MusicHoster

MusicHoster is a web application that provides a forum where we can upload the video

After we have finished the code for the API endpoints and tested them, we must now provide a user interface which can help users take advantage of the backend code.

• BRIEF UNDERSTANDING OF Musichoster WEBAPP:

To design the frontend, we must first be familiar with the functionality of the web application we are developing. This will help us in designing a basic site map of the web application.

Here are the specifications of the MusicHoster web application:

- **USER**
  - **Sign Up: Users can create their accounts**
  - **Sign In: Users can login into their accounts**
  - **Sign Out: Users can sign out of their accounts on the web application.**

When the user is successfuly signIn he can successfully upload the music files.

• THE SITEMAP:

Now based on the understanding of how backend works and what is expected out of our web applications, we will now look at the pages necessary for our web application.

These are listed below:

- **Home Page**
- **Sign In Page**
- **Sign Up Page**
- **Dashboard (It will be display after the user is succesfully signup)**

Description of the pages are:-

Home Page:- It should contain a navbar with the SignIn and SignUp button .

Sign Up:-

It should contain:-First Name ,Last Name,Email Address,Password,Mobile Number

Sign In:-Email Address and Password

DashBoard:-It should contain a navbar with the logout button and in the page we can upload the music.

BACKEND DEVELOPMENT

In this project, you will work on developing REST API endpoints of various functionalities required for a MusicHoster app from scratch. In order to observe the functionality of the endpoints, you will use the **Swagger** user interface and store the data in the **PostgreSQL** database. Also, the project has to be implemented using **Java Persistence API (JPA)**.

This is a group project, you would be working in a group of 3 or 4 students and there would be one final submission. Use Git and GitHub to conduct version control of your assignment code throughout your assignment development.

- As you have learnt in the version control module, it is a good software engineering practice to use version control while developing software.
- In your submission, include a link to your GitHub repo that contains the course 5 project code.

Github Collaboration Instructions

One of the team members should act as the project lead, create a master repository for the project, and push the initial code stub to the master repository. After which, the project lead would create different branches for different functionalities to be developed, and share the repository URL with other team members.

The other team members should then fork and clone the master repository to their own repository on GitHub, so they can work on a specific branch and make updates on the project via pull requests. Also, it would be the project leader's responsibility to merge the pull requests into the master repository. It is always a good practice for each member of the team to review a pull request before it is merged into the master repository and give your comments on the pull request to help the project leader.

Lastly, if you are working off a fork, don't forget to **fetch from the upstream repository** often, so you can get the latest commits and updates of the various branches in the upstream repository. Also, once all the required code implementation is done on a specific branch and is working fine, then the project lead can go ahead and merge the branch with the master repository.

A few additional notes to help your collaboration between teammates:

1. Use Github to **track issues and bugs** for the project.
2. Use Github to **conduct code reviews**, so each push request or commits are reviewed by another teammate before the code changes are merged into the main repository.

Database Schema

The database schema required for the project is designed and provided, as shown below.As we will be using JPA repository, so we have to follow the similar approach to create database entities using @Entity model approach using javax.persistence library.

TABLE DEFENITIONS

USERS MUSICS

1.	UserId	musicId
2.	firstName	music
3.	lastName	name
4.	email	description
5.	Mobile	user_id(FK)
6.	password	TIMESTAMP

- You need to manually create a database named "MusicHoster" in your PostgreSQL.
- You don’t have to create the relations manually, create the model classes with the respective annotations from javax.persistence package.
- You need to do the mappings for the relations. E.g; @OneToMany, @OneToOne, etc wherever needed even bi-directional mappings.
- You need to update the environment variables such as server port, database name, database password in application.properties file in the folder located in src/main/resources to integrate the database into your system with the project.

Let us recall the concept of the foreign key when a column in a table references the primary key of some other table for its reference. The table containing the primary key is a parent table and the child table contains a foreign key. When a table is related by some other table in the database and you try to delete a record from the parent table, what will happen? PostgreSQL gives the following option:
DELETE CASCADE - In this case, all the referenced records in the child table will be deleted first and then the parent record will be deleted.

In the MusicHoster project, we have used DELETE CASCADE option to delete all the referenced records in the child table first and then the record in the parent table. You can use **@OnDelete(action = OnDeleteAction.CASCADE)** annotation in JPA to specify the foreign key attribute in the Java class for DELETE CASCADE option.

Project Structure

The project must follow a definite structure in order to help the co-developers and reviewers for easy understanding. So, for better understanding follow the below project structure :-

MusicHoster(Project folder)

- ➔ Model
- ➔ Controller
- ➔ Repository
- ➔ Security
- ➔ Service
- ➔ Filters
- ➔ Exception

  For Exception handling Please follow the below commit

  Added Exception · rahul10-pu/techblog@0d90027 (github.com)

  You can refer to the technicalBlog Application for the project structure

  GitHub - rahul10-pu/techblog: Backend Development for a Blogging Application.

In this project, you will learn how to develop REST API endpoints in the following controllers:

1. **Signup-controller:** In this controller, the user will able to sign up for an account.
2. **Authentication-controller:** After signing up, the user needs to sign in. This controller authenticates the user based on the credentials provided. After

authentication, the user will be given an ‘access token’, which will be required to perform any further operation.

3. **Music-Upload-controller:** Using the ‘access token’, the user can upload music files through this controller.

Spring, Unit Testing, and Mocking

1. For testing, mocking, adding dependency & testing http endpoints check the commits for reference below:-

   junit postController · rahul10-pu/techblog@fcbbe1d · GitHub

HTTP status

As you have already learnt about different HTTP response status codes, implement the same when creating the API endpoints and return the corresponding HTTP status code based on the functionality or message. The most commonly used response codes in this project are as follows:

- HttpStatus.OK
- HttpStatus.CREATED
- HttpStatus.UNAUTHORIZED
- HttpStatus.FORBIDDEN
- HttpStatus.NOT_FOUND

Version Control Best Practices

Please follow the following best practice as you are using Git and Github to conduct version control of your code. This will make you a more effective software engineer.

- Commit often

- Make small, incremental commits
- Write good commit messages
- Make sure your code works before committing it

Here are some additional readings on best Git practices:

- **Git Common Practices**
- **Commit Often, Fix it Later, Publish Once- Git Best Practices**

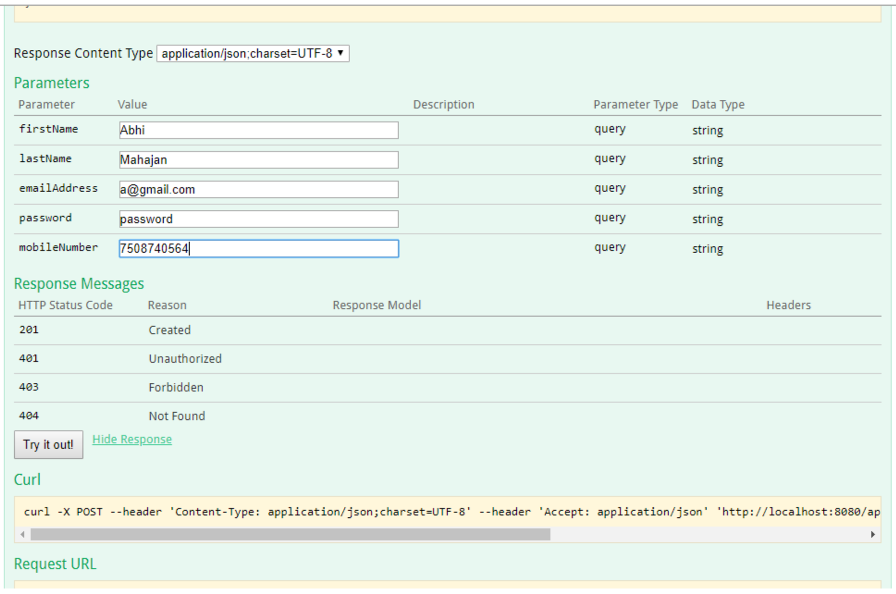
TECHNOLOGY TO BE USED IN THIS PROBLEM STATEMENT:

1. SPRING BOOT
2. SPRING DATA JPA
3. ADVANCED JAVA OOPS CONCEPT
4. EXCEPTION HANDLING
5. JUNIT, MOCKITO
6. WEB-MVC
7. JWT AUTHENTICATION
8. POSTGRESQL
9. SWAGGER/OPEN-API
10. ORM FRAMEWORK

Signup:

In this segment, you will develop signup-controller, which contains the '/usersignup' endpoint :

- It should be a POST request
- Create a new user by passing the required request payload.
- Once the user enters the password, you will need to write a code to store that password in an encrypted form in the database.
- Return a success message in the response "USER REGISTERED SUCCESSFULLY".


Response Content Type application/json;charset=UTF-8

Parameters

Parameter	Value	Description	Parameter Type	Data Type
firstName	Abhi		query	string
lastName	Mahajan		query	string
emailAddress	a@gmail.com		query	string
password	password		query	string
mobileNumber	7508740564		query	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

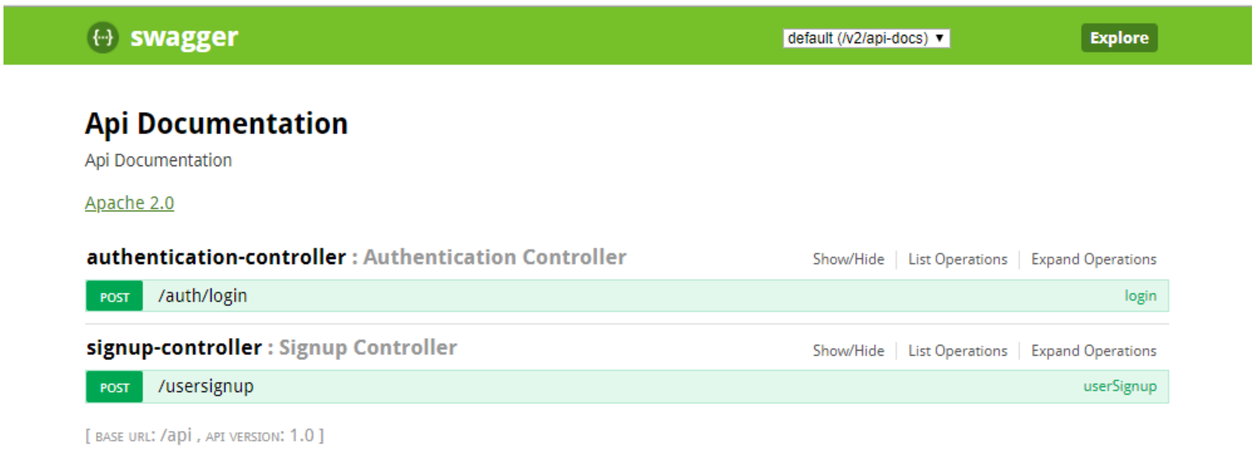
Curl

```
curl -X POST --header 'Content-Type: application/json;charset=UTF-8' --header 'Accept: application/json' 'http://localhost:8080/ap
```

Request URL

Authentication:

In this segment, you will develop authentication-controller, which contains the ‘/auth/login’ endpoint. Any user who wishes to access the functionalities of the Music Hoster application needs to authenticate himself/herself by providing valid credentials.


swagger
default (/v2/api-docs)
Explore

Api Documentation

Api Documentation

Apache 2.0

authentication-controller : Authentication Controller

Show/Hide | List Operations | Expand Operations

POST /auth/login login

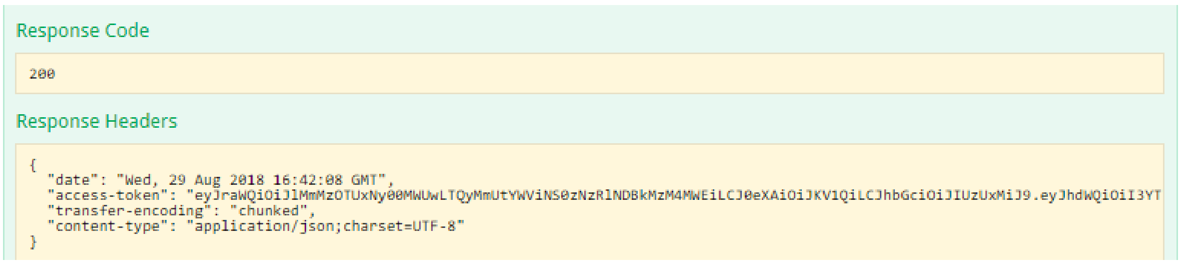
signup-controller : Signup Controller

Show/Hide | List Operations | Expand Operations

POST /usersignup userSignup

[BASE URL: /api , API VERSION: 1.0]

If the credentials of the user's is correct, you will receive the following response:

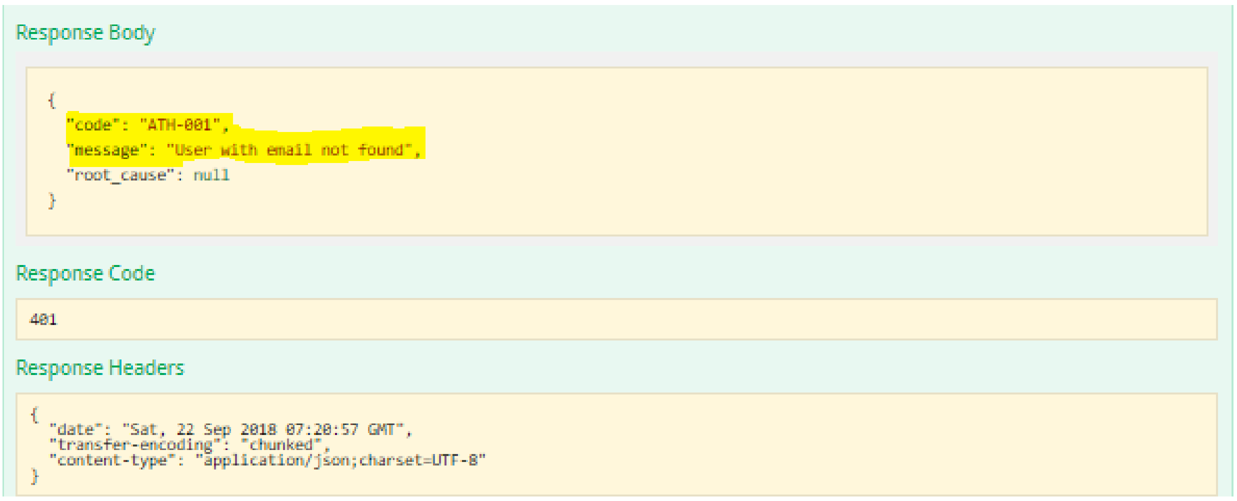

Response Code

```
200
```

Response Headers

```
{
  "date": "Wed, 29 Aug 2018 16:42:08 GMT",
  "access-token": "eyJraWQiOiJlMmMzOTUxNy00MWUwLTQyMmUtYWViNS0zNzRlNDBkMzM4MWEiLCJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJhdWQiOiI3YT
  "transfer-encoding": "chunked",
  "content-type": "application/json;charset=UTF-8"
}
```

If the user's email address is incorrect, you will get a message in the Response Body indicating that the user with that email is not found.


Response Body

```
{
  "code": "ATH-001",
  "message": "User with email not found",
  "root_cause": null
}
```

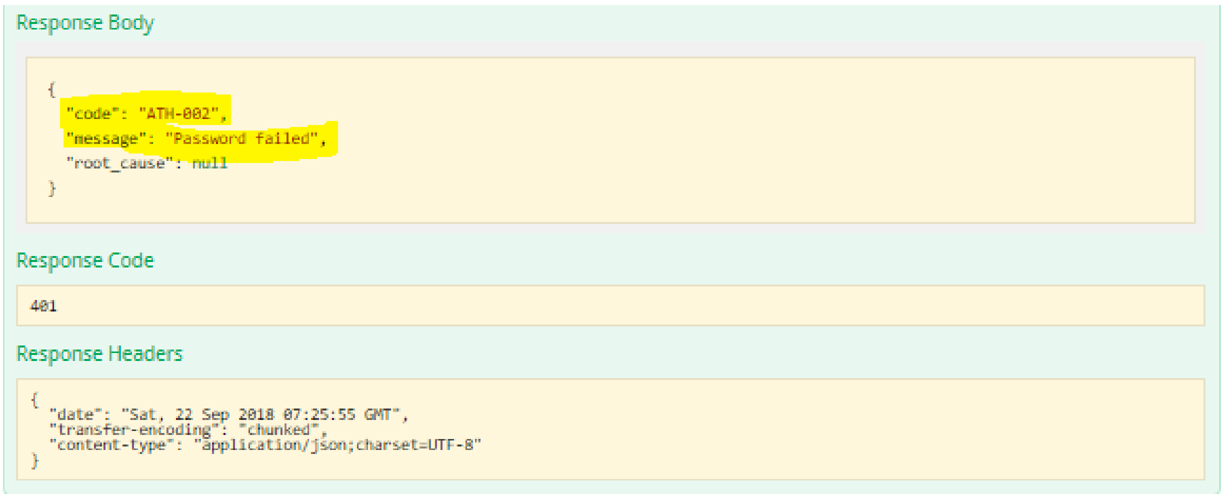
Response Code

```
401
```

Response Headers

```
{
  "date": "Sat, 22 Sep 2018 07:20:57 GMT",
  "transfer-encoding": "chunked",
  "content-type": "application/json;charset=UTF-8"
}
```

If the user's password is incorrect, you will get a message in the Response Body indicating password failure.


Response Body

```
{
  "code": "ATH-002",
  "message": "Password failed",
  "root_cause": null
}
```

Response Code

```
401
```

Response Headers

```
{
  "date": "Sat, 22 Sep 2018 07:25:55 GMT",
  "transfer-encoding": "chunked",
  "content-type": "application/json;charset=UTF-8"
}
```

Music-Upload:

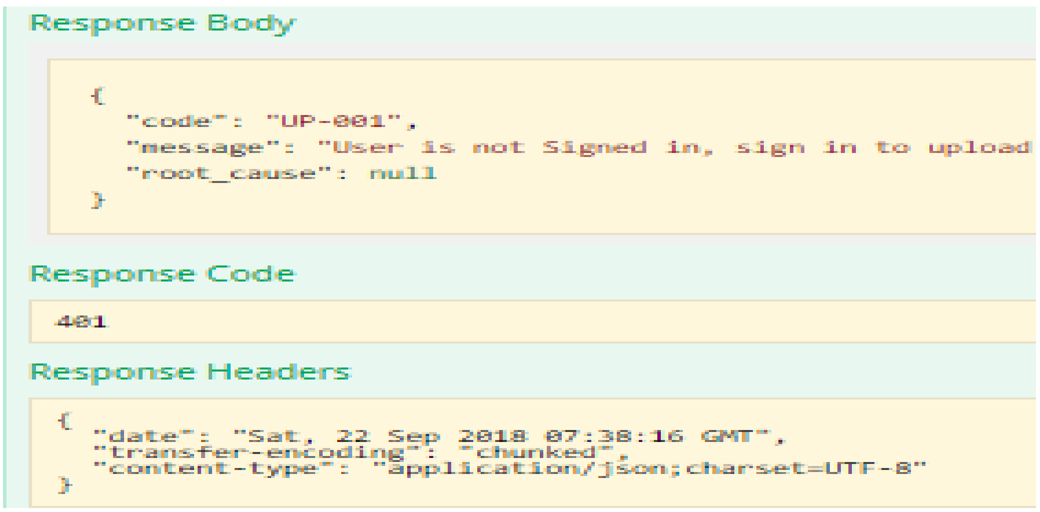
Now you have created **signup-controller** to signup and **authentication-controller** to sign in to the Music Hoster application. In this segment, you will work on **music-upload-controller** to upload a music file. The user needs to enter the access token in the authorisation header to upload a music file url, provided to the user in Response Header at the time of signing in to the Music Hoster application.

Since the size of music file is large, storing and retreiving music files from database is not practical. Therefore here the approach is to store the **url** of the music file instead.

Music-Upload-Controller: '/musicupload'

- It should be a POST request
- Upload a music by passing the required request payload.
- Return a success message in the response “MUSIC UPLOADED SUCCESSFULLY”.

In case of wrong access token being entered, the message in the Response Body says 'User is not Signed in, sign in to upload'.


Response Body

```
{
  "code": "UP-001",
  "message": "User is not Signed in, sign in to upload
  "root_cause": null
}
```

Response Code

```
401
```

Response Headers

```
{
  "date": "Sat, 22 Sep 2018 07:38:16 GMT",
  "transfer-encoding": "chunked",
  "content-type": "application/json;charset=UTF-8"
}
```