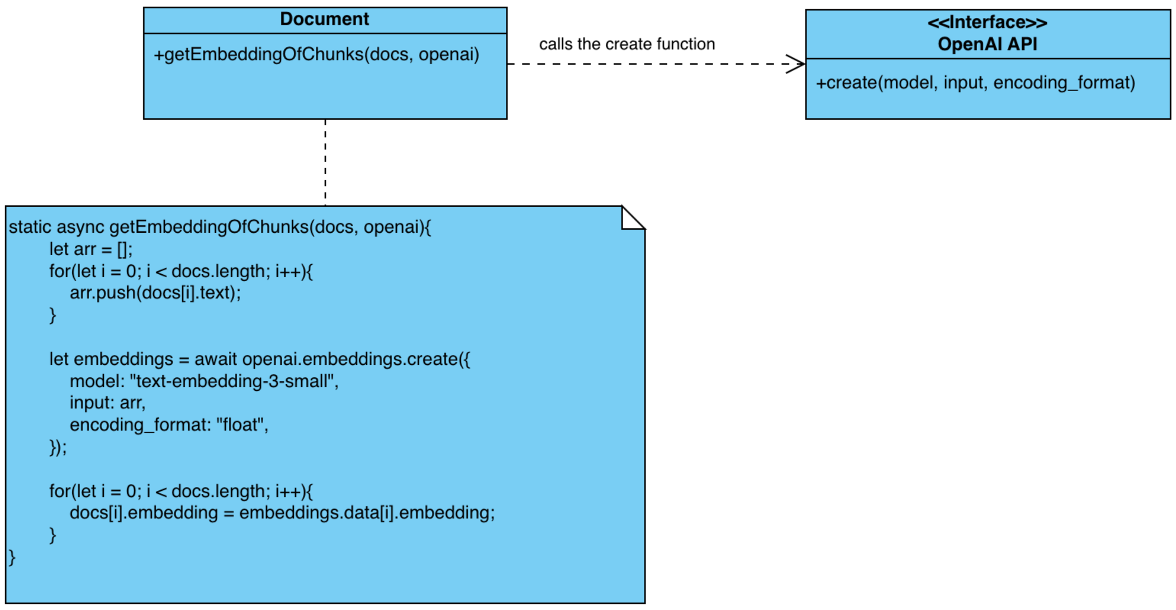
Adapter Design Pattern

The Adapter Design Pattern is a structural pattern that is used for connecting objects with incompatible interfaces. In other words, it acts as a bridge that connects the objects, without changing the original code. In our case, we have used this design pattern in order to use the OpenAI API, in order to meet the expectations of our application by abstracting the usage of OpenAI API and using the API with an interface that is compatible with our application. Lastly, by encapsulating the API usage, it became easier to further replace or update our application in the future. For example, another LLM service can be used instead of OpenAI API, by only changing the adapter classes, which is a good practice for maintainability issues.

Adaptee: The OpenAI API, which provides a set of functionalities such as: text to embedding conversion, LLM for generating responses to queries…

Adapter: Document class, which wraps the functionality of converting the texts to embeddings.


Document
+getEmbeddingOfChunks(docs, openai)
calls the create function
<<Interface>>
OpenAI API
+create(model, input, encoding_format)

```
static async getEmbeddingOfChunks(docs, openai){
        let arr = [];
        for(let i = 0; i < docs.length; i++){
            arr.push(docs[i].text);
        }

        let embeddings = await openai.embeddings.create({
            model: "text-embedding-3-small",
            input: arr,
            encoding_format: "float",
        });

        for(let i = 0; i < docs.length; i++){
            docs[i].embedding = embeddings.data[i].embedding;
        }
}
```

The application expects an interface function that is described in the Document class, more specifically the getEmbeddingOfChunks(docs, openai) function where the docs is an array of Document objects, the openai is the OpenAI object that manages the API calls. By using the getEmbeddingOfChunks function, functionality of create function has been achieved, in a way that is compatible with the Document objects.