DAT076 - Projektrapport

Erik Berglind
Isak Genberg
Luqas Lundahl
Hannah Tu

March 2025

1 Introduction

BeerCritics is a web application designed for people who love beer. It allows beer enthusiasts to discover, review and critique different kinds of beer. Users can explore a variety of beers due to its database, read reviews from others, and also contribute their own opinions.

Entering the website, the user is greeted with a homepage consisting trending beers to capture the user's interest, a search bar for easy navigation, an "about us" section and a responsive navigation bar for navigating different sections of the site.

Anyone can view and read reviews by clicking on a specific beer, but to submit their own reviews and access the personal view "My Reviews" the user must be registered and logged in. Depending on their login status, the users will see different versions of the navigation bar, ensuring better usability and a more functional website at all times.

Here is a link to the git repo: Github link

2 A list of use cases

- The user can search for a specific beer
- The user can view all beers in the database
- The user can read reviews by clicking on a beer
- The user can submit a review while logged in
- The user can edit an existing review while logged in
- The user can delete an existing review while logged in
- The user can log in/register/log out
- The user can change the username
- The user can navigate through different pages using the navbar and footer
- The user can see information about a specific beer by clicking on a beer card.

3 User manual

3.1 Main Page

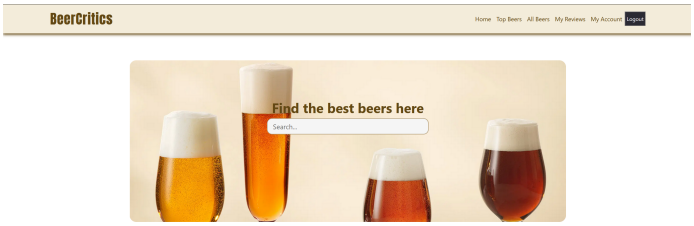


Figure 1: Main page initial state when logged in

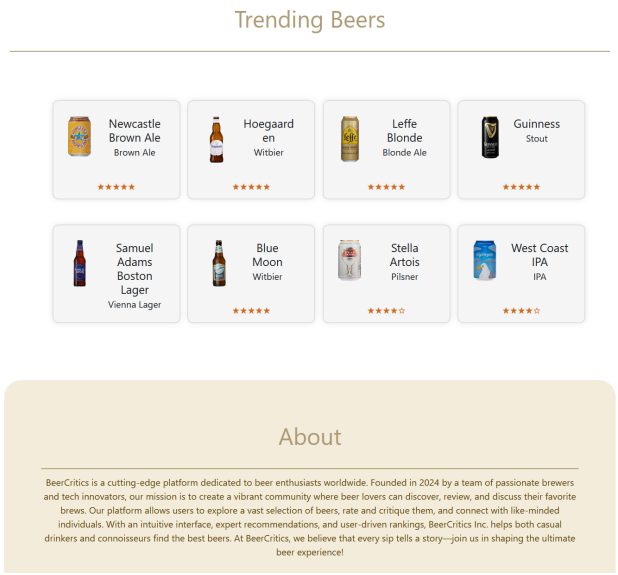


Figure 2: Main page after scrolling down

3.2 Running the application

1. Navigate to Github repository and clone the main branch.
2. Create a .env file in the root of the server directory
3. Add $SESSION\_SECRET$ and $DATABASE\_URL$ to the env file. Set the $DATABASE\_URL =$ `"postgres://radams_favoriter:W665jgMEal9GOW5MAbk7@localhost:5432"`
4. Run this command in a terminal:

```
--env POSTGRES_USER=radams_favoriter --env
    POSTGRES_PASSWORD=W665jgMEal9GOW5MAbk7 --publish
    5432:5432 --name web_apps_db --detach postgres:17
```

5. Start docker desktop and start the container.
6. Use the 'import from file' feature in BeeKeeper to insert the data from the JSON files in the data folder into the database to hydrate the page with content.
7. Open a powershell terminal and run *cd server*
8. In the same terminal, run *npm install* followed by *npm run dev* to start the server
9. Open a second powershell terminal and run *cd client*
10. In the same terminal, run *npm install* followed by *npm run dev* to start the client
11. Navigate to localhost and you should see the homepage of BeerCritics.

3.3 Using the application

3.3.1 Search for a beer

1. Whilst at the homepage, locate the searchbar at the top of the page
2. Click the searchbar and enter the name of an existing beer, e.g Guinness
3. Press enter and you should be brought to the desired beer page.
4. Searching for a beer that does not exist will tell you exactly that. Also note that the input is case sensitive, e.g guinness will not bring you to the Guinness page.

3.3.2 Place a review

1. Click on a beer card
2. Scroll down and click on the add review button
3. Fill in your review
4. Click on add review to save the review

3.3.3 Edit your review

1. To edit your review you must be logged in
2. Click on a beer card
3. Click on the edit review button
4. Edit your review
5. Click on save button to save the edited review

3.3.4 Delete your review

1. To delete your review you must be logged in
2. Click on a beer card
3. Click on the delete review button

3.3.5 View top beers

1. In the navbar, click on "Top beers"

3.3.6 View all beers

1. In the navbar, click on "All beers"

3.3.7 Register a user

1. In the navbar, click on "Register Account"
2. Fill in the users credentials
3. Click on the Register button

3.3.8 Login a user

1. In the navbar, click on "Login"
2. Fill in the users username and password
3. Click on the Login button

3.3.9 Change username

1. Make sure you're logged in.
2. Navigate to "My Account".
3. Click the "change username" button.
4. Provide your new username and press submit.

4 Design

4.1 Packages

The app is built with React, typescript, and bootstrap, ensuring a seamless and interactive user experience. The packages used to build the app were: Express, Node, CORS, Sequelize, ts-node-dev, Jest and Supertest. AXIOS, express-session, and bcrypt.

4.2 A list of the components in the frontend

4.2.1 Pages

- All beers page
  - Purpose: To show all available beers in the database.
  - State: A list of beers.
  - Calls to back-end: Sends a GET-request to ‘/beer‘ to fetch all available beers.
- Beer page
  - Purpose: Display information and reviews for a specific beer.
  - Params: name - The name of the beer.
  - State: The specified beer and it’s list of reviews.
  - Calls to back-end: Sends a GET-request to ‘/beer/BEERNAME‘ to fetch the beer. Sends a GET-request to ‘/review/BEERNAME‘ to fetch reviews for the specified beers.
- Home page
  - Purpose: To get ability to search for beers, display a sample of the beers in the database and read about the creators of the website.
  - Calls to back-end: GET-request to ‘/beer‘ to get all available beers in the back end.
- Login page
  - Purpose: Log in page for users.
  - State: Login credentials (username and password). Errors to handle visualizing them in the front-end.
  - Calls to back-end: Sends a POST to ‘/user/login‘ along with the username and password.
- My Account page
  - Purpose: Show personal details such your username. An ability to change username exists.

  - State: The currently logged in user.
  - Calls to back-end: GET-request to '/user' to find the currently authenticated user.

- My Reviews page
  - Purpose: Collects all of the reviews written by the currently authenticated users.
  - State: The list of reviews.
  - Calls to back-end: GET-request to '/review/myreviews' to find the currently authenticated user's reviews.

- Register page
  - Purpose: To register a new user.
  - State: Login credentials (username and password). Errors to handle visualizing them in the front-end.
  - Calls to back-end: POST to '/user' to register the new user.

4.2.2 Smaller UI Components

- Beer Card
  - Purpose: Cards displaying beers in pages such as all beers page, etc.
  - Props: Image path, name, style, rating.
  - State: None
  - Calls to back-end: None

- Footer
  - Purpose: Get quick access to some useful links to navigate the website.

- Login Modal - (NOT IN FINAL PRODUCT)
  - Purpose: Prompt the user to log in/register
  - Props: show, handleClose()

- Logout Button
  - Purpose: To log out from the web site.
  - Calls to back-end: POST to 'user/logout'

- Review Card
  - Purpose: Displays reviews in beer pages.
  - Props: Beer, rating, user, description, date.

  - State: isCreator: boolean to check if the logged in user is the creator of the review.
  - Calls to back-end: GET request to ‘/user‘.

- Review Card
  - Purpose: Displays a list of reviews.
  - Props: fetchReviews, onAddReview, onReviewsFetched
  - State: reviews, used to get the total number of reviews
  - Calls to back-end: GET request to ‘/review‘.

- Review Modal
  - Purpose: Modal panel that appears when clicking a review card. This is where you leave reviews.
  - Props: show, beer, handleClose(), handleSave().
  - State: rating, user, description, date.
  - Calls to back-end: GET request to ‘/user‘.

- Star Rating
  - Purpose: Displays the average rating as 5 more or less filled stars.
  - Props: rating, onRatingChange(), isClickable, className

- Delete Review Button
  - Purpose: Delete a review in beer pages
  - Props: review
  - State: loading, currentUser
  - Calls to back-end: GET request to ‘/user‘ and DELETE request to ‘/review‘.

- Review Button
  - Purpose: Modular button that allows user to either add or edit reviews.
  - Props: beer, review, mode, onAddReview
  - State: showModal
  - Calls to back-end: POST to ‘/review‘ and PUT to ‘/review‘, based on the ”mode” prop.

- Change User Data Modal
  - Purpose: Modular modal component that allows for user data based on passed in props.

  - Props: btnText, currentUser, update()
  - State: show, formText. inputValue, errors
  - Calls to back-end: PATCH to '/user'.

- Navbar
  - Purpose: Makes page navigation accessible at the top of the page.

- Searchbar
  - Purpose: To search for a specific beer.
  - State: searchInput
  - Calls to back-end: None but navigates to a specific beer page if beer was found from a search string.

- AboutSection
  - Purpose: Provides a text describing the history and purpose of Beer-Critics

4.2.3 Other Front-end Components

- api
  - Purpose: Defines and exports a set of reusable functions to communicate with the backend API using Axios.
  - Calls to back-end: Handles all calls to the back-end from the front-end.

- Layout
  - Purpose: Give a consistent layout for the web site, with a nav-bar at the top, content in the middle, and a footer at the bottom of the page.
  - Props: Children: the content in the middle of the navigation barion bar and the footer.

- AuthContext
  - Purpose: Creates and exports the authentication context to allow all components in the front-end to access the currently logged in user.

- AuthProvider
  - Purpose: Context provider for AuthContext.
  - Props: Children
  - State: isLoggedIn
  - Calls to back-end: GET request to '/user'

4.3 API for the backend

4.3.1 Requests to /beer

- GET - Gets all of the available beers in the database, responds with 200 if the request was a success, or 500 if there was an internal server error.
- GET to /beer/:name - Gets a specific beer from the database. Response with 200 if the request was a success, or 500 if there was an internal server error, such as the beer was not found.

4.4 Requests to /user

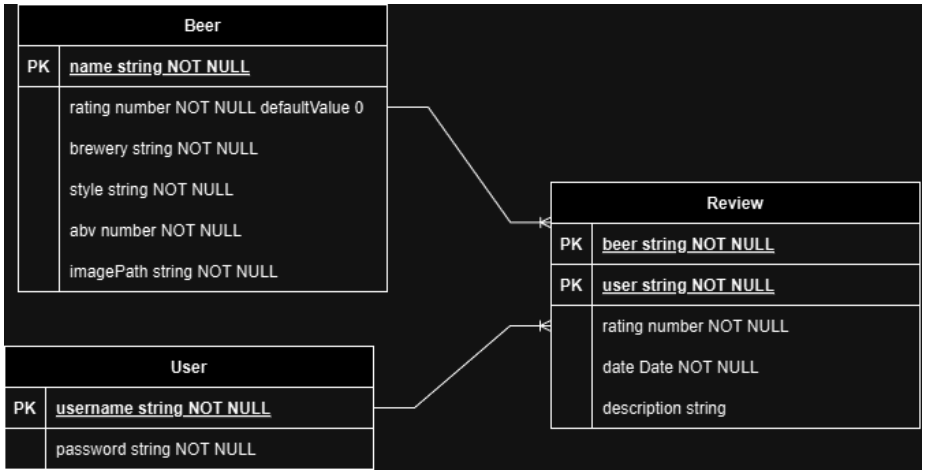
- GET to /user/myreviews - Gets the currently logged in user's written reviews. Responds with 200 if it was a success, 401 if the user isn't logged in, or 500 if there was an internal server error.
- GET to /user/:name - Gets all of the reviews for a specific beer. Response with 200 if the request was a success, or 500 if there was an internal server error.
- POST with body review: Review- Adds a review to the database. Responds with status code 200 if the request was a success, 401 if the user is not logged in, or 500 if there was an internal server error.
- PUT with body review: Review - Updates a review. Responds with 401 if the user was not logged in, 200 if the request was successful, or 500 if there was an internal server error.
- DELETE with body review: Review - Deletes a review. Responds with 401 if the user was not logged in, 200 if the request was successful, or 500 if there was an internal server error.

4.4.1 Requests to /review

- POST with username: string, password: string - Registers a new user. Responds with 200 if the request was successful or 409 if the provided username already exists.
- POST to /user/login with username: string, password: string - Logs in a user. Responds with 200 if the request was successful or 401 if the provided credentials were invalid.
- POST to /user/logout with username: string, password: string - Logs out the currently logged in user. Responds with 500 if there was an error logging out, or 200 if it was successful.
- GET with username: string, password: string - Gets the current user in session. Responds with 401 if the user isn't logged in, or 200 if it was successful.

- PATCH with oldUsername: string, newUsername: string - Responds with 200 if the request was successful, 401 if the user is not logged in, 409 if the provided new username already exists, or 500 if there was an internal server error,

4.5 Entity-relationship (ER) diagram



5 Responsibilities

- Erik: Wrote some of the back-end code, but mostly contributed to the front-end. Worked on multiple components but created AllBeersPage, Footer, LayOut, api.ts, AuthContext, AuthProvider, and MyAccount-Page. Also wrote most of the component and API specifications in the report.

- Isak: Hosted the github repository and set up issue templates. Co-authored the setup for the first 3 labs. Added documentation throughout the application and wrote the readme file. Worked on all parts of the application, but mostly contributed on the front-end, creating components such as BeerCard, ReviewButton, ReviewModal, StarRating and more. Worked on css across the application. Did lots of reviews.

- Luqas: Wrote the majority of the backend code and designed and implemented the database. Designed and implemented all the routes and services. Also made api methods to access said routes. Designed the Top-Beer page, search bar, and various small changes on the frontend as well as connecting all the frontend components to the backend, since he had the more holithic view and understanding of the server components. Implemented sessions and authentication to protect routes and pages (not including authentication context)

- Hannah: Primarily contributed to the front-end of the app, developing components and pages, such as Navbar, Homepage, Login Page, and Register Page. Created the root CSS and also conducted several code reviews. Additionally, participated in the weekly labs, made the presentation slides and contributed to the introduction, a list of use cases and the user manual in the report.