REPORT OF FUTURE.IO

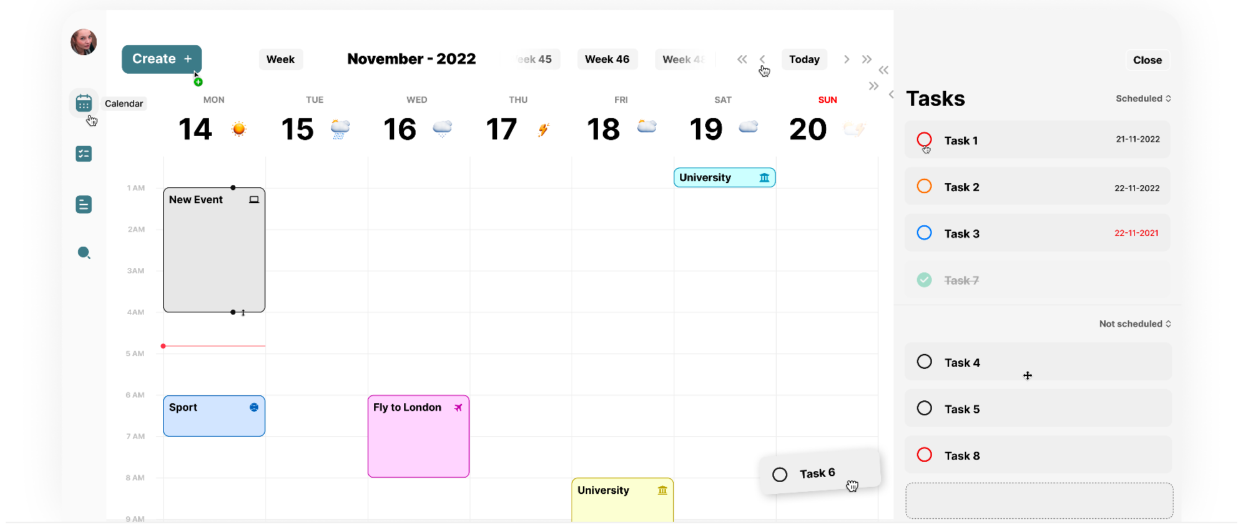
Elia Jabour, Hugo Krul, Pepijn Uuldriks

Informatica introduction project

Coach: Mahsa Ghanavatinasab

[27-1-2023]

Future.io



Utrecht University

Table of contents

You can view our code here: https://github.com/Elia-J/FutureWebsite
The readme inside provides instructions on how to run and view our code.
You can find our vision and midterm-report in the directory of our github.
Check the website https://future-website.vercel.app (demo) for the final product.

Current Status

The Future.io project is a comprehensive and user-friendly platform that provides a wide range of features for users to manage their daily tasks, events, and notes. The website is designed to be intuitive and easy to navigate, with a clear and simple layout that makes it easy for users to find what they need.

When you visit the website, you will be greeted with a landing page where you can sign in, sign up, or access the about page. The about page contains a form that users can use to contact us for support or to provide feedback.

The sign-up process is straightforward and easy to complete. Users can enter their first and last name, email, and password, and during the sign-up process, a function will detect the user's time zone and send that information along with the full name to the database. This information is saved in a separate table called profiles, which contains information about the user's settings, username, and other details. After sign-up, the user will receive an email to confirm their email address.

Users can sign in to the platform using their email and password, a magic link, or by signing in with Google. There is also an option to reset the password, and the user will receive an email with instructions on how to do so. Once the user has signed in, their profile data is loaded into a global variable that is accessible across the website.

After sign-in, users will be directed to the app directory where the calendar is located. Here, users can create, update, and delete events, assign colors to events, create daily or weekly repeated events, assign emojis to events, and add descriptions. Events require a title, begin date and time, and end date and time. Users can view their created events in a week view (default), day view, and month view. Additionally, users can view the weather forecast for the next 5 days, and there is a side panel where they can view their tasks and notes.

On the left side of the screen, users will find three navigation buttons that allow them access to the notes, tasks, or search page. Next to these buttons is the profile image (default is our logo), which when clicked will reveal a drop-down menu where users can access settings, change the theme, open the shortcut panel, sign out, or go to the about page to contact us.

The notes page allows users to create notes, create folders, set notes as priority, and delete notes. Users can edit notes and the editor includes features such as bold, italic, underline, h1, h2, text alignment, font size, and code blocks and quotes.

The tasks page allows users to create tasks, add priorities to tasks, update tasks, add a date and time and description. Users can sort tasks based on alphabet, priorities, date, and time, and create folders.

The settings page includes several options for users to customize their experience, including account settings, general settings, tasks settings, calendar settings, and weather settings. In account settings, users can change or add a username, full name, profile image, delete their account, delete the profile image, and reset the password. In general settings, users can choose a time zone (which is almost done), choose a theme, and sync the theme to the database. In task settings, users can choose to automatically remove checked tasks and show time for tasks instead of the date. In calendar settings, users can set a limit to the timeline, hide or show weekends, and in weather settings, users can choose the location where they want the weather forecast.

Other features of the website include dark mode compatibility, with the option to sync the mode state (dark, light, system) to the user's account. This way, whenever the user signs in on another device, the right theme will load. The global search feature allows users to search through events, tasks, and notes, not only by title but also by description. The search function has a feature to predict what the user is searching for, even if there is a typo in the search input. The website is also mobile compatible, making it easy for users to access their events, tasks, and notes from anywhere. Furthermore, the website boasts an innovative AI assistant feature that assists users in taking notes and creating tasks.

Updated MoSCoW Analysis

All changes made in our MoSCoW analysis are italicized and our reason for said change can be found in between the curly brackets.

Must Have:

- ***Landing page.*** *Our website should have a landing page that will serve as an index page for our website. The landing page should act as a navigational hub which would allow users to navigate to the login/signup page and about page. [Every website needs a landing page on which users will land when they first visit a website.]*
- **User login *and User signup***. The website must support user login to ensure that a user will only ever be able to view their own personalized events, tasks and notes. *The website must also support user signup so users can make accounts to login to. [We made a small oversight in our first MoSCoW analysis by not including this part. Because obviously users cannot login to an account if they do not have the ability to actually make said account.]*
- **Calendar page**. This page should include a calendar on which users can create events that will be displayed on the calendar itself. This will be the main draw of our website.
- **Tasks page**. This page should allow a user to make and save tasks. This way planning will be a lot easier if you have a lot of things to do.
- **Notes page**. Implementing the availability of making notes is another thing that is a must in our application. For example, if you follow a course at the university and the professor has a lot of homework and things to remember for the next class, you would be able to save the notes ~~and link them to an event.~~ *[We have decided to move the linking of a note to an event from our musts to could have, because this in truth hardly a feature which would make or break our website.]*

These features are the core of our website and without them there would be no reason for anyone to use it.

Should Have:

- **Week numbers**. The calendar should show the number of the week it is currently on to help a user with navigating the calendar.
- **Weather**. To make the planning of events easier, we will show little icons next to the date to show what the weather expectations are for that day. This way, a user can hold weather into consideration when planning their daily life.
- ***Repeating events.*** *The events should be repeatable if a user wishes this so. Meaning that an event can be made to repeat daily/weekly/etc. [We added this functionality since we think that this would add a lot to the user-experience.]*
- ***User settings.*** *A user should have certain settings exclusive to their account to personalize their experience. So a user should be able to change their password and avatar. But a user should also be able to set their preferred theme to either dark/light/system. And we would like to have some settings that would be exclusive to a particular page. For example they're might be a setting which would, when activated, automatically delete all completed tasks. This setting would be exclusive to the task page. [We have added this should because we think it would allow a user to really personalize their experience with our website]*
- ***Sorting algorithms.*** *All tasks/notes/folders should have sorting mechanisms so that a user can more easily find things. So for example tasks should be sortable by alphabetical order (a-z) and by alphabetical order (z-a). [The more tasks/notes/folders a user has the easier it might be to lose sight of them, so these sorting algorithms should help with that.]*
- **Priorities**. To make things more customizable tasks, events and notes should have a system which would allow users to assign priorities to them.
- **Folders**. To keep things organized, we would like to add the feature of folders for tasks and notes. This way a user can organize things as they like.
- **Search**. *If one has a lot of tasks, notes and events saved it might be difficult to find a specific one. The search feature should solve this issue by allowing a user to search by name or description for a specific task, note or event. A user should be sent to a task, note or event when they click on them. [Previously our search feature only allowed a user to search through their notes, but we have decided to expand this feature and allow one to search through all of their saved tasks, notes and events to improve the user-experience.]*

We should add these features because all of them will make the experience of the application more comfortable without making it too complicated.

Could Have:

- **Colour.** This feature is good for organization. You should be able to change the colour of tasks, notes, and events so you can distinguish them from others.
- ***About page.*** *Our website could have an about page which would show some information about us, answer some frequently asked questions, and have a way to contact us via an email form.*
- **Shortcuts**. To make navigating easier around the website, the implementation of shortcuts could help. For example 'ctrl' + 'enter' will save the event, task, or note.
- **Dark mode/light mode*/system mode***. Visual pleasantness is important for this application. That's why we would like to add a dark/light/system mode which would

allow one to personalize the website’s look to their liking. *[We have added the system mode to this because we think it will improve use-ability for users.]*
- **Icons**. This is another feature of visual pleasantness. If you have an event where you have to say code or eat you could add an icon to the event.
- ***AI Assistance.*** *We want to make the experience as comfortable as possible for users. To add to this comfortability we would like to have some AI assistance for users. This AI assistance would help a user in making notes and tasks. [This way users can ask the AI for help in creating tasks and notes which will hopefully save them a lot of time otherwise spent manually creating the notes and tasks.]*
- ***Linking.*** *Notes, tasks, and events could be linked together. Meaning that a note might have a reference to a task which would when clicked take a user to said task. [We have added this because we think it will allow for more interconnection between the three main pages of our website.]*

We could implement these features if we have time left. They do not define the application but they still are useful for the experience of the user.

Won’t have:
- ***Offline functionality.*** *Our website won’t work without an internet connection. We believe that our target audience will for the most part be connected to the internet at all times. Because of this belief we don’t think it particularly useful to spend a lot of time making an offline functionality.*
- ***Be an app.*** *You won’t be able to download or use our project from the app/play store.*
- ***Share events/tasks/notes with other users.*** *In teams one can easily share one and the same event (meeting in teams). A user can’t share events/tasks/notes with other users. We want our website to be an individualized experience and not a shared one.*

Our website won’t have these features. We do not believe that these features are necessary for the vision we have of our website. [We previously didn’t have a won’t have section in our MoSCoW analysis but we think it might be useful for some to get an idea of what our website will be like.]

Planning Evaluation

Our planning was fairly simple. In the first week we worked on the first part of the website, which is the landing page, the about page and the sign in and out pages. From there on we divided the remaining work. We are with three people which makes the planning really easy. For our product we needed a page for notes, one for tasks and we needed a calendar. So we divided the three main pages between us and we continued to work on them throughout the whole project. There were some additional features like a search page and settings but we divided those features evenly. Every task we planned for that week was done before Tuesday so our planning was really realistic.

Each week on friday we had a meeting where we discussed what we did in the last sprint, we answered questions we had and we made a planning for the next week. Only in the last two sprints did it happen where we couldn’t finish a task we had, but we held that in mind while planning the next sprint.

We don’t think we should have done the structure of our planning differently. We started with the basic fundamentals of our pages. First the design, then logic behind every button and input and finally linking everything with our database. From there on, each week we worked on tweaking, improving and adding features to those pages.

We think the communication in what specifically is done in a sprint and how we solved the tasks could have been better. For example, two of us needed to make side panels for the folders of notes and tasks. We think we should have discussed, for example, the design and the transitions of those opening panels before actually building them. This goes for a lot more in our project. Everything did work out, but it would have been more efficient if we first thought about every detail before actually implementing this. Another example of this is the way the title of the notes should be stored. Slate takes in an JSON object this way:

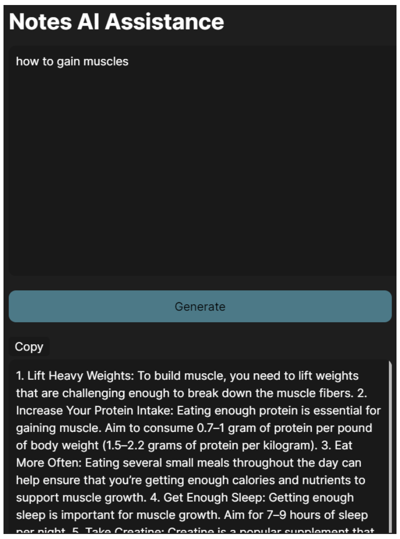
```
SlateTitle = [{ "type": "h1", "children": [{ "text": "TITLE"}]
TextTitle = SlateTitle[0].children[0].text
```

Only later on we came to the conclusion that we needed the title to store only as text not as JSON. That way it can be used by other pages more easily. So now, every time we used the title had to be changed.

Changes

We decided that the linking between tasks, notes and events is something we aren’t able to do. There is just too much error handling we needed to do in a short period of time. We also got rid of the idea of dragging and dropping notes and tasks to folders. NextJS is not compatible with dragging and dropping elements. You still can use folders but you have to make use of buttons to add the tasks and notes to folders.

We also added a way to let AI help with your notes and tasks. You can see the example on the side here. On the task page the AI will come up with tasks you can add to your database with the “add” button. On the notes page, the AI will give a detailed description of what to do, you also have a copy button so you can copy the generated text to your notes. On the task page it will give an answer in tasks so you can add them to your todo list.

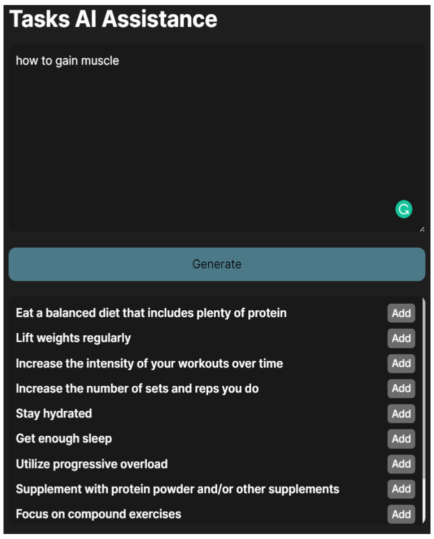

Notes AI Assistance

how to gain muscles

Generate

Copy

1. Lift Heavy Weights: To build muscle, you need to lift weights that are challenging enough to break down the muscle fibers. 2. Increase Your Protein Intake: Eating enough protein is essential for gaining muscle. Aim to consume 0.7–1 gram of protein per pound of body weight (1.5–2.2 grams of protein per kilogram). 3. Eat More Often: Eating several small meals throughout the day can help ensure that you’re getting enough calories and nutrients to support muscle growth. 4. Get Enough Sleep: Getting enough sleep is important for muscle growth. Aim for 7–9 hours of sleep per night. 5. Take Creatine: Creatine is a popular supplement that


Tasks AI Assistance

how to gain muscle

Generate

Eat a balanced diet that includes plenty of protein Add
Lift weights regularly Add
Increase the intensity of your workouts over time Add
Increase the number of sets and reps you do Add
Stay hydrated Add
Get enough sleep Add
Utilize progressive overload Add
Supplement with protein powder and/or other supplements Add
Focus on compound exercises Add

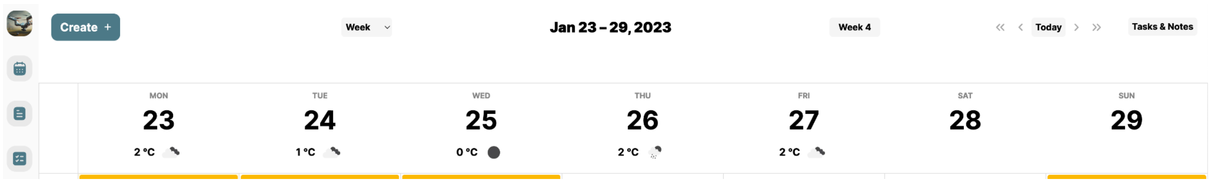
We also tried to implement timezones. If you change the timezone from europe/amsterdam to, for example, ireland. The calendar will update all the times in tasks and events. This however was not possible because one library that we used that shows the current time had a bug in which the timezone change couldn’t happen.

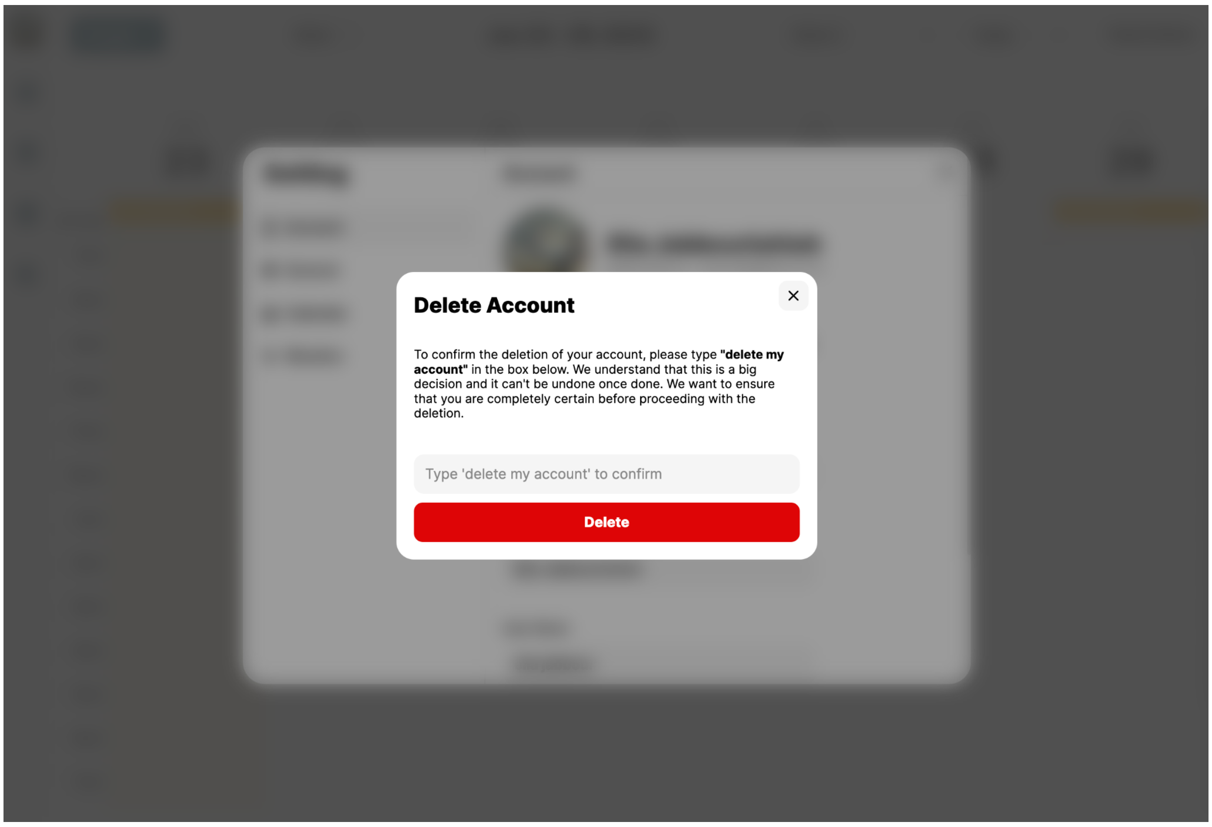
Highlights

Elia:

Next.Js API Route

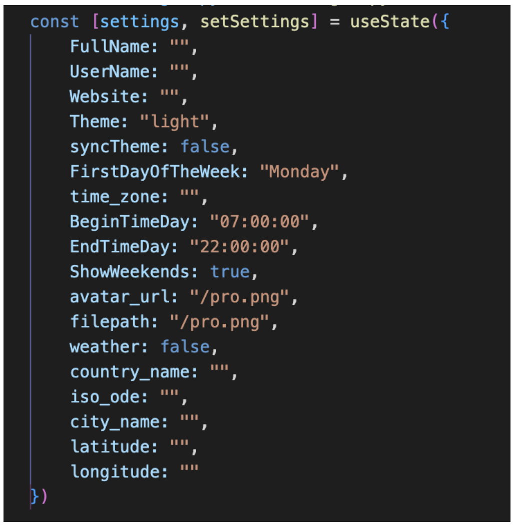
Using Next.js API routes, I was able to create specific endpoints for specific functionality, such as deleting a user account, retrieving current weather information and the AI Assistance, which can be accessed through various HTTP requests like GET or POST. The deleting user account function is protected with a JWT access token, which ensures that each user can only delete their own account. This, combined with the fact that these routes are implemented on the server-side and API keys are kept secure, increases the overall security of the application.
You can view the code in github/pages/api/deleteUser.js or weather.is

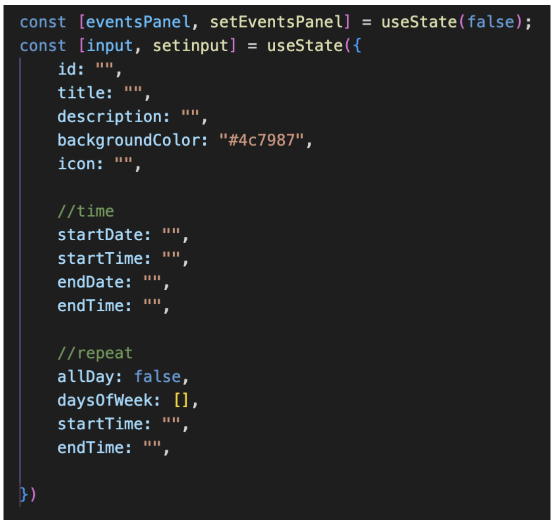



Delete Account

To confirm the deletion of your account, please type **"delete my account"** in the box below. We understand that this is a big decision and it can't be undone once done. We want to ensure that you are completely certain before proceeding with the deletion.

Global variables
Using global variables for settings and events can allow for data to be easily accessible and shared across different parts of a program or application, making it a convenient method for managing and maintaining the state of an application.
You can view the code in github/pages/layouts/stateStore.js or stateStoreEvents.js

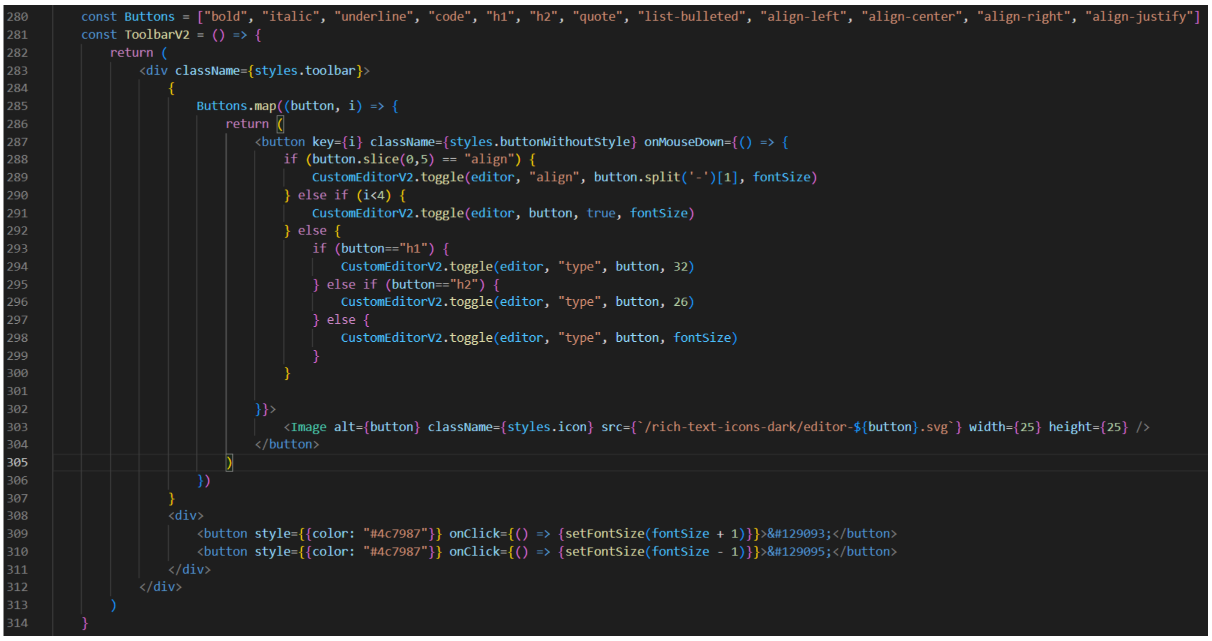

```
const [settings, setSettings] = useState({
    FullName: "",
    UserName: "",
    Website: "",
    Theme: "light",
    syncTheme: false,
    FirstDayOfTheWeek: "Monday",
    time_zone: "",
    BeginTimeDay: "07:00:00",
    EndTimeDay: "22:00:00",
    ShowWeekends: true,
    avatar_url: "/pro.png",
    filepath: "/pro.png",
    weather: false,
    country_name: "",
    iso_ode: "",
    city_name: "",
    latitude: "",
    longitude: ""
})
```


```
const [eventsPanel, setEventsPanel] = useState(false);
const [input, setinput] = useState({
    id: "",
    title: "",
    description: "",
    backgroundColor: "#4c7987",
    icon: "",

    //time
    startDate: "",
    startTime: "",
    endDate: "",
    endTime: "",

    //repeat
    allDay: false,
    daysOfWeek: [],
    startTime: "",
    endTime: "",
})
```

Hugo:
For the rich text editor, we needed a lot of buttons which toggle properties for specific styles. At first, for every button, I had two functions which toggled the property but that was really inefficient. I needed a way to make this more readable and use less lines of code.

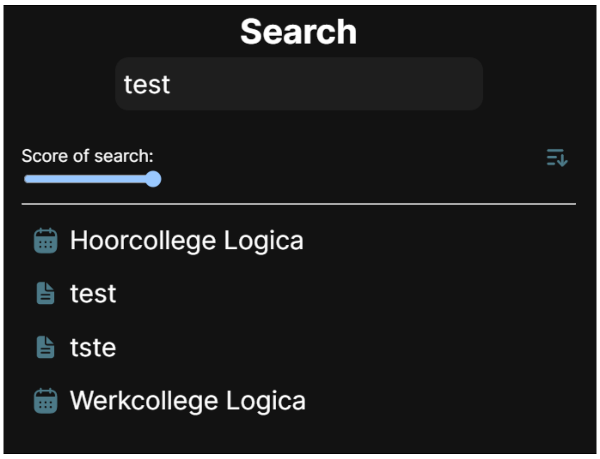


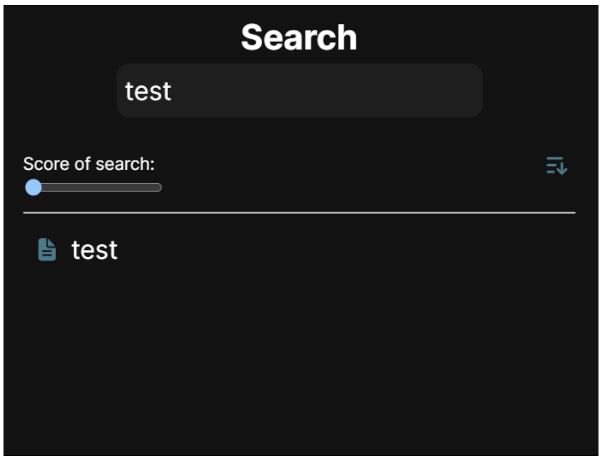
```
const Buttons = ["bold", "italic", "underline", "code", "h1", "h2", "quote", "list-bulleted", "align-left", "align-center", "align-right", "align-justify"]
const ToolbarV2 = () => {
    return (
        <div className={styles.toolbar}>
            {
                Buttons.map((button, i) => {
                    return (
                        <button key={i} className={styles.buttonWithoutStyle} onMouseDown={() => {
                            if (button.slice(0,5) == "align") {
                                CustomEditorV2.toggle(editor, "align", button.split('-')[1], fontSize)
                            } else if (i<4) {
                                CustomEditorV2.toggle(editor, button, true, fontSize)
                            } else {
                                if (button=="h1") {
                                    CustomEditorV2.toggle(editor, "type", button, 32)
                                } else if (button=="h2") {
                                    CustomEditorV2.toggle(editor, "type", button, 26)
                                } else {
                                    CustomEditorV2.toggle(editor, "type", button, fontSize)
                                }
                            }

                        }}>
                            <Image alt={button} className={styles.icon} src={`/rich-text-icons-dark/editor-${button}.svg`} width={25} height={25} />
                        </button>
                    )
                })
            }
            <div>
                <button style={{color: "#4c7987"}} onClick={() => {setFontSize(fontSize + 1)}}>&#129093;</button>
                <button style={{color: "#4c7987"}} onClick={() => {setFontSize(fontSize - 1)}}>&#129095;</button>
            </div>
        </div>
    )
}
```

These 34 lines of code add the toolbar from a list of buttons. They also manage to toggle all the different properties with style because of the way I rewrote CustomEditorV2.
Instead of 24 different CustomEditor functions (because one needs to check if it's toggled already and one needs to toggle it if it's not turned on yet) I have shortened it to 12 different functions, reducing my total lines of code by more than 200.

I am also really proud about my research into searching in big data. I found a library called fusejs which searches through the data based on an input in one line. It gives suggestions on what it would think you were meant to type. For example, if you make a typo, fusejs recognizes this and still gives you suggestions based on what it thinks you meant. It also gives you a score which lets you change how strict the search needs to be.

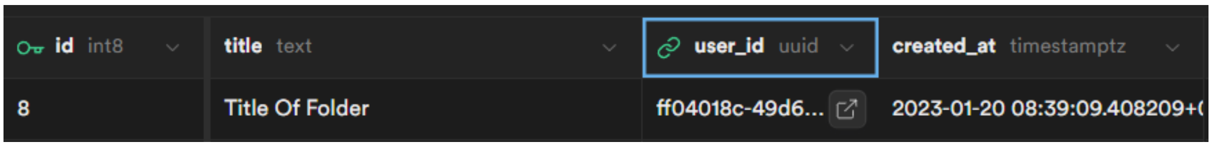




This is all done in 7 lines of code which I think is really efficient.

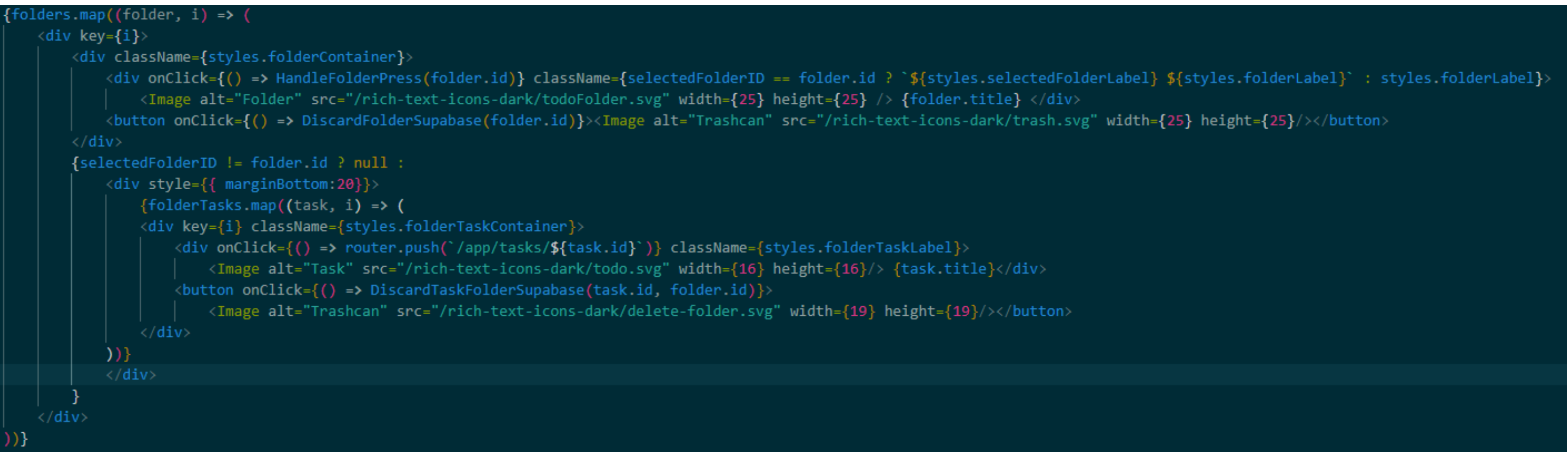
```
const fuse = new Fuse(data, options)
const result = fuse.search(inputTest)
for (let i=result.length-1; i>0; i--) {
    if (result[i].score > score) {
        result.splice(i, 1)
        }
}
setMatch(result)
```

Pepijn:
Early on we decided to have a folder system in place for notes and tasks. It was my job to make a folder system for the tasks. At that point I already had a fully functioning tasks table, and so I had to figure out a way to have these tasks be saveable in folders. Allowing users to create folders was relatively easy. I simply made a table in supabase and then created a way for users to create folders from our website. The table includes an automatically generated unique id, a user given title, the user_id from whoever created the folder and a created_at timestamp.



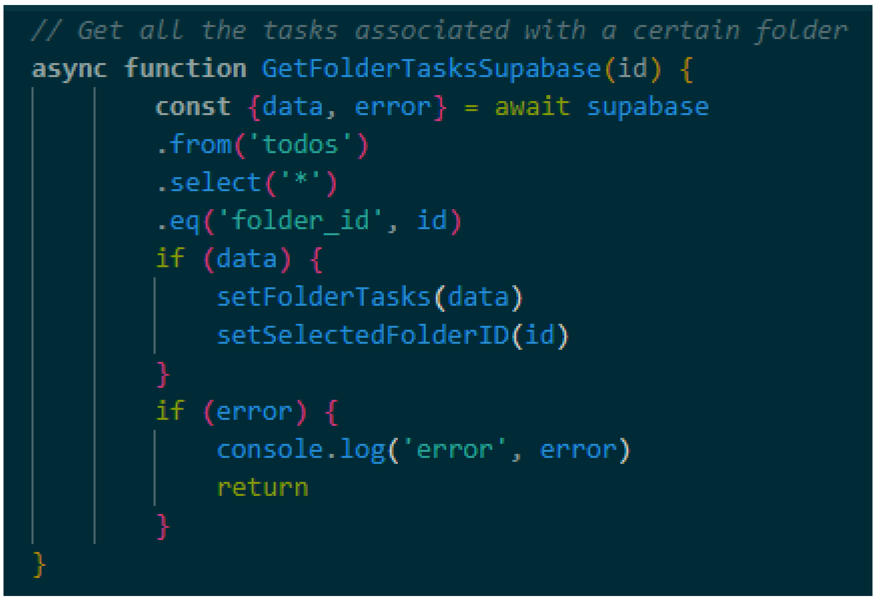
id int8	title text	user_id uuid	created_at timestamptz
8	Title Of Folder	ff04018c-49d6...	2023-01-20 08:39:09.408209+(

The tricky part for me came when I had to actually create a way for users to save tasks in their created folders. I wasn't really sure how I could achieve this. Luckily however I found a way. Firstly I added a new foreign key to my tasks table. This foreign key would either be null or be the id of a folder it would belong to. Below is an image of my code. Now most of it isn't really relevant here. All that is really needed to know is that this code loops through all of the folders of a user and outputs them on the website individually. On rule 4 of this code there is a <div> which activates the function HandleFolderPress(folder.id) when clicked. The folder.id that it gives along as a parameter is the specific id of the folder which is clicked.



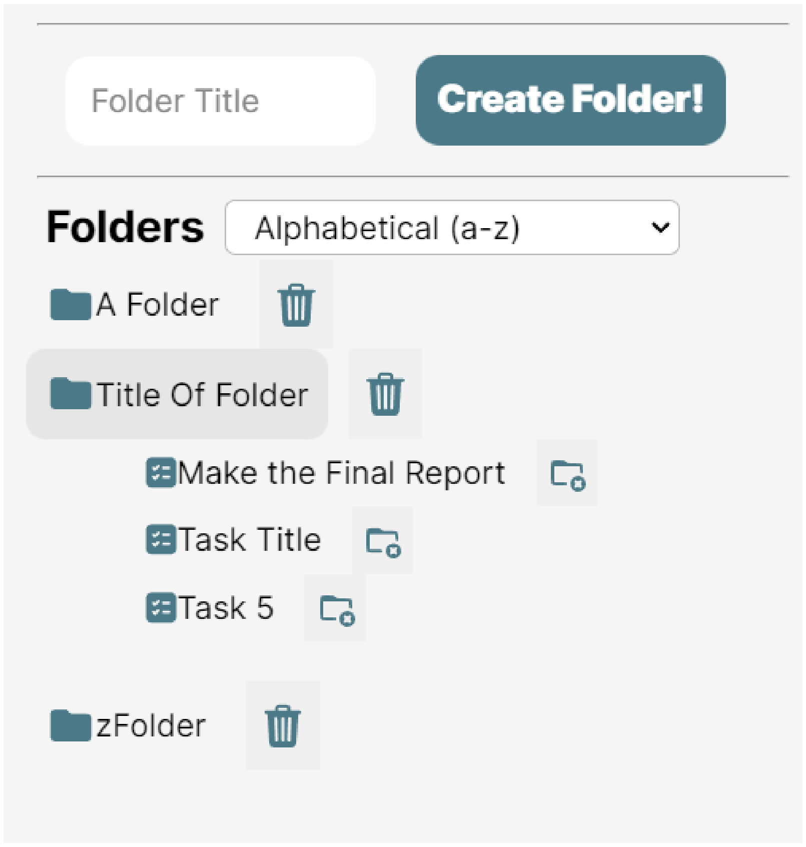
```
{folders.map((folder, i) => (
    <div key={i}>
        <div className={styles.folderContainer}>
            <div onClick={() => HandleFolderPress(folder.id)} className={selectedFolderID == folder.id ? `${styles.selectedFolderLabel} ${styles.folderLabel}` : styles.folderLabel}>
                <Image alt="Folder" src="/rich-text-icons-dark/todoFolder.svg" width={25} height={25} /> {folder.title} </div>
            <button onClick={() => DiscardFolderSupabase(folder.id)}><Image alt="Trashcan" src="/rich-text-icons-dark/trash.svg" width={25} height={25}/></button>
        </div>
        {selectedFolderID != folder.id ? null :
            <div style={{ marginBottom:20}}>
                {folderTasks.map((task, i) => (
                <div key={i} className={styles.folderTaskContainer}>
                    <div onClick={() => router.push(`/app/tasks/${task.id}`)} className={styles.folderTaskLabel}>
                        <Image alt="Task" src="/rich-text-icons-dark/todo.svg" width={16} height={16}/> {task.title}</div>
                    <button onClick={() => DiscardTaskFolderSupabase(task.id, folder.id)}>
                        <Image alt="Trashcan" src="/rich-text-icons-dark/delete-folder.svg" width={19} height={19}/></button>
                </div>
            ))}
            </div>
        }
    </div>
))}
```

HandleFolderPress(folder.id) activates the GetFolderTasksSupabase(id) function if the id of the folder given along isn't already selected (so when a folder is already open it won't activate GetFolderTasksSupabase(id)). GetFolderTasksSupabase(id) would then request all the tasks which would have the folder.id as a foreign key. I would then set the folderTasks to the requested data and I would set the selectedFolderID to the specific id given (see below).



```
// Get all the tasks associated with a certain folder
async function GetFolderTasksSupabase(id) {
        const {data, error} = await supabase
        .from('todos')
        .select('*')
        .eq('folder_id', id)
        if (data) {
            setFolderTasks(data)
            setSelectedFolderID(id)
        }
        if (error) {
            console.log('error', error)
            return
        }
}
```

After this all the tasks associated with the selected folder would then be displayed as shown below:



Group Reflection

One of the key take-aways from this project for us is the importance of communication. We realized at the start of this project that having weekly meetings and open and quick lines of communications would be required to ensure the success of this project. Every Monday we would meet one another at the university and discuss the ongoings of the project. This would also be the day on which we would usually ask one another questions and ask for help when needed. On Tuesday we would have our coach meeting and on Friday we would have a digital meeting via Teams to discuss the current sprint and make a plan for the next one. We created a whatsapp group in which we could communicate with each other over text. We firmly believe that our prudence at the start of this project is what allowed us to get through this without any major issues.

We learned a lot about how to work together as a team on a project. We learned how to effectively make use of Github and about how version control can be a life-saver. Hugo and Pepijn, both quite inexperienced in web development at the start, learned a lot about next.js and web development in general. We also learned how to work with Supabase to create a fully functioning database.

We are quite happy with how this project has gone and so don’t have many things we would do differently given the chance. There are only a couple of minor things that we would do differently, chief among them being the following: at the beginning of this project when making a plan we would give ourselves a couple of major tasks for the following sprint. After some nudging by our coach however we came to realize that these major tasks simply weren’t specific enough. To give an example: in sprint 2 Pepijn had as one of his major tasks to simply make an about page. This however is clearly not very detailed in its definition and could thus be interpreted in almost infinitely different ways. So, because of this realization we changed our approach to planning. We still gave ourselves a couple of major tasks per sprint, but now also accompanied each major task with a couple of minor tasks which would elaborate and specify said major task.