**HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY OF INTERNATIONAL EDUCATION**

Logo

Description automatically generated

**FINAL TERM PROJECT**

**Course name: WEB PROGRAMMING**

**3TL MUSIC WEBSITE**

**Lecturer name:** Mai Anh Tho

**List of members:**

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Student name** | **Contribution (%)** |
| 21110796 | Ong Vu Huu Tai | 100% |
| 21110781 | Nguyen Ngoc Manh | 100% |
| 21110804 | Nguyen Thanh Tinh | 100% |

*Ho Chi Minh City, 12/2023*

# **Acknowledgment**

We sincerely thank the project's supervisor, Mrs. Mai Anh Tho, for her guidance and support throughout the project. Along with skillfully presenting the work, you also provided us with insightful advice and direction on how to finish it and helped us comprehend the complex problems associated with project management. Otherwise, these subtleties would have been lost. Only with your help could our project have succeeded.

Just enough time is allotted in ten weeks to finish the projects. However, the project will undoubtedly contain a large number of errors because there is a lot of new information and the amount of work we do each week is not ideal. We eagerly await all of the feedback from you in order to improve our meager understanding. Thank you very much.

**Table of Contents**

[**Acknowledgment** 1](#_Toc152882725)

[**Chapter 1. Project Description** 3](#_Toc152882726)

[***1.1.*** ***Objective*** 3](#_Toc152882727)

[***1.2.*** ***User benefits*** 3](#_Toc152882728)

[***1.3. Technical specifications*** 3](#_Toc152882729)

[**Chapter 2. Task assignment** 4](#_Toc152882730)

[**2.1 Work contributions and milestone:** 4](#_Toc152882731)

[**Chapter 3. Designs** 5](#_Toc152882732)

[***3.1. Process descriptions*** 5](#_Toc152882733)

[***3.2. Class implements*** 8](#_Toc152882734)

[***3.3. Database connection:*** 9](#_Toc152882735)

[***3.4.*** ***Web servlets:*** 15](#_Toc152882736)

[***3.5. Web Jakarta Server Pages:*** 19](#_Toc152882737)

[**Chapter 4. User Interface** 24](#_Toc152882738)

[**Chapter 5. Conclusion** 27](#_Toc152882739)

[**References:** 28](#_Toc152882740)

**Chapter 1. Project Description**

* 1. ***Objective***

The goal of the Music Library website is to offer a huge selection of music in a variety of genres and styles. Users are welcomed to the website by an easy-to-navigate interface that is clear and uncluttered. Visitors can get an idea of the wide variety of music that is available by perusing sections on the homepage that highlight recent releases, well-liked playlists, and suggested songs.

Users can browse the vast music library on the website by artist, album, song title, or genre thanks to the search feature that is prominently displayed. Furthermore, there are carefully selected playlists that accommodate diverse emotions, pursuits, or themes, offering users pre-made compilations appropriate for broad tastes.

* 1. ***User benefits***

Music websites, which provide a wide range of genres and artists that are accessible to everyone, have completely changed the way that people listen to music. These online platforms are simple to use and allow users to immerse themselves in a vast library of regularly updated music. These websites accommodate individual preferences by letting users make playlists that are specially tailored to their likes. The ability to personalize playlists improves the entire experience of listening to music, whether you're discovering new songs or returning to well-loved favorites. Additionally, by allowing users to share their carefully curated playlists with others who have similar musical interests, these platforms promote a sense of community.

## ***1.3. Technical specifications***

Front-End Technologies:

* HTML5, CSS, JSP for building the user interface
* Back-End Technologies:
* Java, Servlet for the server-side runtime environment
* JPA: an API for working with databases
* Database Management System:

**Chapter 2. Task assignment**

**2.1 Work contributions and milestone:**

|  |  |  |
| --- | --- | --- |
| **Tasks** | **Assignments** | **Day** |
| Choose topic | All members | 18/10 |
| Design web features and ERD | All members | 27/10-30/10 |
| Create database and connection | Tinh | 1-3/10 |
| Add backend servlets | All members | 4-19/11 |
| + Search song, artist | Manh |  |
| + Player, play music | Tai |  |
| + Upload song, create album, favourite list | Tinh |  |
| Design front-end | All members | 20-23/11 |
| Test and fix bugs | All members | 24-25/11 |
| Make report | Tai | 26-27/11 |

**Chapter 3. Designs**

***3.1. Process descriptions***

Creating a music library web application using JSP (Java Server Pages) and Servlets connected to MySQL involves several steps:

*\*Setting Up the Development Environment:*

- IDE: Eclipse (JDK 17, Tomcat 9.0).

- Database: mySQL

*\*Designing the Database:*

- MySQL Database: musicLibrary

- Tables: User, Song, Album, Favourite\_list

*\*Backend Development with Servlets:*

- Database connection and control Java classes:

+ until

+ AlbumDAO

+ FavouritelistDAO

+ UserDAO

+ SongDAO

- Web controller servlets:

+ AddToFavouriteServlet

+ AlbumServlet

+ DownloadSevlet

+ EditFavoSevlet

+ EditPassServlet

+ HomeServlet

+ LoginServlet

+ RegisterServlet

+ SearchServlet

+ ShowAllAlbumsServlet

+ ShowFavouriteListServlet

+ ShowSongsServlet

+ SongServlet

+ UpdateuserServlet

+ UserServlet

\**Frontend Development with JSP:*

- JSP Pages:

+ footer

+ header

+ info

+ search

+ searchheader

+ sidebar

+ Edit

+ favouriteList

+ list

+ Login

+ main

+ music

+ Register

+ showAllAlbums

+ showSongs

+ updateUser

\**Implementing Functionality:*

- Features for user:

+ User login

+ Music Display

+ Display catalog of available music, including search and filter options.

+ Album, favouritelist Creation and Management

+ Music Upload, Download and Playback

+ User Profile Management

*UML Diagram:*

A diagram of a song

Description automatically generated

***3.2. Class implements***

The provided code represents several entity classes in a Java application using JPA (Java Persistence API). These classes define the structure and relationships between the corresponding database tables. Here's a summary of each class:

* + User class: Represents a user in the application with attributes such as email, fullname, password, and username. It has a one-to-many relationship with Favouritelist and Album entities, indicating that a user can have multiple favorite lists and albums.
  + Song class: Represents a song with attributes such as songname, author, releaseday, albumname, type, urlimg, and urlsong. It has a many-to-one relationship with the Album entity, indicating that a song belongs to a specific album. It also has a one-to-many relationship with the Favouritelist entity, indicating that a song can be present in multiple favorite lists.
  + Album class: Represents an album with attributes such as albumname and singername. It has a one-to-many relationship with the Song entity, indicating that an album can have multiple songs.
  + Favouritelist class: Represents a favorite list with attributes such as id. It has a many-to-one relationship with the Song entity, indicating that a favorite list can contain multiple songs. It also has a many-to-one relationship with the User entity, indicating that a favorite list belongs to a specific user.

These entities and their relationships define the structure of the database tables and allow for seamless interaction with the underlying database using JPA. The provided code also includes named queries for each entity, which can be used to retrieve specific data from the corresponding tables.

Here is a example for Entity in our project:

*User class:*

*@Entity*

*@NamedQuery*(name = "User.findAll", query = "SELECT u FROM User u")

public class User implements Serializable {

private static final long ***serialVersionUID*** = 1L;

private String email;

private String fullname;

private String password;

*@Id*

private String username;

*@OneToMany*

*@JoinColumn*(name = "username",insertable = false, updatable = false)

private List<Favouritelist> favouriteLists;

*@OneToMany*

*@JoinColumn*(name = "username",insertable = false, updatable = false)

private List<Album> albums

@Entity and @NamedQuery at the top of the class, indicating that this class is an entity in the data model and has a named query (NamedQuery)

@Id marks the primary key field.

@GeneratedValue configures how the primary key values are generated.

@ManyToOne establishes a many-to-one relationship between Playlist and User entities, where many playlists can belong to one user.

@ManyToMany establishes a many-to-many relationship between Playlist and Music entities, allowing multiple tracks to belong to multiple playlists.

These entities define the structure and relationships between the User, Song, Album, and Favouritelist tables in a JPA-based application, enabling seamless interaction with the underlying database.

-> The most difficult part when doing this part is mapping the entities together

***3.3. Database connection:***

The data layer is responsible for managing data persistence and interacting with the underlying data sources such as databases or external APIs. It handles tasks related to data storage, retrieval, and manipulation. In a Java servlet application, the data layer often involves the use of technologies like JDBC (Java Database Connectivity) or JPA (Java Persistence API) to interact with databases. It encapsulates the logic for database operations, such as querying, inserting, updating, and deleting data. The data layer provides an abstraction for accessing and manipulating data, which is utilized by the business layer.

The database connection, entity classes, DDL generation settings, and logging preferences are just a few of the crucial details that are established in this configuration for the persistence unit. It provides guidance to the JPA provider on how to maintain the database schema and communicate with the designated web entities.

*\*Persistent file:*

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**persistence** version=*"2.2"* xmlns=*"http://xmlns.jcp.org/xml/ns/persistence"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_2.xsd"*>

<**persistence-unit** name=*"Music\_webstite"* transaction-type=*"RESOURCE\_LOCAL"*>

<**class**>music.business.Album</**class**>

<**class**>music.business.Favouritelist</**class**>

<**class**>music.business.Song</**class**>

<**class**>music.business.User</**class**>

<**properties**>

<**property** name=*"javax.persistence.jdbc.url"* value=*"jdbc:mysql://localhost:3306/finalweb"*/>

<**property** name=*"javax.persistence.jdbc.user"* value=*"root"*/>

<**property** name=*"javax.persistence.jdbc.driver"* value=*"com.mysql.jdbc.Driver"*/>

<**property** name=*"javax.persistence.jdbc.password"* value=*"123456"*/>

</**properties**>

</**persistence-unit**>

</**persistence**>

*Until.java:*

package music.until;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

public class until {

public static EntityManager getEntityManager() {

EntityManagerFactory emf = Persistence.*createEntityManagerFactory*("Music\_webstite");

return emf.createEntityManager();

}

}

The EntityManager is part of the Java Persistence API (JPA) and is used for managing entities and their persistence in a relational database. In this case, the code is creating an EntityManagerFactory using the Persistence.createEntityManagerFactory() method, passing the argument "Music\_webstite" as the persistence unit name. The persistence unit name corresponds to the configuration specified in the persistence.xml file.

Once the EntityManagerFactory is created, the code calls the createEntityManager() method on it to obtain an instance of EntityManager, which is then returned by the getEntityManager() method.

This code is typically used in Java applications that utilize JPA for database access. It provides a convenient way to obtain an EntityManager instance for performing database operations such as querying, inserting, updating, and deleting entities.

FavouritelistDAO:

* Manages data related to the user's favorite song list.
* Provides methods to add songs to the list, retrieve the list for a specific user, and remove songs from the list.

SongDAO:

* Handles data operations related to songs.
* Allows adding new songs to the database, searching for songs based on a search term, retrieving songs from a specific album, and retrieving all songs from the database.
* Provides a method to retrieve the name of a song based on its URL.

UserDAO:

* Manages user-related data and operations.
* Includes methods to check login credentials, register new users, retrieve user information based on the username, and update user information.

AlbumDAO:

* Deals with operations related to albums in the database.
* Provides methods to add a new album, retrieve an album by name, retrieve all albums, and retrieve songs associated with a specific album.

Each DAO class encapsulates the logic for interacting with the database and performs CRUD (Create, Read, Update, Delete) operations on the corresponding entities. They utilize the Java Persistence API (JPA) and the EntityManager to interact with the underlying database, perform queries, and persist data.

When implementing JPA (Java Persistence API), there can be several difficulties:

1. Configuration and database connection: Configuring JPA and establishing a database connection can be an initial challenge. It is important to ensure that the database connection information, such as URL, username, password, and driver, is accurately provided in the configuration file orWhen implementing JPA (Java Persistence API), there can be several difficulties. Here is a summary of these difficulties:
2. Configuration and database connection: Configuring JPA and establishing a database connection can be an initial challenge. It is important to ensure that the database connection information, such as URL, username, password, and driver, is accurately provided in the configuration file or through annotations.
3. Session management and query execution: JPA uses the EntityManager object to manage sessions and execute queries. Managing session lifespan, synchronizing changes, and handling errors can be challenging. It is important to use the EntityManager correctly and close sessions after completing query operations.
4. Performance optimization: Performance can be a challenge when using JPA. Slow data loading or generating too many queries can impact performance. Optimizing strategies such as lazy loading and fine-tuning queries to fetch only necessary information is necessary.
5. Handling object relationships: When working with complex object relationships such as one-to-many, many-to-many, or inheritance, managing these relationships can be challenging. Ensuring proper mapping in entity classes and using annotations or JPA configuration to define these relationships is important.
6. Transaction handling: In a multi-threaded or multi-session environment, transaction handling can be complex. Ensuring proper identification and management of transactions in database operations is crucial.
7. Incompatibility between JPA and existing databases: In some cases, there may be incompatibility between JPA and existing databases. Using native SQL queries or other JPA features such as NamedQuery or Criteria API can help address this issue.
8. Debugging and error handling: When using JPA, errors such as query syntax errors, entity not found, or runtime errors can occur. Using debugging tools, logging, and error analysis to identify the causes and handle errors accurately is important.

Here is a example for DAO in our project:

***AlbumDAO:***

package music.data;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.EntityTransaction;

import javax.persistence.Persistence;

import javax.persistence.TypedQuery;

import music.business.Album;

import music.business.Song;

public class AlbumDAO {

private final EntityManagerFactory emf;

public AlbumDAO() {

emf = Persistence.*createEntityManagerFactory*("Music\_webstite"); // Thay tháº¿ báº±ng tÃªn persistence unit cá»§a báº¡n

}

public void closeEntityManagerFactory() {

if (emf != null && emf.isOpen()) {

emf.close();

}

}

public void addAlbum(Album album) {

EntityManager em = emf.createEntityManager();

EntityTransaction transaction = null;

try {

transaction = em.getTransaction();

transaction.begin();

em.persist(album);

transaction.commit();

} catch (Exception e) {

if (transaction != null && transaction.isActive()) {

transaction.rollback();

}

e.printStackTrace();

} finally {

em.close();

}

}

public Album getAlbumByName(String albumName) {

EntityManager em = emf.createEntityManager();

try {

return em.find(Album.class, albumName);

} finally {

em.close();

}

}

public List<Album> getAllAlbums() {

EntityManager em = emf.createEntityManager();

try {

TypedQuery<Album> query = em.createQuery("SELECT a FROM Album a", Album.class);

return query.getResultList();

} finally {

em.close();

}

}

public List<Song> getSongsByAlbumName(String albumName) {

EntityManager em = emf.createEntityManager();

try {

// Using a fetch join to load songs along with the album

TypedQuery<Album> query = em.createQuery("SELECT a FROM Album a LEFT JOIN FETCH a.songs WHERE a.albumname = :albumname", Album.class);

query.setParameter("albumname", albumName);

Album album = query.getSingleResult();

return album.getSong();

} catch (Exception e) {

e.printStackTrace();

return null;

} finally {

em.close();

}

}

}

return query.getResultList();

} finally {

em.close();

}

}

public List<Song> getAllSongs() {

EntityManager em = emf.createEntityManager();

try {

TypedQuery<Song> query = em.createQuery("SELECT s FROM Song s", Song.class);

return query.getResultList();

} finally {

em.close();

}

}

public String getSongNameByUrl(String url) {

EntityManager em = emf.createEntityManager();

if (url.startsWith(".")) {

url = url.substring(1);

}

try {

TypedQuery<String> query = em.createQuery("SELECT s.songname FROM Song s WHERE s.urlsong = :url",

String.class);

query.setParameter("url", url);

return query.getSingleResult();

} finally {

em.close();

}

}

}

***3.4. Web servlets:***

Controllers play a crucial role in web application development, particularly in the context of the Model-View-Controller (MVC) architectural pattern. They act as intermediaries between the user interface (View) and the data and business logic (Model).

The primary responsibility of a controller is to handle incoming requests from clients, perform necessary processing or data retrieval, and generate appropriate responses. Controllers receive requests from the View or client, interpret the data, interact with the Model to retrieve or manipulate data, and finally return a response back to the client.

UploadServlet:

* Handles file uploads for audio and image files.
* Retrieves the uploaded files and saves them to the server.
* Extracts information from the request parameters, such as the song title and artist.
* Performs validation on the uploaded files and parameters.
* Stores the file information and metadata in a database.
* Redirects the user to a specified page after the upload process.

UpdateUserServlet:

* Handles user updates based on submitted form data.
* Retrieves updated user information, such as a new password, full name, and email, from the request parameters.
* Performs validation on the updated information.
* Updates the user's information in a database using a UserDAO.
* Redirects the user to a specified page after the update process.

UserServlet:

* Retrieves user information based on a username parameter.
* Retrieves the username from the request parameters.
* Fetches the user's information from a database using a UserDAO.
* Sets the user object as an attribute in the request.
* Forwards the request to a specified JSP page to display the user's information.

AddToFavouriteServlet:

* Handles both GET and POST requests.
* In the doPost() method, it adds a song to a user's favorite list.
* Retrieves the current user's username from the session.
* Retrieves the song name based on the request parameters.
* Adds the song to the user's favorite list using a FavouritelistDAO.
* Redirects the user to the HomeServlet page.
* In the doGet() method, it retrieves the user's favorite song list and forwards the request to the favouriteList.jsp page.

AlbumServlet:

* Handles both GET and POST requests.
* In the doPost() method, it adds a new album to the database.
* Retrieves the album name and singer name from the request parameters.
* Creates an Album object with the provided information.
* Adds the new album using an AlbumDAO.
* Redirects the user to the HomeServlet page.
* In the doGet() method, it retrieves all albums from the database and forwards the request to the showAllAlbums.jsp page.

DownloadSevlet:

* Handles GET requests for downloading songs.
* Retrieves the song URL from the request parameters.
* Retrieves the song name based on the URL using a SongDAO.
* Sets appropriate response headers for an audio file.
* Initiates the file download by reading the file from the input stream and writing it to the output stream.
* Handles any errors that occur during the file download.

EditFavoSevlet:

* Handles GET requests.
* In the doGet() method, it retrieves the current user's username from the session.
* If the username exists, it retrieves the "songname" parameter from the request.
* Creates a FavouritelistDAO object.
* Removes the specified song from the user's favorite list.
* If the removal is successful, it forwards the request to the "/showFavouriteList" servlet.
* If the removal fails, it forwards the request to the "/views/list.jsp" page.

EditPassServlet:

* Handles GET requests to update user information.
* In the doGet() method, it retrieves user data from the request parameters, such as the username, new password, full name, and email.
* Creates a UserDAO object for database operations.
* Creates a User object with the updated information.
* Calls the updateUser() method to update the user in the database.
* If the update is successful, it forwards the request to the "/LoginServlet" page to handle the login process.
* If the update fails, it forwards the request to the "/views/Edit.jsp" page.

HomeServlet:

* Handles both GET and POST requests.
* In the doGet() method, it retrieves the current user from the session.
* Forwards the request to the "/views/main.jsp" page.
* Sets the user object as an attribute in the request for further use in the JSP page.
* In the doPost() method, it retrieves a list of songs from the SongDAO class and sets the list as an attribute in the request.
* Retrieves the username from the request parameters, saves it to the session, and redirects the user to the "/HomeServlet" page.

LoginServlet:

* Handles both GET and POST requests.
* In the doGet() method, it sets the "MESSAGE" attribute in the request as an empty string and forwards the request to the "/views/Login.jsp" page.
* Retrieves cookies from the request and sets "user" and "pass" attributes in the request based on the cookie values.
* In the doPost() method, it retrieves the username and password from the request parameters.
* Checks the username and password using the CheckLogin() method from the UserDAO class.
* If the login is successful, it saves the username to the session and creates cookies for the username and password.
* Redirects the user to the "/HomeServlet" page.
* If the login fails, it sets an error message in the "MESSAGE" attribute of the request and forwards the request back to the "/views/Login.jsp" page.

When working with controllers in a Java web application, there can be several difficultiess:

1. Handling exceptions and errors: Dealing with exceptions and errors that occur during request processing can be difficult. Implementing proper exception handling mechanisms and returning appropriate error responses is important for a good user experience.
2. Managing dependencies: Controllers often depend on other components or services. Managing dependencies and ensuring proper dependency injection can be complex. Using dependency injection frameworks or design patterns like Inversion of Control (IoC) can help address this challenge.
3. Performance optimization: Optimizing the performance of controllers is crucial for a responsive and efficient application. Caching, minimizing unnecessary computations, and optimizing database queries are some techniques that can be used to improve performance.
4. Versioning and API evolution: As an application evolves, managing backward compatibility and versioning of APIs can become challenging. Implementing proper versioning strategies and handling API changes gracefully is important to avoid breaking existing client applications.

***3.5. Web Jakarta Server Pages:***

***Login.jsp:***

***A screenshot of a computer

Description automatically generated***

***Main.jsp*** *include footer.jsp, header.jsp, info,jsp, search,jsp, sidebar,jsp, searchheader.jsp*

***A screenshot of a computer

Description automatically generated***

***search.jsp***

***A screenshot of a video

Description automatically generated***

**List.jsp**

**A screenshot of a computer

Description automatically generated**

***showAllAlbums.jsp***

***A screenshot of a computer

Description automatically generated***

***Music.jsp***

***A screenshot of a computer

Description automatically generated***

***updateUser.jsp*** ***A screenshot of a computer

Description automatically generated***

***showSongs.jsp***

***A screenshot of a video

Description automatically generated***

***Edit.jsp***

***A screenshot of a computer

Description automatically generated***

***Register.jsp***

***A screenshot of a computer

Description automatically generated***

**Chapter 4. User Interface**

We do not use user permissions, so the website need to register an account to use

*A screenshot of a computer

Description automatically generated*

Easy to register

*A screenshot of a login screen

Description automatically generated*

And now, we can log in with new account

*A screenshot of a video chat

Description automatically generated*

In main page, they will show you songs to play

You can click into song to play, or you can search songs which you want to play

*A heart and arrow pointing down

Description automatically generated*

While listening to music you can like or download the song, and if you move to another page the song is still played

*A screenshot of a computer

Description automatically generated*

You can click into box information of song to play it in favouritelist, and you can click “Remove” to remove it from the favourite lite

A screenshot of a dog

Description automatically generated

You can click to the button to show all songs in album

A screenshot of a video player

Description automatically generated

And you can click to the box information of songs to play

**Chapter 5. Conclusion**

The evaluation of the website indicates that most requirements were met, and the code is clean and reusable. The website follows a simple GUI design for ease of use. However, there are some identified difficulties, such as the learning curve of new technologies and challenges with file storage.

The advantages of the website include its clean code, meeting project requirements, and a user-friendly GUI. The reusability and maintainability of the code are also noted. On the other hand, there are some disadvantages, including the lack of web security measures, the limitation of listening to music on the same page, and the need for GUI optimization.

To further develop the website, suggested ideas include implementing web security measures, improving the music listening experience by allowing continuous playback, enhancing user interaction through features like likes and comments, and introducing new features such as recommendations and trending songs. The GUI can be optimized for a better user experience, and additional features like playlists and artist profiles can be added. Considerations should also be given to scalability and performance to handle multiple users concurrently.

# **References:**

[1] Lecturer Mai Anh Tho: Web Programming course at Ho Chi Minh University of Technology and Education (2023)

[2] Murachs Java Servlets and JSP 3rd Edition (2014) Editing

[3] Servlet API Documentation, [Overview (Servlet API Documentation) (apache.org)](https://tomcat.apache.org/tomcat-5.5-doc/servletapi/index.html)

[4] <https://laptrinhjavaweb.com/khoa-hoc-java-web-co-ban-jsp-servlet-jdbc-va-mysql>