

Sebastian Zimmeck
COMP 333: Software Engineering, Spring 2022
Homework 3: Frontend (JavaScript, React)

Homeworks 2 – 4 will cover an ongoing app development project. A crucial part is to work together as group. It is recommended that you use GitHub to organize your work and keep track of the evolution of your codebase.

Check out the code examples, tutorials, and slides on the class website. They are intended to get you started and illustrate core ideas covered in the homeworks.

Problem 1 [70 Points] Having created the backend for your online music platform, you are getting to work on your frontend. Instead of using Django templates you decide to build your frontend with JavaScript and React.

(a) You decide to develop your full stack CRUD app with a React frontend that integrates with your Django backend. In particular, your app is intended to allow a user to do the following via a graphical user interface:

- Create new songs in a database with title, artist, and rating
- Read a list of songs from the database with title, artist, and rating
- Update title, artist, and rating of existing songs in the database
- Delete songs (title, artist, and rating) from the database

The rating of a song should be based on a scale of 1 to 5.

A user should be only able to rate a particular song once. A user should be notified that they cannot rate a song twice if they try. A user should also be able to change their rating of a song.

Songs (title, artist, and rating) should be displayed for all users in the database. For example, if Sally entered “While we’re young” and Don entered “Sweet Jane,” both Sally and Don will see a song list containing “While we’re young” and “Sweet Jane.” The rating of a song that both Sally and Don see should be the average song rating of all your users.

You can assume that there are no two songs and no two artists with the same name. In other words, you do not need to check for duplicate songs or artists.

You can use an additional ID attribute that increments an integer in any of your database tables as a primary key, e.g., if it proves easier for deleting data consistently across tables. Use of such primary key is just one option. There are many possibilities to achieve consistency across your database.

In addition to React and Django you can use any library you like; for functionality, styling, fonts

Here are some screenshots for inspiration.

The image displays three sequential screenshots of a web application titled "Song Rater".

First Screenshot: The application shows a list of songs on the left: "Even The Losers", "Lyrics To Go", "Born in the USA", "26", "Billie Jean", and "Third Wind". Each song has a pencil icon and a trash can icon to its right. A "New song" button is at the bottom left.

Second Screenshot: The "Even The Losers" song is selected. The edit form on the right has the following fields:

- Title:** Lyrics To Go
- Artist:** A Tribe Called Quest

An "Update" button is located below the artist field.

Third Screenshot: The "Even The Losers" song is now highlighted in orange. The edit form on the right shows the rating section:

- Tom Petty** (with a 4-star rating: ★★★★★(4))
- Rate it** (with a 5-star rating: ★★★★★)

Here are some tips for getting started:

- **Use your Django backend that you have developed for the previous homework.** While probably most of your backend code can stay as is, you may need to adapt parts of it as you are integrating it with your React frontend. Some of the features you had in mind during homework 2 may not work as you integrate your backend with your new frontend.

- **Take inspiration from the Todo Django React app.** You may find it insightful to inspect the Todo Django React app as described in the [react-django-tutorial.md](#). There is also a version of the app that is ready for you to use. The Todo app is a minimum example of a full stack web app based on Django and React with features that you can use in your app as well. Understanding the Todo app will help you developing your app.
- **Sending HTTP requests from your frontend that are properly received at your backend will be the core functionality of your app.** If your frontend properly sends HTTP requests, your backend can then manipulate the database per these requests and respond. In order for your frontend to send HTTP requests you can use an HTTP request library such as axios (you are free to use a different one if you prefer). As shown in the Todo app from the tutorial, your frontend could make async (promise-based) requests as follows:

```
axios
  .get("http://localhost:8000/api/todos/")
  .then(res => this.setState({ todoList: res.data }))
  .catch(err => console.log(err));
```

For now, it is sufficient to use localhost. Once you deploy your app in homework 4, you would need to switch to your real URL online.

- **Test your requests.** You can test your requests and responses using Postman (<https://www.postman.com/downloads/>). Note that different types of requests (GET, POST, ...) are suitable for different types of database actions (create, read, ...). If you have installed the Django REST framework, you can also use its routers page to check the state and workings of your requests (e.g., for the Todo app at <http://localhost:8000/api/todos>) as shown below. Another useful tools for checking the state of your database may also be the DB Browser for SQLite (<https://sqlitebrowser.org/>). Once you determined that your requests are working, you can implement them in your React frontend.



- **index.js -> App.js -> Components is one way to structure your React app.** index.js is the entry point for all user interaction. index.js imports App.js, which is the parent component importing child components (e.g., a component for a song list, a component for changing song entries, ...). There are many ways to structure your app and this way is just one.

- **Develop incrementally and separate work into individual GitHub issues.** It is often quicker to evolve your app step-by-step as opposed to trying to do too much at once. Try to separate your work into individual issues so that you and your partners can each work individually on a separate piece of code. For example, crafting the set of different HTTP requests that trigger a certain database action (create, read, ...) could be one issue.

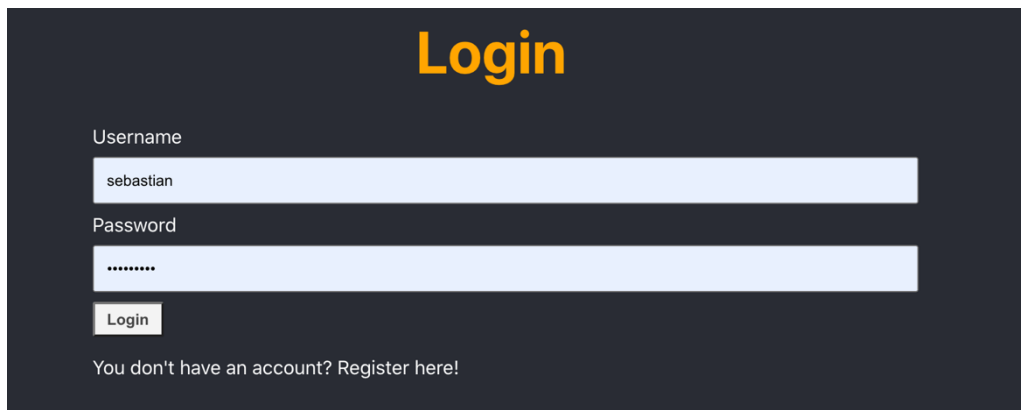
Problem 2 [30 Points] To make your app more exciting and functional for your users you decide to add at least one additional big feature or, alternatively, three additional small features.

It is your choice which features you are implementing and how.

One important consideration is that your choice may have an impact on the mobile version of your app in the next homework. The reason is that your mobile app will share the same backend as your web app. For example, if you decide to implement user authentication for your web app, you would also need to implement it for your mobile app.

Here are some suggestions for big features:

- **User authentication** is not trivial but very important. In addition to checking the database for the username and password a user entered, you would also need to implement the functionality for tokens to be passed along with a user's HTTP requests to keep track of the logged-in state beyond the initial authentication.



If you decide to implement user authentication, your functionality should allow a new user to register with a username and password. If a user enters a username that is already taken, the user should be alerted accordingly and asked to pick a different username. Passwords do not need to be unique.

There are different ways to implement user authentication. One way is to use the Django REST framework (<https://www.django-rest-framework.org/api-guide/authentication/>). To get started you can find sample code for user authentication in the Django tutorial.

- **Functionality for playing songs.** Please only use royalty-free music, e.g., from <https://cchound.com/>.
- **Integration with a third party app.** For example, it may be possible to integrate Twitter (<https://developer.twitter.com/en/docs/platform-overview>), though, I am not really sure; please make sure that you follow all applicable rules if you go that route.

Small features that you could implement are, for example:

- **Keep track of individual users**, e.g., via cookies (<https://stackoverflow.com/a/26211578>) to only display songs and artists the user has entered instead of songs and artists all users entered.
- **Check for song and artist duplicates**. If a user tries to enter a song or artist entry that already exist, disallow such entry and notify the user accordingly.
- **Add React notifications** (<https://www.npmjs.com/package/react-notifications>), e.g., in case of duplicates.
- **Sort or search functionality** (e.g., all songs from a particular artist or with an average rating above a certain threshold).
- **Playlists**. Add functionality for users to create custom lists, e.g., similar to playlists on Spotify.
- **Statistics**. Add lists with top rated songs, average song ratings, number of songs per artist, etc.

You can also implement your own features. If you would like to discuss a feature or you are unsure whether a particular feature is a big or small feature, feel free to discuss it with us.

Submission

Please upload all your files in one zip folder to Moodle. You only need to submit one solution per group. It does not matter which student's Moodle account your group is using. The homework is due on **4/15/2022, 10 am**.

- Include **both your React frontend as well as your Django backend** in the submission.
- Only submit source files, including automatically generated source files. Do not include dependencies or automatically generated non-source files (e.g., .pyc files).
- Provide **clear instructions** in a readme.md on how to install and run your app. Test that your app is running per your instructions. **It must be possible for us to recreate your app on our end by just following your instructions.** The easier you can make it for us to review your work, the better.
- Explain how to use the additional features that you have implemented.

Grading

Please try to distribute the work evenly and indicate who did what in the readme(s) you provide. Unless you tell us otherwise, we assume that all students in a group contributed equally and, thus, the grade of each student will be equal as well. It is not possible to challenge a grade after grading based on one student having contributed more than another. If a student contributes less than their equal share, the student's grade will be reduced accordingly. Similarly, if a student contributes more than their equal share, the student's grade will be increased accordingly up to the maximum available grade.