Examensarbete

HÖGSKOLAN I SKÖVDE

JÄMFÖRELSE AV SVARSTID VID FILTRERING MELLAN VUE.JS OCH REACT

COMPARISION OF RESPONSE TIME IN FILTERING BETWEEN VUE.JS AND REACT

Examensarbete inom huvudområdet Informationsteknologi
Grundnivå 30 Högskolepoäng
Vårtermin 2020

Jacob Svensson (a17jacsv)

Handledare: Marcus Brohede
Examinator: Henrik Gustavsson

Sammanfattning

Arbetet jämför JavaScript-ramverken Vue.js och React huruvida vilket av ramverken som presterar lägst svarstider i ett experiment. Bakgrunden till arbetet är att undersöka visualisering av data och i form av en interaktion som är filtrering. Arbetet är inriktat på parkeringsplatser som blir svårare att finna och att filtrering används för att finna lediga parkeringsplatser. Experimentet består av en webbapplikation skapad i en implementation av respektive ramverk som innehåller en lista med parkeringsplatser som går att filtrera. Mätningar genomfördes automatiserat genom ett egenskrivet skript med användning av Tampermonkey för att få fram ett resultat. Resultatet visar att det fanns en signifikant skillnad mellan ramverkens resultat från experimentet till fördel för Vue.js. Det går att följa arbetet steg för steg hur det genomfördes med mål att studien ska vara replikerbar.

Nyckelord: [Vue.js, React, Svarstid, Interaktivitet, Filtrering]

Innehållsförteckning

1 Introduktion

Webben och webbapplikationer i synnerhet är en viktig del i människors vardag och används bland annat för att presentera och visa information. Webbapplikationers användare blir ofta irriterade när svarstid överstiger två sekunder och därav är det av stor vikt att ha låga svarstider vid användning av en webbapplikation (Butkiewicz et al, 2011). Interaktiv visualisering av data förekommer ofta på internet och det handlar om att användaren själv får välja vad som skall visas genom en interaktion (Godfrey et al, 2016). Arbetet har valt att använda sig av JavaScript-ramverken Vue.js och React. Ett ramverk inom JavaScript är som ett skal för en webbapplikation som kan använda olika funktionaliteter som redan är implementerade, vilket sparar tid för utvecklaren (Perez et al, 2008). Arbetet undersöker vilket av Vue.js och React som presterar lägst svarstider vid en interaktion i form av filtrering. Filtreringen sker på en större och mindre mängd data för att se om det skiljer sig åt. Den data som används i arbetet är parkeringsplatser som filtreras på om den är ledig eller upptagen. Om bilförare kan se om det finns tillgängliga parkeringsplatser sparar tid genom att se vilken specifik parkeringsplats som är tillgänglig istället för att åka runt och leta.

Problemet är att det är svårt att få en exakt bild av vad som skiljer sig i form av svarstid mellan Vue.js och React vid filtrering av parkeringsplatser. För att ta reda på ett resultat genomfördes ett teknikorienterat experiment av en webbapplikation skapad i två implementationer med båda ramverken Vue.js och React. Målet med implementationernas var så likt som möjligt logiskt, kodmässigt och visuellt för att på rättvist sätt kunna jämföra ramverken och dess resultat. Kodstrukturen skiljde sig något men funktionaliteten var sig lik mellan implementationerna. Webbapplikationen innehåller en interaktiv lista där det går att filtrera lediga parkeringsplatser av det totala antalet parkeringsplatser. Statisk data används för att mäta enbart ramverket och inte några serveranrop, uppkoppling till nät eller annan yttre påverkan som kan resultera i störningar. Data som används i arbetet representerar ett totalt antal parkeringsplatser som varierar slumpvist mellan 3150-3500 varav 50-200 är antalet lediga parkeringsplatser. Varje parkeringsplats har egenskaper som definierar varje enskild parkeringsplats vilket lagras i en JSON-fil.

Utöver skapande av webbapplikationen krävdes en metod att genomföra mätningar. Det utfördes genom ett automatiserat skript som skrevs samt exekverades genom Google Chrome-tillägget Tampermonkey.com (2020) för att få fram arbetets resultat. Skriptet mäter från att en av knapparna i webbapplikationen klickas, till att sluta när allt är laddat och klart för rendering. Båda implementationerna använder sig av exakt samma skript för att förutsättningarna ska vara lika.

För att ge arbetet mer styrka har en litteraturstudie genomförts som undersökt några av de källor som använts. Alla källor som har använts har varit från böcker och avhandlingar till officiell dokumentation från Vue.js och React. Hela arbetes progression och genomförande är dokumenterad från planering till färdig implementation i arbetet. Samtlig utvecklingsprocess finns tillgängligt och går att följa på ett publikt GitHub-respository (Svensson, 2020) som gör replikering av arbetet möjligt.

2 Bakgrund

I kapitlet presenteras delar i arbetet som förklaras vilket kan resultera i ökad förståelse för arbetet och dess syfte. Kapitlet berör interaktivitet som används vid filtrering av parkeringsplatser som är den data som används webbapplikationen. Webbapplikationen är skapad med ramverk genom två implementationer med Vue.js och React som beskrivs och sedan jämförs med varandra i kapitlet.

2.1 Interaktivitet

Interaktiv visualisering av data på internet har ökat de senaste åren (Godfrey et al, 2016). Det handlar om att användaren själv får välja vad som skall visas genom en interaktion på en applikation. Interaktion inom applikationer har alltid varit en viktig del trots att datamängden inte alltid har varit så stor som den är nu för tiden. Genom att låta användaren justera parametrar kan applikationen visualisera mer meningsfulla representationer av data.

Användaren till applikationer kan idag själv anpassa upplägget för att få det att uppfylla dennes behov, vilket ökar systemets effektivitet och relevans (Godfrey et al, 2016). Ett vanligt förekommande exempel är filtrering som används för att sortera ut data i form av attribut som användaren eftersöker (Godfrey et al, 2016). Interaktivitet är tvåvägskommunikation mellan användare och visualiseringsverktyg som kan ske på flera olika sätt. Ett exempel kan vara att användaren ändrar på en variabel genom en interaktion som ändrar vad som visas på skärmen (Liere et al, 2009).

2.2 Parkeringsplatser

Flera städer runt om i världen har börjat att övervaka parkeringsområden för att uppskatta lediga platser för att hjälpa bilförare som letar efter parkering att spara tid. För att se om parkeringarna är lediga eller upptagna sätts sensorer upp för att kontrollera det (Ionita et al, 2018). Genom att som förare kunna se om det finns tillgängliga parkeringsplatser och slippa åka runt och leta. När föraren ska planera sin avfärd är det viktigt att filtrering av samtliga parkeringsplatser görs med en kort svarstid. Enligt Zhou et al (2016) ses svarstid ur en användarens synvinkel som väntan och avgör till stor del användarens helhetsintryck av systemets slutprodukt. Därav är svarstidens längd en viktig aspekt för användarens helhetsupplevelse av en webbapplikation.

2.3 Ramverk i JavaScript

Ett ramverk är en webbtjänstbaserad arkitektur som inte behöver installeras på någon hårdvara. Det är som ett skal för en webbapplikation som kan använda olika funktionaliteter som redan är implementerade, vilket sparar tid för utvecklaren (Perez et al, 2008). Utbudet av ramverk är stort och det kan vara svårt att välja ett som passar den applikation som man tänkt bygga. Detta arbete använder sig av två ramverk inom programmeringsspråket JavaScript i form av React och Vue.js. Liknande ramverk enligt Nikulchev et al (2018) är exempelvis AngularJS (AngularJS.org, 2020), Backbone.js (Backbonejs.org, 2020), Ember.js (Emberjs.com, 2020), samt Polymer (Polymer-project.org, 2020). Dessa har valts bort i arbetet för att fokusera på Vue.js (Vuejs.org, 2020) och React (Reacjs.org, 2020).

2.3.1 Vue.js

Vue.js kallas för ett progressivt JavaScript-ramverk av dess skapare. Det innebär att det går att bygga din webbapplikation med minimal ansträngning eftersom kärnbiblioteket i Vue.js endast fokuserar på applikationens vy. Biblioteket kan enkelt användas och integrera tillsammans med

andra bibliotek (Nelson, 2018). Ramverket är enkelt att lära sig för användare med tidigare erfarenhet av JavaScript. Vue.js har tagit del av lösningar från JavaScript-ramverken Ember.js, React och AngularJS när det har utvecklats (Nikulchev et al, 2018). Ramverket är komponentbaserat och använder sig av en virtuell DOM som avgör om ett element eller objekt behövs laddas om eller uppdateras. Det görs för att förhindra onödig omladdning av hela webbplatsen som resulterar i en snabbare inläsning vilket kräver mindre prestanda av din enhet (Nelson, 2018).

Ett exempel på att skapa en lista genom att använda sig av Vue.js visas nedan i figur 1. Li-taggen i HTML-koden använder sig av ”v-for” som används för att skriva ut en lista av ”items” från en array. I JavaScript koden skapas en variabel som tilldelas namnet: ”exempel” som innehåller en array där ”message” innehåller 5 enskilda siffror som skrivs ut som visas i output. Se figur 1 nedan.

HTML:

```
<ul id="exempel">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```

JavaScript:

```
var exempel = new Vue({
  el: '#exempel',
  data: {
    items: [
      { message: '1' },
      { message: '2' },
      { message: '3' },
      { message: '4' },
      { message: '5' }
    ]
  }
})
```

Output:

- 1
- 2
- 3
- 4
- 5

Figur 1: Skapa en oordnad lista med Vue.js

2.3.2 React

React är ett JavaScript ramverk som skapades ursprungligen av ingenjörer på Facebook för att lösa utmaningar i samband med utveckling av komplexa användargränssnitt med data som förändras över tid. React använder sig av komponenter som beskriver ett gränssnitt och är enkelt att manipulera eftersom det går kombinera flera komponenter. Det gör att man kan anpassa så som det önskas. Ramverket använder sig av en virtuell DOM som avgör om komponenten behövs laddas om eller inte för att förhindra onödig omladdning av hela webbplatsen vilket också resulterar i en snabbare inläsning (Gackenheimer, 2015). React är inte kompatibelt med

andra bibliotek som ändrar eller modifierar DOM vilket kan anses som en nackdel. React har en hög prestanda men en större tröskel att lära sig eftersom det har ett mer ovanligt sätt att skriva kod på jämfört med andra ramverk inom JavaScript som är mer lika varandra (Nikulchev et al, 2018).

Ett exempel på att skapa en lista genom att använda sig av React visas nedan i figur 2. I HTML-filen så skapas en div-tagg med ett id som ersätts med en komponent som skapas i JavaScript-koden. Först skapas en array som innehåller 5 siffror. Därefter utförs en loop med användning av funktionen ”map()” som skapar en li-tagg av varje ”item” som finns i arrayen. Därefter renderas en lista med alla listobjekt i den tomma div-tagg som skapades i HTML-koden, som visas i output. Se figur 2 nedan.

HTML:

```
<div id="exempel">
    <!—Detta element innehåll kommer bli ersatt av komponenten -->
</div>
```

JavaScript:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);

ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('exempel')
);
```

Output:

- 1
- 2
- 3
- 4
- 5

Figur 2: Skapa en oordnad lista med React

2.3.3 Jämförelse av Vue.js och React

Nikulchev et al, (2018), anser att båda ramverken Vue.js och React har hög prestanda och innehar en omfattande dokumentation. Båda ramverken är komponentbaserade och använder sig av en virtuell DOM som innebär att de undviker onödiga omladdningar av hela DOM när enbart berörda delar av webbapplikationen kräver omladdning. Ramverken skiljer sig i struktur av kod där Vue.js anses vara enklare och React svårare att lära sig eftersom det är mer komplext. React använder sig av ett mer ovanligt sätt att skriva kod jämfört med andra ramverk. Vue.js har stöd för både envägs och tvåvägsdatabindning medan React har enbart för envägsdatabindning. Kodande med React gör det svårare att skapa en snabb prototyp av en lösning än exempelvis av Vue.js (Nikulchev et al, 2018). React är skapat och anpassat efter storskaliga användargränssnitt istället för mindre projekt (Gackenheimer, 2015). Det går att skriva mindre applikationer men är inte lika smidigt som vid användning av Vue.js och tillåter utvecklaren att börja smått och sedan öka efter applikationens behov (Nelson, 2018).

Det skiljer sig mellan koden i figur 1 och figur 2 vid skapande av en oordnad lista. Figur 1 använder sig av HTML-kod i form av ul/li-tagg medan figur 2 enbart har en tom div-tagg som sedan ersätts av dess komponent. Både koden i figur 1 och 2 skapar en array som innehåller samma data som används för att skriva ut siffrorna. Kodens uppbyggnad och struktur skiljer sig ifrån varandra och används på olika sätt för att få fram samma output.

3 Problemformulering

JavaScript-ramverk används för att spara tid för utvecklaren genom att använda färdigimplementerade funktioner jämfört med att behöva utveckla själv från grunden (Perez et al, 2008). Problemet är att det finns många stora ramverk som används, men det är svårt att veta vad som egentligen skiljer ramverken åt. För utvecklare är det inte enbart viktigt att välja JavaScript-ramverk som tjänar deras nuvarande projekt, utan även ger kod och prestanda av hög kvalitet. Det finns även fler faktorer som spelar in i form av exempelvis ramverkets underhållbarhet och giltighet (Gizas et al, 2012).

I arbetet har valt att jämföra ramverken Vue.js och React för att se vilket av ramverken som presterar lägst svarstid vid filtrering av lokalt lagrad data i form av parkeringsplatser. Anledning till varför valet föll på Vue.js och React var för att Nikulchev et al, (2018) anser att båda ramverken har flera likheter och har en hög prestanda med omfattande och genomgående dokumentation. Båda ramverken använder sig av Virtuell DOM som avgör om en komponent eller ett objekt behöver uppdateras vilket är en viktig egenskap vid filtrering av en lista eftersom inte hela webbplatsen ska behöva påverkas. React (Gackenheimer, 2015) anses vara skapat och anpassat för storskaliga användargränssnitt och Vue.js (Nelson, 2018) för mindre eller mellanstora projekt.

Problemet är att det är svårt att få en exakt bild av vad som skiljer sig i form av svarstid vid filtrering av data. Systemets svarstid är tidsfördröjningen mellan en användares initiering av ett kommando på en dator och systemets slutförande som leder till visning av resultat. Ur användarens synvinkel ses svarstid som väntan och avgör till stor del vad användarens helhetsintryck är av systemets prestanda. Därav är svarstidens längd en viktig aspekt för användarens helhetsupplevelse av applikation (Zhou et al, 2016). Om svarstider i en applikation kan hållas nere kan det göra ett stort avtryck på användaren som ökar chanserna till en mer frekvent användning av applikation.

Ett annat problem är att applikationen ska använda sig av tillräckligt med data för att resultatet ska vara trovärdigt. Otillräcklig data kan leda till felaktiga resultat, och allt för stor data kan leda till slöseri av resurser (Wang et al, 2017). Arbete har valt att använda sig data som representerar ett totalt antal parkeringsplatser som varierar slumpvist mellan 3150-3500 varav 50-200 är antalet lediga parkeringsplatser. Varje parkeringsplats har egenskaper som definierar varje enskild parkeringsplats. Egenskaperna består av ett id, vilken sektion parkeringsplatsen tillhör samt vilken sorts parkering det är. Antalet parkeringsplatser anses i arbetet vara en större respektive mindre mängd data när mätningar skall utföras. Anledning till att varför antalet parkeringsplatser varierar är för att säkerställa att inget specifikt värde sparas lokalt i webbläsaren.

Arbetet har som mål att svara på följande fråga:

- Vilken implementation av React/Vue.js har snabbast svarstid vid filtrering av större respektive mindre mängd data av parkeringsplatser?

Genom att få svar på denna fråga går det att se vad som skiljer sig åt i form av svarstid. Det ger en inblick som visar vilken implementation av Vue.js och React som presterar snabbast vid filtrering av data i webbapplikationen.

Hypotes: Arbetets hypotes är att det inte är någon statistisk signifikant skillnad mellan implementationerna av Vue.js och React när det gäller svarstider vid filtrering av data

innehållandes parkeringsplatser. Det kommer heller inte vara någon statistisk signifikant skillnad mellan implementationerna av Vue.js och React vid filtrering av en större respektive mindre mängd data innehållandes parkeringsplatser.

4 Metod

Den vetenskapliga metod som används i arbetet är ett experiment i teknisk form. Ett experiment är en grundläggande metod som används för att ha kontroll över en situation med mål att kunna manipulera olika beteenden specifikt, direkt och systematiskt. Ett experiment kan ha som fokus på teknik som har hög kontroll på, eftersom man vet vad dess beteende är. Alla människor har inte samma beteende vid olika tillfällen vilket resulterar till att det är mindre kontroll på sitt experiment. Genom utförande av analyser av experiment resulterar det i kunskap som ger förmågan att förändra och förfina modeller över tid. De fall när ett experiment är lämpligast att använda är när en studie är kvantitativ och kräver kontroll över dess situation vilket ger en möjlighet att kunna manipulera faktorer på ett systematiskt sätt. (Wohlin et al, 2012). Detta arbete är kvantitativt och är anledningen till varför ett experiment används. Experimentet är teknikorienterat eftersom inga människor är delaktiga i arbetet. Enligt Wohlin et al. (2012) så är det en risk att använda sig av experiment. Ett experiment kräver mycket tid till planering, noggrannhet och utförande vilket kan vara kostsamt. Det är heller inte någon garanti att de resultat avspeglar verkligheten eftersom miljön som experimentet utförs i inte är verklig även om den utförs i en miljö som liknar verkligheten.

Experimentet kommer att bestå av en webbapplikation som implementeras i två utvalda ramverk i form av React och Vue.js. Webbapplikationen är uppbyggd på samma vis med samma funktionalitet och vy med mål att ha likadana förutsättningar i två olika implementationer. Webbapplikationen innehåller en interaktiv lista där det går att filtrera lediga parkeringsplatser av det totala antalet parkeringsplatser. Statisk data används för att mäta enbart ramverket och inte några serveranrop, uppkoppling till nät eller annan yttre påverkan som kan resultera i störningar.

Detta arbete har valt att använda sig data som representerar ett totalt antal parkeringsplatser som varierar slumpvist mellan 3150-3500 varav 50-200 är antalet lediga parkeringsplatser. Varje parkeringsplats har egenskaper som definierar varje enskild parkeringsplats vilket lagras i en JSON-fil. JSON är ett datautbytesformat som många system använder för att kommunicera data (Bassett, 2015). Egenskaperna är ett id, vilken sektion parkeringsplatsen tillhör samt vilken sorts parkering det är. Antalet parkeringsplatser varierar för att implementationerna ska säkerställa att inget specifikt värde sparas lokalt i webbläsaren. Alla mätningar sker på samma hårdvara med mål att utföras med exakt samma förutsättningar.

4.1 Alternativa metoder

Två andra metoder inom empirisk forskning som inte använts i arbetet är användarstudier och fallstudier. En användarstudie används oftast när man redan har utnyttjat en teknik eller verktyg. Användarstudier används mer för att överse antalet som använder den tekniska artefakten än att se över själva artefakten i sin helhet (Wohlin et al, 2012). Syftet med arbetet är att undersöka artefakten och inte dess användning eller popularitet vilket förklarar varför användarstudier inte kommer att utföras i arbetet.

En fallstudie används för att undersöka ett eller flera fall i en verklig miljö under en begränsad tid. En nackdel med en fallstudie jämfört med ett teknikorienterat experiment är att i miljön inte har lika mycket kontroll. Det kan leda till att resultatet är svårt att generalisera, tolka och att förstå. (Wohlin et al, 2012). För att på ett rättvist sätt kunna jämföra Vue.js och React i arbetet krävs kontroll över hårdvara och programkod vilket gör det svårare att applicera i en fallstudie jämfört med i ett teknikorienterat experiment.

4.2 Etiska aspekter

Ur etisk synpunkt är det viktigt att detta arbete är replikerbart. Orsaken till det är att ett arbete som går att replikera är lättare att lita på eftersom vem som helst kan göra om arbetet för att kontrollera resultatet. Det är av stor vikt att arbetet innehåller dokumentation genomgående för att en replikering av arbetet ska ge ungefär likadant resultat som det ordinarie resultatet (Wohlin et al, 2012). För att arbetet ska kunna replikeras krävs även att JavaScript-ramverken som används använder sig av öppen källkod. Vue.js (Vuejs.org, 2020) och React (Reactjs.org, 2020) används i detta arbete och innehar öppen källkod vilket gör det möjligt att replikera arbetet.

Utöver möjligheten till replikering av arbetet så är antalet mätningar av stor vikt för att förstå analysens validitet. Genomförs för få mätningar ska man vara försiktig med att analysera och diskutera olika mönster i resultatet. Det krävs en stor mängd mätningar för att generera godtagbara värden för att kunna analysera och använda som resultat i sitt arbete (Wohlin et al, 2012). Experimentet har utförts utifrån ett totalt antal parkeringar som varierar slumpvist mellan 3150-3500 och antalet lediga parkeringsplatser mellan 50–200 för att kunna se att data liknar varandra genomgående under mätprocessen och inte skiljer sig mycket åt. Det resulterar i mer korrekta värden som resulterar i ett bättre resultat.

En annan aspekt är att det inte går att säkerställa att de två implementationerna av applikationen är exakt logiskt lika. Arbetets mål är att utveckla koden i applikationerna så lika som möjligt eftersom det inte går att garantera att det resultatet är exakt samma sak. Genom att använda sig av identiska fasta världen på båda implementationerna gör att de innehåller samma data och funktionalitet. Arbetet anser att det bör göra att applikationens logik är den samma på båda implementationerna, men det är inget som detta arbete kan garantera.

5 Genomförande

Arbetets genomförande har fokus på en lösning kring de mål som satts upp samt hur lösningen har tagits fram.

5.1 Litteraturstudie

Detta arbete baseras till viss del på tidigare forskning och kommer i kapitlet att nämna några relevanta litteraturer och artiklar som har tagits i beaktning vid verkställandet av detta arbete. Denna litteraturstudie ger inspiration samt kunskap som leder till att implementationen av experimentet genomförs.

Wohlin, Runesson, Höst, Ohlsson, Regnell och Wesslén (2012) presenterar för läsaren ett experiment genom ett processperspektiv. Fokus i litteraturen ligger på de steg som behövs för att utföra ett experiment. Boken är uppdelad i tre delar där den första delen ger en bakgrund av teorier och metoder som används i experiment. Den andra delen handlar om de fem stegen inom experiment: omfattning, planering, genomförande, analys och resultatpresentation. Den tredje och sista delen innehåller två exempel som förklaras i detalj. Litteraturen ger läsaren en större uppfattning inom empiriska studier särskilt inom experiment men också för fallstudier och användarstudier. Litteraturen presenterade ett teknikorienterat experiment som blev ett naturligt val för detta arbete.

Godfrey, Gryz och Lasek (2016) beskriver i sin artikel att visualisering ger ett kraftfullt sätt för dataanalys, men för att vara praktiska måste visuella analysverktyg vara smidiga och flexibla vid visualisering. Med tanke på den ständigt ökande datamängden i databaser är det svårt att få svarstider på en integrering att inte överstiga flera sekunder. Artikeln går igenom vad en interaktion är och hur en användare idag kan anpassa upplägget efter dess behov, vilket ökar ett systems effektivitet och relevans. Artikeln gav arbetet en insikt till relevansen kring svarstid och hur stor vikt det är att hålla nere svarstider i en applikation.

Inspirationen till arbetets tekniska artefakt kommer från olika källor som lästs och gett idéer som underlättat arbetet. Bland litteraturen som rör React.js finns flera guider i bokform där detta arbete har valt att använda sig av boken: *Introduction To React,* skriven av Gackenheimer (2015). Boken introducerar läsaren genom att definiera ramverket. Den argumenterar varför React bör användas och vilka problem som löses på ett smidigt sätt. Litteraturen går igenom i detalj hur kod skrivs och beskriver både enkla och komplicerade kodexempel med avsikt att få en bredare förståelse.

I litteraturen som rör Vue.js är utbudet lite mindre med tanke på yngre ålder, men en lämplig guide som presenterar grunderna inom ramverket finns i form av: *Getting to know Vue.js,* skriven av Nelson (2018). Boken introducerar läsaren genom att beskriva ramverket. Den presenterar ramverkets styrkor och värdet av att använda Vue.js. Litteraturen presenterar hur kod skrivs och beskriver komplicerade och enklare kodexempel för att få en insikt och en uppfattning hur Vue.js fungerar i praktiken.

Genom att läsa litteratur och artiklar ger det många fördelar men en stor nackdel är att de är skrivna ur en författares tolkning av problem och hur saker skall genomföras sett ur deras perspektiv. I litteratur som funnits i några år finns en stor chans att de inte är aktuella vilket ger risker eftersom funktionalitet uppdateras och förändras över tid. För att att komma närmre dagsaktuell tillämpning av ramverken kompletteras litteraturen med officiell information från

Vue.js (Vuejs.org, 2020) och React's (Reactjs.org, 2020) webbplatser. Hemsidorna innehåller dokumentation som förklarar hur ramverken fungerar med kodexempel sett från dess skapares synvinkel. De förklarar vad som sker i detalj vilket leder till en bredare förståelse.

5.2 Hårdvara och mjukvara

Hårdvara: Detta arbete och dess experiment har utförts på en och samma hårdvara för att undvika eventuella skillnader som skulle kunna påverka arbetets resultat negativt. Den hårdvara som samtlig utveckling och mätning har utförts på har varit på en dator med operativsystemet macOS. Nedan kommer exakt information om vilken hårdvara som används och vad dess specifikationer är.

Modell: MacBook Pro (15 tum 2016)

Processor: 2,6 GHz Quad-Core Intel Core i7

Minne: 16 GB 2133 MHz LPDDR3

Startskiva: Macintosh HD

Operativsystem: macOS Catalina version 10.15.4

Grafik: Intel HD Graphics 530 1536 MB

Webbläsare: Arbetet har valt att avgränsa och enbart använda sig av Google Chrome som webbläsare för att arbetets resultat inte ska varieras genom att bruka flera webbläsare.

Kodeditor: Mjukvaran som all utveckling har utförts i har varit med mjukvaran Visual Studio Code.

Utifrån dessa hårdvaror har arbetets resultat tagits fram utan några problem kopplat till själva hårdvaran. Mjukvaror som använts har inte heller medfört några större problem som bromsat arbetet.

5.3 Installation

För att komma igång med implementationerna av Vue.js och React installerades varsin applikation. Arbetet använder sig av Node.js (Node.js, 2020) för att installera de paket som krävs genom kommando i terminalen.

5.3.1 Vue.js

För att installera Vue.js i detta arbete krävs Vue Cli (Vue CLI, 2020). Det används för att kunna skapa Vue.js-projekt och installeras genom att skriva och använda kommando i terminalen. Vue Cli installeras genom att skriva in kommando som visas i figur 3.

```
npm install -g @vue/cli
```

Figur 3: Installering av Vue CLI

För att skapa ett Vue-js projekt skrivs kommandot nedan som visas i figur 4 där ”vueapp” i arbetes fall är namnet på sin applikations mapp.

```
vue create vueapp
```

Figur 4: Skapande av Vue-projekt

När projektet är skapat får användaren ett val om den vill använda sig av standard funktionaliteter eller manuellt valda funktionaliteter. Arbetet valde ”default”. Därefter navigerar in på applikationsmappen som installerats och startar den lokala webbservern genom att skriva in följande kommando i terminalen. Se figur 5.

```
cd vueapp
npm run serve
```

Figur 5: Lokalisera mappen samt start av vue-applikation

5.3.2 React:

För att skapa ett React projekt är det ett kommando mindre än vid installation av Vue.js. Det som krävs är att skriva kommandot nedan som visas i figur 6 där ”react” i detta fallet är namnet på sin applikations mapp.

```
npx create-react-app react
```

Figur 6: Skapande av React-projekt

Därefter navigera in på applikationsmappen som installerats och starta den lokala webbservern genom att skriva in följande kommando i terminalen. Se figur 7.

```
cd react
npm start
```

Figur 7: Lokalisera mappen samt start av react-applikation

5.4 Implementation

För att kunna utföra en pilotstudie skapades två webbapplikationer varav en var skapad med Vue.js och en i React. Webbapplikationerna är tänkt att se exakt likadana ut för att experimentet som skall utföras ska vara så korrekt som möjligt. Det bör inte finnas något i båda webbapplikationerna som den andre applikationen inte har. All kod som används i detta arbete finns tillgängligt i en repository (Svensson, 2020) på GitHub.com (2020) som är uppdaterad kontinuerligt under utvecklingens gång. Där är det möjligt att se exakt de steg som tagits för att utföra detta arbete.
För att lagra all data skapades en JSON-fil som används av båda implementationerna av Vue.js och React. Denna fil har totalt 17501 rader kod som innehåller 3500 objekt för att representera samtliga parkeringsplatser. Filen delar in varje enskilt objekt i ett antal egenskaper, den första är ”id”, som beskriver vilket nummer objektet har. Den andra egenskapen är ”section” som visar vilken sektion objektet tillhör vilka varierar från A-F. Den tredje och sista egenskapen är ”type” som förklarar vilken typ av parkering det är. De typer som finns är standardparkering,

familjeparkering samt handikapparkering. Filen är uppbyggd och innehar samma struktur som visas nedan i figur 8. Figuren visar enbart 3 av 3500 objekt. [1]

```
[{
        "id": 1,
        "section": "F",
        "type": "Standardparkering"
    },
    {
        "id": 2,
        "section": "E",
        "type": "Familjeparkering"
    },
    {
        "id": 3,
        "section": "D",
        "type": "Handikapparkering"
    }
]
```

Figur 8: parkingslots.json

För att båda implementationerna ska ha exakt samma antal parkeringsplatser vid mätning av webbapplikationerna skapades en JSON-fil. Filen skapades genom att fylla en array genom användning av en for-loop som utfördes i 100 000 iterationer av JavaScript-funktionen ”Math.random” som visas i figur 10. Anledning till att spara resultatet av ”Math.random” och inte köra funktionen och slumpa vid varje användning är att båda implementationerna av webbapplikationen ska ha exakt samma antal vid filtrering för att deras förutsättningar ska vara exakt likadana med målet att få ett så korrekt resultat som möjligt.

```
for (let i = 0; i < 100000; i++) {
        randomArr[i] = Math.random();
    }
```

Figur 9: For-loop för skapande av ”randomArr”

Efter en körning av webbapplikationen var slutförd var arrayen ”randomArr” fylld med 100 000 objekt. För att kunna spara arrayen till en JSON-fil krävdes utskrivning av samtliga objekt för att kunna föra över till en extern fil. Genom att använda sig av ”.innerHTML” var det möjligt att skriva ut hela arrayen ”randomArr” till en temporär div-tagg med id ”copyArray” som visas i figur 10.

```
document.getElementById("copyArray").innerHTML = randomArr;
```

Figur 10: Ersätta kod med hjälp av ”.innerHTML”

Därefter exekverades webbapplikationen ytterligare en gång och alla arrayens ”randomArr” objekt kopierades och lades in i den nyskapade filen randomSeed.json. Filens struktur består av resultatet av 100 000 exekveringar av JavaScript-funktionen ”Math.random”. 5 objekt av resultatet visas i figur 11 nedan.[2]

[1] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/44df42b

[2] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/d7df244

```
[0.42145471062942197,
0.40038028154627847,
0.5194761200823663,
0.8354625959830202,
0.9012702366545586]
```

Figur 11: randomSeed.json

De två implementationerna av applikationen innehar exakt samma vy. Den innehåller en rubrik som definierar webbapplikationens namn och två knappar. Första knappen hämtar samtliga parkeringar och den andra knappen hämtar alla parkeringar som är lediga just vi det tillfället. Parkeringsplatserna listas i en lista och visar id, sektion och dess typ. Det presenteras hur många parkeringar det finns genom att skriva ut längden på den array som lagrar alla parkeringar. Se en skärmbild från webbapplikationens design i figur 12 nedan.

Figur 12: Skärmbild från webbapplikationens design

5.4.1 Vue.js

I detta delkapitel kommer implementationen av ramverket Vue.js presenteras och förklaras steg för steg i dess utveckling. [3] Den första filen som skrevs i implementationen av Vue.js var filen ”App.vue”. Filen ”App.vue” är huvudcontainern till applikationen. Är det något som skall finnas och visas i hela applikationen ska det tillämpas i denna fil (Nelson, 2018). Det är applikationens root och är vanligtvis något som definierar den template för hur sin webbapplikation ska vara strukturerad. I figur 13 nedan visas innanför template-taggarna att webbapplikationen har en enkel struktur innehållandes en div-tagg som framställer hela applikationen. Inuti den förekommer den enda komponent som skall köras i experimentet, ”Parkingslots”, som importeras till filen. Webbapplikationens namn definieras som ”app” och att den enbart kommer använda sig av en komponent.

```
<template>
  <div id="app">
    <Parkingslots />
  </div>
</template>

<script>
  import Parkingslots from './components/Parkingslots.vue'

  export default {
    name: 'app',
    components: {
      Parkingslots
    }
  }
</script>
```

Figur 13: App.vue

Den andra filen som skrevs i arbetet var filen ”main.js”. Filen är utgångspunkten som sätter igång applikationen (Nelson, 2018). Den importerar Vue.js i sin helhet samt filen ”App.vue” som beskrevs ovan som startar filen genom ”$mount(’#app’) som visar applikationens innehåll.

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

new Vue({
  render: h => h(App),
}).$mount('#app')
```

Figur 14: main.js

Den tredje filen som skapades var ”index.html” som ger en webbläsare sin startpunkt för att ett element ska kunna ladda in, i detta fall ”app”. Filen innehåller standard HTML-kod som definierar vilket språk som webbapplikationens text kommer att ha samt vilken titel webbapplikationen ska ha. Sedan importeras filen ”main.js” in för att kunna initialisera webbapplikationen. Se figur 15 nedan.

[3] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/428c25d

```
<!DOCTYPE html>
<html lang="sv">
  <head>
    <meta charset="utf-8">
    <title>Vue app</title>
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

Figur 15: index.html

Den fjärde filen som skapades var filen ”Parkingslots.vue” [4]. Filen är arbetets webbapplikations enda komponent. En komponent är ett anpassat element som går att definiera och återanvända i en applikation (Nelson, 2018). Den komponent som skapats innehåller en lista som presenterar samtliga parkeringsplatser i olika former. Det första som utfördes vara att importera den JSON-fil, som tidigare har introducerats, för att ha tillgång. Definierar komponenten med dess namn ”Parkingslots” och att komponenten returnerar en array som tilldelats namnet ”slotList”. Den returnerar ytterligare en array som har namnet ”randomArr” som består av json-filen ”randomSeed.json” och en variabel med namnet ”randIdx” som tilldelats värdet -1. Under ”methods” i figur 16 så deklareras de funktioner som används i implementationen av Vue.js.

Den första funktionen är ”getRand()” som används för att hämta resultat från JavaScript-funktionen ”Math.random” ur arrayen ”randomArr” som består av JSON-filen ”randomSeed.json”. För att funktionen inte ska returnera samma objekt vid varje exekvering så ökar variabeln ”randIdx” med 1. Funktionen returnerar ett objekt ur arrayen ”randomArr” och genom att använda sig av modolus (%) på arrayens längd för att ta fram rätt objekt i ordningen.

Den andra funktionen har namnet ”getParkingslots()”. Den fyller den tomma arrayen ”slotList” med objekt från JSON-filen ”Parkingslots”. Arrayen ”slotList” blandar ordningen av listan genom att använda sig av JavaScript-funktionen ”sort()” som i detta fallet i figur 16 tilldelas 0.5 vilket sorterar samtliga objekt i arrayen helt slumpvist. Anledningen till att blanda ordningen i arrayen ”slotList” är för att undvika att samma värde mäts om och igen vid varje mätning vilket kan resultera i felaktiga resultat. För att fylla arrayen ”slotList” används JavaScript-funktionen ”slice()” som returnerar de valda objekten i en array till ett nytt array-objekt (W3School, 2020). I detta fall som visas i figur 16 hämtas 90% samt slumpvist upp till 10% av den nyss sorterade arrayen ”slotList”. Det blir alltså 3150 objekt som minst och 3500 objekt som mest som tar plats i arrayen ”slotList”. Funktionen skriver även ut en rubrik genom att använda sig av ”.innerHTML” för att skriva ut önskad text.

Den tredje funktionen som heter ”availableParkingslots()” fungerar på ungefär samma sätt som ”getParkingslots()”. Det skiljer sig åt när det kommer till användandet av JavaScript-funktionen ”slice()” som istället tar exakta värden som alltid varierar mellan 50 och 200 i antal lediga parkeringar. Den använder sig alltså inte av procent utan av fasta värden. Se figur 16.

[4] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/d7df244

```
<script>
import parkingslots from '../../public/parkingslots.json'
import randomSeed from '../../public/randomSeed.json'
export default {
  name: "Parkingslots",
  data() {
    return {
      slotList: [],
      randomArr: randomSeed,
      randIdx: -1
    };
  },
  methods: {
    getRand(){
      this.randIdx++;
      console.log(this.randIdx);
      return this.randomArr[this.randIdx%this.randomArr.length];
    },
    getParkingslots() {
      document.getElementById("title").innerHTML = "Totala parkeringar";
      this.slotList = parkingslots.sort(() => .5 - Math.random()).slice(0,
      Math.floor(this.getRand() * (parkingslots.length * .10)) +
      parkingslots.length * .90);
    },
    availableParkingslots() {
      document.getElementById("title").innerHTML = "Lediga parkeringar";
      this.slotList = parkingslots.sort(() => .5 - Math.random()).slice(0,
      Math.floor(this.getRand() * 151) + 50);
    }
  }
};
</script>
```

Figur 16: Parkingslots.vue 1

Listan som listar samtliga parkeringsobjekt skapades genom följande HTML-kod som visas i figur 17 nedan [5]. För att kunna skriva ut alla parkeringsplatser används en for-loop. Genom att använda sig av Vue.js egna variant av for-loop, ”v-for”, är det möjligt att iterera genom samtliga objekt i en array samt visa varje objekt och dess innehåll som finns (Nelson, 2015). Varje enskilt objekt i arrayen ”slotList” tilldelas namnet ”slot” och vid varje egenskap skriva ut en span-tagg med dess id, sektion och typ av parkering. Om samtliga 3500 parkeringsplatser ritades ut skulle det finnas 3500 div-taggar med klassen ”slotItem”. Se figur 17.

[5] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/2e796c8

```
<div class="list">
  <div v-f or="slot in slotList" :key="slot.id" class="slotItem">
    <div class="slot-id">
      <div>
        <span><b>Parkeringsplats: </b>{{slot.id}}</span>
      </div>
      <div>
        <span class="slot-section"><b>Sektion: </b>{{slot.section}}</span>
      </div>
    </div>
    <div class="slot-type">
      <span><b>Typ: </b>{{slot.type}}</span>
    </div>
  </div>
</div>
```

Figur 17: Parkingslots.vue 2

5.4.2 React

I detta delkapitel kommer implementationen av ramverket React presenteras och förklaras steg för steg i dess utveckling.[6] Den första filen som skapades i React var filen ”App.js”. Filen ”App.js” är huvudet till applikationen. Det som skall finnas och visas i applikationen ska tillämpas i denna fil (Gackenheimer, 2015). Det är applikationens root och är vanligtvis något som definierar webbapplikationen och hur den ska vara strukturerad. I figur 18 nedan skapas en klass som föreställer webbapplikationens helhet som ”extends” dess komponenter. Klassen returnerar en div-tagg som innehåller arbetets enda komponent med namnet ”Parkingslot” som importeras till filen genom att använda sig av ”import”. Komponenten som används i experimentet beskrivs längre fram i kapitlet.

```
import React, { Component } from 'react'
import Parkingslot from './Parkingslot'

class App extends Component {
    data() {
        return {
        };
    }
    render() {
        return (
            <div className="App">
                <Parkingslot />
            </div>
        );
    }
}

export default App;
```

Figur 18: App.js

Den andra filen som skrevs i arbetet var filen ”main.js”. ”Main.js” är utgångspunkten som sätter igång applikationen likt implementationen med Vue.js (Nelson, 2018). Den importerar React i sin helhet samt filen ”App.js” som beskrevs ovan som exekverar filen genom

[6] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/abc840a

”ReactDOM.render(’<App/>, document.getElementById(’app’));” som renderar och visar filen ”App” och dess innehåll. Se figur 19.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';

ReactDOM.render( <App/> , document.getElementById('app'));
```

Figur 19: main.js

Den tredje filen som skapades var ”index.html” som ger en webbläsare sin startpunkt för att ett element ska ladda in, i detta fall ”app” likt implementationen av Vue.js. Filen innehåller standard HTML-kod som definierar vilket språk som webbapplikationens text har samt vad titeln ska vara. Stylingen utfördes i denna fil innanför style-taggarna. Sedan importeras filen ”main.js” in för att kunna initialisera webbapplikationen. Se figur 20 nedan.

```
<!DOCTYPE html>
<html lang="sv">
    <head>
        <meta charset="UTF-8">
        <title>React app</title>
        <style>
            /* CSS-kod */
        </style>
    </head>
    <body>
        <div id="app"></div>
    </body>
</html>
```

Figur 20: index.html

Den fjärde fil som skapades var filen ”Parkingslots.js” [7] som är arbetets webbapplikations enda komponent precis som i implementationen av Vue.js. En komponent är ett anpassat element som går att definiera och återanvända i en applikation (Nelson, 2018). Den komponent som skapats innehåller en lista som presenterar samtliga parkeringsplatser i olika former.

Det första som utfördes vara att importera de JSON-filer som innehåller alla parkeringsplatser och resultat från ”Math.random”, ”parkingslots.json” och randomSeed.json för att ha tillgång. Importerar även ”useState” som används för att använda sig av ”hooks”. En ”hook” är en speciell funktion som låter dig "hook into" React-funktioner. Till exempel är ”useState()” en ”hook” som låter dig lägga till ”state” till funktionskomponenter. Tidigare behövdes en klass för att utföra det (Reactjs.org, 2020).

Därefter skapades en array med namnet randomArr som tilldelats värdet av JSON-filen ”randomSeed.json”. Sedan skapades ytterligare en array som tilldelats namnet ”slotList” som använder sig av Hooks som fyller arrayen genom att använda sig av funktionen ”setSlotList()” genom att använda sig av state. Samma sak utfördes när en variabel skapades med namnet randIdx som har funktionen ”setRandIdx()”. Se figur 21.
Den första funktionen är likt implementationen av Vue.js ”getRand()”. Den används för att hämta resultat från JavaScript-funktionen ”Math.random” ur arrayen ”randomArr” som består

[7] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/d7df244

av JSON-filen ”randomSeed.json”. För att funktionen inte ska returnera samma objekt vid varje exekvering så ökas ”randIdx” på med 1. Funktionen returnerar ett objekt ur arrayen ”randomArr” och genom att använda sig av modolus (%), på arrayens längd för att ta fram rätt objekt i ordningen. Se figur 21.

Den andra funktionen har namnet ”getParkingslots()” likt implementationen av Vue.js. Den fyller den tomma arrayen ”slotList” med objekt från JSON-filen ”Parkingslots”. Arrayen ”slotList” blandar ordningen av listan genom att använda sig av JavaScript-funktionen ”sort()” som i detta fallet i figur x tilldelas 0.5 vilket gör att den sorterar samtliga objekt i arrayen helt slumpvist. Anledningen till att blanda ordningen i arrayen ”slotList” är för att undvika att samma värde mäts om och igen vid varje mätning som kan ge felaktiga resultat. Därefter för att fylla arrayen ”slotList” används JavaScript-funktionen ”slice()” som returnerar de valda elementen i en array som ett nytt array-objekt (W3School.com, 2020). I detta fall som visas i figur 21 hämtas 90% samt slumpvist upp till 10% av den nyss sorterade arrayen ”slotList”. Det blir alltså minst 3150 objekt som minst och 3500 objekt som mest som tar plats i arrayen ”slotList”. Funktionen skriver även ut en rubrik genom att använda sig av ”.innerHTML” för att skriva ut önskad text. Se figur 21.

Den tredje och sista funktionen heter ”availableParkingslots()” och fungerar på liknande sätt likt implementationen av Vue.js. Den skiljer sig åt när det kommer till användandet av JavaScript-funktionen ”slice()” som istället tar exakta värden som alltid varierar mellan 50 och 200 antal lediga parkeringar. Den använder sig inte av procent utan av fasta värden. Se figur 21. [8]

```
import React, { Component } from 'react'
import Parkingslots from './data/parkingslots.json'
import RandomSeed from './data/randomSeed.json'
import {useState} from 'react'

    var randomArr = RandomSeed;
    const [randIdx, setRandIdx] = useState(0);
    const [slotList, setSlotList] = useState([]);

    function getRand(){
        setRandIdx(randIdx + 1);
        console.log(randIdx);
        return randomArr[randIdx%randomArr.length];
    }
    function getParkingslots() {
        document.getElementById("title").innerHTML = "Totala parkeringar";
        setSlotList(Parkingslots.sort(() => .5 - Math.random()).slice(0,
        Math.floor(getRand() * (Parkingslots.length * .10))
        + Parkingslots.length * .90));
    }
    function availableParkingslots() {
        document.getElementById("title").innerHTML = "Lediga parkeringar";
        setSlotList(Parkingslots.sort(() => .5 - getRand()).slice(0,
        Math.floor(getRand() * 151) + 50));
    }
```

Figur 21: ParkingslotList.js 1

[8] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/d7df244

Listan som visar samtliga parkeringsobjekt skapades genom följande HTML-kod som visas i figur 22 nedan. För att kunna skriva ut alla parkeringsplatser krävs användning av en for-loop för att genomföra det som önskat. JavaScript-funktionen ”Map ()” loopar igenom hela arrayens objekt en gång för varje element i ordning (W3Schools.com, 2020). ”Map()” användes och varje enskilt objekt i arrayen ”slotList” tilldelas namnet ”slot”. Samtliga objekt och dess egenskaper skrivs ut genom en span-tagg med dess id, sektion och typ av parkering det var. Om samtliga 3500 parkeringsplatser ritades ut skulle det finnas 3500 div-taggar med klassen ”slotItem”. Se figur 22 nedan.

```
return (
            <div className="wrapper">
                <div className="content">
                    <h1>Parkingslots</h1>
                    <button className="button"
                       onClick={getParkingslots}>
                       Visa Alla Parkeringar
                    </button>
                    <button className="button"
                       onClick={availableParkingslots}>
                       Visa Lediga Parkeringar
                    </button>
                    <h3 id="title"></h3>
                    <div className="counter">{slotList.length}</div>
                    <div className = "list">
                        {slotList.map((slot) => {
                            return  <div className="slotItem"
                                        key={slot.id}>
                                        <div className="slot-id">
                                            <span>
                                                <b>Parkeringsplats:</b>
                                                {slot.id}
                                            </span>
                                        </div>
                                        <div className="slot-section">
                                            <span>
                                                <b>Sektion:</b>
                                                {slot.section}
                                            </span>
                                        </div>
                                        <div className="slot-type">
                                            <span>
                                                <b>Typ:</b>
                                                {slot.type}
                                            </span>
                                        </div>
                                    </div>
                        })}
                    </div>
                </div>
            </div>
        );
}

export default ParkingslotList;
```

Figur 22: ParkingslotList.js

5.5 Implementation av skript

Ett skript som ska mäta båda implementationerna av Vue.js och React skapades först genom att skriva det internt i koden men efter problem och mer fundering kom ett bättre alternativ upp.[9] Det var genom att använda sig av ett verktyg som heter Tampermonkey (Tampermonkey.net, 2020). Det är ett tillägg i webbläsaren Google Chrome som i arbetet används för att simulera klick samt utvinna data från artefakten genom ett automatiserat skript. Fokus på arbetet är svarstid och därav är det av stor vikt att tekniken som mäter tider görs av kvalitet. Arbetet har använt sig av ”performance.now()” för att mäta tider. Enligt Google Developers (2019) bör ”performance.now” användas för att få bästa möjliga resultat eftersom den även mäter i mikrosekunder och inte bara millisekunder.

Skriptet mäter från att en av knapparna i webbapplikationen klickas, till att sluta när allt är laddat och klart för rendering. Arbetet har valt att tiden som själva utritningen av webbapplikationens element tar, inte kommer att ingå i mätningarna. Enbart rendering för båda ramverken tar lika lång tid, men tiden före rendering är det som är intressant att se. Anledningen till det är för att jämförelsen mellan de två implementationerna av Vue.js och React ska vara så lika som möjligt och att båda använder sig av exakt samma skript. Förutsättningarna är lika och gör att data som analyseras och jämförs anses av arbetet vara korrekt [10].

Det första som utfördes vid skapande av scriptet var att skriva variabler. Därefter skrevs en array med namnet ”scrapedData” som skall innehålla resultatet från mätningarna. ”Start” och ”end”-variabler skapades samt en om resultatet. En variabel med namnet ”counter” skapades för att på ett enkelt sätt kunna utföra exakt antal mätningar som önskat. Så många mätningar som skall utföras skrivs i variabeln ”numberOfClicks” med ett önskat antal. Sedan definierades de två knappar som används i skriptet, det vill säga ”allaParkeringar” och ”ledigaParkeringar”. Se figur 23.

```
const URL = 'http://localhost:8888/vue-measurement/scraped_receiver-
vue.php';
//const URL = 'http://localhost:8888/react-measurement/scraped_receiver-
react.php';

var scrapedData = [];
var start;
var end;
var result;
var counter;
const allaParkeringarBtn = document.querySelector('#allaParkeringar');
const ledigaParkeringarBtn = document.querySelector('#ledigaParkeringar');

var numberOfClicks = 10;
```

Figur 23: filter-script.js

[9] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/932a273

[10] https://github.com/a17jacsv/examensarbete-vue-vs-react/commit/f7ff7c1

För att resultat och data som samlas i arrayen ”scrapedData” ska skrivas ut på ett automatiserat sätt så anropas funktionen ”ajaxCall”. Funktionen gör ett ”http-request” som hämtar i fallet för Vue.js, filen scraped_receiver-vue.php och exekverar koden som skriver ut varje enskilt resultat från mätningen till en TXT-fil där funktionen ”ajaxCall” hanterar dataöverföringen. Se figur 24 och 25.

```
function ajaxCall(data) {
   try {
      GM_xmlhttpRequest( {
         method: 'POST',
         url: URL,
         data: 'str=' + encodeURIComponent(data),
         headers: {
            Content-Type': 'application/x-www-form-urlencoded'
         },
      });
   } catch (ex1) {
       console.log(ex1);
   }
}
```

Figur 24: filter-script.js

```
<?php
    $fp=fopen("scrapedData-vue.txt","a");
    fputs ($fp, $_POST['str']);
    fclose ($fp);
?>
```

Figur 25: scraped_receiver-vue/react.php

Skriptets timer startar när en av de två knapparna klickas. För att utföra det skapades en ”eventListener” som gör att när knappen klickas får variabeln start tilldelat ”performance.now()” som returnerar tiden som gått sedan webbapplikationen laddades in för första gången (Google Developers, 2019). Om skriptet inte har körts förut så har variabeln ”counter” värdet ”null” och blir därav tilldelat ”0”. Se figur 26.

```
allaParkeringarBtn.addEventListener('click', function() {
//ledigaParkeringarBtn.addEventListener('click', function() {
   start = performance.now();
});

if (counter == null) {
   counter = 0;
}
```

Figur 26: filter-script.js

Funktionen ”script” är den funktion som utför klick och lagrar dess resultat i en array. Det första som sker är att den för tillfället valda knappen, det vill säga ”allaParkeringarBtn” klickas på. För att kontrollera att mätningarna på båda implementationerna av webbapplikationen är likadana används JavaScript-funktionen ”window.requestAnimationFrame”. Funktionen berättar för webbläsarens att koden i funktionen körs när allt är laddat och klart för rendering av webbapplikation. Därefter sätts slut-tiden genom att tilldela variabeln end ”performance.now()”. För att räkna ut resultatet görs en beräkning där variabeln ”end” subtraheras med variabeln ”start”. Det blir en enskild mätnings resultat. Sedan läggs resultatet in i arrayen ”scrapedData” genom att använda sig av ”push”. För att antalet mätningar ska bli exakt det som önskat används en ”counter” som plussas på vid varje enskild mätning. En kontroll görs vid varje enskild mätning som kollar om värdet av ”counter” är mindre än ”numberOfClicks” som består av antalet önskade mätningar. När mätningarna är utförda skapas en ”alert” som berättar hur många mätningar och klick som utförts. Därefter skickas data från arrayen ”scrapedData” genom funktionen ”ajaxCall” in till PHP-filen som hanterar dataöverföringen till ett TXT-fil som gör det smidigare att sammanställa data.

Slutligen startar scriptet om sig genom att anropa funktionen genom ”script()” längst ner i dokumentet. Se figur 27.

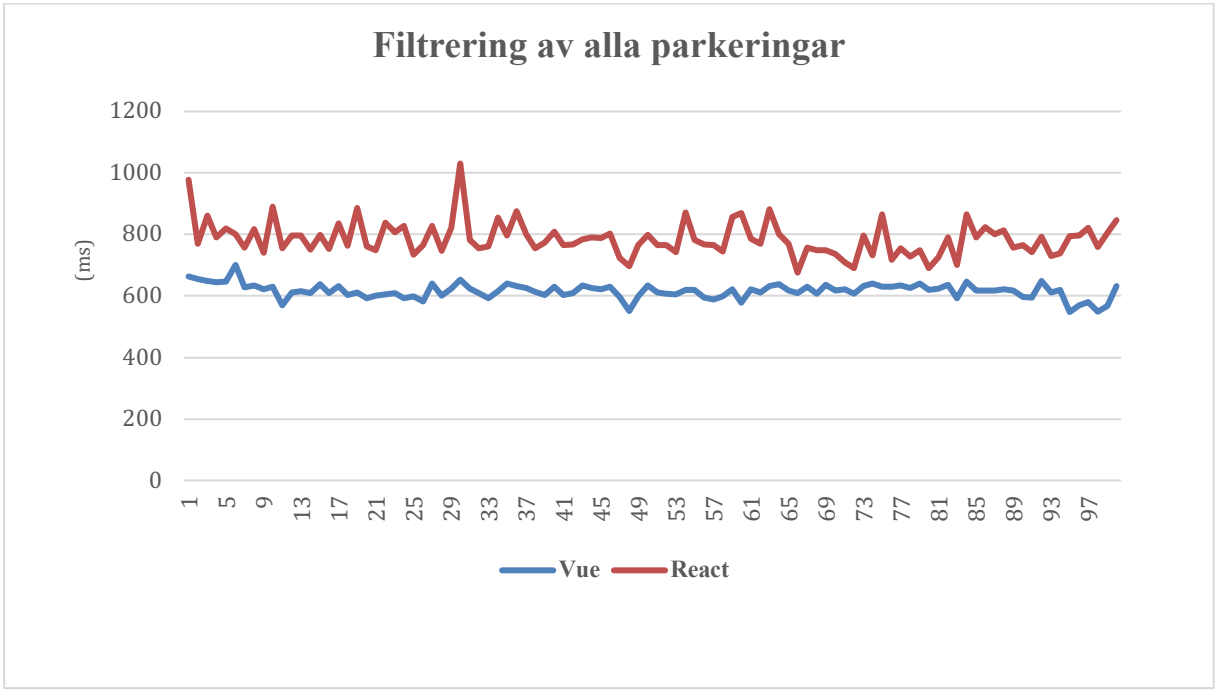
```
function script() {
   allaParkeringarBtn.click();
   //ledigaParkeringarBtn.click();

   window.requestAnimationFrame(function () {
      end = performance.now();
      result = (end - start) + '\n';
      console.log(result);
      scrapedData.push(result);
      counter++;
   });

   if (counter < numberOfClicks) {
      setTimeout(script, 2000);
   }
   else {
      alert("Clicked " + scrapedData.length + " button(s).");
      ajaxCall(scrapedData.join(''));
   }
}

script();
```

Figur 27: filter-script.js

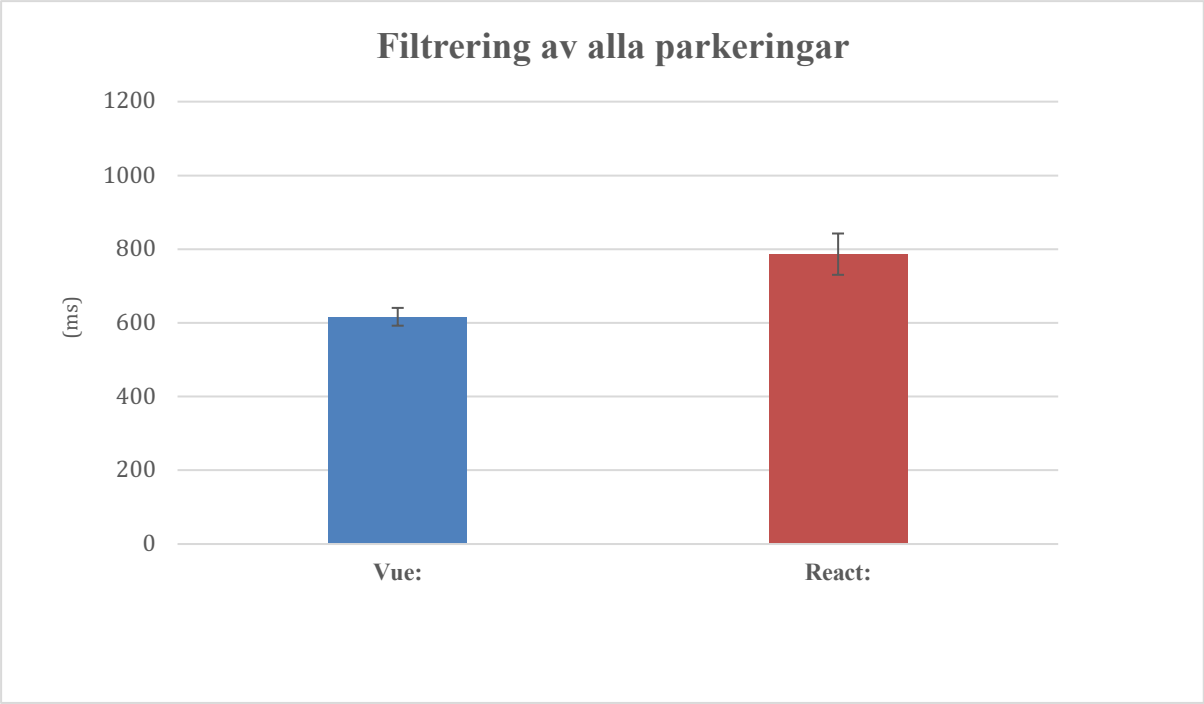
5.6 Pilotstudie

En pilotstudie har utförts för att utvärdera arbetet och den valda metoden i relation till arbetets hypotes som tagits upp tidigare i arbetet. Pilotstudien används för att skapa sig en uppfattning ifall de genomförda mätningarna fungerar på ett korrekt och tilltänkt sätt. Mätningarna sker på båda implementationerna av webbapplikationen genom att används ett egenskrivet skript som jämför ramverken med varandra i millisekunder på likadant sätt.

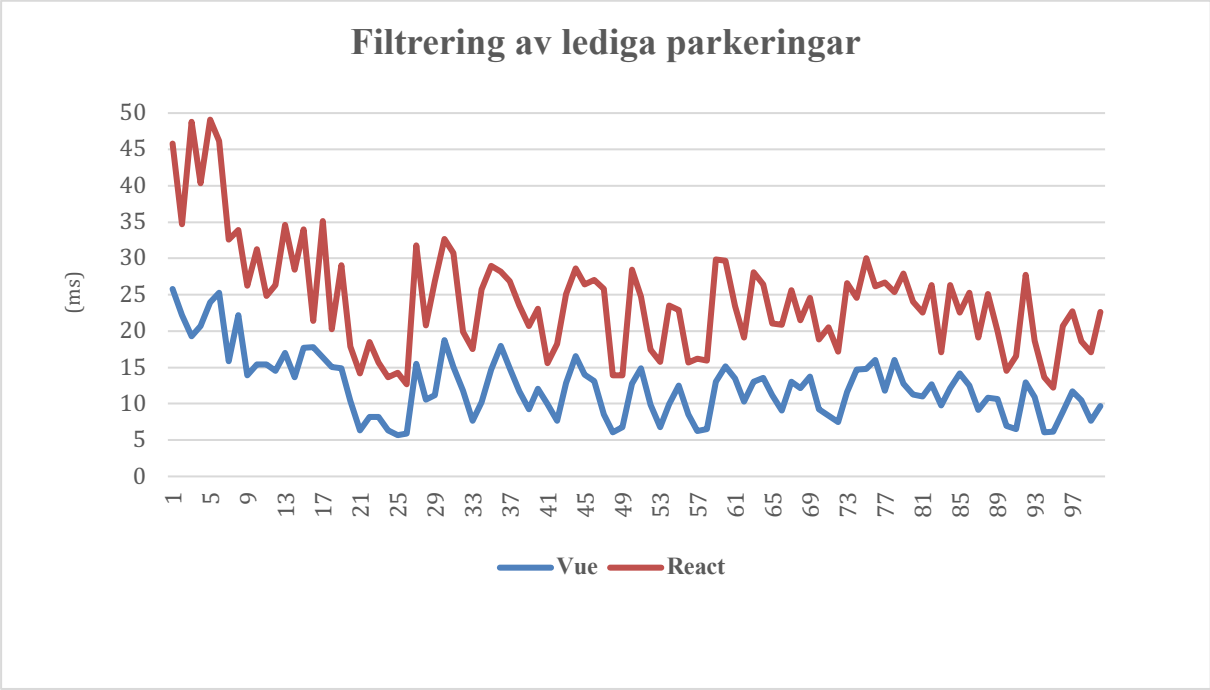
Resultatet av pilotstudien att presenteras i form av linjediagram som presenterar mätdata i form av millisekunder. Figur 28 och 29 nedan representerar 100 mätningar vid filtrering av samtliga parkeringar som innefattar större datamängd. Figurerna 30 och 31 visar resultatet av 100 mätningar av de tillgängliga parkeringarna som består av mindre datamängd.



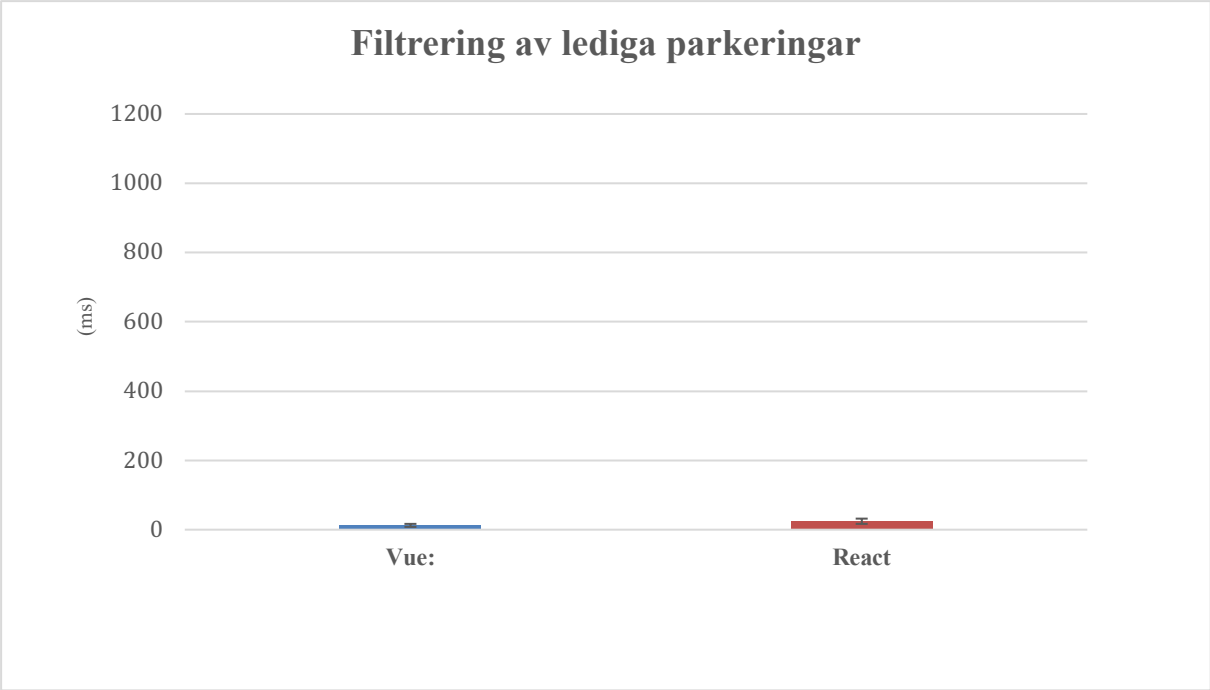
Figur 28: Linjediagram för 100 mätningar av alla parkeringar



Figur 29: Stapeldiagram för 100 mätningar av alla parkeringar



Figur 30: Linjediagram för 100 mätningar av lediga parkeringar



Figur 31: Stapeldiagram för 100 mätningar av lediga parkeringar

5.6.1 Diskussion av pilotstudie

Målet med pilotstudien var att bedöma och se ifall det stundande experimentet går att genomföra. Pilotstudien får anses av arbetet vara lyckad och data som mäts är korrekt. Båda implementationerna av ramverken Vue.js och React mäts med exakt samma skript vilket gör det rimligt att analysera och jämföra resultaten med varandra. Det går inte dra några konkreta slutsatser redan nu med tanke på det mindre dataset som har använts. Utan någon djupare analys var det tydligt att de diagram som presenterades i figurerna 28, 29, 30 och 31 hade märkbara skillnader mellan Vue.js och React mätt i form av millisekunder.

Första jämförelsen som presenteras i figur 28 består av 100 mätningar på filtrering av alla parkeringsplatser vilket variera mellan 3150 – 3500 objekt. Det syns tydligt i linjediagrammet att Vue.js har lägre svarstider än vad React har. Figur 29 presenterar i form av ett stapeldiagram att Vue.js har ett medelvärde på 616 och React har 786 millisekunder. Differens mellan varandra är 170 millisekunder vid filtrering av en större mängd data. Standardavvikelsen visas i figuren via ett t-format streck. Den visar hur stor avvikelse det är data som mäts och hur stor variation det är på resultatet. Vue.js har en standardavvikelse på 24,1 och React på 56,0.

Den andra jämförelsen som visas i figur 30 består även den av 100 mätningar på filtrering av tillgängliga parkeringsplatser som i sin tur varierar mellan 50 – 200 objekt. Det syns även märkbart i linjediagrammet att Vue.js har lägre svarstider än vad React har. Figur 31 visar i form av ett stapeldiagram att Vue.js har ett medelvärde på 12 och React har 24 millisekunder. Standardavvikelsen visas i figuren likt tidigare figurer. Vue.js har en standardavvikelse på 4,3 och React på 7,6. Differens mellan varandra är 12 millisekunder vid en större mängd data. Det innebär att Vue.js presterar dubbelt så snabbt i testfallet.

Av mätningarna som utfördes sågs relativt tydlig skillnad där i mindre antal mätningar presterade Vue.js bäst i form av lägre svarstider än React. Två ANOVA-mätningar utfördes som resulterade i ett p-värde på 7,69 på samtliga parkeringar och 5,23 på lediga parkeringar. Eftersom båda p-värden överstiger 0.05 innebär det inte är en statistisk signifikant skillnad mellan mätningarna. Som tidigare skrivet så används ett litet dataset och mycket kan ändras vid ett större antal mätningar. Längre fram i arbetet kommer större mätningar utföras och mycket kan ändras och det ska bli intressant att se om det är sig likt vid arbetets slutgiltiga resultat.

En brist i arbetet kan vara att ordningen i listan vid varje filtrering mätning slumpas. Det finns inget sparat som gör att varje mätning blir sig lik vilket gör att en exakt replikering av studien inte är möjlig. Antalet parkeringsplatser som filtreras är alltid samma som tidigare med hjälp av en sparad ”seed”. Ramverken Vue.js och Reacts förutsättningar är likdana och resultatet i sig bör inte påverkas av det.

6 Utvärdering

6.1 Presentation av testfall

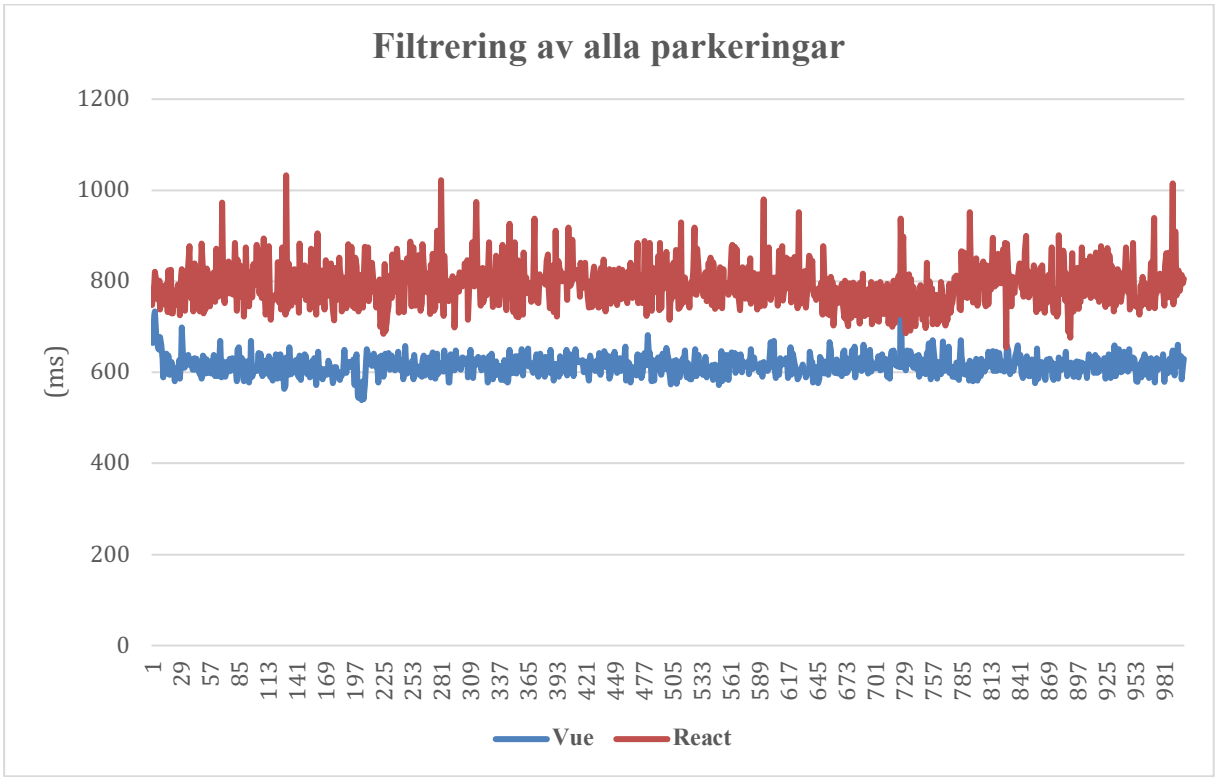
De testfall som finns i arbetet är upplagda på ungefär samma sätt som vid pilotstudien. Det som skiljer sig är antalet mätningar som vid de slutgiltiga mätningarna är 10 gånger fler än vid pilotstudien. Det första testfallet är att utföra 1000 mätningar på en större mängd data, som i arbetet anses vara mellan 3150 – 3500 objekt. Det andra testfallet är 1000 mätningar på en mindre mängd data som i arbetet anses vara 50 – 200 objekt. Dessa två testfall utförs på implementationerna Vue.js och React. Totalt har 4000 mätningar att genomförts för att få fram ett resultat att kunna analysera och diskutera kring. Det blir ett betydligt större dataset än vid pilotstudien vilket gör det möjligt att dra mer konkreta slutsatser och med mål att få ett bättre och mer trovärdigt resultat.

Alla mätningar har utförts på samma dator och på exakta samma vis för att förutsättningarna ska vara likadana. Alla mjukvaror som inte har varit nödvändiga har stängts ner och internet har inaktiverats för att inte skapa några eventuella störningar.

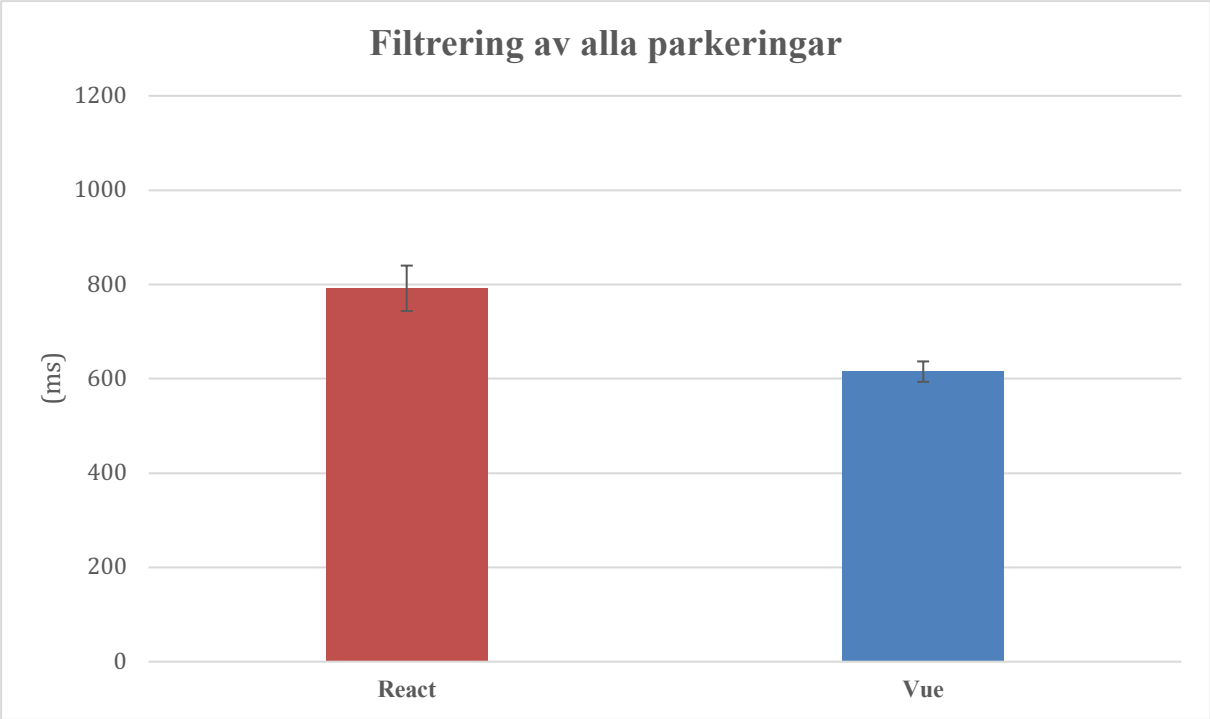
6.2 Resultat av testfall

6.2.1 Större mängd data

Resultaten som tagits fram vid filtrering av större mängd data presenteras nedan i form av två diagram varav ett stapel och ett linje. Valet av diagram anses av arbetet vara det bästa möjliga alternativ för att visualisera resultatet och dess data mätt i millisekunder. Se figur 32 och 33. 1000 mätningar har utförts på implementationen av Vue.js och React på större mängd data.



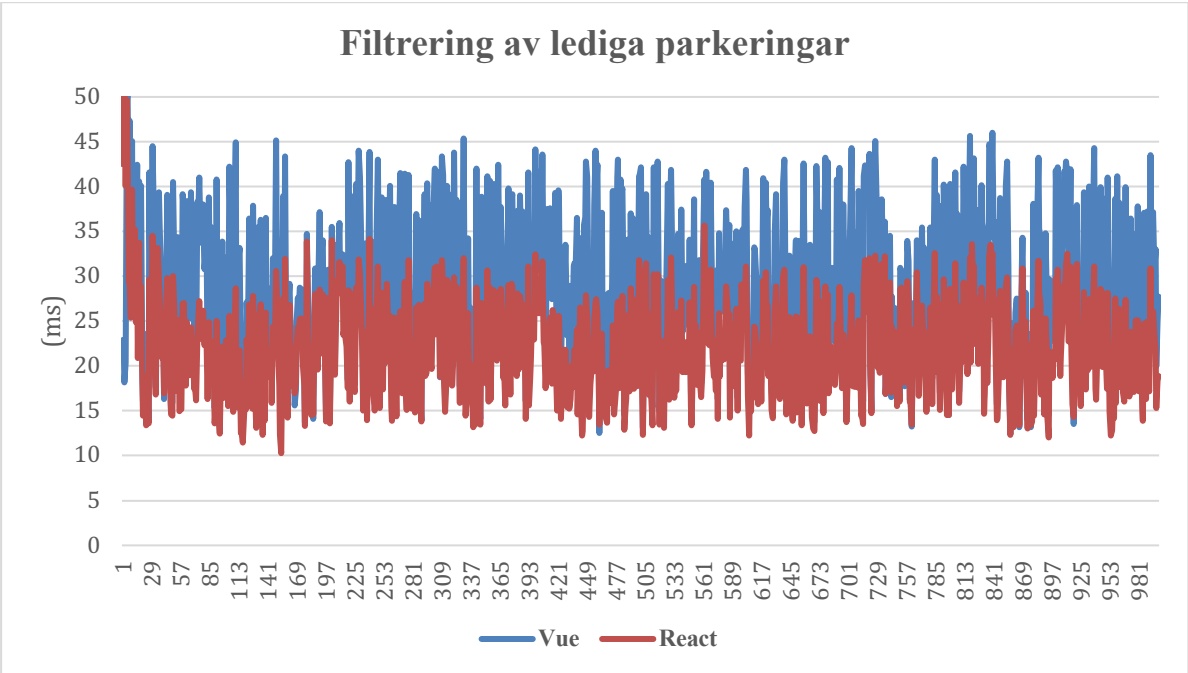
Figur 32: Linjediagram för 1000 mätningar av alla parkeringar



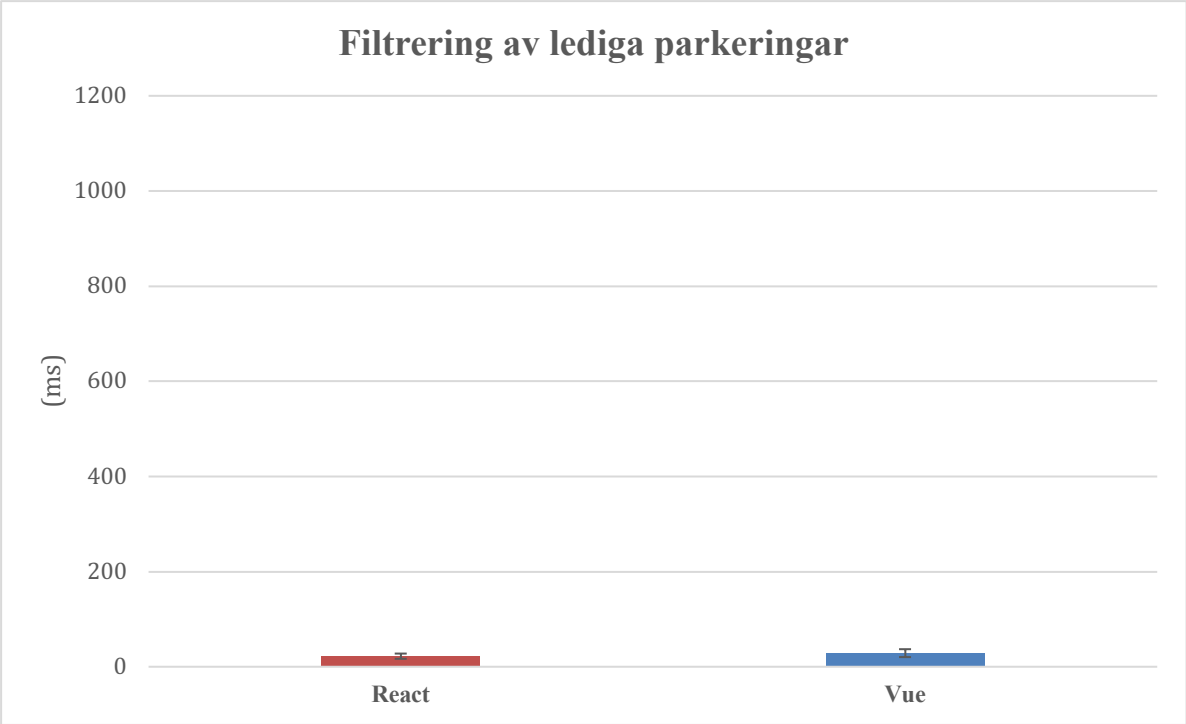
Figur 33: Stapeldiagram för 1000 mätningar av alla parkeringar

6.2.2 Mindre mängd data

Resultaten som tagits fram vid filtrering av mindre mängd data presenteras nedan i form av två diagram varav ett stapel och ett linje. Valet av diagram anses av arbetet vara det bästa möjliga alternativ för att visualisera resultatet och dess data mätt i millisekunder. Se figur 34 och 35. 1000 mätningar har utförts på implementationen av Vue.js och React på mindre mängd data.



Figur 34: Linjediagram för 1000 mätningar av lediga parkeringar

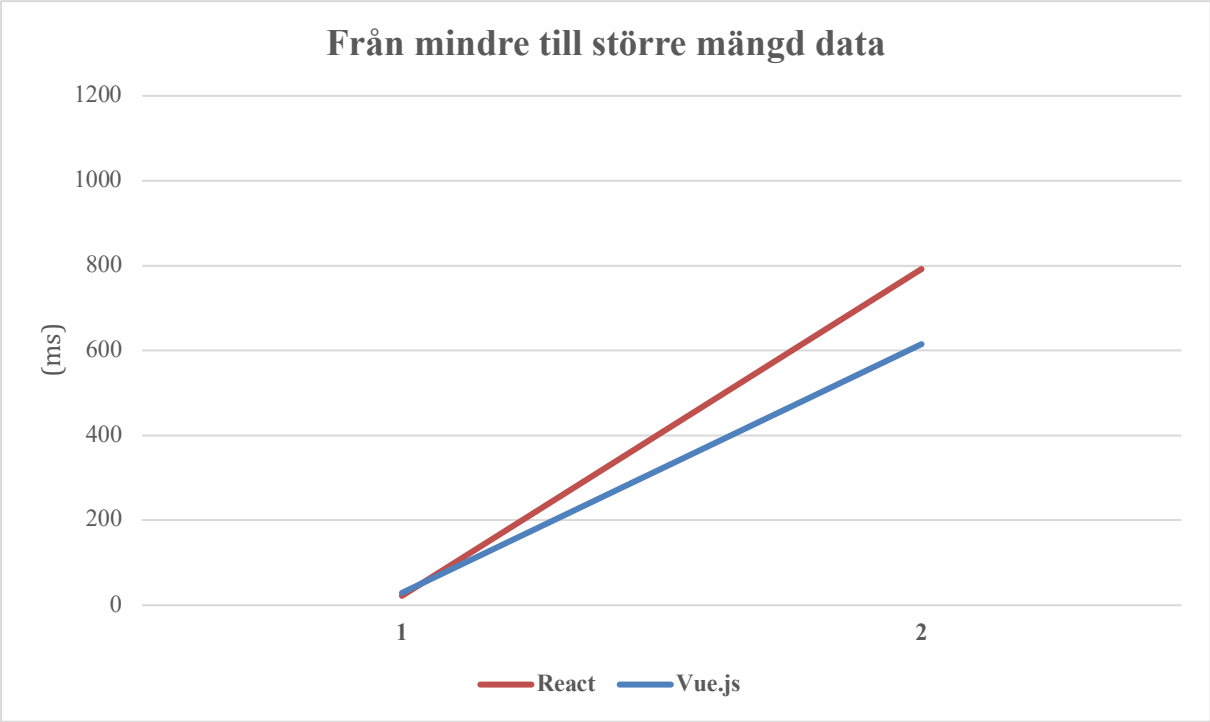


Figur 35: Stapeldiagram för 1000 mätningar av lediga parkeringar

6.3 Analys av resultat från testfall

En analys av experimentets resultat utförs i två steg. Det första steget är att försöka läsa av diagrammen rent visuellt och se vad som kan utläsas genom det. Det steget kan ge mycket och se vad som skiljer testfallen åt. För att få en mer faktisk och konkret skillnad och svar kommer statistik att användas som hjälpmedel i olika former. Det är genom exempelvis genom enklare statistik som medelvärde och standardavvikelser men för att få ett slutgiltigt svar på om nollhypotesen kan avslås eller inte krävs mer djupgående statistik som i arbetet har använt sig av ANOVA-test. ANOVA-testet används för att kunna på ett matematiskt sätt konstatera om det är någon signifikant skillnad mellan de två ramverkens, Vue.js och React, resultat från arbetets mätningar. Om resultatet från ANOVA-testet resulterar i ett p-värde som är mindre än 0.05 går det att konstatera till 95% säkerhet att skillnaden är signifikant och att nollhypotesen kan förnekas.

I figur 36 nedan visas stigningen mellan medelvärdena på mindre och större mängd data ur mätningarna av Vue.js och React. Det är en jämn brant stigning på båda ramverken Vue.js och Reacts kurvor ju fler objekt som läggs till. React resultat visar en något brantare kurva än vad Vue.js vilket kan förklaras med högre svarstider. Det visas en överkorsning vid mindre mängd data eftersom medelvärden var lika, men att vid större mängd data dras kurvorna ifrån varandra och troligtvis ökar exceptionellt ju fler objekt som läggs till. Det går inte att fastslå med tanke på att det endast finns två punkter i diagrammet, men det är sannolikt.



Figur 36: Linjediagram som presenterar ökande av data

6.3.1 Analys av större mängd data

Det finns mycket som kan utläsas genom att visuellt granska linjediagrammet som visas i figur 33. Det syns relativt tydligt att Vue.js presterar lägre svarstider än vad React gör och ingen större överlappning mellan varandra sker. Det förkommer inte några spikdata i någon av mätningarna som urskiljer sig betydligt från resterande data. React visar betydligt mer kraftigt brus än vad Vue.js gör vilket skiljer ramverken åt. Vue.js har också brus, men inte alls lika kraftigt som React och är mer jämn helhetsmässigt. Bruset som visas i linjediagrammet får anses vara på en bra nivå. Det är enligt arbetet avvikelser i data som inte gör att det blir problematiskt.

I figur 34 presenteras ett stapeldiagram som visar de två ramverkens medelvärde och dess standardavvikelser. React har medelvärdet 791 och en standardavvikelse på 48 millisekunder tagits på resultatet. Vue.js har ett medelvärde på 615 och en standardavvikelse på 21 millisekunder. Ramverkens medelvärde har en differens på 176 millisekunder vilket liknar pilotstudiens differens med 170 millisekunder. Genom att utläsa ramverkens standardavvikelse visar det att React har ett brus som nästan är dubbelt så stort som Vue.js där skillnaden mellan ramverken är 27 millisekunder till Vue.js fördel.

För att konstatera att skillnaden mellan de två ramverken, Vue.js och React, är signifikant vid filtrering av en större mängd data utfördes ett ANOVA-test som visas i figur 37 nedan. Där presenteras olika värden som tagits fram ur resultatet där det värde som är av stor vikt är p-värdet. P-värdet har värdet 0 och kan därav fastslå att det är en statistisk signifikant skillnad till 95% mellan ramverkens resultat eftersom det är lägre än 0.05.

ANOVA						
Variation	**KvS**	**fg**	**MKv**	**F**	**p-värde**	**F-krit**
Mellan grupper	15606059,6	1	15606059,6	11179,7673	0	3,84611723
Inom grupper	2789047,96	1998	1395,9199			
Totalt	18395107,6	1999				

Figur 37: ANOVA-test för större mängd data

6.3.2 Analys av mindre mängd data

Det går att utläsa visuellt och granska linjediagrammet som visas i figur 34. Det syns att React presterar något lägre svarstider än vad Vue.js gör och att överlappning mellan ramverken sker. Det förkommer inte några spikdata i något av mätningarna som urskiljer sig betydligt från resterande data. Både React och Vue.js visar samma nivå på brus än och urskiljer sig inte något visuellt från varandra. Bruset som visas i linjediagrammet får anses vara på en rimlig nivå. Det är enligt arbetet avvikelser i data som gör att det inte blir problematiskt.

I figur 36 presenteras ett stapeldiagram som visar de två ramverkens medelvärde och dess standardavvikelser. React har medelvärdet 22 och en standardavvikelse på 5,4 millisekunder tagits på resultatet. Vue.js har ett medelvärde på 28 och en standardavvikelse på 8,3 millisekunder. Ramverkens medelvärde har en differens på 6 millisekunder. Genom att utläsa ramverkens standardavvikelse visar det att bruset är större på Vue.js där skillnaden mellan ramverken är 2,9 millisekunder till Reacts fördel.

För att konstatera att skillnaden mellan de två ramverken, Vue.js och React, är signifikant vid filtrering av en mindre mängd data utfördes ett ANOVA-test som visas i figur 38 nedan. Där presenteras olika värden som tagits fram ur resultatet där det värde som är av stor vikt är p-värdet. P-värdet har värdet 3,4808E-83 som är ett väldigt lågt tal som är lägre än 0.05, vilket gör det möjligt att konstatera att det är en statistiskt signifikant skillnad mellan ramverkens resultat till 95% säkerhet.

ANOVA						
Variation	**KvS**	**fg**	**MKv**	**F**	**p-värde**	**F-krit**
Mellan grupper	20310,4643	1	20310,4643	410,627696	3,4808E-83	3,84611723
Inom grupper	98825,0626	1998	49,4619933			
Totalt	119135,527	1999				

Figur 38: ANOVA-test för mindre mängd data

7 Slutsats

Detta arbete har haft som frågeställning att jämföra de två implementationer som skrivits med ramverken Vue.js och React för att ta reda på vilket av ramverken som presterar lägst svarstid i experiment som utförts. Arbetets hypotes var att det inte skulle vara någon statistisk signifikant skillnad mellan de två implementationerna av Vue.js och React när det gäller filtrering av data innehållandes parkeringsplatser. Det ansågs heller inte att det skulle vara någon skillnad på större respektive mindre mängd data innehållandes parkeringsplatser. Arbetets resultat som tagits fram kan direkt visa att hypotesen inte har varit korrekt. Genom användning av ANOVA-test visar resultatet till 95% sannolikhet att det är en statistisk signifikant skillnad mellan båda testfallen, större och mindre mängd data. Alltså finns det en skillnad mellan implementationerna utförda med React och Vue.js. Att det är skillnad i form av tid är nu konstaterat men hur påverkar dessa skillnader i form av millisekunder användaren.

Differensen på filtrering på en större mängd antal parkeringsplatser medelvärde var 176 millisekunder vilket får anses vara en skillnad som kan vara möjlig att uppfatta trots att det bara är ungefär 0.2 sekunder. Filtrering av en mindre mängd antal parkeringsplatsers medelvärde differens är enbart 6 millisekunder, vilket är extremt kort tid som en användare inte bör uppfatta dess skillnad. Enligt Dabrowski (2001) så märkte inte användare någon skillnad för musinteraktioner upp till 195 millisekunder. Skillnaden finns där mellan ramverken Vue.js och React, men är så liten att det inte bör påverka användarupplevelsen eftersom den inte överstiger 195 millisekunder. Det får anses vara en viktig aspekt vid val av ramverk för sitt projekt att skillnaden i svarstider inte påverkar användaren negativt om den inte uppfattas.

De slutsatser som arbetet kan ta vid filtrering av större mängd data är att Vue.js presterar bättre i form av svarstid än vad React gör. Vue.js har en bättre och mer jämn nivå på sitt brus jämfört med vad React presterade, som varierade betydligt mer. Det gör att vid större och högt antal filtreringar håller en mer jämn nivå vid användning vilket får anses vara positivt ur användarens perspektiv. Vid filtrering av mindre mängd data så är det inte riktigt samma resultat som vid större mängd. React presterar något bättre i form av svarstid än vad Vue.js gör där det skiljer sig 6 millisekunder från ramverkens medelvärde till React's fördel. Det skiljer sig inte mycket när det kommer till brus och både ramverkens data är jämn och inte någon stor variation som urskiljer sig. Om antalet objekt dubbleras skulle kurvan troligtvis fortsätta öka i samma takt som i figur 36, vilket gör att antalet objekt i sig inte har någon större påverkan till varför Vue.js presterar lägre svarstider än React. Det går dock inte att fastslå med tanke på att det endast finns två punkter i diagrammet, men arbetet anser att det sannolikt så är fallet.

Sammanfattningsvis anser arbetet att Vue.js bör användas vid utveckling före React med anledning av ett jämnare brus och innehar lägre svarstid. React hade något lägre svarstider än vad Vue.js hade vid filtrering av mindre mängd data, men som en helhetsbedömning levererade Vue.js bättre på mätningarna än vad React gjorde. Enligt Vue.js (2020) själva, anser att både Vue.js och React vara snabba och att hastighet troligtvis inte skulle vara en avgörande faktor för val mellan ramverken. Denna hypotes fastslås av arbetet eftersom det inte var några exceptionella stora skillnader som skilde ramverken åt.

7.1 Avslutande diskussion

Alla JavaScript ramverk har olika styrkor och svagheter. Både React och Vue.js har fungerat bra och har fyllt arbetets behov med mål att utforma ett teknikorienterade experiment och kunna mäta dess svarstid. Vue.js var enklare att förstå och lära sig utan att ha någon tidigare erfarenhet

i ramverket. Finns kunskap inom JavaScript förstås Vue.js koduppbyggnad och struktur relativt enkelt. Det var lite svårare med React med tanke på att dess struktur och koduppbyggnad urskiljer sig lite mer jämfört med vanlig JavaScript-kod. Det var mer komplext och det märktes att det var mer anpassat för större projekt. Implementationerna hade som mål att vara så lika som möjligt logiskt vilket ibland strulade till det eftersom något av ramverken hade en enklare alternativt svårare lösning på just det problem. Arbetet anser att av de två implementationer som skapades med Vue.js och React så var Vue.js implementation mer enkel och lättare både att förstå och att skriva för just den webbapplikation som skapats i detta arbete.

Arbetet använde sig av ett teknikorienterat experiment för att kunna mäta ramverken Vue.js och Reacts svarstid i ett verkligt dilemma som var en filtrering. Data gick att utvinnas från de två testfall som användes vid mätningarna vilket gick till på ett bra sätt. Mätningarna får anses av arbetet vara lyckade. Det som strävades mot att ta fram lyckades och att med hjälp av ANOVA-test kunna säga att till 95% att det fanns en statistiskt signifikant skillnad mellan de två implementationernas byggda med React och Vue.js resultat. Det visar att Vue.js faktiskt presterat lägre svarstider och har en statistisk uppbackning vilket gör resultatet mer trovärdigt.

Det är viktigt att påpeka att de analyser och slutsatser som tagits har varit helt från mätningarna som tagits fram i arbetet utifrån de två implementationerna av webbapplikation. Det går inte dra några slutsatser på ramverken Vue.js och React i helhet utan de allt som skrivits har utgått helt och hållet från de resultaten som tagits fram i studien. Arbetet anser att Vue.js presterat bättre i denna jämförelse men det betyder inte att de skulle göra det i en annan typ av mätning. Vid filtrering av data som detta arbetet har använt sig av kan utvecklare runt om i världen ta del av denna information som kan hjälpa till att välja ramverk inför ett kommande projekt.

7.2 Samhällsnytta och etik

Ur ett samhällsperspektiv är detta arbete och dess inriktning aktuell. Människor använder sig dagligen av enheter som möter på data och där det finns flera olika sätt att visualisera den. Arbetet anser att visualisering av data är viktigt där användaren själv skräddarsyr den vy som den önskar att se. Det gör att bara användaren ser enbart det den vill se och inget annat, vilket sparar tid. Arbetets mål är att ta reda på vilket ramverk av Vue.js och React som presterar snabbast vid filtrering av data i form av svarstid. Genom att försöka optimera tiden av visualisering av data kan spara mer tid åt annat.

Hållbarhet med en parkeringsapplikation är ur miljösynpunkt bra med tanke på att bilförare slipper åka runt och leta vilket leder till mindre utsläpp från bilarna. Det är en bra lösning för alla parter där miljön, markägaren och användaren påverkas positivt. Vid utveckling av applikation är valet av rätt ramverk viktigt genom att sträva att den ska vara energisnål vilket leder till längre lagring av batteri på sin enhet. En risk med en parkeringsapplikation kan vara vid hantering av registreringsnummer på bilar som kan kopplas till en ägare och dess personuppgifter kan hamna i fel händer. Det gör det möjligt att genom se vart en bil är parkerad kunna veta på ett ungefär vart ägaren av bilen befinner sig vilket är känslig information som kräver lagring med hög säkerhet.

Ett etiskt fokus som har varit genomgående under arbetets gång är möjligheten till replikering. Arbetets progression går att följa på GitHub (Svensson, 2020) där all kod och dess alla versioner är publicerade. På det sättet går det att använda exakt samma kod som arbetet har använt och

genom det finns möjligheten att replikera arbetet. Det gör att vem som helst kan ladda ner allt som krävs för att utföra mätningar på samma sätt som detta arbete har använt. Det går att följa den process som genomförts och se vad som ändrats vid varje ny uppdatering av kod. Det ger mer trovärdighet till det resultat som beskrivits.

7.3 Framtida arbete

Framtida arbeten som utgår från denna avhandling kan fördjupa sig inom filtrering i allmänhet. Det kan vara exempelvis att filtrera en egenskap av ett objekt. Ett annat exempel kan vara att sortera objekt efter alfabetisk eller numerisk ordning. Det går även att studera vad som händer med större antal objekt som till exempel upp mot 10 000 för att se vad som skiljer resultaten åt. Det skulle även kunna utvärderas att filtrera data som är upplagd på en databas och se om det skiljer sig åt. Det finns funktioner som används dagligen som skulle kunna utforskas och försöka optimera och många sätt att fördjupa sig inom ämnet som skulle vara intressant. Det gäller inte enbart de två utvalda ramverken, Vue.js och React, som detta arbete har använt. Ett framtida arbete hade kunnat välja några helt andra ramverk att utföra mätningar på fast på liknande experiment. Experimentet går att variera till ett enklare eller mer komplext beroende på vad som skall undersökas.

Målet bör vara att hitta det ramverk som presterar lägst svarstider och det går att göra på olika vis som inte har med detta arbetes experiment att göra. Ett storskaligt arbete skulle kunna studera och försöka vidareutveckla diverse ramverk med försök till optimering. Det hade resulterat i mer konkret vad som var anledningen till att ramverk presterar lägre svarstider än det andra. Det vore intressant att få en djupare förståelse kring det ämnet för att få en klarare bild om vad som faktiskt skiljer ramverken åt rent praktiskt. Ett annat alternativ som hade varit intressant för framtida arbete vore att se hur svarstider i ett ramverk påverkas när data som används blir mer komplex. Med mer komplex menas till exempel att objekten innehåller sina egna objekt som i sin tur innehåller egna objekt och så vidare.

Mätningar på svarstid ger konkreta svar och visar tydliga skillnader som användare inte alltid uppfattar med tanke på resultatet skiljer i millisekunder. Genom att bara testa svarstid kan resultera i oklarhet. Det finns andra mått som ett framtida arbete kan överväga att göra mätningar och jämförelser på. Det kan exempelvis vara hur komplex koden är samt hur många rader kod det krävs för att utföra samma sak på två utvalda ramverk. På det sättet får inte bara användarens perspektiv utan även utvecklarens. Ett ramverk som presterar några millisekunder snabbare att utföra en filtrering än det andra, men är mycket mer komplex kod kanske det inte är att föredra. Det gäller att ta del av för och nackdelar för att välja rätt ramverk som är bäst anpassat för sitt projekt.

8 Referenser

Angularjs.org. (2020). AngularJS. [online] Available at: https://angularjs.org/ [Viewed 2020-02-13].

Backbonejs.org. (2019). Backbone.js. [online] Available at: https://backbonejs.org/ [Viewed 2020-02-13].

Bassett, L. (2015). Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON. O'Reilly Media. ISBN: 9781491929452

Butkiewicz, M., Madhyastha, H. V. & Sekar, V. (2011). Understanding website complexity: measurements, metrics, and implications. Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference. ACM, ss. 313–328. doi:10.1145/2068816.2068846

Cli.vuejs.org. (2020). Vue CLI [online] Available at: https://cli.vuejs.org/ [Viewed 2020-03-13].

Emberjs.org. (2020). Ember.js [online] Available at: https://emberjs.org/ [Viewed 2020-02-13].

Gackenheimer, C., (2015). Introduction To React. Apress, Berkley, CA. DOI: 10.1007/978-1-4842-1245-5

Github.com. (2020). GitHub. [online] Available at: https://github.com / [Viewed 2020-03-06].

Gizas, A., Christodoulou, S & Papatheodorou, T. (2012). Comparative evaluation of JavaScript frameworks. WWW'12 - Proceedings of the 21st Annual Conference on World Wide Web Companion. DOI: 10.1145/2187980.2188103.

Godfrey, P., Gryz, J. & Lasek, P. (2016). Interactive Visualization of Large Data Sets. IEEE Transactions on knowledge and data engineering, 28(8), ss. 2142-2157.
DOI: 10.1109/TKDE.2016.2557324.

Google Developers. (2019). When milliseconds are not enough: performance.now [online] Available at: https://developers.google.com/web/updates/2012/08/When-milliseconds-are-notenough-performance-now [Accessed 2020-04-10].

Ionita, A., Pomp, A., Cochez, M., Meisen, T. & Decker, S. (2018). Where to Park? Predicting Free Parking Spots in Unmonitored City Areas. In Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics (WIMS '18). Association for Computing Machinery, New York, NY, USA, Article 22, 1–12. DOI: 10.1145/3227609.3227648

Dabrowski, J., Munson, E. (2001). Is 100 Milliseconds Too Fast? In CHI '01 Extended Abstracts on Human Factors in Computing Systems (CHI EA '01). Association for Computing Machinery, New York, NY, USA, 317–318. DOI: 10.1145/634067.634255

Javeed, A. "Performance Optimization Techniques for ReactJS," 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 2019, pp. 1-5. DOI: 10.1109/ICECCT.2019.8869134

Liere, R., Adriaansen, T., & Zudilova-Seinstra, E. (2009). Trends in Interactive Visualization: State-of-the-Art Survey. London: Springer-Verlag. ISBN: 978-1-84800-268- 5

Nelson, B. (2018). Getting to know Vue.js. Apress, Berkeley, CA. DOI: 10.1007/978-1-4842-3781-6
Nodejs.org. (2020). Node.js [online] Available at: https://nodejs.org/en/ [Viewed 2020-03-19]

Nikulchev, E., Ilin, D., Kolyasnikov, P., Zakharov, I., Kasatonov, S. & Biryukov, D. (2018). Selection of Architectural Concept & Development Technologies for the Implementation of a Web-Based Platform for Psychology Research. In: Computing Conference, Volume 1. Springer, pp.672-685.

Perez J. M. M., Bernabe J. B., Manzano D. J. M., Perez G. M. & Skarmeta A. F. G. (2008). Towards the Definition of a Web Service Based Management Framework. 2008 Second International Conference on Emerging Security Information, Systems and Technologies. Cap Esterel, France 25-31 august 2008, ss. 362-367. DOI: 10.1109/SECURWARE.2008.19.

Polymer-project.org. (2020). Polymer.js. [online] Available at: https://polymer-project.org/ [Viewed 2020-02-13].

Reactjs.org 1. (2020). React.js. [online] Available at: https://reactjs.org/ [Viewed 2020-02-13].

Reactjs.org 2. (2020). React.js. [online] Available at: https://reactjs.org/docs/hooks-state.html [Viewed 2020-03-30].

Svensson, J. Examensarbete-vue-vs-react, Repository for ”Examensarbete med inriktning mot webbprogrammering IT606G VT20” by Jacob Svensson. (2020), Github Repository, https://github.com/a17jacsv/examensarbete-vue-vs-react [Viewed 2020-05-25]

Tampermonkey.net. (2020). Tampermonkey for Chrome. [online] Available at: https://tampermonkey.net/ [Accessed 2020-04-10].

Vuejs.org. 1 (2020). Vue.js. [online] Available at: https://vuejs.org/ [Viewed 2020-02-13].

Vuejs.org. 2 (2020). Vue.js. [online] Available at: https://vuejs.org/v2/guide/comparison.html [Viewed 2020-05-15].

Wang, W., Liu, C. & Zhao, D. "How Much Data Are Enough? A Statistical Approach With Case Study on Longitudinal Driving Behavior," in IEEE Transactions on Intelligent Vehicles, vol. 2, no. 2, pp. 85-98, June 2017. DOI: 10.1109/TIV.2017.2720459

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. & Wesslén, A. (2012). Experimentation in Software Engineering. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-642-29044-2

W3Schools.com. (2020) W3 Schools. [online] Available at: https://www.w3schools.com/jsref/jsref_slice_array.asp/ [Viewed 2020-03-31]

Zhou, R., Shao, S., Li, W. & Zhou, L. (2016). How to define the user's tolerance of response time in using mobile applications. 281-285. DOI: 10.1109/IEEM.2016.7797881.