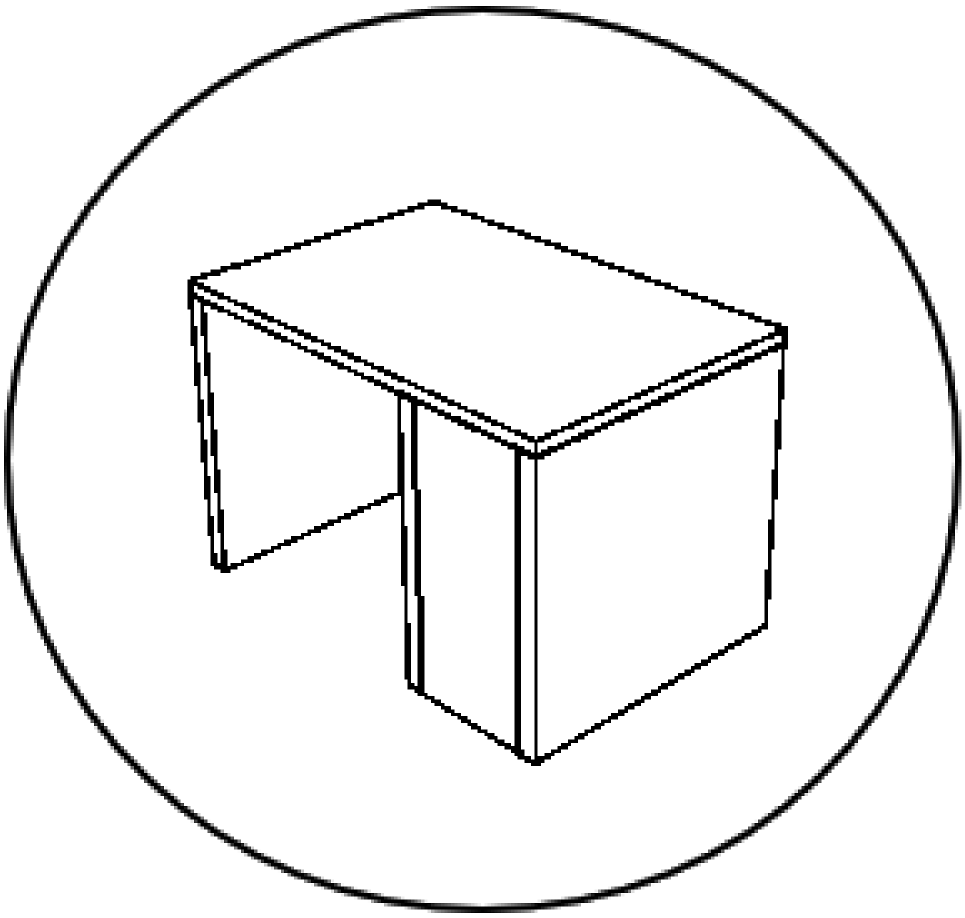
Project Documentation - Desk Sharing Management

WBDG: Web Development Fundamentals (FS 2023)



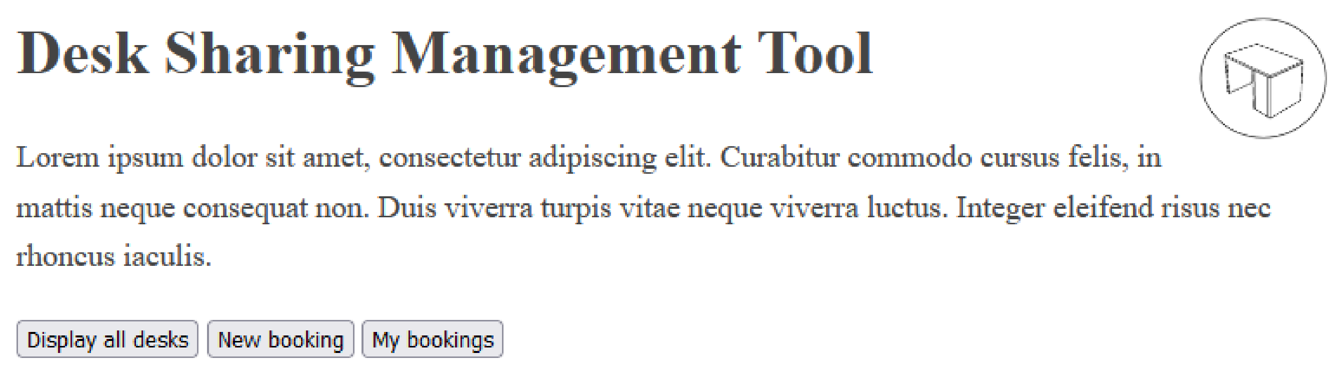
Created by: Noah Weishaupt, BB20 WINF

Task 1: Basic Structure

Basic Program Architecture

As described in the task description, I've created a basic file structure consisting of the `index.html` file, a subdirectory for images, and a placeholder `scripts.js` and `style.css` file. I also created a `docs` folder containing my notes for each task.

The basic layout of the `index.html` file was generated using a command in Visual Studio Code. After generating the basic structure, I added links to the `style.css` and `scripts.js` files. I also created a simple logo with a public domain clipart.


Desk Sharing Management Tool

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur commodo cursus felis, in mattis neque consequat non. Duis viverra turpis vitae neque viverra luctus. Integer eleifend risus nec rhoncus iaculis.

Image 1: First version of the website

I already made a small mistake at this step: When embedding the `scripts.js` file into `index.html`, I added the script tag into the header of the file. This was later a problem, as the `scripts.js` file was loaded before the DOM-Elements were created, because of which I wasn't able to add eventlisteners to the buttons.

The finished file structure can be found in this commit on Github.

Design Sketches

As designing user interfaces is not my strong point, I tried to keep the interface as simple and practical as possible. The tool consists of a header containing the page title and a main body consisting of a table containing all the relevant data about the available desks. Located below the table are the buttons, with which the user can interact with the tool by creating and cancelling bookings.

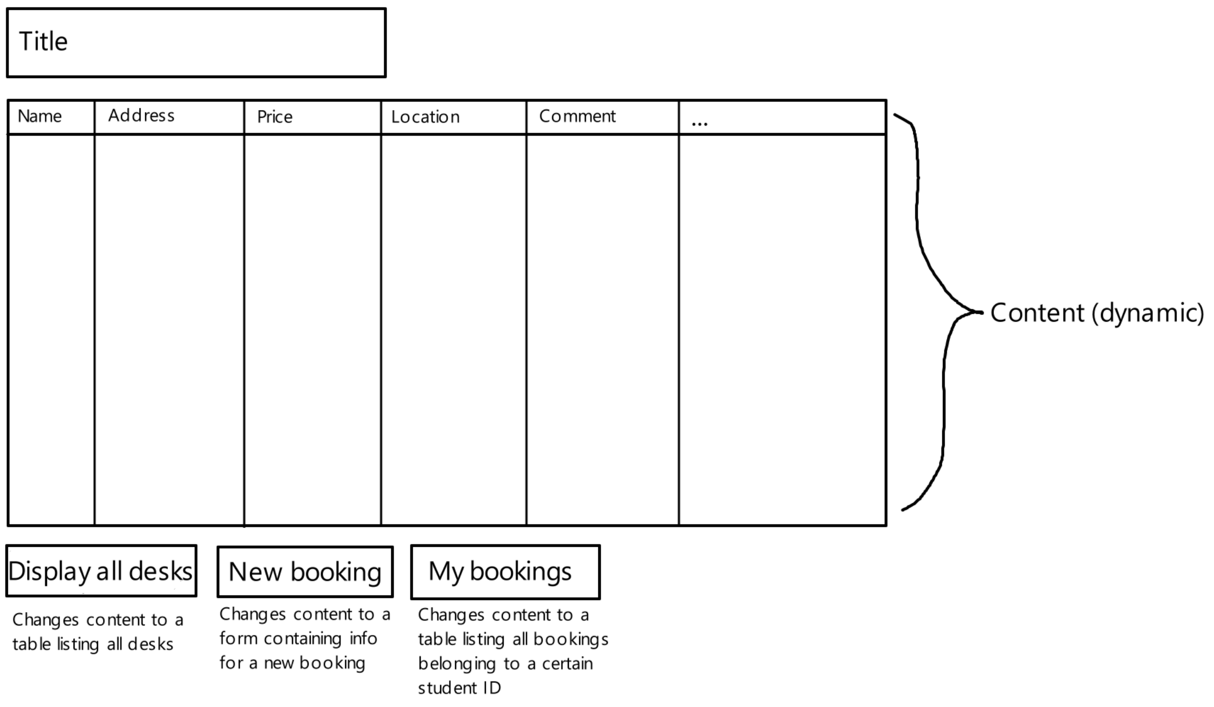

Title
Name
Address
Price
Location
Comment
...
Content (dynamic)
Display all desks
Changes content to a table listing all desks
New booking
Changes content to a form containing info for a new booking
My bookings
Changes content to a table listing all bookings belonging to a certain student ID

Image 2: Sketch of the desktop version

The mobile version is the same as the desktop version, with the only difference being that linebreaks are added to the text within the cell if the window gets smaller. Originally I inteded for the table to collapse if the window got too small, but since the table already looked good enough on my phone (after zooming out a bit) that I didn't think that it was worth the effort of implementing it.

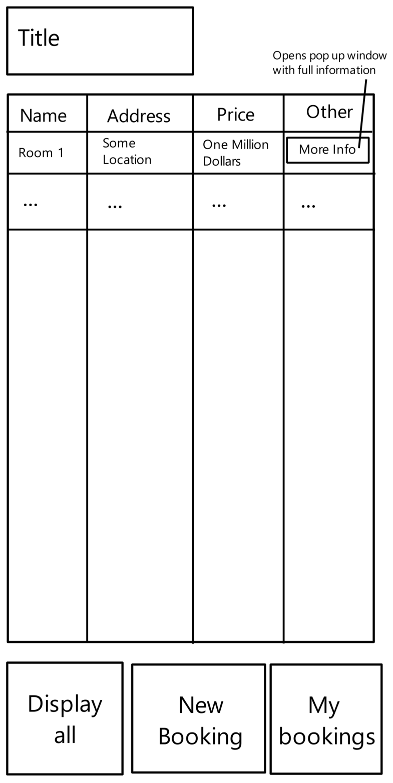


Image 3: Sketch of the mobile version

Task 2: Dynamic Elements

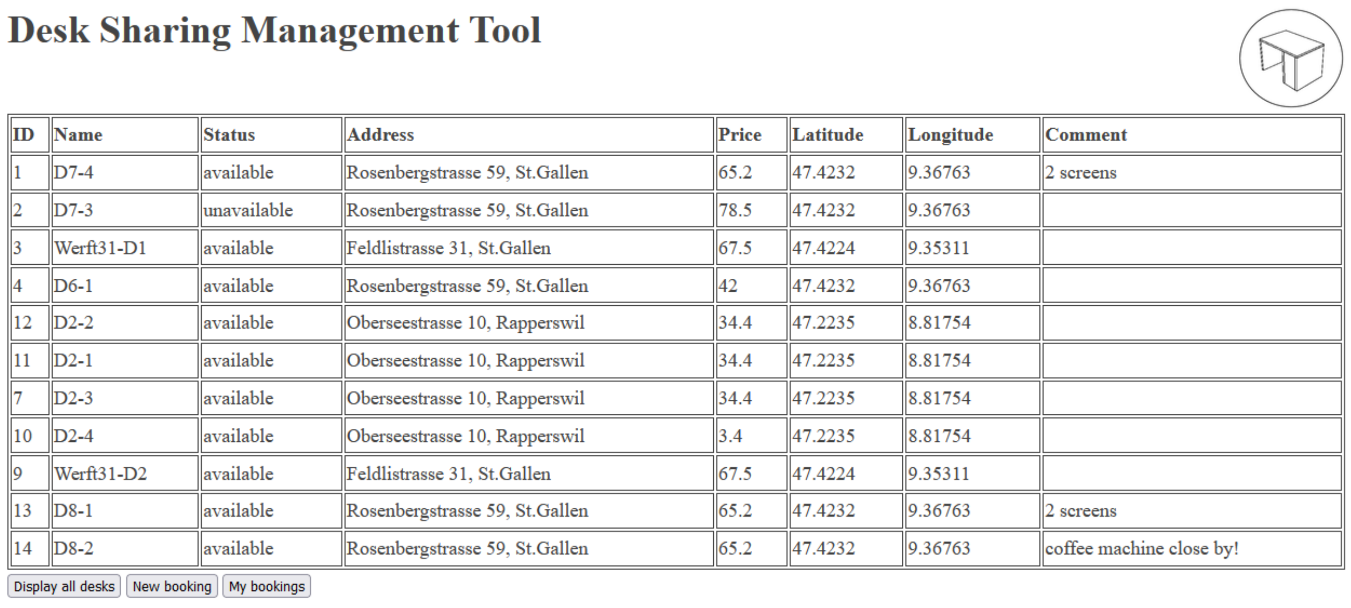
Retrieving the Data

Retrieving the data was quite simple. I create a variable called `baseURL` which contained the URL to the server from which we're supposed to exchange data with. When making the specific API-calls, I appended the corresponding endpoint before sending the request (`baseURL + "/desks"` for receiving the desk data, as an example). This way, if I were to take the course again next year, I could just change the `baseURL` variable to `"https://matthiasbaldauf.com/wbdg24"` without having to change the URL in every function where I make an API-call.

I used the Fetch API for the requests to the server. In this first step, I just stored the response in a variable and converted it into JSON. This way, I could directly use the data without further transformation.

Using the data from the response, I dynamically generated an HTML table containing all the desk data, as described in the design sketch. Here I came across another hurdle: In order to generate the table, I looped through the data by using a nested loop, in order to easily generate table row (`<tr>`) and table data (`<td>`) tags. The response consisted of an array containing objects describing one desk each. Because the outer loop was for an array, I had to use the `of` keyword instead of the `in` keyword, as described here. Because the inner loop iterated through the properties of an object, I had to use the `in` keyword for the inner loop. This was slightly confusing at first.

In each cycle of the inner loop, I checked if the name of the property was `available`. If it was, I used a ternary operator to change the text content of the cell to either "available" or "unavailable".



Desk Sharing Management Tool

ID	Name	Status	Address	Price	Latitude	Longitude	Comment
1	D7-4	available	Rosenbergstrasse 59, St.Gallen	65.2	47.4232	9.36763	2 screens
2	D7-3	unavailable	Rosenbergstrasse 59, St.Gallen	78.5	47.4232	9.36763	
3	Werft31-D1	available	Feldlistrasse 31, St.Gallen	67.5	47.4224	9.35311	
4	D6-1	available	Rosenbergstrasse 59, St.Gallen	42	47.4232	9.36763	
12	D2-2	available	Oberseestrasse 10, Rapperswil	34.4	47.2235	8.81754	
11	D2-1	available	Oberseestrasse 10, Rapperswil	34.4	47.2235	8.81754	
7	D2-3	available	Oberseestrasse 10, Rapperswil	34.4	47.2235	8.81754	
10	D2-4	available	Oberseestrasse 10, Rapperswil	3.4	47.2235	8.81754	
9	Werft31-D2	available	Feldlistrasse 31, St.Gallen	67.5	47.4224	9.35311	
13	D8-1	available	Rosenbergstrasse 59, St.Gallen	65.2	47.4232	9.36763	2 screens
14	D8-2	available	Rosenbergstrasse 59, St.Gallen	65.2	47.4232	9.36763	coffee machine close by!

Image 4: Dynamically generated table containing the desk data

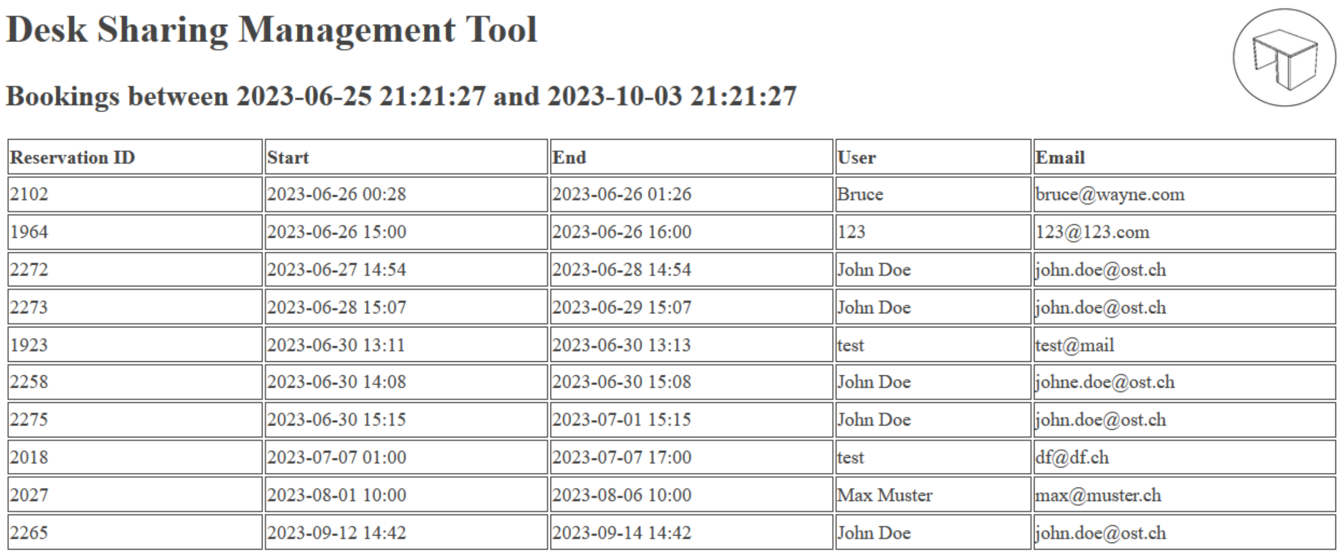
The relevant commit can be found here. At this point, I forgot to add the ability to view the bookings of a table. I added this feature at a later point in this commit.

The Problem with the API

When I went back and added the functionality to list all the current bookings of a desk, I came across the biggest problem I had while working on this project. At first I thought I made a mistake, because I was able to get a list of the booking for a desk, but I when trying to create a booking, I always received the response that the desk was already booked at that time.

At first I thought that there was something wrong with the dates that I supplied, but after about two hours of trying all kinds of different things, I noticed the problem: The API only gives you the bookings that were created with the student ID that you had to send with the request. Because of this, you are not able to give the user a complete list of all the bookings of a table.

I think that the API could be improved by making the student ID parameter optional. If it isn't supplied, the user should get a list of all the bookings from every user, because otherwise he just has to try different timeslots until he finds one that is open.



Desk Sharing Management Tool

Bookings between 2023-06-25 21:21:27 and 2023-10-03 21:21:27

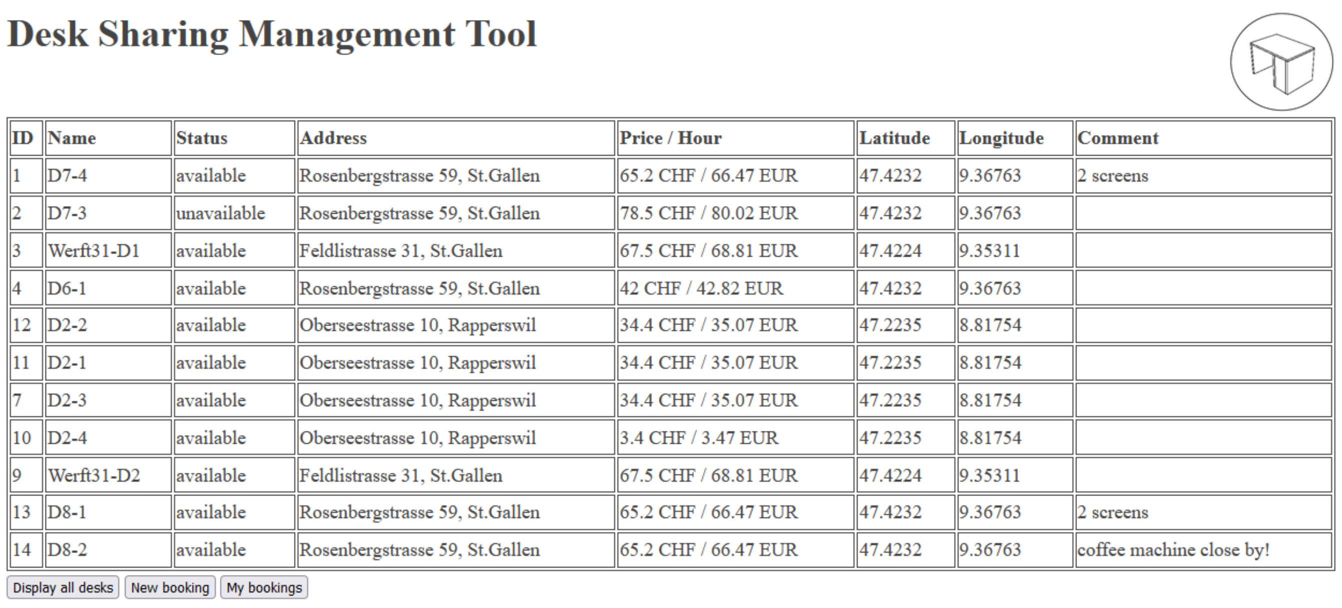
Reservation ID	Start	End	User	Email
2102	2023-06-26 00:28	2023-06-26 01:26	Bruce	bruce@wayne.com
1964	2023-06-26 15:00	2023-06-26 16:00	123	123@123.com
2272	2023-06-27 14:54	2023-06-28 14:54	John Doe	john.doe@ost.ch
2273	2023-06-28 15:07	2023-06-29 15:07	John Doe	john.doe@ost.ch
1923	2023-06-30 13:11	2023-06-30 13:13	test	test@mail
2258	2023-06-30 14:08	2023-06-30 15:08	John Doe	johne.doe@ost.ch
2275	2023-06-30 15:15	2023-07-01 15:15	John Doe	john.doe@ost.ch
2018	2023-07-07 01:00	2023-07-07 17:00	test	df@df.ch
2027	2023-08-01 10:00	2023-08-06 10:00	Max Muster	max@muster.ch
2265	2023-09-12 14:42	2023-09-14 14:42	John Doe	john.doe@ost.ch

Image 5: Bookings for desk 1 of student 1234

Implementing Costs per Hour in Different Currencies

The implementation of the currency conversion worked pretty much the same way as retrieving the desk data. I created a global variable called `conversionRates` where I stored the current rates that I received from the exchangerate.host API. Afterwards, I checked each time when generating a cell in the table if the property name was `price`, and changed the contents accordingly.

The only problem was that I didn't know right away how to quickly round a number to two decimals in JavaScript, but this was fixed with a quick trip to stackoverflow.



Desk Sharing Management Tool

ID	Name	Status	Address	Price / Hour	Latitude	Longitude	Comment
1	D7-4	available	Rosenbergstrasse 59, St.Gallen	65.2 CHF / 66.47 EUR	47.4232	9.36763	2 screens
2	D7-3	unavailable	Rosenbergstrasse 59, St.Gallen	78.5 CHF / 80.02 EUR	47.4232	9.36763	
3	Werft31-D1	available	Feldlistrasse 31, St.Gallen	67.5 CHF / 68.81 EUR	47.4224	9.35311	
4	D6-1	available	Rosenbergstrasse 59, St.Gallen	42 CHF / 42.82 EUR	47.4232	9.36763	
12	D2-2	available	Oberseestrasse 10, Rapperswil	34.4 CHF / 35.07 EUR	47.2235	8.81754	
11	D2-1	available	Oberseestrasse 10, Rapperswil	34.4 CHF / 35.07 EUR	47.2235	8.81754	
7	D2-3	available	Oberseestrasse 10, Rapperswil	34.4 CHF / 35.07 EUR	47.2235	8.81754	
10	D2-4	available	Oberseestrasse 10, Rapperswil	3.4 CHF / 3.47 EUR	47.2235	8.81754	
9	Werft31-D2	available	Feldlistrasse 31, St.Gallen	67.5 CHF / 68.81 EUR	47.4224	9.35311	
13	D8-1	available	Rosenbergstrasse 59, St.Gallen	65.2 CHF / 66.47 EUR	47.4232	9.36763	2 screens
14	D8-2	available	Rosenbergstrasse 59, St.Gallen	65.2 CHF / 66.47 EUR	47.4232	9.36763	coffee machine close by!

Image 6: Updated table containing price in different currencies

The relevant commit can be found here.

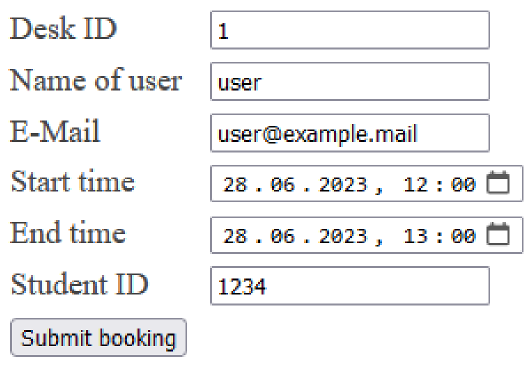
Task 3: Send Data

Validating the Inputs

Creating a booking wasn't a problem either for the most part. In order to validate the inputs, I created a function for each field, which would return either `true` or `false`, depending on if the input was correct. I then chained all those functions together in an if statement that would only go through if all functions returned `true`.

For the validation of the "name" field, I only checked if there was any input at all. You never know when someone with a very special name shows up that uses characters you didn't account for.

For the "email" and "student ID" fields I used regular expressions for input validations. I didn't bother trying to learn the regEx syntax, and just copy-pasted a pattern from somewhere else. I checked if those patterns were correct with this regEx tester.



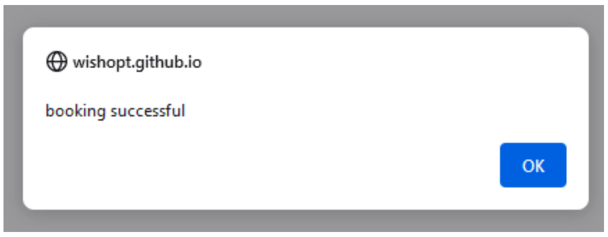
Desk ID	1
Name of user	user
E-Mail	user@example.mail
Start time	28.06.2023, 12:00
End time	28.06.2023, 13:00
Student ID	1234

Image 7: Input fields

The commit containing input validation can be found here.

Sending the Booking Request

I had a little bit of trouble when sending the booking request. At first I used URLSearchParams, as I've only used GET requests before. After some troubleshooting and a hint from a classmate I learned that the POST request requires a request body containing FormData instead of JSON. I later noticed that this is also mentioned on the MDN docs page for POST requests.



wishopt.github.io

booking successful

Image 8: Successful booking

My first unsuccessful try of sending data can be found in this commit. I later fixed it here.

Saving the User Data Locally

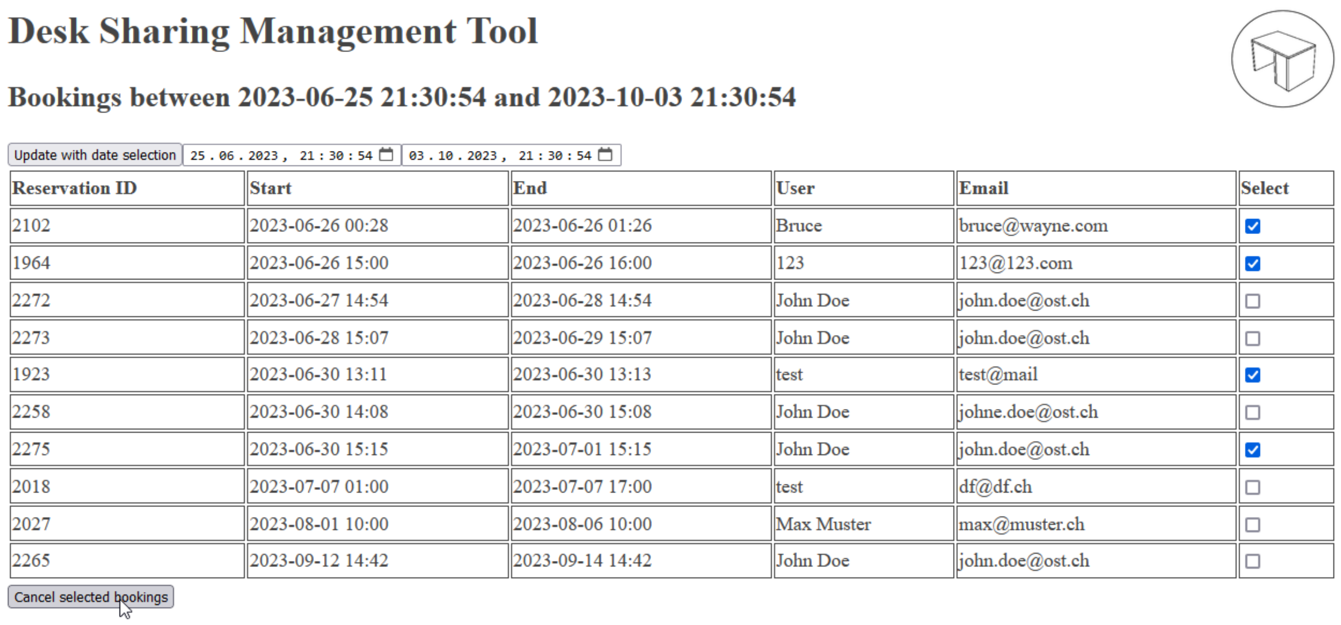
Saving data was very simple with the localStorage API. The only thing you have to watch out for is that the function `localStorage.getItem()` return `null` if there isn't any data available. Therefore, you have to do a quick check after loading the value, and set a default value if it was set to `null`.

The commit containing the save feature can be found here.

Cancelling Bookings

In order to allow the user to cancel bookings, I added an additional column to the desk bookings view, which contains a checkbox. The user can check multiple bookings at once, and delete them all by clicking "Cancel selected bookings".

I achieved this by giving each checkbox an ID corresponding to the position of the booking in the response array. When clicking on the "Cancel selected bookings" button, I loop through the variable containing the booking data once again, and check for each booking if the corresponding checkbox is checked. If it's checked, I add the booking to an array called `deletions`. Afterwards, I loop through the `deletions` array and send a DELETE request for each booking marked for deletion.



Desk Sharing Management Tool

Bookings between 2023-06-25 21:30:54 and 2023-10-03 21:30:54

Reservation ID	Start	End	User	Email	Select
2102	2023-06-26 00:28	2023-06-26 01:26	Bruce	bruce@wayne.com	✓
1964	2023-06-26 15:00	2023-06-26 16:00	123	123@123.com	✓
2272	2023-06-27 14:54	2023-06-28 14:54	John Doe	john.doe@ost.ch	☐
2273	2023-06-28 15:07	2023-06-29 15:07	John Doe	john.doe@ost.ch	☐
1923	2023-06-30 13:11	2023-06-30 13:13	test	test@mail	✓
2258	2023-06-30 14:08	2023-06-30 15:08	John Doe	johne.doe@ost.ch	☐
2275	2023-06-30 15:15	2023-07-01 15:15	John Doe	john.doe@ost.ch	✓
2018	2023-07-07 01:00	2023-07-07 17:00	test	df@df.ch	☐
2027	2023-08-01 10:00	2023-08-06 10:00	Max Muster	max@muster.ch	☐
2265	2023-09-12 14:42	2023-09-14 14:42	John Doe	john.doe@ost.ch	☐

Image 9: Cancelling bookings

The commit for cancelling bookings can be found here.

Conclusion

While working on this project, I learned a lot about working with APIs. Up until now, I've only created websites locally without a lot of API calls from the internet besides some simple GET requests. Working with APIs allows you to create many new and unique services that wouldn't be possible otherwise.

The final result can be found online at https://wishopt.github.io/WBDG-2023/, or you can download the files and open them locally.

This documentation was written while working on the project using MarkDown. The PDF was generated from the MarkDown file using the Markdown PDF plugin for VS Code.