

Uppdragsportalen

Dadi Andrason, Julia Böckert, Steffanie Kristiansson, Emil Lindblad

DAT076

Group 1

Chalmers University of Technology

March 14, 2023

1 Introduction

At Chalmers, there is a reception committee (NollK for short) at each program every year to welcome new students, and each program has its own name for these committees. There is also one central reception committee, which is called Chalmers Reception Committee (in Swedish *Mottagningskommittén*, MK for short). MK coordinates and helps the other NollK:s both before and during the big reception in August/September.

One part of the reception consists of new students performing a so-called Nolluppdrag, which are special missions/assignments to complete during a certain day or week during the reception. Every NollK has to send all their made-up Nolluppdrag to MK for review and has to wait for either approval or denial for each assignment. At this point, the system they use for this is Google Drive, and there is no great structure, so MK wants a better solution. A few years ago they had a website for this but it broke when the Student Union decided to upgrade from PHP version 5 to PHP version 7. This version had some major changes to its core codebase and the website was not maintained properly by MK, and therefore stopped working.

To facilitate the work for MK, a new website would be a perfect solution. A member of the Chalmers Reception Committee is also a member of this project, and has acted as a sort of stakeholder. The idea is to have a site with accounts and roles, different libraries for the Nolluppdrag, the possibility to approve and deny them, and give comments or feedback.

2 Use cases

These are the use cases that are currently implemented and working.

1. Login (both as MK and Nollk)
2. Logout
3. Create a new account
4. Change account information (user/email and password separately)
5. Fetch Nolluppdrag with certain criterias (All, certain year, certain author etc)
6. Create new Nolluppdrag
7. Edit a Nolluppdrag
8. Delete a Nolluppdrag
9. Read a specific Nolluppdrag
10. Sort the table of Nolluppdrag by ascending and descending order (also back to default)
11. Search in table
12. Accept/deny account requests as MK
13. Delete existing accounts as MK
14. Apply different status to Nolluppdrag (Approved, Denied, Return)
15. Add a comment to a Nolluppdrag

3 User manual

When the user goes to Uppdragsportalen they are first met by the login page, figure 1. Here the user logs in with their account and gets redirected to the main page. Depending on what role the user have there are some small differences on the website. Each role is described in the following two sections.

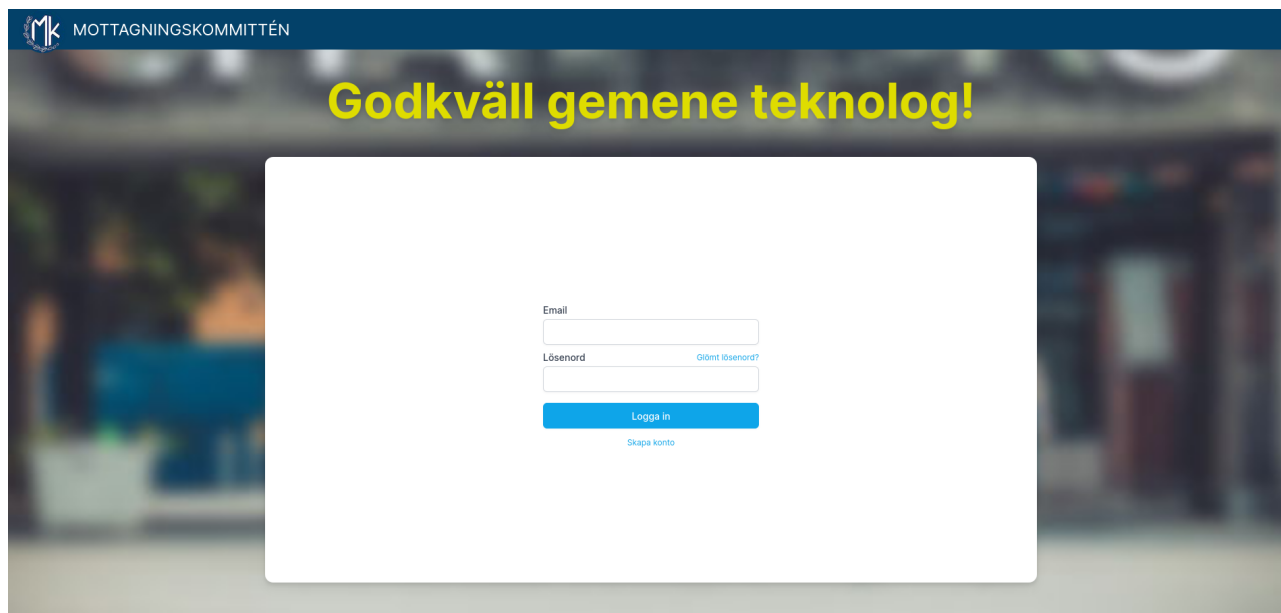


Figure 1: The start page of the web application

If a user does not have an account yet, there is a button *Skapa konto* where it is possible to create one, but access to the account is not given straight away. Since the site is only available for MK and Nollk:n, the new account is sent as a request to the admin. To be accepted the admin of the site, in this case MK, has to review and either accept or deny the account request.



Figure 2: The register page of the application

On the page seen in figure 2, here the user can request access to the web application by registering an account. The user has to be approved by MK, as seen in figure 8 before being able to log in to Uppdragsportalen.

Any user that is logged in can also change their user information such as name, email and password as can be seen in figure 3. It is possible to only change the name and email since it is auto-filled and not have to

change password at the same time. You can also change all the information at the same time.

The screenshot shows a user interface for changing personal information. On the left is a dark blue sidebar with a user profile at the top (MK logo, name Dadi Andrason, email pr@mk.chs.chalmers.se) and menu items: Granska, Konton, Årets Nolluppdrag, Arkiv, and Mitt konto (highlighted in blue). At the bottom of the sidebar is a yellow 'Logga ut' button. The main content area has a heading 'Hej Dadi Andrason!' and a subheading 'Om du vill ändra dina uppgifter kan du göra det här!'. Below this are four input fields: 'För- och efternamn:' with 'Dadi Andrason', 'E-mail:' with 'pr@mk.chs.chalmers.se', 'Lösenord:' with 'Lösenord', and 'Bekräfta Lösenord:' with 'Bekräfta lösenord'. A yellow 'Submit' button is at the bottom right.

Figure 3: A page to change your own user information

3.1 NollK-account

When the user logs in as a NollK, the main page shows a table of the Nolluppdrag that NollK has created thus far, and in the left upper corner the user can see its logo, name, and email, see figure 4. If there are not any Nolluppdrag created yet the table is empty. The table is sortable by clicking on the respective table title, which then sorts it by ascending, descending, or back to the default order. For clarity, an arrow is shown beside the pressed title to show the order, if it is back to default there is no arrow.

The screenshot shows the main page for a user logged in as a Nollk. The sidebar on the left has a user profile (Emil Lindblad, uppdrag.nollkit@chalmers.it) and menu items: Mina Nolluppdrag (highlighted in blue), Årets Nolluppdrag, Arkiv, and Mitt konto. A yellow 'Logga ut' button is at the bottom. The main content area has a search bar 'Sök..' and a table of Nolluppdrag. The table has columns: Namn på uppdrag, Plats, Tid, Privat, and Status. The data rows are as follows:

Namn på uppdrag	Plats	Tid	Privat	Status
Ännu ett uppdrag	Campus	Under MV3	false	DRAFT
BriTney Spears	Campus	Mottagningen	false	APPROVED
Hängigt uppdrag	Din mamma	12:01	false	DRAFT
Komma på dessa fake uppdrag	Satt hemma i soffan	22:13:30 2023-01-31	false	SUBMITTED
foo	bar	123	false	APPROVED
Woop	lkjasfd	asd	false	APPROVED
Hej mk pls accept	Chabo	Någon gång	false	RETURN

Figure 4: Main page logged in as a Nollk

To create a new Nollupdrag there is a button in the bottom right corner to click, and the user gets redirected to a new page, see figure 5. Here there the user is met by compulsory fields which need to be filled, such as title of the Nollupdrag, time, and place and etc. All of these fields need to be filled out to submit it. When done the user can either save or submit it, saving means it is not yet sent in but just a draft, while the submitted one is sent in pending for a review by MK.

Figure 5: Creating a new Nollupdrag

No matter if it is just a draft or submitted, it is added to the table on the main page. Here the user can either click it and edit the Nollupdrag, or see the status of the review (awaiting review, denied, a return, approved). The checkbox private/public allows the user to choose if the Nollupdrag is to be shown for other NollKs or not, if it is public it will be shown for others in the archive *Chalmers nollupdrag* to the left. There are two archives here, *Arkiv* and *Chalmers nollupdrag*, the first one contains all the previous years of Nollupdrag from the logged-in NollK, whilst the other one shows all the approved Nollupdrag from different NollK:n this year, if they are set to public.

3.2 MK account

When logged in as MK the user comes to the *Granska* page, which is almost the same page as for Nollk:n, but instead of met by a table of created Nollupdrag there is a table of Nollupdrag pending a review, see figure 6. This table is also sortable.



Figure 6: Main page logged in as admin, MK

To review a Nolluppdrag the user clicks on a title and gets redirected to the review page, figure 7. Here all the information about the Nolluppdrag is shown. MK can leave a comment, and also decide to deny, accept or ask for a return. A return means that it is not accepted but not denied either, it needs some more work, editing, and then a re-submission.

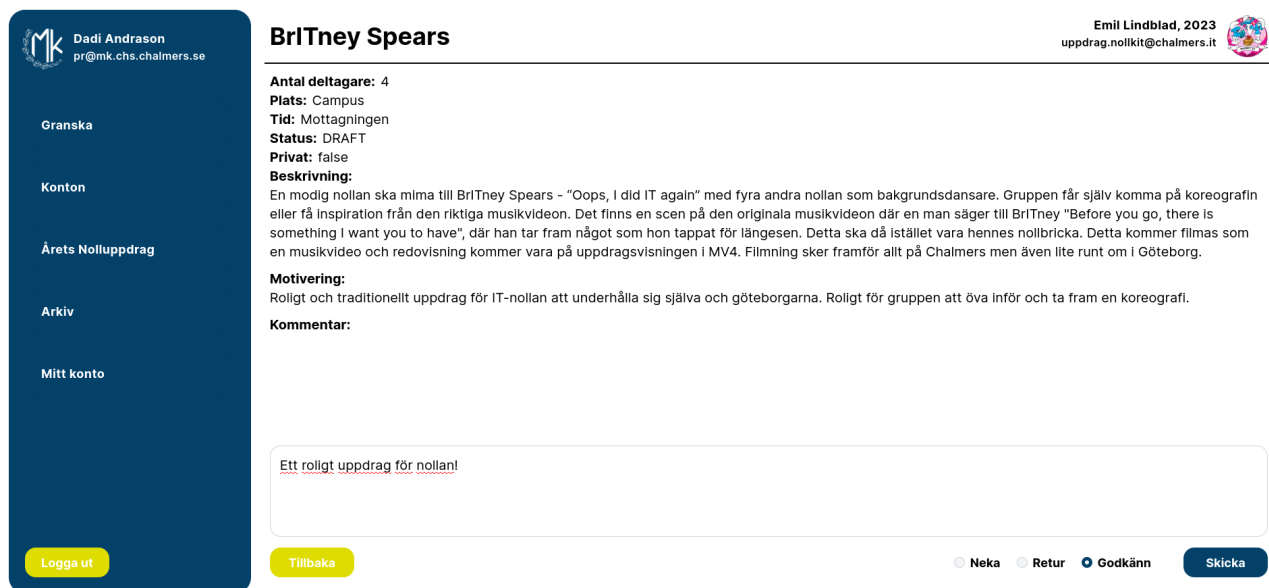



Figure 7: MK reviewing a Nolluppdrag

When logged in as MK, administration of existing and pending accounts is possible, see figure 8. Here MK can delete existing accounts and accept or deny pending account requests.



Dadi Andrason

pr@mk.chs.chalmers.se

Granska

Konton

Årets Nolluppdrag

Arkiv

Mitt konto

Logga ut

Email	Name	NollIK	Accept?
mail@chalmers.elektro	Kalle Anka	EØK	false <div>NoYes</div>

Email	Name	NollIK	Delete?
uppdrag.nollkit@chalmers.it	Emil Lindblad	NollKIT	Delete
uppdrag@dnollk.se	Hacke Hackspett	DNollK	Delete

Figure 8: Manage accounts as MK/admin

4 Design

The tech-stack for Uppdragsportalen is a bit different than the one taught in the course, instead the T3-stack is used. We chose to use this stack since it uses libraries that we like and want to learn more about.

4.1 T3-Stack

The T3-stack has a big focus on typesafety across the whole stack, from database to the frontend. It also provides a cli for initializing a project and lets you pick and choose which libraries you want. All of the including libraries are used, which are the following:

4.1.1 TypeScript

TypeScript is just JavaScript but better. The "strictness" that comes from static typing helps with auto-completion or warns the developer about unsafe calls or when trying pass a value of the wrong type.

4.1.2 Next.js

Next.js is a React framework for creating full stack applications. Next.js provides "backend" like features such as file system routing, server side rendering and methods for data fetching.

Server side rendering means that the react app is rendered on the server before being sent to the user, allowing for less, or sometimes no javascript being sent and run on the client. This decreases load time and increases performance of the application. Next.js allows for high modality and makes it simple to add additional libraries to the application.

4.1.3 TailwindCSS

For styling, the T3-Stack uses TailwindCSS. Compared to other CSS libraries it does not feature predefined components or opinionated styles. Instead it provides pre-defined classes for you to design your interface from scratch. You have full control of the CSS being applied. The library also has dynamic classes which means that a chosen value can be inserted in the HTML-dom for a specific styling such as width. Furthermore, custom classes can be created in tailwinds configuration to be used in unison.

4.1.4 tRPC

tRPC is a library that allows us to create typesafe API's without using a generated schema. Since TypeScript is used throughout the stack, tRPC can share types directly between the client and server.

T3-Stack provides all the boilerplate necessary and also neatly integrates authentication and database functionality, which allows us to quickly create API endpoints for any type of data needed in the application.

4.1.5 Prisma

Prisma is an Object Relational Mapper (ORM) for Node.js applications. The main component is the Prisma schema, where you define your database models and relations. Prisma can then take this schema and apply it to a database, regardless of the type, PostgreSQL, MySQL, MongoDB etc (with a few exceptions). This makes it easy to switch out the database provider.

When used with TypeScript, the Prisma client it provides a typesafe api for interacting with a database. Using the same types defined in the Prisma Schema. Lastly Prisma comes with Prisma Studio, a gui for interacting with your data.

4.1.6 NextAuth.js

NextAuth is an authentication library which makes it easy to implement authentication from many different providers, like Google, Github, Discord or just a simple email and password.

4.1.7 Zod

Zod is a schema validator. Which is used for validating that data is of the correct type, length etc. This validation is used for all data passed to the backend, such as data from form inputs or data passed to database queries.

4.2 Other technologies

By design, the libraries included in the T3-stack cant solve all problems and the maintainers encourage developers to add libraries they need. In addition to the technologies provided by the T3-Stack, the following technologies were used:

- React Hook Form - For creating forms with validation.
- Vitest - For creating unit tests, for testing tRPC endpoints.
- Playwright - For creating end-to-end (e2e) tests. For testing the frontend.
- CockroachDB - For hosting our database.
- Vercel - Hosting and Continuous Integration.

4.3 Routing

Since Next.js uses file-system routing, there is no need to manually create a router with handlers. Simply adding a file in the `pages` directory creates a new route. Nested routes are created using nested folders, the folder structure `pages/foo/bar/hello.tsx` would result in `/foo/bar/hello` as the route.

Dynamic routes can be defined by using square brackets (`[]`) in the filename. This is used for routes that view or edit an uppdrag. The route `/uppdrag/viewuppdrag/clf15sr7a000jxevorbxg0axf` gets handled by the file located at `/pages/uppdrag/viewuppdrag/[id].tsx`. The id supplied in the request can then be accessed in the file via the `useRouter` hook, provided by Next.js.

`src/pages/uppdrag/viewuppdrag/[id].tsx`

```
1  import { useRouter } from "next/router";
2  ...
3  const ViewUppdrag: NextPage = () => {
4    const router = useRouter();
5    const { id: queryId } = router.query;
6    const { data: uppdrag } = api.uppdrag.getById.useQuery({id: queryId as string});
7    ...
8  };
9  export default ViewUppdrag;
```

Listing 1: A code snippet showing how query parameters are used in a dynamic route

The `useRouter` hook is imported and the query is assigned to `queryId`, which is then used to fetch the desired uppdrag using the query on line 6.

4.4 API design

The equivalent of an API endpoint in tRPC is called a router, a router then consist of so called procedures which is essentially a backend function. Our router is contained at `/src/server/api/root.ts` which features sub-routers for uppdrag and users which are defined in `/src/server/api/routers`.

```

src/server/api/routers/uppdragrouter.ts
1  import { z } from "zod";
2  ...
3  import { createTRPCRouter, protectedProcedure } from "../trpc";
4
5  export const uppdragrouter = createTRPCRouter({
6    ...
7    /**
8     * Gets an array of assignments by year
9     */
10   getByYear: protectedProcedure
11     .input(z.object({ year: z.number() }))
12     .query(({ ctx, input }) => {
13       return ctx.prisma.uppdrag.findMany({
14         where: { year: input.year, private: false, status: 'APPROVED' }
15       });
16     }),
17   ...
18   /**
19    * Deletes an assignment based on its Id
20    */
21   delete: protectedProcedure
22     .input(z.object({ id: z.string() }))
23     .mutation(({ ctx, input }) => {
24       return ctx.prisma.uppdrag.delete({
25         where: { id: input.id }
26       })
27     }),
28   ...
29 });

```

Listing 2: A snippet showing two procedures in the uppdrag router.

Listing 4 contains code for two procedures, the first one gets uppdrag based on different parameters and the second one deletes a uppdrag that matches the id supplied. These procedures are of the type **protectedProcedure** which simply means that there needs to be an active session i.e you must be authenticated, in order for the procedure to run. This check is done in a tRPC middleware, which runs before a procedure. The use of zod for the input validation in `.input()` is also implemented as a middleware.

In `.input()` you can specify a zod schema that validates the input to the procedure. So in `getByYear` the input should be an object with the key `year` and its value must be a number. A difference between the two procedures in Listing 4 is that one is defined as a query while the other one is a mutation. The difference of between these is most obvious on the client:

4.4.1 Queries

Queries are used for data fetching. To access our tRPC procedures the entry point provided by the T3-stack's implementation of tRPC is imported to the page.

```

src/server/api/routers/uppdragrouter.ts
1  import { api } from "../utils/api";
2  ...
3  export const HomePageSkeleton = (props: HomeProps) => {
4  ...
5  const { data : chalmersData, refetch : refetchChalmers } =
    ↪ api.uppdrag.getByYear.useQuery({year: 2023},{
6    enabled: false
7  });

```

Listing 3: A snippet showing two procedures in the uppdrag router.

The desired router and which procedure to access is then specified, since `getByYear` is a query the `useQuery` hook is used and parameters are supplied, which follow our schema defined in the backend. In this example the option `enabled: false` is also passed which means that the query does not run unless the `refetch` function is called, defined as `refetchChalmers()` in this case. The data returned is stored in `chalmersData`. For this procedure `chalmersData` is of the type `Uppdrag[] | undefined`. Meaning an array of `Uppdrag` (which is defined in our prisma schema) or undefined if the query is not finished.

4.4.2 Mutations

Mutations are used for changing data. The `useMutation` hook is used with these procedures.

```

src/pages/uppdrag/edituppdrag/[id].tsx
1  ...
2  const deleteUppdrag = api.uppdrag.delete.useMutation({
3    ...
4  });
5  ...
6  deleteUppdrag.mutate({id: id as string});
7  ...

```

Listing 4: A snippet showing two procedures in the uppdrag router.

First the procedure is defined in `deleteUppdrag` using the `useMutation` hook. The mutation is then executed by calling `.mutate()` with the desired data as a paramater, which again must follow the schema defined in the backend.

4.4.3 Procedure context

Going back to Listing 4 the query/mutation has an object with `ctx` and `input` as paramaters. Input is the input from the frontend and `ctx` is the so called context of the procedure. In short, the context is something that is avialable in every procedure. In a T3-app, the context contains the current session and a prisma client, for interacting with the database.

On row 14, we use the prisma client provided by `ctx.prisma.uppdrag.findMany()` to find all `Uppdrag` that match the year specified by the frontend, it should not be private and the status should be `'APPROVED'`.

5 Responsibilities

Figma and Trello were used to have good communication and teamwork. In Figma, the layout were designed where everyone could work together and come with ideas. Trello is a project management tool where assign-

ments are put in different cards, if a member felt like taking an assignment the member assigns the card to them. This method worked well for all members since everyone was familiar with the work process.

Below is a more defined description of which members were responsible for which parts of the project.

Emil Lindblad

- Uppdrag functionality, creation, edit, deletion etc.
- Front-end for uppdrag.
- NextAuth Authentication, sessions.
- tRPC.
- Front-end and validation for forms through out the app.
- Database setup.
- Initial testing setup
- CI/CD

Dadi Andrason

Everything that has to do with any user functionality:

- NextAuth Authentication
- Front-end
- Register, login etc.
- Back-end tRPC procedures for user
- Admin functionality and front-end
- Back-end Testing for user procedures

Generally:

- Middleware for entire web application

Steffanie Kristiansson

- Front-end responsible
- The table of uppdrag
- Sort table of uppdrag (both front- and backend)
- The searchbar (both back- and frontend)
- Frontend tests

Julia Böckert

- Front-end responsible
- Some back-end work in routers
- The side menu
- Structuring the home pages
- Front-end tests