Chatbot Application

Use Case | Pablo R. Alves

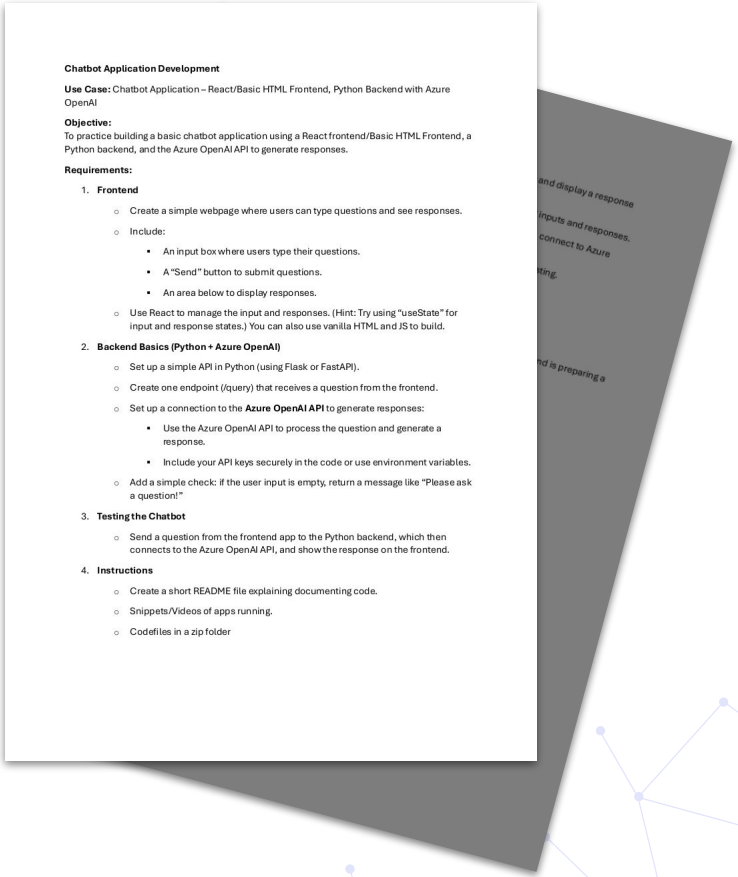
Objective

Build a basic chatbot application

Specifications

User can type questions and see responses.
Elements: input box, send button, text area
Checks: empty user input

1. **Frontend**: React / HTML+JS
2. **Backend**: Python API (Flask, FastAPI), /query Endpoint
3. **API Key**: Azure OpenAI
   Include it safely / env. Variables



Chatbot Application Development

Use Case: Chatbot Application – React/Basic HTML Frontend, Python Backend with Azure OpenAI

Objective:
To practice building a basic chatbot application using a React frontend/Basic HTML Frontend, a Python backend, and the Azure OpenAI API to generate responses.

Requirements:

1. **Frontend**
   - Create a simple webpage where users can type questions and see responses.
   - Include:
     - An input box where users type their questions.
     - A "Send" button to submit questions.
     - An area below to display responses.
   - Use React to manage the input and responses. (Hint: Try using "useState" for input and response states.) You can also use vanilla HTML and JS to build.
2. **Backend Basics (Python + Azure OpenAI)**
   - Set up a simple API in Python (using Flask or FastAPI).
   - Create one endpoint (/query) that receives a question from the frontend.
   - Set up a connection to the **Azure OpenAI API** to generate responses:
     - Use the Azure OpenAI API to process the question and generate a response.
     - Include your API keys securely in the code or use environment variables.
   - Add a simple check: if the user input is empty, return a message like "Please ask a question!"
3. **Testing the Chatbot**
   - Send a question from the frontend app to the Python backend, which then connects to the Azure OpenAI API, and show the response on the frontend.
4. **Instructions**
   - Create a short README file explaining documenting code.
   - Snippets/Videos of apps running.
   - Codefiles in a zip folder

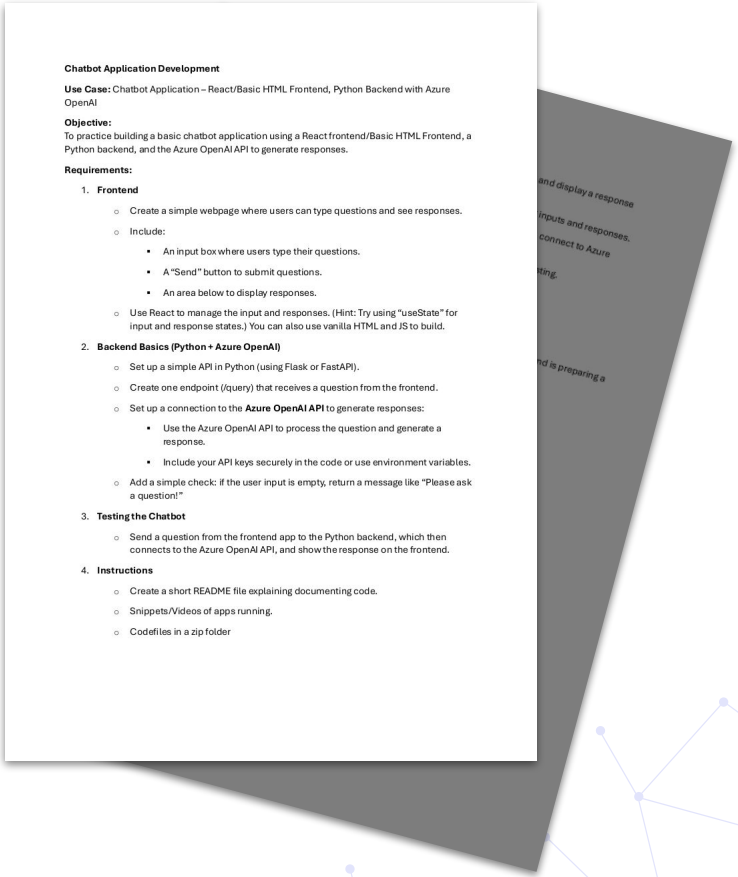
Specifications [II]

Deliverables

1. **Code** (ZIP)
2. **README**.md
3. **Snippets / videos**

Bonus features:

- **Button** (Erase conversation)
- **Loading message** (“Thinking…”)
- **Agentic** AI



Chatbot Application Development

Use Case: Chatbot Application – React/Basic HTML Frontend, Python Backend with Azure OpenAI

Objective:
To practice building a basic chatbot application using a React frontend/Basic HTML Frontend, a Python backend, and the Azure OpenAI API to generate responses.

Requirements:

1. **Frontend**
   - Create a simple webpage where users can type questions and see responses.
   - Include:
     - An input box where users type their questions.
     - A “Send” button to submit questions.
     - An area below to display responses.
   - Use React to manage the input and responses. (Hint: Try using “useState” for input and response states.) You can also use vanilla HTML and JS to build.
2. **Backend Basics (Python + Azure OpenAI)**
   - Set up a simple API in Python (using Flask or FastAPI).
   - Create one endpoint (/query) that receives a question from the frontend.
   - Set up a connection to the **Azure OpenAI API** to generate responses:
     - Use the Azure OpenAI API to process the question and generate a response.
     - Include your API keys securely in the code or use environment variables.
   - Add a simple check: if the user input is empty, return a message like “Please ask a question!”
3. **Testing the Chatbot**
   - Send a question from the frontend app to the Python backend, which then connects to the Azure OpenAI API, and show the response on the frontend.
4. **Instructions**
   - Create a short README file explaining documenting code.
   - Snippets/Videos of apps running.
   - Codefiles in a zip folder

Focus

Design, data/control flow

GPT/Copilot?

Allowed

Thought Process

Show

Process

Building the chatbot

Summary

1 Analyze 

2 Code

3 Test

4 Extend

5 Test

6 Record

Stack

Frontend

React 18 – UI
Components, useState.

Vite 5 – Dev server
Hot module replacement

CSS, **JSX**
Custom styles

Backend

FastAPI – HTTP API,
request/res models (Pydantic)

Uvicorn – ASGI server
(running FastAPI app)

OpenAI Python client
Chat completions.

python-dotenv – Load
AZURE_OPENAI_* from .env.

External Service

Azure OpenAI – GPT-3
API key provided

Motivation

Frontend

React 18 – UI
Simple, widespread

Vite 5 – Dev server
Quick, little configuration

CSS, **JSX**
Simple

Backend

FastAPI – HTTP API,
Quick

Uvicorn – ASGI server
Default (FastAPI), simple CLI

OpenAI Python client
Fixed

python-dotenv – Load
Safety

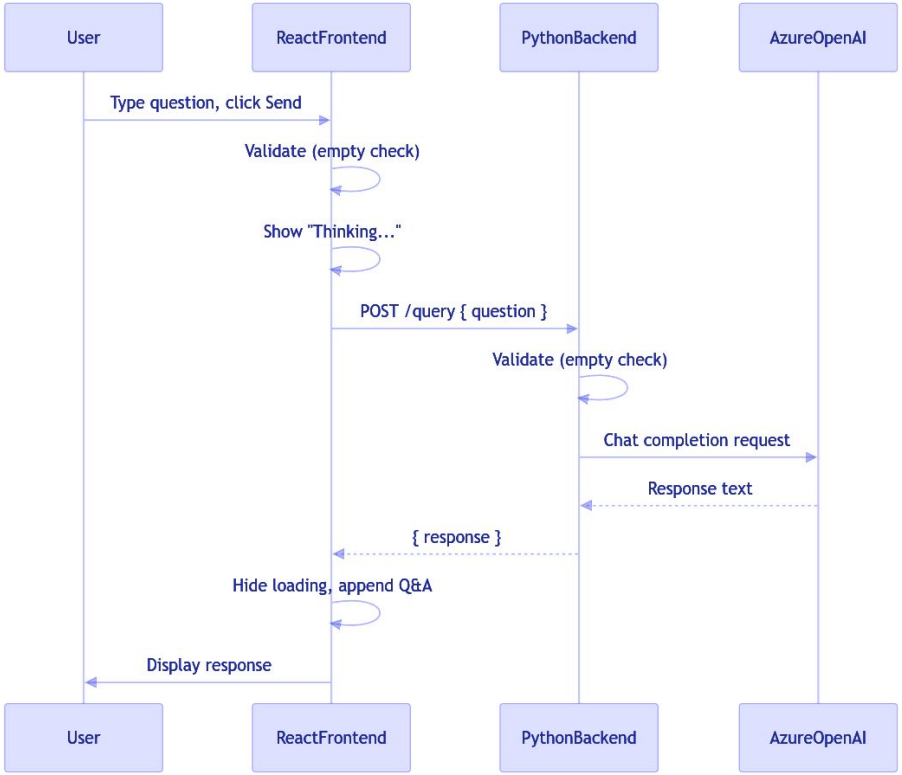
External Service

Azure OpenAI – GPT-3
Fixed

Structure

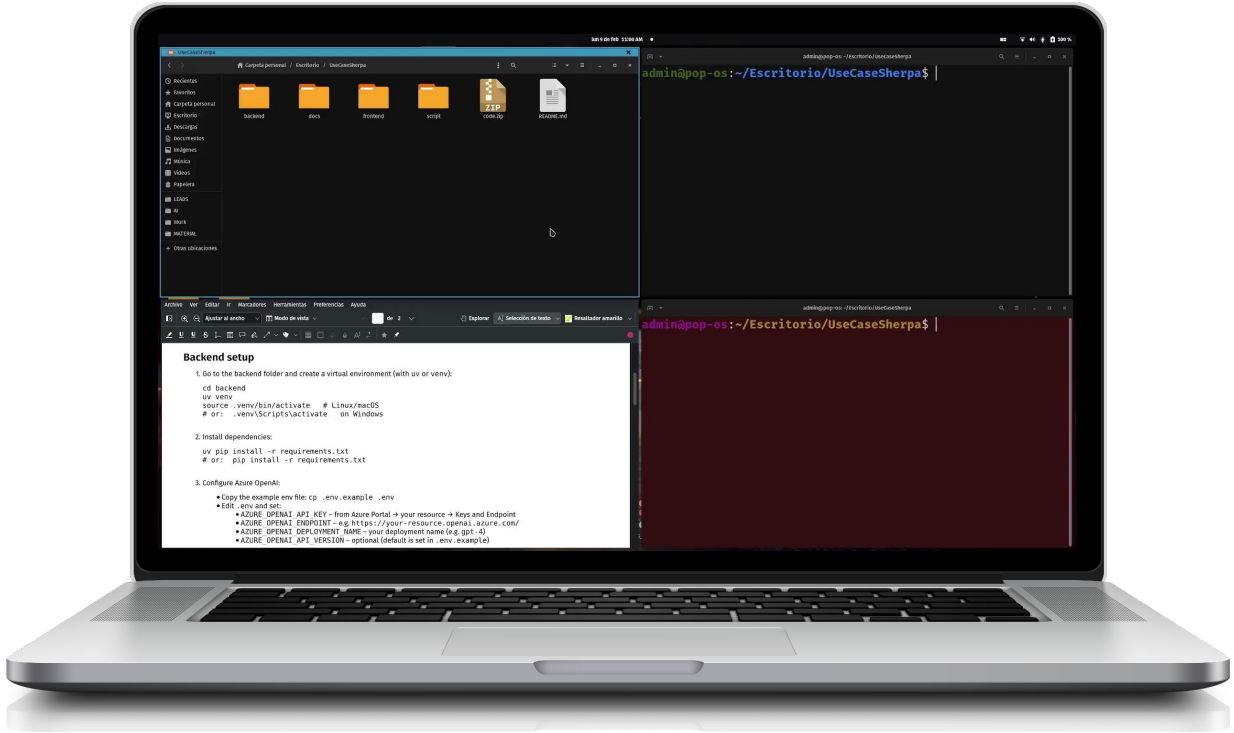
```
├── backend/
│   ├── .env.example     # Template for environment variables
│   ├── requirements.txt
│   └── main.py          # FastAPI app and /query endpoint
├── frontend/
│   ├── index.html
│   ├── package.json
│   ├── vite.config.js
│   ├── public/
│   │   └── favicon.png
│   └── src/
│       ├── main.jsx
│       ├── App.jsx      # Chat UI and state (useState, fetch)
│       ├── App.css
│       └── StarsBackground.jsx
├── docs/
│   └── screenshots/
├── script/
│   └── create_zip.sh    # Utility to generate ZIP without API key
└── README.md
```

Data Flow

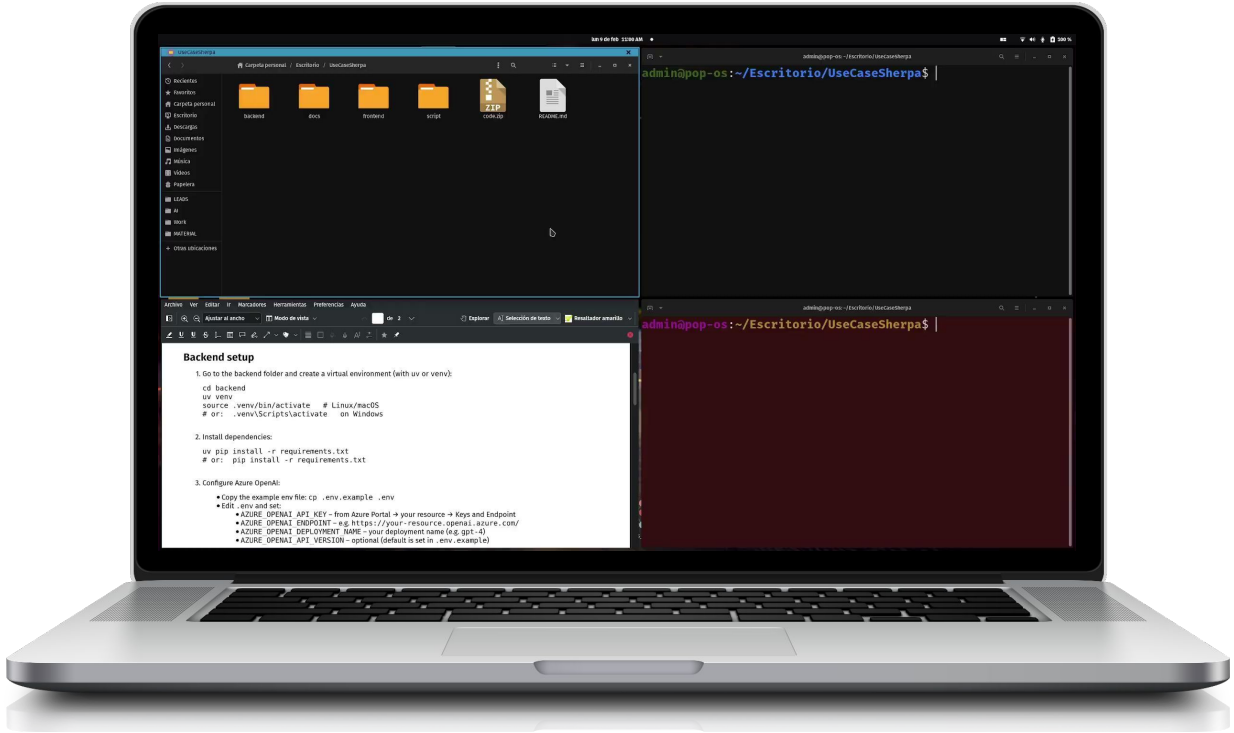

User
ReactFrontend
PythonBackend
AzureOpenAI
Type question, click Send
Validate (empty check)
Show "Thinking..."
POST /query { question }
Validate (empty check)
Chat completion request
Response text
{ response }
Hide loading, append Q&A
Display response
User
ReactFrontend
PythonBackend
AzureOpenAI

Result

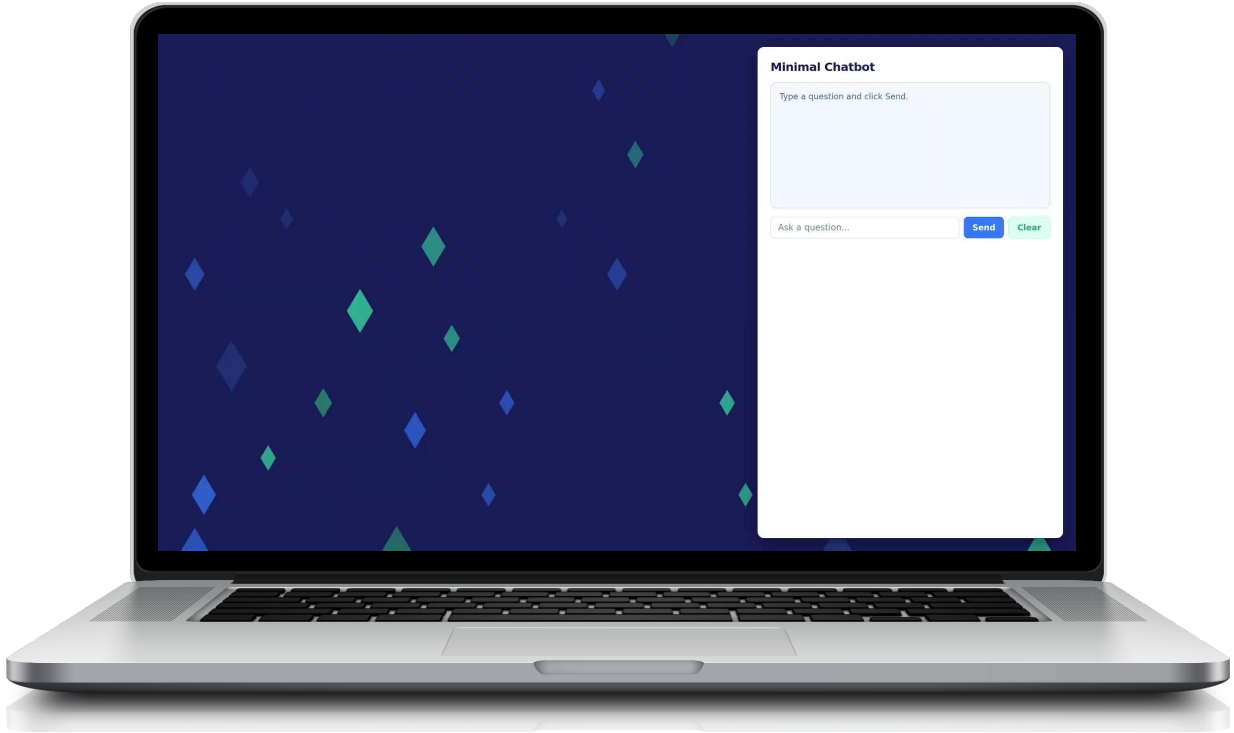
The chatbot in action


backend
docs
frontend
script
code.zip
README.md
admin@pop-os:~/Escritorio/UseCaseSherpa$
admin@pop-os:~/Escritorio/UseCaseSherpa$
Backend setup
1. Go to the backend folder and create a virtual environment (with uv or venv):
cd backend
uv venv
source .venv/bin/activate # Linux/macOS
or: .venv\Scripts\activate on Windows
2. Install dependencies:
uv pip install -r requirements.txt
or: pip install -r requirements.txt
3. Configure Azure OpenAI:
• Copy the example env file: cp .env.example .env
• Edit .env and set:
• AZURE_OPENAI_API_KEY – from Azure Portal → your resource → Keys and Endpoint
• AZURE_OPENAI_ENDPOINT – e.g. https://your-resource.openai.azure.com/
• AZURE_OPENAI_DEPLOYMENT_NAME – your deployment name (e.g. gpt-4)
• AZURE_OPENAI_API_VERSION – optional (default is set in .env.example)

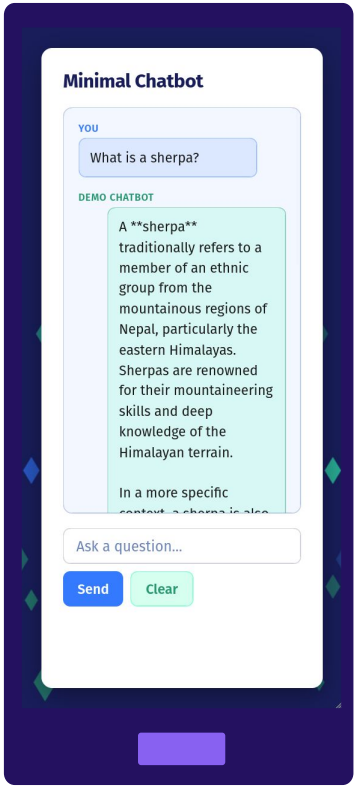
Setup


backend
docs
frontend
script
code.zip
README.md
admin@pop-os:~/Escritorio/UseCaseSherpa$
admin@pop-os:~/Escritorio/UseCaseSherpa$
Backend setup
1. Go to the backend folder and create a virtual environment (with uv or venv):
cd backend
uv venv
source .venv/bin/activate # Linux/macOS
or: .venv\Scripts\activate on Windows
2. Install dependencies:
uv pip install -r requirements.txt
or: pip install -r requirements.txt
3. Configure Azure OpenAI:
• Copy the example env file: cp .env.example .env
• Edit .env and set:
• AZURE_OPENAI_API_KEY – from Azure Portal → your resource → Keys and Endpoint
• AZURE_OPENAI_ENDPOINT – e.g. https://your-resource.openai.azure.com/
• AZURE_OPENAI_DEPLOYMENT_NAME – your deployment name (e.g. gpt-4)
• AZURE_OPENAI_API_VERSION – optional (default is set in .env.example)

Setup

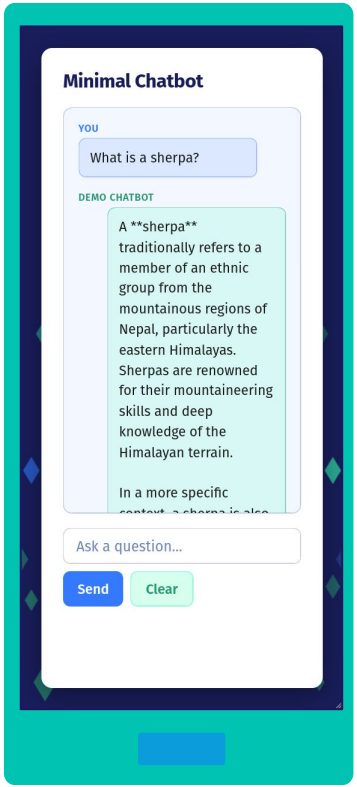

Minimal Chatbot
Type a question and click Send.
Ask a question...
Send
Clear

Demo



Evaluation

- ✅ **Functionality**
  Receive questions and display responses
- ✅ **React Basics**
  Manage inputs and responses
- ✅ **Python basics**
  Setup API, connect and respond
- ✅ **Instructions**
  Easy-to-follow (setting and testing)



Bonus

- ✅ **Extra Button**
  User can clear conversation
- ✅ **Loading Message**
  User sees “Thinking…” from assistant
- ✅ **API Key safety**
  Python-dotenv helps protect it
- ✅ **ZIP Script**
  Facilitates replicability

Wrap Up

What have we learned?

Chatbots

Simple to create

AI Tools

Accelerate development

Stack

Safeguards future development

Thank you!

Do you have any questions?