Week 4 YTM project launched

NPM project build

we learned about the basic use of NPM, i.e. installation and introduction. In some large projects, such as vue and other open source front-end templates, we only need to go through two lines of command:

```
git clone xxx

npm i
```

You can quickly install and build project dependencies directly, and be able to directly start a project using commands such as npm run dev.

For example, use `npm run dev` Start a project directly, how is this done?

The construction of a Node project relies on a file called package.json, and most of the tasks performed by npm are related to it.

http://nodejs.cn/learn/the-package-json-guide

http://nodejs.cn/learn/the-package-lock-json-file

Task1

Create a project called YTM-KOA and temporarily introduce axios, koa, koa-jwt, music-metadata, jpeg-js dependencies， and then use `npm start` command to run the script：

```
node main.js
```

In the ytm-koa folder, create a new file main.js and write the following:

```
const Koa = require('koa');

const app = new Koa();


app.use(async ctx => {
  ctx.body = 'Hello World';

});


app.listen(3000);
```

Run the following code

```
npm i

npm start
```

The project is successfully launched and access localhost:3000 to see Hello World.

Local simulation environment build

Since the internship environment could not provide a database, we simulated the mongoDB database in a local way, and the basic format of mongo is actually Json.

It is required to implement a libraryInit (path) method, the function is to read all audio files under /Library when the APP is started (users need to put some MP3 files in advance, the number is about 100), obtain the tag information of the file through the music-metadata library, and generate index.json files to store in /Library in the following format:

```


    "track_id": "",

    "title": "",

    "artist": ["", ""],

    "album": "",

    "album_id": "",

    "genre": "",

    "copyright": "",

    "length" : "",
    "track_number": 0,

    "quality": "STD",

    "file": "file path"

}
```

Among them, the generation of track_id adopts the method of concatenating the three strings of artist, title, and album and using MD5 encryption to take the first 16 characters. track_id is the unique index value of the track in the music library, so different songs cannot be repeated. (That is, track_id is not strictly unique in the database, and there will be multiple sound quality files corresponding to the same song in the later stage)

For more information about the Tag Key corresponding to the information, please refer to https://www.npmjs.com/package/music-metadata。

Also, if there is an Album Artwork Image, the imageBuffer in the tag is stored in /Library/cover/<album_id>.jpg. Note: Sometimes imageBuffer is PNG, so you need to use the jpeg-js library for encoding and saving, and the quality is 100.

This JSON file is sequential and stored in a separate line structure, i.e. one of the above objects per line.

The database creation of each file should be asynchronous and parallel, using various methods to control parallel threads within a limited number. The specific software behavior is that after building a list of MP3 files through methods such as walk, an asynchronous method similar to indexCreate (filePath) is executed for each file in the list in the thread pool of an n thread. This function is responsible for reading the id3Tag of the file, getting the above information, writing the JSON variable, dumping the imageBuffer to the file, and then blocking the writing to /Library/index.json. (i.e., if index.json is being used by another thread, the function should wait until it can access the file) After each file is created, log the following structure in the console:

```
Index Created: <track_id> <file>
```

The number of threads in this thread pool n should be determined by obtaining information about the CPU through the os module. At the same time, write a libraryLoad (filePath) method, which directly reads index.json and loads it into memory, returning object.

Write a libraryUpdate (lib, filePath) method to update the index.json file, which contains:

1. Compare and add unindexed MP3 files
2. Delete entries that do not find the corresponding file in the local search

Ps: This library is unique to the user's index, track_id. At the file level, however, the file path is the only index value. The above function name is self-proposed.

libraryLoad(filePath) and libraryUpdate(path) should occupy the file until the end of the running process and then release the hold. So now the logic is very clear: after starting the project at 'npm start', first try 'libraryLoad(filePath)', and run 'libraryInit(path)' if the file does not exist. If the file exists, mount '.then((lib) => { libraryUpdate(lib, filePath) })' to complete the loading and refreshing of the index library.

Data on the "cloud"

In this project, we use the non-relational database mongoDB to build the database system of the whole system.

Basic concepts and an introduction to mongo:
https://www.runoob.com/mongodb/mongodb-intro.html

In node, we use the mongoose library to handle Node's interaction with MongoDB. In fact, Node natively integrates support for MongoDB .js, but the mongoose middleware is easier to use. Mongoose Chinese manual:
http://mongoosejs.net/docs/index.html

After installing the MongoDB library locally, we need 4 databases: users, library, playlists, history, for subsequent projects.

Among them, the data structure of the library has been pointed out earlier, the total collection is named index, and the user's private library collection is named u_\<uid\>, and the structure is as follows:

```
{

   "type": "track / album / playlist",


     "id": "id",


     "added_date":

}
```

The data structure of users is as follows, and there is no need to manage collections:

```
    {
            "uid": "",

            "name": "",

            "secret": "",

            "subscribe": "Premium",

            "subscribe_expired":,

          "last_login": ,

          "playing": "machine_id"

}
```

playlists are divided into total index collection and single-list collection, and the structure is as follows:
The total collection is named index

```
{

    "pid": "id",

    "author": "uid",

    "name": "",

    "description": "",

    "added": 0,

    "liked": 0,
    "shared": 0,

    "played": 0,

    "public": true,

    "image": "path",

    "type":"playlist / album",

    "last_update":

    }
```

The single-list collection is <pid>:

```
{
    "tid": "track_id",
    "order": 0          // order
}
```

At the same time, replace all the file processing related functions of the library built in the previous section with the operation of the Mongose database with Mongoose, and migrate index.json to MongoDB, but retain the content in the cover, and still store it in the form of a local file path index.

When writing data to a database such as MongoDB, you can ignore the blocking problem, and the middleware and database engine will handle the concurrency problem by themselves.