Open Powerlifting Analytics

Advanced Web Development

Interim Report

Supervised by: Hugh Shanahan / Erjill De Vera

Department of Computer Science

Royal Holloway University of London

Biddiscombe-Barr, Jake

100948416

Table of Contents

1 Abstract

Powerlifting is a strength sport that consists of a 1 rep max attempt of 3 compound movements, these movements are as followed, squat, bench press and deadlift [1]. The term used to describe a competitor of this sport is called a Powerlifter[2]. Open powerlifting is an online database that contains thousands of powerlifter competition results over many years now, this information is freely available to inspect [3] however, the user interface is confusing, and the data is raw and not particularly useful. The aim of my project is to create a web tool that uses this data in a more functional manner, allowing users / powerlifters to run analytics on their accounts, filter through lifters or search via various categories, compare against a single lifter or a group of lifters where an average will be created for the parameters selected. An example would be, comparing against all the U83kg Junior Male powerlifters.

The focus of this project is to provide the user with a seamless experience, where results are returned rapidly, the UI is attractive with smooth animations and the UX is intuitive. To achieve this, I want to use the state-of-the-art technologies that are most popular in industry to date. The project will be created using Next.js coupled with TypeScript which will allow me to capitalize on its server-side rendering capabilities and simple page routing. Next.js is also described as a full stack framework allowing me to not need a separate BE system [4]. As Next.js uses React.js I will also be able to benefit from Reacts state management system [5], allowing pages to effortlessly connect only requiring small renders depending on state changes instead of whole page changes. I will also be using skeleton loading to display loading segments to the user. This approach is followed by many large corporations such as Google, Facebook, LinkedIn and more [6]. The last few technologies I will use are StoryBook to help design components more effectively [7], Eslinter to adhere to industry standards and follow efficient coding practices [8], and Sass styling which will increase readability, reusability and productivity of my web styling [9]. The project will also have elements of statistics, when attempting to create comparison results and the analytics results. There are a few models I would like to further research to create these.

There are a few approaches I can take to the project. I discovered using the web development tools that I can query Open Powerlifting's API to collect user information without the need of an API key. I could create some helper classes, in Next.js alongside node to utilize their full stack capabilities, to handle the data and mathematics. Another possibility is that I could use an entirely new BE system to add some complexity. However, these are both risks as I am not in control of the Open Powerlifting system so another approach is localizing this data via a Firebase database with dummy data as this will allow me to quickly access info directly from the FE. I will also be using Firebase alongside Next Auth as a user authorization system [10].

2 Timeline

The focus of term 1 will be around finalizing research, creating all the designs and all the static pages. The second term will be focused around implementing the logic of the problem, these tasks will be more intensive, so I have made sure to follow proper time management and give myself a couple weeks for these more complex tasks.

Term 1

- Week 1: Continue research of Next.js and Storybook.
- Week 2: Finalize Next.js course, research of authentication system and create basic repo structure / common files.
- Week 3: Create the designs and flow for the Homepage, Comparison and Analytics.
- Week 4: Create the designs and flow for the Search, Landing and Login.
- Week 5: Ensure that the designs are responsive on all devices.
- Week 6: Create the common reusable components of the project such as buttons and navbars.
- Week 7: Test the API thoroughly to understand the Helper class / BE structure or if I need to use a Firebase BE.
- Week 8: Solidify the mathematical models used to create the comparison / analytics.
- Week 9: Create the static Homepage and Analytics page.
- Week 10: Create the static Comparison and Search pages.
- Week 11: Prepare for the interim report and presentation.

Term 2

- Week 1: Create the static Landing and Login pages.
- Week 2: Implement the router.
- Week 3: Create the code for the login system.
- Week 4-6: Create the state management system for the website.
- Week 7-8: Create the Helper / BE code.
- Week 9: Finalize styling and animations.
- Week 10-11: Prepare for the final report.

3 Risks and Mitigations

Risk	Impact	Probability	Mitigation
Open Powerlifting API not having expected customizability	Medium	20%	The API has no documentation, I have done some simple testing, and I should be able to get all the information I require through a single query. If this becomes impossible, I will have to use the Firebase approach.
Testing and debugging overtime	High	15%	Make sure to test as I code to keep testing simple instead of large confusing chunks
Scope Creep	High	10%	Do not increase the scope throughout the project and adhere to the plan
Poor time estimation	High	10%	Realistic time management is essential, some tasks could take longer than expected so giving myself a buffer to allow for some tasks will be prevent delay and frustration.
Designs are not responsive	High	10%	The designs need to be fluid across all resolutions as the focus of the project is user experience. To allow for this I will create individual designs for mobile, tablet and desktop with industry break points. This has been allocated sufficient time to design effectively and carefully.
Chosen technologies not performing as expected	Low	5%	From my research, I have selected technologies that should all enhance user experience from design to speed. For example, if Next.js has server-side rendering to enhance client speed, there is a chance that I do not optimally utilize this due to lack of experience with the technology. I have given myself some time to learn these technologies during the first couple of weeks.
Uneven project balance	High	5%	Focusing purely on just the project could hinder my report quality and vice versa, to allow for these reports and deliverables, I will give myself a week or two towards the end of each term to focus on the reports.
Security Concerns	Medium	5%	Following good security standards is an essential web development practice. As this is not going to a real client, it is

			not a priority however, I will attempt to adhere to good standards to increase my personal coding skills as I will not want to bring bad habits into industry
Open Powerlifting removing their data	Medium	< 1%	As this is a third party, if they do anything unwanted with the data on their end it will be unusable to me. This could include removing the API for maintenance or moving to a different system.

4 Background Theory

Main Technologies

As mentioned in the Abstract, I will be using state of the art web technologies for this project and since the creation of the Abstract there have been some further technologies added that I wish to discuss.

The project is mainly written using Next.js. This is a full-stack web framework that is one of the most modern and is built upon React.js which is extremely popular among web developers in many industries, some popular websites built with Next.js are TikTok, Netflix and Notion [11]. The main difference between Next.js and React.js is that Next.js allows for server-side rendering, taking a lot of processing power away from the client, in a lot of cases vastly increasing rendering and speed performance [4].

I have coupled this framework alongside Typescript, this won't add too much benefit to the project compared to vanilla JavaScript, but it is a nice to have. This is because Typescript requires variables to have specified types throughout the code base, this makes it more readable and maintainable which is great for large projects with larger teams but not particularly necessary to use in small projects such as this one [12]. However, if it is used by the industry in general, I think it would be great to include it within the project.

Styling is very important for a web developer, during my placement styling was about 50% of my job and daily activities so it is essential to use the most efficient tools. I will be using Sass throughout this project which is built upon CSS and allows for some efficient methodologies such as indenting styling, drastically decreasing lines of code compared to CSS [9]. An alternative to this was to use Tailwind, which is very popular. Tailwind removes the need for dedicated CSS files, instead you can do the styling within the class tag of an HTML element [13]. I do not like this approach, as the styling can get very long making it hard to read and it becomes difficult to decipher what each element's purpose is as there is no class name to describe it. For example.

- Sass:
- Tailwind:

This makes up most of my front-end. I originally was not planning on creating a back-end system as Next.js is a full stack framework which makes it extremely powerful. But during my work around week 7, I realized that there were some unseen limitations of the Open Powerlifting API. This was regarding CORS; this was preventing me from accessing the response of my request. CORS is a security measure on web browsers to prevent unauthorized access to resources from a different domain [14]. In this case the creators of the API I am trying to access could have just forgotten to add a single line of code to allow this as other platforms use this API and it works correctly.

The solution to this was to create a proxy server [15]. I would then forward my API requests through this proxy which would solve the domain issue. This is not exactly ideal as there is some slight additional computation time, but this increase is barely noticeable from the user's perspective. I set up the server using Express which is a small Node.js framework. This allowed me to access request information I send to the proxy so I can inject variables into the API URL. I used Express.js as my back end is going to mainly used to just reroute my requests with some slight adjustments to the response so I did not any specialized proxy solutions. Express allowed me to very quickly set up the server. It is a part of the node-ecosystem allowing me to access lots of powerful node functionality [16] that I could use in the future such as caching and load balancing.

Since the project plan, there has been one technology that I wish to remove which is Storybook. This tool would allow me to develop and test my UI components in isolation without any project dependencies interfering. This seems to be useful on projects with more than one developer whilst working with designers to achieve a component that works for both parties. In my case, this does not seem to add much value to the project and will increase my development time unnecessarily along with the learning curve of the tool. Although, it would be nice to design each component in detail I am comfortable with my styling capabilities in addition to this, the project is small so no project dependencies should interfere.

Libraries

I have installed some additional libraries for this project to streamline my workflow. The first library is called classnames. This is a library I have gotten on well within the past, it allows me to create conditional classnames on html elements [17]. This is extremely powerful as I could then change the class of an element depending on the contents of a variable. An example of when I used this is within the navbar component, I have a variable which can determine if the browser window is in a mobile resolution and updates to then True or False. Depending on the Boolean value the library will add "mobile" to the classname which then applies my mobile CSS navbar code.

The next library is called styled components. This is another library regarding styling and again is quite powerful. It allows me to inject variables into the styling via object literals [18], this is extremely useful for animations or complex components such as progress bars which are a common component within my project. An example of how I would use this would be from calculating the performance level of user's squat in the form of a percentage, I can

then pass this percentage to the styled component to access styling attributes like width for the progress bar which is other words would be how complete the bar is. The downside to this is it includes all the styling with my Typescript file, however I will only be using this method rarely as mentioned, for the complex components.

Extensions

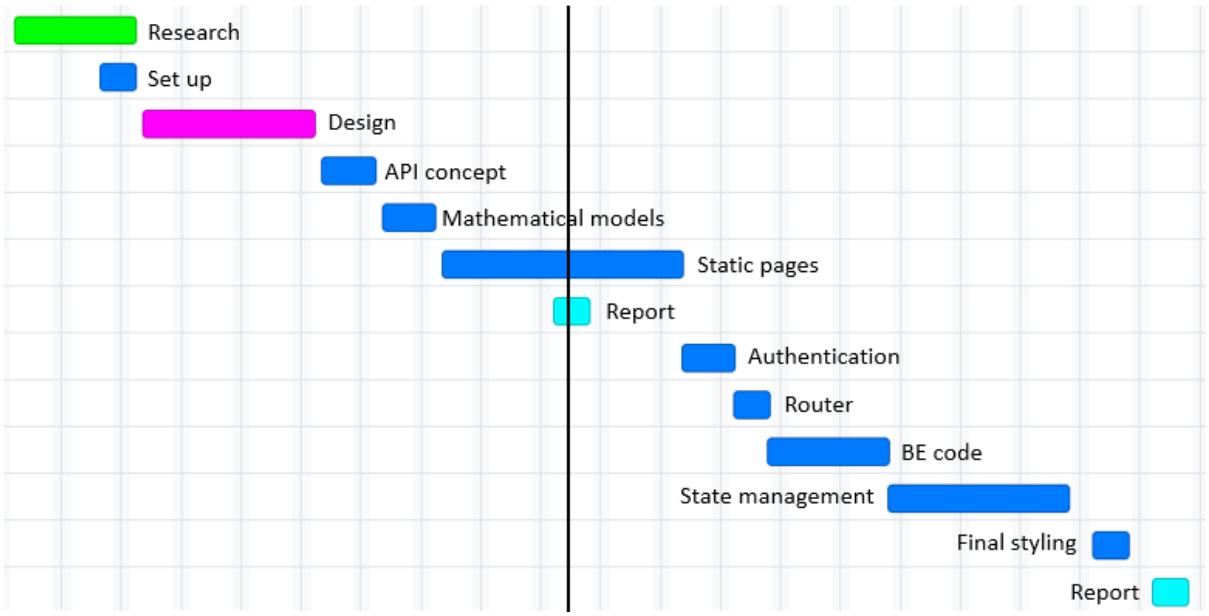
Alongside these libraries, I am using some IDE extensions for formatting my code to the industry standards using Prettier [19]. I have set this up so when I save my code it automatically formats the entire file which saves a lot of time and helps keep it as readable as possible.

In addition, I have included the Eslinter extension and library. This ensures that my code meets software engineering standards depending on my configuration [8]. The configuration I chose is the Airbnb configuration used by many developers even outside of the Airbnb company as the rule set is very well established. I added some extra configurations to double check my Typescript and Prettier set ups and then removed any overlapping standards that could cause unnecessary errors. Now my IDE is set up so that it will automatically highlight any code breaking these standards and prevent me from committing until they have been resolved.

5 Software Engineering

Completed Work

Here is a gnat chart to help visualize the current progress of the project. The black line represents now.



This term was sectioned into a few different stages: Research, Setup, Designing, API proof of concept, Mathematical models, Creating static pages. At the start of the project, I miscalculated the amount of time I have so everything was completed but the last two static pages which will be completed the week after this report is submitted to ensure I remain fully on schedule for term 2, apart from this everything has gone according to the previously established plan, that you can find at the top of this report.

The design was a very linear process and straightforward utilizing Figma which is a very popular tool used by expert UX/UI designers to create fluid designs and as a developer I can see the code that makes the elements I have designed via drag and drop [20]. The development has also been very linear too as the pages are all static. This means I have just hardcoded all the values on the page instead of them being dynamic and reacting to input or calculations, this is so the second term can be focused on just making these pages dynamic and computing all the calculations. Before that I needed to create a proof of concept and test if there were any limitations with API (which are mentioned in the section above). The proof of concept I created was attempting to collect information I can validate on Open Powerlifting. To do this I injected certain parameters into the request header such as weight class and age class then printed out the response after modifying the response to work with my system. I then compared these changes with Open Powerlifting after applying the same filters, the results matched.

The last section was the mathematical model I wanted to use to create the comparison. In preparation I did some more testing of the API. The API has two attributes included which are start index and an end index, so I assumed that this meant I could return any number of lifters I required however, the if the difference is more than 100, it will only return a max of 100 lifters. This limitation required another work around, but one that was probably

inevitable. This is because Open Powerlifting is an extremely large database with 100,000s of lifters so when I set everything to my default request (all fully tested, IPF approved lifters) there were still a large quantity of lifters, after applying a single filter there were still 40,000 results. Therefor the max number of lifters I would collect in one go would be 40,000.

Collecting 40,000 results would take an unacceptable amount of time, due to this I decided to take samples in regular increments. I wanted to test that this would not alter the average, so I conducted the following experiment:

Increments	Average total	Time in seconds
0	540	>60
1000	548	22
2000	556	12
5000	569	6
6000	565	4-6
7000	573	4-6
8000	592	4-6
9000	570	4-6
10000	607	4

As we can see from this table, the best case is to do increments of 5000. This means that for every 5000 lifers I will collect 100, as they are ordered by rank the average should not change that much. As you can see the average has only increased by 5% and results in a 6 second render of the data. I noticed that the average almost always seems to increase as the increments increase, so after some further tests I could investigate making it closer to the real average by offsetting this change.

Practices

This term has been very focused on designing and creating the static pages therefore in-depth design patterns or testing has not been necessary yet. However, I plan to include more elements of both in the second term.

Regarding testing, I will be mainly testing the BE as testing the FE to ensure proper flow can be quite tedious and I do not believe will provide much value. Therefore, I will be testing that the BE responses are in the correct format and contain the correct number of lifers per request. I can do this using a combination of Jest and the React Testing Framework [21]. These tools in unison will allow me to create mock objects and ensure that the BE is sending over the correct information. If not, I can then handle these errors accordingly by potentially showing different pop ups to users to keep them informed.

So far, I have been following the correct methodology of Next.js where all elements need to be component based to greatly reduce repetition. Therefore, I have created a section for reusable components such as the Layout, Navbar and Button components. Then for the pages I have split them up into components and subcomponents, this is great for maintainability. An example would be on the Home page I have the three link components, if I wanted to update these, I could change the HomeLink component and easily update all three at the same time.

In terms of design patterns, I will continue to use the Next/React approach to development of utilizing redux and state. Redux allows Next/React components to read data from a Redux store and then dispatch actions to update this store [5]. This will be mainly used to store all the user information once collected from firebase, so I do not have to constantly fetch it. Instead, it now can be fetched locally, making it significantly faster. In more depth Redux consists of reducers and actions. We can call upon actions, which are functions, in our component files. This can then update the reducers which contain an object of all the global information that we might want to change. It is also great for caching, so once the averages have been computed, I would not need to update them until an input is changed by the user.

Some other common design patterns I will be incorporating will be file based routing. Next.js has the option to route to different pages depending on the file structure of your system [4] instead of needing a specific router component like React.js. All you do is simply label the folder of the page and include an index.js file within. When compiled the browser will then know what file to load when the matching URL is entered. This system can manage complex scenarios including nested pages and events where you would have pages that are unique for different users.

Another important pattern I will be taking advantage of is the server-side rendering pattern. React.js has two common hooks for rendering data which are *useState* and *useEffect*. These allow us to update state on certain events which can then determine what content is being rendered. This all happens locally on React on the client browser but with Next.js there are some additional hooks *getServerSideProps*, *getStaticProps* and *getInitialProps.* These are very similar to the React.js hooks mentioned but will tell the system to render on the server on the initial page load rather than on the browser [22]. The latter is specifically used on class-based components, I will only have one of these in my project called the Helper class. Its main job will be formatting any BE code into more usable formats for different use cases. An example of this would be when I am sending the data to have its averages calculated for the squat and extracting just the squat. The other two are used depending on if the page is rendered frequently or not. Whilst the page is being rendered, I am going to be using a technique called skeleton loading. It's been shown that during web applications people seem to dislike the typical loading bar and get a negative association. Skeleton loading provides a more engaging and less frustrating loading experience by showing what the components are going to like so then I can just populate the component and the user can expect where to see it. I will be using this in my progress bars on the analytics and comparison pages as this content will take a moment to render.

Repository

In my repository there are two main folders

- FE
- BE

The FE contains most of the code as this project is very front end heavy. The structure of the project is the same format that I used in industry. This includes a variety of folders:

- Assets -> Contains images and icons.
- Components -> Contains all my different components.
- Pages -> Contains the different pages to switch between.
- Reducers -> Contains the reducers and callable actions to manage global state.
- Styles -> Contains all the styling.
- Utils -> Contains helpers and API class.

Each of these contains subfolders regarding the different pages just to section up the remaining files. An example of this would be that within the Components folder there would be an Analytics containing all the components unique to Analytics page alongside additional folders for more global components such as buttons or navbars. The FE structure will grow over time as I will need to include additional folders for redux.

The BE is a lot simpler, as it just contains one file for the routing along with the designated node modules for that file.

Future Capabilities

By the end of this term, I will be in a position where almost all the static pages have been created which is in alignment with the original plan. The next term will be focused on making these pages functional. This is a brief description of the functionalities each page will have.

Login / Sign up: Allows the user to login with their account via firebase or create an account to be added to the database.

Search: The user will be presented with a search bar and enter the name of a lifer to view their stats specifically. This will use the Open Powerlifting API to collect the lifter information and return it to my FE.

Analytics: The user will submit their stats which is saved to the firebase database, so it automatically loads in the future. The user will then get scored against the other lifters of the same category.

Comparison: The user will be able to compare themselves against a single lifer via search or a group of lifters with their averages for each lift being graphically compared.

Setup

The project requires NPM to install all the dependencies for the FE and node to run the BE. When you install the Node package manager it includes both commands that we can then use in the terminal to start the project. I created a YouTube video to help with the set-up process from scratch. Here is the Link.

Reflection

Overall, I am happy with my current progress in the project. I am glad that I decided to do proof of concepts so that I could test if my theory was going to apply correctly. This allowed me to see limitations in the Open Powerlifting API and create solutions to these limits both mentioned in the report, to save my development time later in term 2.

7 Diary

Week 1: I have been researching the technologies I want to use in more depth, specifically Next.js and Storybook. For Storybook I have been watching YouTube tutorials and for Next.js I have purchased a course on Udemy.

Week 2: I continued my research into Next.js and Storybook. I have decided that Storybook might not provide much value to the project. After further research it seems that it is suited for larger enterprise projects with bigger teams that include many reusable components. Although I plan to use reusable components, my library will not be at the scale to make efficient use of the tool.

I also gained access to the repo locally and online, beginning to make my first few commits which have been mainly based on: project structure, dependencies, README and removing the clutter of initializing a Next.js project.

After looking into the authentication system, I have decided to use Firebase by Google alongside Next Auth to manage users and their respective information.

Week 3: I have almost finished all my Next.js course, now I believe I have the knowledge to successfully optimize the server-side rendering capabilities of Next.js.

I spent most of the week designing creating the designs for the account, homepage and comparison. The comparison was the trickiest as it had multiple stages, 4 possible screen states. I utilized some free graphics I found online called undraw, they are quite minimalist and use the same color scheme of the website. These are all, however, subject to change there are some small tweaks I will make in the future for consistency purposes.

Week 4: Most of my time this week was spent completing the designs for the login, sign up, search and landing pages. These pages are less complicated than last week as there were fewer possible states, I utilized the same tools as last week to create these designs.

Week 5: This week was focused around finalizing designs and the layout such as navbars etc. and ensuring they are responsive to any screen size. The main idea of this week was to be a buffer week to allow for any unforeseen priority tasks.

Week 6: This was the first week of programming, I created the navbar for both desktop and mobile resolutions, the buttons, the layout component/ wrapper and some of the routing to different pages.

The hardest part of this week was completing part of the Layout component. As the way that I implemented the container was not working smoothly with the navbar so had to be readjusted which was not necessarily complicated, just tedious.

Week 7: This week was focused around creating a proof of concept for connecting to the Open Powerlifting API. I initially used the connect fetch function and just entered the URL for the API however, I was receiving CORS errors preventing me from accessing the response.

CORS is a security measure on web browsers to prevent unauthorized access to resources from a different domain. In this case the creators of the API I am trying to access could have just forgotten to add a single line of code to allow this as other platforms use this API and it works correctly.

To overcome this, I needed to create a proxy server, I would then forward my API requests through this proxy which would solve the domain issue. I managed to set this up and now I can access the required resources through the API.

I set up the server using Express which is a small Node.js framework. This allowed me to access request information I send to the proxy so I can inject variables into the API URL.

Week 8: This week I was focusing on how to create the averages for the comparison module. To do this I wanted to further extend the limits of the API but during this I came across another issue.

The API structure includes a start index and an end index so I assumed I could request the amount of data I required; however, the API is limited to only 100 lifters no matter the start and end limit.

My solution to this was to make a certain number of requests and create samples of lifters in certain increments as they are returned in order of strength. This means the average should theoretically not change much. The question then became, what is the increment and how would that affect the average.

I discovered the best-case increments is increments of 5000. This is because the average has only increased by 5% and results in a 6 second render of the data. In the future, after further tests, I could begin to offset this small increase.

Week 9: This week I created the full homepage; this is a relatively simple page. It just contains the links to the other pages and a description of what each page is for.

I then created the static analytics page; this page is more complex including a form and the rating bars. This page challenged my styling skills a lot further as I included animations, the bars would also need to extend up to points relevant to a user's lifts.

To have the styling react in this way, I needed to learn a new tool called styled components. These is a powerful tool that lets me inject variables into a string containing all the styling code via object literals. This for example would let me set the width of the bar dependent on their score out of 100.

8 Glossary / Acronyms

U83kg - refers to the weight category of a powerlifter, this format indicates the competitor is above 74kg but below 83kg in body weight.

UI – User Interface

UX – User Experience

FE – Front End

BE – Back End

API – Application Programming Interface

NPM – Package manager for JavaScript software

9 References

[1] Ferland, P.M. and Comtois, A.S., 2019. Classic powerlifting performance: A systematic review. *The Journal of Strength & Conditioning Research*, *33*, pp.S194-S201. Link

[2] Ball, R. and Weidman, D., 2018. Analysis of USA powerlifting federation data from January 1, 2012–June 11, 2016. *The Journal of Strength & Conditioning Research*, *32*(7), pp.1843-1851. Link

[3] Ferland, P.M., Allard, M.O. and Comtois, A.S., 2020. Efficiency of the wilks and IPF formulas at comparing maximal strength regardless of bodyweight through analysis of the open powerlifting database. *International journal of exercise science*, *13*(4), p.567. Link

[4] Dinku, Z., 2022. React. js vs. Next. js. Link

[5] McFarlane, T., 2019. Managing State in React Applications with Redux. Link

[6] Experience, W.L. in R.-B.U. (n.d.). *Skeleton Screens 101*. [online] Nielsen Norman Group. Available at: https://www.nngroup.com/articles/skeleton-screens/#:~:text=Summary%3A%20A%20skeleton%20screen%20is [Accessed 2 Oct. 2023].

[7] Lanciaux, R., 2021. *Modern Front-end Architecture: Optimize Your Front-end Development with Components, Storybook, and Mise en Place Philosophy*. Apress. Link

[8] Tómasdóttir, K.F., Aniche, M. and Van Deursen, A., 2018. The adoption of javascript linters in practice: A case study on eslint. *IEEE Transactions on Software Engineering*, *46*(8), pp.863-891. Link

[9] Cederholm, D., 2013. *Sass for web designers*. A Book Apart. Link

[10] Moroney, L. and Moroney, L., 2017. Using authentication in firebase. *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*, pp.25-50. Link

[11] Showcase / Next.Js by Vercel – *the React Framework.* Link

[12] Jansen, R.H., Vane, V. and De Wolff, I.G., 2016. *Typescript: Modern Javascript Development.* Packt Publishing Ltd. Link

[13] Le Duy, H., 2023. Web Development with T3 stack. Link

[14] Hossain, M., 2014. *CORS in Action: Creating and consuming cross-origin APIs*. Simon and Schuster. Link

[15] *What is a CORS error and how to fix it (3 ways)* (2023). Link

[16] Mardan, A. 2014. *Express. js Guide. The Comprehensive Book on Express.js.* Azat Mardan Link

[17] *npm: classnames* Link

[18] Phan, H.D., 2020. React framework: concept and implementation. Link

[19] Khan, O.M.A. and Habib, K., 2020. Developing Multi-Platform Apps with Visual Studio Code: Get up and running with VS Code by building multi-platform, cloud-native, and microservices-based apps. Packt Publishing Ltd Link

[20] Staiano, F., 2022. Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop. Packt Publishing Ltd. Link

[21] Tran, M., 2023. Testing React Applications Using React Testing Library. Link

[22] Konshin, K., 2018. *Next. js Quick Start Guide: Server-side rendering done right*. Packt Publishing Ltd. Link