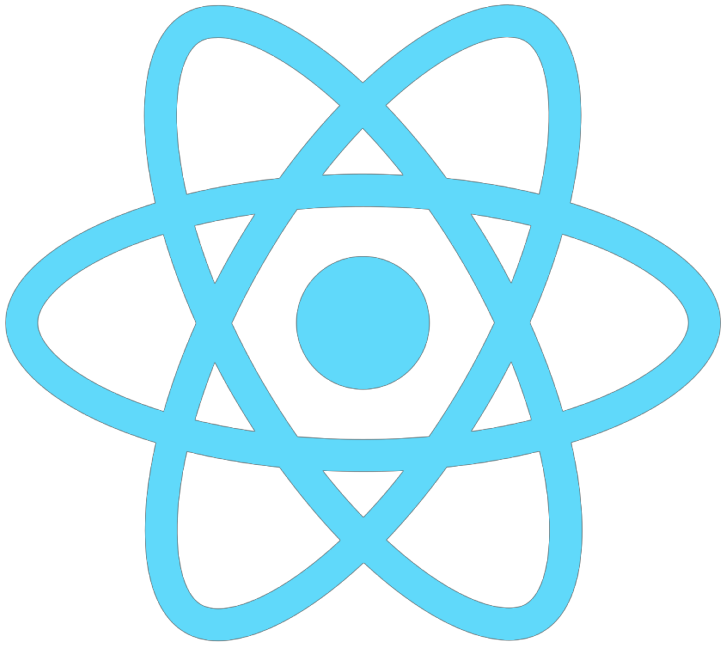
CHALMERS



Utveckling av en skalbar webbapplikation i JavaScript och ReactJs

Uppbyggande av komponenter för en webbapplikation i JavaScript och ReactJs

Examensarbete inom Data- och Informationsteknik

JOHANNES EDENHOLM

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2018

EXAMENSARBETE

JOHANNES EDENHOLM

Version : 2018-06-30

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2018

Uppbyggande av komponenter för en webbapplikation i JavaScript och ReactJs

JOHANNES EDENHOLM

Examinator: Peter Lundin

Omslag: En bild på ReactJs logga från https://commons.wikimedia.org/wiki/File:React-icon.svg där senaste access var 2018-06-02.

Institutionen för Data- och Informationsteknik Chalmers Tekniska Högskola / Göteborgs Universitet 412 96 Göteborg
Telefon: 031-772 1000

Institutionen för Data- och Informationsteknik Göteborg 2018

Sammanfattning

IGDB är ett företag med avsikt att samla all viktig data om spel på ett och samma ställe. Kodbasen för den nuvarande hemsidan är stökig, gammal och restriktiv. Utvecklarna har svårt att skapa ny funktionalitet och hantera den äldre. Rapporten beskriver ett arbetet med syftet att skapa en ny kodbas i JavaScript med hjälp av biblioteket, ReactJs. Arbetet avgränsas till front-end utveckling då allt annat omhändertas med hjälp av IGDB's egna API. Det här arbetet täcker utvecklingen av komponenter med hjälp av React som bibliotek och material-UI som grund för användargränssnittet. Arbetet täcker metoden och genomförandet där det beskrivs hur arbetet har gått tillväga samt vad som har använts och hur det har använts. Rapporten täcker även en utvärdering av slutprodukten samt en diskussion om huruvida andra alternativ kan ha varit möjliga för att uppnå samma resultat.

Nyckelord: ReactJs, JavaScript, Webbapplikation, Komponent, ESLint, Scrum, Enzyme, Material-UI

Abstract

IGDB is a company intending to collect all essential data about games in the same place. The codebase for the current website is messy, old and restrictive. Developers have difficulty creating new functionality and managing the old. This report describes a work with the purpose of creating a new code base in JavaScript using the library, ReactJs. The work is limited to front-end development as everything else is dealt with using IGDB's own API. This project covers the implementation of components with the help of React as library and material-UI as basis for the user interface. The work also covers the method and implementation describing how the work has been carried, what has been used and how it has been used. The report also covers an evaluation of the final product and a discussion of whether other options may have been possible to achieve the same result.

Förord

Examensarbetet har gjorts vid institutionen för data- och informationsteknik på Chalmers Tekniska Högskola i Göteborg, för högskoleingenjörer 2018.

Jag vill tacka alla på IGDB för all hjälp och stöd jag fått vid utförandet av examensarbetet under mina veckor där. Jag vill även tacka Uno Holmer för all handledning under perioden.

Förkortningslista

API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
HTML	*Hypertext Markup Language*
IGDB	*Internet Game Database*
JS	*JavaScript*
Npm	*Node Package Manager*
Prop/props	*Property/Properties*
UI	*User Interface*
DoD	*Definition of Done*
App/appar	*Applikation/Applikationer*
REST	*Representational State Transfer*

Ordlista

AngularJs	*ett ramverk för utveckling av dynamiska appar.*
Boolean	*ett värde som antingen är sant eller falskt.*
Back-end	*utveckling som håller sig till det förser front-end med data och uppgifter.*
Enzyme	*en testmiljö för ReactJs .*
ESLint	*håller reda på och begränsar kodskrivandet på ett sätt så att det håller vissa fördefinierade standarder för radavstånd.*
Front-end	*utveckling som endast håller sig till det grafiska.*
HTML	*ett programspråk för att programmera hemsidor.*
Instansiering	*användning och upp-startande av en klass.*
JavaScript	*ett programspråk som används för att konstruera webbapplikationer.*
Komponent	*en JavaScript klass som ger ett specifikt utseende och funktion i hemsidan. Allt som en användaren kan interagera med är en komponent.*
Material- UI	*en mängd av förprogrammerade komponenter i ReactJs.*
Microsoft VSC	*Microsoft Visual Studio Code, programmeringsmiljö.*
Npm	*en pakethanterare för JavaScript.*
Open source	*mjukvara där ursprungskoden är tillgänglig för allmänheten för att användas fritt.*
Pluralsight	*en hemsida med diverse kurser.*
ReactJs/React	*ett JavaScript bibliotek för att bygga användargränssnitt.*
Rendera	*beräkningen som ett program gör för att visa en bild.*
Ruby on rails	*ett ramverk som utnyttjar MVC arkitekturen och tillhandahåller standardstrukturer för databas, webbtjänster och webbsidor.*
Scrum	*ett arbetsmetod för agil utveckling.*

Innehållsförteckning

1. Inledning 1
   1.1. Bakgrund 1
   1.2. Mål 1
   1.3. Frågeställning 1
   1.4. Avgränsningar 1
2. Teknisk bakgrund 2
   2.1. Scrum 2
   2.2. MVC 2
   2.3. Github 3
   2.4. ReactJs 3
   2.5. Enzyme och Jest 5
   2.6. NPM 5
   2.7. Microsoft Visual Studio Code 6
   2.8. ESLint 6
   2.9. HTML & CSS 6
   2.10. Pluralsight och Codesandbox 6
3. Arbetsmetod 7
   3.1. Arbetsplats 7
   3.2. Val av teknik 7
4. Genomförande 8
   4.1. Föregående system 8
   4.2. Föregående kod 8
   4.3. Planering 9
   4.4. Arbetsgång 9
5. Resultat 14
6. Diskussion 16
7. Slutsats 18

Referenser 19

1. Inledning

IGDB är ett företag med avsikt att samla information om nysläppta och gamla spel på ett och samma ställe, vilket kan vara allt från recensioner till vad spelet handlar om, när det släpptes, genre, tillgängliga plattformar samt mycket mer. Deras nuvarande hemsida behöver rekonstrueras för att göra det enklare for befintliga och nya utvecklare att komma in i koden men även bibehålla nuvarande prestanda for slutanvändaren.

1.1.Bakgrund

Den befintliga kodbasen är rörig, gammal och restriktiv. Den tekniska skulden har blivit ett stort problem och utvecklarna har svårt att skapa ny funktionalitet samt upprätthålla hemsidan. En justering är nödvändig för utveckling och framtida arbete.

1.2.Mål

Projektet avser att resultera i en solid grund for att kunna implementera ny kodbas samt en dokumenterad verktygsuppsättning för att underlätta underhåll, funktionsskapande och lättare åtkomlighet för befintliga och nya utvecklare. Den bör också leverera en snabb och trevlig upplevelse till slutanvändaren.

1.3.Frågeställning

Hur kan man bygga upp generella komponenter med tekniker för att konstruera en snabb, smidig och enkel webbapplikation.

1.4.Avgränsningar

Projektet kommer att begränsas till utveckling i JavaScript med ReactJs som bibliotek, implementering av komponenter och skriva tester med hjälp av Enzyme. Programmeringen kommer begränsas till front-end-utveckling då databasen, autentisering och all annan server-logik redan är omhändertaget på grund av IGDB’s API, vilket är den tjänst som allt kretsar kring.

2. Teknisk bakgrund

Mer ingående beskrivningar av viktiga delmoment och tekniker för projektet. Avsnitten nedan behandlar Scrum, MVC, Github, ReactJs, Enzyme och Jest, NPM, Microsoft Visual Studio Code, ESLint, HTML & CSS samt Pluralsight och Codesandbox.

2.1.Scrum

Scrum är ett agilt tillvägagångssätt för hantering av mjukvaruprojekt/produkter som har blivit extremt populärt bland företag. En huvuddel är att man delar in veckor i så kallade sprintar, där en sprint är en tidsperiod av uppsatta mål från en produktlogg som skall utföras innan sprintens slut. Kraven för en produkt anges i produktloggen och det är produktloggen som man arbetar med att uppfylla under ett förbestämt antal sprintars tid. I produktloggen finns en lista av uppgifter som utvecklarna vill lägga till. Det som skall finnas i produktloggen kan specificeras av utvecklarna eller kommas överens om mellan en kund och utvecklarna tillsammans. Idén bakom en avklarad sprint är att vid varje avslut skall man kunna ha en produkt med ’’färdig’’ funktionalitet att visa upp[1][2].

Dagliga möten kombinerades in och diverse större sprintplaneringsmöten samt avslutningsmöten (sprintretrospective) för när en sprint är klar. Under sprintplaneringsmötet bestämmer gruppen vilka uppgifter som skall utföras under den kommande sprinten och man kan också anta poäng för hur svår en uppgift(user story) är. En user story är en del av hela produktloggen och är oftast en färdig funktionalitet, när klar. För dessa så finns också en Definition of Done (DoD), vilket är en mening som beskriver när user storyn är helt klar.

Under en sprintretrospective går man igenom och visar resultat, om vad som har uppnåtts under sprinten. Man går också igenom hur man uppfattat att det har gått och vad som hade kunnat utförts bättre.

I ett Scrum-lag finnas olika roller. Produktägaren, Scrum-master och Scrum-gruppen. Produktägaren har ansvaret för att hantera och ha äganderätt över produktloggen. Under utvecklingsstadiet har alla i gruppen rätt till att lägga in något i produktloggen men produktägaren har rätt att bestämma vad som skall prioriteras. Det skall också finnas en Scrum-mästare, som tar hand om gruppens planering och ser till att alla är på rätt spår samt att Scrum implementeras på rätt sätt. Slutligen, Scrum-gruppen, här sitter oftast 5–7 utvecklare i en gemensam miljö som tacklar olika uppgifterna i produktloggen[2].

2.2.MVC

MVC är ett designmönster som definierar hur ett program eller applikation skall kodas. M står för Model, och bör innehålla ett objekt som håller data, den kan också innehålla logik som uppdaterar C, Controller. Controller kontrollerar dataflödet till M objektet och uppdaterar View. View bör innehålla allt som visas grafiskt för användaren, från modellen[3].

2.3.Github

Github är en plattform för utveckling, av vanligen programkod. Github tillåter användare att arbeta lokalt på sin egen dator, och sedan ladda upp till Github, i en webbaserad samlingsplats för sitt projekt. Här kan alla som arbetar på ett och samma projekt se vad olika utvecklare arbetat med, och vilka ändringar som gjorts. Vilket tillåter utvecklare att arbeta med olika funktionalitet parallellt med varandra och är mycket smidigt att använda tillsammans med Scrum.

2.4.ReactJs

ReactJs är ett open-source JavaScript bibliotek, konstruerat av Jordan Walke, en mjukvaruingenjör hos Facebook. ReactJs nyttjas för att hantera vy lagret framför allt för 'single-page applications' och tillåter utvecklare att skapa återanvändningsbara UI-komponenter[4].

Med ReactJs kan utvecklare producera storskaliga webbapplikationer som kan modifiera sin data utan att ladda om hemsidan. Fokuset för ReactJs ligger vid att vara snabbt, skalbart och enkelt. React interagerar endast med användargränssnittet i applikationen och kan jämföras med view i MVC-modellen[4].

Istället för vanlig JavaScript mallhantering använder ReactJs JSX. JSX gör det realiserbart för utvecklare att skriva i HTML i JavaScript filen och är genomförbart på grund av att HTML syntaxen översätts till JS anrop i ReactJs ramverket[4].

I ReactJs bestäms en uppsättning av statiska värden till komponenten när de renderas, på så sätt som HTML taggar fungerar. Komponenten kan inte ändra något av dessa värden men kan använda sig av dem för att reagera på olika sätt. De kallas kort för ”props” som står för properties och kan vara mycket praktiskt för att göra en komponent mer generisk för iterativ användning i en webbapplikation[4].

Nedanstående definieras i renderingsklassen för `Button`,

```
<Button labelName='My Button' chosenColor='red' > </Button>
```

Nedan kallas ”propsen” `chosenColor` och `labelName` i komponenten vid användning,

```
<Button color={this.props.chosenColor}>
{this.props.labelName}</Button>
```

Resultatet blir en röd knapp med fördefinierade props,

Figur 1 Röd knapp

Varje komponent har även ett tillstånd. Här definieras de variabler som komponenten kan ha tillgång till och ändra. För att bruka dessa variabler i klassen kan man skriva `this.state.X`, där `X` är variabeln man vill inspektera, som har definierats vid rendering . Ett exempel på vad som kan finnas i `state` är `inputValue`, för en textruta. Vilket är något som komponenten kan behöva ha tillgång till för en komponent som till exempel tillåter användaren att skriva in text, och sedan få ut texten i realtid. Nedan kommer ett exempel för det,

```
constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.state = { inputValue: "" };
}
```

I konstruktorn definieras `state` och `inputValue` sätts till en tom sträng, för att ha tillgång till `state` i andra metoder behöver man binda metoden till `this`, och eftersom `handleChange` ändrar `state`, behöver den bindas,

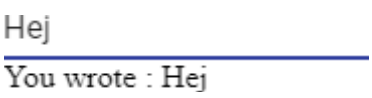
```
handleChange(event) {
    this.setState({
        inputValue: event.target.value
    });
}
```

`handleChange` är vad som hanterar förändringen i textboxen, och ändrar `inputValue` i `state`,

```
render() {
    return (
        <div>
            <TextField onChange={this.handleChange}
                       hintText="Hint Text"/>
            <div>
                You wrote: {this.state.inputValue}
            </div>
        </div>
    );
}
```

Alla klasser som man vill visa i ett användargränssnitt behöver ha en renderingsmetod, `render`, som returnerar komponentens vy i form av HTML. För att kunna använda metoden `handleChange` måste man skriva `this.handleChange` för att visa att metoden finns i klassen.

Sedan har man tillgång till tillstånden, via raden `this.state.inputValue`. Resultatet blir som i bilden under när man skriver något i textrutan och det uppdateras utan att användaren tvingas ladda om hemsidan.



Figur 2 Exempel på hur state ändras med användarens värde.

2.5.Enzyme och Jest

Färdig kod behöver testas mot användningsfall så allt fungerar som det skall vid lansering. För detta så användes Enzyme med Jest. Jest är skapat av Facebook och är ett test-ramverk för att testa JavaScript och React kod[5]. Enzyme är ett verktyg för att sedan köra tester. Testerna körs i terminalen med hjälp av npm.

Med hjälp av Jest kan man skriva tester som simulerar knapptryck, förändringar i input data samt i tillståndet. Man kan dessutom testa så att data man definierar med hjälp av props inte förändras på vägen vid användning i komponenten[5].

2.6.NPM

NPM används dels som en pakethanterare men även för att starta och automatiskt upprätthålla hemsidan temporärt, på en port på datorns lokala nätverk. Npm kompilerar även och uppdaterar automatiskt hemsidan vid sparande av ny kod. För att starta hemsidan så skrivs *”npm start”* i terminalen. Även för att hämta tredje parts paket/komponenter så används NPM till att installera dessa. Det görs genom att skriva *”npm install X”* där *X* är namnet på det paket/komponent paket man vill installera. Sedan har man tillgång till dem genom att importera dem i JSX filerna på följande sätt *”import X from Y”*. *X* i exemplet är den variabel man döper det man importerat till för användning och *Y* är hur man hittar det i projektet. Ett exempel på ovanstående kommer nedan,

```
import Button from 'material-ui/Button';

<Button color=’red' labelName='My Button' >
```

Vilket resulterar i en knapp från material-ui och hade inte vart möjligt utan importen, se figur 1 från avsnitt 2.4.

2.7.Microsoft Visual Studio Code

Microsoft Visual Studio Code 2017 används som utvecklingsmiljö eftersom det har bra stöd för ReactJs och JavaScript. Med Microsoft VSC kan man även ladda ned paket, för att underlätta kodningen samt bibehålla kodkvaliteten, tex ESLint.

2.8.ESLint

ESLint är ett open source projekt skapat av Nicholas C. Zakas. Målet är att skapa ett plugin som kontrollerar koden och håller koll så att särskilda standarder används. Med ESLint kan utvecklaren även skapa egna regler som skall följas vid kodning[6].

2.9.HTML & CSS

HTML bestämmer strukturen för webbsidan och CSS bestämmer hur HTML elementen beskrivs på webbsidan [7], [8]. I projektet används HTML och CSS i JS koden, i JSX filerna för att visa användargränssnittet för användaren av slutprodukten.

2.10.Pluralsight och Codesandbox

Pluralsight är en hemsida med kurser inom många områden. I projektet användes deras ReactJs och JavaScript kurs för att komma igång med ReactJs och JS. Codesandbox är en hemsida som tillåter användaren att implementera ReactJs komponenter utan att själv behöva något mer än en webbläsare. Codesandbox användes för projektet tillsammans med Pluralsight för att genomföra de olika uppgifterna som gavs vid den webbaserade kursen.

3. Arbetsmetod

Nedan beskrivs hur arbetet har lagts upp. Kapitlet behandlar arbetsplats och val av tekniker.

3.1.Arbetsplats

Allt arbete skedde hos IGDB i deras kontor på Lindholmen där alla de ingående personerna i företaget satt i ett rum tillsammans. Vilket var fördelaktigt då alla alltid vad nära till hands om hjälp behövdes. Det skapade genast en gemensam miljö. Nackdelarna var att det kunde bli väldigt varmt då många datorer placerades i samma rum.

3.2.Val av teknik

Val av de tekniker som använts var redan vid åtagande av projektet bestämt av beställaren. Av denna anledningen användes ReactJs som bibliotek för användargränssnittet och deras funktioner genom projektet. JavaScript blev det språket som användes, eftersom den tidigare front-end utvecklaren redan hade mycket erfarenhet inom området. För informationssökande användes sökmotorn Google och som webbläsare Google Chrome. Sedan eftersom man i teamet ville använda samma IDE, så blev valet här Microsoft VSC. För att skriva och köra tester användes Enzyme med Jest. Det som kontrollerades var att diverse props fungerade, att tillståndet kunde ändras och simulering av användarinput. Som hårdvaruplattform för utveckling av mjukvaran har det använts en Macbook Pro. För att avancera som grupp användes Scrum som arbetsmetod.

4. Genomförande

Kapitlet beskriver mer ingående på tillvägagångssätten i arbetet. Innehåll i kapitlet är föregående system, föregående kod, planering och arbetsgång.

4.1.Föregående system

Även fast det inte är något fel med MVC i sig, så användes ett gammaldags utförande av arkitekturen. Genom att rendera front-enden med data som härrör direkt från webbplatsens back-end. Den nya front-enden kommer istället ta enkla datastrukturer och montera dem på klienten för att skapa användargränssnittet. Det kommer göra att klienten tar åt sig den större delen av belastningen och serven kommer ha mycket mer prestanda till att hantera trafik. Det betyder att när hemsidan är under mycket användning är det klienten som kommer få göra arbetet och inte serven som ligger på IGDB's håll.

Även om styrenheterna och databasen kommer förbli ungefär densamma, kommer back-end ansvaret vara att leverera data i ett simpelt format till front-enden, tillskillnad från nu, när back-enden är ansvarig för att rendera varenda begäran från klienten. Som service skulle den nya front-enden utnyttja REST standarden till att kommunicera med back-enden, vilket kommer ge stor förenlighet med koden.

Databasen, som förblir densamma, har kontakt med IGDB's API. APIet begär data med hjälp av diverse kommandon, från databasen. Kommandona kan filtrera hur mycket data som kommer tillbaka och även filtrera om man endast söker viss information. Databasen levererar tillbaka med den sökta datan i form av en lista med all information som hittats och sökts efter i databasen om det sökta spelet. För webbsidan i detta fallet hämtas en länk som dirigerar om användaren till IGDB's hemsida för det spelet som söktes efter.

Den föregående strukturen fungerade vid mindre trafik av hemsidan men nu när den blivit mer populär är det inte längre hållbart för dem att serven gör allt arbete för alla besökare. Det skapar onödig belastning på serven om den ska hantera renderingen av klientens komponenter och även all trafik på webbsidan.

4.2.Föregående kod

Den ursprungliga front-end kodbasen har ingen komponentarkitektur. Varje externt bibliotek fanns i det globala namnutrymmet. På grund av den tekniska skulden för originalkoden var många bibliotek föråldrade och kostnaden för att behålla eller uppdatera koden är för hög för IGDB's nuvarande gruppstorlek. Vilket gör att en uppdatering krävs för att kunna bibehålla en likvärdigt användargränssnitt men för en lägre kostnad. Komponent-tekniken som react erbjuder i sitt sätt att skapa single-page applications är precis det som behövs för att hantera hemsidan på ett skickligare sätt. Eftersom komponenterna i sig inte är beroende av varandra för att fungera och koden som renderar användargränssnittet för användarna inte ligger i ett kluster av HTML kod utan snarare var för sig.

Gällande komponenten som beskrivs i 4.3, se ComboBox, passade inte deras ursprungskod deras ändamål. De hade använt sig av ett redan fördefinierat bibliotek och kom därav inte med de alternativ som de ville ha och snarare med andra användarsätt som var för avancerade och onödiga för deras fall.

Biblioteket blev väldigt stort och tog därav mycket prestanda vid renderingen. Ett av alternativen som biblioteket hade, och som de ville använda sig av gav en oattraktiv implementation och det medförde att gränssnittet ibland buggade ur. Därav krävdes det något nytt, mer skräddarsytt för deras mål, och mindre tungt vid import.

4.3.Planering

De första två veckorna gick åt till att skaffa kännedom om JavaScript samt ReactJs som introducerades med hjälp av en webbaserad nybörjarkurs i ReactJs på pluralsight[9]. Kursen gick igenom grundläggande hur man arbetade med tillstånd, props och vad en komponent var. Därefter började det med lite enklare övningar med ReactJs och JavaScript med hjälp av Codeseandbox.io[10]. Efteråt så installerades Microsoft VSC och npm på datorn för att starta med projektet.

4.4.Arbetsgång

Uppbyggnaden av webbsidan skulle ske i form av komponenter, där en komponent är en JavaScript klass med ett eget fördefinierat användargränssnitt och funktion. Avsikten med att koda olika komponenter är för att göra den slutgiltiga webbapplikationskoden så simpel som möjligt genom att enkelt addera en komponent vid behov. På grund av att komponenter skall kunna återanvändas är det därför nödvändigt att de är generiska. Därför kan man bestämma olika variabler att skicka in till komponenten vid rendering, till exempel bestämma vad en textruta skall ha för informationstext till användaren.

För att få tillgång till dessa variabler i JS kan man skriva `this.props`.Variablerna får man tillgång till genom att definierar dem från utsidan av klassen vid instansiering. Variablerna kan vara allt från booleska värden, till variabler som håller data, strängar och heltal.

Komponenter gör också att det blir enklare för nya utvecklare att sätta sig in i koden vid vidareutveckling, om något behöver ändras eller läggas till, kan det enkelt göras och koden ser inte lika stökig ut.

ComboBox
Första komponenten var en autosuggestion box(döptes senare till ComboBox), där tanken var att om något skrivs in får man upp förslag på diverse ord som användaren kan tänkas mena, tex namn på olika spel. Dessa ord avgränsas till ord som utvecklaren fördefinierade i props vid instansiering av komponenten och de kan antingen komma in som en array av objekt eller hämtas från en URL.

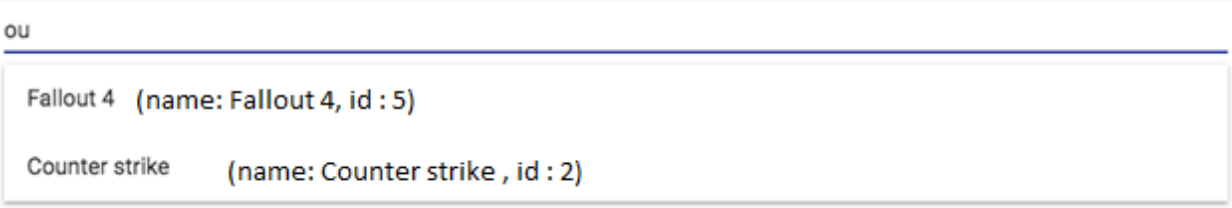
Det komponenten behöver för att kunna visa upp data i användargränssnittet var ett id samt ett namn. Id var viktigt då det kunde finnas spel med likadant namn, och då behövdes de åtskiljas med ett id.

Nedan är ett exempel av en array med olika nycklar och värden, samt hur den används vid instansiering av ComboBox.

En array `test` skapas med nycklarna `name` och `id`. Dessa tilldelas olika värden,

```
const test = [
  {name: 'Fallout 4', id: 5},
  {name: 'Counter strike', id: 2}
];
```

ComboBox instansernas och `data` sätts till `test`,
`<ComboBox data={test} />`
Alternativen finns nu vid användning av ComboBox. Användaren har skrivit in *”ou”* som input och får således upp båda alternativen som förslag, då *”ou”* finns i båda namnen.



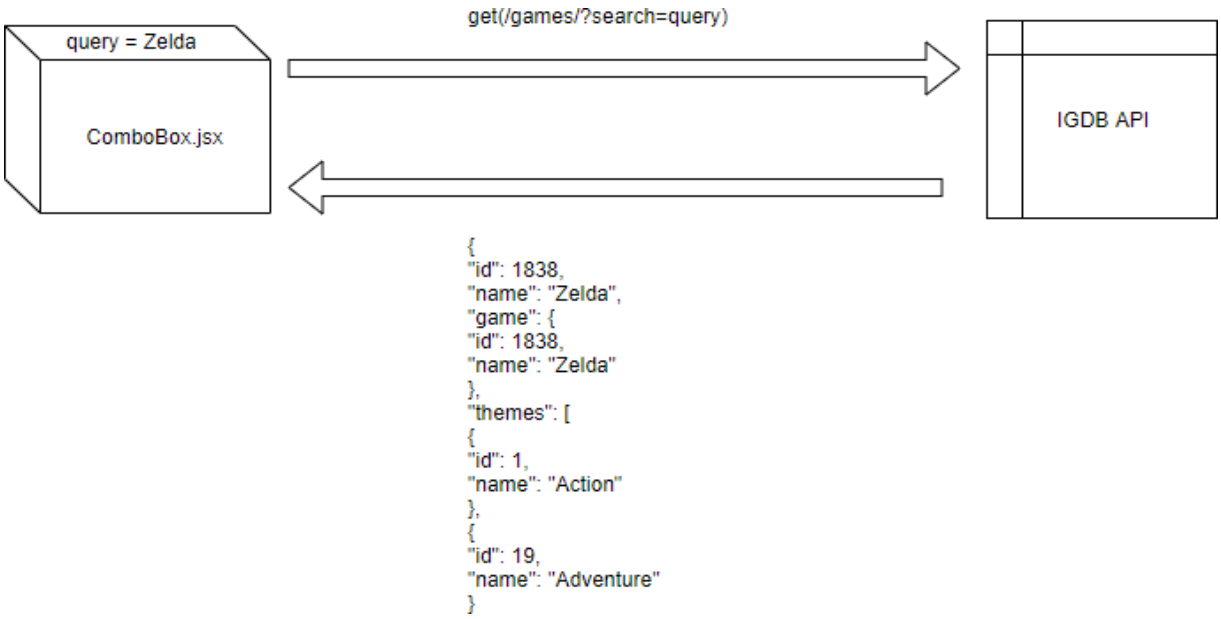
Figur 4 ComboBox vid sökning

Mycket av koden för att implementera ComboBox fanns redan på Material-UI[11], dock behövde mycket modifieras för att anpassas till de fördefinierade användningsfallen. Koden behövde även förändras för att bli mer generisk. Mer alternativ behövdes, såsom att kunna välja lokal data samt data från en URL.

Datahandler
Variabelnamnen i komponentens array som visar upp data döptes till `id` och `name`, men man kunde inte garantera att det namngavs så för lokal data eller en URL. Därför skapades metoden `dataHandler`, för att hantera just dessa fall. Metoden omvandlar data från utomstående källor, identifierar vilket som är namnet samt id och placerar dem i komponentens array.

Multiple
Multiple är ett alternativ för ComboBox som tillåter användare att välja fler sökalternativ än ett. Om man tex, vill söka efter fler spel och det demonstreras mer tydligt i resultatdelen.

Kommunikation med API
ComboBox kommunicerar med IGDB’s egna API, i bilden nedan begär den data från ett visst spel, och får tillbaka all nödvändig lagrad information för efterfrågat spel,



Figur 5 Hur IGDB's API kommunicerar med ComboBox

Tester

För att skriva tester gjordes en ny klass för komponenten som skulle testas. I klassen monterades komponenten och tester skrevs för olika användningsfall. Fördelarna med att använda sig av Enzyme med Jest var att testerna är snabba och nästan ingen förkonfiguration alls krävdes.

Nedan kommer två exempel på hur dessa skrevs.

Exempel på test för att se om props fungerade,

```
it('use of multiple choice option prop', () =>
    { const wrapper = mount(<ComboBox multiple />);
    expect(wrapper.props().multiple).to.equal(true);
});
```

Exempel på test för att se om tillståndet kunde ändras,

```
it('Check state change', () => {
    const wrapper = mount(<ComboBox data={test} />);

expect(wrapper.update().state('inpuValue')).to.not.equal('Johannes')
    wrapper.setState({ inputValue: 'Johannes' });
    expect(wrapper.update().state('inputValue')).to.equal('Johannes');
});
```

ESLint
Redan vid början av projektet hade den andre front-end utvecklaren installerat och konfigurerat ESLint för att anpassa sig till våra behov. ESLint var mycket användbart eftersom det var som ett säkerhetsbälte. Vid varje uppdatering som gjordes efter skrivande av kod lades koden korrekt och passande efter den tidigare koden som skrivits, i form av radavstånd till var och en.

Användargränssnitt
När ett användargränssnitt implementerades så gjordes det, om det endast var lite modifieringar som krävdes, i `style`. `Style` är en av de taggar som kan jämföras med dem från HTML och ett exempel på ovanstående kommer under,

```
<Button
style={{ backgroundColor: "red", padding: 50 }}>Press me
</Button>
```

Vilket resulterar i nedanstående bild och kan jämföras med figur 1 i avsnitt 2.4 där `padding` alternativet inte används.

Figur 3 Exempel på hur CSS ändrar utseende

Om lite större konfigurationer krävdes eller om det inte önskades bland HTML koden kan det definieras i en egen variabel. Vilket var smidigt och användes ofta eftersom variabeln kunde innehålla alla olika objekt som skulle ändras på i hela komponenten. Ett exempel på ovanstående visas nedan,

```
const styles = theme => ({
    root: {
      flexGrow: 1,
      height: 250,
    },
    container:
      { flexGrow:
      1,
      position: 'relative',
    },
    paper: {
      position: 'absolute',
      zIndex: 1,
      marginTop: theme.spacing.unit,
      left: 0,
      right: 0,
    },
    inputRoot:
      { flexWrap:
      'wrap',
    },
  });
```

För att senare använda tex den fördefinierade `paper` konfigurationen användes `className`, och här hänvisades det till variabeln som innehåller CSS för paper,

```
<div className={this.props.classes.paper} >
```

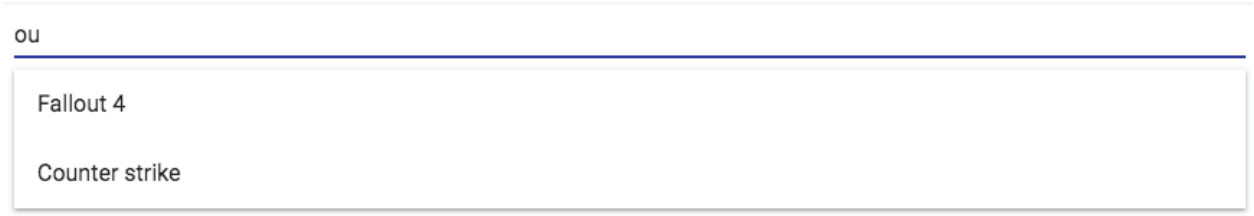
5. Resultat

Resultatet blev en funktionell ComboBox med mycket valmöjligheter för att göra den så generell som möjligt. Komponenten blev väldigt lyckat eftersom den blev helt skräddarsydd för IGDB's behov men ändå så pass generisk att den även skulle kunna användas för många andra fall och webbsidor. Nedan skildras ett exempel på användargränssnittet samt hur en sökning kan gå till med *Multiple* som alternativ.

Exemplet under visar hur ComboBox såg ut i sitt slutskede. I *"Search for Something"* placeras texten när man skriver på tangentbordet.

Figur 6 Instans av ComboBox innan sökning

Här har användaren skrivit *"ou"* och två alternativ på spel som innehåller *"ou"* i namnet visas.



Figur 7 Användaren har sökt efter "ou" och fått upp alternativ

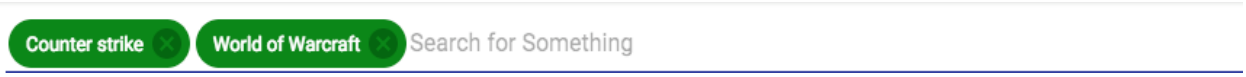
Användaren har sedan klickat med musen på alternativ 2, Counter strike och objektet har nu placerats i komponentens textruta.

Figur 8 Användaren har valt alternativ 2, Counter strike

Nedan gör användaren ytterligare en sökning, med *”a”* och får upp tre alternativ på spel som innehåller *”a”* i sitt namn.

Figur 9 Använder söker igen efter något som börjar på "a"

Vid bilden nedan så valde användaren alternativ 1, World of Warcraft.



Figur 10 Användaren har nu valt World of Warcraft

Efter användaren har valt alternativ så kan användaren trycka på en sökknapp, alternativt trycka enter på tangentbordet med idén att få fram dessa söktermer i form av mer ingående sidor för eventuella spel. Sökningen sker med hjälp av IGDB’s API som kontaktar databasen och hämtar länkar till spelets egna dedikerade hemsidor på IGDB.

6. Diskussion

I skrivande stund har enbart en komponent skapats och genomgått tester, medan tanken från början var att hinna med mer än så.

Arbetet under dessa veckor med JavaScript och ReactJs visar att det finns många lösningar för utveckling av webbapplikationer då man kan använda npm med JavaScript. Vilket gör det ännu viktigare att faktiskt veta vad man vill eftersom om man hittar något från internet man vill implementera kan man enkelt göra så genom npm. Om man använder sig av någon annans lösning kan den komma med mycket mer funktionalitet än vad man vill ha, vilket kan vara påfrestande för applikationen vid slutanvändning. Det gör det viktigt att identifiera vad man inte behöver.

Erfarenhet
Eftersom tidigare kunskap inom dessa olika verktyg (JavaScript, & ReactJs) var minimal var arbetet till en början ansträngande. Fullt med problemlösning och mycket hjälp av internet för att hitta lösningar som för en lite mer erfaren JavaScript utvecklare förmodligen inte hade varit några problem. Det tog mer tid än väntat att komma in i JavaScript och till en följd av det, även ReactJs. Från en tidigare kurs ’Webbapplikationer’ som har lästs hade jag redan en del kunskap gällande HTML och CSS men kodningen i den kursen hanterades av Java. Jämfört med i detta fall, JavaScript vilket jag trots allt föredrog. Det beror främst på att JavaScript känns mycket friare och tillåter användaren att programmera på ett sådant sätt att webbutveckling helt enkelt passar in perfekt med sättet som JavaScript är skapat. Jag är säker på att om jag hade haft liknande kunskap i JavaScript som i Java hade mycket av utvecklingen gått snabbt.

ReactJs
Även fast det till en början var svårt att komma igång, tror jag inte det låg på react utan på JavaScript. React var aldrig ett hinder vid utvecklingen av komponenter eftersom det har varit ganska tydligt från början hur det fungerade.

Vid projektets start beslutades det att ReactJs skulle nyttjas, men det finns enligt mig åtminstone ett till alternativ. AngularJS, Angular var på marknaden redan 4 år innan React och är skapat av Google. Det som verkar vara den största skillnaden mellan ReactJs och AngularJs är att Angular använder sig av den fulla MVC arkitekturen, medan React endast använder sig av V, view delen i MVC[12]. På grund av Angular’s arkitektur brukar det prestera långsammare än ReactJs då det måste hantera mer ting.

Även om Angular påstås vara enkelt att lära sig så kommer det förmodligen att ta mer tid än React. Eftersom man behöver lära sig mycket mer nytt, medan react endast introducerar tre nya begrepp. Komponenter, states, och properties.

I AngularJS skapar man applikationer med hjälp av direktiv, vilket betyder att man blandar JS, CSS och HTML kod. I React används endast JS, som använder ett syntax som tillåter användaren att skriva HTML med JS i samma fil, som där kalas JSX. För någon som är ny med react kan det verka avskräckande, det känns dock som det smidigare alternativet[12].

Kodkvalitet
Eftersom ett av syftena med den slutgiltiga produkten var att göra det enklare för nya utvecklare att komma in i koden, så att vidareutveckla eller modifiering kunde ske var det viktigt att kodkvalitén höll en hög standard. Med hjälp av ESlint genomdrevs en från början bra kvalitet på koden.

Tester
Syntaxen för att skriva testerna var mycket enkel, och kom naturligt efter några minuters användning. Dessvärre så uppdaterades material-UI framåt slutet. Resultatet blev att inte Enzyme kunde användas längre och man fick sitta och köra tester för hand.

Scrum
Eftersom jag skulle få tillgång till hjälp från en front-end utvecklare på företaget och webbsidans renovering var i deras backlogg så implementerades jag i företagets egna Scrum-grupp. Man kan i efterhand tänka att för mycket tid spenderades på att sitta vid dem veckovis måndagsmötena eftersom vi var en större grupp och endast det som den andre front-end utvecklaren höll på med ibland var relevant för projektet. Jag ansåg dock inte det som något större problem, då det kändes givande att få sitta med och lära sig mer om vad de andra höll på med. Det gav även tillgång till mycket hjälp från andra sidor eftersom alla visste vad jag arbetade på den veckan.

Framtida arbete
Ett framtida arbete hade varit att avsluta det jag påbörjat, fortsätta med utvecklingen av komponenter för att avsluta implementationen av den nya hemsidan. Ett annat intressant framtida arbete skulle kunnat vara att jämföra den äldre hemsidan med den nya. Kodkvalitén, användarvänlighet, användarupplevelse och snabbhet.

Eftersom ReactJs användes för projektet, kunde ett annat projekt varit att välja ett annat typ av bibliotek eller ramverk, som AngularJS, och se om man kan replikera resultatet.

7. Slutsats

Om någon med mer erfarenhet inom JS hade tagit sig an projektet hade förmodligen mer blivit avklarat. Då cirka två veckor i början ägnades åt att utbilda sig så mycket som möjligt inom JS och react. Då det visade sig att react inte alls var speciellt besvärligt att lära sig hade den tiden kunnat läggas till annat om redan kunskap inom JS hade funnits.

Oavsett så lyckades en komponent skapas, ComboBox. Komponenten var oerhört lyckad och har mycket funktionalitet som gör den både generell och precis för olika ändamål.

Även fast Enzyme med Jest slutade fungera på grund av en uppdatering med material-UI så han mycket av användarfallen testas innan.

Användningen av ESLint var som tänkt, det gick bra och mycket av kodstandarderna kunde enkelt hållas eftersom det inte gick att kompilera koden om någon av dessa bröts. Vilket forcerade utvecklaren att aktivt följa alla rekommendationer och meddelanden som ESLint föreslog. Dock kunde det vara aningen irriterade vid testning av komponenten då ESLint inte tillät användaren att spara variabler i koden som inte användes, vid det tillfället.

Hur som helst så visade sig det vara relativt enkelt att utveckla generella komponenter med ReactJs för att bygga upp en hemsida och om en redan erfaren JavaScript användaren hade valt att göra projektet hade denne förmodligen kommit mycket längre än vad författaren av rapporten gjorde.

Etik och miljö
Eftersom IGDB hänvisar och ger information om spel automatiskt är det mycket möjligt att det ibland kan bli oklart på vilka typer av spel som faktiskt sprids. Ett ansvar ligger då på företaget att möjligtvis sätta åldersgränser för att komma in på olika sidor, som extra skydd för att hindra till exempel underåriga för att se material som inte är lämpligt för dem. Vad gäller miljö så är elförbrukningen en faktor som bör uppmärksammas. Förbrukningen består grovt sett av klienternas egna datorer samt det nätverk som oftast är involverat. Underlättande och uppmuntran till spel ökar troligen förbrukningen hos alla ingående komponenter. Den aktuella hemsidan har dock mycket små möjligheter att påverka detta. Generellt bidrager den allmänna teknikutvecklingen av datorerna till minskad energiförbrukning.

Referenser

1. *Scrum*
   https://www.mountaingoatsoftware.com/agile/scrum
   (acc 2018-05-25)

2. *Dan Radigan.*
   Scrum. https://www.atlassian.com/
   agile/scrum (acc 2018-05-25)

3. *MVC Framework - Introduction https://www.tutorialspoint.com/*
   mvc_framework/mvc_framework_introduc tion.htm
   (acc 2018-05-25)

4. *Nitin Pandit*
   What Is ReactJS and Why Should We Use It?
   https://www.c-sharpcorner.com/article/what-and-why-reactjs/
   (acc 2018-05-25)

5. *Berry de Witte*
   Unit testing your react application with Jest and Enzyme
   https://medium.com/wehkamp-techblog/unit-testing-your-react-application-with-jest-and-enzyme-81c5545cee45
   (acc 2018-05-25)

6. *About*
   https://eslint.org/docs/about/
   (acc 2018-05-25)

7. *CSS Introduction https://*
   www.w3schools.com/css/css_intro.asp (acc
   2018-05-25)

8. *HTML Introduction https://*
   www.w3schools.com/html/html_intro.asp (acc
   2018-05-25)

9. *https://www.pluralsight.com/*
   (acc 2018-05-25)

10. *https://codesandbox.io/*
    (acc 2018-05-25)

11. *https://material-ui-next.com/*
(acc 2018-05-25)

12. *Catalin Cimpanu*
Basic Differences Between AngularJS and React https://
news.softpedia.com/news/Basic-Differences-Between-AngularJS- and-
React-484781.shtml
(acc 2018-05-25)