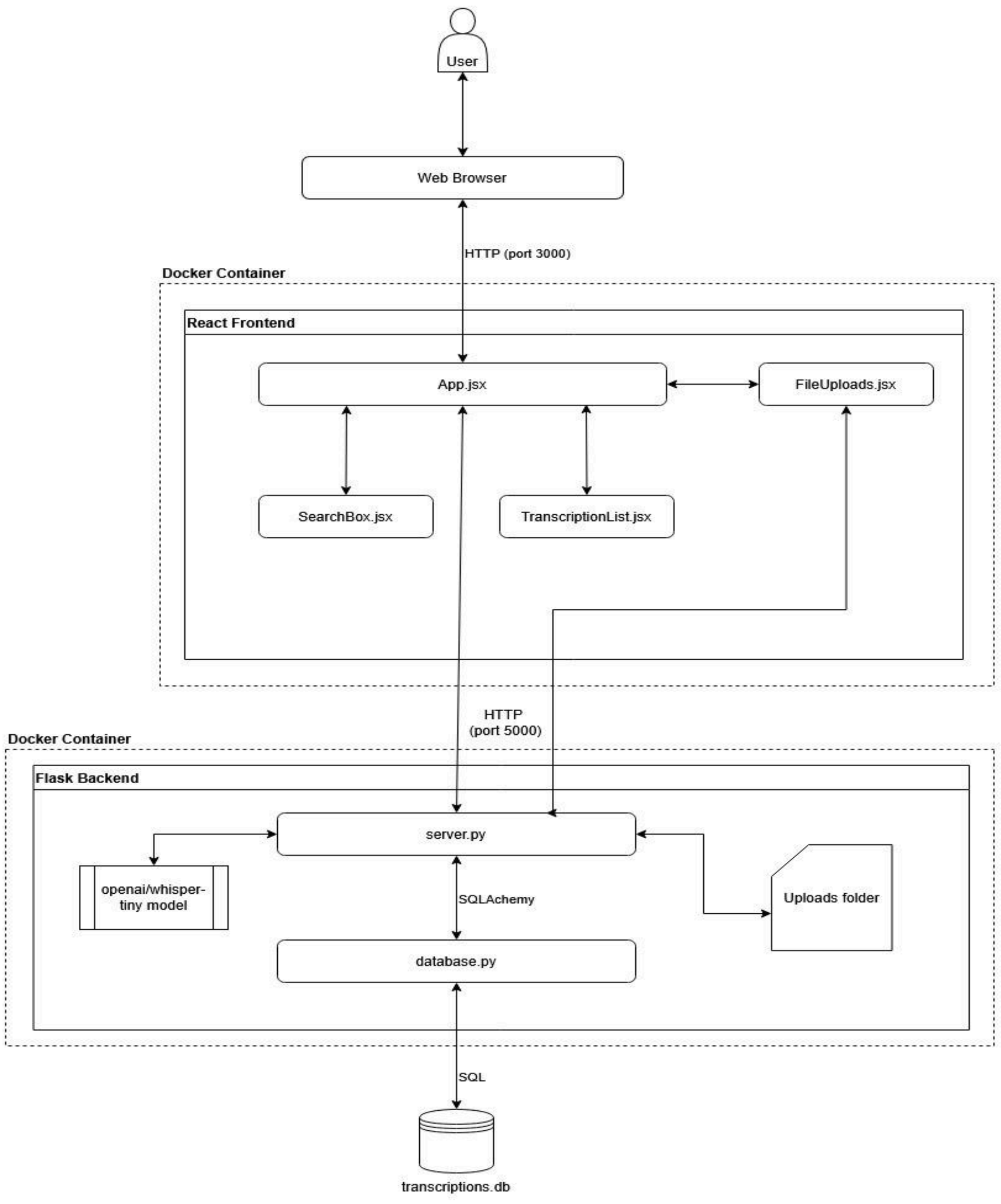
Audio Transcription Application Architecture Design

This document aims to outline the architecture of the Audio Transcription Application with the help of an architecture diagram, and is substantiated with further explanation regarding the design considerations and assumptions made.

Architecture Diagram



Overview of architecture

The application is split into the frontend component and backend component, each containerised using Docker. There are a few reasons for doing so:

1. **Isolation**
   - Each of the services can run in its own environment, preventing conflicts in dependencies, libraries, runtime versions, etc.
2. **Separation of Concerns**
   - The frontend is using React while the backend is using Flask. The different stacks will have different requirements and lifecycles.
3. **Easier Maintenance**
   - Updating or restarting one service will not affect the other.
4. **Portability**
   - Each of the containers can be run on any system that has Docker, regardless of the host OS or installed software.

Frontend Architechture

The frontend of the application is made up of the following components:

- App.jsx (Main component)
- FileUpload.jsx
- SearchBox.jsx
- TranscriptionList.jsx

App.jsx

This is the main component of the frontend that handles the main UI of the application. It handles the file uploading, search functionality and the display of transcriptions.

For the file upload interface, it will call the FileUpload component when audio files are uploaded, and display the transcriptions returned by FileUpload after transcription is complete. As this functionality is more dynamic and sequential than the search and display transcription function, both JavaScript and HTML code are put in FileUpload.jsx

The search functionality allows the user to search for specific audio files by file name. It will send a GET request to the backend based on the user's input text to query for specific audio files and display the transcriptions if they are found.

The display of all transcriptions is a toggle that will send a GET request to the backend to fetch all the transcriptions. The button allows the user to hide the transcriptions if they do not wish to display them.

FileUpload.jsx

This component handles the uploading of audio files by the user and proceeds to send the files to the backend using a POST request. A progress bar for each file will be displayed so the user is able to track the progress of the upload.

After a response is received from the backend, the transcription for the successfully processed audio files will be passed and displayed. The failed files will be logged in the console and not displayed to the user.

The file type accepted by the file upload is restricted to audio files like .mp3 and .wav. When the files are being uploaded, the ‘Upload & Transcribe’ button will be disabled and show ‘Uploading…’.

SearchBox.jsx

This component handles the display of the search box and search button. When the search button is clicked, it calls back to the handleSearch function in App.jsx.

TranscriptionList.jsx

This component handles the display of Transcription records that are fetched from the backend through the different endpoints.

Backend Architecture

The architecture for the backend is relatively simpler than the frontend, with 2 modules:

- server.py
- database.py

The Whisper-Tiny model is integrated as a pipeline directly in server.py, and the uploads folder is shown in the architecture diagram to represent how key backend components interact, even though these elements are not standalone modules within the application.

server.py

This is the main module for the backend of the application. It initialises the Flask application, loads the Whisper-tiny model into a pipeline for transcription and sets up the various endpoints with their functionalities

The first GET endpoint /health is a simple health check endpoint to verify that the server is running. The frontend does not call this endpoint, but developers can test this endpoint by directly calling http://localhost:3000/health in their browser or in Postman.

The second POST endpoint /transcribe handles the transcription functionalities. It accepts the audio file sent over from the frontend and stores it in the upload folder first. Next, the file in the folder will be fed into the pipeline with the Whisper-tiny model for transcription. The transcription will then be stored in the database together with the audio file name and timestamp. The audio file name, timestamp and the transcription are then returned to the frontend as a JSON object to be displayed.

The third GET endpoint /transcriptions queries all the transcription records in the database, with the newest records first and returns them as a list of JSON objects to the frontend for display.

The final GET endpoint /search will search the database for the file name that partially matches the query entered by the user. The search functionality ignores the extension of the file name, like .mp3, to avoid cases where all the audio files with the .mp3 extension are returned when the user searches for file names that contain 3. It then returns the search results as a list of JSON objects to the frontend, similar to the /transcription endpoint.

database.py

The database.py module sets up the database layer of the backend using the SQLAlchemy library for the server.py module to interact with the database more easily. It has the following functionalities:

1. Configures the database connection
2. Creates the SQLAlchemy engine and session
3. Defines the Transcription Object-Relational Mapping model (ORM) that allows the table in the database to be represented as a Python class