DAT076 - Report - Group 6

Alen Mukaca - Emil Berzelius
Leonard Bagiu - Tor Falkenberg Gunberg

Introduction

This project aims to reproduce and upgrade the already existing site https://htek.se/. Htek is the website of a Chalmers student union and provides information about what the student union is up to. This includes contact information, information about the branches within the union, an event calendar, important documents and what to think of if you've been accepted to chalmers.

One of the main features which htek.se does not have in its current form is the ability to log in by several users and with this, add and remove events from an event calendar.
This has been one of the main tasks to achieve with this project. The other major difference between the current htek.se and the new implementation is that htek.se is powered by wordpress, with automated admin functionality. In the new implementation however admin controls need to be added manually, allowing control to edit most of the text and images displayed on the site.

Another focus of this project has been to improve the user interface and user experience of the website, both on desktop and on mobile devices.

Use cases

- A student can read about the organization, education programs and find various contact info.
- A student can access the various student committees and associations that are dynamically updated.
- An admin can update, add or delete student associations that are shown on the site.
- An admin can add, edit or remove accounts that can access the admin panel.
- An admin can edit members of the Student Union board that will be shown under the Styret subpage.
- Associations accounts and admins can access the admin panel and have different permissions.
- A student can read about board members (Styret) and the page is dynamically updated.
- The students can find a calendar that is dynamically updated and they can click on it to view available events for that day.
- An admin or association account can add, edit or delete events from the admin panel.

User manual

When the project is production ready then it will be accessible at htek.se however in the meantime the project can be downloaded from our Github repository and be built from there. Or download pre-built versions from the Releases page on the repository.

Building the website and accessing it

To build the project you need to clone the project using the following command *git clone https://github.com/ShinzenATT/DAT076-project.git* and afterwards you build the *client* folder and *server* folder by themselves. To be able to build you need to have Node.js and NPM installed on your system.

Building Client

First you open a command line and navigate into the client folder (*cd client*). Then you need to install the project dependencies with the command *npm install --include=dev* and then you can build with the command *npm run build*. Afterwards the files can be hosted like any other site using a tool like NGINX or Apache2 but if you want to preview the result then you can use the command *npm run preview* and then go to the url http://localhost:4173/ in your browser.

Building Server

First you open a command line and navigate into the server folder (*cd server*). Then you need to set up a PostgreSQL database. It can be done by either downloading it from their web page and running the *sever.sql* found in the db folder with the command *psql -f db/server.sql* or you can use docker to setup a finished database that way with the command *docker compose up*. Afterwards you need to install the project dependencies with the command *npm install --include=dev* and then you can build with the command *npm run build*. The finished project will then appear in a newly created *dist* directory where you can run the server by running the *node* command in the *dist* directory.

Using the website

When entering the website then the user is greeted to a long landing page where the user can get an overview of the contents of the website by scrolling. Here the user can view the programs available in the division and click the icons to read further on Chalmers website.

Further down there's a calendar which shows dots on days that contain events. When the user clicks on a day then it will show a pop up that shows the event details such as title, time and location.

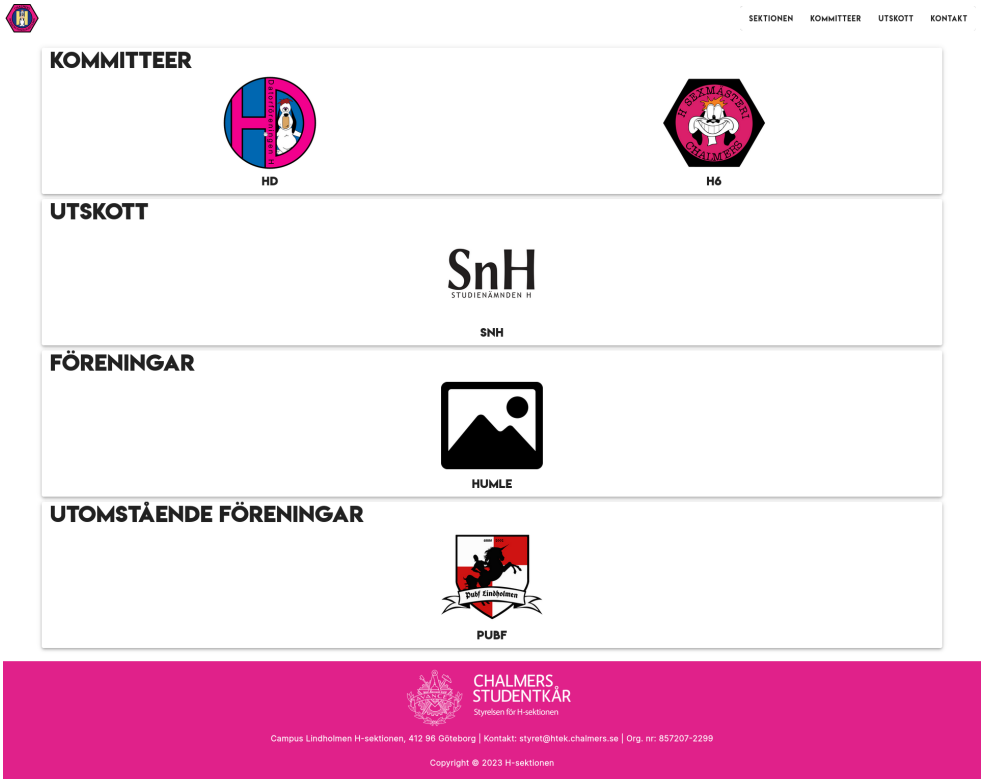
Afterwards come a preview of the available committees and student associations that the user can click on to read more.

Then there's the navigation bar and footer which appears on every page. The navigation bar contains links to every page and every top level button opens a dropdown with additional links. The "Kommittee" and "Utskott" dropdowns are dynamically updated as well and may change over time. As for the Footer it displays contact and other identifying information about the organization.

The Landing page of the website, *The Navigation bar dropdowns in desktop and mobile view*

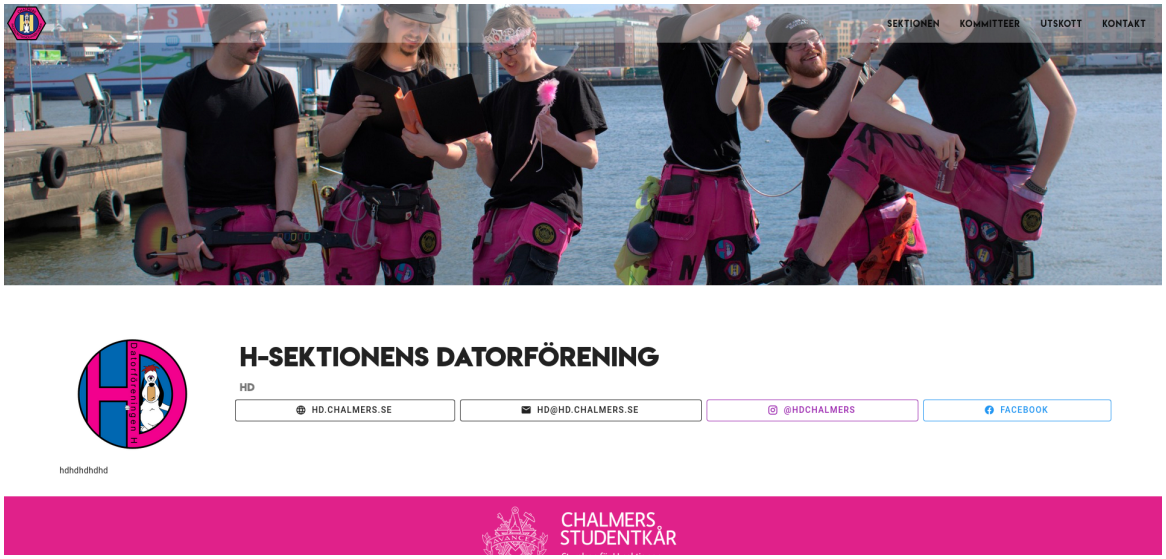
Viewing Committees

To view info about various student associations or committes you can find them by scrolling on the homepage, the navigation bar dropdown or the committees list page and clicking their icon.



The Committees List Page accessed by clicking Sektionen -> Föreningslivet in the navbar

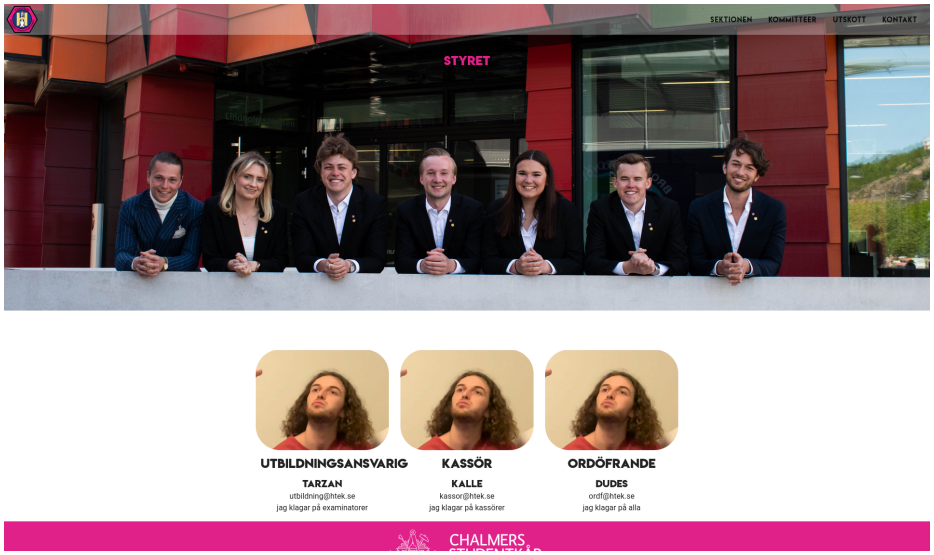
Afterwards you enter the info page of the clicked committee where you can read their description and access their social media links.



The Committee Info page accessed by clicking their icon

About H

There are various informational pages for the functions within or contact info as well. Such as the organization board and program team.



The Styret page accessed by clicking Sektionen -> Styret in the navbar

About page accessed by clicking Sektionen -> Om H

SEKTIONEN KOMMITTEER UTSKOTT KONTAKT

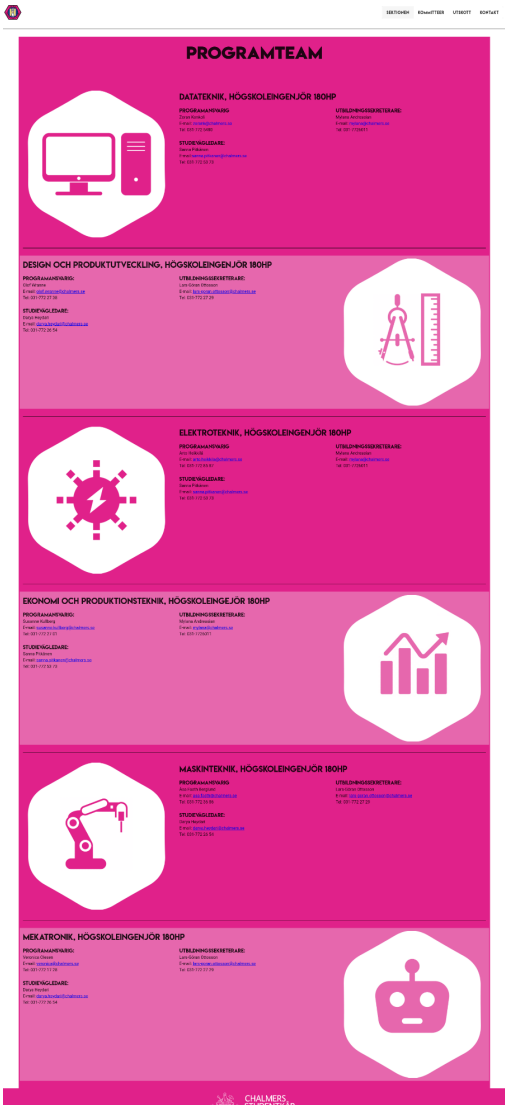
Studerande arbetsmiljöombud (SAMO)

Jag heter Moa Sidhagen och läser Design och produktutveckling år 3. Det är jag som är SAMO här på H-sektionen och är därmed din kontaktperson i alla arbetsmiljöfrågor, både fysiska och studiesociala.

Jag sitter i H-styret och jobbar för att du som student på H-sektionen ska trivas så bra som möjligt på din utbildning. Vet du inte vart du ska vända dig i en fråga så kan du alltid kontakta mig, så hänvisar jag dig rätt.

Bra att veta är också att jag har tystnadsplikt, så tveka inte att kontakta mig om du funderar över något.

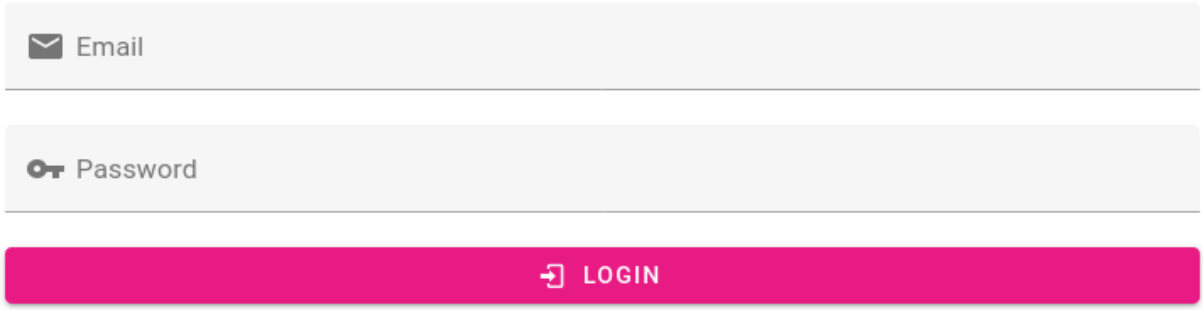
Samo page accessed by clicking Sektionen -> SAMO



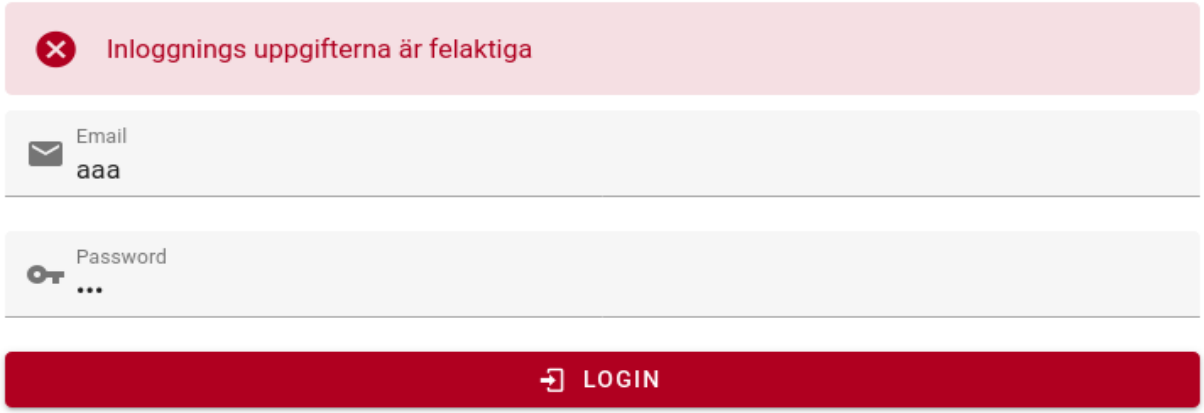
The Program Contact page accessed by clicking Kontakt -> Programteam

Admin Panel

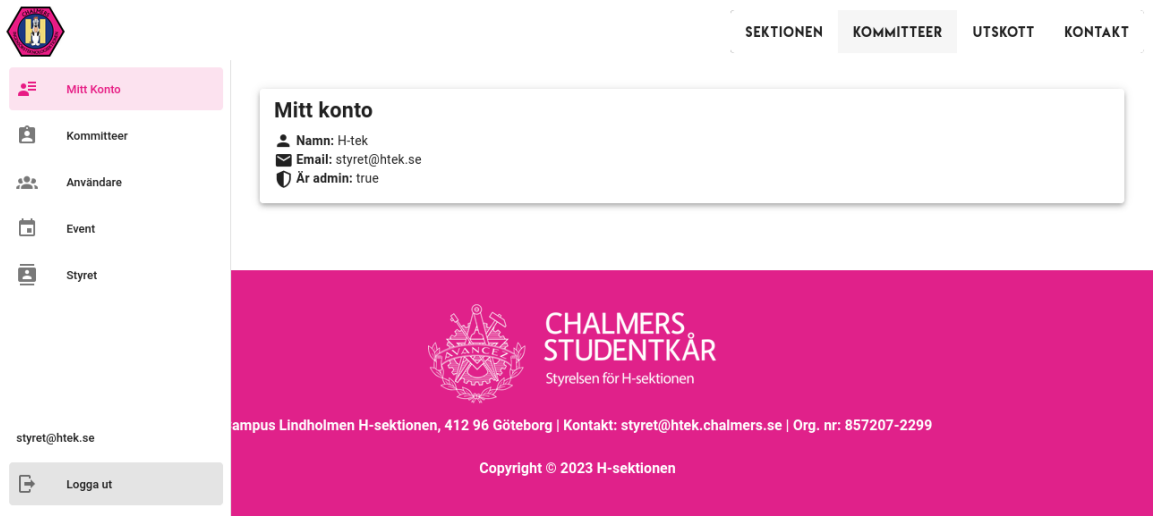
The website has an admin panel that can be accessed by appending */login* to the website domain and logging in with an account. Once logged in the user is presented with a My Account page that shows user information. In addition, there's a sidebar to the right with links to various admin pages to edit site contents such as accounts, committees, events and styret.



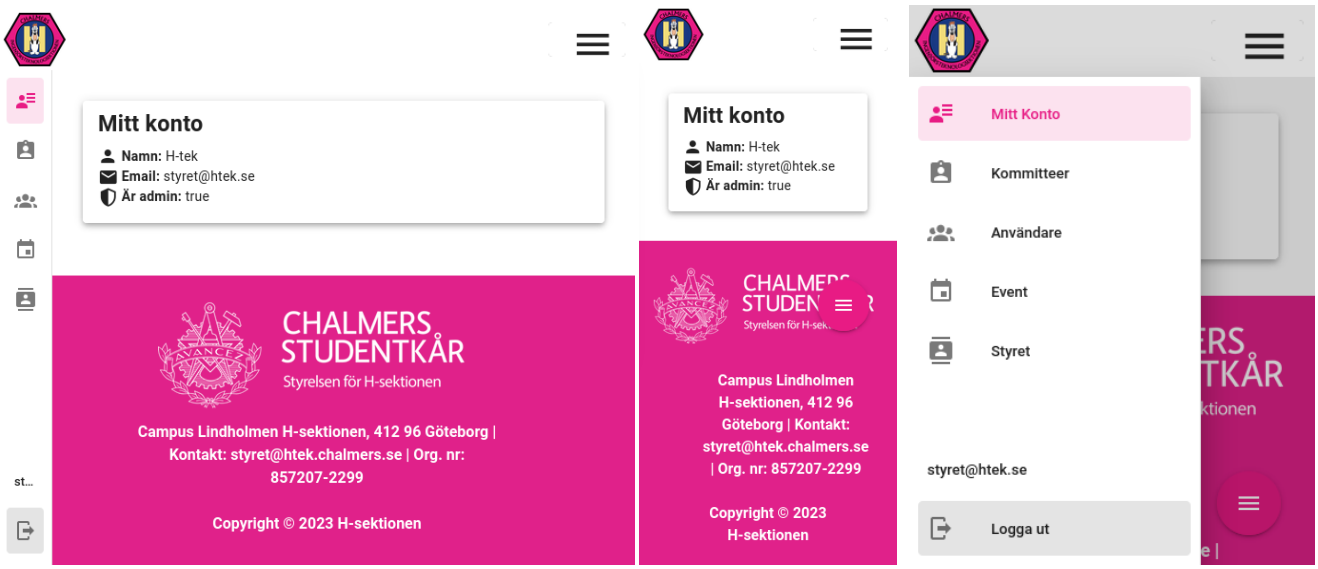
The sign in page



The sign in page when the user inputs wrongful credentials



The My Account page in desktop view



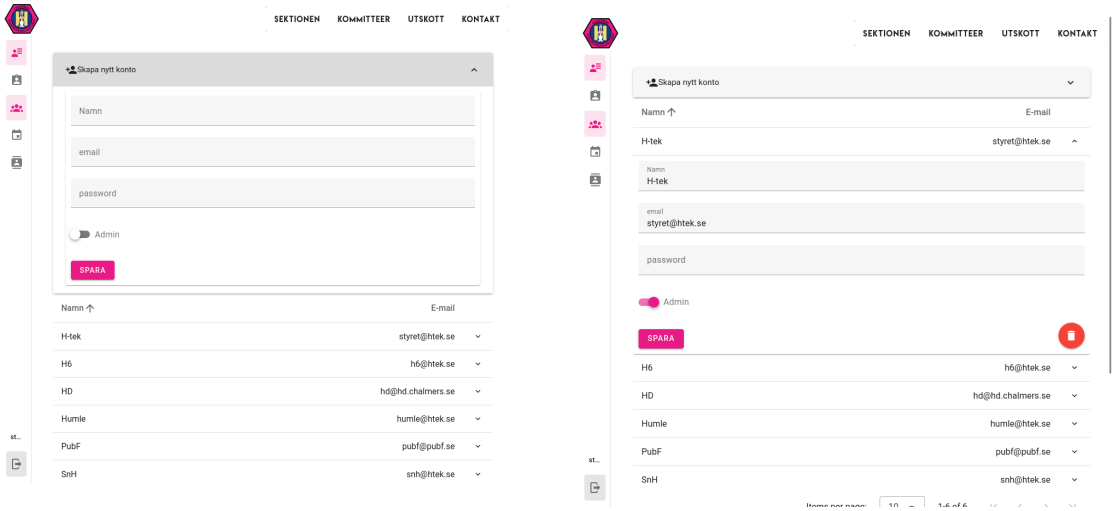
My account page in tablet view and Mobile view with navigation drawer hidden and expanded

The admin pages share several concepts when interacting with it. The navigation drawer will collapse and hide on smaller screen sizes but it can be expanded again by hovering it or clicking the menu button that appears in the bottom right on mobile.

When interacting with admin pages then the user is faced with a table that contains existing entries in the database. To add a new entry then there's a button at the top that expands into a form that can be filled. While if an existing entry is to be edited then the user can click on the arrow to the right of a table row to expand it. There the entry can then be edited or deleted. When saving the button will show a loading animation and then become green for success or red when an error has occurred.

Account admin page

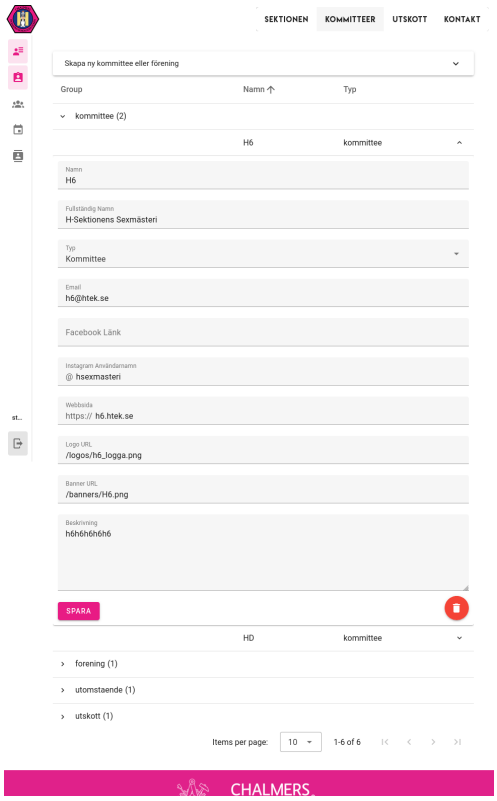
Used for managing accounts that can sign into the admin panel. No form inputs are optional.



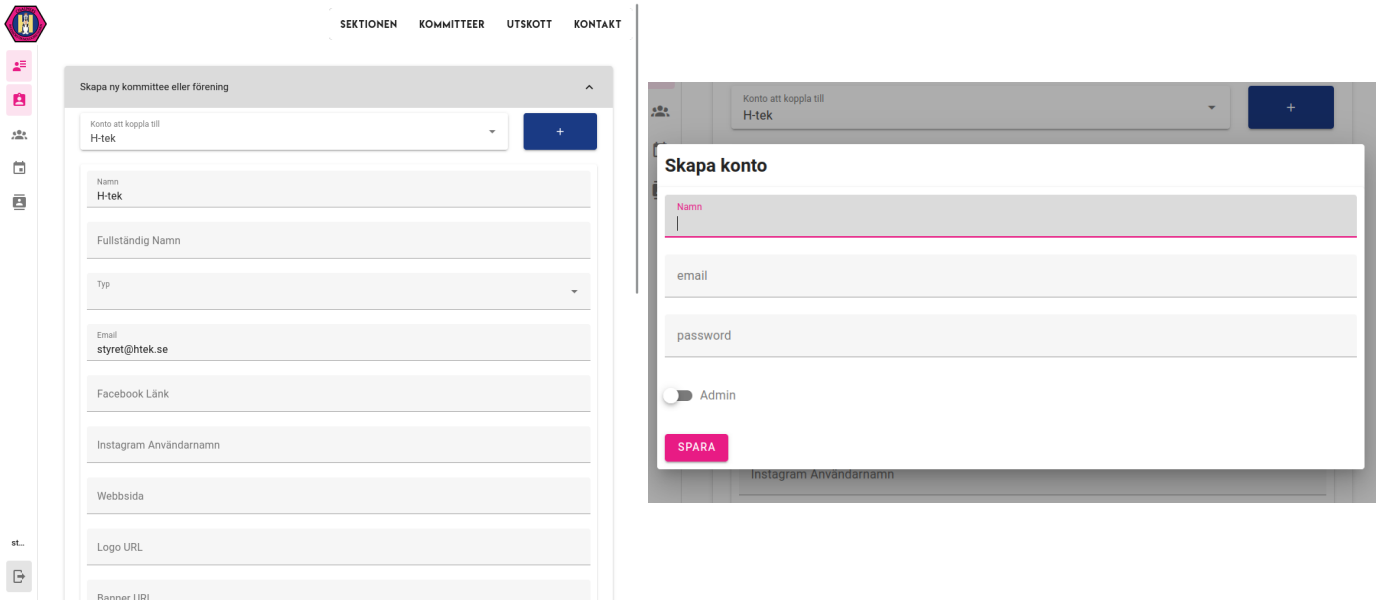
The Admin Account page with top area expanded and a table row expanded

Committee Admin Page

Used for managing committees that appear on the website. Each committee needs to be connected to an account and have a type selected while all other values are optional.



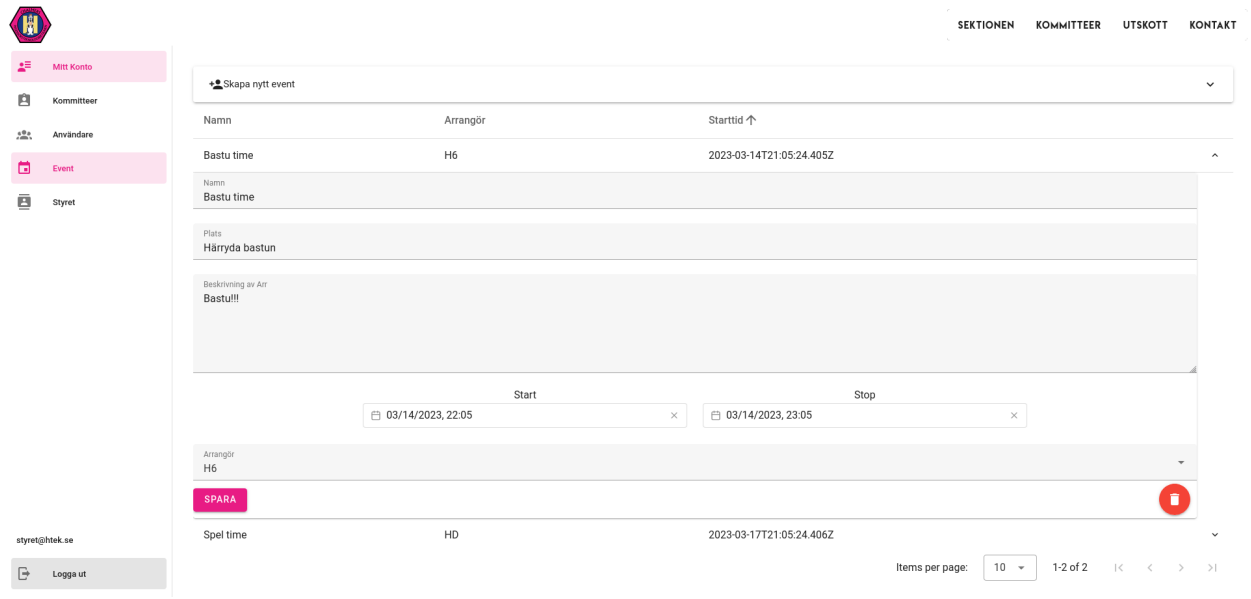
The Committees admin page with a table row expanded



The Committees admin page with top item expanded and one with the create account dialog (+ clicked)

Event Admin Page

Used for managing events that appear on the calendar. When creating an event then an organizer (committee) needs to be chosen for the event and the fields name, location, start and end needs to be filled.



The Event Admin page with a table row expanded

Styret Admin Page

Used for editing board member information available on the site. All fields other than imagepath are required.

The Styret Admin page with a table row expanded

Design

The framework used to design this application is Vue.Js with Vite rather than React, which is recommended in the project description. While the backend uses Express.js to host a HTTP JSON REST API. The project is split up into a client- and server part, which will be explained in two separate sub-sections.

Server

The server folder consists of the HTTP API backend which uses Express.js to host the API. The codebase follows a MVC and microservice architecture to organize the code. Each folder specified in the server-directory will be specified with a separate documentation below.

Database (DB)

The server of this application has been set up towards a PostgreSQL database through Docker. In order for this to work, database.ts holds the port, username, password and host url. Furthermore, all database objects to be used within the application are specified within the export of the database.ts file.

In order to run with Docker, a dockerfile is set up for targeting purposes when running “docker compose up” within a terminal. The backend uses the @databases package which allows for ORM interaction and typescript type checking with the database. The package provides a CLI that when run the database is analyzed, and the “generated” folder becomes available with typescript objects of the database tables which is the primary way of communicating with the database.

Model

Every database-object is specified within the model, with the purpose of exporting data-types of the database for writing consistent data for every HTTP request. Furthermore, some fields such as id are specified as read only to let the assignment of the value to the database whenever insertion occurs.

Another optional field within some model-objects is to serialize the data, allowing for a specified object to be inserted into HTTP-functions for further type safety.

Service

Services are classes that define how the application communicates with the database. First off, every service holds an interface defining at least a get, edit and delete function to be used within

the class with its respective signature. Furthermore, the service class connects the database with the requirements specified within the model class for the same database object.

The get functions of the service classes either accept name as a string, or an id as a number and give a promise with the services model object.The promise is used with asynchronous functions, and represents the eventual completion of the function. When calling get, the function fetches a matching object, if it exists, from the database and returns the object specified from the model. Similarly, some services have the option of fetching the entire database, which is useful for an admin to be able to edit the information. Then an array of objects is then returned, masking sensitive data (such as passwords).

Edit works in a similar fashion to get, but instead takes a whole object as input data with a promise as a return value. The edit functions then overwrite an already existing value in the database, and throws an error if it does not.

When calling delete, an id as a specifier is required. The function then calls for the database to delete the object, and does not hold a return value.

Router

Within the router classes, express is used for responding to HTTP-requests using the respective service classes for the appropriate database-related function. Every router has functionality for GET, PUT, DELETE or POST with specifications for how the header should be declared for the function to be called. Each function in a router wraps a service call which converts parameters, returned values and errors into appropriate types and HTTP responses. For instance, if a service throws due to incompatible parameter objects then the router will respond with a status code of 400 and the error text in the body.

Root files

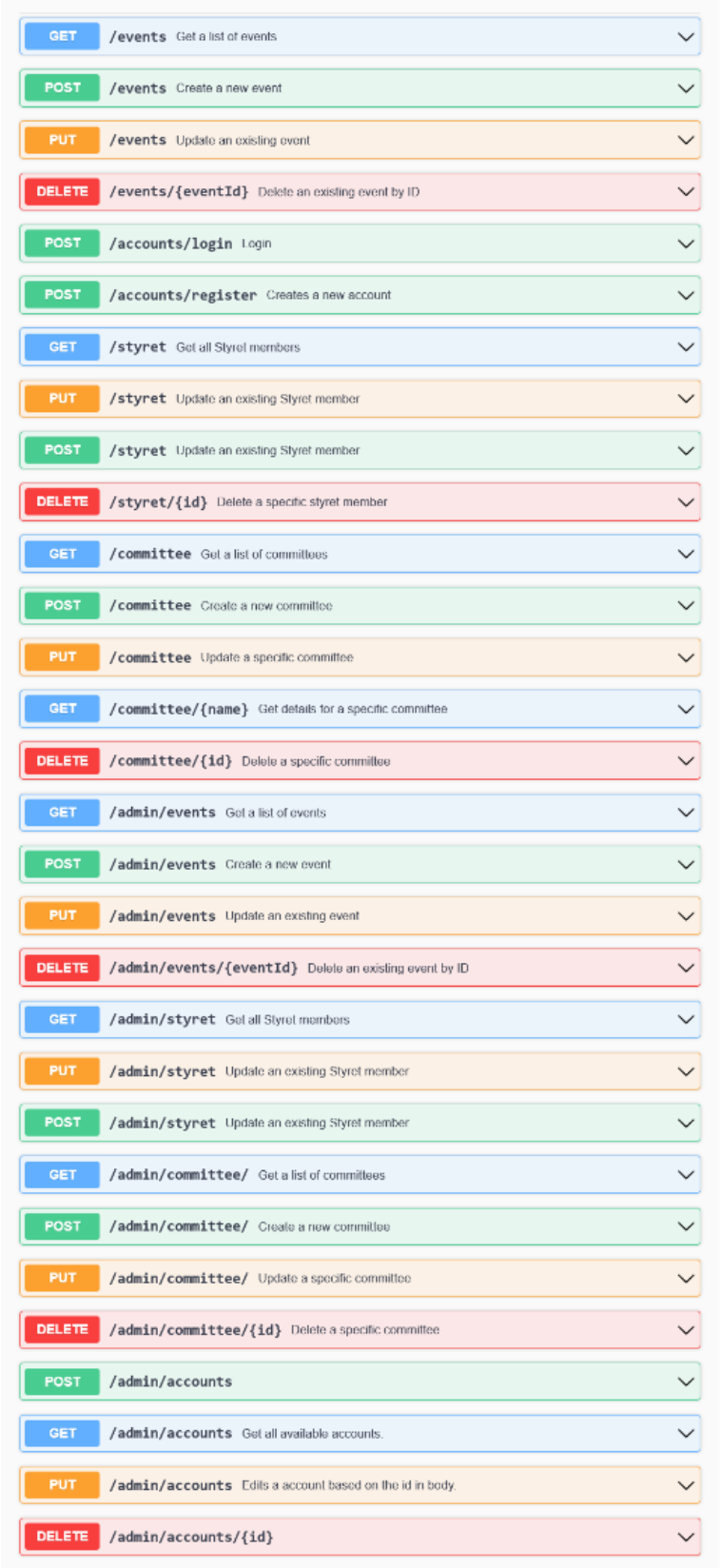
These files set up the server for instance the *start.ts* file registers all the routers and the *index.ts* file starts the Express server.

Test files

The project uses the Jest testing framework to test various services and routes. Therefore each test suite will share a similar filename to the service or code it tests such as *eventService.test.ts*. The test can be run with the following command, *npm test*.

HTTP API Specification

There's an OpenAPI specification available in the file *api-spec.yml* and can be previewed with swagger UI. It can be viewed by using the repository file with the following link otherwise it's available under */spec* route when hosting API server. The UI can execute HTTP requests for testing and experimentation if the API server is running on the user's system.



A list of the available API routes in Swagger UI

Client

The client has several directories allocated to different purposes. The files placed directly under /client hold the purpose of application configuration, which will be listed below:

- Tsconfig.json - configuration for how typescript should be run within the application
- Tsconfig.node.json - configuration for the compiler of node with TypeScript
- Vite.config.ts - Config the Vite tool that builds and hosts the project
- Package.json - name of the application, required scripts, dependencies and developer dependencies used by npm.
- Package-lock.json - The dependencies of the project dependencies
- .eslintrc.js - Lint configuration for Vue3

Each directory directly under the will be specified with its own sub-section.

Types

Each file holds specifications for the database-objects from the model as interfaces. These are used within scripts that send updates through the router to assert that a correct data type has been used.

Public

Any data that should be left available for a user to see with the possibility of being interplaced. Any files within the public folder are targetable by database-related functions.

Src

This is the largest of the folder, and holds all .vue frontend files. All frontend within the application has been based from two layouts, where Default.vue within the “layouts” folder has the general layout of the page, importing the header and footer, and links to the router. Vue uses components to link parent and child files within the application, of which header and footer are good examples. Any component can be an independent file, or have children of its own. The other layout available is admin.vue, which is only accessible through having an account marked “true” as admin in the database.

The currently selected route is linked to Default.vue, which can be found under the “router” folder. Index.ts within holds all possible routes of the application, with all routes being specified to its own separate component. The admin-route has several subcomponents, as they should only be accessed if the login credentials are set as admin.

As mentioned, each route corresponds to a specific view. Some views are made static, whilst other views use components to make the page more dynamic. If the parent component holds data that must be passed to a child, it’s important to notice that the “v-model” tag is used, which

specifies that the data can be both read and sent. Components are generally used for re-iterable elements that reside within a parent page, such as using a slideshow to show which committees exist in the home-page.
Components are more focused on transmuting objects from the database to create frontend-objects, and thus hold much of the logic of the application. An example would be the header, where the amount of buttons existing in the “kommitteer” and “utskott”-tabs are generated from the amount of objects present in the database. This, of course, corresponds to the routing of the same objects.

Here’s a brief overview of all the folders in src:

Assets

This folder contains static assets that are not dynamically changed by the admin panel such as the site logo and homepage background images.

Components

Contains Vue files that do not represent an entire page rather reusable page parts such as navigation bars, footer, forms and so on.

Helpers

Contains some Typescript classes that handle common functions such as accesses to the browser’s LocalStorage and converting them to appropriate types.

Layouts

Contains Vue files that contain pages meant to be layouts are base files for other pages. This automatically adds the navigation bar and footer to pages for instance. Vue files need to have the router-view tag within them to work properly.

Plugins

Contains plugin configuration for various packages used by the Vue site such as Vuetify.

Router

Contains definitions for Vue router that connects URLs to VUe components/pages.

Views

Contains Vue files meant to be actual pages such as the homepage.

Responsibilities

Alen Mukaca

Worked on both the frontend and the backend. Largely responsible for the pages Committees, Committees Admin, Accounts Admin, login page and Admin layout. As for the backend he worked on the Event service/router, Committee service/router and account service/router. Also made database tables for Events, Accounts, Committees and SessionToken. In addition, he wrote the backend tests, OpenAPI specification and docker image.

Emil Berzlius

Created the homepage, calendar component, event page, styret page, styret service/router and admin router. Did several miscellaneous fixes.

Leonard Bagiu

Largely responsible for the Header and footer component as well as the event admin component. Did additional changes to the styret admin page. Also wrote a portion on the report and did some miscellaneous fixes.

Tor Falkenberg Gunberg

Created the first version of the event class, otherwise mostly frontend work. Created the About-page, SAMO-page and ‘Programteam’-page. Largely responsible for commenting the code, as well as some general code-cleaning.