Groq key

VS code

Extension – install Python, Python Debugger, Jupyter Notebook, Pylance

Create a new Virtual Env

python -m venv EnvName

Activate the env

EnvName\Srcipts\activate

Check python version

python –version

End to End Modular coding using LangGraph

1. State Schema
2. Nodes are functions
3. Edges connect nodes
4. State Graph

Modular coding for deployment

Modular coding for reusability

Step by Step coding

1. Create virtual env
2. Activate the env
3. Create requirements.txt
4. Add the packages in requirements.txt
5. File ->Auto Save
6. Terminal: pip install -r requirements.txt
7. Open Github, to parallely commit code to github
8. Github: Repositories -> New -> Give a project name ([**Agentic-Project**](https://github.com/Paramita1985/Agentic-Project)) -> Create repository (no need to fill any details)
9. To commit code, you need to have git downloaded - <https://git-scm.com/downloads/win>
10. Once git is downloaded, go to github, copy the command git init
11. Git init – initializes the git repository, run this in the VS code terminal

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/Paramita1985/Agentic-Project.git

git push -u origin main

1. Run git init in the vs terminal – an empty git repository is created in the a file path, the names will be in green and U means untracked
2. In VS code create a file .gitignore directly, it will have a diamond shape. We need gitignore for files that are not required in the git repository like the virtual environment
3. In the gitignore write envname/ (eg. AgenticProj/). The green tracking is off now.
4. Create a README.md file, Now execute git add README.md. README file is to write basic information about the project
5. git add . will add all files in U mode for commit
6. git commit -m "first commit" run in terminal. Will commit the files in Git repository (-m is message here it is “first commit”)
7. We have committed the code but not pushed the code
8. Create a branch - git branch -M main naming the branch as main
9. Now run: git remote add origin https://github.com/Paramita1985/Agentic-Project.git
10. Now run: git push origin main to push code from origin (local) to main (branch)
11. Pull request – if another developer has updated code, pull the latest code first from main to origin
12. Now create the folder structure for coding efficiency

Modular structure

Src folder and \_\_init\_\_.py (so that a package is created)

Execution will start from app.py

A screenshot of a computer program

AI-generated content may be incorrect.

Start with LLMs – create groqllm.py

Then ui -> create folder streamlitui -> create loadui.py to load ui and display\_result.py

Under Ui folder -> create a uiconfigfile.ini (for all data to be shown in Ui). Ini is a file type for plain text similar to yaml

Code:

[DEFAULT]

PAGE\_TITLE = "LangGraph: Build Stateful Agentic AI graph"

LLM\_OPTIONS = Groq

USECASE\_OPTIONS = Basic Chatbot, Chatbot with Tools, SDLC Workflow

GROQ\_MODEL\_OPTIONS = mixtral-8x7b-32768, llama3-8b-8192, llama3-70b-8192, gemma-7b-i

Create uiconfigfile.py to read the config file

A screen shot of a computer program

AI-generated content may be incorrect.

ConfigParser will read the config file

Start coding in loadui.py

If you see the code, the first that you load in the UI is the config

Commit the code again and run to see the UI in streamlit

git add .

git commit -m "loadui and configuration and folder structure commit"

git push origin main

Now back to loadui.py

Create the front end design with streamlit

            self.user\_controls["selected\_llm"] = st.selectbox("Select LLM", llm\_options)

here user\_controls is a dict where whatever is selected in the selectbox is stored in this dict

Run the streamlit from app.py

Write code in main.py and app.py and display\_result.py

Main.py – has the right hand side UI

A screenshot of a computer screen

AI-generated content may be incorrect.

App.py – starting point, from here the main.py loads

from src.langraphagenticai.main import load\_langgraph\_agenticai\_app

if \_\_name\_\_=="\_\_main\_\_":

    load\_langgraph\_agenticai\_app()

app.py -> main.py ->loadui.py -> uiconfigfile.py -> uiconfigfile.ini

streamlit run app.py

Till here we have built the modular structure, UI and run the streamlit with UI

Second phase:

Start with LLMS -> groqllm.py

Now continue writing in main.py

In the groqllm.py capture the values entered from main.py

Main.py captures all user inputs and passes to rest of the .py files

Now we create the graph

From main.py -> graph\_builder.py -> state.py and basic\_chatbot\_node.py (here one method will be written to compile the graph)

Push to Git

Git add .

Git commit -m “basic chatbot”

Git push origin main

In the graph\_builder.py (setup\_graph) there will be compilation

Now Stream and Display result

Go to display\_result.py write code to display result

Now run streamlit run app.py

Ctrl+C to come out of terminal