Design

William Woodward - will@woodwardweb.com

9 May 2024

Contents

1 Initial Design

1.1 Full-Stack App Design

1.1.1 Frontend Framework

I chose to use React, as I am already familiar with this framework, and it is relatively industry-standard. I want to develop my skills further, and hope that this project will be a good way to do so. Ideally, since React has a large community, people will be able to contribute further / notice bugs more commonly.

1.1.2 Backend Framework

Initially, I chose to use NodeJS for the backend. However, as this project became larger, it became a lot less easy to manage. I chose to switch to using ASP.NET in C#, as this had strong types, better performance, better integration with IMAP and POP, and was something completely new to me.

1.2 AI Integration Design

In line with the initial functional requirements, I want to employ models to serve the following functions:

1. Email autocomplete based on email writing style and conversation history
2. Previous conversation summary and suggested points to mention
3. AI spam filtering to suggest emails that might have been moved to spam

In line with these requirements, since there is going to be a heavy use of LLMs, I will be using foundation models discussed in the next section to achieve the first two functions. I will create my own spam detection model to meet the third requirement.

1.2.1 Foundation Models

Since many LLMs are currently being developed, I want to ensure that this project is fairly agnostic. I will be using Mixtral, as this is a free LLM, and will choose to interact via API, as this seems to be the standard for most models. This leaves room to upgrade to better models in the future, such as GPT-4. I am aware of GPTs, however I want to develop the integration with these models myself, partly for experience and partly to ensure that the project is LLM agnostic.

1.2.2 Knowledge Base

To fine tune the foundation model, I will need to create a knowledge base. This will have to store conversation history, and a representation of email writing style. For now, this will probably involve storing each sent email in a database, however in future it would make sense to condense this.

Two potential options for knowledge bases are graph databases and vector databases. I will use a vector database as this seems to be the industry standard for encoding large amounts of information, but I will leave graph databases open for further investigation.

Conversation history needs to be quickly swapped, depending on the email thread. The vector database should be in the backend, as there needs to be a lot of data processing to get it into the correct format. To do this with minimal delay, when opening an email thread, an API request should be sent, letting the vector database load pre-processed documents in and out of the vector database.

1.2.3 Spam Detection AI

1.2.4 AI Inference

An API request will be sent to a Python API server, containing the spam dection model binaries, whilst interaction with the foundation models and knowledge bases will also be done on this server, using LangChain. At least in the initial design phase, it makes sense organisationally to include all AI related endpoints on the same server. I anticipate as the system design progresses and response times become more demanding, this underlying design might change. The client should be able to interact with both the AI server and Backend Web server, to anticipate the use of in-memory database caching such as redis in future.

1.3 Testing Environment

1.4 CI/CD