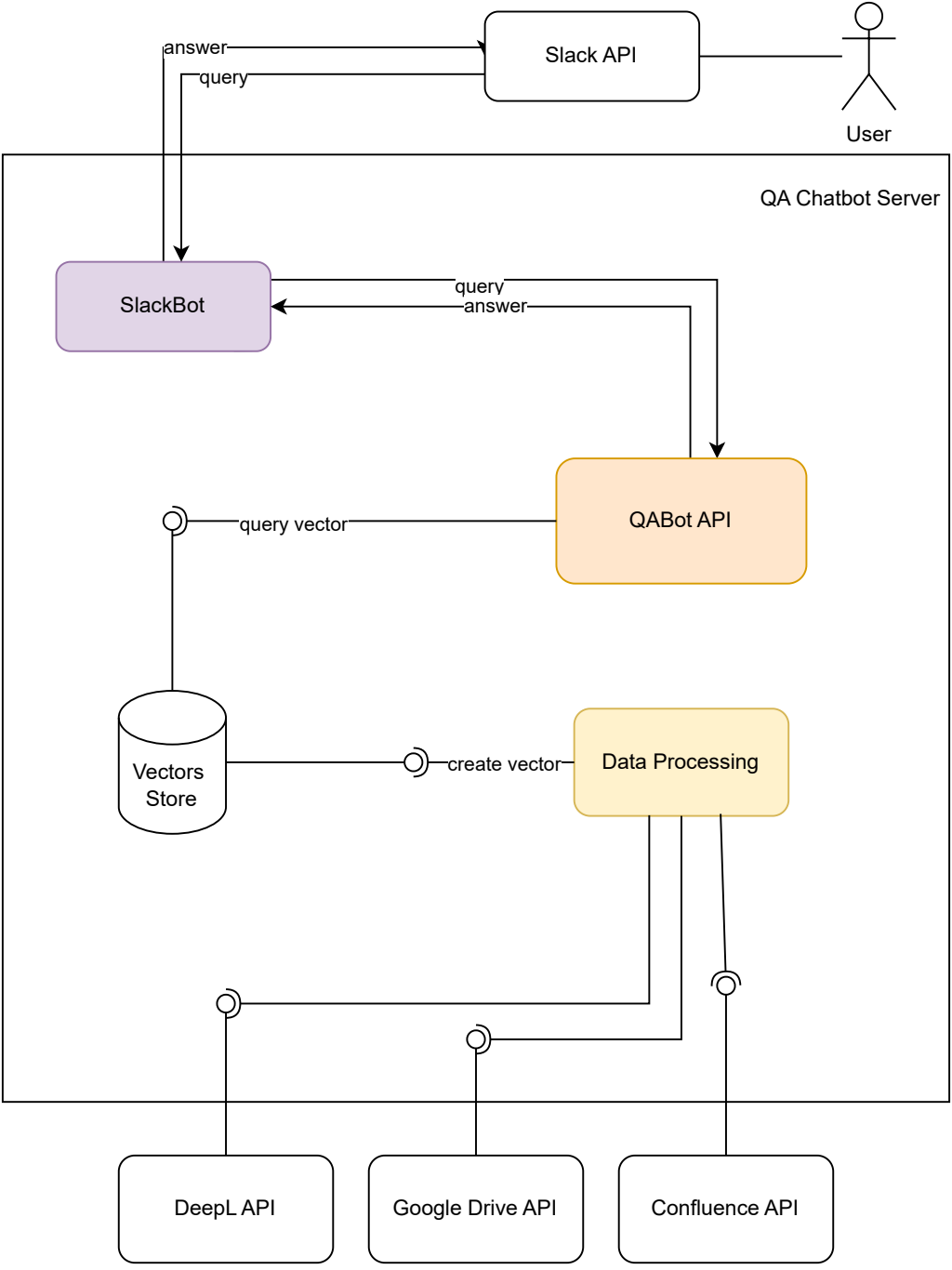
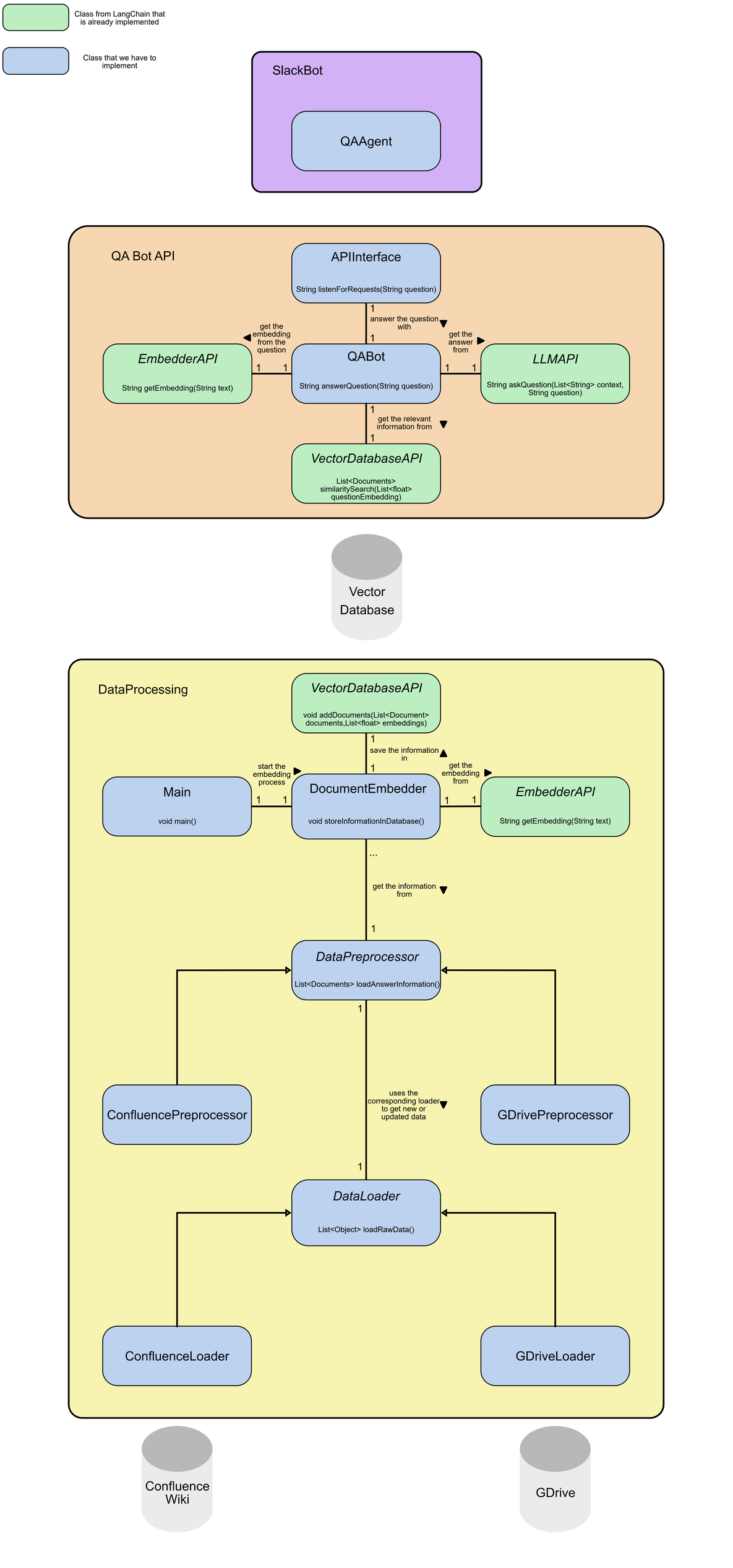

answer
query
Slack API
User
QA Chatbot Server
SlackBot
query
answer
QABot API
query vector
Vectors Store
create vector
Data Processing
DeepL API
Google Drive API
Confluence API


Class from LangChain that is already implemented
Class that we have to implement
SlackBot
QAAgent
QA Bot API
APIInterface
String listenForRequests(String question)
1
answer the question with
1
get the embedding from the question
get the answer from
EmbedderAPI
String getEmbedding(String text)
QABot
String answerQuestion(String question)
LLMAPI
String askQuestion(List<String> context, String question)
1
get the relevant information from
1
VectorDatabaseAPI
List<Documents> similaritySearch(List<float> questionEmbedding)
Vector Database
DataProcessing
VectorDatabaseAPI
void addDocuments(List<Document> documents,List<float> embeddings)
1
save the information in
1
start the embedding process
get the embedding from
Main
void main()
DocumentEmbedder
void storeInformationInDatabase()
EmbedderAPI
String getEmbedding(String text)
...
get the information from
1
DataPreprocessor
List<Documents> loadAnswerInformation()
1
ConfluencePreprocessor
uses the corresponding loader to get new or updated data
GDrivePreprocessor
1
DataLoader
List<Object> loadRawData()
ConfluenceLoader
GDriveLoader
Confluence Wiki
GDrive

Design Documentation

Date: 30.05.2023

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 API is our first component, which captures the question and forwards it to the SlackBot. The SlackBot then attempts to answer the user's question by utilizing our QABotAPI [1].

The QABotAPI then queries the required information from a vector database, for this the information whose embedding are closest to the embedding of the question are loaded. The queried information and the question are sent to a 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 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 DeepLAPI 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 bot.

Remarks:

[1] Separating the SlackBot and QABotAPI provides advantages such as that for testing we are not required to have any dependency to Slack dependencies. Furthermore, it allows for flexibility in incorporating other chat platforms like WhatsApp in the future.

Explanation of the code components

The code components are closely linked to the runtime components. There is a code component for the SlackBot that captures user input and forwards it to the QABotAPI. The QABotAPI receives the question through an APIInterface and forwards it to the main code block, the QABot. [1]

The QABot follows a similar process to the QABotAPI component. First, it embeds the question using an EmbedderAPI, then employs a VectorDatabaseAPI to find relevant information, and finally utilizes an LLMAPI 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]

Remarks:

[1] The APIInterface 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.

Technologies Stack Explanation

Programming Language

We recommend using Python as the programming language. 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

For an initial prototype, the OpenAI API is likely the best choice since it is licensed under the MIT license, easy to use, and considered one of the best options in the market. We suggest exploring other models, such as LLaMA, in the future when time permits. Since everything is implemented against a common interface, swapping out models will be a straightforward process, and some are already implemented in LangChain. 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.

Cloud Infrastructure

Cloud infrastructure is necessary for hosting our SlackBot, QABot, and executing the preprocessing. Since the customer uses Google Cloud, it is recommended that we also utilize Google Cloud for consistency and integration purposes.