Information Management & Systems Engineering

Milestone 1
Team 27

Adrian Boghean
a11742914

Calin-Nicolae-Virgil Ploscaru
a12142816

● 1.1.1 Application domain description

Our project will store all the relevant information required for a music streaming service

To be able to use a music streaming service, firstly a user has to register. To do so, a user has to sign up using an **e-mail address**. While creating a new account, the users need to also fill in their name and password. Another relevant attribute related to the user is the registration date. This streaming service will also support some sort of social features. For example, a user can follow another user (similar to Spotify's feature). A major difference between the existing streaming platforms and our service is that the users can review/rate albums.

Since it is a music streaming service, our application requires songs. A song is identified by a unique **song id.** Each song has a title, a length, and a release date. While streaming music, the user can like (add to favorites) different songs.

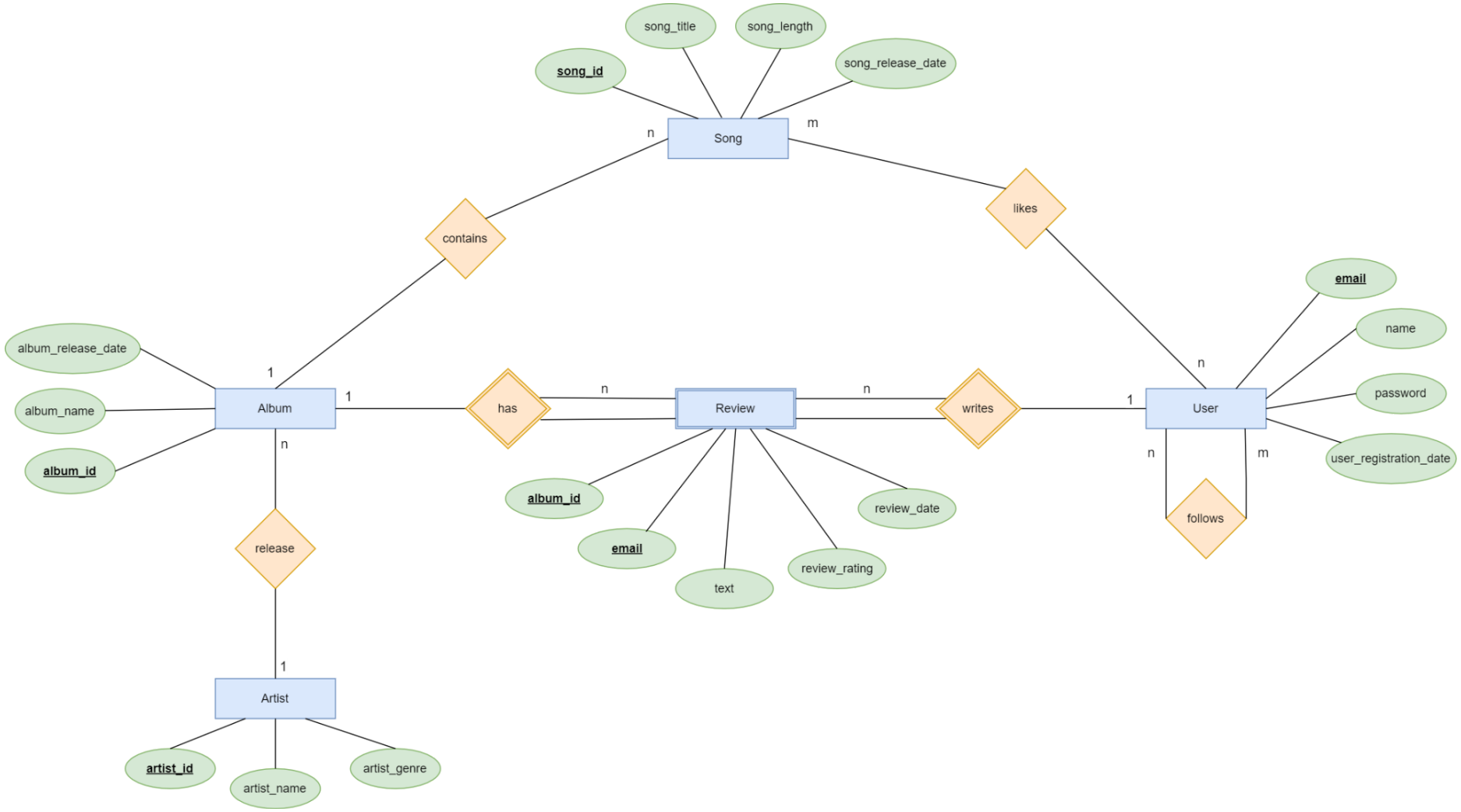
Each song belongs to an album. An album is identified using a unique **album id**. To simplify the problem, we will consider every single release to be an album. This is not far from the reality since most of the single "albums" have another song on the "B-Side". For example, one of the most famous single songs "Wind of Change" by the Scorpions has on the B-Side another song called "Restless Nights". As expected, each album has a name and a release date.

An album is released by an artist. To identify an artist, an **artist id** is used by the platform. Other attributes related to the artist are the artist's name and the artist's musical genre.

To make it easy for the users to discover new music, the platform will show some of the best-rated albums that are available. The album's rating is based on user reviews. Since on this service a review can't be anonymous, a review is identified by the **album id** and by the **user's email**. Each review has to specify a rating between 1 and 5 (5-star rating system) and a text which contains different users' remarks about the album. Another relevant attribute related to the review is the date when it was published.

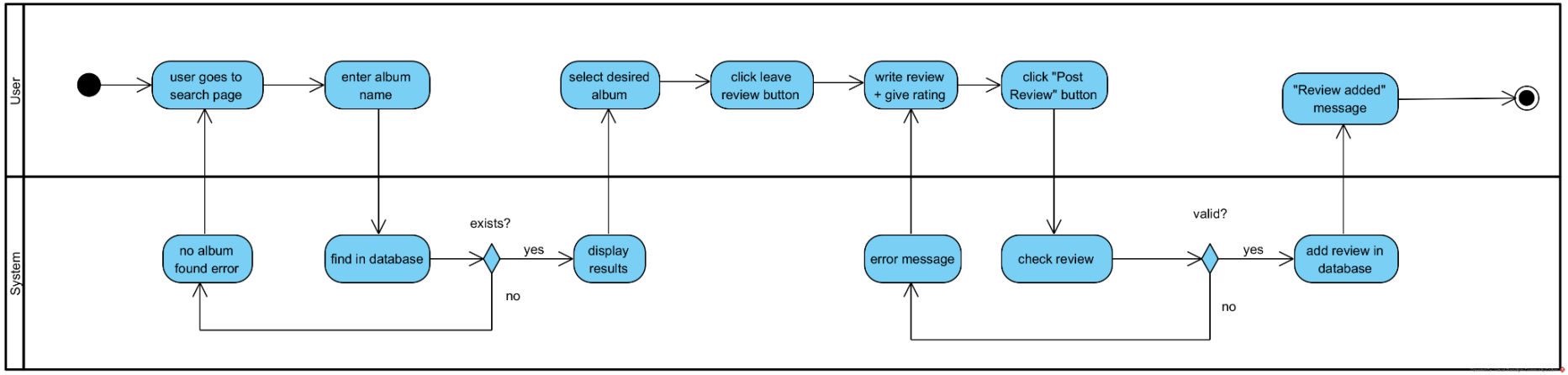
If a user decides to delete their account, all the reviews are deleted as well. The same happens if an album is removed from the streaming platform.

- **1.1.2 Entity-Relationship model**



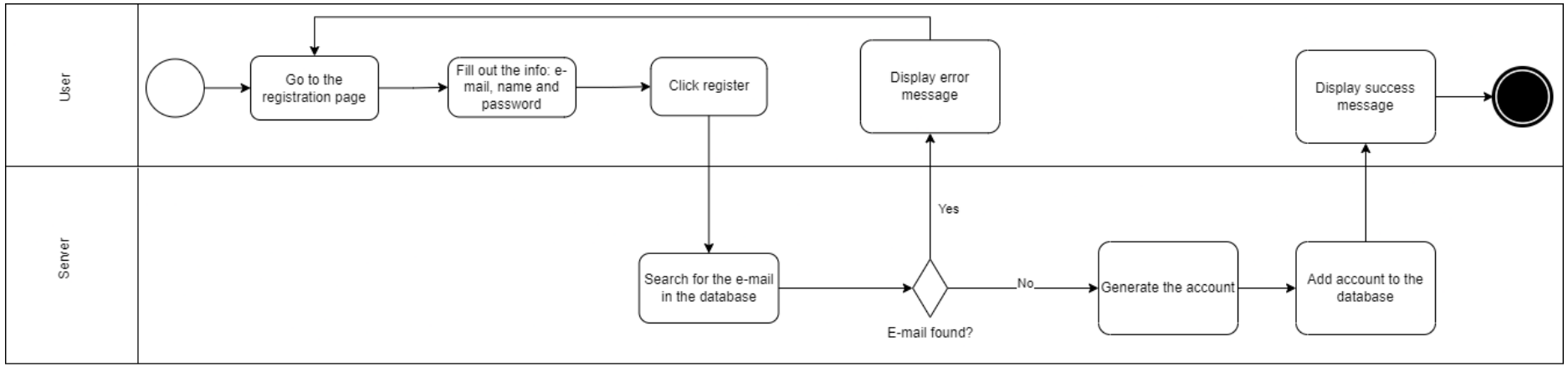
● 1.2 Use-Case Design

1.2.1 First main use case: Posting a review (Adrian Boghean)



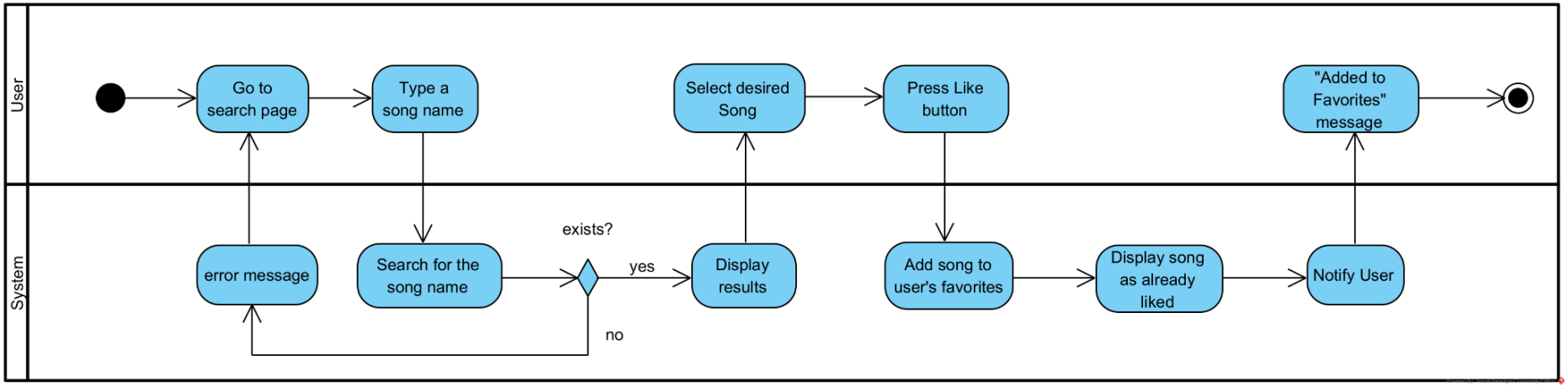
- **Objective:** Leave a review on an existing album.
- **Short Description:** A user of the streaming service searches for an album and if the album exists, the user can post a review.
- **Preconditions:** The user is registered and the album exists.
- **Expected execution:** The user searches for an album and finds it. Then, the user selects the desired album, clicks on the “Leave Review” button, writes the review, and then clicks on the “Post Review” button.
- **Postcondition on success:** The review is successfully added to the database and it is displayed in the review section of the album. The user gets notified that the review was added successfully.
- **Postcondition on error:**
    - **Case 1**: The user searches for an album that does not exist. The user is notified that it does not exist.
    - **Case 2:** The user has already posted a review on the same album. The user is notified and can choose between canceling and updating the last review.

1.2.2 Second main use case: User creates an account (Calin Ploscaru)



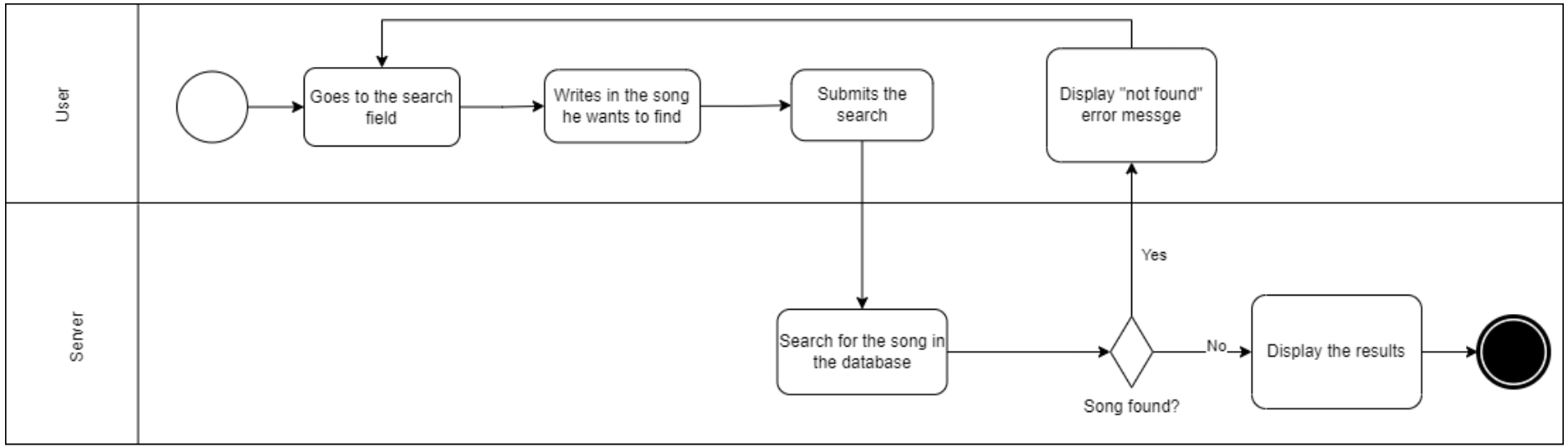
- **Objective:** Create an account on the streaming service
- **Short Description:** The user will try to create an account in order to be able to use the music streaming service.
- **Preconditions:** None.
- **Expected execution:** The user will go to the registration page. There, the user will fill out the necessary information such as an e-mail, name, and a password. The user will then click the register button. Afterwards, the server will search for the e-mail in the database.
- **Postcondition on success:** The account will be generated and then added to the database. Then a success message will be shown to the user.
- **Postcondition on error:** An error message will get displayed and it'll prompt the user back to the registration page to fill out the info again.

1.2.3 Third use case: User likes a song (Adrian Boghean)



- **Objective:** Add a song to the user's favorite list.
- **Short Description:** A user of the streaming service searches for a song and wants to add it to the favorites. To do so, the user has to press the like button that appears near the song's name.
- **Preconditions:** The user is registered and the song exists.
- **Expected execution:** The user goes to the search page and types a song name. The system returns a list of songs that matches the user's search. Near the song's name, a like button is visible. The user can click that button and the song is added to their's favorite list. The like button also changes the way it looks so that the user can be aware that the song already exists in the user's favorite list.
- **Postcondition on success:** The song is added to the user's favorite list. A "success" message is displayed.
- **Postcondition on error:** Song does not exist or is not available on this service. The user is notified and is returned to the search page.

1.2.4 Fourth use case: User searches for a song (Calin Ploscaru)



- **Objective:** Search for a song on the streaming service
- **Short Description:** The user will search for a song he wants to find on the streaming service
- **Preconditions:** The user is registered into the platform.
- **Expected execution:** The user will go into the music search field and type in the song he wants to find on the platform. He will then click the submit button and the server will try and find the song in the database.
- **Postcondition on success:** The server finds the matching songs and it will display these results to the user
- **Postcondition on error:** A “not found” error message will get displayed that will prompt the user back into the search field.

● 1.3 Reporting

1.3.1 Each artist's top-rated album. (Adrian Boghean)

- **Description:** This report will display for each artist the best-rated album. When a user posts a review, the user has to give a rating to the album. The top-rated album for each artist is the one with the highest average user score. To do so, the report will filter all albums by artist and find the one with the best rating.
- **Entities:** Album, Artist, Review
- **Filter by**: Artist
- **Sort by:** Review Rating

1.3.2 Albums with the most reviews from accounts created last year (Calin Ploscaru)

- **Description:** This report aims to find out the albums that have had the most reviews from new accounts that have been created last year. In order to achieve this, we get the albums that have had the most reviews and then we filter them out by the account's registration date.
- **Entities:** Album, User, Review
- **Filter by:** User
- **Sort by:** Review count