INFO6250 WEB DEVELOPMENT TOOLS & METHODS
SEC 06
Final Project Report

Web Music Library

Summary Of the Project:
This is an Online Music Player Library web application. In this application a user can register and create a library of the kind of music they like. The user can select songs from a large pool of songs database. The user can add songs to his playlist and save it for later.

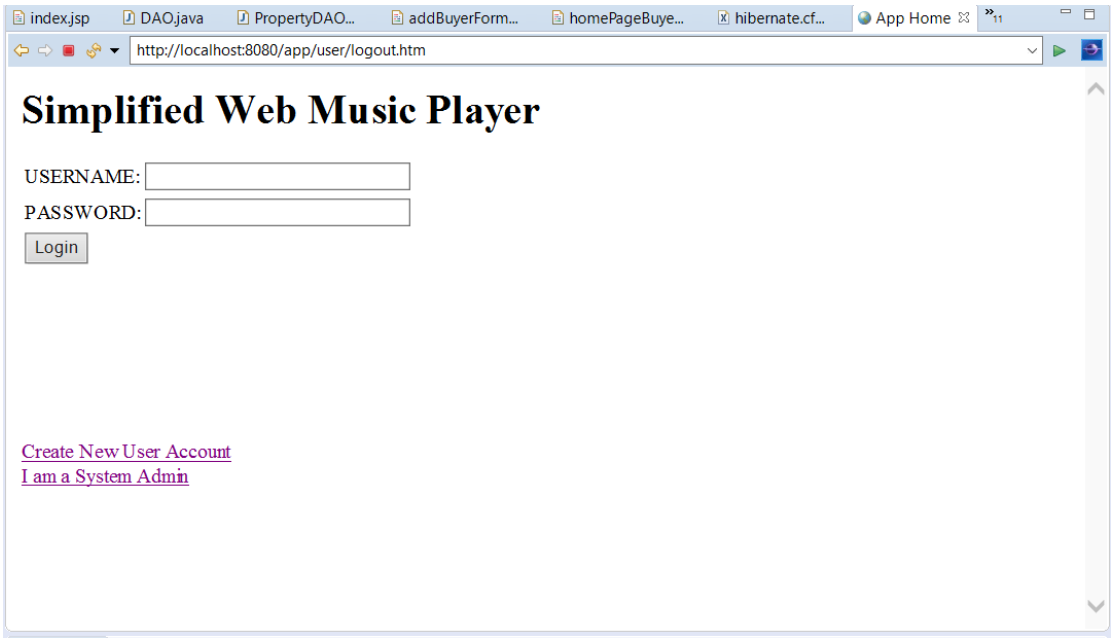
Summary of the Technologies Used:
1. Spring MVC- The Spring MVC framework provides the Model-View- Controller architecture. The Model consists of the Pojo classes, the View generates the HTML output for the users to interpret and input values and the Controllers are responsible for processing the request and build an appropriate model and passes it to the View.
2. Hibernate- Hibernate is responsible for mapping the Java classes (Pojo's) to the database. Mapping is done as what Java class object to store in which table in the database.
3. MySQLWorbench- It is an open source database management system, which allows us to create tables, update table, drop, delete values from it etc. These tables are mapped automatically using Hibernate in the Web application.

Types of Users:
- **User**
- **Admin**

Tasks by User:
1. User Registration: A new user can register to the music web application by creating a new username and password.
2. User Login: A user can now log into the music web application by using the username and password created by him.
3. Create new Playlist: A user can create as many playlist as they want.
4. Add a song: A user can add n number of songs in the playlist they created.
5. View Song: A user can view their playlist and song inside it.

Admin Login:
I added j_security_check which is the role-based security restricting resources at the user level. I created the roles and restrict the roles. Defined different roles in the tomcat-users-xml file. The admin can then login using the Tomcat file username and password.
Tasks by Admin:
1. **Artists Create:** An admin can create and add artist to the artist table in the table of the database.
2. **Add Songs:** An admin can add songs and the URL to the artist that have been created by the user before.

Screenshots:



User Logout Home Page My Playlist

User Dashboard

Welcome, mansi !

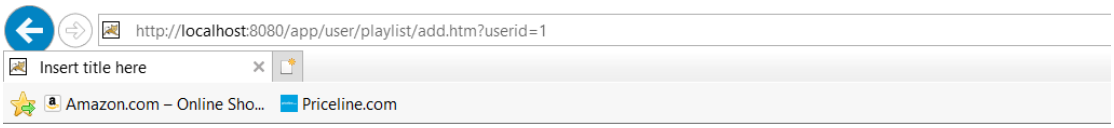
Search for Music:

Search

User Logout Home Page My Playlist

Search Result

1 Indian Summers Go To Song Add to Playlist



User Logout Home Page My Playlist

Add New Playlist:

Songs | Create Playlist

User Logout Home Page My Playlist

Song Added Successfully!

Pojo Classes:

1. **artist.java**

```
@Entity
public class artist {

@Id
@GeneratedValue (strategy=GenerationType.IDENTITY)
@Column(name="artistId", unique=true)
public int artistId ;

@Column(name="artistName")
private string artistname;


@OneToMany(mappedBy="artist", cascade = CascadeType.ALL)
private List<Song> songlist;



public Artist(){

}

public int getartistId() {
        return artistId;
}

public void setartistId (int artistId) {
        this.artistId = artistIsd;
}

public String getartistName() {
        return artistname;
}

public void setartistName(String artistName) {
        this.artistName = artistName;
}

public List<Music> getMusiclist() {
        return musiclist;
}

public void setMusiclist(List<Music> Musiclist) {
        this.musiclist = musiclist;
}
}
```

2. **user.java**

```
@Entity
public class user {

 @Id
 @GeneratedValue (strategy = GenerationType.IDENTITY)
 @Column(name="userId", unique = true)
 private int userId;

 @Column (name="userName")
 private String userName;

 @Column(name="password")
 private String password;



 @OneToMany(mappedBy="user", cascade = CascadeType.ALL,
orphanremoval=true)
 private List<list> playlistadded;

 public user(){

 }

 public int getuserId() {
        return userId;
 }



 public void setuserId(int userId) {
        this.userId = userId;
 }



 public String getuserName() {
        return userName;
 }

 public void setuserName(String userName) {
        this.userName = userName;
 }

 public String getpassword() {
        return password;
 }
```

```
public void setpassword(String password) {
        this.password = password;
}

public List<list> getplaylistAdded() {
        return playlistAdded;
}

public void setPlaylistAdded(List<list> playlistAdded) {
        this.playlistAdded = playlistAdded;
}
```

3. playlist.java

```
@Entity
public class playlist {
 @Id
 @GeneratedValue(strategy=GenerationType.IDENTITY)
 @Column(name="playlistId", unique = true, nullable = false)
 private int playlistId;

 @Column(name="playlistName")
 private String playlistName;


 @OneToMany()
 @JoinColumn(name = "playlistId")
 List<Music> Musiclist;



 public Playlist(){

 }



 public String getplaylistName() {
        return playlistName;
 }



 public void setplaylistName(String playlistName) {
        this.playlistName = playlistName;
 }
```

```
public int getplaylistId() {
        return playlistId;
}



public void setplaylistiId(int playlistId) {
        this.playlistId = playlistId;
}



public User getuser() {
        return User;
}



public void setuser(user User) {
        this.user = user;
}
```

4. **music.java**

```
@Entity
public class Music {

 @Id
 @GeneratedValue(strategy=GenerationType.IDENTITY)
 @Column (name="musicId", unique = true, nullable = false)
 private int musicId;

 @Column (name="musicName")
 private String musicName;

 @ManyToOne
 private artist Artist;

 public Music(){

 }
 public artist getartist() {
        return artist;
 }
```

```
    public void setArtist(Artist artist) {
        this.artist = artist;
    }



    public String getmusicName() {
        return musicname;
    }

    public void setmusicName(String musicName) {
        this.musicname = musicname;
    }

    public int getmusicId() {
        return musicId;
    }

    public void setmusicId(int musicId) {
        this.musicid = musicid;
    }
```

Dao:

1. musicdao.java

```
public class musicDao extends Dao{
    public List<Artists> showArtists(){
        try{
            begin();
            Query query = getSession().createQuery("from Artists");
            List<Artists> result = query.list();
            Commit();
            Return result;
        }catch(HibernateException e){
            rollback();
            System.out.print("cant find all artists" + e.getMessage());
            Return null;
    }
}
    public List<Artists> getArtistsByID(int id){
        try{
            begin();
            Query query = getSession().createQuery("from Artists
where artistsid="+id);
            List<Artists> result = query.list();
            Commit();
            Return result;
```

```
            }catch(HibernateException e){
                rollback();
                System.out.print("cant find all artists" + e.getMessage());
                Return null;
    }
}
    public List<Artists> deleteArtists(){
            try{
                begin();
                Query query = getSession().createQuery("delete Artists");
                List<Artists> result = query.list();
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant find all artists" + e.getMessage());
                Return null;
    }
}
    public List<Artists> addNewArtists(Artists artists){
            try{
                begin();
                getSession().save(artists);
                List<Artists> result = query.list();
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant add artists" + e.getMessage());
                Return null;
    }
}
    public List<Songs> showSongs(){
            try{
                begin();
                Query query = getSession().createQuery("from Songs");
                List<Songs> result = query.list();
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant find all artists" + e.getMessage());
                Return null;
    }
}

    public List<Songs> getSongsByID(int id){
            try{
                begin();
```

```
                Query query = getSession().createQuery("from Songs
    where songsid="+id);
                List<Songs> result = query.list();
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant find all songs" + e.getMessage());
                Return null;
        }
    }

        public List<Artists> deleteSongss(){
            try{
                begin();
                Query query = getSession().createQuery("delete Songs");
                List<Songs> result = query.list();
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant find all songs" + e.getMessage());
                Return null;
        }
    }
```

3. playlistdao.java

```
public class playlistDao extends Dao{
        public List<Playlists> displayAllPlaylists(){
            try{
                begin();
                Query query = getSession().createQuery("from Playlists");
                List<Playlists> result = query.list();
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant find all playlists" + e.getMessage());
                Return null;
        }
    }
    public List<Artists> getPlaylistsByID(int id){
            try{
                begin();
                Query query = getSession().createQuery("from Playlists
    where playlistsid="+id);
                List<Playlists> result = query.list();
```

```
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant find all artists" + e.getMessage());
                Return null;
        }
    }
        public List<Artists> deletePlaylists(){
            try{
                begin();
                Query query = getSession().createQuery("delete Playlists");
                List<Playlists> result = query.list();
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant find all playlists" + e.getMessage());
                Return null;
        }
    }
        public List<Artists> addNewPlaylists(Playlists playlists){
            try{
                begin();
                getSession().save(playlists);
                List<Playlists> result = query.list();
                Commit();
                Return result;
            }catch(HibernateException e){
                rollback();
                System.out.print("cant add Playlists" + e.getMessage());
                Return null;
        }
    }
```

4. userdao.java

```
public class UserDao extends Dao{
   public Users get(String user_name, String password){
      try{
        begin();
        Query query = getSession().createQuery("from Users where username =:
user and password =: password");
        query.setString("user",username);
        query.setString("password",password);
        Users users (User) query.getSingleResult();
        commit();
        return user;
```

```
      }catch(HibernateException e){
         rollback();
         return null;
      }
   }

   public User findUserById(int id){
      try{
         begin();
         Query query = getSession().createQuery("from Users where userid =" + id
);
         Users users (User) query.getSingleResult();
         commit();
         return users;
      }catch(HibernateException e){
         rollback();
         return null;
      }
   }

   public User addUser(String username, String password){
      try{
         begin();
         User users = new User();
         users.setUserName(username);
         users.setPassword(password);
         users.setPlaylistAdd(new ArrayList<Playlists>());
         Query query = getSession().createQuery("from Users where username =:
user");
         query.setString("username",username);
         List list =query.list();
         if(list.size()>0){
            return null;
         }
         getSession.save(users);
         commit();
         return users;
      }catch(HibernateException e){
         rollback();
         return null;
      }
   }
   public void deleteUser(User users){
      try{
         begin();
         getSession.delete(users);
         commit();
      }catch(HibernateException e){
```

```
            rollback();
            return null;
        }
    }
    public List<Users> showAllUsers(){
        try{
            begin();
            uery query = getSession().createQuery("from Users");
            List<Users> results = query.list();
            commit();
            return results;
        }catch(HibernateException e){
            rollback();
            return null;
        }
    }
}
```

4. dao:

```
public class Dao {

     private static final ThreadLocal<Session> sessThr = new
ThreadLoca<Session>();
    public SessionFactory sesFac = new
Configuration().configure("hibernate.cfg.xml").buildSessionFactory();

    protected Dao() {
    }

    public static Session getSession(){
        Session sess = (Session) Dao.sessThr.get();

        if (sess == null){
            session = sesFac.openSession();
            Dao.sessThr;
        }
        return session;
    }

    protected void begin() {
        getSession().beginTransaction();
    }

    protected void commit() {
        getSess().getTransfer().commit();
    }

    protected void rollback() {
```

```
        try {
            getSession().getTransaction().rollback();
        } catch (HibernateException e) {
            log.log(Level.WARNING, "Cannot rollback", e);
        }
        try {
            getSession().close();
        } catch (HibernateException e) {
            log.log(Level.WARNING, "Cannot close", e);
        }
        DAO.sessionThread.set(null);
    }

    public static void close() {
        getSession().close();
        DAO.sessionThread.set(null);
    }
}
```