Information about course curriculum
    - Courses available


- intent: web_development_topics
  examples: |
    - Can you outline the topics covered in the [Web Development
Course](course_name) course?
    - What topics are included in your [Web Development](course_name)
curriculum?
    - I'm interested in the specifics of the [Web Development](course_name)
course content.
    - What are the main areas of focus in the [Web Development](course_name)
course?
    - Could you share the syllabus for the [Web Development](course_name)
course?
    - Topics in the [Web Development](course_name) course
    - [Web Development](course_name)
```

- **Stories:**

  Provide example interactions showing how the bot should respond to different intents and sequences of user inputs. They guide the chatbot's behaviour and ensure appropriate responses based on the context of the conversation.

  The stories are usually stored in a file named "stories.yml". Each story outlines a series of steps that include user intents and the corresponding actions the chatbot should take in response. This helps the chatbot understand and manage the context of a conversation effectively.

```yaml
version: "3.1"

stories:
- story: greet and course information
  steps:
  - intent: greet
  - action: utter_greet
  - intent: course_info
  - action: utter_course_info

- story: provide web development topics
  steps:
  - intent: course_info
  - action: utter_course_info
  - intent: web_development_topics
  - action: utter_web_development_topics
```

5.3 Rule Policies

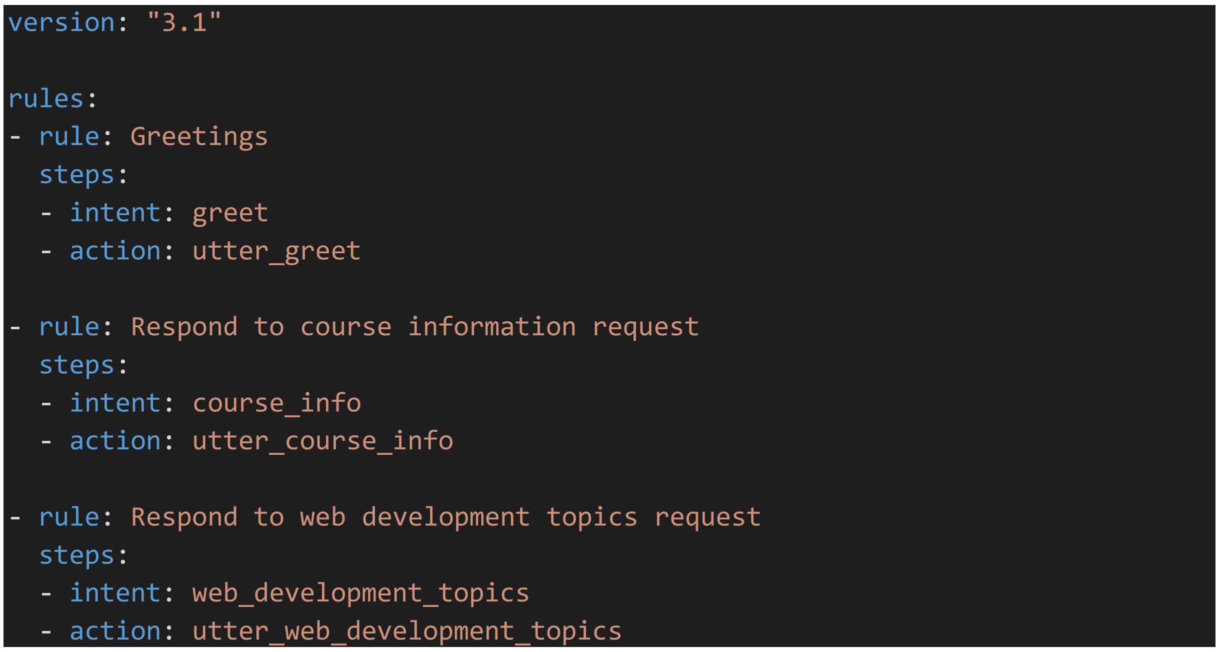
- **Rule Policies:**

  Define specific rules for how the chatbot should respond to certain intents or entities. These rules handle predictable user inputs in a straightforward manner/

Rule policies are defined in the “rules.yml file”. This file outlines the various rules and their corresponding actions, ensuring that the chatbot follows a clear and consistent interaction pattern.

- **Rule Examples:**

If the user greets the bot, the rule might specify that the bot responds with a predefined greeting message, such as "Hello! How can I assist you today?"



```yaml
version: "3.1"

rules:
- rule: Greetings
  steps:
  - intent: greet
  - action: utter_greet

- rule: Respond to course information request
  steps:
  - intent: course_info
  - action: utter_course_info

- rule: Respond to web development topics request
  steps:
  - intent: web_development_topics
  - action: utter_web_development_topics
```

5.4 Domain Definition

- **Domain File:**

Specifies the intents, entities, actions, responses, and slots used by the chatbot. It serves as the configuration for the chatbot's behaviour.

The domain file is typically named “domain.yml”. It serves as a blueprint for the chatbot, outlining all the elements required for it to function and respond accurately to user interactions.

- **Slots:**

Store information that the bot needs to remember during the conversation, such as the user’s selected course or preferred learning mode. Slots help the bot maintain context and provide relevant responses.

```
version: "3.1"

intents:
  - greet
  - goodbye
  - course_info
  - web_development_topics
  - data_science_topics
  - mobile_development_topics
  - programming_languages_topics
  - game_development_topics
  - database_design_development_topics
  - software_testing_topics
  - software_engineering_topics
  - software_development_tools_topics
  - no_code_development_topics
  - Beginners_javascript
  - Advance_javascript
  - Beginners_React JS
  - Advance_React JS
  - Beginners_Machine Learning
  - Advance_Machine Learning
  - Beginners_Python
  - Advance_Python
  - Beginner_Google Flutter
  - Advance_Google Flutter
  - Beginner_Android Development
  - Advance_Android Development
  - Beginner_java
  - advanced_java
  - Beginner_C++
  - advanced_C++
```

```yaml
  - Beginners_flutter_flow
  - Advance_flutter_flow
  - Beginners_web_design
  - Advance_web_design
  - Beginners_docker
  - Advance_docker
  - Beginners_Github
  - Advance_Github
  - Beginners_postman
  - Advance_postman
  - Beginners_api_testing
  - Advance_api_testing
  - Beginners_automation_testing
  - Advance_automation_testing
  - Beginners_data_structure
  - Advance_data_structure
  - Beginners_microservices
  - Advance_microservices
  - Beginner_MySQL
  - Advance_MySQL
  - Beginner_MongoDB
  - Advance_MongoDB
  - Beginner_2D Game Development
  - Advance_2D Game Development
  - Beginner_Unity
  - Advance_Unity
  - Fallback




entities:

  - course_name:
      type: text
  - course_level:
      type: text
slots:
   course_name:
      type: text
      influence_conversation: true
      mappings:
      - type: from_entity
        entity: course_name
   course_level:
      type: text
      influence_conversation: true
      mappings:
      - type: from_entity
```

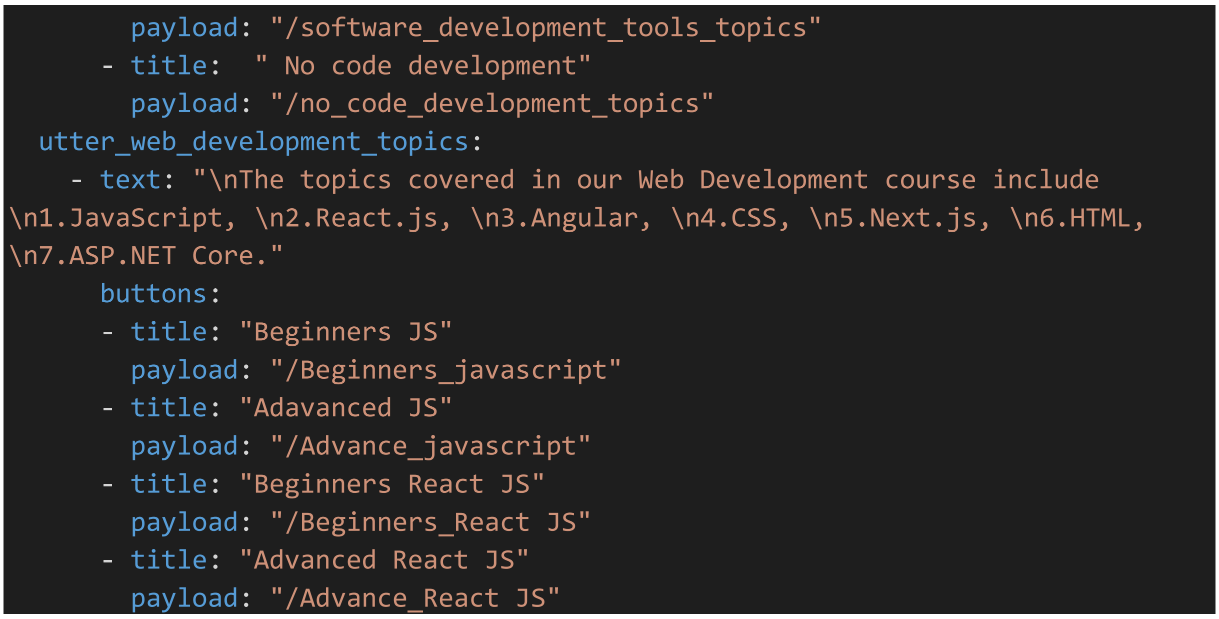
```yaml
        entity: course_level

responses:
  utter_greet:
    - text: "Hello! How can I assist you today?"

      buttons:
      - title: "course list"
        payload: "/course_info"
  utter_goodbye:
    - text: "Bye! If you have any other questions, feel free to ask."
    - text: "See you later! Don't hesitate to return if you need help."
    - text: "Bye! It was a pleasure helping you today."
    - text: "See you later! I'm always here, ready to assist whenever you
decide to return."
    - text: "Take care! Remember, I'm here to help whenever you need."

  utter_course_info:
    - text: "We offer a variety of courses like \n1.Web Development,\n2.Data
Science, \n3.Programming Languages, \n4.Database Design and Development,
\n5.Software Engineering, \n6.Mobile Development, \n7.Game Development,
\n8.Software Testing , \n9.Software Development Tool, \n10.No-Code
Development."
      buttons:
      - title: "Data Science"
        payload: "/data_science_topics"
      - title: "Web Development"
        payload: "/web_development_topics"
      - title: "Programming languages"
        payload: "/programming_languages_topics"
      - title: "Software Engineering"
        payload: "/software_engineering_topics"
      - title: "Database design and development"
        payload: "/database_design_development_topics"

    - text: "We offer courses in areas like \n1.Data Science, \n2.Programming
Langugaes, \n3.Web Development, \n4.Database Design and Development,
\n5.Software engineering, \n6.Mobile Development, \n7.Game Development,
\n8.Software Testing , \n9.Software Development Tool, \n10.No-Code Development
."
      buttons:
      - title: "Mobile development"
        payload: "/mobile_development_topics"
      - title: "Game development"
        payload: "/game_development_topics"
      - title: "Software testing"
        payload: "/software_testing_topics"
      - title: "Software development tools"
```



```
        payload: "/software_development_tools_topics"
      - title:  " No code development"
        payload: "/no_code_development_topics"
  utter_web_development_topics:
    - text: "\nThe topics covered in our Web Development course include 
\n1.JavaScript, \n2.React.js, \n3.Angular, \n4.CSS, \n5.Next.js, \n6.HTML, 
\n7.ASP.NET Core."
      buttons:
      - title: "Beginners JS"
        payload: "/Beginners_javascript"
      - title: "Adavanced JS"
        payload: "/Advance_javascript"
      - title: "Beginners React JS"
        payload: "/Beginners_React JS"
      - title: "Advanced React JS"
        payload: "/Advance_React JS"
```

5.5 Custom Actions

- **Custom Actions:**

  Perform specific tasks that go beyond standard responses, such as querying a database or integrating with external APIs.

- **Example:**

  When a user asks about a specific course, a custom action retrieves course details from the database and returns this information to the user.

5.6 Configuration File

The configuration file in Rasa, named “config.yml”, specifies the NLU pipeline and dialogue policies used by the chatbot. The NLU pipeline consists of various components that process user inputs to understand their meaning and extract relevant information. Dialogue policies determine how the chatbot should respond to different user intents based on the context of the conversation.

The configuration file defines which components to use in the NLU pipeline and how to manage the dialogue. It ensures that the chatbot processes user inputs efficiently and maintains coherent and contextually appropriate interactions.

5.7 E-Learning Chatbot:

1. **User Input:**
   - **User:** "Tell me about the Python course."
2. **Intent Recognition:**
   - The bot identifies the intent as "ask_course_details" and extracts the entity "course_name" with the value "Python."
3. **Rule Matching:**
   - The bot checks the rules to determine the appropriate action for the "ask_course_details" intent with the "course_name" entity. For example, it may have a rule that specifies how to handle queries about course details.
4. **Custom Action Execution:**
   - The bot calls a custom action to fetch details about the Python course from the database or external API.
5. **Response Generation:**
   - The bot generates and sends a response with the course details, such as: "The Python course covers basic to advanced topics and is 10 weeks long."

6.FRONTEND:

In this project, we have utilized HTML, CSS, and JavaScript to create an engaging and interactive frontend for our eLearning chatbot. These technologies work together to structure the web page, style the interface, and enable dynamic interactions, providing users with a seamless and intuitive experience.

7. INTEGRATION:

Integrating Flask with a Rasa chatbot allows you to create a web service that facilitates communication between users and the chatbot through HTTP requests. Flask is a lightweight Python web framework that excels in building web applications and APIs. Its simplicity and flexibility make it ideal for creating a server that can handle user interactions with the chatbot.

7.1 Integration Process

1. Setting Up Flask:

- **Installation:** Flask is installed via pip and set up to create a web server.
- **Application Setup:** A Flask application is created with routes that handle incoming HTTP requests.

2. Connecting to Rasa:

- **Model Loading:** The Flask application loads the Rasa models (both NLU and dialogue models) to process user inputs and generate responses.
- **Endpoint Creation**: A Flask route (e.g., /webhook) is defined to receive POST requests from users. The

route sends user messages to Rasa and returns the chatbot's responses.

3. Running the Servers:

- **Rasa Server**: The Rasa server is started with API support, allowing it to handle requests from the Flask application.
- **Flask Server:** The Flask server is started, listening for requests and forwarding them to Rasa.

```python
from flask import Flask, render_template, request, jsonify
from rasa.core.agent import Agent
import asyncio
import logging

# Initialize Flask app
app = Flask(__name__)

# Setup logging
logging.basicConfig(level=logging.DEBUG)

# Load the Rasa model
try:
    rasa_model = Agent.load(r"models\20240727-201032-hot-bay.tar.gz")
except Exception as e:
    logging.error("Error loading Rasa model: %s", str(e))
    rasa_model = None

@app.route('/health')
def health_check():
    return 'Frontend is connected to the backend.'

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/send_message', methods=['POST'])
async def send_message():
    if not rasa_model:
        logging.error("Rasa model is not loaded.")
        return jsonify({'bot_response': 'Rasa model is not loaded.'}), 500

    try:
        user_message = request.json.get('message')
        logging.debug("Received message: %s", user_message)
        # Handle text using Rasa model
        loop = asyncio.get_event_loop()
        response = await loop.run_in_executor(None, rasa_model.handle_text,
user_message)

        # Extract buttons from Rasa response
        buttons = response[0].get('buttons', []) if response else []

        # Extract text from Rasa response
        bot_response = response[0]['text'] if response else "Sorry, we don't
have that course yet. We will add it later or try with the full name."

        # Check if the bot response indicates course unavailability
        course_unavailable = "action_course_unavailable" in bot_response

        # If course is unavailable, trigger the custom action
        if course_unavailable:
            action_response = await loop.run_in_executor(None,
rasa_model.handle_text, "/action_course_unavailable")
            bot_response = action_response[0]['text'] if action_response else
"Sorry, we don't have that course yet. We will add it later or try with the
full name."
            buttons = action_response[0].get('buttons', [])

        # Send the response along with course availability flag
        return jsonify({'bot_response': bot_response, 'buttons': buttons,
'course_unavailable': course_unavailable})
    except Exception as e:
        logging.error("Error processing message: %s", str(e))
        return jsonify({'bot_response': 'An error occurred while processing
your message.'}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

7.INTERACTIVE BUTTONS FOR NAVIGATION:

Interactive buttons are designed to present users with actionable options directly within the chatbot interface. They allow users to select from predefined choices, which simplifies interactions and helps guide users through the conversation flow. This approach eliminates the need for users to type out queries and enables them to quickly obtain the information they are seeking.

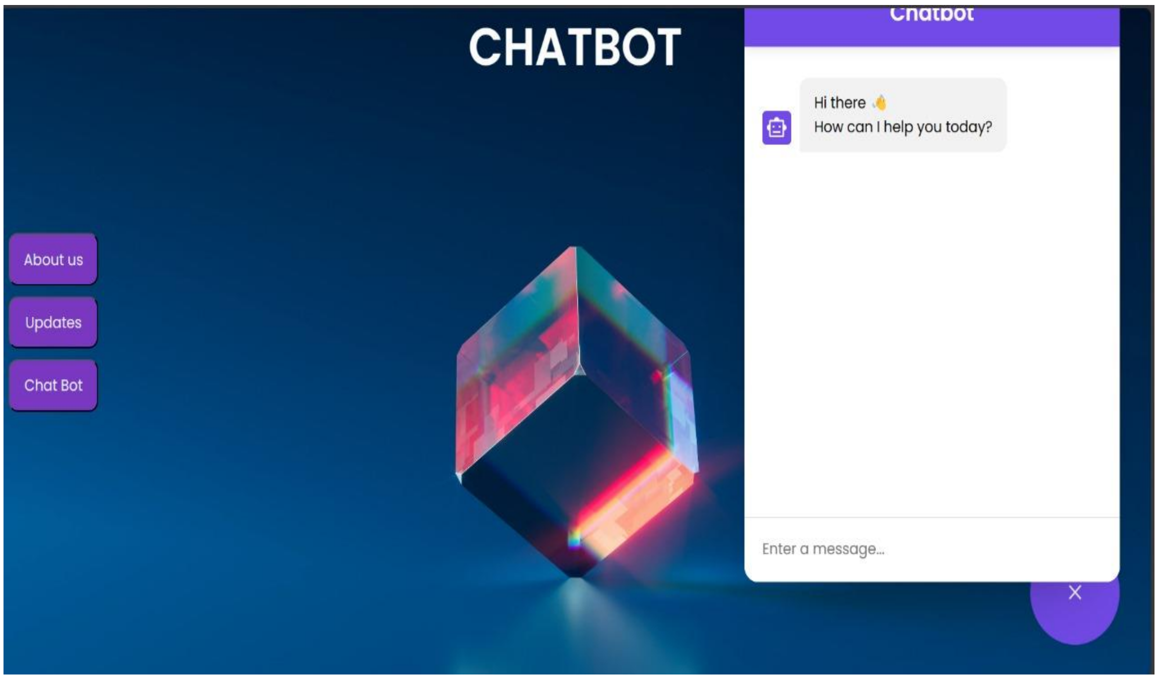
Button Functionality:

When users request information about available courses, the chatbot responds with a list of domains related to the courses offered. Each domain is presented as an interactive button, allowing users to select the domain they are interested in.

Once a user selects a domain, the chatbot provides a list of specific courses available within that domain. The courses are displayed as interactive buttons, allowing users to choose a particular course for more details.

8.IMAGES OF CHATBOT WORKING:

CHATBOT HOMEPAGE:

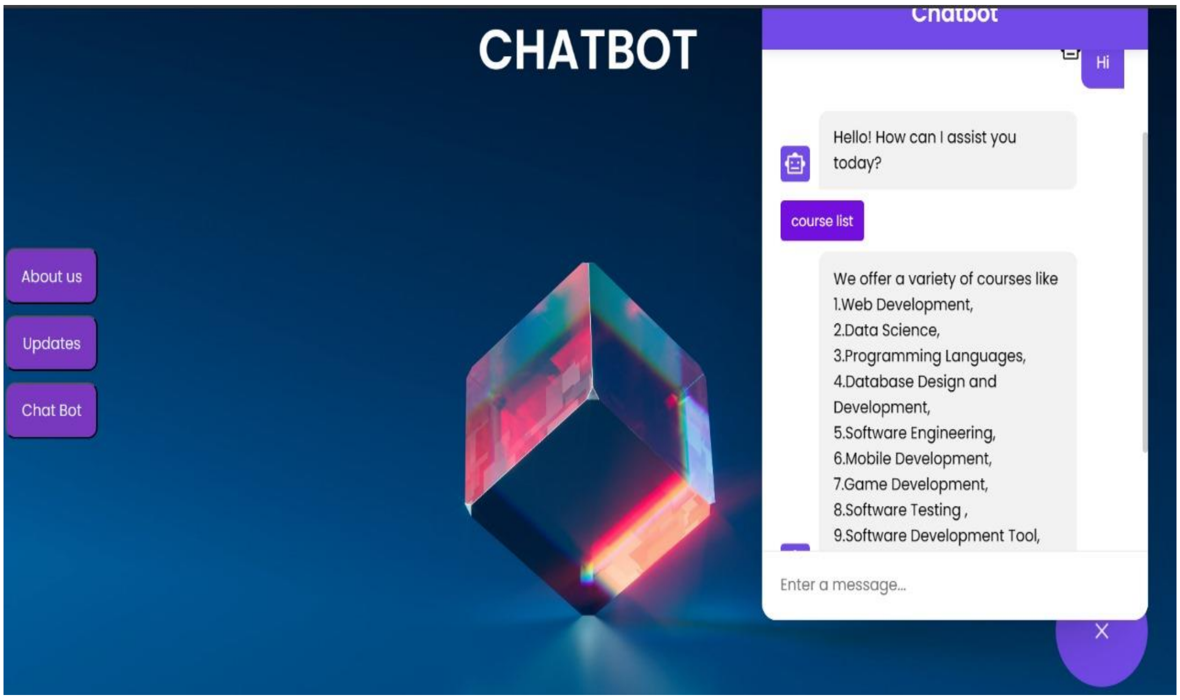


Hi there 👋
How can I help you today?

Description:

This image shows the homepage of the chatbot, where users can initiate interactions and access various features of the virtual assistant.

EXPLORE COURSES WITH CHATBOT:



Hi

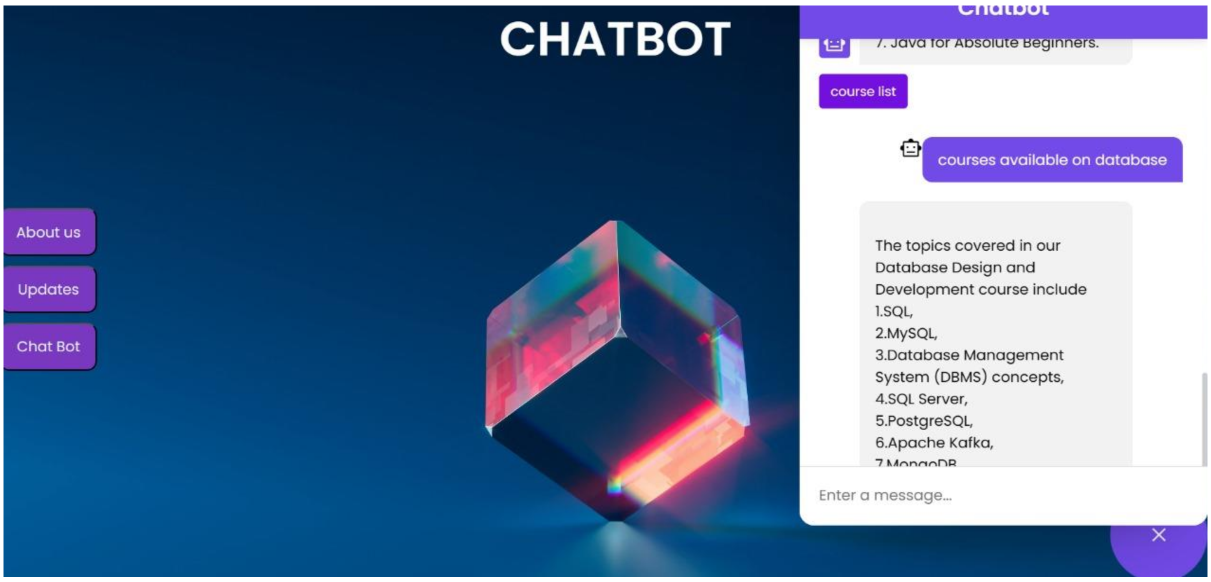
Hello! How can I assist you today?

We offer a variety of courses like
1.Web Development,
2.Data Science,
3.Programming Languages,
4.Database Design and Development,
5.Software Engineering,
6.Mobile Development,
7.Game Development,
8.Software Testing ,
9.Software Development Tool,

Enter a message...

Description:

This image demonstrates the chatbot's functionality in providing course-related information to users. It likely includes options or a menu for exploring different courses.



7. Java for Absolute Beginners.

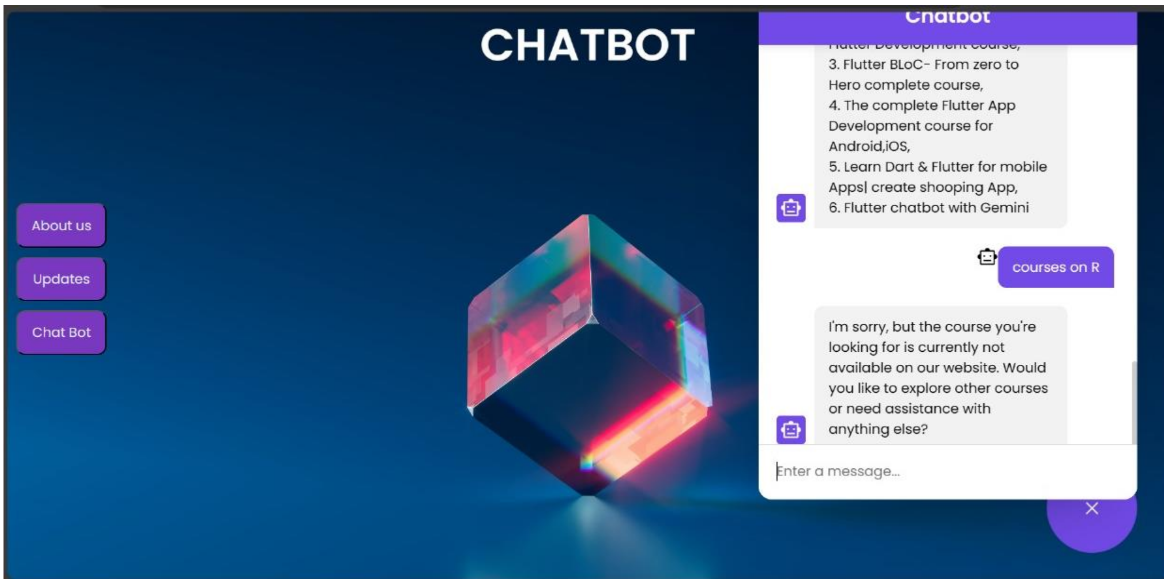
courses available on database

The topics covered in our Database Design and Development course include
1.SQL,
2.MySQL,
3.Database Management System (DBMS) concepts,
4.SQL Server,
5.PostgreSQL,
6.Apache Kafka,
7.MongoDB

Description:

This image highlights how users can explore various courses using the chatbot interface. It may include visual elements that guide users through the course selection process.

CHATBOT COURSE FOR UNAVAILABLE COURSE:



Flutter Development course,
3. Flutter BLoC- From zero to Hero complete course,
4. The complete Flutter App Development course for Android,iOS,
5. Learn Dart & Flutter for mobile Apps| create shooping App,
6. Flutter chatbot with Gemini

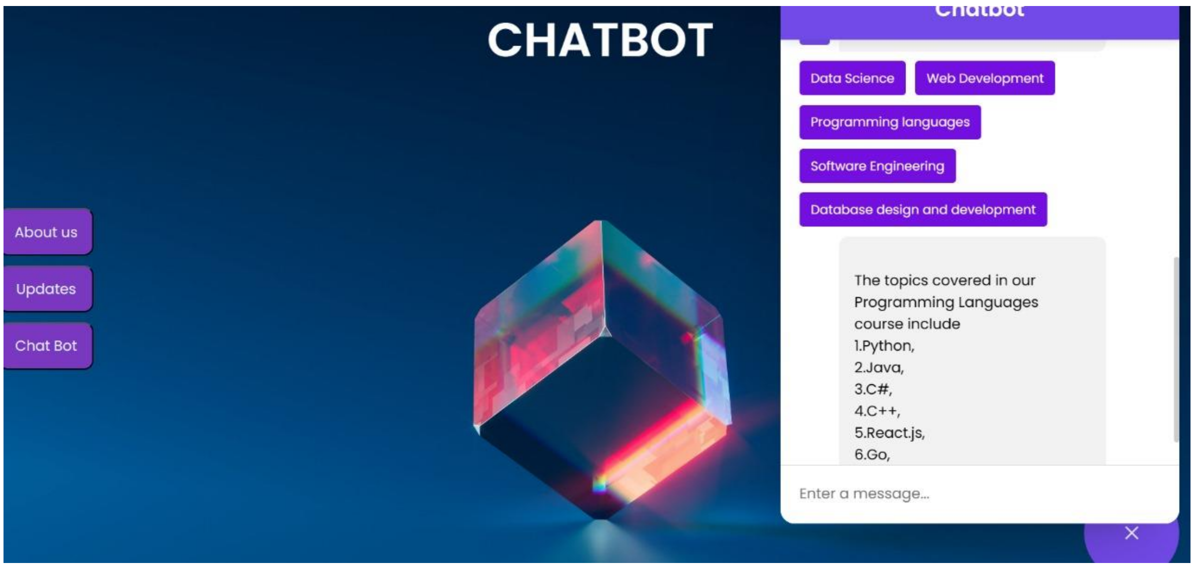
courses on R

I'm sorry, but the course you're looking for is currently not available on our website. Would you like to explore other courses or need assistance with anything else?

Description:

This image shows that the chatbot effectively handles situations where a requested course is not available by providing a courteous and helpful response.

EXPLORE COURSES WITH INTERACTIVE BUTTONS:



The topics covered in our Programming Languages course include
1.Python,
2.Java,
3.C#,
4.C++,
5.React.js,
6.Go,

Description:

This image showcases the interactive buttons within the chatbot that allow users to navigate and explore different courses. It emphasizes the interactive and user-friendly design of the chatbot.

THANK YOU