# Week 1 Objectives

- [ ] Setup Codebase
- [ ] Setup Docker build
- [ ] Implement audio recording part of the api
- [ ] Implement a script to test the recording api

## Setup codebase

- Using your choice of code management tools setup the initial spring boot codebase
- The code base at this point in time can be the initial commit of the springboot application.

## Setup Docker build

- [ ] Setup Dockerfile to build the Springboot application
    - [ ] https://medium.com/@office.yeon/dockerizing-with-multi-stage-builds-in-spring-boot-multi-module-project-1fd3aa886afc
- [ ] Use Docker compose to build and run the container
    - [ ] Our aim is to be able to build our image using `docker compose build`
    - [ ] and to be able to run our image using `docker compose up -d`
    - [ ] Volume mounting

## Implement audio recording

Reminder of what our API should look like:

```
Unset
POST /api/consult/{uuid}

Request Body:
{
   audioData: String
   sampleRate: Long
   sampleDepth: Long
   sampleLength: Long
   psuedoTimestamp: Long
}
```

**audioData** base64 encoded raw audio bytes (Encoded as LPCM)
**sampleRate** is the original sample rate of the audio i.e. 44100
**sampleDepth** is the original depth of the audio i.e. 16
**sampleLength** represents the length of the audio bytes. i.e. if we audio take every 2 seconds from the audio stream, then sampleLength should have the value 2000
**pseudoTimestamp** represents the start time of audio data relative to the first sample in milliseconds

How the API is intended to work is for the duration of a recording, every 2 seconds the caller will send in the last 2 seconds of audio data.

To keep things simple at the start we assume that we receive the audio bytes in order. We can create a new file with the format `{uuid}.bin` where the `uuid` is the uuid passed in via the request. For each `audioData` we receive we can keep appending it to the file.

To check our audio file we are storing is correct what we can do is create an endpoint to retrieve the file back:

```
Unset
POST /api/consult/{uuid}/audio-file

Response: The generated audio file
```

What this endpoint needs to do is take the `{uuid}.bin` file and pass it through ffmpeg to convert it to a `.wav` file. From there you can return the file over the HTTP request.

```
Unset
 "ffmpeg -f s16le -ar 44100 -ac 1 -i {your_file.bin} -acodec pcm_s16le -o
output.wav"

-f s16le // Specify the input format as signed 16-bit little-endian
-ar 44100 //Sample rate 44.1khz
-ac 1 // 1 audio channel
-i input file
-acodec pcm_s16le // encoded as PCM, 16 bits little endian
-o specifies output file
```

https://www.baeldung.com/run-shell-command-in-java

Here is an example of how to do it for an image or pdf, which you can adapt for a media file:
https://www.geeksforgeeks.org/returning-an-image-or-a-file-with-spring/

# Implement a script to test the recording api

To test our API works for receiving audio the easiest way is to write a python script that hooks into our local microphone. Here is a script that explains a simple way to do so: https://gist.github.com/mabdrabo/8678538 You can adapt this script to instead of writing to a file, sends that data to our REST api endpoint (once you have the springboot app running locally).

Once you're done recording, kill the script and use a tool like postman to make a request to the `/api/consult/{uuid}/audio-file` endpoint and listen if the audio file recorded by the backend is what we were speaking into the microphone.