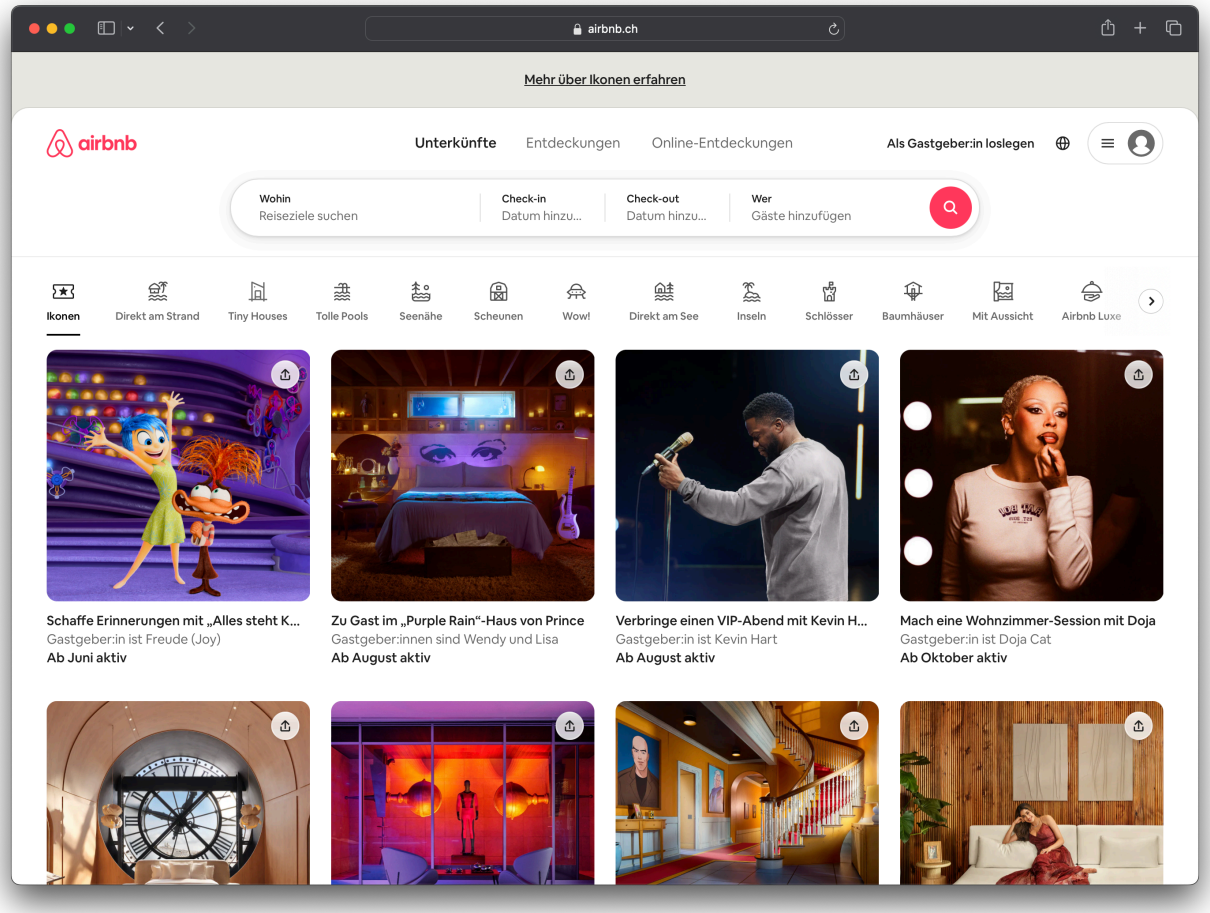
Project Documentation

WINF.WBDG.FS2024 - ARLIND GURGUROVCI

Introduction
For this module, we were required to create a functioning website using CSS, JavaScript, HTML and the Bootstrap framework. The aim of the website is, to book tables via a user interface. We needed to complete tasks, by implementing it in to our websites.

Project Planning
To approach this project, I first thought how I would like to have the website done for me, and how I would wish to use it, so I got a good idea how this website should look. I approached the project by considering how I would use the website from start to finish. Instead of first creating a basic layout and then building on it, I envisioned the overall structure and the features that would be included. I tackled the tasks progressively, guided by the logic of user experience, which proved to be somewhat effective.

Basic Structure
First, I had to consider how the desks would be displayed. My goal was to provide the user with as much necessary information as possible with the fewest interactions. Based on my experience, the less effort required from the user, the better. This was the underlying thought process. Then, I thought of the Airbnb website and liked the layout of displaying all Airbnb listings at once, so I wanted to build on that idea.



Sketches

Frontend Sketch

These are sketches of the user interactions and their appearance. The section is divided into multiple UIs.

Navigation Bar

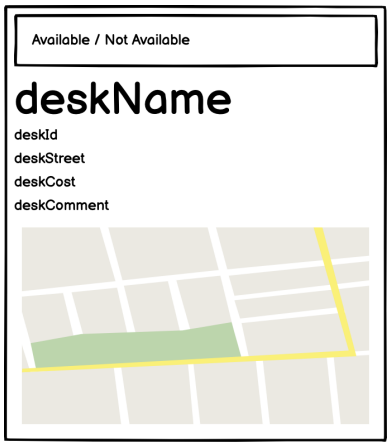
The navigation bar consists of elements that are aligned on a grid. The user should have access to the following three elements on the navigation bar:

Footer

The sketch is very simple and will only include the copyright text.

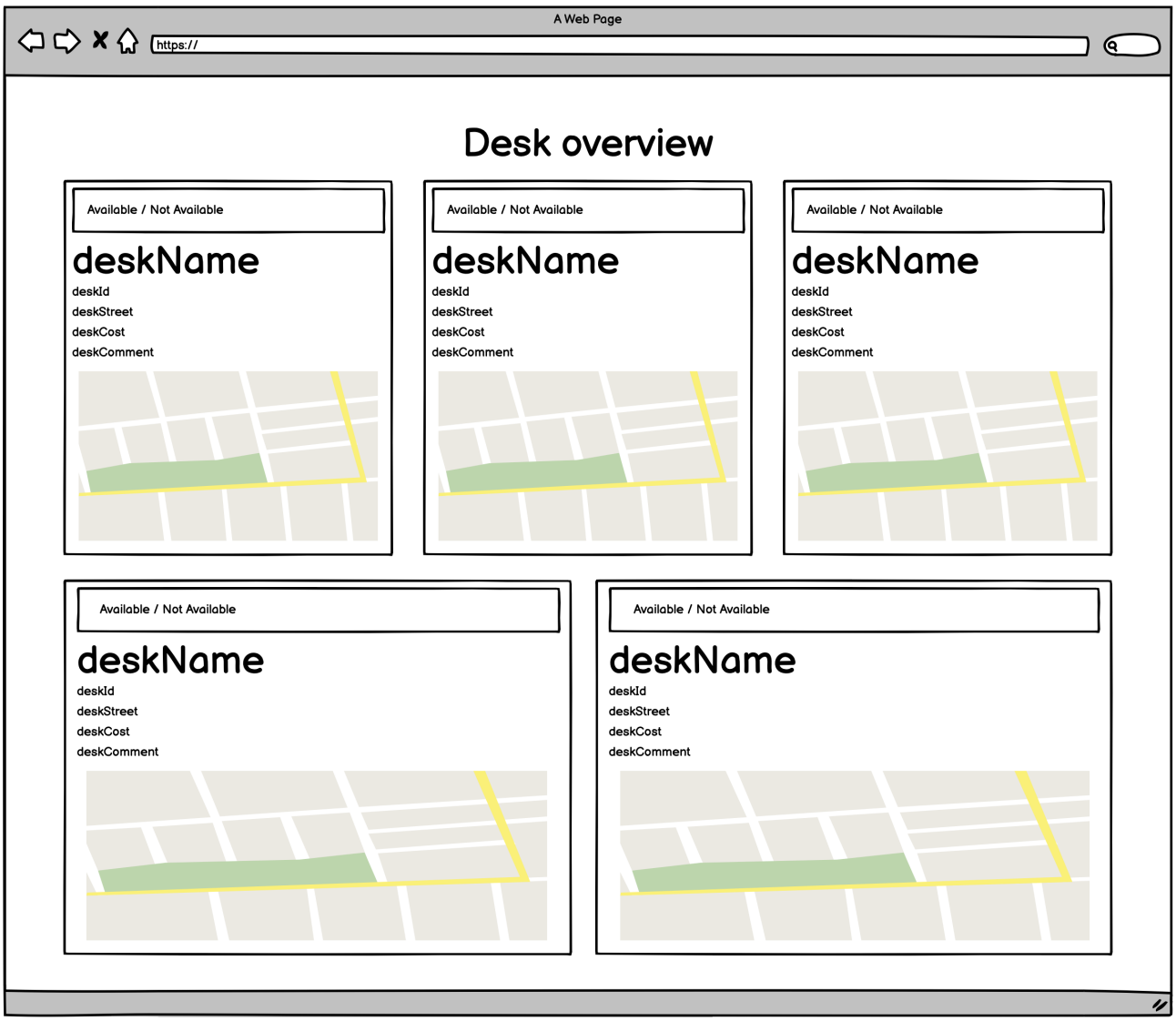
Desk Card

Each desk should be displayed as a card, allowing the user to see all the information in one view without needing to click around to find more details. The desk card will contain the information the user needs to preselect a desk.

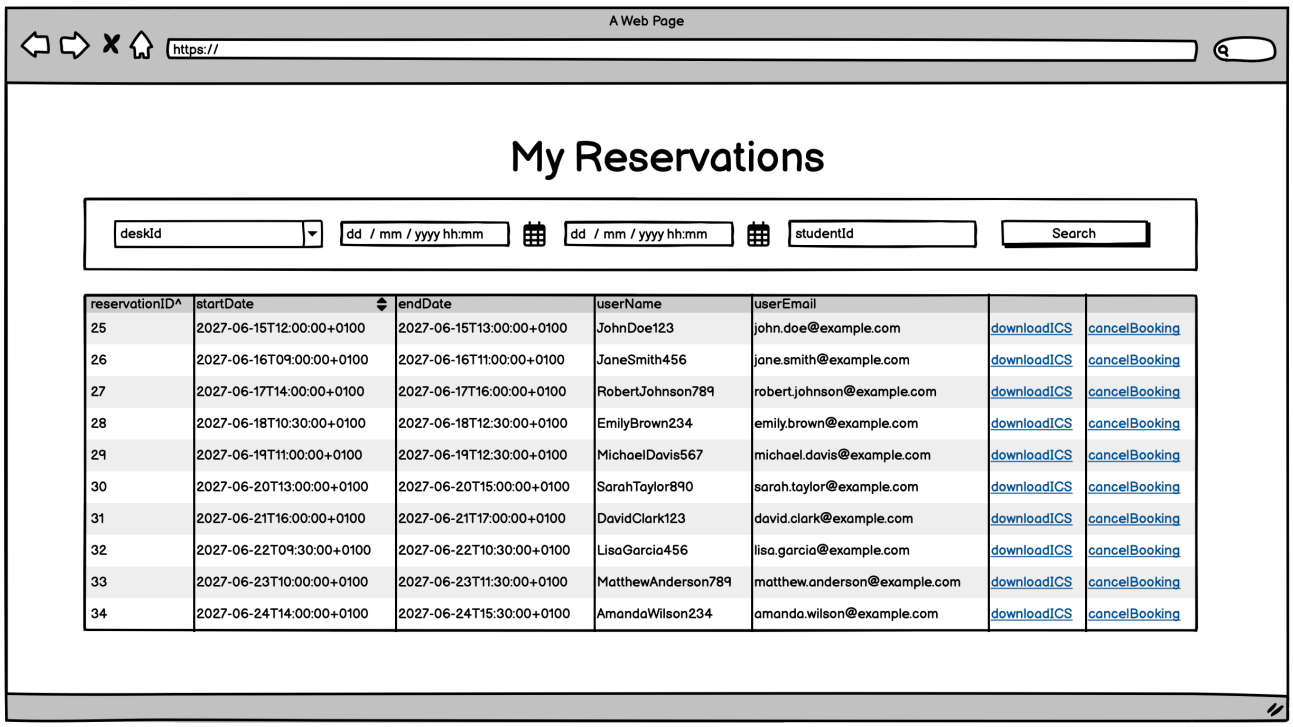


Overview

The overview consists of a grid-like style. The desk cards are aligned in three columns when there are enough elements; if not, the columns will adjust dynamically.

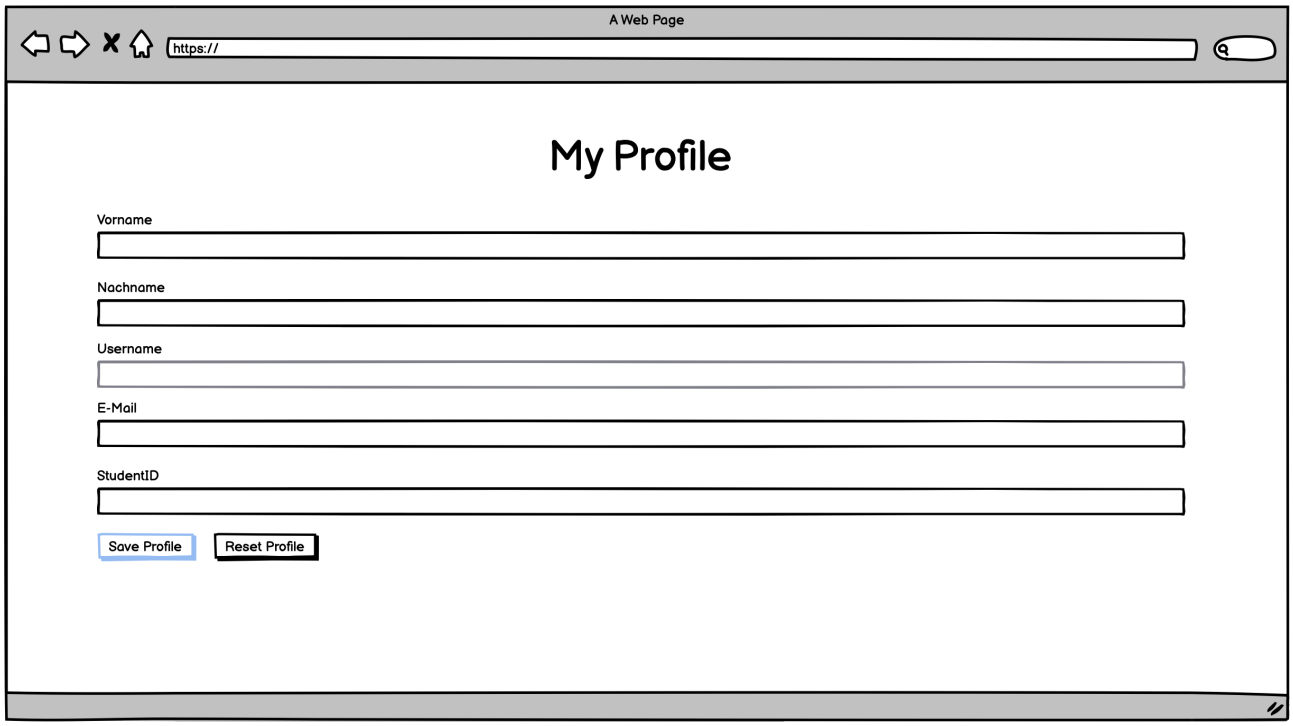


My Reservations



The my reservations page is an overview of the reservations the user made. He'll see every reservation per desk, the timeframe and some additional information. He can always redownload the ICS file for the calendar and can cancel bookings at any time.

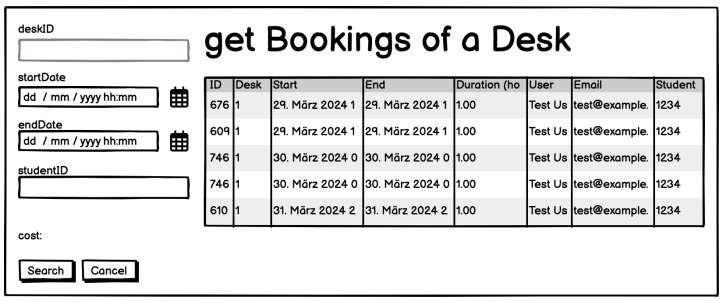
My Profile

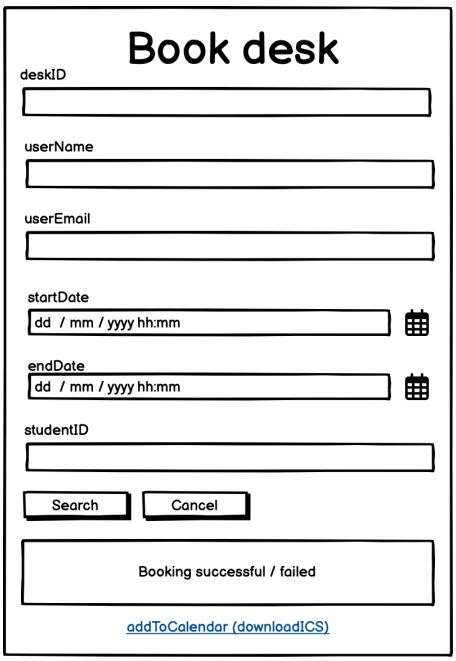


The "My Profile" page provides an overview of the reservations the user has made. It displays each reservation per desk, the timeframe, and some additional information. The user can always redownload the ICS file for the calendar and cancel bookings at any time.

Forms

To accept and process user input, we need to implement forms on the website. There are only two forms available for users to book a desk (besides the “My Profile” page).





Features and Logic

By definition, we had some tasks to complete, and I solved them using a sketch. Sketching is an important step in organizing thoughts and seeing how features interact with each other, helping us avoid double work and adjustments by adding each feature sequentially.

Overview

- Show the user all desks when they open the webpage.
- Make all cards clickable, so the user can comfortably select the desk they want to book.

Booking the desk

- Automatically transfer the clicked desk card's deskId into the form, so the user doesn’t need to make unnecessary interactions.
- Fill out the startDate and endDate from local datetime and round the minutes to the next 30 minutes.
- If the user has a profile, the necessary information should be automatically filled from the locally saved profile; if not, it should leave them empty for the user.
- Display all bookings of that specific desk in a table format, easily readable by the user, so they can pick a time and refill the form.
- Display the cost of booking that desk in EUR or CHF using the deskCost and the selected hours.
- After all conditions are met, open the new booking form to actually book the desk. The deskId, startDate, and endDate should be transferred to the next form, so the user doesn’t need to make unnecessary interactions.
- After successfully booking the desk, let the user download the ICS immediately and provide understandable feedback if the booking was successful or failed.

My Reservations

- When the user visits the page, it should fill out all inputs that can be populated from the profile.
- Set the startDate to the minimum (01.01.0000) and the endDate to the maximum (31.12.9999) to show all bookings.
- Display all bookings in a table format, with two buttons in the table:
  - Cancel the booking: The user can cancel their bookings with one click.
  - Redownload the ICS: The user can recreate a new ICS link for their booking.

My Profile

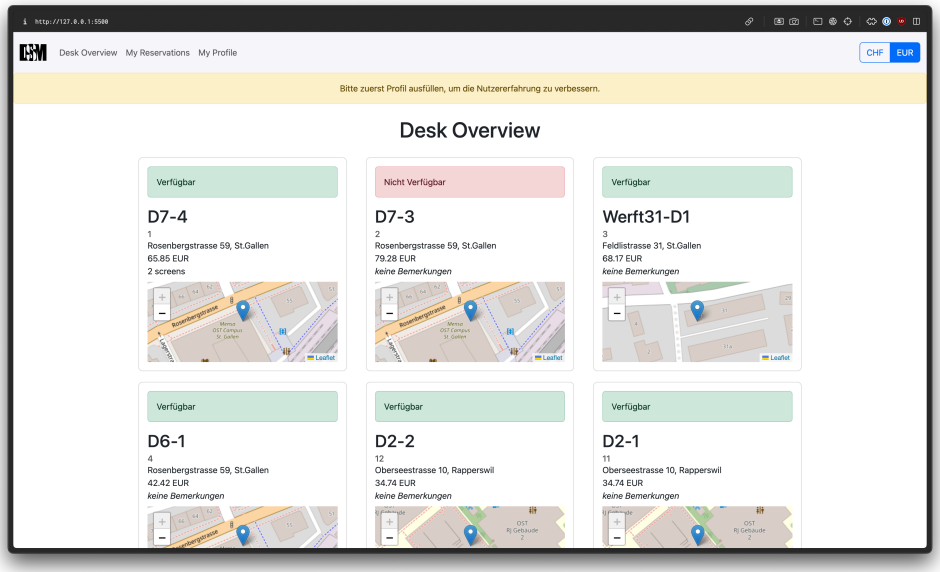
Whenever the user visits this website, it should fill out their information automatically if they have a profile saved locally; if not, the input fields should be empty. The user can save the profile by clicking the save button, and reset their profile by clicking the reset button.

Additional functions

Some inputs have validations to check if they are an email address, a number, or only characters, applied whenever it makes sense to do so. Some inputs are disabled to prevent the user from accidentally typing in incorrect information or information that the API can’t handle.

Display a non-distracting alert if the user hasn’t filled out their profile. This alert is not intended to disturb or force the user to fill in their profile; it only suggests doing so to improve their experience.

Implementation
Desk Overview

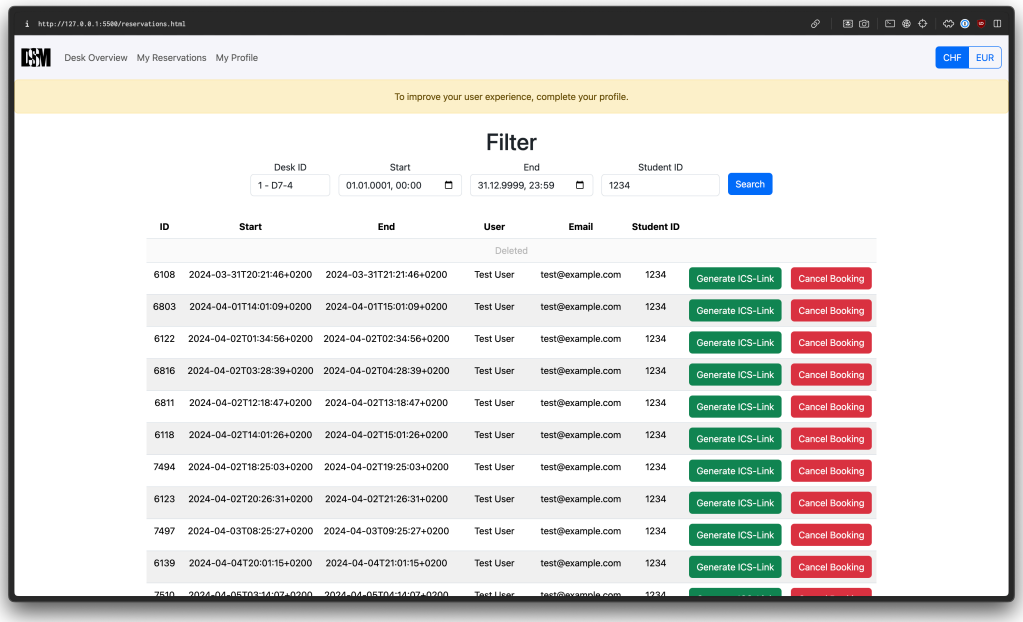


The basic structure of the overview homepage is pretty simple. It consists of the navigation bar, a dynamic list item that adds the desk cards via JavaScript, and the footer. The list starts empty, and the JavaScript in the backend dynamically adds the desks, forming the basic structure of the card HTML. It loops through the entire response JSON, creating a new card for each object and adding it to the list.

The following JavaScript code implements each desk dynamically in to the HTML. It creates a deskElement and fills it out with information and more HTML and adds it to the homepage.

```
    <div class="card-body">
    <div class="availability-indicator alert ${availabilityClass}" role="alert">${isAvailable ? 'Available' : 'Not Available'}</div>
        <h2 class="card-title">${desk.name}</h2>
        <h6 class="card-subtitle text-body-secondary" id=${desk.id}>${desk.id}</h6>
        <div class="col">${desk.address}</div>
        <div class="col price">${desk.price} CHF</div>
        <div class="col">${desk.comment || '<i>no comments</i>'}</div>
        <div class="desk-map pt-2 mb-2" style="height: 150px;"></div>
    </div>
`;
```

My Reservations



While designing and programming the logic of the "My Reservations" page, I encountered several problems that required solutions:
- Make the page easily understandable and usable for the user.
- Make the filtering option very intuitive, standardizing it to show all bookings of a desk.
- Ensure canceling a booking is very easy and can be done with one click.
- Allow the user to redownload an ICS file whenever needed.

I decided to create a simple table with all the information the user needs to see. Each row includes two buttons: one to cancel the booking and another to redownload the ICS file. To prevent invalid input, we have input validation for the filter. The filter fetches all available desks along with their deskId and deskName. Since users can't easily memorize which ID corresponds to which desk, I included both the deskId and deskName in the dropdown. However, for processing, only the deskId is used as the value.

```
function fetchDesks() {
    fetch('https://matthiasbaldauf.com/wbdg24/desks')
        .then(response => response.json())
        .then(desks => {
            const dropdown = document.getElementById('deskIdDropdown');
            desks.forEach(desk => {
                const option = document.createElement('option');
                option.value = desk.id;
                option.text = desk.id +" - " +desk.name;
                dropdown.appendChild(option);
            });
        })
        .catch(error => console.error('Error when loading the desks:', error));
}
```

We utilize the API to fetch all available desks and set the option.value to the deskId, but the text shown on the frontend includes both the deskId and the deskName. For canceling bookings and creating the ICS, we use the `.closest` function in JavaScript to gather the necessary information from the nearest `tr` element.

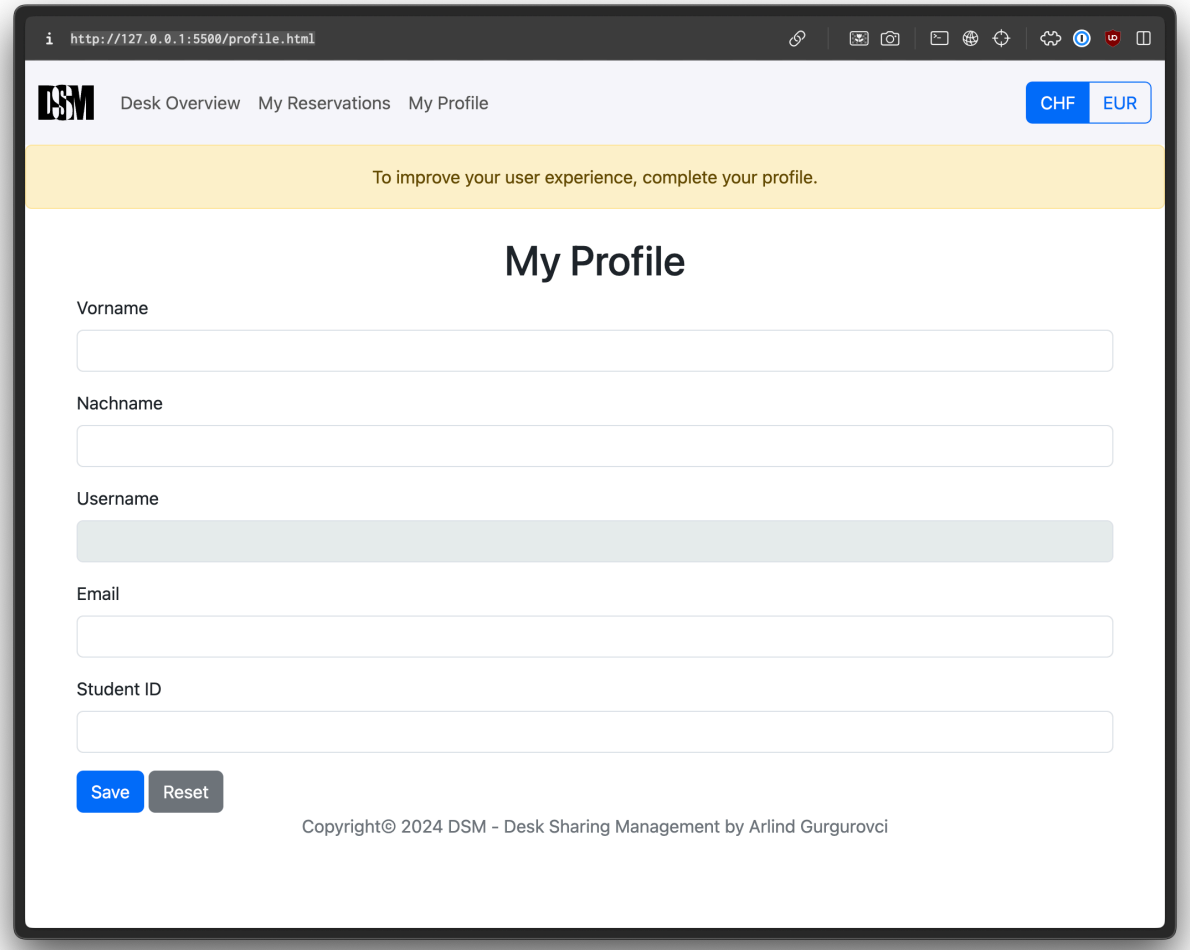
```
const currentRow = event.target.closest('tr'); // find the nearest element where the clicked button is included

const deskId = document.getElementById('deskIdDropdown').value;
const startDate = currentRow.querySelector('#bookingStart').textContent;
const endDate = currentRow.querySelector('#bookingEnd').textContent;
```

My Profile

The task was to automatically save user input for future reservations. Initially, I tried to create a logic to monitor user inputs and save any information that could be useful for future reservations. However, this approach proved to be too complex and ultimately failed.

Instead, I came up with a simpler solution: prompt the user to fill out their profile on the webpage for a better experience. The user can complete their profile, and all the form inputs will be pre-loaded on the pages that require the provided information. This ensures a better user experience, making it more convenient for the user to only input variable values while the static ones (name, first name, email, student ID) are automatically filled in by the script.

In my opinion, this is a more elegant solution than passively monitoring the user and saving information in the background to surprise them with pre-filled forms. The user can opt out of this feature simply by not filling in their profile.

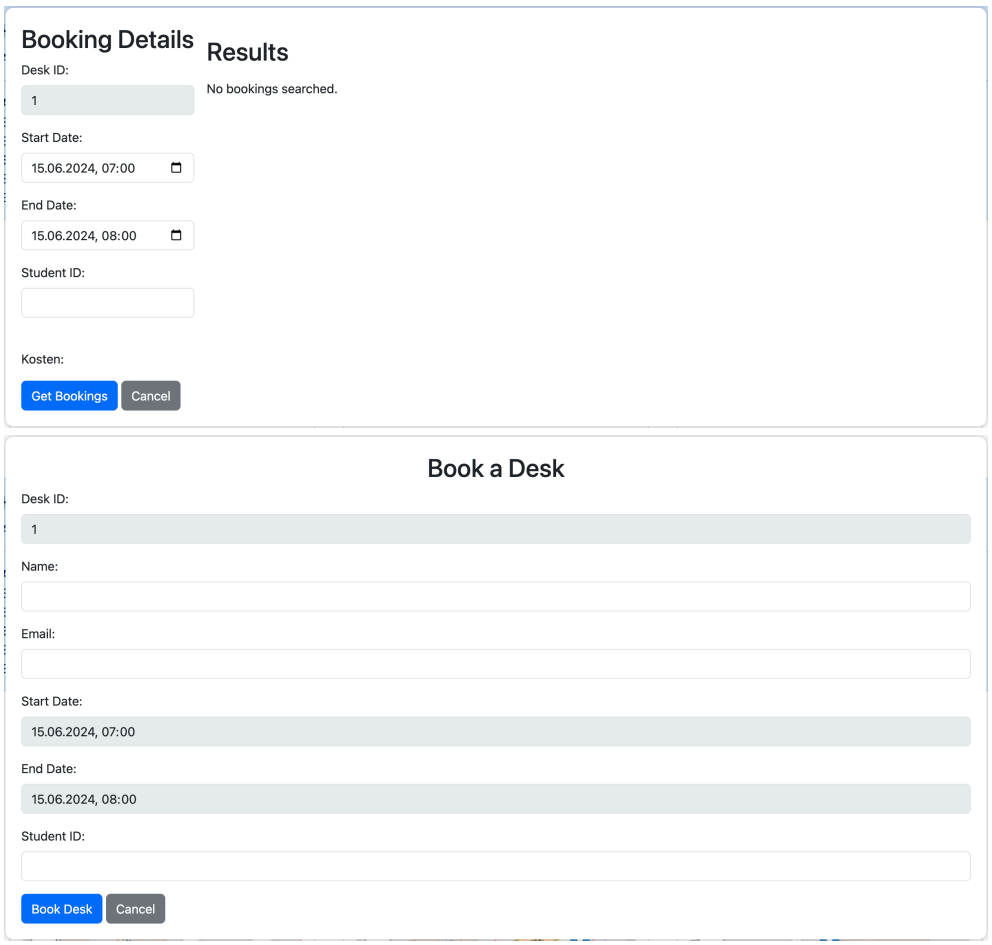


Information about the profile can be dynamically loaded whenever needed on a homepage. For example in the My Reservations page:

```
function loadStudentIdFromProfile() {
    const profile = JSON.parse(localStorage.getItem('profile'));
    if (profile && profile.studentId) {
        document.getElementById('studentId').value = profile.studentId;
    }
}
```

Forms

Both forms have been implemented and designed to look very pleasant to the user and are not overly complicated.



Problems and their solutions

Input Validator

As for the task given, we need to validate all inputs from the user to prevent them from typing numbers in their name or characters in number inputs. This validation is done via JavaScript. Rather than adding separate JavaScript code to each text input on the webpage, I created a function that checks the input based on its classes. This allows me to easily add more text inputs on the go; I just need to add the class to the function, and it will work.

```
function validateNumberInput() {
    const numberInputs = document.querySelectorAll('input[data-validate="number"], input[id="bookStudentId"], input[id="studentId"]');
    const numberPattern = /^[0-9]*$/;

    numberInputs.forEach(input => {
        const errorElement = document.getElementById(input.id + 'Error');
        if (!numberPattern.test(input.value)) {
            input.classList.add('is-invalid');
            if (errorElement) {
                errorElement.style.display = 'block';
            }
        } else {
            input.classList.remove('is-invalid');
            if (errorElement) {
                errorElement.style.display = 'none';
            }
        }
    });
}
```

This is just an example of checking if the user inputs only numbers and no characters. It uses regular expressions to validate the input. If the user's input doesn't match the regular expression, it adds a CSS class to highlight that something is wrong. If the input is correct, it removes the CSS class.

Ensure Seconds

While programming the logic behind everything and using the API, I encountered a problem where the API would throw an error while handling my request. This was because of an oversight I made while reading the API documentation. The date input was using the format `YYYY-MM-DDTHH:MM`, which I thought was correct, but upon reviewing the format required to successfully make an API call, I noticed that it also needs seconds to process correctly. So, the function `ensureSeconds()` was born. This simple function ensures that the startDate and endDate have seconds in their value, so they can be processed correctly by the API. The function simply checks if the value given is 16 characters long; if not, it adds `:00` at the end. If it's already 16 characters, it will just return the value.

```
function ensureSeconds(datetime) {
    if (datetime.length === 16) { // check if the value is 16 chars long.
        return `${datetime}:00`; // if condition met, add :00 to the string and return it.
    }
    return datetime; // if not, the format is correct, return the value.
}
```

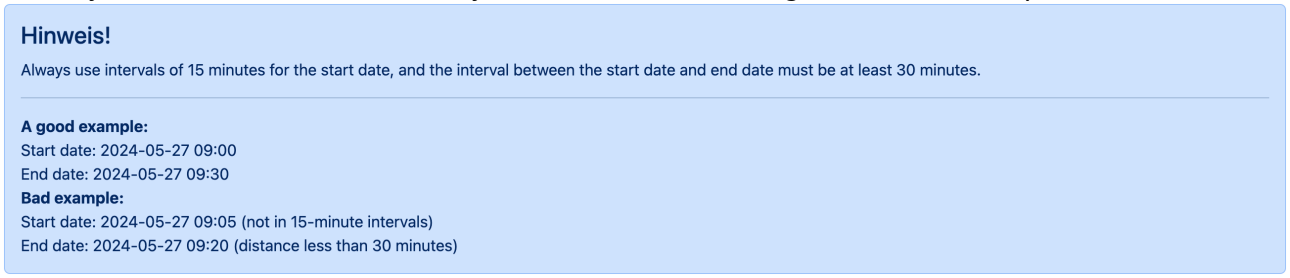
Restrict users control over inputs

Throughout the website, the user can input values into forms. I decided that some inputs need to be disabled and filled out by JavaScript to prevent the user from making mistakes and inadvertently having a poor experience. For example, the user might not know what to put in as the deskId or what username to provide for the API to process their request. That's why some inputs on the forms are disabled and automatically filled out by JavaScript. This approach is used on the getBookings form and the bookDesk form, as well as on the my profile page, where the username is disabled and generated based on the user's first and last name.

Ensure startDate and endDate are in 15-minute Intervals

I faced a new problem: the user can input a startDate and endDate whenever they want, allowing them to book a table from, for example, 15:03:00 to 15:05:00. The issue is that the startDate should not be such a specific time, and the endDate should be a sufficient duration from the startDate. I tried to restrict the user from entering such values but was unsuccessful. So, I came up with a less ideal solution: I created an alert that asks the user to follow these rules.

However, this is not the best solution since it still allows the user to make invalid inputs. Unfortunately, I have not been able to find a restrictive solution that works without breaking much of the code. On the other hand, considering this is an internal desk booking tool, someone might actually need to book a desk for only 5-10 minutes for an urgent call or other quick tasks.



Version Control with GitHub

During the development of the project, I encountered many problems. Often, changing one part of the code inadvertently broke other parts. Fortunately, I was able to restore much of the code to its state before integrating the new changes. To prevent further issues, I decided to use GitHub for version control. Now, for each new feature implementation, I create a new branch and only merge it when I'm satisfied with the results.

Modular Programming

While programming, I noticed that I had all JavaScript functions and features in a single file. Over time, this file grew to several hundred lines, making it difficult to maintain an overview. Searching for a specific function became very time-consuming.
I decided to modularize my JavaScript files and split them into separate files:

- bookDesk.js: Responsible for all scripts related to the function of booking a desk.
- bookingsDesk.js: Responsible for all scripts related to get all bookings of a desk.
- currency.js: Responsible for currency changes on the homepage.
- filterDesk.js: Filters desks on the "My Reservations" page.
- inputValidator.js: Validates user inputs to prevent invalid entries.
- overviewDesk.js: Provides an overview display of all desks.
- profile.js: Handles all functions for users to create, load, save, and edit profiles.
- misc.js: Contains all other JavaScript scripts that could not be categorized elsewhere.

Reflection

In this project, I learned a lot of new things. I realized the importance of planning. Taking the time to think about how to approach a problem and how to solve specific tasks is crucial. Simply jumping into programming and adjusting as you go can lead to wasted time. I made this exact mistake and lost valuable time until I decided to plan everything first and then start programming. This improved my efficiency and gave me a clearer overview of the project, allowing me to tackle problems more effectively and easily.

In addition to the challenges I faced, it was a valuable experience to learn about and use the tools for development. I was able to try out essential tools, such as GitHub, which are standard in today's industry. I also saw the advantages these tools bring to the development process.

I learned that it is challenging to teach a computer what you actually want. I had to learn to think logically and sequentially, and also to establish connections to other functions in the program to provide interfaces (for example, My Profile).

Overall, the project was a great experience. It was special for me to turn an idea into something real that works according to my wishes and expectations. I hope we can have more projects like this in the future.

Sources and Tools used

Tool	Link	Comment
Youtube	YouTube.com	Lookup for coding tutorials
ChatGPT	chat.openai.com	Refactoring code, getting basic structure of a webpage, helping coding, explanation of code, explanation of error messages, bug fixing
Github Copilot	github.com/features/copilot	Refactoring code, translation from German text in to English text in visual studio
Visual Studio Code	code.visualstudio.com	My IDE of choice, using to setup the whole project
LiveServer (VS Code Plugin)	marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer	For setting up a local server to test and develop the project
Google	google.com	Looking up code snippets, documentation of apis used in the project
calndr.link	calndr.link/api-docs	Documentation of the calendar ICS api
Exchangerate-API	www.exchangerate-api.com/docs/free	Documentation of the exchangerate api
Postman	https://www.postman.com/downloads/	API testing.