

2022-2023

# Web Technologies: Project Report

Jolien Andries, Mats Van Molle, Abdullah

Sabaa Allil

2022-2023

## Contents

1. Web Application Architecture .....	2
2. Functional Requirements .....	3
2.1. User Management .....	3
2.1.1. Registration .....	3
2.1.2. Signing in .....	3
2.1.3. Signing out .....	4
2.2. Profile .....	4
2.2.1. Account Overview .....	4
2.2.2. Modifying profile information .....	4
2.3. Home .....	5
2.3.1. Highlighted attraction .....	5
2.3.2. Feeds .....	5
2.4. Attractions .....	5
2.4.1. Top 10 attractions .....	5
2.4.2. Adding attractions .....	6
2.4.3. Browsing attractions .....	6
2.4.4. Viewing attractions .....	6
2.4.5. Modifying attractions .....	7
2.5. Theme parks .....	7
2.5.1. Adding theme parks .....	7
2.5.2. Browsing theme parks .....	8
2.5.1. Viewing theme parks .....	8
2.5.2. Modifying theme parks .....	8
2.6. Events .....	8
2.6.1. Adding Events .....	8
2.6.2. Browsing Events .....	8
2.6.3. Viewing Events .....	9
2.7. Social Aspect .....	9
2.7.1. Rating and reviewing attractions .....	9
2.7.2. Join events .....	9
3. Technical Requirements .....	10
3.1. AJAX .....	10

3.2.	Form Validation.....	10
3.3.	CSS.....	10
3.4.	HTML5 .....	10
3.4.1.	Local storage .....	10
3.4.2.	Drag and drop .....	10
3.4.3.	Pattern matching .....	10
3.5.	Web Services .....	10
3.6.	Publish Content.....	11
3.7.	Mobility and Responsiveness.....	11
3.8.	Maps.....	12
3.9.	Additional Technical Features.....	12
3.10.1.	Pagination .....	12
3.10.2.	Visibility of system status.....	12
3.10.3.	Handling images upload.....	13
3.10.4.	deployment of the site.....	13
4.	Starting the site locally.....	13
5.	The Source Code .....	14
Appendix: Timetable .....		14
Abdullah .....		14
Mats .....		15
Jolien .....		16

## 1. Web Application Architecture

The project works according to the client-server model. The server is responsible for storing all the data of the website, such as the user data and the attraction data. It provides functionalities to the client to manipulate or extend this data. The client side is responsible for the graphical user interface with which the user will be interacting. The client will communicate with the server to get, add or update some data.

For the front end, we use the React framework, with React-Bootstrap for components, and React-Router-Dom for the front-end routing. In the back end, we use Express with Node.js. For our database, we use SQLite 3. The programming language we used in the project is Typescript.

The codebase of the project was divided into different parts, each representing a major component of the project. We have a home page, containing a highlighted attraction and a feed with the recent activities. We have a map, which shows all theme parks and their current weather. We have a theme park section, where the options to browse or add a theme park are provided. The same goes for the

event section, the options to browse or add an event are available as well. There is also an attraction section, with the option to browse or add attractions or to see the top 10 attractions based on the user rating. Each attraction and theme park has its own page, in which its information can be viewed or modified. For attractions, there is a reviews section where users can leave reviews. Attractions has also a gallery section, displaying the image uploaded by the user.

The last important section of our project is the user profile. One has the option to register and login, and once that happens, they can see their profile, containing their account information, like avatar, name, e-mail and the events they joined.

## 2. Functional Requirements

### 2.1. User Management

Thrillreview has a token-based user management system. When the user logs in, the server sends a cookie with a JWT token to the client, which will be used by the server to recognize the user and authorize activities that require authentication (e.g., rating and adding attractions).

#### 2.1.1. Registration

New users can register a new account by providing their desired username, their email, and their password. This happens using a post request. The server will check whether the username already exists. If it does, the server will return status code 400 with a message indicating that the username already exists in the database. The same thing will happen with the email address that the user provided.

The provided password will be hashed before being stored in the database. This happens using the *Bcrypt* library, which uses the *bcrypt* password-hashing function. This hashing function is one-way and uses salts to make hashes more secured.<sup>1</sup>

If the user is correctly added, the server will send a message with the status code 200 indicating that the user is added to the database.

#### 2.1.2. Signing in

The user can log in by providing their username and password. This happens using a POST request. The server will first validate the user input and check if the username exists and if the provided password matches the stored hashed password. If the username exists and the passwords match, the server will create an access token as well as a refresh token using JWT. These tokens will be sent to the client in a cookie. The access token is 15 minutes valid while the refresh token is 1 day valid.

The server will then send the username and the user ID to the client indicating that the user is logged in successfully.

While the access token is valid every request that requires the user to be logged in will check if the access token is valid. Once the access token expires the browser will automatically remove that cookie. If the user does a request after the access token is expired, the refresh token will make it possible to

---

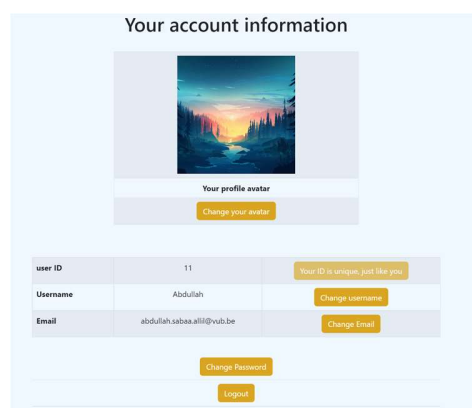
<sup>1</sup> <https://en.wikipedia.org/wiki/Bcrypt>

generate a new access token that is valid for another 15 minutes. Besides the new access token, the server will also send a new refresh token.

### 2.1.3. Signing out

Once a user signs out, the refresh and access tokens are removed from the browser. While these are removed, they would still be valid if someone managed to get these tokens. This is why we store all our current valid refresh tokens in our database. When generating a new access token with the refresh token we make sure that the refresh token is in our database, if not we will not generate a new refresh token. So, when a user signs out we also remove that token from our database so it can never generate a new access token.

## 2.2. Profile



The screenshot displays a web interface titled "Your account information". At the top, there is a placeholder for a profile picture with the text "Your profile avatar" and a "Change your avatar" button. Below this is a table containing user details:

User ID	11	Your ID is unique, just like you
Username	Abdullah	Change username
Email	abdullah.sabaa.ahli@vub.be	Change Email

At the bottom of the table, there is a "Change Password" button. Below the table, there is a "Logout" button.

Figure 1: The profile information of the user.

### 2.2.1. Account Overview

Every user has a unique ID, a username, an email and an optional avatar (profile picture) that they can upload. All this information will be shown in the *Profile* section on the client side. Via the same page, users can request to modify their information (e.g., their avatar, email address, username).

### 2.2.2. Modifying profile information

A user can modify their information in the same *Profile* page. The user can provide their new desired information, and the client will send it to the server in a PUT request.

When the user uploads a new avatar, the client will send the uploaded avatar to the server as a multipart/form-data. Handling uploaded images (e.g., avatar) in the server happens using the *Multer* library that provides middleware for handling multipart/form-data.

## 2.3. Home

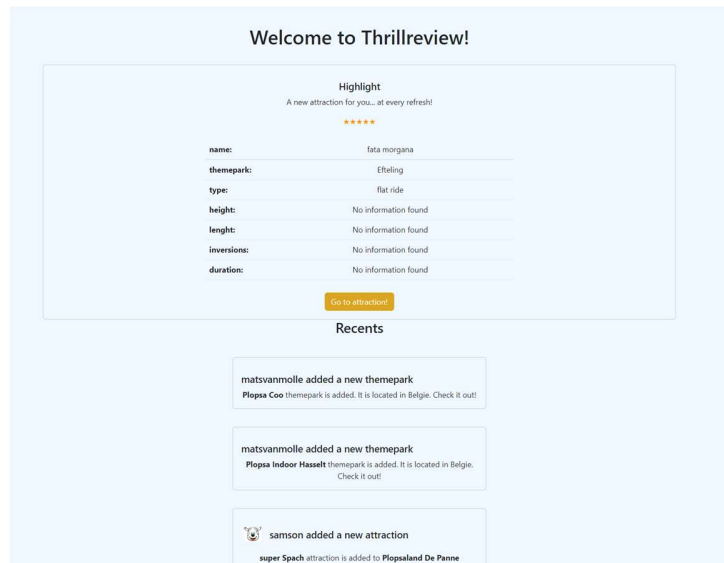


Figure 2: The home page of Thrillreview.

### 2.3.1. Highlighted attraction

At every refresh, a random attraction is highlighted, showing the information and its rating, the button links to the attraction page. The attraction is randomized by first requesting the number of attractions to the server, generating a random number between one and this number, then use this as an id to GET an attraction.

### 2.3.2. Feeds

The website has a feeds page that shows the nine most recent activities by other users. These activities can for example be a review that a user wrote or a new attraction or theme park that a user added. This is achieved by requesting the 3 most recent elements of the attractions, theme parks, and reviews tables from the database, and then randomly mixing them before sending them to the client.

## 2.4 Attractions

### 2.4.1. Top 10 attractions



Figure 3: Top 10 attractions.

The main attraction page has next to the browse/add attraction, a top 10 of attractions. This top 10 is based on the average rating of an attraction, so people immediately see which attractions they would like to check out.

### 2.4.2. Adding attractions

When users are logged in, they can add an attraction. If they are not logged in, they are shown a card leading them to the login page. The information that can be provided has different front-end validation checks (as well as server-side validation after the submission). This validation is done through pattern matching and the basic HTML input type validation. The theme park can only be one of the theme parks that already exist on our site, as we connect the attraction to the theme park. To see your images in preview, you can click on the “see images” button, to show them in a slideshow.

### 2.4.3. Browsing attractions

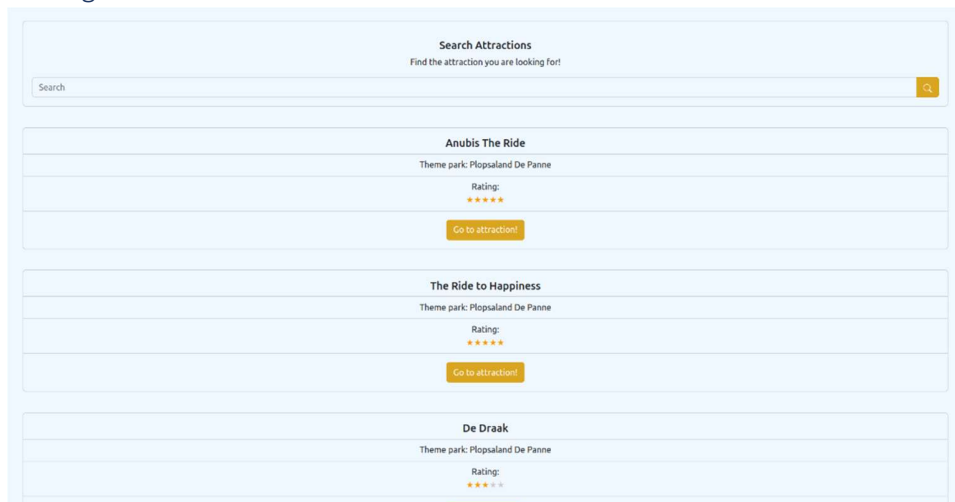


Figure 4: Searching and browsing the attractions.

When browsing attractions, one can search for both the attraction name and the theme park they are in. The results are paginated (back-end), and we use infinite scrolling for requesting the pages (front-end). All results get a preview, when clicking on the button, you go to the attraction page.

### 2.4.4. Viewing attractions

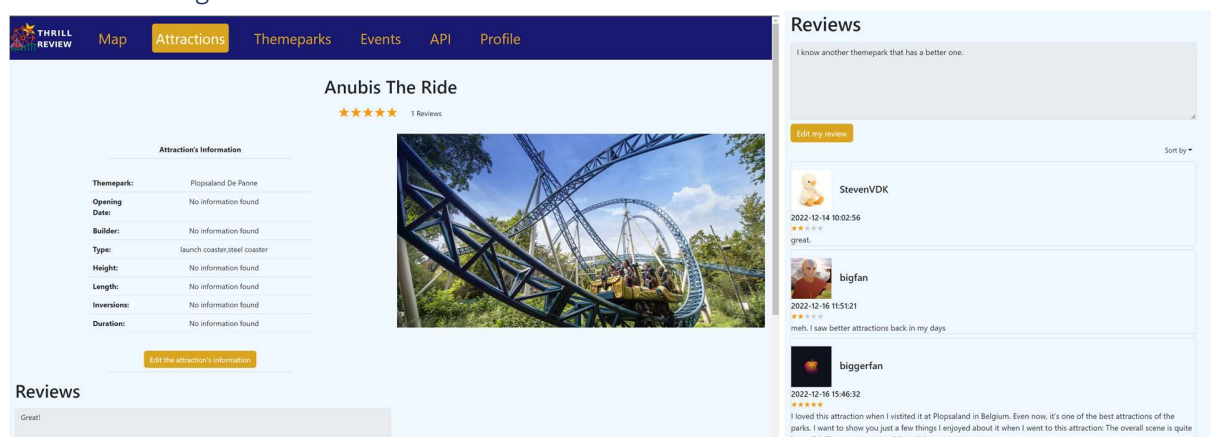


Figure 5 and 6: The attraction page, together with a reviews section.

Every attraction has its unique page consisting of its information, a gallery with its images, and a reviews section. On the attraction's page, users can leave reviews with ratings about the attraction. The image's gallery is implemented using *React Photo Gallery*, a responsive image gallery component.

#### 2.4.5. Modifying attractions

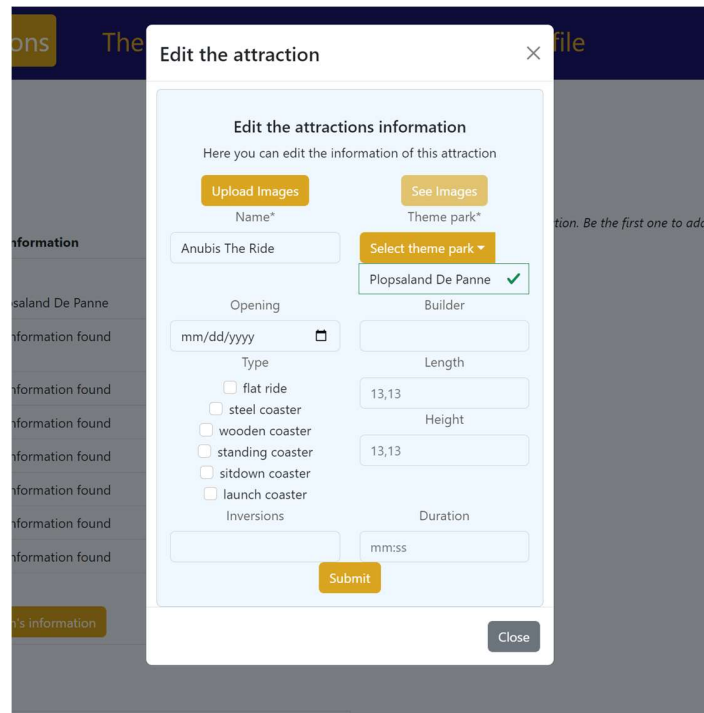
The image shows a modal window titled "Edit the attraction" with a close button in the top right. Inside the modal, there's a section "Edit the attractions information" with a subtitle "Here you can edit the information of this attraction". Below this, there are two buttons: "Upload Images" and "See Images". The form fields include: "Name\*" with the value "Anubis The Ride"; "Theme park\*" with a dropdown menu showing "Plopsaland De Panne" and a green checkmark; "Opening" with a date input field showing "mm/dd/yyyy" and a calendar icon; "Builder" with an empty text field; "Type" with radio button options: "flat ride", "steel coaster", "wooden coaster", "standing coaster", "sitdown coaster", and "launch coaster"; "Length" with a text input field showing "13,13"; "Height" with a text input field showing "13,13"; "Inversions" with an empty text field; and "Duration" with a time input field showing "mm:ss". At the bottom of the form is a "Submit" button. A "Close" button is located at the bottom right of the modal.

Figure 7: The “Edit the attraction” modal.

Users can modify the information about the attraction. They can for example add additional photos or add missing pieces of information. This can be done on the attraction's unique page. Pressing the “Modify the attraction's information” button will display a modal that has a form, allowing the user to modify the attraction's information.

### 2.5. Theme parks

#### 2.5.1. Adding theme parks

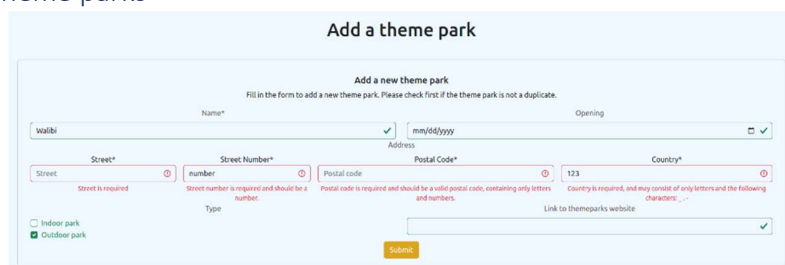
The image shows a form titled "Add a theme park" with a subtitle "Add a new theme park" and a note "Fill in the form to add a new theme park. Please check first if the theme park is not a duplicate." The form fields include: "Name\*" with a text input field; "Opening" with a date input field showing "mm/dd/yyyy" and a calendar icon; "Address" with sub-fields for "Street\*" (containing "Wallei"), "Street Number\*" (containing "number"), "Postal code\*" (containing "123"), and "Country\*" (containing "123"); "Type" with radio button options: "Indoor park" and "Outdoor park" (which is selected); and "Link to themeparks website" with a text input field. At the bottom of the form is a "Submit" button. Red error messages are visible below the address fields: "Street is required", "Street number is required and should be a number", "Postal code is required and should be a valid postal code, containing only letters and numbers", and "Country is required, and may consist of only letters and the following characters: ,-'".

Figure 8: Adding a theme park requires a valid address.

When adding a theme park, the address should be provided, this will be checked on validity in the backend, by requesting the latitude and longitude to Nominatim. When the address does not exist

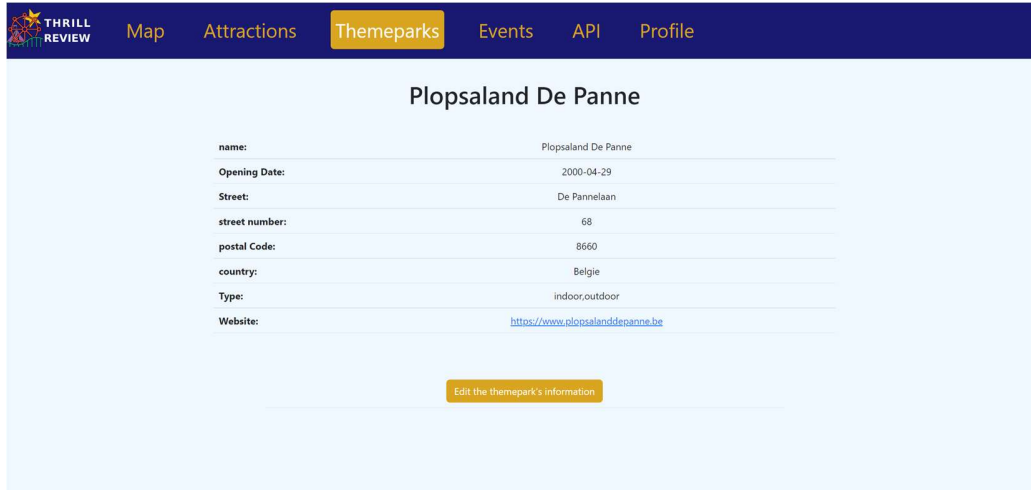


(because there is no geocoding for it), a pop-up is shown on the screen to make it even more visible than the standard form validation error.

### 2.5.2. Browsing theme parks

When browsing theme parks, you can search for the theme park as well as the country it is in. It is implemented with pagination and infinite scrolling. Again, a preview is shown with a button that leads you to the specific theme park page.

#### 2.5.1. Viewing theme parks



Plopsaland De Panne	
name:	Plopsaland De Panne
Opening Date:	2000-04-29
Street:	De Pannelaan
street number:	68
postal Code:	8660
country:	Belgie
Type:	indoor,outdoor
Website:	<a href="https://www.plopsalanddepanne.be">https://www.plopsalanddepanne.be</a>
<button>Edit the theme park's information</button>	

Figure 9: The theme park page.

Every theme park has its unique page consisting of its information (such as the address, the website and the type). There is a button to allow the user to modify the theme parks information.

### 2.5.2. Modifying theme parks

Users can modify a theme parks information if they want to correct some information or add some missing pieces of information about the theme park. This can be done by pressing the “Edit the theme park’s information” button in the theme parks page. This will display a modal that has a form, allowing the user to modify the theme parks information.

## 2.6. Events

### 2.6.1. Adding Events

Every user can add an event, when filling out the form. It gets connected to the theme park, that is why you can only choose existing theme parks. The user can then specify the date and time using the date picker or by entering the date, which will be validated as a correctly formatted date. This is achieved using an HTML input field with the *date* type. The same goes for the time, by using the html type *time*.

### 2.6.2. Browsing Events

When browsing events, you can search on both event name and the theme park where the event takes place. The results are shown with pagination and infinite scrolling.

### 2.6.3. Viewing Events

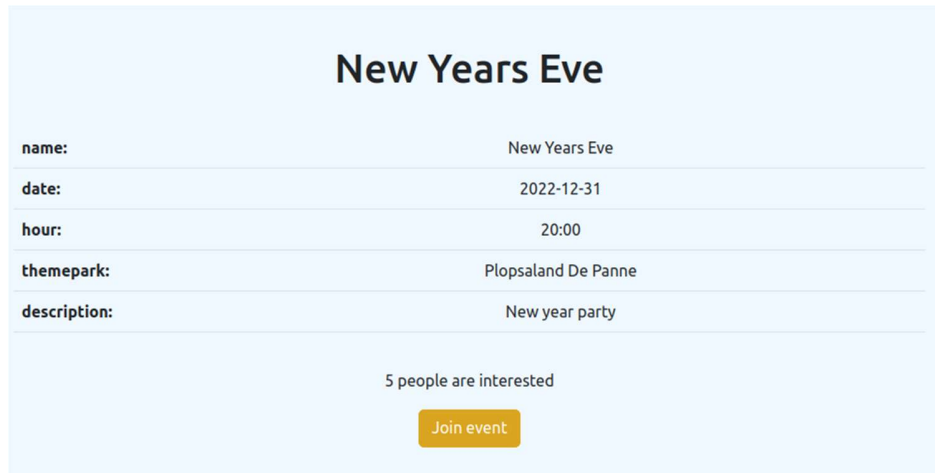


Figure 10: The event page.

When viewing the event, you see all the event information and how many people already joined it. There is also a button to join (see Social aspects). No modification of an event is possible, because we do not want users to edit an event.

## 2.7. Social Aspect

### 2.7.1. Rating and reviewing attractions

Users can write reviews about attractions along with their own rating (on a scale from 1 star to 5 stars, 1 star being the worst) to that attraction. Both reviews and ratings are stored in the database along with the time the review is posted, the attraction ID and the user ID. Users can also modify their review and their rating if they want to, which will then update the database and change the row corresponding to that review.

Every attraction's page has a reviews section that will display all the reviews that users wrote about that attraction. Reviews can be ordered by date or by how many stars, ascending or descending. The ordering happens at the server side, when the reviews are queried.

The attraction's page will also show the average rating of that attraction along with how many reviews there are. The server will execute a query for the database to compute the average rating and the total amount of reviews for that attraction.

### 2.7.2. Join events

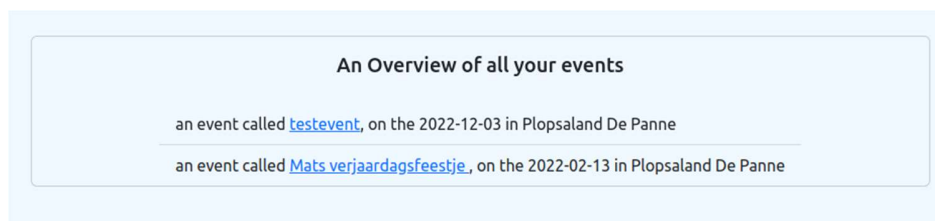


Figure 11: In the profile page, an overview of personal events is displayed.

On the event page, you have a button to join the event when you are logged in. This button is disabled when you joined already. Anyone can see the number of users that joined the event.

When you go to your profile page, you see a list of all events you joined. This is paginated, with infinite scrolling, and sorted by date.

### 3. Technical Requirements

#### 3.1. AJAX

The client-side uses *Axios*, which is a promise-based HTTP client. *Axios* is used to send AJAX requests and thus communicate with the server.

#### 3.2. Form Validation

The project validates forms in two ways: server-sided validation, and client-sided validation. The server will validate that the input data does not break any constraints (e.g., when a new user registers an account, they cannot use a username that is already used, otherwise, the server will send an error message to the client). The client-side will validate that all required input fields are filled in and have the correct format.

#### 3.3. CSS

CSS was used in the project to style different elements. For example, to add a background to the login and register page, to center cards in the middle of the screen, to make our elements decently sized no matter what screen you are on, to make sure reviews are correctly previewed on each screen size. It was also used to change the text size in some parts of the project, and colors of different components (such as buttons).

#### 3.4. HTML5

##### 3.4.1. Local storage

When the user logs in, the client side will store their information in the local storage reserved for the website. This will allow the client side to know that the user is logged in, and get fast access to their username and ID. This is achieved by using the Web Storage API of HTML5, i.e., *localStorage*, *setItem*, *getItem* and *removeItem* methods.

##### 3.4.2. Drag and drop

When uploading their avatar, the user can drag and drop the image inside the "Upload Image" card, and the dropped image will be selected and previewed. This is achieved using *onDrop* and *OnDragOver* properties of HTML5. Callbacks are implemented to handle the action when an item is dropped inside the card.

##### 3.4.3. Pattern matching

To perform the form validation in the front end, we use HTML patterns. The patterns are written as a regular expression. We use this in the forms for adding events, theme parks and attractions.

#### 3.5. Web Services

We chose to display the weather at the moment for each theme park on the map, because it is genuinely useful to know the weather in a theme park. It has a huge influence on which attractions you will be able to do, and how nice your day will be.

### 3.6. Publish Content

The server's API is RESTful. There is a one-to-one mapping between the HTTP methods and the CRUD operations. Modifying the attractions for example happens via a PUT request to `serverlink/attractions/:attractionID`.

User-generated data can thus be requested by third parties via a certain part of the API. This part has been made public and is documented using *apiDoc*, which is a tool for inline documenting RESTful API. Once the documentation is generated, the client can request them from the server and display them in the front end. The server has an API method to show the documentation, and since they are static and do not change that much overtime, the client will just request the page from the server that hosts the API documentation, instead of storing it in the client side of logic.

### 3.7. Mobility and Responsiveness

Bootstrap was used in the client side of the codebase, which helped a lot in making the website responsive. *React Bootstrap* was therefore used, which is a library that provides each bootstrap component as a React component, making it easier to integrate Bootstrap with the graphical interface.

Bootstrap offers a great, responsive, mobile-first grid system to divide pages and sections in rows and columns. The grid system was used in different parts of the project such as the input forms to add or modify an attraction or a theme park.

Bootstrap also offers a navigation bar, that we used as a basis for our navigation bar, customizing it a bit with additional CSS.

The figure consists of two side-by-side screenshots of a web application's 'Add a theme park' form. The left screenshot (Figure 12) is a desktop view showing a wide layout with multiple columns for address fields. The right screenshot (Figure 13) is a mobile view showing a narrow layout with fields stacked vertically. Both forms include fields for Name, Opening date, Address (Street, Number, Postal Code, Country), Type (Indoor/Outdoor), and Website link.

Figure 12 and 13: Adding a theme park from a PC (left) and a phone (right).

Additional media queries were also written to make sure the reviews section in the attraction's pages is displayed well on the different screens. On big screens (laptops, PCs) the reviews section will be placed to the left of the screen, which is more natural for the user. On small screens (e.g., phones), the reviews section will have around the same width as the phone screen width.

*React Photo Gallery* is used to display the gallery of the attractions. This library will adapt to the screen and change the size of the gallery and the number of displayed images per row or column according to the screen size.



Figure 14 and 15: An attraction gallery on a PC (left) and a phone (right).

### 3.8. Maps

We opted for Leaflet in combination with OpenStreetMap for our map. This choice was made based on the fact that Google Maps requires you to give your credit card number. Later on, we also saw it in the WPOs, but this had no influence on our choice, as it was already made.

The map focuses on the user's location when the user permits it. This is another HTML5 feature, namely the Geolocation API. The markers on the map represent theme parks, they are dynamically added when the theme parks are asynchronously fetched.

Whenever you click on the pop-up, you see the name, the link to the theme park and the weather at the moment. This seamlessly leads us to the next section.

### 3.9. Additional Technical Features

#### 3.10.1. Pagination

When requesting the results from the server, the client can add in its query which page it wants to have, and how many results (be it reviews, attractions, theme parks, events, ...) should be inside the page.

#### 3.10.2. Visibility of system status

Whenever the client sends a request to the server and is waiting for a response, the client will show some indicator to communicate with the user that a response is currently loading. This could be inside a button of a form for example, or in a small text that says the client is loading the information. This is achieved by using *React Promise Tracker*, which will track the Axios promises when there is one in progress (e.g., a form is submitted to the server, and the client is waiting for a response).

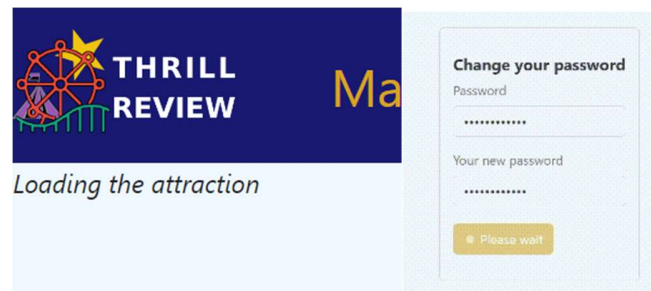


Figure 16 and 17: Examples of loading indicators in the project.

### 3.10.3. Handling images upload

The server uses *Multer* to handle image uploads. Multer is a middleware that is used for handling file upload to the server. Multer is set up so that it only accepts images of types png, jpg and jpeg which have size less than 8 megabytes. Each image will then have its unique URL that the client can access. For avatars, the link would be *serverlink/user/:userid/avatar*, :userid representing the ID of the user of which we are requesting the avatar.

For attraction images, the link would be *serverlink/attractions/:attractionID/photos/id=photoid*, :attractionid representing the ID of the attraction, and *photoid* representing the ID of the photo we are searching for. Additional functionalities are implemented to help the client know how many photos an attraction has.

### 3.10.4. deployment of the site

Our site is deployed on [www.thrillreview.com](http://www.thrillreview.com)

The front end of the site is hosted via a virtual static host using Nginx. SSL certificates are generated using Let's Encrypt and certbot, all HTTP requests are redirected to an HTTPS request so all requests are secure.

The backend is hosted via a reverse proxy using Nginx. All requests to [api.thrillreview.com](http://api.thrillreview.com) are proxied to the loopback address of the server with the port on which the NodeJS backend server is running. These requests are also using the HTTPS protocol on port 443.

## 4. Starting the site locally

The front end of the site is running on <http://localhost:3000>, and the backend is on <http://localhost:5001>.

We used yarn as a package manager so starting the site is easily. First, you need to install yarn on your computer. Afterward, you use your terminal to start it.

To start the front end: go to `./thrillreview/client` and run `'yarn install'` followed by `'yarn start'`. This will start the front end.

To start the backend: go to `./thrillreview/client` and run `'yarn install'` followed by `'yarn start'`. This will start the backend.

## 5. The Source Code

The source code of our project can be found in our Github repository, in the *deploy* branch. The Github repository can be found in the following link: <https://github.com/matsvanmolle/thrillreview>

## Appendix: Timetable

Here we show who worked on what, and when/ how long we worked.

### Abdullah

Date	Hours	Description
10/13/2022	1.5	First meeting, Final decision on used language and frameworks
10/14/2022	6	Getting started with Typescript, nodejs, expressjs and react
10/16/2022	1	Setting up the server/client architecture and the environments
10/21/2022	3	User management system that allows users to register/log in
10/22/2022	3	additional security features added to the user management system
10/30/2022	1	additional improvements to the backend of the user management system
10/30/2022	2	integrating the user management system with the front end
10/30/2022	5	Implemented the front end of the user management system (login/register/profile pages)
10/31/2022	2	Made the website logo and updated the website to have the new logo
11/1/2022	2	Added the possibility for the user to change their information (backend side implementation)
11/1/2022	2	Added the possibility for the user to change their information (frontend side implementation)
11/6/2022	2	improved login screen, advanced form validation (e.g., uniqueness of username)
11/7/2022	1	improved register screen, advanced form validation in the register screen
11/7/2022	1	added advanced form validation to all user management forms
11/13/2022	1	added indicator in all user management forms to indicate the client is waiting for a response from the server
11/13/2022	2	started implementing the attraction page (frontend and backend)
11/18/2022	1	figuring out how to upload/retrieve images from the server
11/18/2022	2	users can now set an avatar for their profile (backend)
11/18/2022	1	users can now set an avatar for their profile (frontend - first iteration)
11/19/2022	2	users can now set an avatar for their profile (frontend/backend)
11/19/2022	2	users can update their avatar (frontend/backend)
11/19/2022	2	improvements in usermanagement (frontend)
11/20/2022	1	first iteration of attraction page (frontend)
11/20/2022	1	first iteration of reviews (front/backend)
11/21/2022	2	reviews can now be shown in every attraction page (frontend)
11/21/2022	1	users can now add ratings to their reviews
11/23/2022	2	made existing server functionalities RESTful. updated the front end to handle the new changes

11/23/2022	2	fixed a number of major bugs in the front and backend (e.g. tokens not updated after changing information, credentials not asked in some forms)
11/25/2022	2	researched pagination + first stage of server-sided reviews pagination
11/26/2022	1	improved the reviews pagination and fixed some bugs
11/26/2022	2	front-end reviews pagination
11/26/2022	2	edit attraction information (not finished)
11/27/2022	2	edit attraction information (now is showing as a modal, frontend)
12/4/2022	1	added average rating of an attraction (frontend/backend)
12/4/2022	1	finished the frontend of editing an attraction
12/4/2022	2	added the page of the themepark (frontend/backend)
12/4/2022	1	themeparks can be edited (frontend)
12/4/2022	1	themeparks and attractions can be modified (backend)
12/5/2022	1	updated the old functions of modifying information to work with the normalised tables
12/5/2022	1	finalisation of modifying the attractions/themeparks
12/5/2022	1	first iteration of feeds (backend)
12/5/2022	1	first iteration of feeds (frontend)
12/6/2022	2	finalisation of feeds (front/backend)
12/14/2022	1	viewing attraction images (backend)
12/14/2022	2	viewing attraction images (frontend)
12/14/2022	1	modifying attraction images (front/backend)
12/15/2022	1	bugfixes when retrieving images
12/16/2022	3	additional bugfixes to forums. Added possibility to sort reviews by rating/date ascending/descending (backend)
12/16/2022	1	sorting reviews by rating/date ascending/descending (frontend)
12/18/2022	5	Researchd documenting APIs. Made api documentations
12/18/2022	1	bugfixes when modifying attractions/themeparks
12/19/2022	2	various bugfixes regarding images, adding attractions, modifying attractions
12/20/2022	3	bugfixes when adding or modifying attractions
12/21/2022	2	bugfixes when adding attractions
12/23/2022	1	bugfixes when modifying an attraction (frontend requires users to log in first)
12/23/2022	3	finalizing the API documentation. Improved the error responses in some parts of the server.

## Mats

Date	Hours	Description
13/10/2022	1.5	First meeting, Final decision on used language and frameworks
14/10/2022	6	Getting started with Typescript, nodejs, expressjs
04/11/2022	3	Add attraction backend
06/11/2022	4	setting up the server for our intermidate presentation
05/12/2022	2	add themepark backend
06/12/2022	4	normalisation of the database
05/12/2022	1	logout, naive version with just removing the token
12/12/2022	4	logout with refresh tokens



13/12/2022	2	giving back the themeparks for pinpoints on the map
13/12/2022	3	backend for adding images to an attraction
13/12/2022	2	search for attractions using pagination
13/12/2022	1	search for themeparks using pagination
14/12/2022	4	whole event backend, adding events returning events given an id, joining events, nr of attendees, search for events etc
16/12/2022	3	placing command in the code
17/12/2022	6	deployment of the server
18/12/2022	1.5	SSL certification for the server
20/12/2022	3	changed the search so that its based on multiple fields
20/12/2022	2	stored the id of the themepark in db instead of just the name + change of code to work with it
20/12/2022	3	searching for bug that let the side crash
21/12/2022	1	fixing the bug
21/12/2022	3	fixing small bugs that come with deployment of the site
22/12/2022	2	documenting the api

#### Jolien

Date	Hours	Description
13/10/2022	1.5	first meeting; final decision of frameworks
18/10/2022	2.5	installing everything; tutorial expressJS
20/10/2022	2.45	react & typescript tutorials; figuring out how to show something on screen through react/yarn
23/10/2022	1.45	working out how to work with bootstrap; start navigation bar
24/10/2022	2	work further on navigation bar; make navigation bar responsive
31/10/2022	4	research google maps & leaflet/openstreetmap. Make map fill whole screen (navbar still visible, no scrollbar)
02/11/2022	2.45	basic layout of attraction homepage (backend not yet implemented; thus not connected)
03/11/2022	2	basic layout to add an attraction (making the form: no validation, no connection to backend, no keeping state)
05/11/2022	2	link between attractions - add attraction; trying to understand formik (no luck)
06/11/2022	2	make add-attractionform grid-like + client side validation
07/11/2022	2.5	map starts on location + change attractionform/try to make types ok (no luck)
11/11/2022	1.45	form validation in html5 (with patterns/regular expressions) for add attraction + navbar collapse on click working
13/11/2022	3.15	basic theme park page (with form validation) + navbar changes
20/11/2022	0.15	add theme park fields toevoegen
21/11/2022	5	navbar changes + button colour changes, request to nominatim for lat/long coordinates of address
22/11/2022	2	show location on map with marker + map overflow problem + css for add attraction/theme park

27/11/2022	2.45	some css, looking into adding an image, trying nominatim user agent, researching footer possibilities
04/12/2022	7	frontend searching theme parks, problem with pagination, tried to solve it, did not work.
11/12/2022	4.5	events : add event, browse event, event home page
11/12/2022	2	weather api for pop up on map ; pop up on map further
14/12/2022	8	browse attractions [infinite scrolling] & pinpoints made from async data: neither worked
15/12/2022	3	browse attractions [infinite scrolling] & pinpoints made from async data: now worked
16/12/2022	10.5	browse theme park, browse events, map
17/12/2022	9	browsing cards refactoring, event browsing, attractions top 10
18/12/2022	7	connect attraction to theme park (with infinite scrolling), connect theme park to event, login first card
19/12/2022	6	random attraction at the start; login knop overal; event joining
20/12/2022	3	problem with attractions -- turned out to be lots of pending requests
21/12/2022	8	events in profile, clean up