MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 495 – CLOUD COMPUTING

25-26 FALL – TERM PROJECT

CLOUD BASED MUSIC SHARING & RATING PLATFORM

Name: Ahmet Caner

Surname: Karaca

ID: 2585123

Project: Cloud-Based Music Sharing & Rating Platform

Report: Phase 2 – Progress Report

TABLE OF CONTENTS

LIST OF FIGURES

1-GENERAL INFORMATION ABOUT PHASE 2

Incremental approach is carried out during the implementation of the project up until phase 2. The back bone of the backend side of the project is implemented with efficient database integration and enhanced overall project architecture. This phase also contains security concerns, data integrations and interactions of backend components. Detailed API service testing, debugging and evaluating the system partially has been the primary objective.

2-TECHNOLOGIES INCLUDED SO FAR

2.1 Spring Boot:

Spring boot is a java framework that is used in this project phase to develop backend properties such as entity classes, repositories, interfaces, controllers and services. Also, provides an efficient test cases for end-point testing and development, ensuring code integrity with the application.

2.2 PostgreSQL:

PostgreSQL is used in this phase to efficiently handle database management. Fields of user, artist and song are stored in the tables generated. PostgreSQL acts as a medium to hold these tables and efficiently manage them.

2.3 Postman:

Postman is a platform that is used to test API endpoints. In this phase of the project, Postman is utilized to send http requests in order to verify the functionalities of RESTful API services. Examples will be provided in the following chapters.

2.4 Hibernate:

Hibernate ORM is a Java framework that performs object-relational mapping (ORM) to help developers persist Java objects to a relational database

3- SYSTEM ARCHITECTURE

3.1 Overview of the System Architecture

System architecture is designed based on modular and incremental approach that ensures scalability and modularity. Backend of the system consists of several layers that is providing a seperation of concerns and efficient interaction between entities. Architecture involves additional services such as cloud AWS S3 for file storage and utilizes the PostgreSQL for database management. Also, http security between the client and the server is integrated into the system using Spring Boot's built in security configurations. Overall, this features provide an efficient backbone for backend system.

3.2 Backend Architecture

3.2.1 Layered Architecture

Backend of the system consists of several layers that are in touch with each other. Each layer has a well defined functionality and role. These layers are organized into packages in Spring Boot application. Organizing the system into Controller, Service, Entity model and repository

layers allows the system to achieve a modular structure where the business logic, API request handling and data definitions remains independent from each other. Here below are the brief definitions of each package layer and their usages.

Controller Layer:

-Handles incoming http requests from the client server as mappings

-Provides the API endpoints to be tested

-Manages the mappings for CRUD operations.

-Returns http responses to the relevant client server

Service Layer:

-Implements the business logic of the application.

-Serves as a coordinator between controller and repository layers

-Ensures bussiness rules are applied efficiently and constantly.

-Processes the data between the API and the database.

Repository Layer:

-Responsible for data persistance operations.

-Main link between the application and the database.

-Utilizes the Spring Data JPA to interact with PostgreSQL

-Constructs relational entity tables using Hibernate Object Relational Mapping (ORM)

-Makes query operations efficient and easy to handle

These layers are organized into packages to ensure modular structure and each plays a crucial role in the backend coding schema and database design.

3.2.2 Database Layer

Database layer of the system is implemented using PostgreSQL which is a robust DBMS for handling structured queries and achieving seamless integration with Spring Data JPA. Within the code schema, entities that are annotated with **@Entity** model are automatically mapped to the relational tables using JPA and Hibernate. Also, each field of these entity classes are mapped to the columns through JPA annotation **@Column**. Hibernate handles these operations and design while providing an abstraction from the logic behind. Additionally, JPA manages the relationships between entities using the annotations such as **@OneToMany, @ManyToOne** and **@JoinColumn.** Here below is a brief overview of the tables generated in the database for the 2nd phase.

User:

-Stores fields such as

-unique ID

-username

-password

-email

-role

Artist:

-Contains artist specific fields, some of them include

-unique ID

-stage name

-real name

-biography

Song:

-Store information such as

-unique ID

-title

-duration

-genre

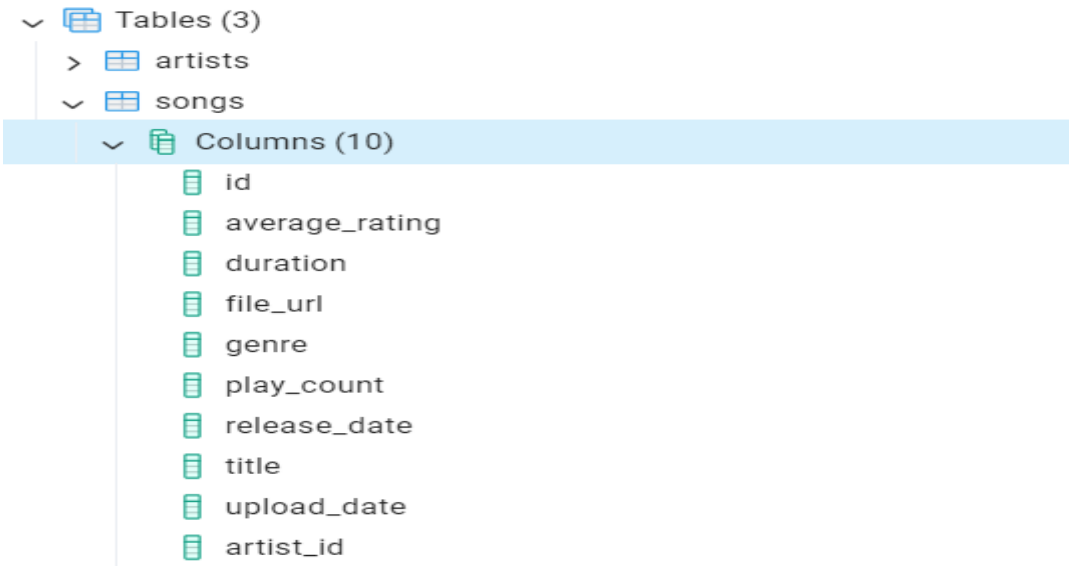

Figure 1: Example of Tables in pgAdmin 4

3.2.3 Integration of External Cloud Service

Amazon S3 is integrated into the system to efficiently handle the media storage process using cloud technologies. S3 is ideal for storing large audio files that is uploaded by artists, providing scalability and automatic scaling. To keep the media files organized and maintain a modular structure, S3 bucket system is integrated with the backend side of the application. Once the MP3 file is uploaded to the bucket, S3 returns the URL link. This URL link is not stored in the bucket, rather stored in the database table that is related with the song entity. This will benefit users to not store large amounts of mp3 storage in the backend, rather retrieving them from the cloud storage.

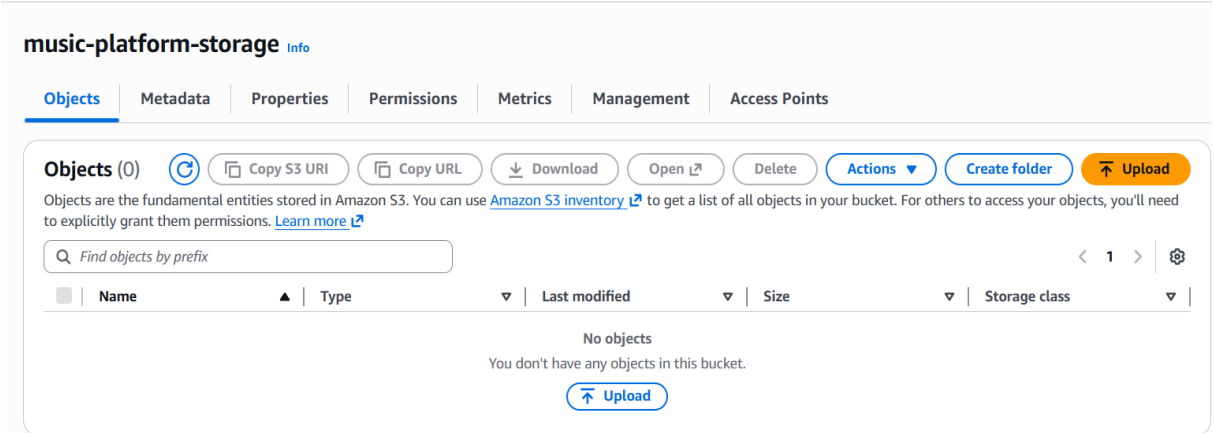

Figure 2: Amazon S3 music-platform-storage Bucket

3.2.4 General Application Request Flow

Request flow between the client and server sides follows a well-structured and pre-defined application flow path. Client interactions triggers a sequence that passes through the Controller, Service and Repository layers before arriving at the database or any other external storage services.

When an http request arrives at the backend, its first destination is the Controller layer. This layer handles the endpoints, request mappings and CRUD operations. Controller forwards the request to the service layer to handle the necessary business logic.

Service layer contains the applications fundamental business logic. This layer processes the incoming process, applies necessary operations and serves as a bridge between repository layer and any other external services.

Repository layer handles the database interactions using Spring Data JPA. It translates high-level CRUD operations into SQL queries executed on the PostgreSQL database

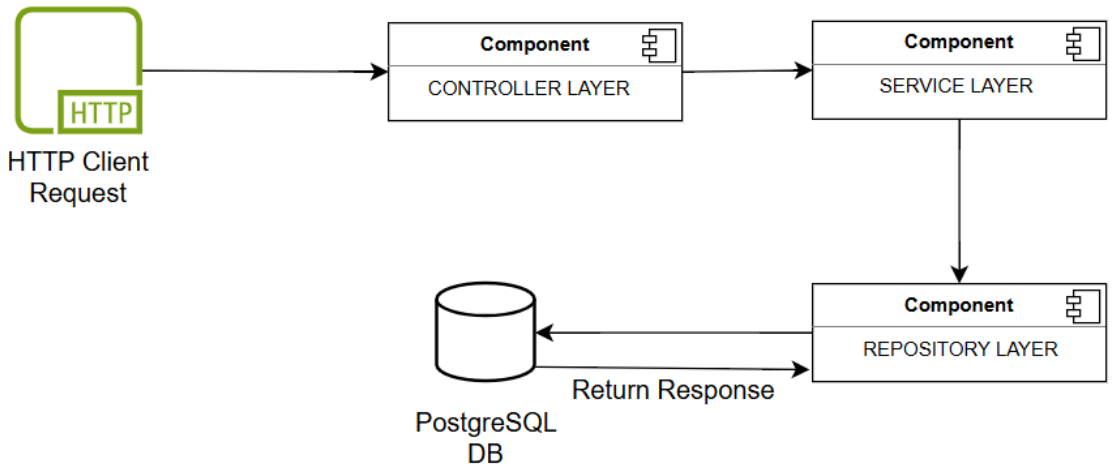

Figure 3: General Application Request Flow

4- MILESTONES ACHIEVED

4.1- OdtuClass Week 3 November – 9 November

4.1.1 Repository Creation in GitHub

Project environment and infrastructure is established during the first week of development. A dedicated GitHub repository is created in order to maintain modularity, track progress changes and pushing project related material such as reports, code segments etc. The initial Git repository includes a README file. Project is also cloned to the local computer using command **git clone /address.** Additionally, .gitignore is configured for Spring Boot files. GitHub repository structure also follows the layered architecture that is organized into packages such as controller, repositories and services etc.

Figure 4 : GitHub Repository Snippet

4.1.2 Spring Boot Initialization

Backend of the project is developed using Java framework Spring Boot. Generation of the Spring Boot application is carried out using Spring Initializr using these metadata settings:

- Group: com.capstoneproject
- Artifact: musicplatform
- Name: MusicPlatformBackend
- Language: Java
- Type: Maven
- JDK : 17
- Packaging: JAR

Various dependencies are also included in the initialization to ensure seamless workflow and efficient coding environment. Some of them are listed below:

- Spring Web: provides integration features such as multipart file upload functionality and a web-oriented application context.
- Spring Data JPA: makes it easy to easily implement JPA-based (Java Persistence API) repositories
- Spring Security: Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications
- PostgreSQL JDBC Driver: The PostgreSQL JDBC Driver allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

To include these dependencies and functionalities into the coding environment, **application.properties** file under the **resources** directory. Database configuration such as host URL, username, password and driver class name are included in this file to achieve efficient database connection. In addition, Hibernate JPA settings such as auto update in database tables, displaying SQL queries for debugging, formatting the SQL outputs, detailed error logging settings are included in the file. Here below is the initial phase of the **application.properties** file

```
spring.application.name=MusicPlatformBackend
# ================================
# = DATABASE CONFIGURATION
# ================================
spring.datasource.url=jdbc:postgresql://localhost:5432/musicdb
spring.datasource.username=postgres
spring.datasource.password=Ack1350+
spring.datasource.driver-class-name=org.postgresql.Driver

# ================================
# = JPA SETTINGS
# ================================
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=INFO
```

. Figure 5: application.properties

4.1.3 Initial Postman API Testing with Artist Entity

Backend API structure is validated through a test entity Artist. To test the API endpoints, a basic REST controller is implemented. This controller is put through the POST and GET operation tests using Postman with temporary data. Sample JSON bodies are sent to the /api/artists endpoint to validate that the various layers are interconnected successfully and provides the necessary communication with PostgreSQL database. Three initial fields (name, genre, country) are put into a POST request in JSON format. As a result, GET request from the same API, resulted in the same JSON format, verifying that the endpoints are in efficient use, and service, controller and repository layers are communicating decently. This initial testing provided a move signal before proceeding to implement the application, preventing bigger debugging sessions later in the development process. Below is the result of GET operation using /api/artists endpoint.

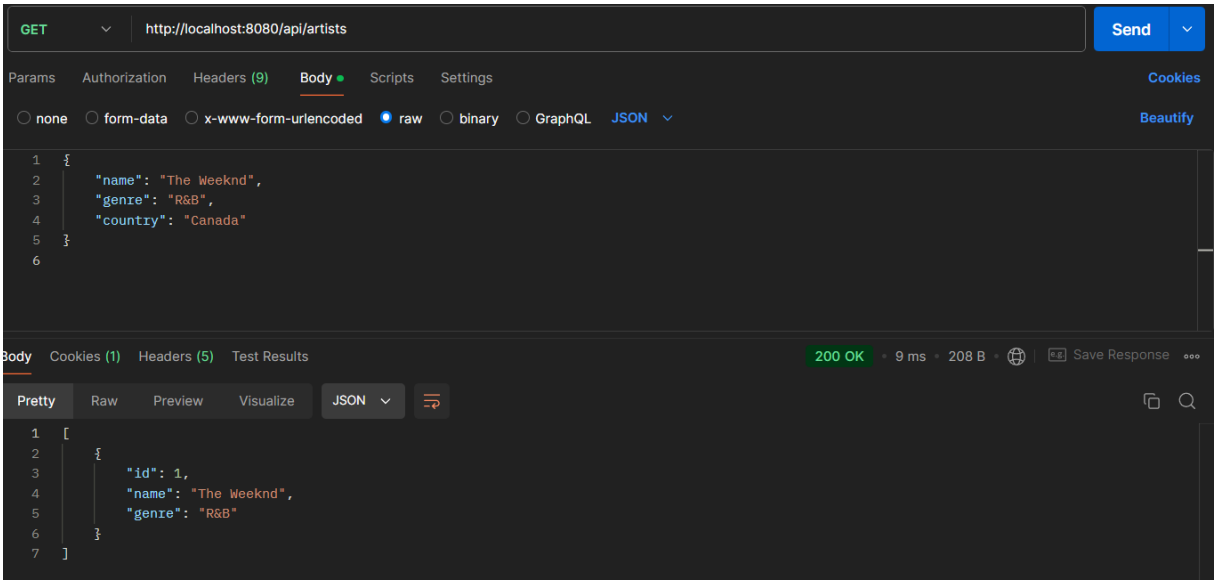

Figure 6: GET Request Example

4.1.4 Deliverables for Week 3 November – 9 November

This week mainly included the initial project setup and initialization. This initialization also contains basic testing of API endpoints and layer interactions. Here below is the provided deliverables for this week:

- A project skeleton in GitHub
- A tested and working Spring Boot backend skeleton
- MusicDB created in PostgreSQL
- API endpoints tested
- Postman is setted up
- JPA Java DataBase Connection is set
- First GitHub snippet is pushed

4.2- OdtuClass Week 10 November – 16 November

4.2.1 Construction of Core Entities

Core of the backend of the application consists of three entities – which may be extended in the final phase -. These entities include user, artist and song. Each entity is structured with necessary field and id generations and mappings between them. Entity classes are gathered together in the model package in order to maintain a structured and modular approach. JPA annotations are utilized to establish relations between the database and interconnections between entities. The system will rely on this data model throughout the implementation of the project. Detailed explanations of these entity classes are provided in the **chapter 3.2.2**. Entity classes contains essential fields, table and column definitions, getter and setter methods etc. Implementation of entity classes are pushed into the relevant **model** package in the GitHub repository. Here below is the URL link for the package of interest.

Model Package: https://github.com/CanerKaraca06/cng495-capstone-project-2025/tree/main/backend/src/main/java/com/capstoneproject/musicplatform/model

Example snippet of a core entity class can be found below:

```java
package com.capstoneproject.musicplatform.model;     //Include the model package

import jakarta.persistence.*;              //JPA



@Entity         //User entity  13 usages  CanerKaraca06
@Table(name = "users")        //Set table name from default to "users"
public class User {      //User entity defined
    @Id             //ID as the primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY)     //Generate an auto-increment value
    private Long id;         //Unique ID

    @Column(unique = true, nullable = false)     //username column, not nullable and unique  3 usages
    private String username;

    @Column(nullable = false)         //password column, not nullable  3 usages
    private String password;

    @Column(unique = true,nullable = false)   //email column, not nullable and unique  3 usages
    private String email;

    @Column(nullable = false)  3 usages
    private String role = "USER";     //Default role

    public User(){        //Default constructor  CanerKaraca06

    }
```

Figure 7: Snippet of User.java

4.2.2 Repository Development

Following the creation of user, song and artist entity classes, repository development has been carried out for each entity. Repository includes custom query methods, built-in CRUD operations. Main objective of implementing this layer is to reduce boilerplate code and provide rapid access and manipulation of data. After implementing the repositories, codes are pushed into the Git **repository** package, where relevant. Here below is the URL link for the package of interest.

Repository Package: https://github.com/CanerKaraca06/cng495-capstone-project-2025/tree/main/backend/src/main/java/com/capstoneproject/musicplatform/repository

Example snippet of User repository can be found below:

```
package com.capstoneproject.musicplatform.repository;

import com.capstoneproject.musicplatform.model.User;       //Import entity user
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;        //Null control

public interface UserRepository extends JpaRepository<User, Long> {     //Data type user, id type long  3 usages  CanerKaraca06
    Optional<User> findByUsername(String username);     //Find user in database by username - may not exist  1 usage  CanerKarac
    boolean existsByUsername(String username);         //Find user in database by username  1 usage  CanerKaraca06
    boolean existsByEmail(String email);        //Find user in database by email  1 usage  CanerKaraca06
}
```

Figure 8: Example Snippet of User Repository

4.2.3 Service Development

The service layer is mainly responsible for implementing encapsulated and abstracted business logic for each entity stated above. For example, user service is created to securely handle password and login verifications. These service layer operations are working in parallel with the repository features. Various CRUD are embedded in the service layer. Unit tests are carried out to validate that the services are implemented efficiently and coordinated with other layers successfully. Finally, these service code segments are pushed into the relevant GitHub package. Here below is the URL link for the package of interest.

Service Package: https://github.com/CanerKaraca06/cng495-capstone-project-2025/tree/main/backend/src/main/java/com/capstoneproject/musicplatform/service

Example snippet of User service can be found below:

```
package com.capstoneproject.musicplatform.service;

import ...

@Service  3 usages  CanerKaraca06
public class UserService {

    private final UserRepository userRepository;  5 usages
    private final PasswordEncoder passwordEncoder;  2 usages

    public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {  CanerKaraca06
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    public User registerUser(User user) {  1 usage  CanerKaraca06
        if (userRepository.existsByUsername(user.getUsername())) {
            throw new RuntimeException("Username already exists");
        }
        if (userRepository.existsByEmail(user.getEmail())) {
            throw new RuntimeException("Email already exists");
        }

        user.setPassword(passwordEncoder.encode(user.getPassword()));
        return userRepository.save(user);
    }
```

Figure 9: Example Snippet of User Service

4.2.4 Controller Development

REST controllers are developed to utilize CRUD operations for each of the three entities. Each of these controllers contains mappings for retrieving, updating, adding and deleting records. Testing of these endpoints are carried out by sending http requests using Postman to validate the endpoints are correct and the connections are well-established. These tests confirms that the Controller -> Service -> Repository flow are functioning and efficient. Additionally, these controller code segments are pushed into the relevant controller package in the Git repository. Here below is the URL link for the package of interest.

Controller Package: https://github.com/CanerKaraca06/cng495-capstone-project-2025/tree/main/backend/src/main/java/com/capstoneproject/musicplatform/controller

Example snippet of User controller can be found below:

```
@CrossOrigin(origins = "*")  CanerKaraca06
@RestController          //Indicate RESTapi controller
@RequestMapping("api/users")        //Endpoint path
public class UserController {
    private final UserService userService;  3 usages
    public UserController(UserService userService) { this.userService = userService; }

    @PostMapping("/register")  CanerKaraca06
    public ResponseEntity<User> registerUser(@RequestBody User user){
        User savedUser = userService.registerUser(user);
        return ResponseEntity.ok(savedUser);
    }
```

Figure 10: Example Snippet of User Controller

4.2.5 Security Configuration Package

During the implementation of this security configuration file, Spring Security is utilized. This configuration class establishes the fundamental security settings and deals with security concerns. For the sake of simplicity, **CSRF (Cross Site Request Forgery)** is disabled during the development and testing. This disablement allows the application to receive request from external entities such as Postman without additional concerns and settings.
SecurityFilterChain is typically a **Java Bean** that is used to determine which Spring Security Filter instances should be invoked for the current request. In this phase of development, all incoming http requests are permitted using the phrase **.permitAll().** This approach ensures that all accesses are unrestricted while the application is still under construction process. Additionally, **BcryptPasswordEncoder** provides a **PasswordEncoder** that generates hashed user passwords before securely storing sensitive information into the database. This class represents the initial security concerns for phase 2, which may be extended or modified during the later phases of development. Finally, this security configuration class is organized into the Config package both in Git repository and Spring Boot application. Here below is the URL link for the package of interest.

Config Package: https://github.com/CanerKaraca06/cng495-capstone-project-2025/tree/main/backend/src/main/java/com/capstoneproject/musicplatform/config

Example snippet of Security Config can be found below:

```java
@Configuration  CanerKaraca06
@EnableWebSecurity
public class SecurityConfig {

    @Bean  CanerKaraca06
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(auth -> auth
                        .anyRequest().permitAll()
                );
        return http.build();
    }

    @Bean  CanerKaraca06
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Figure 11: Example Snippet of Security Configuration File

4.2.6. Testing the Song Entity API Endpoints

In this phase of the development, implemented entities and interconnections between them are tested and debugged using Postman. Mock data is presented to the application in the JSON format and test results are inspected. Song entity table is in @ManyToOne relation with the Artist entity table. These table columns are joined together in the database. So, the result of the GET request for /api/songs yields a nested JSON structure along with Artist details. Here below is an example of a GET request for /api/songs:

GET http://localhost:8080/api/songs

Params Authorization Headers (9) Body Scripts Settings

Body Cookies (1) Headers (11) Test Results

{} JSON Preview Visualize


```json
[
    {
        "id": 2,
        "title": "Nabiz",
        "duration": 183,
        "genre": "R&B",
        "releaseDate": "2023-10-15T14:30:00",
        "artist": {
            "id": 7,
            "stageName": "beatzByAck",
            "realName": "caner karaca",
            "bio": "He came up making beats, now the world knows him as heir to the throne for Dr.Dre",
            "profileImageUrl": "http://example.com/images/caneringo.jpg",
            "genre": "R&B",
            "country": "Türkiye",
            "debutDate": "2010-05-15",
            "followersCount": 50000,
            "averageRating": 4.9
        },
        "fileUrl": "https://cloudstorage.com/music/nabiz.mp3",
        "uploadDate": "2025-11-13T17:48:59.701163",
        "averageRating": 4.21,
        "playCount": 15036
    }
]
```


Figure 12: Example GET Request for /api/songs

4.2.7. Deliverables for Week 10 November – 16 November

This week primarily contained the implementation of User, Artist and Song entities along with their controllers, services, repositories. Here is the listed deliverables for this development week.

- Implemented entity classes (User, Artist, Song)
- Implemented repository classes for each entity
- Implemented service classes for each entity
- Implemented security configuration class and package
- Successful endpoint API unit testing
- Organized package structure in Git and Spring Boot

4.3- OdtuClass Week 17 November – 23 November

4.3.1 S3 Bucket Setup

Storage layer is configured through Amazon AWS utilizations. In this phase, a specially assigned S3 storage bucket was created to store song audio files uploaded by artists. This bucket is created with the name: **music-platform-storage**. Also, public access is not restricted to allow flexible privacy. Appropriate bucket policies were defined to limit operations strictly to authenticated requests. Naming conventions are considered to establish a modular and structured storage process. This setup is crucial for efficient media file storage and data handling.

4.3.2 IAM User Creation and Access Credentials

A dedicated IAM user is created to allow access to the S3 storage from programmable backend side. This IAM user is created with the username: **springboot-s3-user.** Provide access to the AWS Management Console button is ticked in order to access the web interface panel not only by Spring program, rather by password and username. Also, AmazonS3FullAccess policy is applied to ensure full authorization to the IAM user. Access Key ID and Secret Access Key is generated automatically. These security credentials variables are required for SDK-based authentication. Also, these ID and Key generations allow the program to interact with AWS Cloud services without exposing sensitive information.

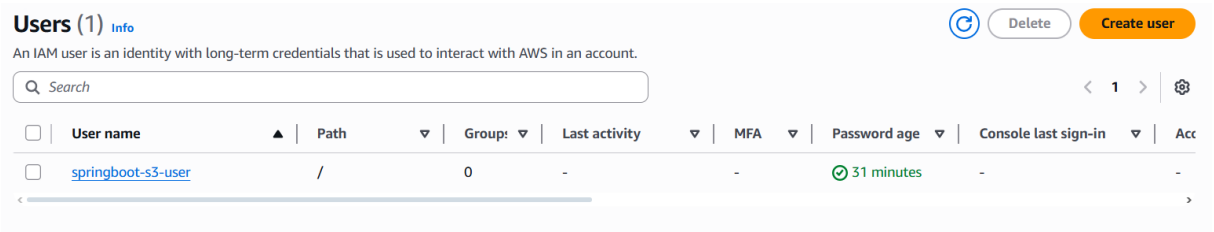

Figure 13: IAM User Panel

Console Sign-In Details:

URL: https://523103530984.signin.aws.amazon.com/console

username: springboot-s3-user / console password: pD91z|V

4.3.3 AWS SDK Dependency

To seamlessly interconnect the Spring Boot application and AWS services, **Amazon AWS SDK for Java is** included in the **pom.xml** file. **Pom.xml** file is a special file that comprises of dependency definitions that the program needs, located under the target directory. This AWS SDK dependency allows the program to perform operations such as:

-S3 Bucket operations,

-Uploading files to and deleting files from S3

-Generating S3 object URLs

Overall, this dependency is a key factor to transform the backend into a cloud-based service that is utilized to store media content

```
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>s3</artifactId>
    <version>2.25.30</version>
</dependency>
```

Figure 14: AWS SDK Dependency

4.3.4 Environment Variables

Rather than storing the sensitive AWS information directly into the plain code, an external .env file is constructed to store environment variables. This environment variables include:

-AWS ACCESS_KEY_ID

-AWS SECRET_ACCESS_KEY

-AWS S3_BUCKET_NAME

-AWS_REGION

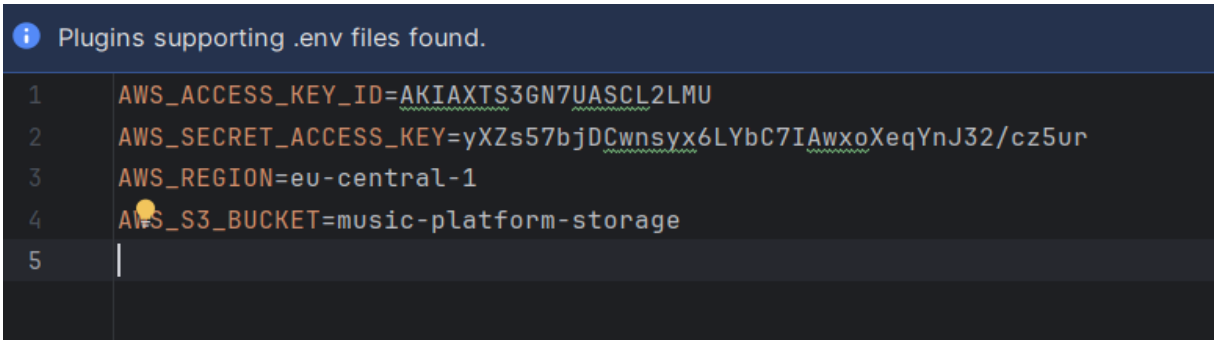

Figure 15: .env File

These variables are also added to application.properties file to ensure seamless connection.

Figure 16: application.properties AWS

Additionally, **dotenv** dependency is included in the pom.xml to automatically load and update these variables during the start of the application. This approach ensures a synchronized configuration management across different layers and environments of overall program.

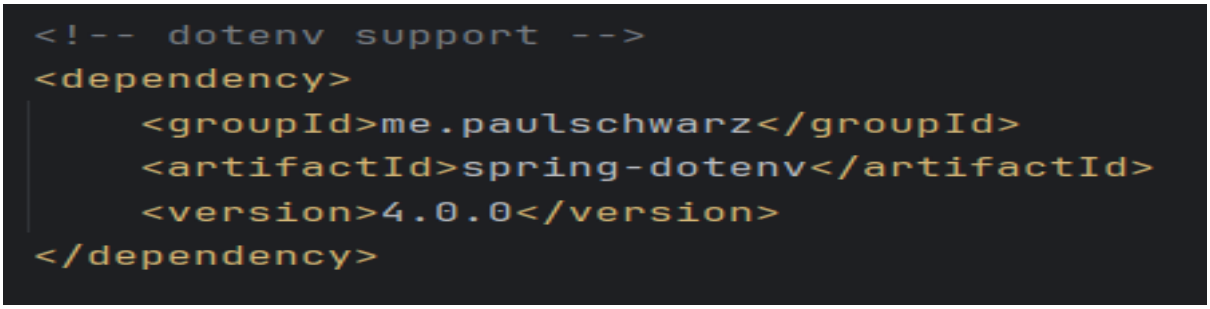

Figure 17: dotenv Dependency

4.3.5 Deliverables for Week 17 November – 23 November

This week of development mainly deals with Cloud environment initialization and necessary setup. This flow of work is crucial for later phases where these environment is going to be utilized mostly in the media storage. Here is the listed deliverables for this development week:

- Completed Cloud environment initialization
- Created S3 bucket
- Created IAM User
- Retrieved access credentials
- Added AWS SDK dependency
- Configured environment variables

5- MILESTONES REMAINING

While the project is progressing to its next stages, remaining key features are expected to be completed. These features mostly contain front-end development and handling overall project interconnection. Remaining tasks are crucial for achieving full application functionality and met final requirements. During the first phase of development, core backend features are already established with slightly adjustable and modifiable parts remaining. Finally, front-end side of the application will be implemented and tested using React to ensure easy to use user interface and efficient backend – frontend connection. This section focuses on and outlines the tasks remaining in a goal-oriented and structured approach, organized by OdtuClass weeks.

5.1 Milestones Remaining for OdtuClass Week 1 December – 7 December

This week will primarily include the implementation of core functionalities such as routing, displaying pages and basic frontend design and integration using React frameworks. Axios will be utilized to integrate the frontend into the backend. Login/Register operations are expected to be implemented with a basic UI approach. Here below are the approximately expected deliverables for week 1-7 december:

- Listed songs
- Login / Register form
- Upload button
- Notification Logic
- May be extended..

5.2 Milestones Remaining for OdtuClass Week 8 December – 14 December

This week will mostly handle the front-end features and beautification process. These features may include a rating system where users can rate songs, a top 10 chart based on these ratings, comment and feedback system for users. Below is the approximate estimation of tasks to be completed for week 8 December – 14 December.

- Song play component
- Rating system
- Top 10 chart
- Comment and feedback system
- May be extended..

5.3 Milestones Remaining for OdtuClass Week 15 December – 22 December

This weeks main objective will be implementing the artist following system and profile generation. In addition, playlist creation and access are expected to be finalized in this chapter. With these features, an integrated and mostly functional application are expected to be tested and put into use. Final tests are to be carried out to ensure that the system is functioning effectively. Below is the approximate estimation of tasks to be completed for week 15 December and 22 December.

- Follow entity
- Notification system
- Artist profile UI
- Playlist creation
- Following mechanism
- Overall working demo
- GitHub Repository

All of the implementation and development will be carried out by **Ahmet Caner Karaca.**

6. FINAL REMARKS

The progress achieved in the second phase provides a solid foundation for the backend side of the project. Core backend features included a clear layered structure organized into packages, successful PostgreSQL database connection with JDBC and AWS S3 file storage. Throughout the development process, entity classes, repositories, services and controllers are implemented for user, artist and song entities along with security configuration file. Application.properties and pom.xml files are properly instantiated according to the projects requirements.

Additionally, cloud technologies initialization included S3 bucket creation, retrieving IAM user access credentials and AWS SDK dependency attachment. These tasks are crucial for implementing a successful cloud environment early-on which will be utilized for later deliverables.

Also, various API endpoint testings are carried out after the implementation of various entities and their corresponding fields. Testing these classes provided a solid foundation that the backend is up and running. These CRUD operations testing confirms that the system behaves as expected.

An important portion of the backend is now functional, with remaining front end implementations and interconnections. Overall, this phase contains considerable progress towards a working and efficient music sharing and listening platform.

7. COMPONENT DIAGRAM

Below is the component diagram for the backend of Music Sharing and Listening Platform:

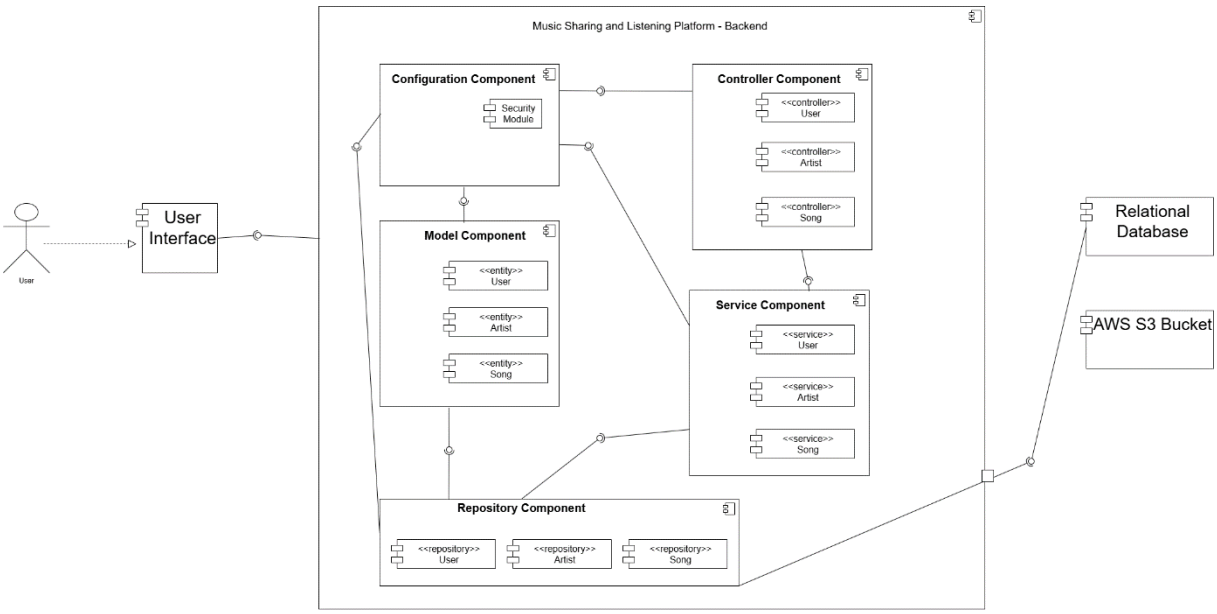

Figure 18: Component Diagram

8. GITHUB REPOSITORY LINK

Main GitHub repository can be accessed via this link:

https://github.com/CanerKaraca06/cng495-capstone-project-2025

REFERENCES

Spring. (n.d.). ***Spring Data JPA*****.**
https://spring.io/projects/spring-data-jpa

Maven Repository. (n.d.). ***spring-web*****.**
https://mvnrepository.com/artifact/org.springframework/spring-web

PostgreSQL Global Development Group. (n.d.). ***PostgreSQL JDBC Driver*****.**
https://jdbc.postgresql.org

OWASP Foundation. (n.d.). ***Cross-Site Request Forgery (CSRF)*****.**
https://owasp.org/www-community/attacks/csrf

Spring. (n.d.). ***Spring Security: Servlet architecture*****.**
https://docs.spring.io/spring-security/reference/servlet/architecture.html