1. **What bugs did you find ? How did you solve these and why ?**

App.js

When a user's availability status is updated, the change is not reflected in the list of users right away. The list still shows the old availability status until the page is manually refreshed.

1. **Added local state in `App`:**

```
const [refreshAvailableUsers, setRefreshAvailableUsers] =
useState(false);
```

2. **Passed an `onAvailabilityChange` callback to `Header`:**

```
<Header onAvailabilityChange={() => setRefreshAvailableUsers((r) =>
!r)} />
```

Drawer.js

The section menu does not open or close properly because the toggleable `open` state is not managed correctly across `DrawerDesktop` and `Section`.

What changed:

1. Introduced a toggleable `open` state in the Drawer

```
const [open, setOpen] = React.useState(false);
```

2. Passed `open` and `setOpen` to the `DrawerDesktop` and `Section`

```
<DrawerDesktop open={open} setOpen={setOpen} />
<Section title="Général" setOpen={setOpen}>
  {open && (
    ...
  )}
</Section>
```

3. Updated `Section` to toggle the menu on click

```
onClick={() => setOpen((o) => !o)}
```

Edit.js/Project

1. Redirect after Delete

Change:
`history.push("/projects");`

To:

`history.push("/project");`

- I **corrected the redirect path** after deleting a project.
- This ensures users land on the main projects list view, not a noncorrect route.

Project.js Backend

When I tried to open the project detail page, the page failed to load correctly because the backend returned an array instead of a single project object. This caused `data` to be `undefined` when trying to access its properties **List.js/Project**

The modification from:

```
const data = await ProjectObject.find({ _id: req.params.id });
```

to:

```
const data = await ProjectObject.findOne({ _id: req.params.id });
```

View.js / project

If `project.name` is `null` or `undefined`, calling `.toString()` on it would throw an error:

`null.toString()` Whereas this: `{project.name}` will simply render nothing `("")` `or undefined`, which React can handle gracefully.

from:

```
<span className="w-fit text-[20px] text-[#0C1024] font-
bold">{project.name.toString()}</span>
```

to:

```
<span className="w-fit text-[20px] text-[#0C1024] font-
bold">{project.name}</span>
```

List.js /User

Changed input `name="username"` to `name="name"`

Before:

```
<input name="username" value={values.username} />
```

After:

```
<input name="name" value={values.name} />
```

There is a user schema that has:

```
{
  "name": "Alesia",
  "email": "alesia@example.com"
}
```

and **not** a `username` field.

View.js /User

Fixed button functionality

The `LoadingButton` used `onChange` instead of `onClick`. As a result, clicking the button did **not trigger** the `handleSubmit`function , so the update action did nothing.

Before:

```
<LoadingButton ... onChange={handleSubmit}>
```

After:

```
<LoadingButton ... onClick={handleSubmit}>
```

Index.js/ Activities

1. `onChange={(e) => setProject(e ? e.name : "")}`

- Prevents a crash if `SelectProject` returns `null` or `undefined`.

Before:

```
onChange={(e) => setProject(e.name)}
```

After:

```
onChange={(e) => setProject(e ? e.name : "")}
```

2. Added `project` as a dependency to the `useEffect` in `Activities`

- Without it, changes to the selected project wouldn't trigger a refetch of activities.
- Now the effect reruns when either `date` or `project` changes.

Before:

```
useEffect(() => { ... }, [date]);
```

After:

```
useEffect(() => { ... }, [date, project]);
```

3. Filtered activities based on project name

instead of filter function in frontend, I implemented it in backend based on the project name and the date selected, now there are shown all the activities for each project selected

```
router.post("/search", passport.authenticate("user", { session: false }), async (req, res) => {
  try {
    const { date, projectName } = req.body;

    if (!date) {
      return res.status(400).send({ ok: false, error: "Missing date" });
    }

    const selected = new Date(parseInt(date));
    const firstDay = new Date(selected.getFullYear(), selected.getMonth(), 1);
    const lastDay = new Date(selected.getFullYear(), selected.getMonth() + 1, 0);

    const match = {
      organisation: req.user.organisation,
      date: { $gte: firstDay, $lte: lastDay },
    };

    const pipeline = [
      { $match: match },
      {
        $lookup: {
          from: "projects",
          localField: "projectId",
          foreignField: "_id",
          as: "project",
        },
      },
      { $unwind: "$project" },
```

```
    ];

    if (projectName && projectName.trim() !== "") {
      pipeline.push({
        $match: {
          "project.name": { $regex: projectName, $options: "i" },
        },
      });
    }

    pipeline.push({ $sort: { created_at: -1 } });

    const data = await ActivityObject.aggregate(pipeline);

    return res.status(200).send({ ok: true, data });
  } catch (error) {
    console.log(error);
    res.status(500).send({ ok: false, code: SERVER_ERROR, error });
  }
});
```

Why this is correct:

Previously, activity filtering and project name matching were done in the frontend. This caused unnecessary data to load and the filtering/search logic to run client-side, which could be slow and inconsistent, especially with large data.

Activity/model

Previously, the projectId in the Activity model was defined as a string. This caused issues when joining with the Project collection, because MongoDB uses ObjectId for _id fields. As a result, lookups and queries on projectId did not work properly, breaking filters and joins. So, I changed the project id as an object instead of a string in activity model.

```
const Schema = new mongoose.Schema({
  //projectId: { type: String },
  projectId: { type: mongoose.Schema.Types.ObjectId, ref: "project" },
  userId: { type: String },
  userAvatar: { type: String, default: "ht
```

2. Which feature did you develop and why ?

The web application already shows who worked, how many hours, and the project's budget usage, so project leads and managers can monitor progress anytime.

In real startup life, founders and project leads are often busy or not logged in.
Investors or incubators frequently ask for updates in a shareable format.
Without this feature, the lead has to log in, look at the dashboard, manually copy or export data, and send it by email.
In this way, with "Send Report Now", the same up-to-date project data is compiled and emailed immediately — with one click — to the project lead, stakeholders, or investors.
This ensures that everyone has the latest figures, total budget spent, total hours, and detailed contributions per team member, directly in their inbox.

Benefit:

- Saves time for project leads — no manual reporting.
- Keeps communication proactive and professional — easy to forward to stakeholders.
- Reinforces trust — no surprises, everyone stays informed even without logging in.
- Positions our platform as an active reporting tool, not just a passive dashboard.

3. Do you have any feedback about the code / architecture of the project and what was the difficulty you encountered while doing it ?

Overall, the project structure is quite clear and straightforward, which made it easy to understand in general the main flow quickly.

A clearer separation of concerns, such as isolating data fetching into a custom hook and keeping the component focused purely on rendering would make the codebase easier to maintain, test, and reuse, especially as the project grows.

One problem I encountered was that, initially, the filtering and search logic were handled entirely on the frontend, which made the code less efficient and harder to maintain. To improve this, I moved the filtering logic to the backend. This required restructuring the API, building flexible query conditions, and testing various edge cases to ensure the frontend displays accurate results. It took time to refactor and verify, but it made the overall solution cleaner and more scalable.