

Projektuppgift

Webbutveckling III

Webbtjänst
Projekt

Albin Rönkvist



Mittuniversitetet

MID SWEDEN UNIVERSITY

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Albin Rönnkvist, alrn1700@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: HT, 2018

Sammanfattning

I detta projekt har jag skapat en RESTbaserad webbtjänst. Webbtjänsten konsumeras på en webbplats där jag skapat funktionalitet för att läsa in, skapa, uppdatera och radera resurser. I rapporten går jag igenom uppgiften, relevant teori för att lösa uppgiften, vilka metoder jag använde och hur jag konstruerade webbtjänsten och webbplatsen. Jag går även igenom resultatet av det jag skapat för att sedan diskutera det tillsammans med projektets genomförande.

Innehållsförteckning

Sammanfattning.....	iii
1 Introduktion.....	1
1.1 Bakgrund och problemmotivering.....	1
1.2 Avgränsningar.....	1
1.3 Detaljerad problemformulering.....	1
1.3.1 Planering.....	1
1.3.2 Skapande.....	2
2 Teori.....	3
2.1 Git.....	3
2.2 Sass.....	4
2.3 Gulp.....	4
2.4 TypeScript.....	4
2.5 Webbtjänster och REST.....	4
3 Metod.....	6
3.1 Planering.....	6
3.2 Skapande.....	6
4 Konstruktion.....	7
4.1 Planering.....	7
4.2 Skapande.....	10
5 Resultat.....	16
6 Slutsatser.....	17
Källförteckning.....	18

1 Introduktion

1.1 Bakgrund och problemmotivering

Webben används mer och mer för kommunikation mellan applikationer. Gränssnittet som gör detta möjligt kallas webbtjänster vilket är mjukvarusystem som stöder interoperabel maskin-till-maskin-interaktion över olika typer av nätverk.

I detta projekt ska jag skapa en webbtjänst i form av en poängliga som sedan ska konsumeras på en webbplats där användare dynamiskt ska kunna göra ändringar.

1.2 Avgränsningar

Denna rapport kommer gå in på ämnen som: Git, SASS, Gulp, TypeScript och Webbtjänster. Förkunskaper inom CSS, HTML, PHP och JavaScript är en fördel då rapporten berör dessa ämnen men inte beskriver de ingående.

I brist på tid kommer det inte finnas något inloggningssystem på webbplatsen och inte heller någon typ av autentisering för att kommunicera med webbtjänsten. Detta är därför ingen komplett webbplats utan endast ett exempel på hur man kan jobba med webbtjänster.

1.3 Detaljerad problemformulering

1.3.1 Planering

Först måste jag göra en planering över webbtjänstens och webbplatsens utformning. Hur ska webbtjänsten fungera, vad ska lagras i databasen, hur ska webbplatsen se ut samt vilka funktioner ska finnas på webbplatsen.

Webbtjänst & databas: webbtjänsten ska hantera en poängliga med spelarstatistik. I webbtjänsten ska det finnas metoder för att skapa, hämta, uppdatera och radera resurser. Man ska kunna skapa en spelare med dess namn och lag samt kunna lägga till / ändra antalet mål och assist som spelaren har gjort. Målen och assisten räknas sedan ihop till ett sammanlagt antal poäng. Man ska alltså kunna skapa och ändra / uppdatera en spelare med tillhörande information samt kunna radera enskilda spelare tillsammans med dess information. Man ska även kunna hämta alla spelare och deras antal gjorda mål, assist samt total poäng, till exempel i form av en lista.

I databasen ska spelare med ovanstående information lagras i en tabell som ska kunna gå att anslutas till via webbtjänsten.

Webbplats: webbtjänsten med poängligan samt ovanstående funktionalitet ska sedan konsumeras och presenteras på webbplatsen där listan uppdateras dynamiskt och användare kan göra ändringar.

1.3.2 Skapande

Webbplatsen, webbtjänsten och databasen ska sedan skapas utifrån föregående planering. Webbplatsen ska vara konstruerad så att den enkelt kan flyttas från lokal utvecklingsmiljö till en publik webbhost. Samtliga filer som behövs för att installera och köra webbplatsen måste även finnas tillgängliga i ett remote repository på Github eller motsvarande hosting-tjänster.

2 Teori

2.1 Git

Git är ett versionshanteringssystem som är skapat främst för mjukvaruutvecklare och hantering av kod, till en början för att hantera källkoden till operativsystemskärnan Linux. Ett versionshanteringssystem är till för att hålla reda på historik och förändringar i ett arbete / projekt. Man kan backa tillbaka till tidigare versioner och se vem som gjort vad i projektet. Utöver det så möjliggör systemet även för flera utvecklare att kunna arbeta på samma projekt oavsett om de arbetar på samma plats eller är utspridda över hela världen [9]. Nedan följer ett antal Git-begrepp som kommer användas i rapporten:

- **Workspace:** är i princip en kopia av en *repository* som jag jobbar med lokalt, en ”work in progress”. Alla ändringar som jag gör sker i mitt *workspace* och sparas lokalt på min datorn i till exempel en projektmapp. Sedan finns det en process för att kopiera dessa ändringar till ett *lokalt repository* för att sedan kopiera de till en *remote repository* där andra kan ta del av mina ändringar. [8]
- **Staging area:** i detta steg har filer från *workspace* lagts till i *index-filen / staging area*. Detta är inte den officiella versionen av projektet utan filerna ligger endast och väntar på att bli tillagda(*committed*) i den officiella versionen. Man kan sedan välja vilka filer man vill *commita* och vilka som inte är riktigt färdiga ännu och får ligga kvar i *workspace*. [8]
- **Repository:** kan förkortas till ”repo” och är oftast ett projekt. Man kan likna det med en mapp/katalog innehållande filer och undermappar som kan versionshanteras. När man initierar Git i en projektmapp så skapar man alltså ett repo där filerna kan versionshanteras . Här finns även 2 olika repos, *lokal repo* och *remote repo*.

Lokala repot kan ses som den officiella och senaste versionen av projektet som ligger lokalt på datorn. När man *commitar* filer från *staging area* så gör man det till ett *lokalt repository*. Vill man dela repot med andra så kopierar man det till ett *remote repo*. Då laddar man i princip bara upp det *lokala repot* på en fjärrserver som kan nås över internet eller nätverk. Detta möjliggör för flera utvecklare att kunna samarbeta på ett projekt och dela med sig av sitt arbete. Användare kan till exempel hämta(pull) och ladda upp(push) data samt se vem som gjort ändringar. GitHub är ett exempel på en webbtjänst för hosting av *remote repositories* som även kommer användas i detta projekt. [8]

2.2 Sass

Sass (*Syntactically Awesome Style Sheets*) är en CSS preprocessor, alltså ett skriptspråk som förlänger CSS genom att tillåta utvecklare skriva kod i ett språk som sedan kompileras till CSS. Med hjälp av *Sass* kan man då skapa saker som variabler, nesting, mixins, extends, if/else-satser och imports. Dessa faktorer leder till att filerna blir mer organiserade och går fortare att skapa. [10]

I *Sass* kan man välja mellan att använda 2 olika syntax, *SCSS* och *Indented*. I projektet kommer jag använda *SCSS*-syntax med `”.scss”` som filändelse eftersom det är fullt kompatibelt med CSS-syntax. *Indented* använder `”.sass”` som filändelse och går fortare att skriva men är inte fullt kompatibel med CSS-syntax. [10]

I detta projekt kommer verktyget Gulp användas för att kompilera Sass till CSS.

2.3 Gulp

Gulp är ett verktyg baserat på Node.js och npm som är till för att automatisera sysslor så som kompilering, konkatenering och minifiering av filer. Till exempel kompilering från Sass till CSS, konkatenering av JavaScript-filer och minifiering av bilder eller HTML-filer. Med detta arbetssätt kan man utveckla kod som sedan automatiskt görs redo för publicering.

2.4 TypeScript

TypeScript är ett programspråk utvecklat av Microsoft som skapades för att ersätta JavaScripts brister när det kommer till att utveckla större applikationer. Problem uppstod med komplex JavaScript-kod vilket ledde till en stor efterfrågan av ett verktyg som underlättar utveckling av komponenter i det språket [22].

TypeScript är baserat på samma syntax som JavaScript men det är inte samma sak, det är en så kallad ”superset” med funktioner som överstiger det som är möjligt i JavaScript. Filerna måste kompileras till JavaScript som då kan köras i webbläsare, Node.js eller vilken JavaScript-motor som helst som stöder standarden ECMAScript 3 eller senare. [23]

Språket stöder funktioner så som klasser, moduler, namespaces, type annotations, interfaces och mycket mer. Dessa funktioner förenklar utvecklandet av större applikationer och skapar hållbar kod. [22]

2.5 Webbtjänster och REST

En webbtjänst kan definieras på följande sätt [6]:

- En applikation för kommunikation mellan klient och server.
- En metod för kommunikation mellan 2 enheter över nätverk.

- Ett system för interoperabel maskin till maskin kommunikation. Till exempel kan webbtjänsten implementeras i PHP men anropas från en klient skriven i JavaScript.
- En samling standarder eller protokoll för utbyte mellan två enheter eller applikationer.

Det finns huvudsakligen 2 typer av webbtjänster, SOAP- och RESTbaserade webbtjänster. I detta projekt kommer jag använda REST eftersom det anses lämpa sig bäst när man behöver skapa en grundläggande webbtjänst på kort tid då REST-metoder enkelt kan testas i webbläsare och inlärningskurvan är liten eftersom man jobbar med HTTP som protokoll. Jag kommer även jobba med JSON-formaterad data vilket SOAP inte har stöd för. [7]

REST står för "Representational State Transfer" och är en arkitekturstil för utveckling av applikationer som kan nås över nätverk [7]. En RESTbaserad webbtjänst är en webbtjänst som visar sin data och funktionalitet i form av resurser som klienter kan manipulera. Resurserna anpassar sig efter ett antal principer [5][7]:

- Alla resurser är unikt adresserbara, ofta genom olika URI som är en kompakt sträng som används för att just identifiera eller namnge en resurs [3].
- Alla resurser kan bli manipulerade genom ett begränsat antal funktioner, ofta CRUD(Create, Read, Update, Delete), som ofta presenteras med HTTP-verbena POST, GET, PUT och DELETE.
- Resursernas data överförs via ett begränsat antal dataformat, ofta genom JSON, XML eller HTML.
- Kommunikationen mellan klient och webbtjänst sker över ett protokoll som är "stateless", ofta HTTP. Detta innebär att webbtjänsten inte lagrar någon information om klientens session på server-sidan. Varje begäran som klienten gör måste därför innehålla alla nödvändiga detaljer som krävs för att servern ska kunna förstå och förse klienten med rätt innehåll [4].

3 Metod

3.1 Planering

Webbtjänst & databas: webbtjänsten kommer beskrivas utförligt under konstruktion-delen. Databasen med dess tabeller kommer beskrivas med hjälp av ER-diagram.

Webbplats: webbplatsens layout kommer presenteras med hjälp av wireframes. Webbplatsen kommer vara responsiv men layouten kommer vara så lika på alla olika skärmstorlekar så det kommer endast skapas wireframes för desktop-enheter. Webbplatsens funktioner kommer beskrivas både med text och flödes-scheman för att få en tydlig bild över upplägget.

3.2 Skapande

Till att börja med kommer jag skapa två separata Git repositories, en för webbtjänsten och en för webbplatsen som ska konsumera webbtjänsten. Repot för webbplatsen kommer publiceras i ett remote repo på GitHub. Repot för webbtjänsten kommer stanna lokalt på grund av känslig information kring databasen.

Webbtjänst & databas: webbtjänsten kommer skapas i PHP med grundläggande HTTP-metoder utifrån beskrivningen i planeringen. Jag kommer använda tillägget RESTClient[2] i Firefox för att testköra min webbtjänst innan jag konsumerar den på webbplatsen. Databasen kommer skapas med verktyget phpMyAdmin [1] utifrån databasbeskrivningen i planeringen.

Webbplats: Webbplatsen kommer skapas i HTML och designas med SASS-genererad CSS-kod. Webbtjänsten kommer konsumeras med hjälp av TypeScript eller JavaScript till HTML-dokumentet. Jag kommer att använda verktyget Gulp för att automatisera sysslor. Webbplatsen kommer testas i några av de mest använda webbläsarna samt på olika enheter. Webbplatsens HTML- och CSS-kod kommer även testas med verktygen *CSS Validation Service*[20] och *Nu Html Checker*[21] för att kunna lösa eventuella fel och varningar.

4 Konstruktion

4.1 Planering

Webbtjänst & databas: I databasen lagras en lista med spelare. Dess namn, vilket lag de tillhör och dess gjorda poäng. Det som lagras är: *id*, *namn*, *lag*, *mål*, *assist* och totalt antal *poäng* (Se Figur 1).

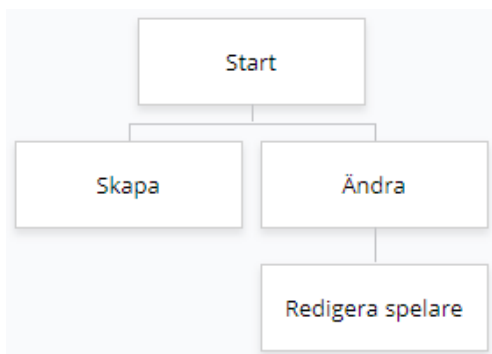
spelare
id: INT(11), A_I, (PK)
Fullname: VARCHAR(64)
Team: VARCHAR(64)
Goals: INT(4)
Assists: INT(4)
Points: INT(4)

Figur 1

I tabellen spelare lagras spelare och deras information. *Id* är primärnyckel och består av en unik siffra som automatiskt genereras. *Name* och *Team* lagras som en sträng med max 64 tecken. *Goals*, *Assists* och *Points* lagras som siffror med max 4 siffrors längd.

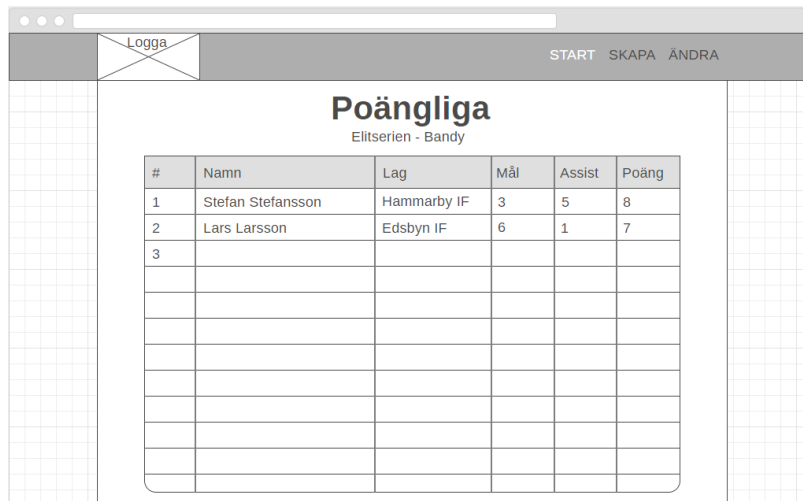
Webbtjänsten ska ansluta till databasen. I webbtjänsten ska det finnas metoder för att skapa, hämta, uppdatera och radera resurser. Webbtjänsten ska vara RESTbaserad och använda JSON-formaterad data för kommunikation med klienter.

Webbplats: Webbplatsen består av fyra sidor: startsida(*index.html*), ”skapa”-sida(*create.html*), ”ändra”-sida(*edit.html*) och ”rediga spelare”-sida(*edit-players.html*) (se Figur 2).



Figur 2

- *Index.html*: På startsidan visas en tabell med alla spelare och deras poäng i fallande ordning, flest poäng högst upp (se wireframe i Figur 2).

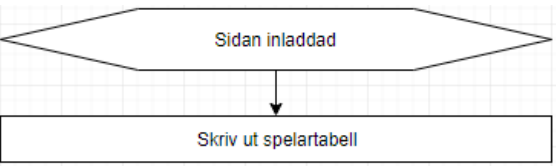


The wireframe shows a web browser window with a header containing a 'Logga' button and navigation links 'START', 'SKAPA', and 'ÄNDRA'. The main content area is titled 'Poängliga' with the subtitle 'Elitserien - Bandy'. Below the title is a table with the following data:

#	Namn	Lag	Mål	Assist	Poäng
1	Stefan Stefansson	Hammarby IF	3	5	8
2	Lars Larsson	Edsbyn IF	6	1	7
3					

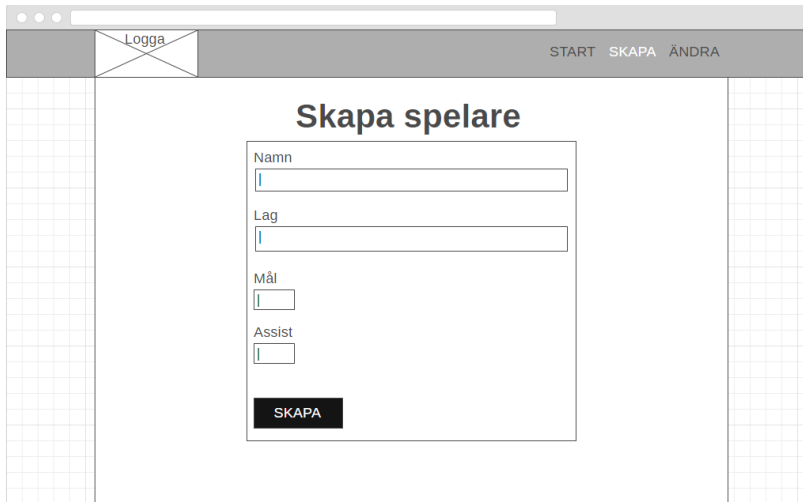
Figur 2

Denna tabell ska hämtas från webbtjänsten med JavaScript och skrivas ut efter inladdning av sidan (se flödesschema i Figur 3).



Figur 3.

- *create.html*: På skapa-sidan kommer det finnas ett formulär för att skapa ny spelare. Ett input-fält för namn, ett select-element för att välja vilket lag och 2 input-element för mål och assist (se wireframe i Figur 4).

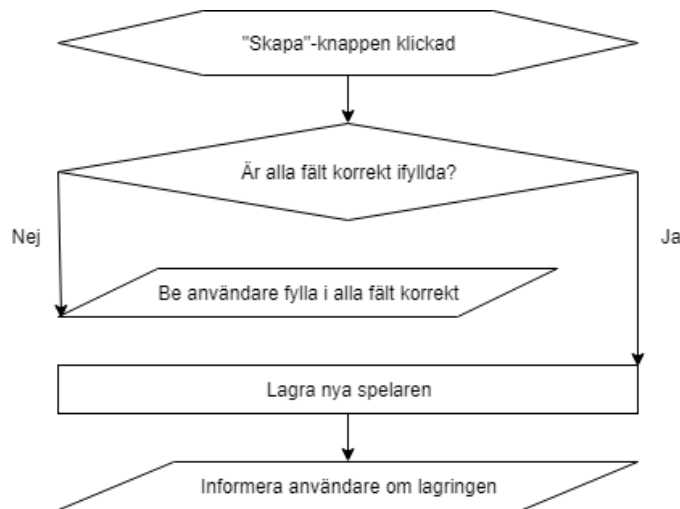


The wireframe shows a web browser window with a header containing a 'Logga' button and navigation links 'START', 'SKAPA', and 'ÄNDRA'. The main content area is titled 'Skapa spelare'. Below the title is a form with the following fields:

- Namn:
- Lag:
- Mål:
- Assist:
- SKAPA:

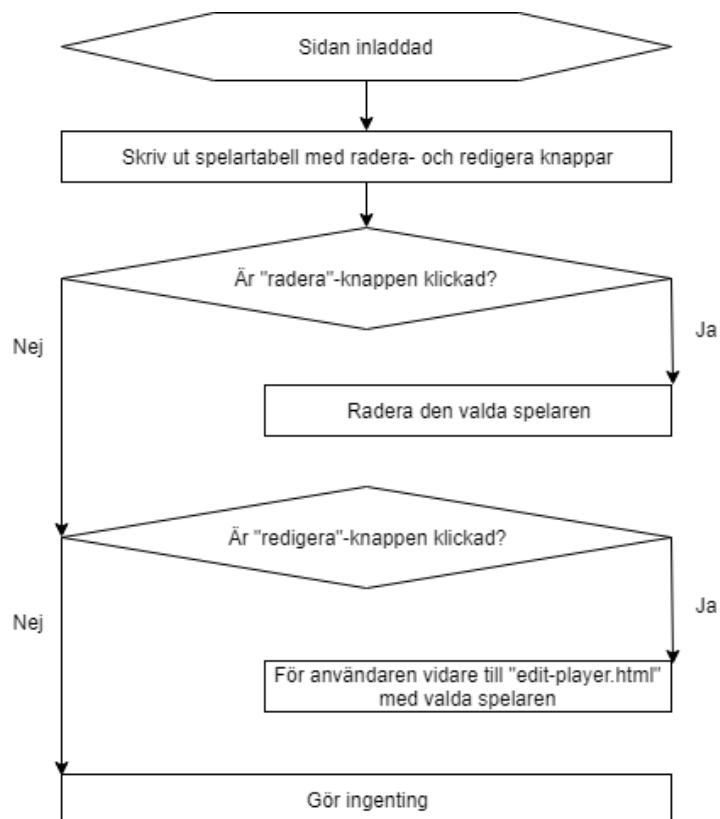
Figur 4

Vid klick på skapa kommer en ny spelare lagras med de värden som anges i formuläret (se flödesschema i Figur 5).



Figur 5.

- *edit.html*: På ändra-sidan visas samma lista som på startsidan (se Figur 2) fast med knappar till vänster om varje enskild spelare för att radera och redigera spelare. Efter inladdning visas tabellen med knappar. Vid klick på radera tas spelaren och all dess information bort. Vid klick på redigera förs användaren vidare till *edit-player.html* med ett id som matchar den spelare man vill redigera (se Figur 6).



Figur 6

- *edit-player.html*: På denna sida finns funktionalitet för att ändra information om spelaren. Sidan är lik ”skapa”-sidan (se Figur 3), skillnaden är att alla fält är ifyllda med den valda spelarens information och man uppdaterar istället för skapar. Efter inladdning fylls fälten i formuläret i automatiskt med spelarens information. Vid klick på uppdatera så lagras spelaren med de nya värdena som anges i formuläret likt flödesschemat i Figur 5.

4.2 Skapande

Webbtjänst & databas: jag började med att skapa databasen lokalt i phpMyAdmin utifrån tidigare planering (se Figur 1). Jag skapade sedan en egen mapp för webbtjänsten där jag initerade Git med kommandot ”`git init`” för att möjliggöra versionshantering.

Jag började med att skapa funktionalitet som listar ut vad olika begäranden innehåller. Jag behöver veta: metoden av begärandet, URL:en av begärandet och eventuell data som kom med det. För detta använde jag PHP:s superglobala variabel `$_SERVER`.

- `$_SERVER['REQUEST_METHOD']` för att returnera begärande-metoden som användes för att ansluta till webbtjänsten [11], i detta fall HTTP-metoderna *GET*, *POST*, *PUT* eller *DELETE*. Metoden lagras i variabeln *\$method* som används senare.
- `$_SERVER['PATH_INFO']` för att se i vilken del av URL:en användaren gjorde sin begäran och på så sätt se vad den begär samt hur man ska agera utifrån begäran [11]. Variabeln tar bort hela sökvägen till och med filnamnet vilket gör att endast ”frågesträngar”(query strings) visas [11] [16]. Jag delar sedan in dessa frågesträngar i en array med substrängar med hjälp av funktionen `explode()`. Jag lagrar denna array i variabeln *\$request* som jag kan använda för att se vilken resurs användaren efterfrågar.
I detta fall kan till exempel endast resursen ”*spelare*”(databastabellen) anropas. Detta kollar jag genom att se om första substrängen i arrayen är resursen ”*spelare*” eller inte: `if($request[0] != "spelare")`.
- För att läsa in data som skickas med i begärandets body(request body) använder jag filesystem-funktionen `file_get_contents()` med `'php://input'` som parameter vilket läser in JSON-data som en sträng [12][13]. Efter det använder jag JSON-funktionen `json_decode()` för att omvandla JSON-strängen till ett objekt / array som kan användas i PHP [14].

Efter det anger jag att innehållet ska vara i JSON-format, kodad i teckenkodningen UTF-8. Detta görs med HTTP-headern: `header("Content-Type: application/json; charset=UTF-8");`. När detta är klart kan jag ansluta till databasen jag skapade tidigare. Detta gör jag med funktionen `mysqli_connect()`

där jag även anger vilken databas som ska vara standard för alla SQL-frågor med funktionen `mysqli_select_db()`. Jag anger även att UTF-8 ska användas som standard teckenuppsättning när data skickas från och till databasservern, detta görs med funktionen `mysqli_set_charset()` [15].

När jag hade funktionalitet för att förstå ett begärande och även en databas att ansluta till så implementerade jag funktionalitet för de tidigare nämnda HTTP-metoderna *GET*, *POST*, *PUT* och *DELETE*. Det är dessa metoder som användare kommer använda för att ansluta till min webbtjänst, en metod för varje begäran. Jag skapade då en switch-sats som tar emot ett värde från tidigare nämnda variabel `$method` som sedan delas in i 4 cases, en för varje HTTP-metod. Beroende på vilken begärande-metod som användes så ändras en SQL-sats enligt följande:

- *GET*: SQL SELECT-sats med ORDER BY för att läsa in spelare med flest poäng först.
Här anger jag även att om användaren försöker hämta en resurs med ett specifikt id så hämtas endast den resursen. Detta gör jag genom att kolla andra substrängen i arrayen i tidigare nämnda variabeln `$request`:

```
if(isset($request[1])) $sql = $sql . " WHERE id = " . $request[1] . ";"
```
- *PUT*: SQL UPDATE-sats för att uppdatera spelare.
- *POST*: SQL INSERT INTO-sats för att lägga till ny spelare.
- *DELETE*: SQL DELETE-sats för att ta bort spelare.

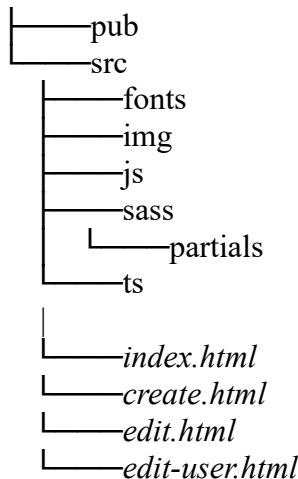
I varje case exekveras en SQL-sats som sedan används för att skicka en fråga till databasen med hjälp av funktionen `mysqli_query()`.

Jag vill sedan att webbtjänsten ska svara / överföra data i form av JSON-arrays vilket enkelt kan konsumeras med JavaScript senare [17].

Jag börjar då med att skapa en tom JSON-array. Sedan skapade jag en while-loop som loopar igenom databasen utifrån SQL-satsen och hämtar de resulterande raderna från databastabellen i form av en associativ array med hjälp av funktionen `mysqli_fetch_assoc()`. Elementen från denna array läggs sedan till i den tomma JSON-arrayen jag skapade tidigare med hjälp av funktionen `array_push()`. Jag stänger sedan databaskopplingen med funktionen `mysqli_close()` för att förhindra att flera öppna anslutningar påverkar webbtjänstens prestanda. Jag använder sedan funktionen `json_encode()` för att presentera värdena i JSON-format.

Jag har nu skapat en RESTbaserad webbtjänst som kan ta emot HTTP-begäranden och svara med JSON-formaterad data. Jag testade webbtjänstens funktionalitet med hjälp av RESTClient[2] innan jag modifierade koden och databasen för att publicera den på en publik webbserver. Jag lade till (`git add .`) och committade (`git commit`) alla ändringar till repot som då blir min officiella version av webbtjänsten.

Webbplats: när jag var klar med webbtjänsten så skapade jag webbplatsen som ska konsumera den. Det gjorde jag i en ny mapp där jag initierade Git för att möjliggöra versionshantering. Jag började med att definiera filstrukturen där jag delade in utvecklings-filerna(src) och publicerings-filerna(pub) i två olika mappar:



I src-mappen ligger alla utvecklings-filer: typsnitt, bilder, JavaScript för konsumering av webbtjänst och övrigt, Sass för design, TypeScript för viss funktionalitet samt alla HTML-filer från min sitemap (se Figur 2). För att kunna optimera och kopiera dessa filer till pub-mappen så använde jag mig av pakethanteraren *NPM* och automatiseringsverktyget *Gulp*.

Jag skapade filen *package.json* i mappen med kommandot: `npm init`. Filen innehåller metadata om projektet som till exempel namn, beskrivning och licens. Filen hanterar även projektets olika *paket*(dependencies) [18].

Jag installerade sedan *Gulp* i mappen med kommandot: `npm install gulp --save-dev`.

Jag skapade sedan filen *gulpfile.js* i mappen som är till för att skapa tasks. Dessa tasks utför sedan någon typ av automatisering som till exempel kompilering av Sass.

I *gulpfile.js* lade jag först till alla *paket* som används(samt installerade dessa):

- **Gulp** - jag använder Gulp som verktyg för att automatisera utvecklar-sysslor.
- **gulp-concat** - sammanslår filer. Jag använder detta paket till att sammanslå mina JavaScript- och Sass-filer till endast en av varje.
- **gulp-uglify** - minifierar JavaScript-filer. Tar bort till exempel kommentarer, radbrytningar och indenteringar.
- **gulp-clean-css** - minifierar CSS-filer på samma sätt som gulp-uglify.
- **gulp-htmlmin** - minifierar HTML-filer.

- **gulp-imagemin** - minifierar bilder(PNG, JPEG, GIF & SVG).
- **gulp-sass** - Sass-plugin för Gulp.
- **gulp-sourcemaps** - Skriver inline source maps för att kunna se vilka filer viss kod kommer från i en konkatenerad fil.
- **gulp-typescript** – plugin för att kompilera TypeScript.

Sedan skapade jag alla tasks:

- **copyhtml** - minifierar HTML-filer och kopierar de till publiceringsmappen
- **copyimages** - minifierar bilder och kopierar de till publiceringsmappen.
- **convertjs** - sammanslår och minifierar JavaScript-filer och kopierar till publiceringsmappen samt genererar sourcemaps.
- **convertsass** - konverterar, konkatenerar och minifierar Sass-filer till en CSS-fil samt genererar sourcemaps.
- **copyfonts** - kopierar typsnitt till publiceringsmappen.
- **convertts** - kompilerar, konkatenerar och minifierar TypeScript-filer till en JavaScript-fil samt genererar sourcemaps.
- **watcher** - kollar efter ändringar i filer och kör automatiskt tasks som är kopplade till den filen som ändras.
- **default** - kör alla angivna tasks samtidigt så att man slipper köra de en för en. Detta kan enkelt initieras med kommandot "gulp" som då kör alla angivna tasks. Efter initiering av "default" kommer den köras automatiskt varje gång systemet uppdateras med hjälp av task:en "watcher".

Med hjälp av dessa task så optimeras utvecklings-filerna automatiskt vid sparning och läggs till i pub-mappen, redo för publicering.

När automatiseringsprocessen var klar så började jag skapa webbplatsen sida för sida utifrån föregående planering (se Figur 3 – 6). Jag använde AJAX-anrop (XMLHttpRequest-objektet) för att kommunicera med webbtjänsten där jag genom XMLHttpRequest-metoden *send()* skickar olika begäranden med valda HTTP-begärande-metoder(tidigare-nämnda *\$method*) och URL:er(tidigare-nämnda *\$request*) för att hämta och påverka resurser.

Jag har skapat olika anrop på varje HTML-sida:

- *index.html*: Här skickar jag en GET-begäran till webbtjänsten med standard URL utan frågesträngar. Det jag får tillbaka är en lista med alla spelare och dess statistik i JSON-format vilket jag sedan parsar till ett JavaScript-objekt och skriver ut i en tabell.

Utöver begäran så har jag skapat funktionalitet för att sortera tabellen efter mål, assist och poäng.

- *create.html*: Här skickar jag en POST-begäran till webbtjänsten. Jag tar värdena från input-fälten i formuläret och lägger de i en korrekt formaterad begärande-”beskrivning”(request body). Jag omvandlar sedan denna beskrivning till en JSON-sträng och skickar begäran till webbtjänsten. Innan jag skickar min begäran så anger jag även att webbtjänsten kommer ta emot min request body som JSON-formaterad data med hjälp av XMLHttpRequest-metoden: `setRequestHeader('Content-Type', 'application/json')`.
- *edit.html*: Här hämtar jag och skriver ut spelare i en tabell likt som på *index-sidan*. Jag skriver dock även ut en knapp för att redigera och en för att radera för varje spelare i tabellen. Redigera-knappen är en länk som skickar vidare användaren till *edit-player-sidan* med den valda spelares id som frågesträng i webbläsarens URL. Vid klick på radera-knappen körs en funktion som skickar en DELETE-begäran till webbtjänsten med den valda spelarens id som frågesträng i begärande-URL:en.
- *edit-player.html*: här hämtas en spelare med en GET-begäran där begärande-URL:en tar spelar-id från frågesträngen i webbläsarens URL med hjälp av egenskapen: `location.search`. Spelarens information skrivs ut i formuläret där jag sedan tar de eventuellt nya värdena och skickar med de i en PUT-begäran likt POST-begäran som jag skickar på *create-sidan*. Enda skillnaden här är att jag använder HTTP-begärande-metoden ”PUT” istället för ”POST” som då uppdaterar resursen istället för att skapa en ny.

När webbplatsen var klar så kunde jag enkelt publicera den på en publik webbhost genom att ladda upp den automatiskt generade pub-mappen.

Jag lade sedan till(`git add .`) och commitade(`git commit`) alla ändringar till mitt lokala repo innan jag laddade upp det på GitHub med hjälp av kommandona: `git remote add origin [länk till remote repo].git` och `git push -u origin master`. Innan jag lade ändringar i repot så skapade jag filen `.gitignore` där jag exkluderade / ignorerade filer som inte var nödvändiga att ta med, till exempel pub-mappen som kommer genereras ändå och alla dependencies som går att installera med ett kommando.

Länkar:

Webbtjänstens URI:

<https://albinronnkvist.se/dt173g/projekt/webbtjanst/playerstats.php/spelare>

Länk till Webbplats:

<https://albinronnkvist.se/dt173g/projekt/webbplats/pub/index.html>

Länk till Remote Repo:

<https://github.com/albinronnkvist/poangliga>

Klona repot med följande kommando:

```
git clone https://github.com/albinronnkvist/poangliga.
```

5 Resultat

Webbtjänsten skapades i PHP. Den är RESTbaserad och jobbar med JSON-formaterad data mot en databas. I webbtjänsten finns metoder för att skapa, hämta, uppdatera och radera resurser. Man kan skapa en spelare med dess namn och lag samt lägga till antal mål, assist och poäng som spelaren har gjort. Man kan även ändra / uppdatera en spelare med specifik information samt radera enskilda spelare och dess information. Det går också att hämta alla spelare och deras antal gjorda mål, assist samt total poäng.

Webbplatsen är skapad i HTML och designad med Sass-genererad CSS-kod. Verktyget Gulp användes för automatisering av utvecklarsysslor och Git användes för versionshantering. Webbplatsens huvudsakliga funktionalitet är skapad med JavaScript och ansluter till webbtjänsten. Det finns funktioner för att dynamiskt skapa, hämta, uppdatera och radera webbtjänstens resurser. Man kan skapa en spelare med dess namn och lag samt lägga till antal mål och assist som spelaren har gjort. Målen och assisten räknas sedan ihop till ett sammanlagt antal poäng. Man kan även ändra / uppdatera en spelare med specifik information samt radera enskilda spelare och dess information. Spelare hämtas i olika tabeller med statistik eller övrig information.

Webbplatsen är testad enligt tidigare nämnda metod som resulterade i korrekt validerad HTML- och CSS-kod på alla sidor samt funktionalitet som fungerar i de vanligaste webbläsarna: Chrome, Safari, Firefox, Opera, Edge och IE 11[19]. Webbplatsen fungerar även korrekt på såväl desktop-enheter som på mobila enheter.

6 Slutsatser

Jag tycker projektet flöt på bra och jag stötte inte på några riktigt stora problem. De enda problemen som uppkom var att viss kod inte fungerade i Internet Explorer 11 men det var bara att ändra lite i koden vilket inte tog särskilt lång tid att lösa. Ett exempel är att hämta spelarens id från webbläsarens URL, jag hittade en bra lösning på det men den fungerade inte i Internet Explorer 11 vilket i detta fall ledde till en sämre temporär lösning.

Jag kände mig tillräckligt bekväm med TypeScript för att kunna använda det genom hela projektet och eftersom jag inte hade jättemycket tid så körde jag på med välbekanta JavaScript. Nu när projektet är klart och jag vet hur jag ska gå tillväga så skulle det vara spännande och nyttigt att lösa den igen fast med TypeScript.

Nu finns inget inloggningssystem eller autentisering för att ansluta till webbtjänsten vilket inte skulle fungera så bra om den skulle användas av ett företag eller liknande. Om jag skulle fortsätta med detta projekt så skulle jag definitivt fokusera på säkerhetsaspekter och inte tillåta alla att göra POST-, PUT- och DELETE-begäranden.

Jag skulle även kunna utöka webbplatsen och ha flera ligor, olika sökalternativ för till exempel lag och övrig statistik. Kanske till och med någon typ av matchrapportering som automatiskt lägger in statistik från varje match till tabellen.

Källförteckning

- [1] phpMyAdmin, "About" <https://www.phpmyadmin.net/> Hämtad 2018-10-15
- [2] Firefox, "RESTClient, a debugger for RESTful web services." <https://addons.mozilla.org/sv-SE/firefox/addon/restclient/> Hämtad 2018-10-16
- [3] Wikipedia, "Uniform Resource Identifier" https://sv.wikipedia.org/wiki/Uniform_Resource_Identifier Publicerad 2018-10-16. Hämtad 2018-10-18
- [4] REST API Tutorial, "Statelessness" <https://restfulapi.net/statelessness/> Hämtad 2018-10-18
- [5] elkstein, "Learn REST: A Tutorial" <http://rest.elkstein.org/> Hämtad 2018-10-18
- [6] javatpoint, "What is Web Service" <https://www.javatpoint.com/what-is-web-service> Hämtad 2018-10-18
- [7] JournalDev, "Web Services Interview Questions – SOAP, RESTful" <https://www.journaldev.com/9193/web-services-interview-questions-soap-restful/#what-is-web-service> Hämtad 2018-10-18
- [8] intertech, "Introduction to Git Concepts" <https://www.intertech.com/Blog/introduction-to-git-concepts/> Publicerad 2012-12-02. Hämtad 2018-10-18
- [9] Linnéuniversitetet, "Kom igång med GitHub" <http://coursepress.lnu.se/info/manual/kom-igang-med-github/> Publicerad 2015-08-19. Hämtad 2018-10-18
- [10] Creative Bloq, "What is Sass?" <https://www.creativebloq.com/web-design/what-is-sass-111517618> Publicerad 2018-04-30. Hämtad 2018-10-18
- [11] php, "\$_SERVER" <https://secure.php.net/manual/en/reserved.variables.server.php> Hämtad 2018-10-20
- [12] php, "file_get_contents" <https://secure.php.net/manual/en/function.file-get-contents.php> Hämtad 2018-10-21

- [13] php, "php:/" <https://secure.php.net/manual/en/wrappers.php.php> Hämtad 2018-10-21
- [14] php, "json_decode" <https://secure.php.net/manual/en/function.json-decode.php> Hämtad 2018-10-21
- [15] php, "mysqli_set_charset" <https://secure.php.net/manual/en/mysqli.set-charset.php> Hämtad 2018-10-21
- [16] Wikipedia, "Query string" https://en.wikipedia.org/wiki/Query_string Publicerad 2018-10-10. Hämtad 2018-10-21
- [17] Jenkov, "Web Service Message Formats" <http://tutorials.jenkov.com/web-services/message-formats.html> Publicerad 2014-05-23. Hämtad 2018-10-22
- [18] nodejitsu, "what is the file 'package.json'?" <https://docs.nodejitsu.com/articles/getting-started/npm/what-is-the-file-package-json/> Publicerad 2011-08-26. Hämtad 2018-10-23
- [19] statista, "Global market share held by the leading web browser versions as of August 2018" <https://www.statista.com/statistics/268299/most-popular-internet-browsers/> Hämtad 2018-10-29
- [20] W3C, "CSS Validation Service" <https://jigsaw.w3.org/css-validator/> Hämtad 2018-10-29
- [21] W3C, "Nu Html Checker" <https://validator.w3.org/nu/> Hämtad 2018-10-29
- [22] WikiPedia, "TypeScript" <https://en.wikipedia.org/wiki/TypeScript> Publicerad 2018-10-17. Hämtad 2018-11-04
- [23] TypeScript, "JavaScript that scales" <https://www.typescriptlang.org/> Hämtad 2018-11-04