Take Home Project

This take-home project is specifically designed to evaluate hands on programming skills for our future Full Stack Engineer/Backend Engineer/Frontend engineer. Problem statements in this project align very closely with day-to-day tasks our engineers do. Hence this is an honest & fair attempt to evaluate our future engineers with something relevant what he/she is likely to work on.

We want to respect your time commitment. Typically, this project takes 2 to 3 hours to finish. If you think it is taking longer than that, you can stop after 2 to 3 hours and submit it.

Use github.com to submit your solution. Once finished, email us the github repository link. Include a README file.

We use Python almost exclusively in our backend stack. It is indeed preferred if you can do the backend work in Python. However, if Python is not your first language of choice, that's fine. Feel free to pick one that you're most comfortable with. Just include a README file with instructions on how to run it.

ReactJS is our framework of choice for front end. Although we'd love to see if you can code front end using ReactJS, you're free to use any other JavaScript framework that you think you're strongly skilled in. Just include a README file with instructions on how to run it.

Bonus Points: Software development is incomplete with testing. If you add Unit tests you get Bonus Points. That also signals strongly completeness & readiness

If you're a Full Stack engineer, refer all sections in following problem statement.
If you're a Frontend engineer, refer problems in sections 1.1 & 1.3.
If you're a Backend engineer, refer problems in sections 1.1 & 1.2.
If you're software architect/lead, implement & tell us & preferable do what else need to be done here to make it close to production software.

Problem 1: Build a Full Stack Application with a given dataset.

1.1 Data Processing.

Attached is a sample dataset of songs playlists in JSON format. Each JSON file has multiple maps & each map has 100 key-value pairs. Key is an incremental integer whereas value is an attribute of a song.

Example:


```
{
        "id": {
            "0":"5vYA1mW9g2Coh1HUFUSmlb",
```

```
        "1":"2klCjJcucgGQysgH170npL",
        "2":"093PI3mdUvOSlvMYDwnV1e",
        "3":"64yrDBpcdwEdNY9loyEGbX",
        "4":"2jiI8bNSDu7UxTtDCOqh3L"
    },
    “title”: {
        "0":"3AM",
        "1":"4 Walls",
        "2":"11:11",
        "3":"21 Guns",
        "4":"21"
    }
}
```


This example JSON has 2 attribute maps `id` and `title`. Each map has 5 key-value pairs. Key is an incremental integer, whereas value is *song-id* and *song title* respectively.

Write a working program that takes this JSON file as input and normalizes it to generate a table as below. You can keep this normalized data in memory or in database or write to a file.

Normalized data should look something like below:

index	id	Title	danceability	energy	mode	acousticness	tempo	Duration_ms	Num_sections	Num_segments
0	5vYA1mW9g2Coh1HUFUSmlb	3AM	0.521	0.673	1	0.00573	108.031	225947	8	830
1	2klCjJcucgGQysgH170npL	4 walls	0.735	0.849	0	0.212	125.972	207477	7	999
2										
3										

1.2 [Backend] API to serve this normalized data.

Write a backend REST APIs to serve normalized data generated in step 1.1. We believe APIs for enabling following two use cases are MUST HAVE in order to make it practically useful backend project.

1.2.1 [MUST HAVE] Front end should be able to request ALL the items in a normalized data set.
Bonus point if you can implement pagination in API.

1.2.2 [MUST HAVE] Given a title as input, return all the attributes of that song

1.2.3 [NICE TO HAVE] User should be able to rate a song using star rating (5 start being highest). Please be mindful of the fact that this requires normalized data to have one more column say star rating.

1.2.4 [BONUS] Write Unit tests

1.3 [Fron-End] Build dashboard to view this data on a webpage.

Build a webpage using JavaScript (using ReactJS or AngularJS or Bootstrap) to show the normalized data in a tabular format. Tasks 1.3.1 to 1.3.6 are MUST HAVE. Remaining are optional and nice to have.

1.3.1 When application is loaded on launch, it should make an API request to backend requesting ALL the items in a normalized data set.

1.3.2 Response received should then be rendered in a tabular view.

1.3.3 Table should Show all the items in a batch of 10 rows at a time. Use pagination in table view to see all the batches of 10 rows.

1.3.4 Make each column of the table sortable. Clicking on the column should toggle the column in Ascending or Descending order.

1.3.5 Provide an option to download all the data shown in a table in CSV format at a click of a button.

1.3.6 Allow user to input song title. Put a button named `Get Song` and on click on that button should make a request to backend and fetch a row if there exists on to match the title user entered.

1.3.7 If you implemented an API from 1.2.3 above (that allows star rating a song), show one more column to the end of the table with 5 unmarked stars. When user gives star rating by marking these stars, make an API call to backend to update that song rating in generalized dataset.

1.3.8 Build a scatter chart for the songs using *danceability* value.

1.3.9 Build a histogram using song *duration* values (in seconds).

1.3.10 Build bar charts for the *acoustics* and *tempo* value

1.3.11 [BONUS] Write Unit Tests