L1Blom / **rag**

Code | Issues | Pull requests | Discussions | Actions | Projects | Security | Insights | Settings

RAG Service is a template service for a retrieval-augmented generator based on the examples of LangChain. See: [Build a Retrieval Augmented Generation (RAG) app: https://python.langchain.com/v0.2/docs/tutorials/rag/

MIT license

Code of conduct

Security policy

0 stars **0** forks **2** watching **1** Branch **1** Tags Activity

Public repository

main **1** Branch **1** Tags Go to file t Go to file Add file Code

Leen Blom Extra measures for persistence refelcting changes in model, temp and … af8f3fd · 5 hours ago

.github/ISSUE_TEMPLATE	Added feature request file	5 days ago
constants	Extra measures for persistence refelcting chan…	5 hours ago
data/_unittest	Completed unit tests for image recognition	yesterday
docs	Moved default files to .github repositiry	5 days ago
services	Worked on documentation and README	2 weeks ago
.gitignore	Moved location of vectorestore	12 hours ago
.pylintrc	Moved the globals to Flask aware context proc…	2 weeks ago
LICENSE	Tidying up to create a release	last week
README.md	Moved location of vectorestore	12 hours ago
config.py_example	Initial commit	last month
nginx_example.conf	updated import according to langchain-ai/langc…	2 weeks ago
ragservice.py	Extra measures for persistence refelcting chan…	5 hours ago
ragservice_unittest.py	Implemented persistency in vectorstore	12 hours ago

README | Code of conduct | MIT license | Security

RAG Service

RAG Service is a template service for a retrieval-augmented generator based on the examples of LangChain. See: Build a Retrieval Augmented Generation (RAG) App

Technologies

Installation

1. Clone L1Blom/rag to your project directory
2. Move config.py_example to config.py and add your API Key
3. Choose an ID for your instance, like MyDocs

4. Copy constants/constants.py to constants_MyDocs.py
5. Change the contents of this file to reflect your situation
6. Create a MyDocs/ directory in data/ and in MyDocs/ a directory vectorstore/
7. Add the files for your RAG context, like text-files, to your MyDocs/ directory
8. Create a Python virtual environment for your project (optional)
9. Modify the example services/rag.service_template file to point to the right place of your project directory
10. Copy the services file to /etc/systed/system/rag_MyDocs.service
11. Enable and start the service

```
sudo systemctl enable rag_MyDocs
sudo systemctl start rag_MyDocs
sudo systemctl status rag_MyDocs
```

Commands

All calls support POST and GET. For <ID> use your chosen ID like MyDocs

1. /prompt/<ID>/

   Parameter: prompt (string)

   Your prompt to be send to openAI

2. /prompt/<ID>/model

   Parameter: model (string)

   Your model to be used, like "gpt-4o"

   Checking on valid models with OpenAI client.models.list(). Can result in http 500 error (non-fatal)

3. /prompt/<ID>/temp

   Parameter: temp (string, will be cast to float)

   Temperature setting, between 0.0 and 2.0

   Settings above 1.0 can give significant halicunations and degrades performance too.

   Timeout can result in http 408 error (non-fatal)

4. /prompt/<ID>/reload

   Parameters: none

   After adding files to your data directory, use this to reload the vector store

5. /prompt/<ID>/clear

   Paramters: none

   Clears the cache, the in-memory history

6. /prompt/<ID>/cache

   Paramaters: none

   Prints the cache contents to the response object

7. /prompt/<ID>/image

   Parameters: prompt (string), image (URL to image)

   Uploads the image to openAI and use prompt to get the desired contents like: 'What is the mood of the persons?'

   Note: only works if model is set to 'gpt-4o'. Other models result in http 500 error (non-fatal)

Usage

```
# change the model
curl -X POST --data-urlencode "model=gpt-4o" http://<your server>:<your port>/prompt/<ID>/model
```

```
Model set to: gpt-4o
# prompt to your data
curl -X POST --data-urlencode "prompt=your question?" http://<your server>:<your port>/prompt/<ID>

Your answer based on the context files provided in data/<ID>
```

Constants file

```
# simple string like "mydata"
ID="<your identifier>"
# any unused port, will run the Flask server
PORT=9100
# Set to True if you want to enable Flask debug
DEBUG=False
# Any other level will make it less verbose
LOGGING_LEVEL="INFO"
# Where the files reside for unit testing
HTML="data/_unittest/html"
# Directory that will be scanned for files to be added to the context
DATA_DIR="/home/data"
# All the file extentions you want to be part of the context, see LangChain documentation
DATA_GLOB="*.txt"
# The lower, the more precise. Needs to be between 0.0 and 2.0
TEMPERATURE=0.0
# Influences the way answers are produced
contextualize_q_system_prompt = (
        "Given a chat history and the latest user question "
        "which might reference context in the chat history, "
        "formulate a standalone question which can be understood "
    )
# Influences the way answers are produced
system_prompt = ( # choose your language, English works best
        "You are a chatbot and gives answer in not more than 3 sentences"
    )
chunk_size=1000    # depending on your data, seel LangChain documentation
chunk_overlap=100  # idem
```

Unit tests

To run the unit tests, run the program in the project directory using the ID '_unittest'. It will start a local RAG service accessible at port 8888 (see constants__unittest.py for all defaults).

```
<your virtual environment>/bin/python ragservice.py _unittest
INFO:httpx:HTTP Request: GET https://api.openai.com/v1/models "HTTP/1.1 200 OK"
INFO:root:Context loaded from 1 documents
INFO:chromadb.telemetry.product.posthog:Anonymized telemetry enabled. See                     https://docs.trychroma.
INFO:httpx:HTTP Request: POST https://api.openai.com/v1/embeddings "HTTP/1.1 200 OK"
INFO:root:Chain initialized: gpt-4o
 * Serving Flask app 'ragservice'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8888
 * Running on http://192.168.2.200:8888
INFO:werkzeug:Press CTRL+C to quit
```

Now you are able to run the unit tests:

```
<your virtual environament>/bin/python ragservice_unittest.py -v
test_chache_content (__main__.RagServiceMethods.test_chache_content)
Test to print the contents of the cache ... ok
test_clear (__main__.RagServiceMethods.test_clear)
Test to clear the cache ... ok
test_image (__main__.RagServiceMethods.test_image)
Test image ... ok
test_model (__main__.RagServiceMethods.test_model)
Test model setting, correct or incorrect model according to OpenAI ... ok
test_prompt (__main__.RagServiceMethods.test_prompt)
Test prompt ... ok
test_reload (__main__.RagServiceMethods.test_reload)
Test reload of the data ... ok
```

```
test_temperature (__main__.RagServiceMethods.test_temperature)
Test to set temparature too high, low, within boundaries 0.0 and 2.0 ... ok

----------------------------------------------------------------------
Ran 7 tests in 5.923s

OK
```

TODO's and wishes

- More file formats like PDF
- Upload files
- Other API's than OpenAI and other LLM's
- A simple frontend from the GitHub community

Contributing

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change.

Please make sure to update tests as appropriate.

License

MIT

The image used in the unittest is licensed CC BY-NC-ND 4.0 and was found at Trusted Reviews

Releases 1

Initial release Latest
last week

Packages

No packages published
Publish your first package

Languages

Python 94.0% **Shell** 6.0%