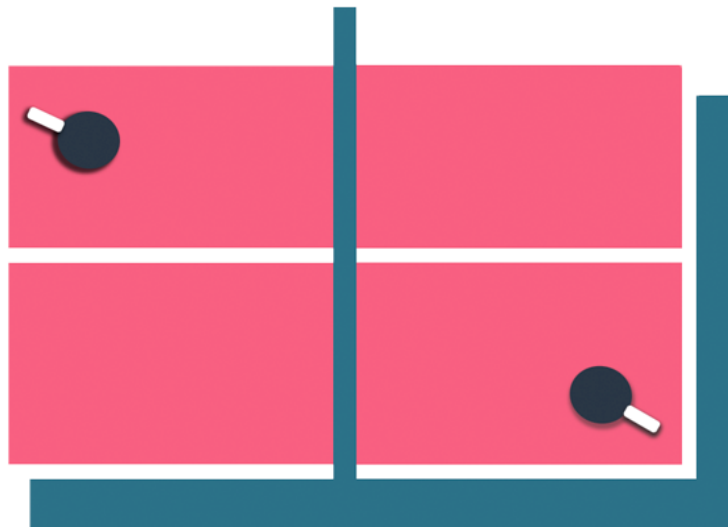


Exam Project
IDG2100 Full-Stack
Table Tennis League App



Created by

Lisa Mari Myrene
Alexandra Eloise Vanje
Anosh Chaudhry

Norwegian University of Science
and Technology Gjøvik,
06.06.2023

Table of contents

1.0 Introduction.....	2
2.0 Development process.....	2
2.1 Collaboration.....	2
2.2 Design.....	3
2.2.1 Accessibility.....	4
2.3 Technology stack.....	4
2.4 Back-end development.....	5
2.4.1 REST API.....	5
2.4.2 Swagger.....	5
2.5 Front-end development.....	6
2.5.1 Context API.....	7
2.5.2 Web Component.....	7
2.5.3 Storybook.....	7
3.0 Product description.....	8
3.1 Table Tennis League App.....	8
3.2 User roles.....	8
3.2.1 Anonymous.....	9
3.2.2 User.....	9
3.2.3 Admin.....	9
3.3 Pages.....	9
3.3.1 Homepage.....	9
3.3.2 Login page / New user page.....	10
3.3.3 Players page.....	10
3.3.4 Favorite players page.....	11
3.3.5 Matches page.....	11
3.3.6 Match page (id).....	13
3.3.7 Profile / Admin page.....	14
3.4 Security.....	16
3.4.1 Backend.....	16
3.4.2 Frontend.....	18
4.0 Reflections.....	19
4.1 Achievements.....	19
4.2 Discussion.....	19
5.0 References.....	21

1.0 Introduction

Table tennis has become a very popular activity on NTNU in recent years, with more players and matches than ever before. Despite this, there has been no way for the players to track their matches and get an overview of their statistics and rank, thus players had no way of knowing how well they were doing on a bigger scale. To keep the initiative going strong it was deemed necessary to develop a web application to do exactly that: keep track of matches with their scores, as well as all the active players. Since there are several matches played every day, the need for a score tracking system at the school was in high demand.

As the Dean gave his approval, this project was developed precisely for that purpose: to solve the problem of all the undocumented matches and players. Table Tennis League App is a web application where you are able to create an account, play a match, and then save the match to the system. You can also visit the web-app without registering if you simply want to get an overview of how many matches have been played, how many registered players there are, as well as the top 5 players in the league.

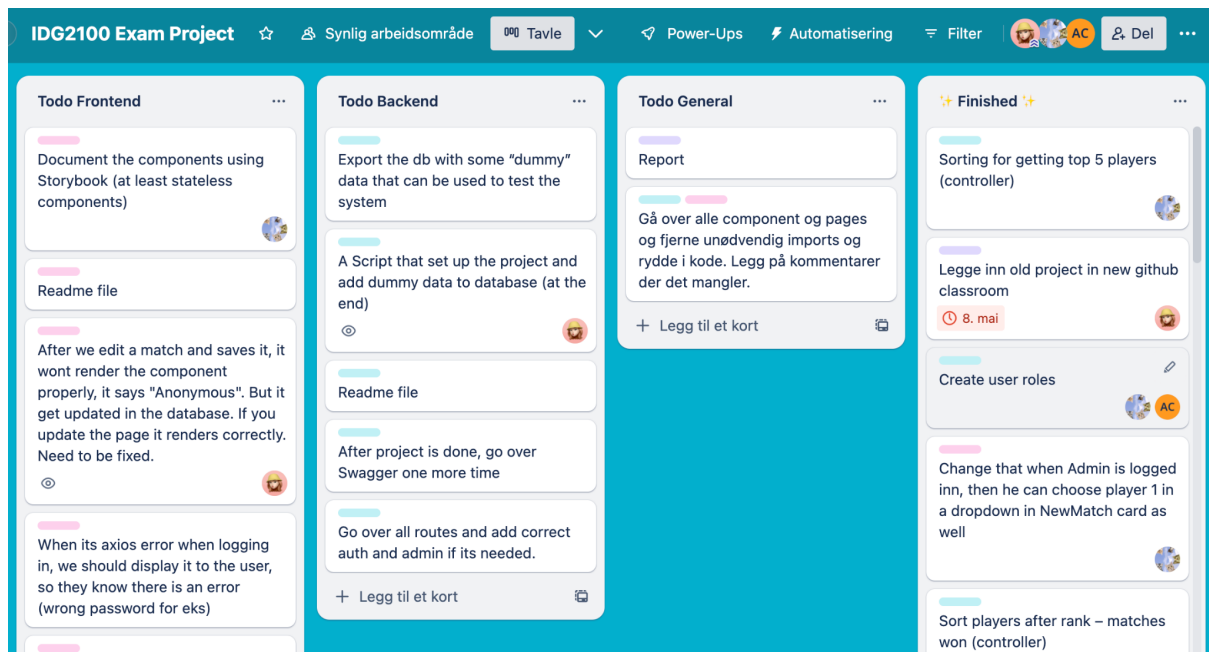
By developing this system, it allows all players to keep track of their rank and wins as well as other statistics. It also allows for admin users to make necessary changes to user accounts and matches in case any wrong information has been given. To summarize, it solves the problem of having untracked matches and players. It also contributes to keeping students motivated to play more Table Tennis in order to improve their scores and get to the top of the leaderboard. This application was designed for the students at NTNU Gjøvik, but it has the possibility to expand to the other universities.

2.0 Development process

2.1 Collaboration

When we received the brief for this project, we chose to break it down into smaller tasks into Trello to get a better overview. Trello is a visual work management tool that helps manage work in an organized way. We therefore created a board which we divided into four different categories, *Frontend*, *Backend*, *General* and *Finished tasks*. In that way we could pick the tasks we wanted to work on, while having an

overview of what each individual member was working on. This made sure no one was working on the same task at the same time.



Screenshot: Trello board

Together as a group, we have had thorough communication via messenger and physical meetings. In the start we had a lot of physical meetings to discuss and be creative together. Later on, when we all had different tasks to work on, we were more free to choose where to work. Even though we had less frequent meetings, we aimed to collect the group for a meeting at least once every week. This worked great for our group, since we had a good communication flow throughout the whole project.

2.2 Design

We designed this app first and foremost with the user in mind. We used Figma to create wireframes and plan our pages. We consistently followed a mobile-first approach to ensure that it would be fully responsive. Knowing our target-group consists of mostly students, having a focus on mobile design was essential. As we value web accessibility highly, we made many of our design decisions with that in mind. We made sure our page was easily navigable with multiple markers to show the user their location, and took advantage of hover states such as using text underlines. We incorporated an "active" state on our navigation bar to show the

user which page they are currently on. We also made sure to have sufficient error messages to the user.

2.2.1 Accessibility

We put a lot of thought into the web application's design, especially the colors. We wanted the application to reflect the fun of playing table tennis. You can see a lot of blues, with a touch of pink and yellow. When choosing colors, we made sure it was inside the range of great contrast, so all users are able to read everything on the application. All the different color combinations in our web application have approved color contrasts, both WCAG AA and AAA.



Screenshot: Color scheme for Table Tennis Web Application

To make sure our web applications were accessible, we used the WAVE web accessibility evaluation tool to check every page on the application. We used this tool throughout the process of developing, which saved us a lot of work in the long run. We are proud to say that the application does not have any errors, and fulfills the WCAG requirements. Since the players would either use the product on either a phone or desktop, we made sure the whole application is fully responsive.

2.3 Technology stack

The MERN stack is a popular JavaScript-based web development framework that consists of MongoDB, Express.js, React, and Node.js, which helps us build full-stack web applications. The backend part of the application is run on Express.js which handles the server-side logic, and communicates with the MongoDB Database. React is used in the frontend part to build the interface and handles all dynamic and responsive web components. Node.js is helping the server-side, allowing execution of JavaScript code (MongoDB, 2023). We have separated the backend part and the frontend part of the project into two different sub folders.

2.4 Back-end development

As the assignment asks for, our React application uses Express.js for the backend, and Node.js to execute JS code. Inside the backend folder, we have multiple sub folders to easily maintain the code. All these sub folders are maintained inside the “project” folder that acts like a parent. These sub folders are separated based on the content inside the files. We have folders for the controllers, routes, schemas, and middlewares. The schema defines the structure of the database, the controllers implement the logic for handling requests and coordinating actions, and the routes map the URLs to the appropriate controller methods.

We also have the “server.js” file which creates the Express.js environment that establishes the connection to the MongoDB database, configures middleware for handling CORS and cookies, and defines the applications routes for handling the different API endpoints.

2.4.1 REST API

The server communicates and handles the requests through REST APIs, by providing a standardized set of rules and conventions. These APIs define the endpoints, methods, and data formats for exchanging information between the client and the server (IBM, 2023). We have divided the main API endpoints into four main routes. “Public”, “Players”, “Matches”, and “Auth”.

2.4.2 Swagger

Swagger is an open-source framework that allows developers to design, build, document and consume RESTful APIs. It refers to a tool called Swagger UI, which is used to document and test APIs, and provides a user-friendly interface that maps out the blueprint for APIs (Semyon, 2023). By using Swagger, it allows us to easily test the endpoints, which are beneficial during the development phase.

Players, Matches, Auth and Refreshtoken API 1.0.0 OAS3

API endpoints for managing players and matches

Servers

http://localhost:5005 - Local server

Players

GET	/api/players	Get all players	⌵
GET	/api/public/playerLength	Get the number of players	⌵
PATCH	/api/players/{id}	Update a player	⌵
DELETE	/api/players/{id}	Delete a player	⌵
GET	api/public/getTopPlayers	Get the top 5 players	⌵
GET	api/players/playersRanked	Get all players sorted by rank	⌵
POST	/api/players/{id}/addFavoritePlayer	Add a favorite player	⌵
DELETE	/api/players/{id}/removeFavoritePlayer	Remove a player from favorites	⌵
GET	/api/players/{id}/getFavoritePlayers	Get favorite players of a player	⌵

Screenshot: Swagger Docs

2.5 Front-end development

As mentioned, the frontend part of the application uses React to build the interfaces, and is where we see everything come to life. The frontend has an organized structure with an easily navigable hierarchy. All pages have separate folders and are all collected into the “pages” parent folder. The same goes for the components, each component has its own folder with the jsx and css file, that are again collected into the “components” parent folder. In addition, we have a “utils” folder where the protected routes (outlet) are set up, and a “providers” folder where the Context API is initialized. We chose to use vanilla CSS in this project, since we continued on the project from oblig 3, where we used ordinary CSS. If we could start over again, we would probably have gone with Tailwind CSS to eliminate the amount of files in the project.

We have used several different tools and modules to build our interface. All the forms are using the useForm hook from React Hook Forms. By using the useForm hook from React Hook Form, we could easily initialize and control the form state, and track the input values. It also allowed us to define validation rules for the form, for example required fields, max length, etc.

2.5.1 Context API

To be able to share data between components, we are using Context API, which provides a way to create a global state that can be accessed and updated by components at different levels of the component tree. Inside the context we have the state of the logged in state of the user, the access token, and the user role. We have wrapped the Context API around App.js, which serves as the main component that will provide the shared states to all the child components within the application.

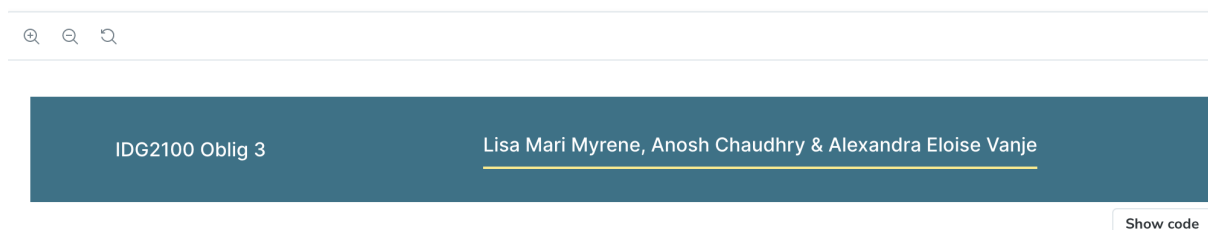
2.5.2 Web Component

We successfully managed to implement the web component from oblig 1 into this project. A JavaScript web component is mainly used to create reusable and encapsulated components for building web applications. It includes custom elements that enables you to define your own custom HTML elements, which can be helpful when you want to extend the HTML vocabulary. It uses a shadow DOM that provides a scoped and isolated DOM subtree, separated from the main DOM (MDN, 2023). When we wanted to listen on the properties in the web component, we now needed to initialize a useRef, which is a React hook that allows you to create mutable reference to the DOM element in a functional component. By using this hook, we are able to access and persist a value across component renders without triggering re-render (React, 2023).

2.5.3 Storybook

Storybook is a front-end “workshop” that can allow you to re-build your components in isolation from your application. For this project we have used it mainly for documentation purposes, with the intention of showcasing some of our components individually. We will elaborate more on our Storybook usage in our reflection notes.

Footer



Screenshot: What a component can look like in Storybook

3.0 Product description

In this chapter we will describe the final product using text, visualizations, screen-shots and code snippets.

3.1 Table Tennis League App

The Table Tennis League App has a wide variety of different functionalities and features. Further in this report, we will go more in detail of the different roles, and what different pages the user gets access to depending on that role. We will also talk about all the pages with its unique components and functions.

3.2 User roles

As required, we incorporated user roles in our application to ensure that the user's access to certain content aligned with what type of user is logged in. We accomplished this by implementing roles in our database players schema. We could access the role in the front-end by decoding the user's access token from the refresh token. By decoding the access token we gained the user ID, and based on that, we could check for that user's role. This was then further saved in a context API, which gave us access to the role in all the components and pages. This allowed us to display certain content within a single component based on the given role.

In the back-end we have a middleware where we created "auth" and "admin" (within `verifyToken.js`) that would respectively verify if a user is authorized to access certain API endpoints, and to check if a user is admin. By doing this we could add "auth" and/or "admin" to our route paths, ensuring that they are safe. We also had protected routes in the front-end, where the user would see different pages based on their role.

3.2.1 Anonymous

The anonymous user has access to the homepage and the login page. As they don't have an account, and therefore are not logged in, they don't have any assigned role. They are then considered "anonymous" by default.

3.2.2 User

Players that have registered an account get assigned the role “user” by default. The user-role has access to more content, such as their own profile, matches page and players page that displays various information.

3.2.3 Admin

There is an API endpoint that allows you to create an admin profile if you are a verified admin. This means there can be one or multiple admins, and they have more content and features accessible to them than any other user. They have the option to edit users, edit matches and more, and all of this admin-only access has been verified both in the back-end as well as the front-end. The page will render additional content on the different pages to the admin, allowing for easy access to make changes directly in the application.

3.3 Pages

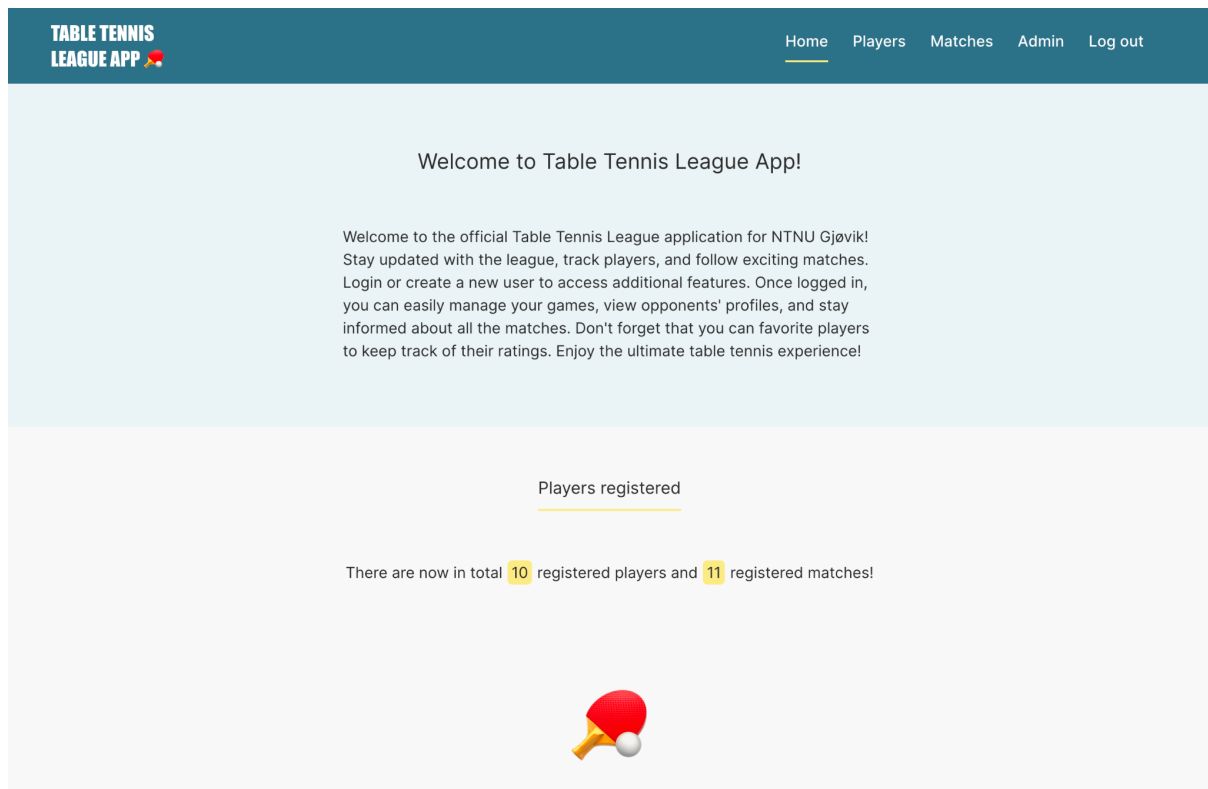
As the assignment requested, we have created several pages that are displaying different content based on the user’s role. The report will further elaborate all the different features that each page in the web application contains.



Screenshot: The navbar in different views

3.3.1 Homepage

In the homepage you will be welcomed to the web application where you can read about the Table Tennis league. If you scroll down you will be able to see how many players there are currently in the league, including the number of matches played. There will also be displayed a table with the current top five ranked players in the league.


















Screenshot: Hero of the homepage






3.3.2 Login page / New user page

If you want to access more features in the application, you will need to log in with email and password. If you are not a user yet, you are able to create a new account. After you have created an account, you will then be redirected to the login page again where you now can login. You will also receive a welcome email, with your user account credentials.

3.3.3 Players page

On the “players” page, you will find all information surrounding the players aspect of the Table Tennis league. The players page is divided into two sub pages, which can be accessed through the sub navigation menu. In the first subpage “Leaderboard”, you will see the leaderboard with all the players in the league. It will display the players full name, belonging institute, matches won in number, and current points. You are also able to favorite players if you want to keep their information at an easy access. If you are logged in as admin, you will have more features available, such as creating new players, editing existing players, or simply delete a player from the league.

Name	Institute	Matches won	Points	Favorite	Edit/Delete
Olivia Brown	Helsevitenskap	4	14	 Favorite	 Edit  Del
Michael Johnson	Vareproduksjon og Byggeteknikk	2	6	 Unfavorite	 Edit  Del
Emily Davis	Design	1	4	 Favorite	 Edit  Del
Emil Bakke	Design	1	3	 Favorite	 Edit  Del
Sophia Anderson	Vareproduksjon og Byggeteknikk	1	3	 Unfavorite	 Edit  Del

Name	Institute	Matches won	Points	Favorite
Olivia Brown	Helsevitenskap	4	14	 Favorite
Michael Johnson	Vareproduksjon og Byggeteknikk	2	6	 Favorite
Emily Davis	Design	1	4	 Unfavorite
Emil Bakke	Design	1	3	 Unfavorite
Sophia Anderson	Vareproduksjon og Byggeteknikk	1	3	 Favorite

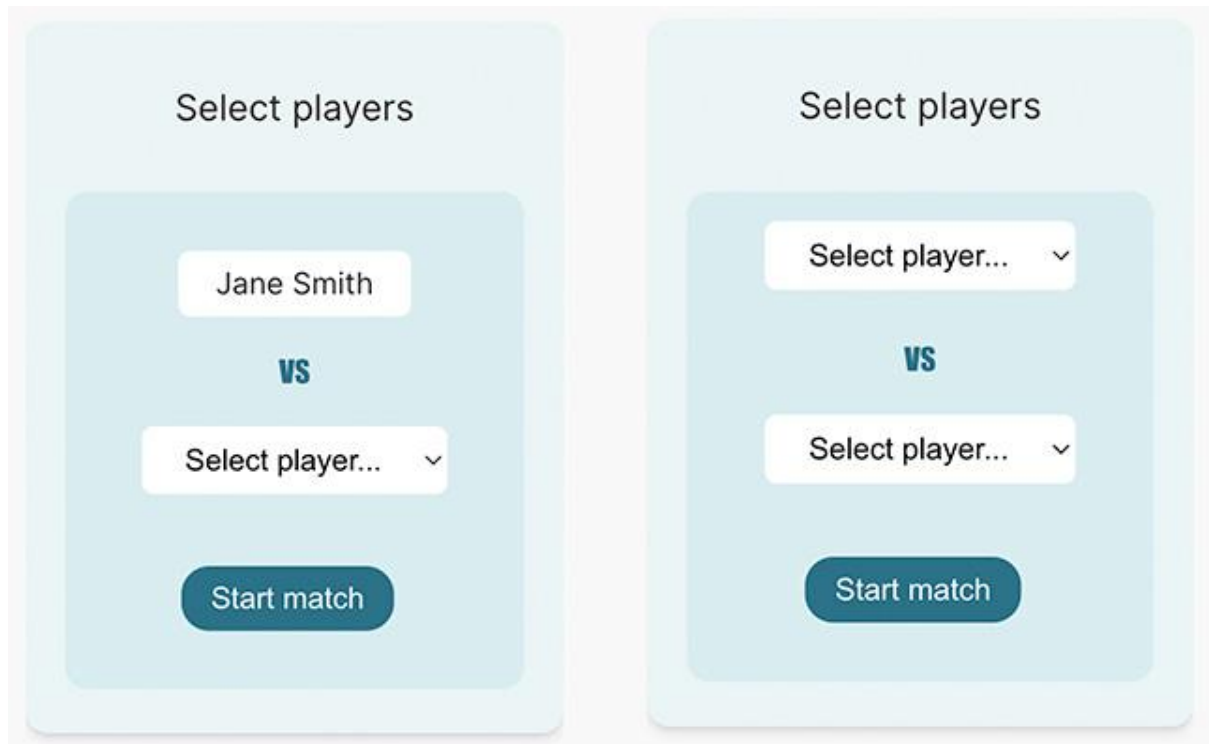
Screenshot: All players with different views depending if you are logged in as admin or not.

3.3.4 Favorite players page

In the second subpage “Favorite players”, you will see the same table, but now only displaying all the favorite players of the logged in user. Here you can keep track of your favorite players, or remove them from your favorites.

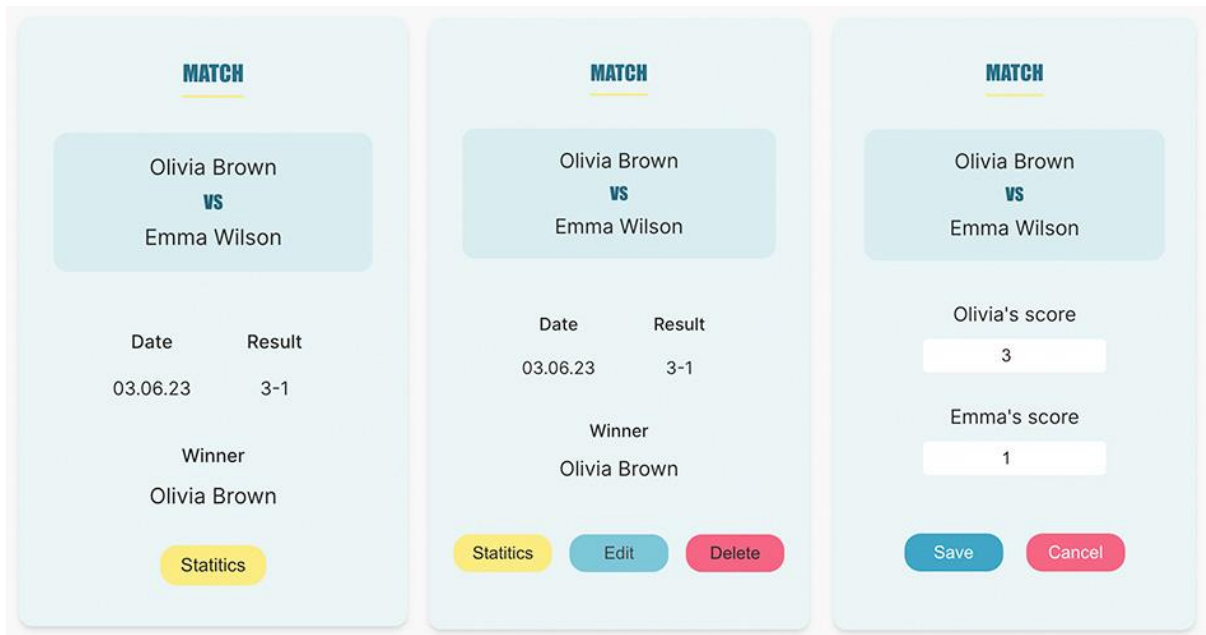
3.3.5 Matches page

By navigating to the matches page, you will find a lot of different features to explore. If you want to play a new table tennis match and save the results, this is the place to be. By clicking on the “new match” button, a new component will render where you can choose your opponent player. If you are logged in as admin, you are able to choose both player1 and player2. Of course, you can’t choose the same player for both player1 and player2, which an error message will show if you accidentally choose the same player. After you have chosen your opponent, you can start the match. From here you will be navigated to the match page.



Screenshot: After you click “new match” you can choose your opponent, or both players, based on whether you are logged in as user or admin.

In the matches page you will also have an overview of all the matches in the league, displayed as a card shape. Each card represents a match, including its overall stats such as players, the date, overall score, and the winner. If you are logged in as a regular user, each match will be populated with a “statistics” button. If you click on this button you will again be navigated to the match page. How the application displays the match page differently, we will explain shortly. On the other hand, if you are logged in as admin, you will see two more buttons in addition to the “statistics” button. You are now able to edit or delete the match you’ll like. When editing or deleting a match the score and winner will be updated, and the players points and matches won will be calculated appropriately with the new stats.

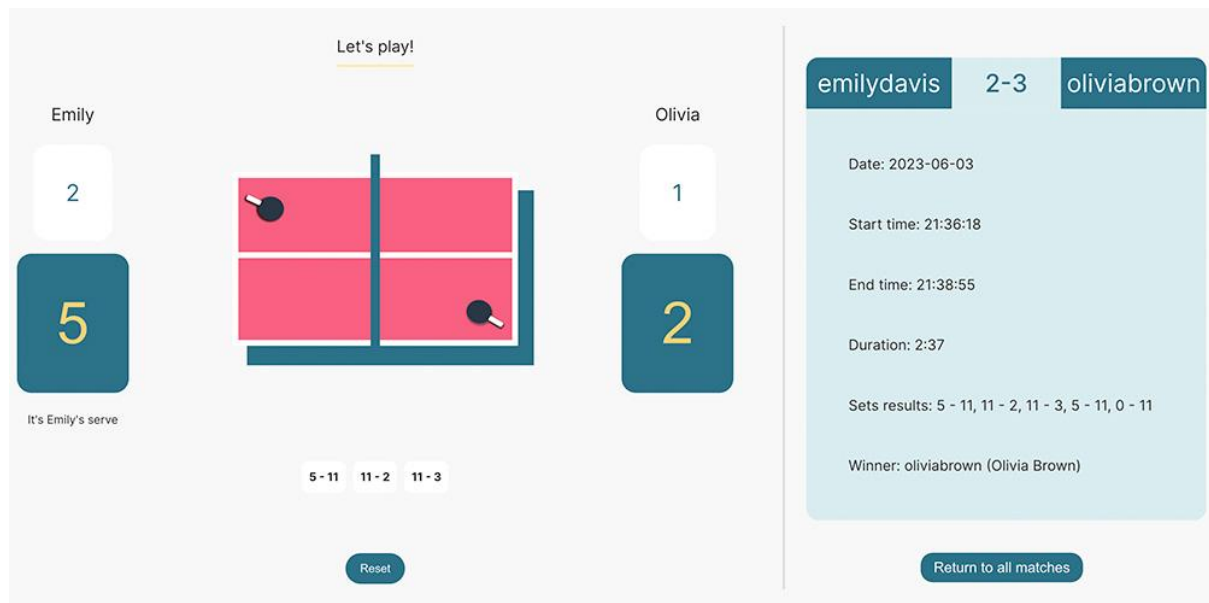


Screenshots: One of the matches with different views depending on which user is logged in.

3.3.6 Match page (id)

When you arrive at the match page, it will render differently depending on whether a match is finished or not. If you start a new match, the match will not be marked as finished, and therefore the implemented web component from oblig 1 will be rendered. The web component will keep track of the serve, how many points and winning sets each player has and all the finished sets. The first player that wins either three rounds or has two more winning rounds than the opponent, wins the match. You are also able to reset the game at any time. The web component also keeps track of the start time and the end time. When a match is complete it will display the winner, and when the “finish match” button is clicked, the match page is re rendered and the match statistics is displayed.

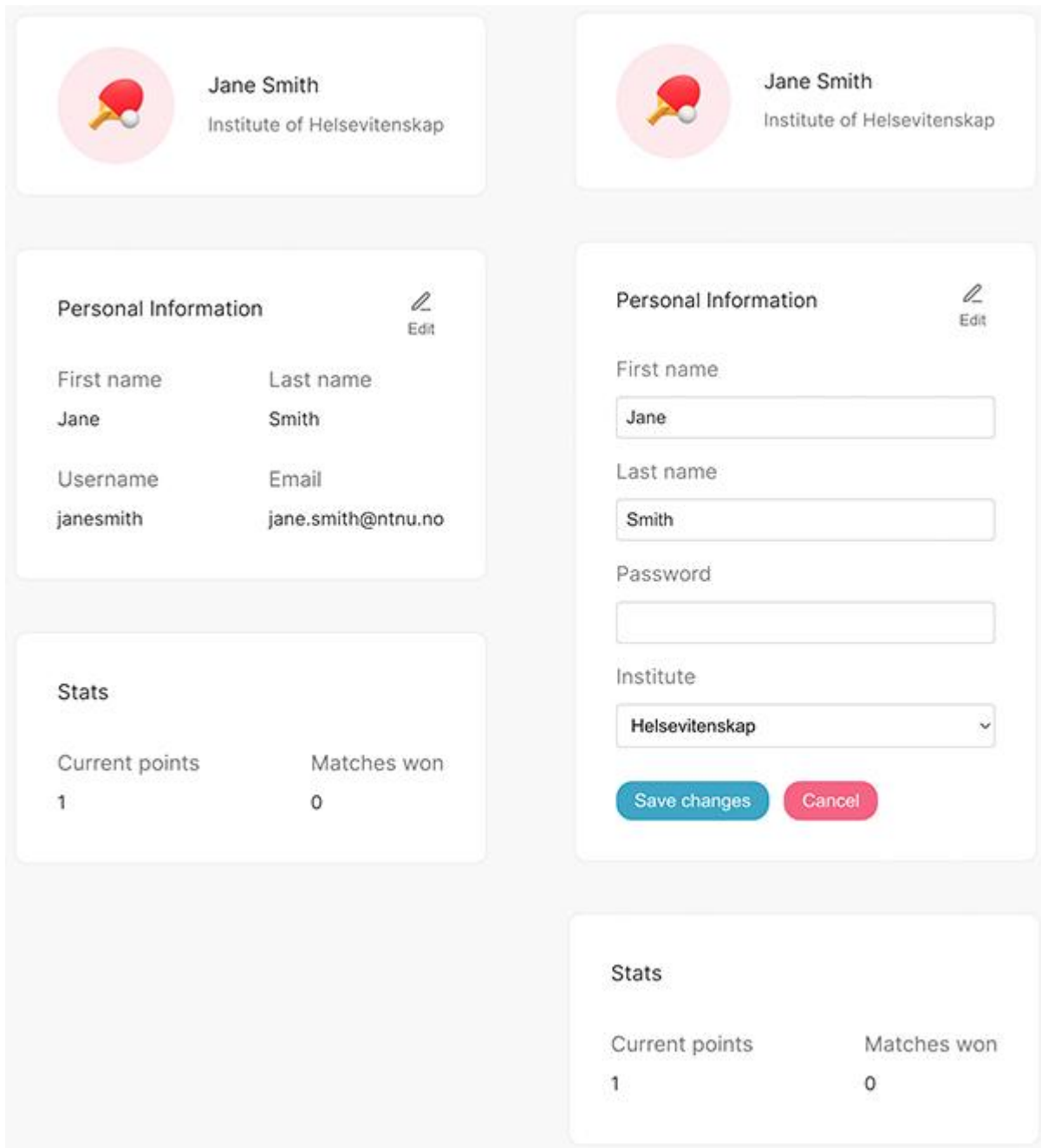
As mentioned earlier, when you click on the “statistic” button on one of the matches in the matches page, you will also be navigated to the match page. The functionality behind how the match page renders and what is displayed, is that the application sends a get request to the server, retrieves the current match from the database, and then checks if the match currently has been updated with “finished” true or false. If the match is finished: false, the web component will render, and if the match is set to finished: true, the statistics of the match will be displayed. Every time a match is finished played, it will update the matches “finished” to true in the database.



Screenshot: Match page with different views: web component or match statistic component.

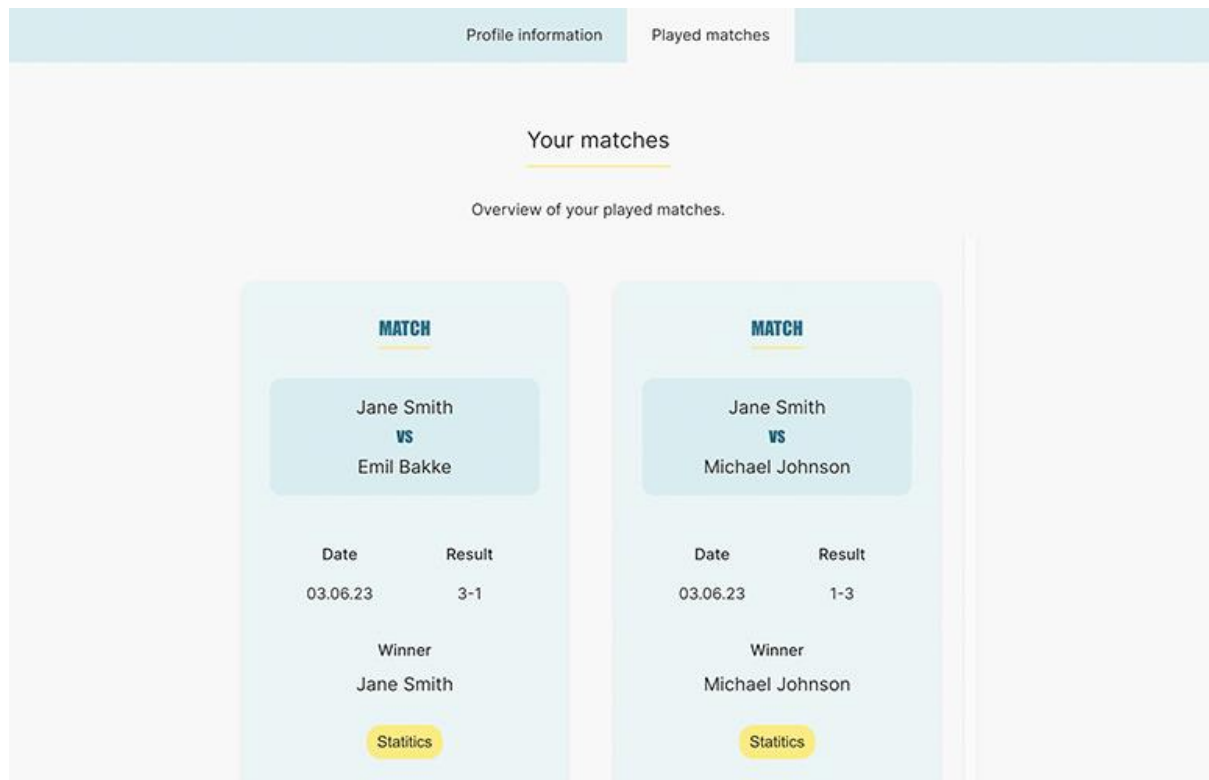
3.3.7 Profile / Admin page

We have decided to implement all the admin functionalities on the dedicated pages. For example, to edit a player, the admin can do that in the players page. Therefore the profile page and the admin page will have the same layout and display your profile information. In the profile / admin page you will see an overview of all the user information such as name, department, email, username, matches won, and current points. If you want to update your user information, you can easily do that here.



Screenshot: Your profile in view and edit mode.

In the profile page you can also see the history of all your played matches. You can choose the different views in the sub navbar.



Screenshot: Your played matches on the profile page.

3.4 Security

3.4.1 Backend

This project uses several different tools to make the backend more secure. Almost every function is using try-catch constructors that make the error handling a lot easier. If a procedure doesn't go well, we can easily figure out the error.

To make sure the password for the user is safe, we have implemented input validation. If the user is trying to log in, the server gets both the email and password inputs, checks them up against each other, and then gives the user access or not based on the result. All user passwords are hashed, meaning the original password is encrypted into a random generated string. Every time a user logs in, the server decrypts the password in the database and compares it with the user input. By doing this we have at least basic authentication implemented.

To implement authorization to the application, we have created different roles for the users. Therefore, you will have access to different API endpoints depending on what role you have. Some API endpoints require the user to only be logged in (not anonymous). By creating an "auth" verify token middleware, we can add that to the

routes we want protected. If some information is only accessible for the admin, we have also created an "admin" verify token middleware to add to several api endpoints.

Access Token and Refresh Token are also implemented to further secure the authentication and authorization processes within the application. The Access Token is a short lived token that grants access to specific resources for the authenticated user, which are included in API requests. If the user tries to refresh the page, the Access Token will be destroyed and you will therefore be logged out. Further, we have the Refresh Token that is not included in an API request, instead it is securely stored by the client-side, including in a HTTP-only Cookie. The Refresh Token is used to retrieve a new Access Token when the current one expires. When you try to refresh the page after you've logged in, the server checks if the Refresh Token is still valid in the database, and based on the result let you stay logged in or not. By implementing Refresh Token, it creates a balance between security and user experience.

```
const mongoose = require('mongoose')

const refreshTokenSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
  },
  role: {
    type: String,
    required: true,
  },
  generatedTokenString: {
    type: String,
    required: true,
  },
  expireDate: {
    type: Date,
    required: true,
  }
});

module.exports = mongoose.model('RefreshToken', refreshTokenSchema,
  'refreshToken')
```

Code snippet: Refresh Token Schema

3.4.2 Frontend

One of the security measures we did in the frontend was to implement protected routes. This is a mechanism that is used to handle and manage access to certain routes or pages based on the user authentication or authorization. We are rendering access to the different pages based on the logged in state of the user. If the user is not logged in, it will only display the homepage and the login page. After the user logs in they will have access to more content.

```
import React, { useContext } from "react";
import { Outlet, Navigate } from "react-router-dom";
import { TechnologyContext } from "../providers/TechnologyProvider";

// Protected routes
const ProtectedRoutes = () => {
  // Get the loggedIn state from the context api
  const { loggedIn } = useContext(TechnologyContext);

  // Tell React when nested elements will be rendered depending on the
  // loggedIn state.
  return loggedIn ? <Outlet /> : <Navigate to="/" />;
};

export default ProtectedRoutes;
```

Code snippet: Initializing protected routes

In the Context API we are also decrypting the Access Token using the `jwt_decode` module, to get the role of the logged in user. Depending on the user's role, it will get access to different elements and/or functionalities. The status of the role would then be accessible to all the children of the application.

```
// Render decode role
useEffect(() => {
  if (!accessToken) {
    return;
  }
  const role = jwt_decode(accessToken).role;
  setUserRole(role); // Set the user role in the context/provider state
}, [accessToken]);
```

Code snippet: Decrypting the `accessToken` to get the user's role

4.0 Reflections

Throughout this project we have consistently been working with a mindset that we will accomplish all the given tasks, including the 3-star ones. Our goal is to receive a high grade, as well as gain new experiences. We have aimed high and put in the work to show for it, therefore we are hoping to get a top grade.

4.1 Achievements

During this project we have had high expectations of our work and strived to implement all the one, two and three star tasks. We are therefore happy to say that we managed to complete all the 1-star, all 2-star and all 3-star tasks, and have implemented them to the best of our abilities into our project. When there was a choice between choosing the 2-star or the 3-star task, we always chose the 3-star task.

4.2 Discussion

As we have worked with high ambitions, we are happy and confident in how the final web application turned out. We think we have been successful in implementing and completing all the tasks given, and how our web application comes together as a whole. During development, some things that have gone particularly well have been creating REST API endpoints and all that is behind them. By some degree of trial and error, we have been successful in creating a variety of controllers that all accomplish different functionalities and data handling. Seeing how it's all tied together - the routes, schemas and controllers - and how it further ties into the front-end, has been a rewarding experience and given us a better understanding of the MERN stack. On that note, we also thought documenting our API's with Swagger went very well and it gives us a nice overview of all our endpoints.

We are also proud of how we managed to incorporate the web component from oblig 1 in our application, and how the page renders different information depending on the status of the match. This was not a simple task, since we are now dealing with React and not the HTML DOM. But after some hit and miss the web component got successfully implemented. Both the functionality, as well as the styling of the component, now work seamlessly with our application. To add to

that, we were also successful in changing the users points and matches won if a match were to be edited or deleted by the admin.

Other things that went well included creating the Refresh Token in regards to our security functionalities. This increases overall application security, while also giving the user an enhanced user experience by eliminating the need to login every time a page refreshes. Sending confirmation emails upon user creation and after updating a player's password is also something we are proud of achieving.

However, no project is perfect, and despite being happy overall there are some parts we would like to improve. One thing that perhaps isn't ideal is the fact that we could have taken better advantage of React. As components are reusable, there are some components we could have designed better. For example, we have created multiple list components, but instead we could have created one and used props to make it reusable. We should've created a button component as well. Buttons are something we use a lot in this project so it would have made more sense to incorporate this as a component rather than making multiple buttons. In future updates of this project we will make changes to ensure that we make better use of the component-based nature of React, in order to avoid code duplication.

On the topic of components, it was when documenting our components using Storybook that we realized our mistake in regards to how we utilized our components. We didn't really have many stateless components that were ideal for Storybook unfortunately, but still made our best attempt to document those components that we could. However, we realize that our Storybook documentation could be a lot better. For future projects we will work on our Storybook skills as well as learning how to document components that have a useContext hook. Since this was our first time implementing it, we will have more experience next time.

To conclude the report, we are happy with our efforts and proud to present our Table Tennis League App, that has all star tasks implemented. Although there is room for improvement, this just means we can make an even better application in the future. The learning outcomes we have gained from developing this application is something that we are very appreciative of, as we have learnt a lot along the way throughout this process.

5.0 References

MongoDB. (n.d.). *What Is The MERN Stack? Introduction & Examples.*

<https://www.mongodb.com/mern-stack>

useRef – React. (n.d.).

<https://react.dev/reference/react/useRef>

Semyon. (2023). *Swagger for Technical Writers: Benefits and One Big Limitation.*

<https://clickhelp.com/clickhelp-technical-writing-blog/swagger-for-technical-writer-benefits-and-one-big-limitation/>

Web Components - Web APIs | MDN. (2023, April 24).

https://developer.mozilla.org/en-US/docs/Web/API/Web_components

What is a REST API? | IBM. (n.d.).

<https://www.ibm.com/topics/rest-apis>