Avancerad Webbutveckling med Ramverk

Front-End Sportstats

Max Jonsson
Maxjonsson9@gmail.com
Hassan Sheikha
Sheikha.hassan.rb@gmail.com
Hasse Kazan
Baran.kazan@hotmail.com

2019-10-30

Projektrapport 7.5 HP,
för Avancerad Webbutveckling med ramverk
på dataingenjörs-programmet

Examinator: Åke Wallin
Handledare: Åke Wallin

1 Introduktion

I detta projekt har ett front-end system byggts till det back-end system vi byggde tidigare under året. För att göra detta har vi använt oss av ramverket react.js, plus några andra ramverk för styling av sidan och hämtning av data från API:et. En noggrannare beskrivning av dessa ges nedan.

1.1 React

I detta projekt har vi använt oss av react.js. React.js är främst ett användargränssnitt-ramverk skapat av Facebook. react.js -ramverket fungerar single-page dvs. sidan ska aldrig behöva laddas om. Det innebär att allt som placeras på skärmen är komponenter. När navigeringen ska ske till en annan sida så byter man endast ut den komponent som ska ersättas istället för att ladda om hela sidan. Vissa komponenter finns färdiga att användas genom ramverket Reactstrap, men egna komponenter kan också skapas genom att göra en klass som ärver från klassen React.Component. I react.js har också varje komponent ett eget state, vilket möjliggör att det som renderas på skärmen anpassas efter ett givet tillstånd. Varje gång en komponents state förändras kommer komponenten automatiskt att renderas om och på så vis anpassa sig till sina nya förutsättningar. Komponenterna har också vissa livscykelmetoder så att man kan utföra vissa instruktioner till exempel varje gång komponenten placeras någonstans, eller varje gång komponenten uppdateras.

1.2 Reacstrap och Bootstrap

För styling på sidan har vi använt oss av ramverken Reactstrap och Bootstrap. Reacstrap är en implementering av Boostrap-ramverket specifikt gjort för react.js. Det är gjort för att passa in med react.js komponent-baserade struktur. Reactstrap är likt Bootstrap i och med att båda är CSS ramverk, skillnaden är endast att Reactstrap är anpassat för ett specifikt ramverk.

1.3 Font Awesome

Font Awesome användes också för stylingen på sidan. Font Awesome är ett ramverk som gör att man kan använda sig av speciella fonter och ikoner som ramverket tillhandahåller. I detta system används Font Awesome primärt för ikonerna i sidebar komponenten.

1.4 Axios

Axios är ett ramverk för webbanrop som användes i vårt system för att anropa API:et. Axios körs async, vilket innebär att hämtningen körs i bakgrunden på en egen tråd. Just att det är async tillåter systemet att fungera smidigare med mindre fördröjning eftersom resterande del av koden kan fortsätta köras medan man inväntar svar på anropet. För att sedan få programmet att anpassas efter svaret på anrop används callbacks.

2 Metod

2.1 Utveckling av systemet

Till en början jämfördes olika ramverk för Javascript, bland annat de mest populära; vue.js, ember.js, angular.js samt react.js. Efter jämförelsen föll valet på ramverket react.js. De första två dagarna av projekttiden spenderades på att förstå sig på react.js. Där lärde vi oss normer i kodningen, struktur på filer samt hur dess komponent-baserade struktur fungerade. Därefter skapades det ett repository för projektet på GitLab.

Eftersom react.js inte själv har någon inbyggd funktion för att kommunicera med externa API:er behövdes ett sätt att kommunicera med datalagringen som tilldelades. För detta installerades ramverket Axios. När det fanns ett sätt att koda logiken i att hämta datat behövdes också ett CSS-ramverk för styling av användargränssnittet. Det ramverk som valdes var primärt Reacstrap, vilket är likt Bootstrap, men utvecklat specifikt för react.js.

Efter att ramverken var på plats påbörjades nu implementeringen av de dokumenterade funktionaliteter som fanns hos API:et. Den först implementerade funktionaliteten var *getAllSports*, vilket hämtar alla sporter från databasen. Detta data sparades i react.js state-funktion där det senare skickades vidare för att modelleras via komponenterna och visas på skärmen. Till en början var där problem med att göra hämtningar från API:et eftersom anropen blockerades av webbläsarens CORS-funktion. Detta löstes dock genom att lägga till en proxy till API-domänen i filen package.json. När denna fanns med började hämtningarna fungera. Detta är dock en tillfällig lösning under utvecklandet av programmet och bör ersättas när applikationen görs för produktion.

Då det nu fanns en mall för hur en icke-specifik hämtning utförs, implementerades det nu en funktion för att mata in data till databasen. Denna funktion som skapades var *addSport* som nu tillät tillägg av nya sporter. En ny sida skapades för användargränssnittet där tillägg av sport modellerades. Då detta fungerade som det skulle implementerades det felkontroller för de fält som används vid inmatningen, så att endast validerade namn och nummer kan matas in.

När vi hade funktioner för att lägga till och hämta data behövdes slutligen en funktion för mer specifika datahämtningar, där man exempelvis skulle kunna hämta alla ligor kopplade till en vald sport. För att implementera detta skickades hela tiden id:n och andra nödvändiga data med i location-objektet som användes för att dirigera mellan olika paths. Location-objektet implementerades till att innehålla en path (länken dit navigeringen ska ske), samt ett state-objekt där man kan fylla på med de attribut som behöver skickas med. När en ny sida öppnas upp kan denna då komma åt datat genom *this.props.location.state.attribute*.

När det sedan fanns mallar för alla typer av funktioner som kan implementeras kunde resterande del funktioner från API:et skapas på samma sätt. Med jämna mellanrum under projekttiden refaktorerades även klasser till att följa gemensam struktur. Komponenterna sågs även över till att använda gemensamma utseenden, om där möjligt genom att bryta ut gemensamma drag till egna komponenter som sedan kunde återanvändas (dessa återfinns under mappen *reusables*). Detta tillät en lättare navigering mellan filer när systemet blev större under utvecklingen.

2.2 Arbetssätt

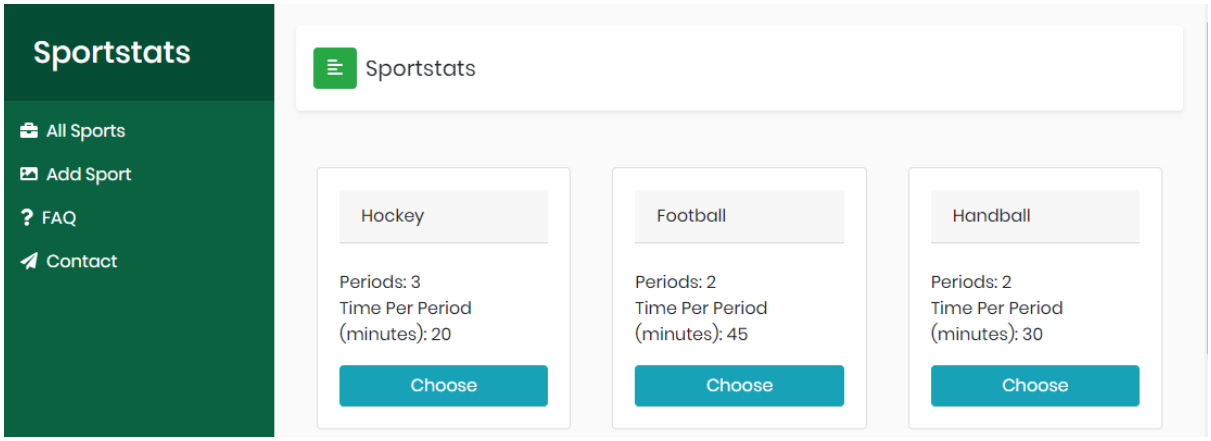
Arbetssättet som användes under projektet var flexibelt och agilt, men det var inte enligt en populär metod. Innan kodningen för dagen påbörjades erhölls det mindre möten om *vad* som ska göras, *vad* som gjorts, *vad* för problem som fanns och *hur* dessa problem ska lösas innan arbetet gick vidare. Detta möjliggjorde att systemet inte brast helt plötsligt under senare del av projektet då hänsyn till grunden alltid togs med jämna mellanrum.

För de problem som ansågs vara större infördes även användandet av par-programmering, vilket tillät en snabbare lösning på lite mer komplexa problem. Par-programmering användes även för att effektivisera vissa av programmets funktionaliteter. De funktioner som ansågs lite mera vitala för systemet tilldelades inte endast till en programmerare för att lösa, utan löstes istället av minst två för att ge högre garanti på att lösningen var effektiv nog.

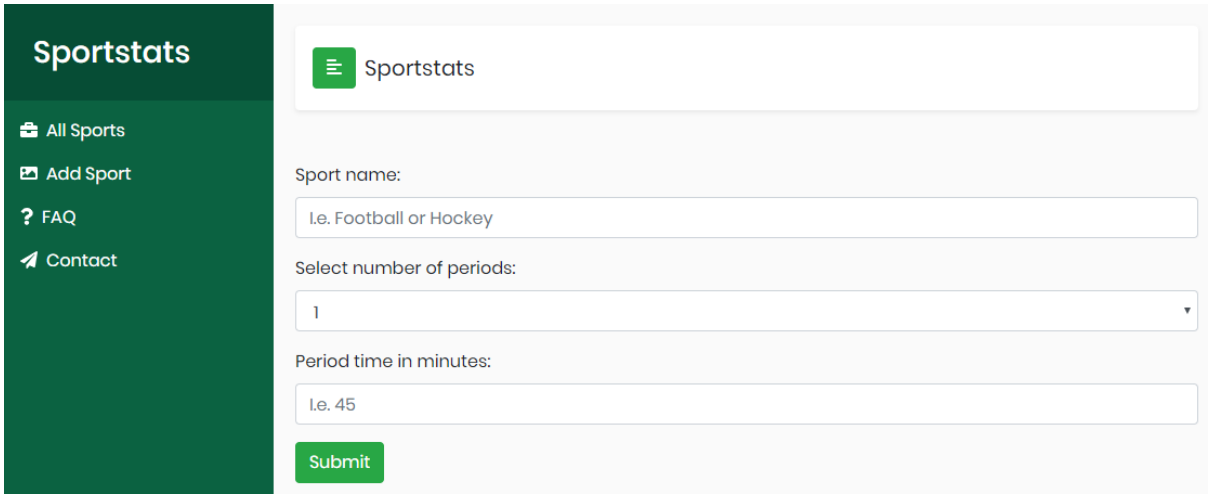
GitLab användes för systemet under hela processen med jämna uppdateringar av koden via först en *developer*-bransch som eventuella felfria ändringar eller tillägg av systemet lades till via vardera programmerares egna branscher. I slutet av varje pass gjordes det en kontroll av det som nu fanns i *developer*-branschen innan det slutligen skickades upp till *master*-branschen. Därefter hölls en liten kort sammanfattning av vad som gjorts och vad som planeras att göras senare, innan dagen avslutades.

3 Resultat

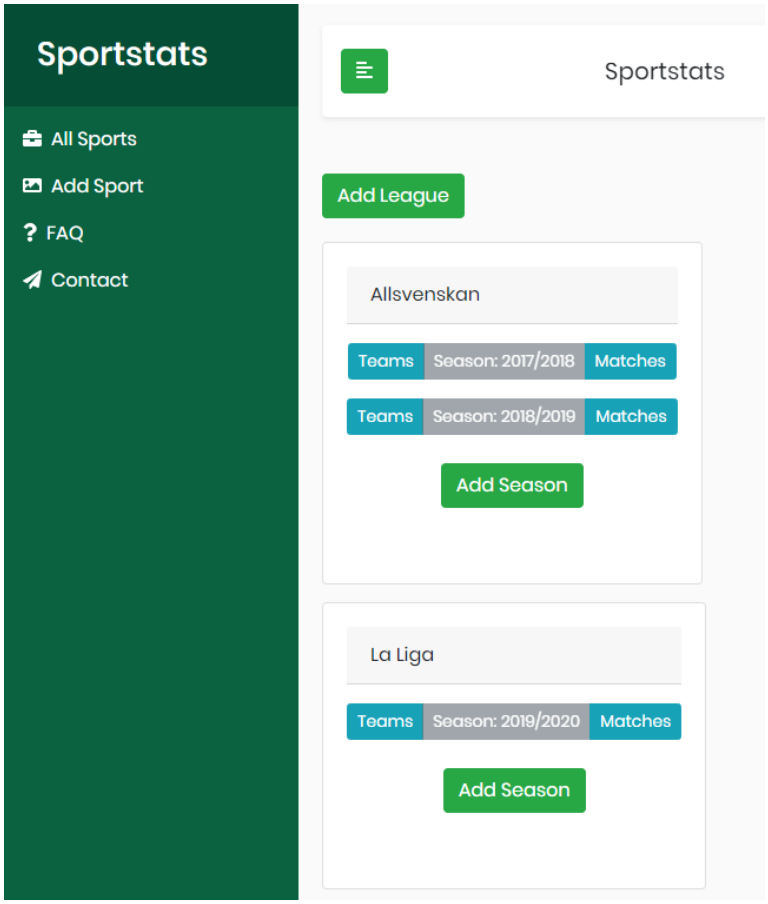
Detta projekt resulterade i ett system som nu finns distribuerat på Internet. Alla funktionaliteter som var möjliga att implementera som dokumenterades i det giva API:et har implementerats. Sidan har också kontrollerats så att den är responsiv och anpassas automatiskt till olika skärmstorlekar. Nedan finns bilder på exempel på det nuvarande systemets utseende, samt dess filstruktur som visas i Figur 9. Sidan har också publicerats via GitLab Pages och finns att nå här. För att API-hämtningen ska fungera krävs ett tillägg till Google Chrome som fixar problemen med CORS som kommer från server-sidan. Den som använts under utvecklingen är heter ”Allow CORS: Access-Control-Allow-Origin”. Anropen fungerar också endast om sidan läses in med protokollet http, eftersom API:et inte tillåts köras i https.



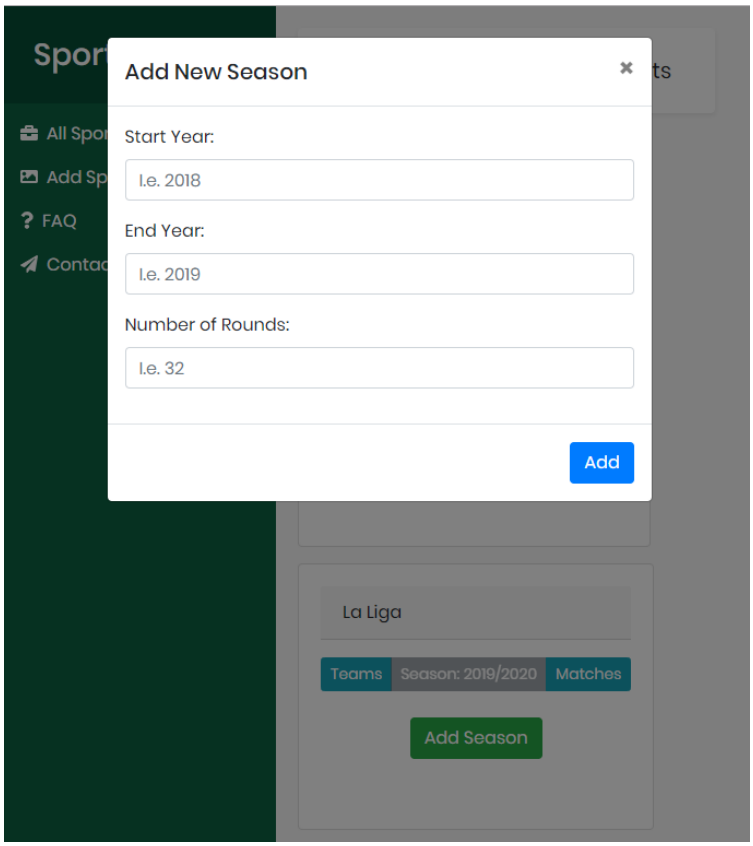
Figur 1: Huvudsidan som listar sporter



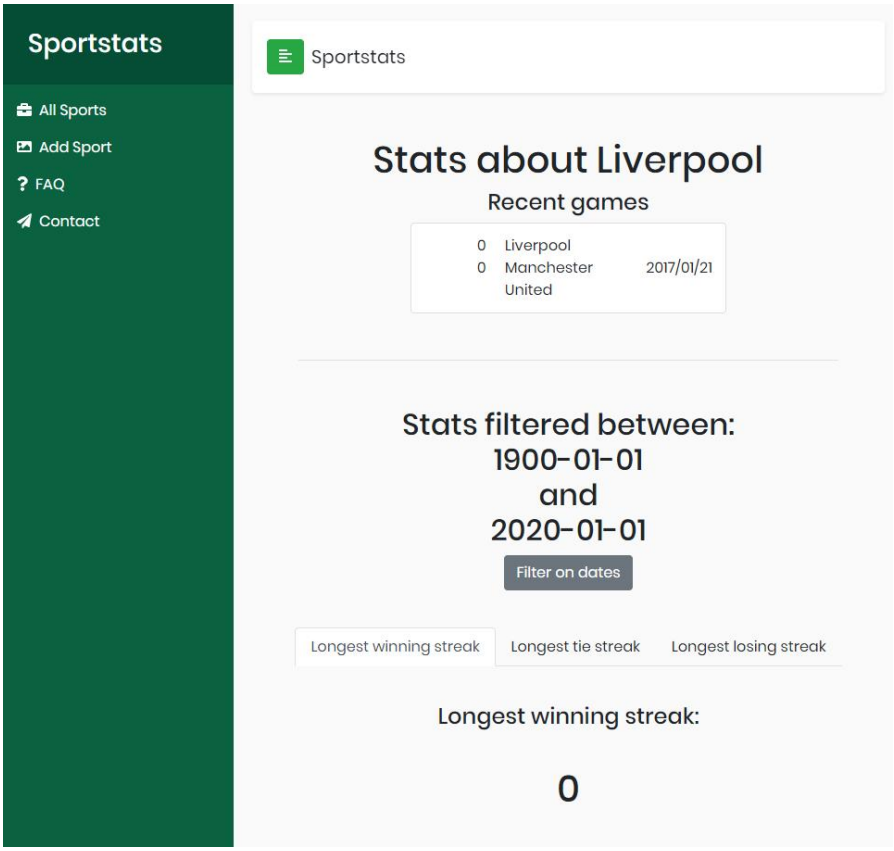
Figur 2: Sidan för Add Sport



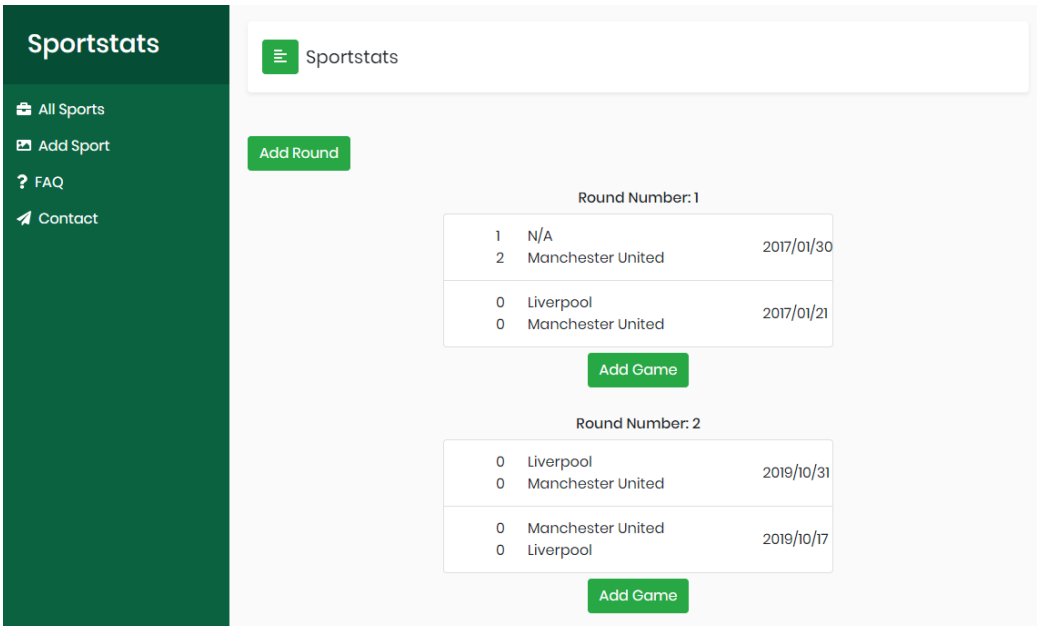
Figur 3 Ligor för sporten Fotboll



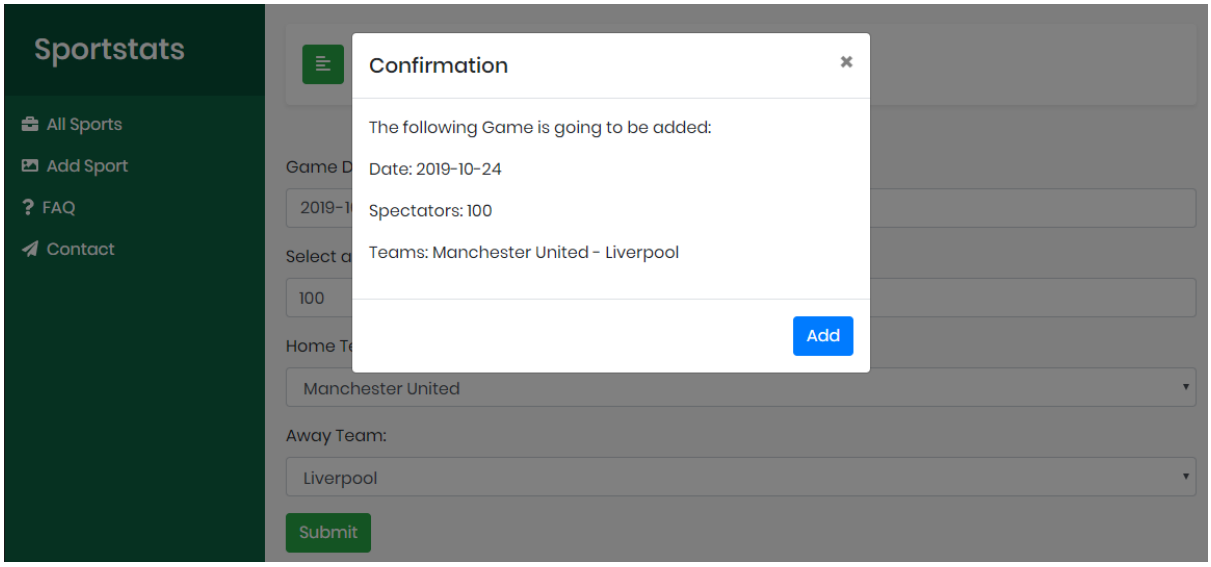
Figur 4 Modal som visas när Add Season knappen trycks



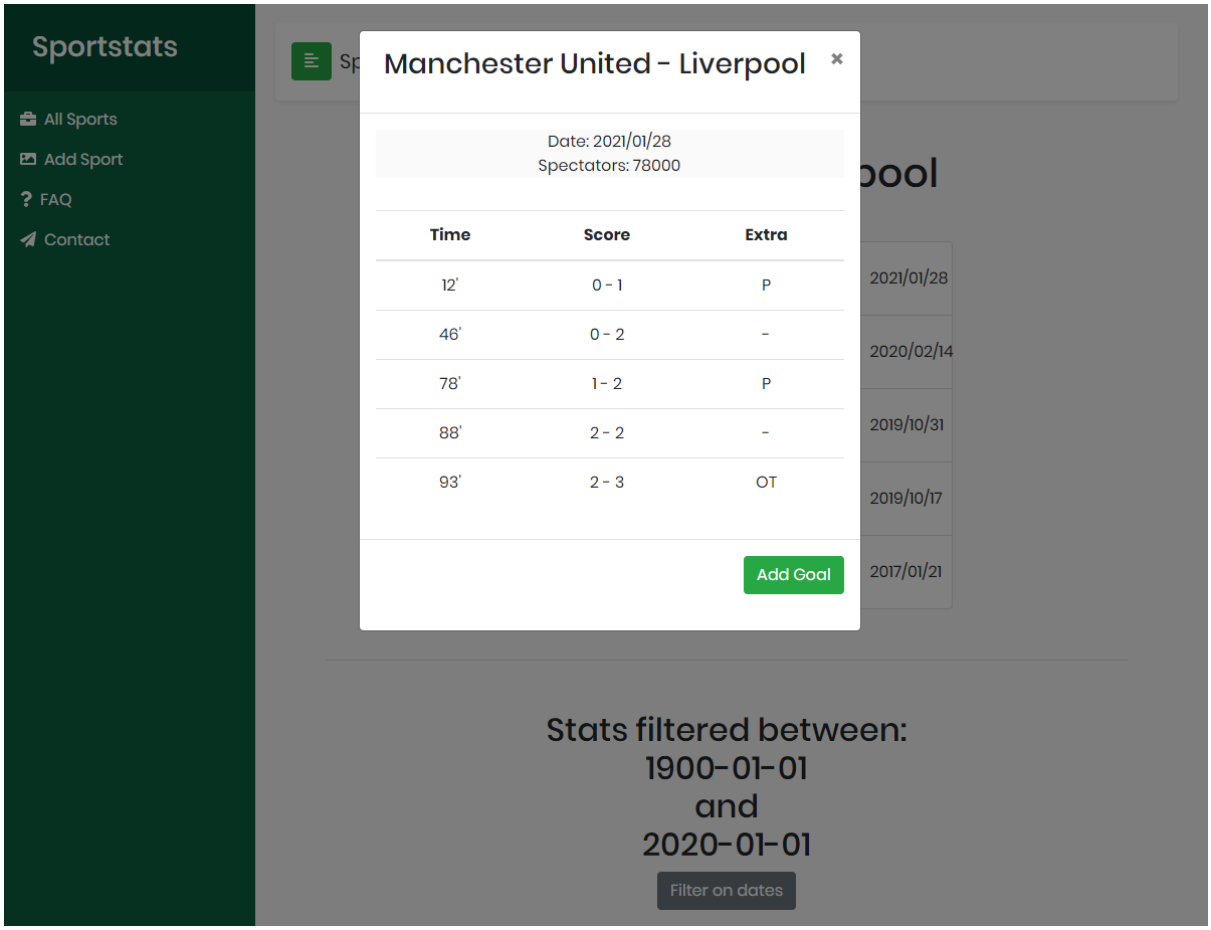
Figur 5 Statistik för ett specifikt lag



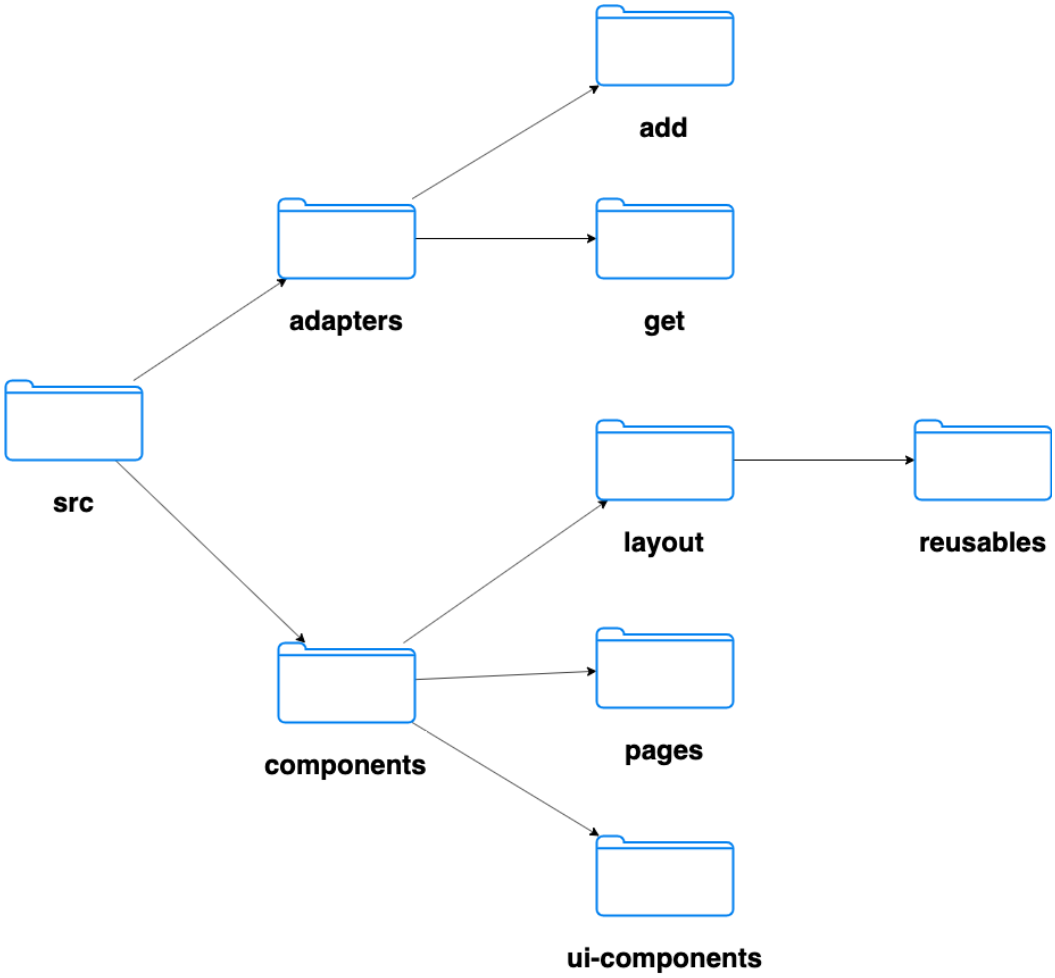
Figur 6 Matcher baserat på runda



Figur 7 Konfirmation av tillägg av nytt data



Figur 8 Statistik för en match



Figur 9 Filstrukturen på systemet

4 Diskussion

Nedan kommer vi redogöra för några av de problem vi stött på under utvecklingen av projektet och hur vi löst dem.

4.1 Async

Ett problem som uppstod kontinuerligt under projektet var att några av Reacts inbyggda funktioner kördes async. Ibland kunde det hända att när vi försökte ändra en komponents state med metoden *setState* hann den inte köra färdigt innan nästa funktionsanrop påbörjades. Detta skapade problem när funktionen som kördes efteråt var beroende av det state som sattes på raden innan, då den istället hämtade de tidigare inaktuella värdena. Lösningen till detta problem var att använda oss av callbacks när vi satte en komponents state. Det innebär att de anrop som är beroende av att rätt state har blivit satt inte körs förrän efter det är garanterat att state har hunnit blivit satt.

4.2 Reactstrap vs Bootstrap

Fastän Reactstrap var ett CSS ramverk, baserat på Bootstrap för just React, kunde det ibland uppstå problem med dess komponenter. Dessa hade främst med attributet *className*, som ska vara ekvivalent till Bootstraps *class*, att göra. Problemet var att komponenten ibland inte reagerade på attributen. Lösningen för att åtgärda detta problemet var att använda sig av *div*-taggar med Bootstrap *class* istället för att använda Reactstrap.

4.3 http vs https

Ett problem som uppstod, och som fortfarande finns, är att API:et körs med protokollet http, medan GitLab Pages kör med https. Detta gör att innehållet som hämtas från API:et blockeras av GitLab på grund av att protokollen inte stämmer överens. En tillfällig lösning på detta var att sätta om några inställningar i Gitlab så att sidan inte kräver användandet av https. När detta var gjort kunde vi manuellt ändra länken till webbsidan till http istället för https och på så vis började anropen att fungera.

4.4 API

Back-end systemet som vi tilldelades hade sammanlagt 46 dokumenterade funktionaliteter. Ett antal av funktionerna dokumenterade i API:et kunde vi inte använda på grund av att formatet som returnerades inte var av typen JSON. Utav dem som faktiskt returnerade data i JSON-format implementerades samtliga funktioner. Det fanns även ytterligare funktioner som vi skulle haft nytta av som inte fanns att tillgå från API:et, så detta fick istället lösas med hjälp av att återanvända och sammanfoga flera API-anrop. Ett exempel på ett sådant anrop var då vi ville få tag på ett specifikt lagnamn via respektive lags unika id. Eftersom det inte fanns något färdigt anrop för detta gjordes det genom att hämta alla lag för en säsong, och sedan plocka ut alla id-namn-par från den säsongen. Därefter jämfördes lagets id med samtliga id:n från listan tills en matchning hittades.