Project Phase - 2

Team-5

- Arth Raj (2019101094)
- Ashish Gupta (2019101061)
- Isha Gupta (2019101111)
- Soma Datta Reddy (2019111008)
- Vaibhav Kashera (2019111003)

Feature 1: Better user management

For Better User Management, we have added a new **Signup feature** such that a new user can sign up to the Music app from the login page, instead of having to login through the admin account and change password, which had been the signup process.

To implement the signup functionality, we created a JAX-RS resource method in UserResource.java file that handles the HTTP POST request to create a new user account.

```
@POST
    @Path("signup")
    public Response signup(
        @FormParam("username") String username,
        @FormParam("password") String password,
        @FormParam("locale") String localeId,
        @FormParam("email") String email
    ) {
        // Validate the input data
        username = Validation.length(username, "username", 3, 50);
        Validation.alphanumeric(username, "username");
        password = Validation.length(password, "password", 8, 50);
        email = Validation.length(email, "email", 3, 50);
        Validation.email(email, "email");

        // Create the user
        User user = new User();
        user.setRoleId(Constants.DEFAULT_USER_ROLE);
        user.setUsername(username);
        user.setPassword(password);
        user.setEmail(email);
        user.setCreateDate(new Date());

        if (localeId == null) {
            // Set the locale from the HTTP headers
            localeId = LocaleUtil.getLocaleIdFromAcceptLanguage(request.getHeader("Accept-Language"));
        }
        user.setLocaleId(localeId);

        // Create the user
        UserDao userDao = new UserDao();
        String userId = null;
        try {
            userId = userDao.create(user);
        } catch (Exception e) {
            if ("AlreadyExistingUsername".equals(e.getMessage())) {
                throw new ServerException("AlreadyExistingUsername", "Login already used", e);
            } else {
                throw new ServerException("UnknownError", "Unknown Server Error", e);
            }
        }

        // Create the default playlist for this user
        Playlist playlist = new Playlist();
        playlist.setUserId(userId);
        Playlist.createPlaylist(playlist);

        // Raise a user creation event
        UserCreatedEvent userCreatedEvent = new UserCreatedEvent();
        userCreatedEvent.setUser(user);
        AppContext.getInstance().getAsyncEventBus().post(userCreatedEvent);
```

```
        return okJson();
    }
```

Implementation details:

1. The @POST and @Path annotations are used to define the HTTP method and path for this resource method. In this case, the resource path is "signup" relative to the base URL of the application.
2. The method signature includes several @FormParam annotations, which are used to extract form data submitted in the HTTP POST request. The form data includes the username, password, locale, and email fields.
3. The input data is then validated using a Validation utility class. The Validation class checks the length of the username, password, and email fields, and also validates that the username is alphanumeric and the email is in a valid format.
4. A new User object is created with the input data, and the user's role ID is set to the default user role constant defined in the Constants class. The user's locale ID is also set based on the Accept-Language header of the HTTP request.
5. A new UserDao object is created, and the user is persisted to the database using the create() method of the UserDao class. If the username already exists, a ServerException is thrown with the message "AlreadyExistingUsername". If any other error occurs during user creation, a ServerException is thrown with the message "UnknownError".
6. A new Playlist object is created and associated with the new user, and a UserCreatedEvent is raised with the new user's information.
7. Finally, the response is returned with a 200 OK status code and an empty JSON response body.

There are several **design patterns** that have been used while creating the signup functionality.

1. Model-View-Controller (MVC) pattern: The code separates the user interface (view) from the data (model) and the logic that controls the data (controller). The JAX-RS resource method acts as a controller that handles the HTTP POST request, the User class acts as a model that represents a user, and the response returned to the client acts as a view.
2. Factory pattern: The code uses a factory method to create a new UserDao object.
3. Singleton pattern: The code uses a Singleton pattern for the AppContext class, which provides a single instance of the AsyncEventBus object that is used to raise the UserCreatedEvent.
4. Data Access Object (DAO) pattern: The UserDao class follows the DAO pattern, which separates the persistence logic from the business logic and provides a simple interface to interact with the underlying database.

Feature 2: Better Library Management

To implement the desired feature, we modified the Playlist class by introducing an additional parameter, isPublic, which accepts a Boolean value. This modification necessitated altering the table structure in the SQL files. Finally, adjustments were made in the frontend to incorporate the ability to toggle between making playlists public and private

---

Feature 3: Online Integration

In online integration through last.fm and spotify **Strategy Pattern** has been used.

Strategy Pattern:
By utilizing the strategy design pattern, you can establish a group of algorithms and assign each of them to a distinct class, enabling their objects to be substituted. For the purpose of integrating lastfm and spotify features into the music application, the related classes for each platform were isolated into separate files.

Our task for this feature involved integrating Spotify and Last.fm's search and recommendation functionalities. Initially, we reviewed their API documentation to identify the specific APIs required for the task. The APIs that we utilized are listed below:

1. Last.fm

a. Track.search for search functionality

b. Track.getSimilar for recommendation functionality

2. Spotify

a. Search: Search for Item

b. Recommendation: Get Recommendations

To utilize all these four APIs, an OAuth or API Key is necessary. We utilized the API key provided in the repository for Last.fm, while for Spotify, the user was required to provide the Bearer Token. We accessed these APIs via Angular's `$http` service in `Search.js` and `Playlist.js` files, and made the necessary changes to search.html and playlist.html to exhibit the buttons and responses effectively on the frontend.

The implementation code for search APIs is given below:

1. Last.fm

```
$http.get('https://ws.audioscrobbler.com/2.0/', {
      params: {
        method: 'track.search',
        api_key: '7119a7b5c4455bbe8196934e22358a27',
        format: 'json',
        limit: 10,
        track: query
      }
    }).then(function (response) {
      var lastFmTracks = response.data.results.trackmatches.track
      $scope.results.lastFm = lastFmTracks;
    });
```

2. Spotify

```
$http.get('https://api.spotify.com/v1/search', {
      params: {
        q: query,
        type: 'track',
        limit: 10
      },
    headers: {
      'Authorization': 'Bearer ' + 'BQBYPQmJKh5hne90YROkRoDY-bZ1ANM2f-LXL7OrLwt0oBnJtNZ6fnKYrxCOubt07kO0vgcyxfy6L4vMlaDmzLH1PYmWb52Gek
    }
    }).then(function (response) {
      var spotifyTracks = response.data.tracks.items;
      if ($scope.results) {
        $scope.results.spotify = spotifyTracks;
      } else {
        $scope.results = {
          spotify: spotifyTracks
        };
      }
    });
```

Recommendation APIs were also implemented in a similar way.

Contribution

- **Ashish Gupta and Isha Gupta -** Implemented Feature 3: Search and Recommendation functionality in Lastfm and Spotify and it's documentation
- **Soma Datta Reddy -** Implemented Feature 1: Better user management and it's documentation
- **Arth Raj and Vaibhav Kashera -** Implemented Feature 2: Better Library Management and it's documentation