Design Documentation

*** Date: 18.07.2023 ***

Textual Explanation

Relation of the runtime components

When users want to ask a question, they open their Slack Application and submit their query in a chat. The Slack-Bot is our first component, which captures the question and forwards it to the QABot. The QABot then attempts to answer the user's question[1].

The QABot queries the required information from our vector database weaviate. Weaviate then returns the chunks of information whose embedding are closest to the embedding of the question. The queried information and the question are sent to an LLM Model that generates an answer based on the provided data.

The Data Processing component is employed to add, modify, and delete data from the vector database. This component accesses various data sources, such as Google Drive, Slack, and the Confluence API, to obtain information. After loading the raw data, the Data Processing component preprocesses it, which may involve splitting it into shorter texts or translating it. The DeepL-API is used for translation tasks. Once the data has been loaded and preprocessed, it is saved in the vector store, so it can later be accessed by our bot[2].

Remarks:

[1] Separating the SlackBot and QABot provides advantages such as that for testing we are not required to have any dependency to Slack. Furthermore, it allows for flexibility in incorporating other chat platforms like WhatsApp in the future. [2] As we are using an embedded database, the transfer of the database data to the database of the QABot is handled via Google Cloud.

Explanation of the code components

The code components are closely linked to the runtime components. There is a code component for the SlackBot(qa_agent.py) that captures user input and forwards it to the QABotAPI-Interface(qa_bot_api_interface.py). The QABotAPI-Interface receives the question and forwards it to the QABot(qa_bot.py). As the QABot and SlackBot run on two different VMs in Google Cloud the request to the QABot is handled via a Flask Server[1]

The QABot first embeds the question using an EmbedderAPI, then uses the internal weaviate client to query the relevant information, and finally utilizes an LLM-API to generate the user's answer. [2]

The DataProcessing unit saves information in the vector database. It first uses a DataLoader to load raw data without preprocessing. Then, for each data source type, there is a separate DataPreprocessor that transforms the raw data into documents to be saved in the vector database. [3] Also in the DataProcessing run integrated is a PDF Reader, that can scrape the text or information from a PDF. Therefore it uses pdfminer.six as a way to get the text information by reading the data. As there is also the possibility that the given pdf is a scan and therefore has no text information, we added also a second way to read data using the optical image analysis from Tesseract. As a last code component for the DataProcessing, there is a script with a text_transformation function. This splits the text into chunks for the database.

Remarks:

[1] The QABotAPI-Interface is used as a c in the QABotAPI enables the separation of the SlackBot from the QABotAPI, allowing them to function as independent programs that can even run on different servers.

[2] There are already several EmbedderAPIs, VectorDatabaseAPIs, and LLMAPIs available with a common interface (e.g., in LangChain), which allows us to test and switch between different providers easily.

[3] This approach facilitates the addition of more data sources later if necessary.

The role of Translation in the project

As the project company has a lot of data in German and also most of the user-interaction will be in German, but most embedder & LLMs work best in English, we translate all non-English text to English and the answer back to the language the user initially asked the question. Therefore we use DeepL which can translate from and into nearly every language and returns the source language with the translated text. This can be used to enable, that questions get asked in Spanish and the Bot can answer in Spanish.

Technologies Stack Explanation

Programming Language

We are using Python as the programming language for all components. Although the programming language for the SlackBot, QABotAPI, and DataProcessing can be chosen independently, it is advantageous to focus on a single language for consistency. Python is suitable because it supports Slack Bots and offers numerous APIs for working with LLMs and embeddings as libraries.

LLM Model API and Embedding API

As we wanted to work with as many OpenSource Tools as possible we chose to use also OpenSource Models. Currently, the embeddings are created by instructor-xl, which showed the best performance in creating embeddings during our tests. For answering the questions, we chose Wizard Mega, a LLaMA derivate, which seems to be the best open source model for our purposes at the moment.

Vectordatabase

Supabase appears to be a promising option for the vector database. As an open-source alternative to Firebase, it includes a PostgreSQL database with embedding and similarity search support. Moreover, it is already implemented in LangChain. This was also our initial move, which worked very well. But as we wanted to increase data security and wanted to prevent GDPR problems we chose to use a self-hosted database. As we already had the LangChain APIs implemented, we wanted to make it an as easy as possible switch. Therefore we selected the Embedded Version of Weaviate, which also has an integration with Langchain. This also shows, that by the modularity we implemented in the code, we were able to switch out dependencies and components as easily as possible.

To ensure fast and good results, we split long texts into shorter chunks. This is done by using the NaturalLanguageToolKit(NLTK) from Langchain. There only needs to be set a ChunkSize and ChunkOverlap as needed and it will split up the text. The resulting DataInformations get added a chunk-index to later identify the order of multiple text chunks.

Cloud Infrastructure

Cloud infrastructure is necessary for hosting our SlackBot, QABot, and executing the preprocessing. Since the customer uses Google Cloud, we decided to also utilize Google Cloud for consistency and integration purposes. Therefore we created three different VMs that run the different Code components. This separated architecture not only allows for modularization and easy integration for other Question Sources but also enabled us to save hosting costs, by having specific time schedules for the different VMs and for example letting the more expensive QABot VM only run at working hours.

Technology Stack

Programming languages and framework

- Slack Bot:
    - Python 3.8
- QA Bot:
    - Python 3.8
- DataProcessing:
    - Python 3.8

Database management systems

- Weaviate

Third-party libraries and components

- Slack Bot:
    - Slack API
    - Slack-sdk
    - slackclient
    - slack-bolt
- QA Bot:
    - WizardLM
    - flask
    - huggingface_hub
    - gunicorn
    - llama-cpp-python

- DataProcessing
  - pdfminer.six
  - pdf2image
  - pytesseract
  - nltk
  - Web-scraper/loader:
    - Atlassian-python-api
    - Selenium
    - beautifulsoup4
    - requests
  - Connection to the different data sources:
    - Slack-sdk
    - google-api-python-client
    - pydrive2
    - google-auth
    - google-auth-oauthlib
    - google-cloud-storage
- Common
  - prettytable
  - LangChain
  - Instructor
  - InstructorEmbedding
  - DeepL API
  - Spacy
  - sentence-transformers
  - google-cloud-storage
  - python-dotenv

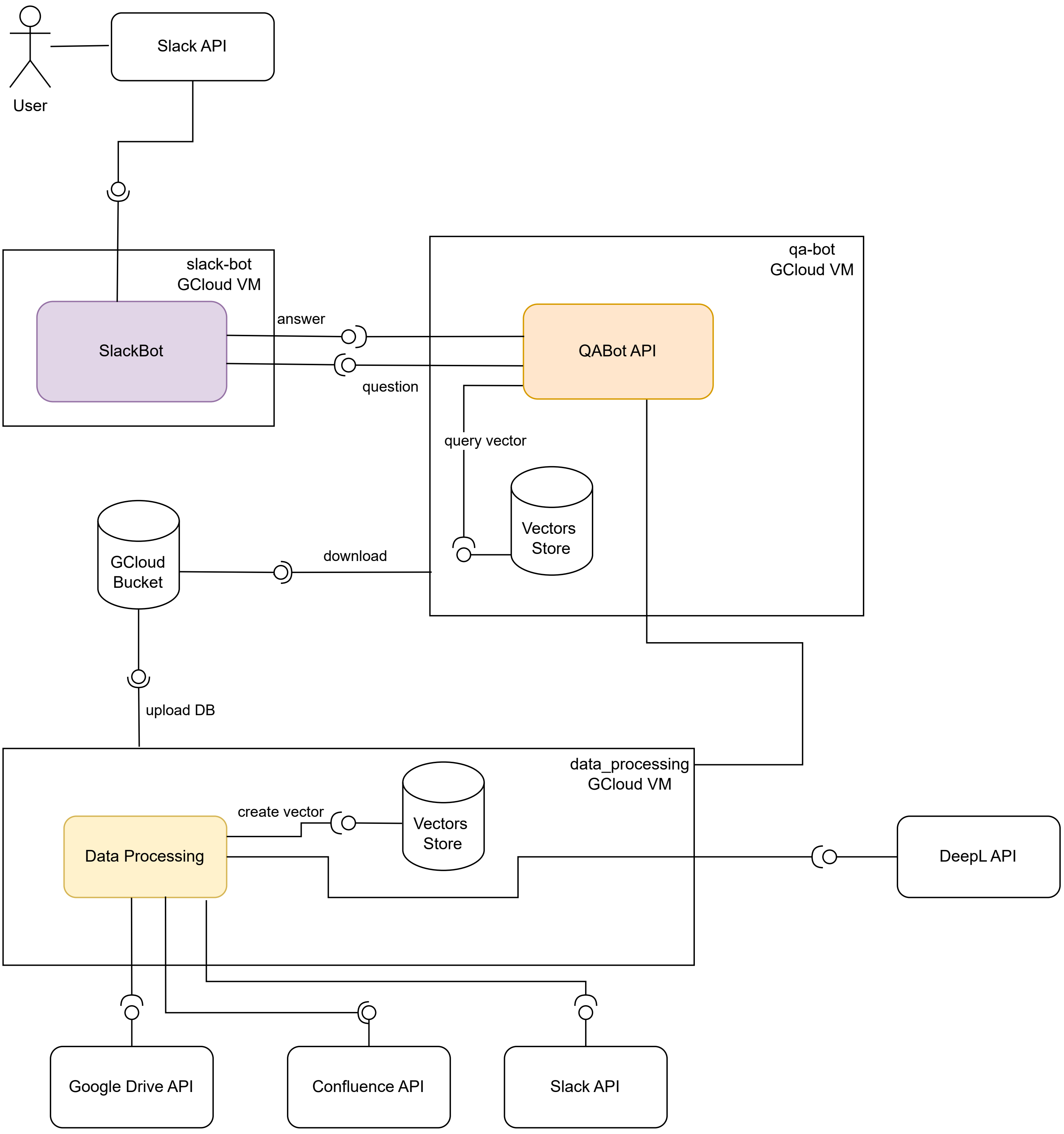
Development tools and environments

- Version Control System:
  - Git / GitHub
- Package Manager:
  - pip (bundled in python)
  - venv (bundled in python)
- IDE
  - PyCharm
- CI/CD:
  - GitHub Action

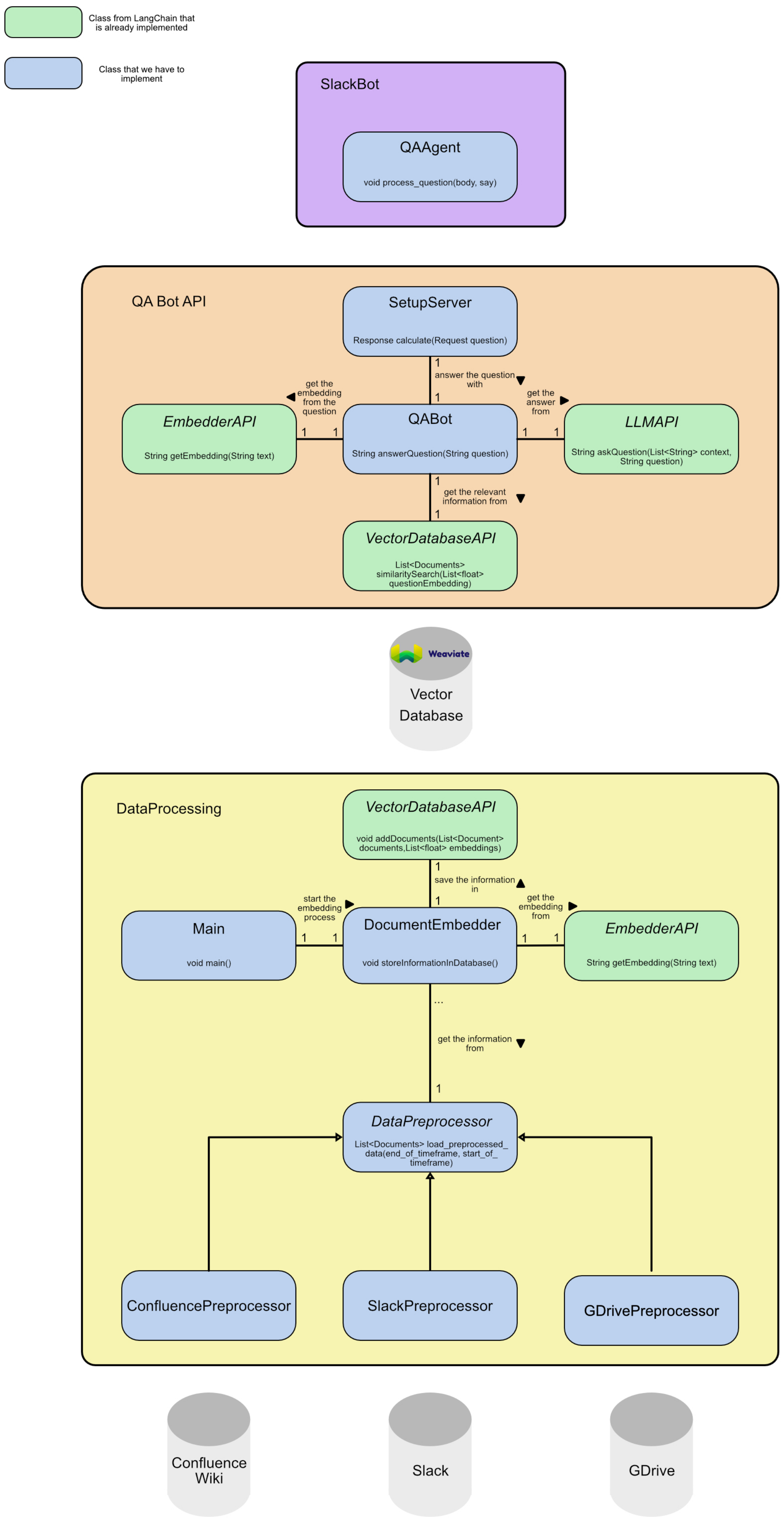
Infrastructure components

- Google Cloud
- Weaviate

Runtime Component Diagram



Code Component Diagram


Class from LangChain that is already implemented
Class that we have to implement
SlackBot
QAAgent
void process_question(body, say)
QA Bot API
SetupServer
Response calculate(Request question)
1
answer the question with
1
get the embedding from the question
EmbedderAPI
String getEmbedding(String text)
1
1
QABot
String answerQuestion(String question)
get the answer from
1
1
LLMAPI
String askQuestion(List<String> context, String question)
1
get the relevant information from
1
VectorDatabaseAPI
List<Documents> similaritySearch(List<float> questionEmbedding)
Weaviate
Vector Database
DataProcessing
VectorDatabaseAPI
void addDocuments(List<Document> documents,List<float> embeddings)
1
save the information in
1
start the embedding process
Main
void main()
1
1
DocumentEmbedder
void storeInformationInDatabase()
get the embedding from
1
1
EmbedderAPI
String getEmbedding(String text)
...
get the information from
1
DataPreprocessor
List<Documents> load_preprocessed_data(end_of_timeframe, start_of_timeframe)
ConfluencePreprocessor
SlackPreprocessor
GDrivePreprocessor
Confluence Wiki
Slack
GDrive