Examensarbete

HÖGSKOLAN I SKÖVDE

FILTRERING AV DATA HOS OLIKA JAVASCRIPT RAMVERK

Med fokus på svarstider

DATA FILTERING IN DIFFERENT JAVASCRIPT FRAMEWORKS

With focus on response times

Examensarbete inom huvudområdet Informationsteknologi
Grundnivå 30 högskolepoäng
Vårtermin 2024

Emma Sarri Kayani

Handledare: András Márki
Examinator: Henrik Gustavsson

Sammanfattning

Ett viktigt koncept vid utvecklandet av webbapplikationer är att skapa interaktivitet. För att presentera data på ett lättläst och organiserat sätt kan filtrering användas. Filtrering ger användarna möjligheten att välja vilken data som ska visas upp. Information som visas på webben ska vara både enkelt att hitta samt ge snabba svar. Den här studien bygger på ett experiment där skillnader i svarstider vid filtrering av data undersöks hos olika JavaScript ramverk. Fem tester med olika datamängder har genomförts för att se hur ramverkens svarstider responderar vid skalbarhet. Utifrån ett antal urvalskriterier valdes tre ramverk ut för studien: React, Vue.js och Svelte. Testerna visar att det finns signifikanta skillnader mellan ramverken. Resultatet visar att Svelte är det ramverk som ger snabbast svarstider vid samtliga datamängder. Framtida arbeten kan driva den här studien vidare genom att lägga till fler JavaScript ramverk eller genom att jämföra helt andra JavaScript ramverk med varandra. Ett mer omfattande framtida arbete kan involvera att konstruera en lösning som underlättar integrationen mellan olika ramverk.

Nyckelord: Svarstider, Filtrering, JavaScript ramverk, React, Vue.js, Svelte

Innehållsförteckning

1 Introduktion

Interaktivitet har länge haft en viktig roll inom utveckling av system och applikationer. Genom att skapa interaktion mellan människor och system kan användare själva styra över vad som ska ske på skärmen. För att kunna låta användare välja att endast visa upp relevant data till de som dem eftersöker används filtrering (Godfrey, Gryz & Lasek, 2016). Vid användandet av webbapplikationer förväntar sig användare att webbsidorna ska vara både välorganiserade och snabba (Bartuskova & Krejcar, 2015). Inom webbdesign anses det vara ett misstag att ha långa svarstider (Nielsen, 1997) eftersom användare blir både irriterade och frustrerade om det dröjer innan ett resultat visas på skärmen (Shneiderman, 1984).

För att skapa interaktivitet hos webbapplikationer kan det populära programmeringsspråket JavaScript användas. Inom JavaScript är det möjligt att lägga till olika händelsehanterare för olika komponenter som lyssnar och reagerar efter händelser från användaren (Ocariza, Bajaj, Pattabiraman & Meshbah, 2017). Programmeringsspråket kan beskrivas som lätt, flexibelt och kraftfullt (Jiang, Zhong & Meng, 2021).

I och med att JavaScript har ökat i popularitet har allt fler ramverk utvecklats. Ramverk kan beskrivas som ett applikationsskelett som besitter en hel del olika egenskaper och används flitigt av utvecklare eftersom det underlättar skapandet av webbapplikationer (Pano, Graziotin & Abrahamsson, 2019). Dock kan det anses vara en utmaning för utvecklare att välja de ramverk som uppfyller ett projekts krav på bästa sätt (Graziotin & Abrahamsson, 2013). Det finns ett flertal olika faktorer som utvecklare tar hänsyn till vid valet av ramverk. Enligt en modell som skapats av Pano et al. (2018) finns det fyra områden innehållande flera olika eftertraktade funktioner som utvecklare tittar efter vid valet av ramverk.

När det kommer till hantering och uppdatering av data så finns det olika tillvägagångssätt som ramverken använder sig av (Ollila, Mäkitalo & Mikkonen, 2022). Problemet är att en webbapplikations svarstider kan påverkas beroende på vilka tekniska funktionaliteter som används. Enligt Gizas, Christodoulou & Papatheodorou (2012) är valet av ramverk viktigt för att eftersträva kod med hög kvalité och hög prestanda. Webbapplikationen ska samtidigt leva upp till användarnas förväntningar över svarstider, annars finns risken att användaren tappar intresset och lämnar sidan (Weinberg, 2001).

Syftet med studien är att jämföra olika JavaScript ramverk vid filtrering av data i tabellform för att se om det finns några skillnader i svarstider. För att utvärdera svarstiden hos JavaScript ramverken kommer ett experiment att utföras. Flera implementationer av en webbapplikation kommer att utvecklas för att sedan testas med olika datamängder vid filtrering för att få fram data som kan analyseras och utvärderas.

2 Bakgrund

I det här kapitlet kommer grundläggande begrepp och tekniker att förklaras för att ge insikt till problemformuleringen, metodbeskrivningen och experimentets genomförande.

2.1 Interaktivitet & filtrering av data

Meningen med interaktion är att kunna välja vilken data som ska presenteras vid analysering av stora datamängder. Även om datamängderna inte var lika stora förr i tiden som de är idag så har interaktivitet alltid haft en betydande roll inom system. En mer meningsfull och representativ data kan framföras genom att låta användaren själv justera värden på parametrar utefter behov. I fall användaren inte är nöjd med de valda inställningarna går det enkelt att ändra inställningar för att få fram ny data (Godfrey, Gryz & Lasek, 2016).

I de flesta fall börjar människor med att göra några få val för att sedan förfina och minimera det som visas till en mindre och mer detaljerad skala. Genom att sortera vad som ska väljas så förväntas det inte att den data som visas ska vara exakt utefter det som söks, dock förväntas snabba resultat. Ett sätt att dela upp data på är filtrering, vilket innebär att endast data som matchar de valda attributet/attributen presenteras. Med filtrering går det att minska mängden data som visas upp genom att den data som inte anses vara lika relevant för sökningen inte tas med (Godfrey et al., 2016). I figur 1 visas ett exempel på hur filtrering av data i en tabell kan se ut.

Figur 1 Bild på filtrering av data i en tabell

2.2 Laddningstider & svarstider

Webbsidor har mycket att leva upp till och förväntningarna över deras prestationer ökar med åren. Enligt Nielsen (1999) har den grundläggande svarstidsrekommendationen för webbsidor varit densamma i flera år. För att användare ska kunna navigera fritt på en sida ska svarstiden ligga på undersekunden. Allt som tar längre än 1 sekund drabbar användarna negativt (Nielsen, 1999).

Ett systems svarstid definieras av den hastighet som det tar från och med att en användare har matat in ett kommando och till dess att användaren fått ett svar (Selvidge et al., 2002). År 1997 beräknades det att högsta tillåtna svarstiden var upp till 10 sekunder innan användarna tappade sitt intresse. Långa laddningstider hos en webbsida låg högt upp på listan gällande misstag inom webbdesign (Nielsen, 1997). Användare blir både irriterade och frustrerade om det tar för långt tid innan ett resultat visas på skärmen efter att de gjort en kommandoinmatning. De flesta föredrar snabba interaktioner med systemet och det kan även leda till en ökning i produktivitet. Långa svarstider kan leda till en större risk av att mänskliga fel inträffar samt att användare känner sig mindre nöjda (Shneiderman, 1984).

Tiden det tar för en webbsida att ladda om är en avgörande faktor när det kommer till användarupplevelsen (Zari, Saiedian & Naeem, 2001). Användare anser att en webbsida inte endast ska vara snabb utan den ska också vara välorganiserad, så att det lätt går att hitta den information som eftersöks (Bartuskova & Krejcar, 2015).

2.3 JavaScript

JavaScript är ett programmeringsspråk som har legat i topp under flera år och är idag en av de populäraste för webbprogrammering (Pano, Graziotin & Abrahamsson, 2018). Anledningen till att JavaScript blivit populärt är på grund av sin lätthet, flexibilitet och kraftfullhet. Några av de egenskaper som JavaScript besitter är följande: dynamiskt, förstklassiga funktioner, objektorienterat programmeringsspråk (som är klassfritt och använder sig av prototypiska arv) samt har möjligheten att låta objekt ärva egenskaper från andra objekt direkt (Jiang, Zhong & Meng, 2021).

JavaScript ligger på klientsidan hos en webbapplikation och kan användas för att skapa interaktivitet. Interaktivitet kan skapas genom att lägga till händelsehanterare för olika komponenter som exempelvis knappar, länkar eller inmatningsrutor som lyssnar och reagerar efter händelser som sker när användare interagerar med webbapplikationen. JavaScript hanterar även asykron exekvering, vilket betyder att webbsidan kan uppdateras med ny information efter att ha skickat en förfrågan till servern. Det tar dock inte slut där utan JavaScript är även byggt på ett sådant sätt att det kan interagera med DOM, som står för Document Object Model (Ocariza, Bajaj, Pattabiraman & Meshbah, 2017).

Vidare förklarar Ocariza et al. (2017) att DOM består av en dynamisk trädliknande struktur som har hand om komponenterna i webbapplikationen och hur de är organiserade. Genom att använda sig av DOM API-anrop med JavaScript är det möjligt att nå eller manipulera komponenter som finns lagrade i DOM trädet. Vilket resulterar i att webbsidan inte behöver laddas om för att kunna ändras (Ocariza et al., 2017).

2.4 JavaScript Ramverk

Ett ramverk är ett så kallat applikationsskelett som innehåller en uppsättning av verktyg, funktioner och abstraktioner på en hög nivå. Det finns olika ramverk som är designade för olika syften. I takt med att JavaScript har stigit i popularitet så har allt fler ramverk släpps på marknaden medan andra redan existerande har utökat sina funktionaliteter (Pano et al., 2018). Allt fler ramverk innebär utmaningar och med många olika valmöjligheter kan det bli svårt för utvecklare att välja de ramverk som är mest gynnsamt för deras projekt. (Graziotin & Abrahamsson, 2013). Att välja rätt ramverk kan ha en stor betydelse vad gäller både kvalité och prestanda av kod (Gizas, Christodoulou & Papatheodorou, 2012). Utöver ramverk så finns även JavaScript bibliotek som innehåller förskriven JavaScript-kod. Biblioteken ger stöd till de funktionella och estetiska sidorna av webbutveckling och kan antingen grupperas efter funktionalitet eller vara en del av ett ramverk (Pano et al., 2018). Ett programmeringskoncept som används hos en del ramverk och bibliotek är virtual DOM. En virtual DOM är en idealisk eller ”virtuell” representation av nodträdet som hålls i minnet och synkroniseras med den ”riktiga” DOM (Diniz-Junior et al., 2022).

2.4.1 Vue.js

Vue.js är ett open-source ramverk inom JavaScript under MIT-licens. Ramverket används för att bygga webbanvändargränssnitt och kan beskrivas som ett progressivt ramverk med ett intuitivt API (Vue.js, 2024a). Vue.js använder sig av Single-File component vilket kapslar in JavaScript, HTML och CSS i en och samma fil (Vue.js, 2024b). Det finns flera olika inbyggda direktiv inom ramverket och några exempel är: *v-for* som renderar element flera gånger utifrån källdata, *v-bind* som binder ett eller flera attribut eller binder en komponentprop till ett uttryck dynamiskt och *v-on* som knyter en händelsehanterare till elementet (Vue.js, 2024c). Vue.js är mångsidigt då de skalar mellan ett bibliotek och ett fullfjädrat ramverk (Vue.js, 2024a). I figur 2 visas ett kodexempel från Vue.js.

Input :

```
<script setup>
import{ref} from 'vue'
const msg = ref('Hello Vue')
</script>
<template>
<h1>{{ msg }}</h1>
<h2>{{ msg }}</h2>
<h3>{{ msg }}</h3>
</template>
<style scope>
h1 {color: darkorange}
h2{color: dodgerblue}
h3{color: darkred}
</style>
```

Output:

Figur 2 Exempelkod för Vue.js

2.4.2 Angular

Angular är ett ramverk som är anpassat för att kunna bygga skalbara webbapplikationer. Ramverket inkluderar en samling av välintegrerade bibliotek samt en uppsättning av utvecklarverktyg som är till för att hjälpa användarna utveckla, bygga, testa och uppdatera

kod. Komponenter är en viktig del i Angular som syftar till att ge en välorganiserad struktur till applikationen så att koden kan bli underhållsbar och skalbar (Angular, 2023). Angular är open-source under MIT licens Angular, 2024). I figur 3 visas ett kodexempel från Angular.

Input :

```
@Component({
  selector: 'app-root',
  standalone: true,
  template: `
    <h1 style="color: darkorange;">{{msg}}
    </h1>
    <h2 style="color: dodgerblue;">{{msg}}
    </h2>
    <h3 style="color: darkred;">{{msg}}
    </h3>
  `,
})
export class DemoComponent {
   msg = 'Hello Angular';
}
```

Output:

Figur 3 Exempelkod för Angular

2.4.3 React

React är ett JavaScript bibliotek som används för att bygga användargränssnitt av olika komponenter som sedan kan kombineras (React, 2024a). Komponenterna består av JavaScript funktioner och kan bestå av endast en knapp eller en hel sida. React använder sig ”hooks” och de funktioner som använder sig av detta börjar med ”use”. Ett exempel på en av Reacts hooks är ”useState ” som gör att komponenter kan ta emot data och returnera det som ska visas upp. En useState returnerar två värden: det aktuella tillståndet och en funktion som gör det möjligt att uppdatera tillståndet (React, 2024b). React är open-source under MIT (Github facebook/react, 2022). React benämns även som ett ramverk, i en artikel skriven av Ollila, Mäkitalo & Mikkonen (2022) hänvisar de till React som ett ramverk. I figur 4 visas ett kodexempel från React.

Input :

```
function App() {
const msg = 'Hello React'
return (
<div>
<h1 style={{color:"darkorange"}}>{msg}</h1>
<h2 style={{color:"dodgerblue"}}>{msg}</h2>
<h3 style={{color:"darkred"}}>{msg}</h3>
</div>
  );
}
export default App
```

Output:

Figur 4 Exempelkod för React

2.4.4 Svelte

Svelte är ett JavaScript ramverk som är open-source under MIT licens och används för att skapa webbapplikationer (Svelte, 2023a). Komponenterna är ramverkets byggstenar och Sveltes filer består av ett så kallat supeset av HTML där skript, styling och markup kan skrivas i samma fil. Påståenden (engelska: 'statements') som inte ligger inuti ett block eller en funktion kan göras reaktiva genom att använda "$:". Om de värden som påståendet är beroende av ändras kommer det reaktiva påståendet köras efter annan skriptkod och innan en komponents markup har renderats (Svelte, 2023b). Mycket av arbetet i Svelte flyttas från webbläsaren till kompileringssteget så att manuella optimeringar inte längre ska vara nödvändigt (Svelte, 2023a). I figur 5 visas ett kodexempel från Svelte.

Input :

```
script>
let msg = 'Hello Svelte'
</script>
<h1>{msg}</h1>
<h2>{msg}</h2>
<h3>{msg}</h3>
<style>
  h1{color: darkorange}
  h2{color: dodgerblue}
  h3{color: darkred}
</style>
```

Output:

Hello Svelte

Hello Svelte

Hello Svelte

Figur 5 Exempelkod för Svelte

2.4.5 Qwik

Qwik är open-source under MIT licens (Github QwikDev/qwik, 2024). Qwik är ett webbramverk som riktar in sig på att kunna ge omedelbar laddning av webbapplikationer. Eftersom ramverket inte kräver någon hydratisering, är det möjligt skapa appar som har omedelbar interaktivitet, oavsett storlek och komplexitet. När det gäller prestanda erbjuder Qwik helsidesladdningar under sekunden (Qwik, 2023). I figur 6 visas ett kodexempel från Qwik.

Input :

```
const msg = 'Hello Qwik'
export default component$(() => {
  useStyles$(`
    h1 {color: darkorange;}
    h2 {color: dodgerblue;}
    h3 {color: darkred;}
  `);
  return(
    <div>
       <h1>{msg}</h1>
       <h2>{msg}</h2>
       <h3>{msg}</h3>
    </div>
  )
```

Output:

Hello Qwik

Hello Qwik

Hello Qwik

```
});
```

Figur 6 Exempelkod för Qwik

2.5 Vad kan leda till valet av ett JavaScript ramverk?

Det finns ett flertal olika JavaScript ramverk att välja mellan med en mängd olika egenskaper och funktioner.

I en studie gjord av Pano et al. (2018) utforskar dem de olika faktorerna som leder till valet av ett JavaScript ramverk. Arbetet är en vidareutveckling av teorier och modeller som gjorts av tidigare forskning genom åren. För att ta reda på faktorerna som ligger till grund vid valet av ett ramverk utfördes semi-strukturerade intervjuer med expertutvecklare, beslutsfattare inom företag eller entreprenörer. Alla med tillräckligt mycket erfarenhet för att kunna motivera sina val av ett JavaScript ramverk.

Målet med studien var att ta reda på vad som influerar valet av ett JavaScript ramverk gentemot ett annat och identifiera samt förstå de olika faktorerna bakom valet. Resultatet visade på fyra olika områden med diverse eftertraktade funktioner. Områdena och funktionerna sattes ihop till en modell och kan användas som utgångspunkt för att utvärdera och bekräfta nya eller redan valda JavaScript ramverk (Pano et al., 2018).

I tabell 1 presenteras de fyra olika områdena och dess respektive funktioner. Utövare är fria att välja hur de ska gå till väga i valet av ett ramverk och utifrån vilka faktorer de baserar sitt val på. Modellen som Pano et al. (2018) har tagit fram kan användas som en guide i valet av ett JavaScript ramverk.

Tabell 1 Områden och funktioner presenterade av Pano et al. (2018).

Område	Funktioner
Förväntad prestanda	• Prestanda • Storlek
Ansträngningsförväntning	• Automatisering • Lärbarhet • Komplexitet • Förståelighet
Socialt inflytande	• Konkurrensanalys • Kollegial rådgivning • Gemenskapens vidd (engelska: 'community size') • Gemenskapens lyhördhet (engelska: 'community responsiveness')
Underlättande förhållanden	• Lämplighet • Uppdateringar • Modularitet • Isolering • Sträckbarhet
Prisvärde	• Kostnader

3 Problemformulering

Under de senaste åren har webben blivit en betydelsefull källa för människor när det kommer till att få åtkomst till information. Webbtabeller spelar en avgörande roll som förmedlare av omfattande information på internet och används flitigt av webbsidor (Xu, Yang & Shi, 2010). Användare vill få sin information snabbt och enkelt samtidigt som det ska vara lätt att hitta på webbapplikationen (Bartuskova & Krejcar, 2015). En viktig faktor inom utveckling är att ge användare möjligheten att interagera med applikationen. Detta tillåter användare att själva kunna styra över vad som ska ske på skärmen. Med hjälp av filtrering kan ett mer representativ resultat framföras eftersom endast efterfrågad data visas upp (Godfrey et al., 2016).

JavaScript är ett programmeringsspråk som används för att kunna skapa interaktivitet hos webbapplikationer. Interaktivitet implementeras genom händelsehanterare för olika komponenter (Ocariza et al., 2017). Användandet av JavaScript har ökat vilket även har lett till att allt fler ramverk tillkommit, både nya och uppdaterade versioner tillförs frekvent. JavaScript ramverk används ofta av utvecklare eftersom de innehåller flera egenskaper som underlättar vid byggandet av webbapplikationer (Pano et al., 2018). För att eftersträva kod med hög kvalité och hög prestanda spelar valet av ramverk en viktig roll (Gizas et al., 2012).

När det kommer till att hantera uppdatering av data använder sig ramverken av olika tillvägagångssätt. En teknik är att gå igenom underträdet till den komponent som berörs, vilket betyder att komponenter som finns längre upp i trädet inte påverkas av eventuella ändringar. På så sätt bearbetas alla nödvändiga komponenter vid en förändring, men det kan kräva onödigt arbete för de efterkommande komponenterna och funktionerna vars utgång inte förändrats. En annan teknik för att hantera uppdateringar av data är att endast bearbeta den komponent där en förändring har skett. Tekniken använder sig av begränsad reaktivitet, vilket gör det möjligt att deklarera värden som skapats genom beräkningar utifrån andra värden. När ett värde ändras kommer beroende värden att uppdateras automatiskt. Därmed behöver inte opåverkade komponenter kontrolleras eftersom de komponenter och funktioner som berörs markeras automatiskt (Ollila et al. 222).

Problemet är att svarstiden hos en webbapplikation kan påverkas av de tillvägagångssätt som ett ramverk använder sig av för att hantera uppdateringar av data. Ramverken i sig är beroende av deras underliggande implementationer som har en inverkan på hur data bearbetas. Enligt Nielsen (1999) ska en webbapplikation leva upp till snabba svarstider under sekunden. Weinberg (2001) menar på att användare kommer att lämna en sida om den inte är tillräckligt snabb enligt användarens preferenser. Zari et al. (2001) nämner även att laddningstiderna hos webbapplikationen en avgörande faktor för att kunna ge en bra användarupplevelse.

Syftet med studien är att jämföra olika JavaScript ramverk vid filtrering av data i tabellform för att kunna dra en slutsats om vilket ramverk som ger snabbast svarstider. Studien kommer även undersöka om det finns några signifikanta skillnader i svarstider hos ramverken vid skalbarhet.

3.1 Frågeställningar

De frågeställningar som ska besvaras i studien är följande:

1. Vilket JavaScript ramverk har kortast svarstider vid filtrering av data i tabellform?
2. Finns det några signifikanta skillnader i svarstider hos ramverken vid skalbarhet?

3.2 Hypoteser

H1.0: Det kommer inte att finnas en signifikant skillnad på svarstider beroende på vilket JavaScript ramverk som används.

H1.A: Det kommer att finnas en signifikant skillnad i svarstider beroende på vilket JavaScript ramverk som används.

H2.0: JavaScript ramverken kommer inte att ha någon signifikant skillnad i svarstider oavsett datamängd.

H2.A: JavaScript ramverken kommer att ha signifikant skillnader i svarstider beroende på storleken av datamängden.

3.3 Delmål

Ett antal delmål för studien har satts upp. Delmålen ligger till grund för att komma fram till studiens centrala mål samt ge svar till frågeställningarna.

1. Identifiera potentiella JavaScript ramverk och välja ut möjliga ramverk enligt tidigare identifierade urvalskriterier.
2. Identifiera ett sätt att mäta och jämföra svarstiden på hos de utvalda ramverken.
3. Genomföra jämförelser i en kontrollerad miljö och lagra resultaten.
4. Sammanställa, visualisera och analysera resultaten från delmål 3 och dra en slutsats för att besvara frågeställningarna.

3.4 Metodbeskrivning

Vid utförandet av en forskningsstudie är det viktigt att använda sig av en lämplig metod utifrån studiens mål. Valet av metod är avgörande för hur data samlas in, analyseras och tolkas. Beslutet kan påverka både arbetets kvalité och relevans. I den här studien kommer en empirisk metod att användas. Enligt Wohlin et al. (2012) finns det tre olika sätt att utföra en empirisk studie på;

- **Undersökning** används för att samla in generell information och data om eller från en stor mängd människor. Vid en undersökning är det viktigt att se till så att urvalet representerar hela populationen av de som ska studeras. De huvudsakliga sätten som används för att samla in information är intervjuer eller enkäter. En slutsats dras sedan från de resultat som undersökningen har visat.

- **Fallstudie** är en observationsstudie som används vid forskning av projekt, aktiviteter eller uppdrag. Under hela studiens gång samlas data in för ett specifikt syfte som det sedan görs statistiska analyser på. En fallstudie använder sig av olika beviskällor för att studera en instans i sin sanna miljö och ger en mer djupgående inblick.

- **Experiment** utförs i en kontrollerad miljö för att studera förhållandena mellan olika variabler och hur de påverkar varandra. Detta kan ske med hjälp av manipulering. Olika behandlingar tillämpas på en variabel medan de andra hålls oförändrade för att kunna mäta effekterna och utifrån resultaten göra statistiska analyser.

Vid experiment i en kontrollerad miljö används ofta kvantitativ forskning som designstrategi (även kallad fast design). Kvantitativa undersökningar passar bra för att testa effekten vid manipulering av variabler då kvantitativa data ger stöd åt jämförelser och statistiska analyser. Vid analysering av kvantitativa data innefattas bland annat beskrivande statistik och hypotestestning. För att få en förståelse för den data som samlats in används bland annat histogram och standardavvikelser (Wohlin et al., 2012).

I ett experiment ligger fokuset på att mäta olika variabler, göra en förändring och sedan mäta igen, vilket leder till att beroende och oberoende variabler kan mätas (Wohlin et al., 2012). I den här studien behöver olika variabler kunna ändras för att sedan mätas på nytt. Den data som samlas in vid mätningarna består av kvantitativa data som sedan kommer att presenters visuellt. Därav har den empiriska metoden experiment med en kvantitativ designstrategi valts ut för den här studien. Experimentet kommer att genomföras på en webbapplikation som skapas med olika JavaScript ramverk. Webbapplikationen kommer att utvecklas med en likadan vy och funktionalitet med målet att varje implementation har samma förutsättningar. Webbapplikationerna kommer att ha en interaktiv tabell som det går att filtrera på för att få fram önskad information. Mätningarna i experimentet kommer att utföras på svarstiden för respektive implementation.

Vidare kommer en litteratursökning att genomföras för att identifiera potentiella JavaScript ramverk till experimentet. Wohlin et al. (2012) tar upp och förklarar tillvägagångssättet systematisk litteraturgenomgång som kan användas vid en empirisk studie. Vidare beskriver Wohlin et al. (2012) att målet vid en systematisk litteraturgenomgång är följande:

> "As it aims to give a complete, comprehensive and valid picture of the existing evidence, both the identification, analysis and interpretation must be conducted in a scientifically and rigorous way."
>
> Wohlin et al., 2012, s. 45

En litteratursökning liknar en systematisk litteraturgenomgång men är inte lika heltäckande. I en litteratursökning är resultatet mer generaliserbart och syftar till att söka upp och identifiera litteratur relaterat till ett ämne.

Ollila et al. (2022) utför en liknande studie där syftet är att mäta prestandaskillnader mellan ramverk med hjälp av en uppsättning riktmärken (engelska 'benchmarks'). Ollila et al. (2022) studie har en annan inriktning än vad den här studien har och i deras artikel tydliggörs det inte huruvida de har använt sig av forskningsfrågor och hypoteser.

3.5 Alternativa metoder

Andra alternativa metoder till studien är undersökningar eller fallstudier som också går att utföra i en empirisk studie.

En undersökning (som beskrivits under rubrik 3.4) riktar in sig på att hämta information från människor i form av intervjuer eller enkäter. Fokuset ligger då på personernas uppfattningar, syn och kunskaper i stället för kontrollerade mätningar eller observationer (Wohlin et al., 2012). Syftet med den här studien är inte att undersöka användningen av ramverken eller hur de uppfattas av användare. Därmed har metoden undersökning uteslutits för den här studien.

En fallstudie (som beskrivits under rubrik 3.4) handlar om att göra observationer utifrån verkliga fall. En fallstudie sker inte i en lika kontrollerad miljö som ett experiment (Wohlin et al., 2012) och valdes bort eftersom studien inte utgår från ett specifikt fall i dess verkliga miljö.

3.6 Tillvägagångssätt

Det första steget är att utföra en litteratursökning för att identifiera potentiella JavaScript ramverk som kan användas för experimentet. Ett urval av de potentiella ramverken kommer att ske baserat på funktioner tagna från tre områden av modellen som Pano et al. (2018) tagit fram (se tabell 1 under kapitel 2.4). Områdena är: ansträngningsförväntning, socialt inflytande och prisvärde och de funktioner som valts ut är: lärbarhet, förståelighet, gemenskapens vidd, gemenskapens lyhördhet samt kostnader. Anledningen till att valet av ramverk baseras på de här fem funktionerna är; de ska vara enkla att lära sig, det ska finnas dokumentation tillgängligt (så att det går att förstå hur de fungerar), de ska ha en tillräckligt stor gemenskap (så att det enkelt går att hitta och få hjälp på internet) och de ska vara gratis att använda.

Nästa steg blir att utveckla olika webbapplikationer för respektive ramverk som valts ut. Grunden kommer att vara uppbyggd med hjälp av HTML och CSS och bestå av en tabell samt ett antal tillhörande checkboxar som kommer att användas för filtreringen. Den grund som skapas kommer att vara den samma för respektive implementation. Därefter läggs JavaScript ramverken till för att skapa interaktivitet mellan checkboxarna och tabellen. Den data som presenteras i tabellen ska uppdateras utefter de parametrar som skickas när en checkbox blir i klickad.

Den data som kommer att användas är ifrån Bolin Centre for Climate Research och representerar historiska väderobservationer i Stockholm (Moberg, 2019). I den här studien kommer endast data för åren 2013–2018 att användas då det är tillräckligt för att generera en större mängd data. Efter att webbapplikationerna har utvecklats kommer ett skript att skapas för att mäta svarstiderna av respektive interaktion som sker vid filtrering av data. En start och stopp klocka kommer att användas för att lagra tiden från att valt antal checkboxar klickas i tills det att tabellen har laddat in all data. Ett flertal mätningar kommer att utföras för att få en mer korrekt uppfattning av det verkliga värdet. Dataserierna kommer att extraheras och sparas ner i en fil som sedan kommer att presenteras visuellt för att kunna jämföras och analyseras. För att kunna besvara frågeställning 2 kommer mätningar att ske på flera olika filtreringsval innehållandes olika stora datamängder. När ett experiment utförs används oberoende och beroende variabler. De variabler som förändras kallas för oberoende variabler och de variabler som studeras för att upptäcka effekten av de förändringarna som gjorts kallas för beroende variabler. Ett experiment består oftast bara av en oberoende variabel och den är

bestående under hela experimentets gång (Wohlin et al., 2012). I detta experiment kommer ramverken och datamängden att vara de oberoende variablerna och svarstiden kommer att vara den beroende variabeln.

3.7 Etiska aspekter

Vid utförandet av ett experiment finns det en del etiska aspekter som behöver beaktas.

Det är viktigt att säkerhetsställa att den insamlade datamängden som används vid de statistiska testerna är tillräckligt stor. Annars finns risken för att en felaktig slutsats dras eftersom det sanna mönstret upptäcks vid statistiska tester. Används en för liten datamängd kan det ske att vissa mönster inte visar sig, vilket kan leda till ett missvisande resultat (Wohlin et al., 2012).

Replikering av ett experiment innebär att det ska gå att återskapa experimentet utifrån liknande förutsättningar. Replikering är en viktig del eftersom det ökar tillförlitligheten av arbetet då experimentet kan göras om av vem som helst för att kontrollera resultatet (Wohlin et al., 2012). All kod som används för experimentet sparas och publiceras på Github för allmänheten att ta del av. Tillvägagångssätt, genomförande och resultat beskrivs och redovisas tydligt, vilket ger möjlighet till att replikera studien och utföra egna experiment.

Datasetet från Bolin Centre for Climate Research som studien kommer att använda sig av finns i deras öppna databas och ligger under licensen Open Data Commons Attribution License (ODC-By) (Moberg, 2019). De har även kontaktats för att säkerhetsställa att deras data får användas och publiceras ihop med det här arbetet. En viktig aspekt att ta hänsyn till är att de verktyg och den data som används för studien har tillåtna rättigheter och inte innehåller känslig information.

Alla mätresultat från experimentet sparas ner och redovisas, finns en stor redundans eller utstickande värden som tydligt visar på felvärden kan de sållas bort (Wohlin et al., 2012). Det görs inga antaganden på värden och resultat, utan de analyseras och motiveras med tydliga grunder.

Wohlin et al. (2012) tar även upp flera etiska aspekter som bör tas i beaktande om en studie involverar människor. Den här studien kommer inte att involvera människors deltagande och kan därmed utesluta vissa etiska aspekter. All text i studien som är hämtad eller inspirerad från andra källor har en korrekt referenshänvisning till respektive författare.

4 Relaterad/tidigare forskning

Det har tidigare gjorts forskning kring liknande områden och i detta kapitel nämns två artiklar som är relevanta för studien.

Diniz-Junior et al. (2022) utvärderar prestandan för olika webbrenderingstekniker som är baserade på JavaScript. I sin artikel skriver Diniz-Junior et al. (2022) att renderingsramverk kan hjälpa till att optimera en applikations prestanda. Studien utvärderade JavaScript ramverken Angular, React och Vue baserat på virtuella och inkrementella DOM-paradigm. Det som analyserades var aspekterna av DOM-manipulationstid, tiden att interagera och byggstorleken. En webbapplikation utvecklades för respektive ramverk.

Ollila et al. (2022) lyfter i sin studie att det har utvecklats en ny generation av UI-ramverk för webbapplikationsutveckling. Renderingsstrategierna som används för den nya generationens UI-ramverk skiljer sig en del från tidigare generationers. Enligt Ollila et al (2022) har ramverkens prestandaegenskaper inte studerats tillräckligt hitintills. I deras undersökning utvärderas ramverken Angular, React, Vue, Svelte och Blazor. Studien fokuserar på att mäta prestandaskillnader mellan ramverken vid skapandet av statiska element, skapandet av komponenter och uppdateringsåtgärder.

5 Genomförande

För att få svar på frågeställningarna, som anges i kapitel 3.1 Frågeställningar, genomförs ett experiment baserat på de delmål och tillvägagångssätt som angetts i tidigare kapitel, 3.3 Delmål och 3.6 Tillvägagångssätt. I detta kapitel beskrivs experimentets genomförande mer detaljerat.

5.1 Litteratursökning

5.1.1 Sökstrategi

Det första steget i sökprocessen är att göra olika sökningar inom det valda området, som i detta fall är JavaScript ramverk. Sökningarna genomförs i några kända databaser som är listade nedan:

- IEEE Xplore
- SpringerLink
- Wiley Online Library
- Google Scholar

Vid sökningarna används olika söktermer för att få fram en variation av JavaScript ramverk. Sökningarna genomförs med en begränsning på årtal. För att mer relevanta och aktuella artiklar ska visas är begränsningen satt från år 2020 och fram till nutid. De söktermer som användes är listade nedan:

- Popular JavaScript frameworks
- JavaScript front-end frameworks
- Modern web frameworks

5.1.2 Litteratur

Några av de mest använda och ledande ramverken är Angular, React, Vue.js och Svelte. Det finns ett flertal moderna ramverk men de här fyra är väldigt populära att använda sig av (Ollila et al., 2022). Marko.js och Qwik är två ramverk som använder sig av en kompilatorbaserad metod. Qwik är ett ganska nykommet ramverk medan Marko.js har varit med i några år. Astro däremot är ett ramverk som använder sig av något som kallas för islands architecture, vilket innebär att webbsidans olika delar kan behandlas separat som olika öar (Vepsäläinen, Hellas & Vuorimaa, 2023). Två andra ramverk som också har varit populära är Ember och Backbone (Ferreira, Borges, Valente, 2021). Angular, React och Vue.js benämns även i artikeln skriven av Vepsäläinen et al. (2023) och i artikeln skriven av Ferreira et al. (2021) som populära ramverk.

5.1.3 Urval av JavaScript ramverk

Urvalet kommer att baseras på de nio ramverk som presenterats i litteraturen (under kapitel 5.1.2). De utvalda ramverken kommer att ställas mot ett antal kriterier för att få fram de som är mest lämpade för studien. Kriterierna kommer att delas upp i två olika uppsättningar och ramverken måste uppfylla villkoren som ställs för respektive kriterium. I uppsättning 1 behöver alla kriterium uppnås för att ett ramverk ska följa med till andra uppsättningen.

Den första uppsättningen av kriterier kommer att utgå från modellen som skapats av Pano et al. (2018). Ur den modellen har fem funktioner valts ut: lärbarhet, förståelighet, gemenskapens vidd, gemenskapens lyhördhet och kostnader.

- **Lärbarhet:** Ramverket ska vara relativt enkelt att förstå och använda utan att man behöver vara en skicklig programmerare.
- **Förståelighet:** Ramverket ska ha tillgänglig dokumentation.
- **Kostnader:** Ramverket ska vara gratis att använda i fullversion.
- **Gemenskapens vidd och lyhördhet:** Ramverket ska ha en tillräckligt stor gemenskap så att det enkelt går att hitta och få hjälp på internet.

Enligt Pano et al. (2018) går gemenskapens vidd och lyhördhet hand i hand och därav har de valts att läggas in under samma kriterium. För att uppfylla de etiska aspekterna ska ramverken vara helt gratis att använda. Det ska finnas dokumentation tillgängligt så att det enkelt går att förstå hur ramverket och dess olika funktioner fungerar. Ramverket ska dessutom vara enkelt att lära sig och förstå sig på, även för någon som är en nybörjare inom programmering. Det ska alltså inte vara alltför komplext. Det sista kriteriet är att ramverket ska ha en bredare gemenskap där det enkelt går att hitta hjälp på internet från andra utvecklare som använder eller har använt sig av ramverket. För att ett ramverk ska vara aktuellt för det här experimentet måste det uppnå alla tre kriterier.

Tabell 2 Uppsättning 1 – kriterier som utgått från modellen gjord av Pano et al. (2018).

JavaScript ramverk	Ska vara gratis att använda i fullversion	Ska finnas dokumentation tillgängligt	Ska vara relativt enkelt att förstå och använda utan att vara en skicklig programmerare	Ska ha en tillräckligt stor gemenskap så att det enkelt går att hitta och få hjälp
Angular	Ja	Ja	Nej	Ja
Astro	Ja	Ja	Ja	Nej
Backbone	Ja	Ja	Ja	Ja
Ember	Ja	Ja	Nej	Ja
Marko.js	Ja	Ja	Nej	Nej
Qwik	Ja	Ja	Ja	Nej
React	Ja	Ja	Ja	Ja
Svelte	Ja	Ja	Ja	Ja
Vue.js	Ja	Ja	Ja	Ja

Den andra uppsättningen består av kriterium som valts ut specifikt för den här studien. Ramverken som väljs ut ska vara populära bland utvecklare idag och skilja sig från varandra när det kommer till användningen av reaktiva system och hanteringen av DOM. Anledningen är att kunna studera närmare hur svarstiderna hos några av de populäraste ramverken idag kan skilja sig från varandra beroende på dess tekniska funktionaliteter. De kriterium som specifikt valt ut till den här studien är:

- **Populärt:** Ramverket ska vara populärt bland utvecklare idag. Huruvida ett ramverk anses vara populärt eller inte baseras på en undersökning gjort av Stack Overflow år 2023 (Stackoverflow, 2023).
- **Reaktivitet:** Ramverket använder sig av reaktivitet vilket innebär att en automatisk uppdatering sker av användargränssnittet när ett tillstånd eller beroende ändras.
- **Virtual DOM:** Ramverket använder sig av virtual DOM som är en lättviktsrepresenation av den riktiga DOM. En virtual DOM används för att kunna uppdatera användargränssnittet utan att det påverkar resten av sidan.

I detta urval behöver inte ramverken uppfylla alla tre kriterium. Meningen med urvalet är att få fram om ett ramverk använder sig av både virtual DOM och reaktivitet, använder sig av ett av dem eller inget av dem. Däremot måste kriteriet ”populärt bland utvecklare” uppfyllas för att ramverket fortfarande ska vara aktuellt för studien.

Tabell 3 Uppsättning 2 – specifika kriterier utvalda för den här studien.

JavaScript ramverk	Populärt bland utvecklare	Använder sig av reaktivitet	Använder sig av virtual DOM
Backbone	Nej	Nej	Nej
React	Ja	Nej	Ja
Svelte	Ja	Ja	Nej
Vue.js	Ja	Ja	Ja

I undersökningen av Stack Overflow som gjordes år 2023 fanns ramverken React, Vue.js, Svelte med på listan bland de webbramverk och teknologier som är de mest populära (Stackoverflow, 2023). React, Vue.js och Svelte har både likheter och olikheter vad gäller deras tekniska funktionaliteter. Därav har alla tre ramverk valts ut för studien eftersom det är intressant att undersöka om ramverkens skillnader kan ha en påverkan över deras svarstider.

Tabell 4 – De tre utvalda JavaScript ramverken för studiens experiment.

Valda JavaScript ramverk för experimentet
React
Vue.js
Svelte

5.2 Implementation

Första steget i implementationsfasen är att installera ner de valda JavaScript ramverken. För att kunna göra de installationer som krävs via terminalen behövs Node.js pakethanterare. Därav kommer även Node.js att installeras och användas i arbetet. Alla ramverk kommer att installeras ihop med Vite som är ett front-end verktyg och respektive version av ramverken listas i tabell 5 nedan. Vid installationen av ramverken tillkommer det en färdig struktur med filer som kommer att följas och byggas på med egna komponenter. I implementationen för Vue.js och Svelte används JS filer och i React används JSX filer. Anledningen till att JS filer inte används i implementationen med React är att JSX filer skapas automatiskt vid installering av React ihop med Vite.

Tabell 5 Versioner som installerats.

Ramverk	Version
Vue.js	3.4.21
React	18.2
Svelte	4.2.12

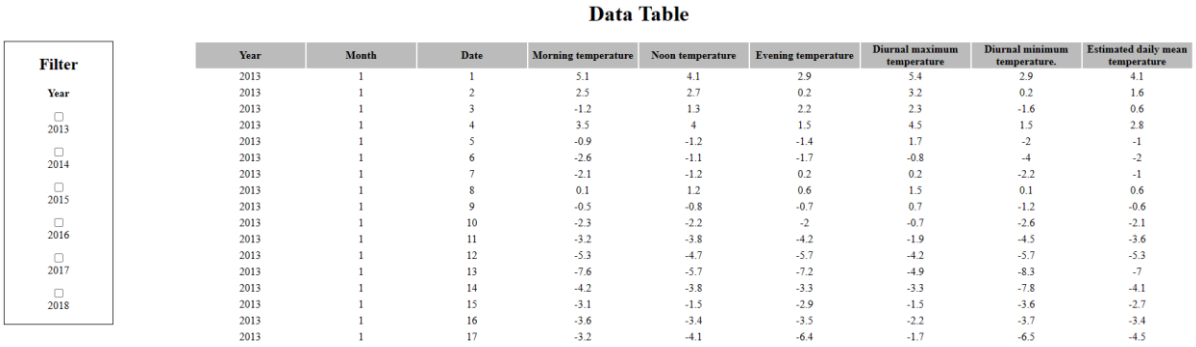
Applikationerna kommer att följa samma struktur och ha samma funktionalitet och vy. I respektive implementation kommer kod att läggas till i tre olika filer. En av filerna innehåller endast CSS. De två filer som kommer att användas för att skapa applikationernas funktionalitet heter ”App” och ”DataTable” i samtliga implementationer. All kod som används i detta arbete finns tillgängligt i ett repository[1] på Github.com. Den data som använts i arbetet låg först lagrad i en text-fil när den laddades ner. Textfilen har konverterats om till en JSON-fil för att underlätta utvecklingen då det är lättare att använda sig av data i JSON format. JSON-filen har totalt 24 103 rader kod som innehåller data av historiska väderobservationer mellan åren 2013–2018 och samma fil har använts i alla tre applikationer. I figur 7 visas ett objekt från JSON-filen för att demonstrera hur filens uppbyggnad och struktur ser ut.

[1] https://github.com/g21emmka/Examensarbete

```
{
      "year": "2013",
      "month": "1",
      "date": "1",
      "t1": "5.1",
      "t2": "4.1",
      "t3": "2.9",
      "tx": "5.4",
      "tn": "2.9",
      "tm": "4.1"
},
```


Figur 7 StockholmHistoricalWeatherObservations.json

Samma vy har skapats för alla tre implementationer av applikationen. Vyn innehåller en tabell med data och på vänster sida finns 6 checkboxar som används för att kunna filtrera tabellens data efter år. I figur 8 nedan visas en skärmbild på hur webbapplikationen ser ut. Fokus i det här arbete är mätning av svarstider vid filtrering, inte designen hos applikationen. Därav har endast en enklare styling lagts till för att skapa en struktur i applikationen. Det ska vara enkelt att se och förstå vad som visas upp.



Figur 8 Skärmbild från webbapplikationen på dess design

5.2.1 Hämtning av data

Data från JSON filen hämtas i ”App” som är applikationens förälder komponent. Hur data hämtas och sparas ner skiljer sig lite mellan de olika implementationerna. I Vue.js (se figur 9) hämtas data med *onMounted* och läggs sedan in i en *ref array*. I React (se figur 10) används *useEffect* för att hämta data, vilket är en hook som berättar för React att komponenten måste göra något efter rendering. Data som hämtats läggs sedan in i en array med *useState*, som gör det möjligt att lagra och uppdatera data. Hur data hämtas i Svelte (se figur 11) liknar tillvägagångssättet som användes i Vue.js implementationen. Skillnaden är att en vanlig array används. Anledningen till att all data sparas ner i en array är för att den ska kunna användas till andra funktioner.

```
const weatherData = ref([]);
onMounted(async () => {
  const response = await fetch("/StockholmHistoricalWeatherObservations.json");
  weatherData.value = await response.json();
});
```

Figur 9 Hämtar data från JSON-filen i Vue.js

```
const [weatherData, setWeatherData] = useState([]);
 useEffect(() => {
   fetch("./StockholmHistoricalWeatherObservations.json")
     .then((res) => res.json())
     .then((data) => {
       setWeatherData(data);
     });
 }, []);
```

Figur 10 Hämtar data från JSON-filen i React

```
let weatherData = [];
onMount(async () => {
    const res = await fetch('./StockholmHistoricalWeatherObservations.json');
    weatherData = await res.json();
  })
```

Figur 11 Hämtar data från JSON-filen i Svelte

5.2.2 Skriva ut data i tabellform

Den data som hämtats i ”App” komponenten skickas vidare till barnkomponenten ”DataTable” som tar emot all data med hjälp av props. För att skriva ut data i en HTML tabell används olika funktioner för respektive implementation. I Vue.js används Vue´s egna variant av for-loop som heter *v-for* (se figur 12). I React används funktionen *map()* (se figur 13) och i Svelte används *each block* (se figur 14). Alla tre funktioner används för att rendera en lista med element så att varje innehåll i ett objekt kan visas upp.

```
<tbody>
    <tr v-for="data in filterData" >
         <td>{{ data.year }}</td>
         <td>{{ data.month}} </td>
         <td>{{ data.date }}</td>
         <td>{{ data.t1 }}</td>
         <td>{{ data.t2 }}</td>
         <td>{{ data.t3 }}</td>
         <td>{{ data.tx }}</td>
         <td>{{ data.tn }}</td>
         <td>{{ data.tm }}</td>
    </tr>
</tbody>
```

Figur 12 Tabell med data i Vue.js

```
{filterData.map((data) => {
     return (
     <tr>
         <td>{data.year}</td>
         <td>{data.month}</td>
         <td>{data.date}</td>
         <td>{data.t1}</td>
         <td>{data.t2}</td>
         <td>{data.t3}</td>
         <td>{data.tx}</td>
         <td>{data.tn}</td>
         <td>{data.tm}</td>
     </tr>
   );
})}
```

Figur 13 Tabell med data i React

```
{#each filterData as data}
<tr>
   <td >{data.year}</td>
   <td >{data.month}</td>
   <td >{data.date}</td>
   <td >{data.t1}</td>
   <td >{data.t2}</td>
   <td >{data.t3}</td>
   <td >{data.tx}</td>
   <td >{data.tn}</td>
   <td >{data.tm}</td>
<tr>
{/each}
```

Figur 14 Tabell med data i Svelte

5.2.3 Få ut värde från checkbox

För att kunna få ut värdet från respektive checkbox har funktionen ”handleCheckboxFilter” skapats. Funktionen är kopplad till en händelsehanterare för att kunna hantera vad som ska ske när en checkbox blir i klickad. När en ändring sker i en checkbox (klickas i eller klickas ur) kommer värdet antingen att läggas till eller tas bort i arrayen ”dataFilter”. Hur funktionen har skapats i implementationen med Vue.js (se figur 15) skiljer sig från hur den skapats i React (se figur 16) och Svelte (se figur 17), vilka är väldigt likartade.

```
function handleCheckboxFilter(e) {
    const checked = e.target.value;
    if (dataFilter.value.includes(checked)) {
           dataFilter.value.splice(dataFilter.value.indexOf(checked), 1);
    } else{
        dataFilter.value.push(checked);
    }
};
```

Figur 15 Funktionen handleCheckboxFilter i Vue.js

```
function handleCheckboxFilter(e) {
   const {value, checked} = e.target;
   if(!dataFilter.includes(value) && checked){
      setDataFilter([...dataFilter,value])
   } else {
      setDataFilter(dataFilter.filter(f=>f!==value))
   }
};
```

Figur 16 Funktionen handleCheckboxFilter i React

```
function handleCheckboxFilter(e) {
    const {value, checked} = e.target;
    if(!dataFilter.includes(value) && checked){
        dataFilter = ([...dataFilter,value])
    } else {
        dataFilter = (dataFilter.filter(f=>f!==value))
    }
};
```

Figur 17 Funktionen handleCheckboxFilter i Svelte

5.2.4 Uppdatera tabellen vid data filtrering

HTML tabellen som skapats för applikationen ska kunna uppdatera sin data utifrån vilken filtrering som valts med hjälp av checkboxarna. Även här skiljer sig funktionen i Vue.js implementationen sig mest från hur funktionen är skapad i implementationerna React och Svelte. I Vue.js (se figur 18) används egenskapen *computed* som automatiskt spårar dess reaktiva beroenden och uppdaterar automatisk när ett beroende ändras. I denna funktion uppdateras ”filterData”, som är kopplad till HTML tabellen, om värdena i arrayen dataFilter ändras. I React (se figur 19) har const filterData konstruerats annorlunda och innehåller tecknen såsom frågetecken och kolon. De här tecknen innebär att om tillståndet är sant, i detta fall om en checkbox är i klickad, så ska vald data visas annars presenteras all data från weatherData arrayen. I Svelte (se figur 20) skapas funktionen på samma sätt som i React fast med en *$:* som är en reaktiv deklaration.

```
const filterData = computed ( () => {
   let weatherData = props.weatherData;
    if (dataFilter.value.length){
        weatherData = weatherData.filter(data =>
        dataFilter.value.includes(data.year));
    }
   return weatherData;
});
```

Figur 18 filterData Vue.js

```
const filterData = dataFilter.length > 0 ? weatherData.filter(data  =>
dataFilter.includes(data.year)) : weatherData;
```

Figur 19 filterData i React

```
$:filterData = dataFilter.length > 0 ? weatherData.filter(data  =>
dataFilter.includes(data.year)) : weatherData;
```

Figur 20 filterData i Svelte

5.2.5 Mätskript

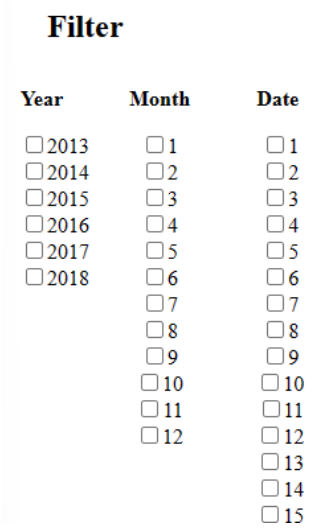
För att kunna mäta svarstiderna för respektive ramverk har ett automatiserat skript skapats. Skriptet har byggts i verktyget Tampermonkey, som är ett tillägg i webbläsaren Google Chrome. För att kunna göra flera mätningar i rad används en for-loop som kör skriptet lika många gånger som variabeln counter är satt till. Inom for-loopen används syntaxen await för funktionerna, vilket betyder att skriptet kommer att vänta på svar innan det går vidare. En starttid har satts innan funktionen som klickar i checkboxar körs och därefter läggs en stopptid in, se figur 21. På så vis kan svarstiden vid filtrering av data tas fram. Både tiden och de år som valts vid filtreringen lagras och sparas ner i en extern fil när skriptet körts klart. För respektive applikation kommer skriptet att filtrera data på ett varierat antal checkboxar. Olika antal checkboxar används för att se om det finns någon skillnad i svarstider hos ramverken vid filtrering av varierande datamängder.

```
async function script() {
        for (let i = 0; i < counter; i++) {
            await new Promise((resolve) => {
                setTimeout(async () => {
                    await uncheckCheckboxes();
                    await loadPage();

                    let startTime = performance.timeOrigin + performance.now();
                    await selectCheckboxesRandomly();
                    let endTime = performance.timeOrigin + performance.now();

                    let result = endTime - startTime;
                    var object = {Years: checkedYears, Time: result};
                    filtration.push(object);
                    checkedYears = '';
                    if (i === counter - 1) {
                        saveToTxtFile();
                    }
                    resolve();
                }, 1000);
            });
        }
    }
```

Figur 21 Kod från filter-script

5.3 Progression

Den första implementationen som skapades var med ramverket Vue.js. Till en början var tanken att det skulle finnas checkboxar för flera kolumner. Meningen var att ha flera filtreringsval utspridda över flera kolumner. Se exempel på hur det hade kunnat se ut i figur 22.



Figur 22 Filtreringsval med checkboxar för flera kolumner

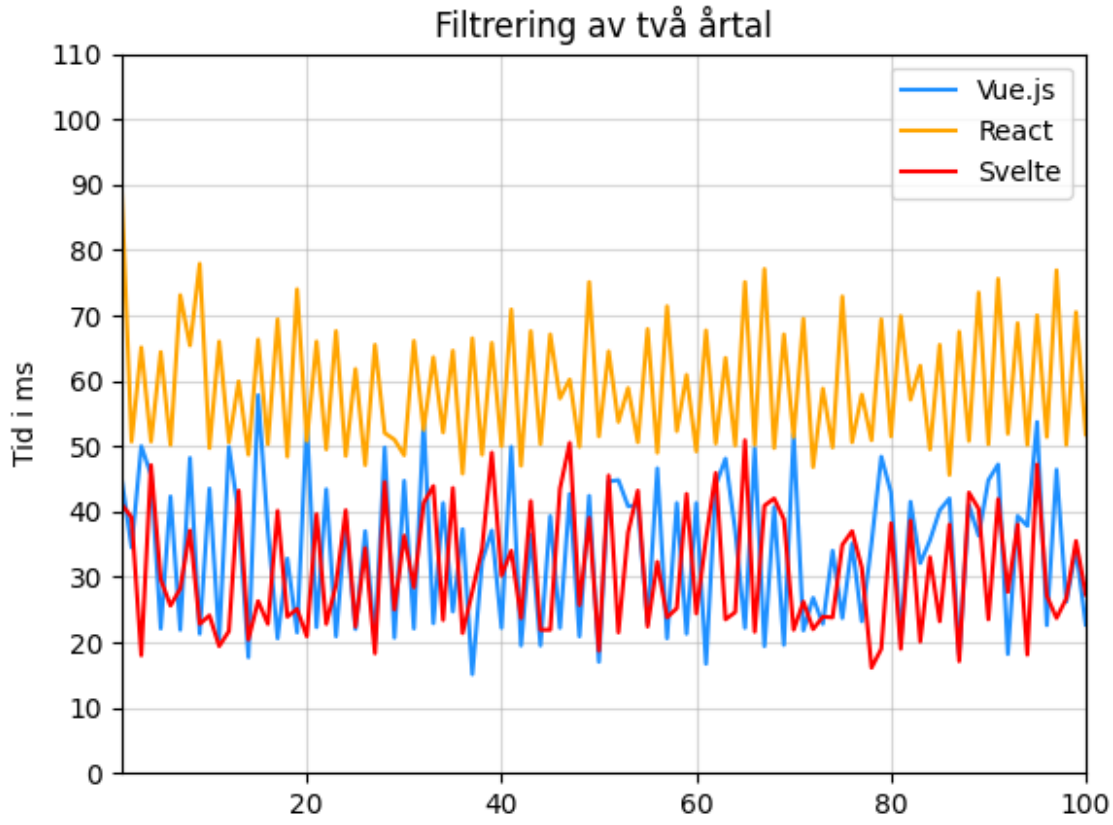
Ursprungstanken var att använda många olika filtreringsval fördelade över tre kolumner, vilket gjorde experimentet alldeles för komplext. Alternativet blev att använda endast en kolumn för filtrering, se figur 23. Kolumnen som valdes ut var den för årtalen. Datamängden för respektive år är tillräckligt stor för att kunna genomföra en varians av mätningar med olika datamängder.

Figur 23 Filtreringsval med checkboxar från en kolumn

5.4 Pilotstudie

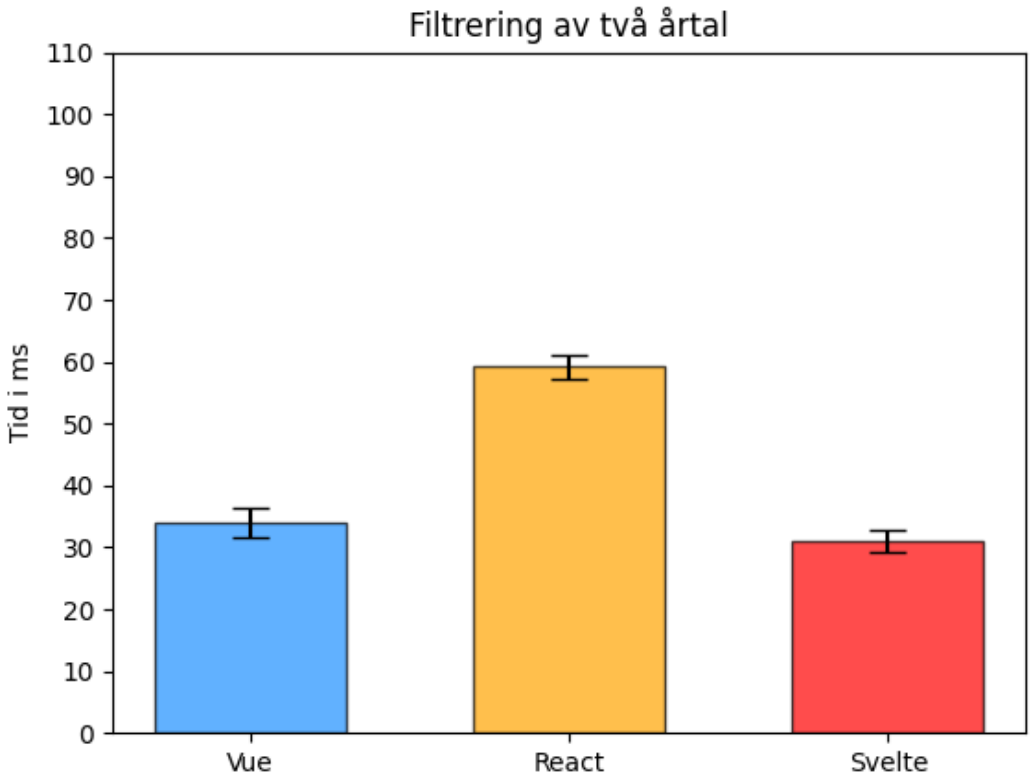
En pilotstudie utförs innan mätningarna för studien kan genomföras. Pilotstudien är till för att upptäcka och utvärdera om mätningarna kan användas och jämföras med varandra. Vilket sker genom att säkerställa potentiella mätfel eller andra faktorer som kan påverka resultatet. För mätserien i pilotstudien behövs inte ett lika stort antal mätningar genomföras som i den faktiska studien. För att bedöma om resultatet kan användas kommer 100 mätningar att utföras för respektive ramverk och endast vid filtrering av en mindre datamängd.

Resultatet från linjediagrammet som presenteras i figur 24 påvisar att det finns en skillnad mellan ramverken. Dock ligger mätpunkterna för Vue.js och Svelte på samma nivå medan mätpunkterna för React ligger lite högre.



Figur 24 Linjediagram vid filtrering av två årtal

Resultatet presenteras även i ett stapeldiagram med konfidentintervall som presenteras i figur 25. Vue.js och Svelte ligger väldigt nära varandra i svarstider och det ser ut som att de kan finnas överlappning mellan de två ramverken.



Figur 25 Stapeldiagram med konfidentintervall vid filtrering av två årtal

Eftersom det kan finnas en överlappning så genomfördes även ett ANOVA test (en faktor) för att se om de finns en statistisk skillnad. Resultatet från ANOVA testet presenteras i figur 26 och det visar på att det finns en statistisk signifikant skillnad mellan ramverken. Testet gav ett p-värde på 2,8259E-60.

SAMMANFATTNING

Grupper	*Antal*	*Summa*	*Medelvärde*	*Varians*
Vue	100	3393,699951	33,93699951	133,5579279
Svelte	100	3097,900391	30,97900391	88,27013119
React	100	5918,600342	59,18600342	98,05743431

ANOVA

Variationsursprung	*KvS*	*fg*	*MKv*	*F*	*p-värde*	*F-krit*
Mellan grupper	48063,22526	2	24031,61263	225,3770158	2,8259E-60	3,026153369
Inom grupper	31668,66385	297	106,6284978			
Totalt	79731,8891	299				

Figur 26 ANOVA-test

För att undersöka vidare genomfördes även ett tukey-test som presenteras i figur 27. Detta på grund av att ANOVA testet visar på att det finns en statistisk skillnad. Eftersom det är mer än två ramverk som jämförs med varandra krävs ett tukey-test för att kunna jämföra ramverken parvis. Tukey-testet visar på att det finns en signifikant skillnad mellan React och Svelte samt mellan React och Vue.js. Det visar dock att det inte finns någon signifikant skillnad mellan Vue.js och Svelte.

TUKEY						
Grupp1	*Grupp2*	*Medeldiff*	*p-adj*	*Lägre*	*Övre*	*Avvisa*
React	Svelte	-28,207	0	-31,6468	-24,7672	SANT
React	Vue	-25,249	0	-28,6889	-21,8092	SANT
Vue	Svelte	2,958	0,1079	-0,4819	6,3978	FALSKT

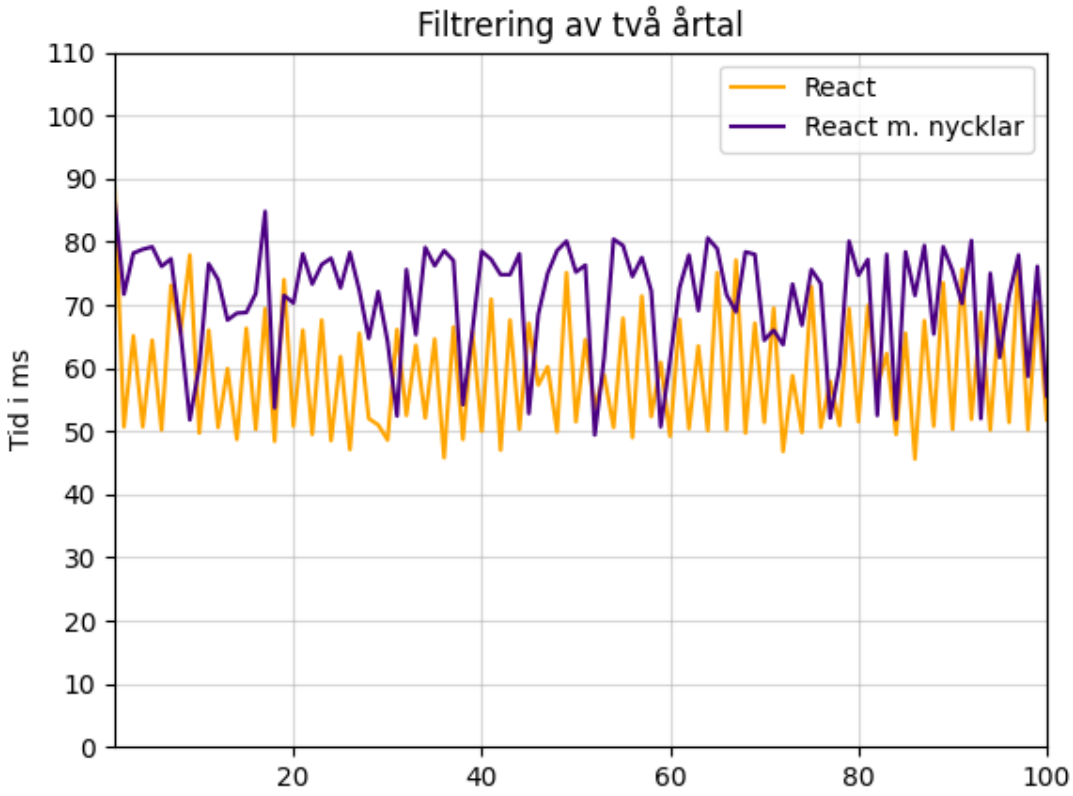
Figur 27 Tukey-test flera jämförelser av medelvärdet

5.4.1 Diskussion

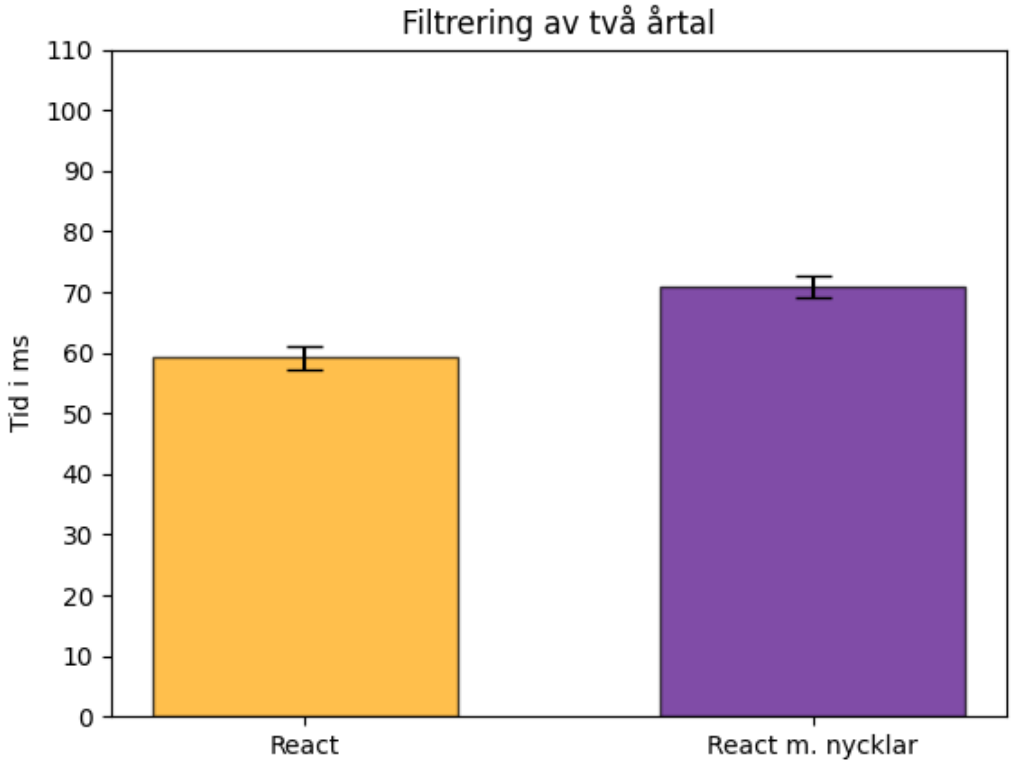
Under genomförandet av pilotstudien var det ett fåtal saker som uppmärksammades.

Vid mätningarna upptäcktes det att random seed som används för att välja slumpmässiga checkboxar inte tog samma ordning i React implementationen som i de andra två implementationerna. Funktionen i sig fungerar som den ska i React och tar alltid samma ordning av checkboxar vid flera mätningar. Efter att ha undersökt random seed funktionen närmare visade det sig att ordningen blev annorlunda på grund av att React ihop med Vite använder sig av JSX filer. JSX är ett syntaxtillägg för JavaScript som gör det möjligt att använda sig av HTML-liknande markup i en JavaScript fil (React, 2024c). JSX är till för att underlätta utveckling och det finns ingenting som påvisar att de ska påverka svarstiderna på något sätt. Eftersom ett flertal mätningar kommer att utföras anses det inte att denna faktor kommer påverka experimentet negativt. Den enda skillnaden är att ordningen som checkboxarna väljs utifrån är annorlunda i applikationen skapad med React jämfört med applikationerna skapade med Vue.js och Svelte. Anledningen till att samtliga applikationer skapades med Vite är för att Vue CLI är i underhållsläge förtillfället och inte går att installera. Att Vue CLI inte går att installera i dagsläget upptäcktes under implementationsfasen och därav installerades alla ramverk med Vite så att ramverken byggs med samma grund.

En annan upptäckt som gjordes var att React vill att den data som ska användas och skrivas ut ska ha unika nycklar. Om inga unika nycklar finns dyker det upp en varning i konsolen som säger att unika nycklar bör finnas. Den data som används för implementationerna har inget i sin fil som unikt kan identifiera olika objekt. Eftersom React ger en varning för att unika nycklar inte finns kan det vara bra att lägga till unika nycklar för varje objekt. För att se om de ger någon skillnad i svarstider med unika nycklar så installerades paketet uuid ner. Uuid genererar unika id:n av kryptografiska slumpmässiga värden. I figur 29 visas resultatet från 100 mätningar med unika nycklar genererade med uuid samt utan unika nycklar i React implementationen. Mätningarna visar på att den finns en viss skillnad. I figur 30 visas resultatet från mätningarna upp i ett stapeldiagram med konfidentintervall. I stapeldiagrammet går det tydligare att se att det finns en viss skillnad och svarstiderna kan bli lite längre vid användandet av unika nycklar. I pilotstudien har React implementationen utan unika nycklar använts.



Figur 28 Linjediagram för React applikationen med och utan unika nycklar.



Figur 29 Stapeldiagram med konfidentintervall för React applikationen med och utan unika nycklar.

6 Utvärdering

Pilotstudien visade på ett resultat som säger att studien är genomförbar.

6.1 Förändringar från pilotstudien

Efter genomförande av pilotstudien uppmärksammades ett fåtal saker som är beskrivet under kapitel 5.4.1 Diskussion. JSON-filen med data innehöll inte något som unikt kunde identifiera de olika objekten. Uuid testades, som genererar unika nycklar. Det som framkom var att de olika implementationerna hade fått olika nycklar för objekten om uuid använts. Eftersom uuid skapar och genererar nycklar efter att data har hämtats. Därav valdes alternativet att generera id:n till alla objekt i JSON-filen[2]. På så sätt har objekten samma unika identifierare i samtliga implementationer. Genereringen av id:n skapades med hjälp av JavaScript där en for loop användes för att tilldela id:n till respektive objekt. Ett objekt från den JSON-filen med ett tilldelat id visas i figur 30.

```
  {
     “id”: 1,
     “year”: “2013”,
     “month”: “1”,
     “date”: “1”,
     “t1”: “5.1”,
     «t2»: «4.1»,
     «t3»: «2.9»,
     «tx»: «5.4»,
     «tn»: «2.9»,
     «tm»: «4.1»
   }
```

Figur 30 JSON-fil med id

6.2 Presentation av undersökning

Vid utförandet av experimentet sker mätningar först på en mindre datamängd bestående av endast ett filtreringsval. Därefter trappas datamängden upp med ytterligare ett filtreringsval för varje mätserie. Totalt kommer fem mätserier att genomföras med fem olika filtreringsval för att få fram hur svarstiderna hos ramverken skalar vid olika datamängder. Mätningarna kommer att utföras i webbläsaren Google Chrome och i tabell 6 visas en del specifikationer för enheten som används.

Tabell 6 Specifikationer för den enhet som används

Operativsystem:	Windows 11 Home version 23H2
Processor:	Intel™ Core ™ i7-1165G7 @ 2.80GHz
Ram:	16,0 GB

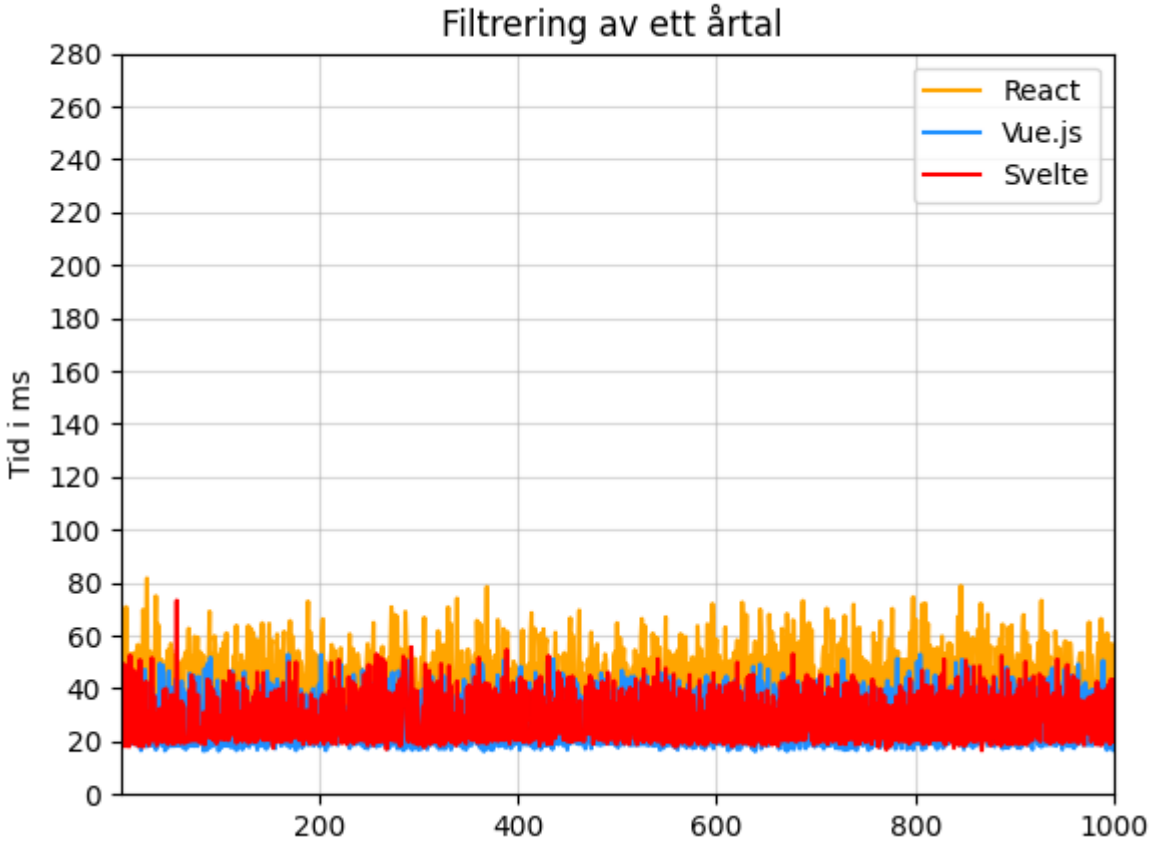
[2] https://github.com/g21emmka/Examensarbete/commit/a60294f07b2f89969d736f8c535c8f4583cd148f

6.3 Analys

Mätningarna genomfördes med hjälp av mätskriptet som beskrivits i kapitel 5.2.5 Mätskript. Vid varje mätserie har 1000 mätningar genomförts för respektive ramverk. För att undersöka om det finns några signifikanta skillnader i svarstider vid olika datamängder har fem mätserier genomförts med olika filtreringsval. Filtreringsvalen består av årtal och respektive årtal innehåller 365 rader data, förutom år 2016 som innehåller 366 rader data. Resultatet från samtliga mätserier som utförts kommer att presenteras med hjälp av linjediagram, ANOVA-test och tukey-test. Ett linjediagram och boxplot diagram över alla mätserier kommer att presenteras i kapitel 6.3.6 Skalbarhet. Diagrammen ger en bättre överblick över hur svarstiderna responderar hos ramverken vid olika datamängder.

6.3.1 Mätserie 1

Resultatet av första mätserien med filtrering av ett årtal visas i figur 31.



Figur 31 Linjediagram för filtrering av ett årtal

Ett ANOVA-test genomförs för att se om det finns någon överlappning. Testet visar att det finns en statistisk signifikant skillnad mellan ramverken med p-värde 7,5509E-178, se figur 35.

SAMMANFATTNING

Grupper	*Antal*	*Summa*	*Medelvärde*	*Varians*
Vue	1000	30339,9978	30,3399978	107,7070391
React	1000	43172,99902	43,17299902	185,1059445
Svelte	1000	28940,19653	28,94019653	101,0320437

ANOVA

Variationsursprung	*KvS*	*fg*	*MKv*	*F*	*p-värde*	*F-krit*
Mellan grupper	123072,6769	2	61536,33844	468,7351687	7,5509E-178	2,998728735
Inom grupper	393451,1823	2997	131,2816758			
Totalt	516523,8592	2999				

Figur 32 ANOVA-test för filtrering av ett årtal

För att kunna jämföra ramverken parvis med varandra utförs även ett tukey-test. Testet visar att det finns skillnader i svarstider mellan samtliga ramverk, se figur 33.

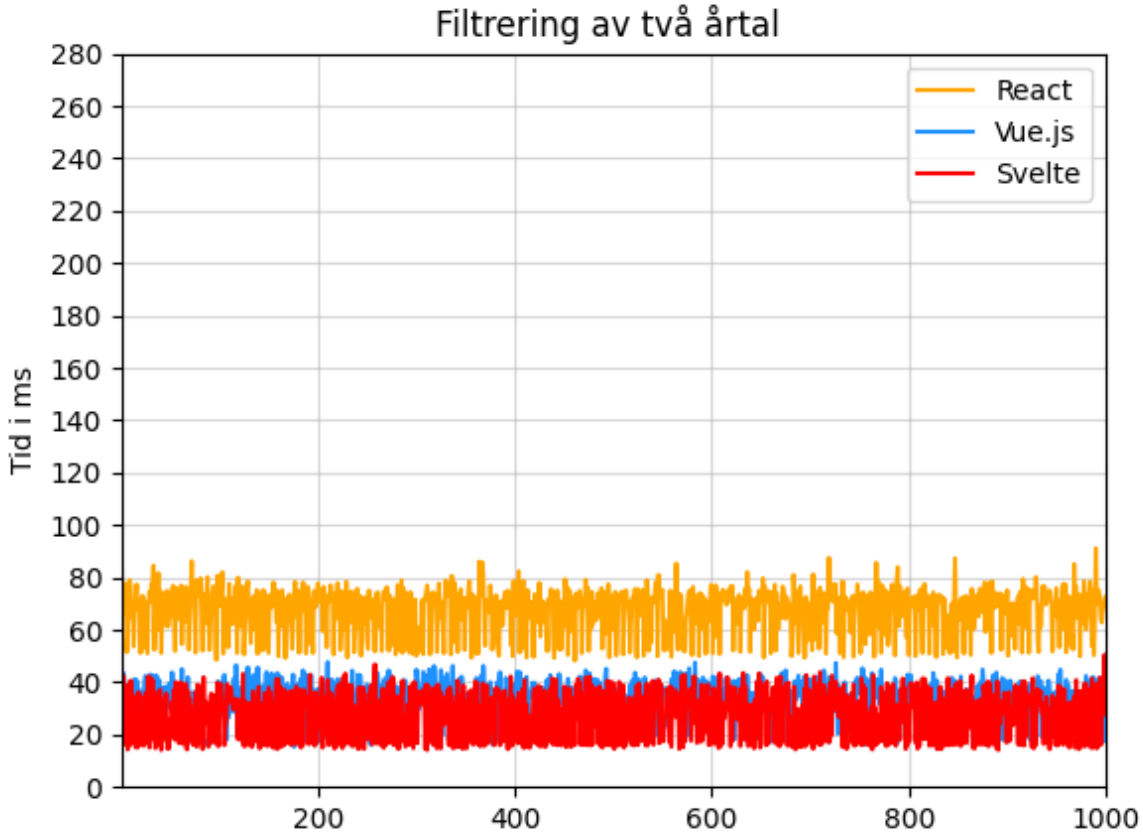
TUKEY

Grupp1	*Grupp2*	*Medeldiff*	*p-adj*	*Lägre*	*Övre*	*Avvisa*
React	Svelte	-14,2328	0	-15,4343	-13,0313	SANT
React	Vue	-12,833	0	-14,0345	-11,6315	SANT
Svelte	Vue	1.3998	0,0174	0.1983	2,6013	SANT

Figur 33 Tukey-test flera jämförelser av medelvärdet vid filtrering av ett årtal

6.3.2 Mätserie 2

Resultatet av andra mätserien med filtrering av två årtal visas i figur 34.



Figur 34 Linjediagram för filtrering av två årtal

Ett ANOVA-test genomförs för att se om det finns någon överlappning. Testet visar att det finns en statistisk signifikant skillnad mellan ramverken med p-värde 0, se figur 35.

SAMMANFATTNING

Grupper	*Antal*	*Summa*	*Medelvärde*	*Varians*
Vue	1000	33185,60229	33,18560229	56,57324394
React	1000	67674,80396	67,67480396	74,20895365
Svelte	1000	27562,90088	27,56290088	87,47502543

ANOVA

Variationsursprung	*KvS*	*fg*	*MKv*	*F*	*p-värde*	*F-krit*
Mellan grupper	943361,5236	2	471680,7618	6483,369786	0	2,998728735
Inom grupper	218038,9658	2997	72,75240767			
Totalt	1161400,489	2999				

Figur 35 ANOVA-test för filtrering av två årtal

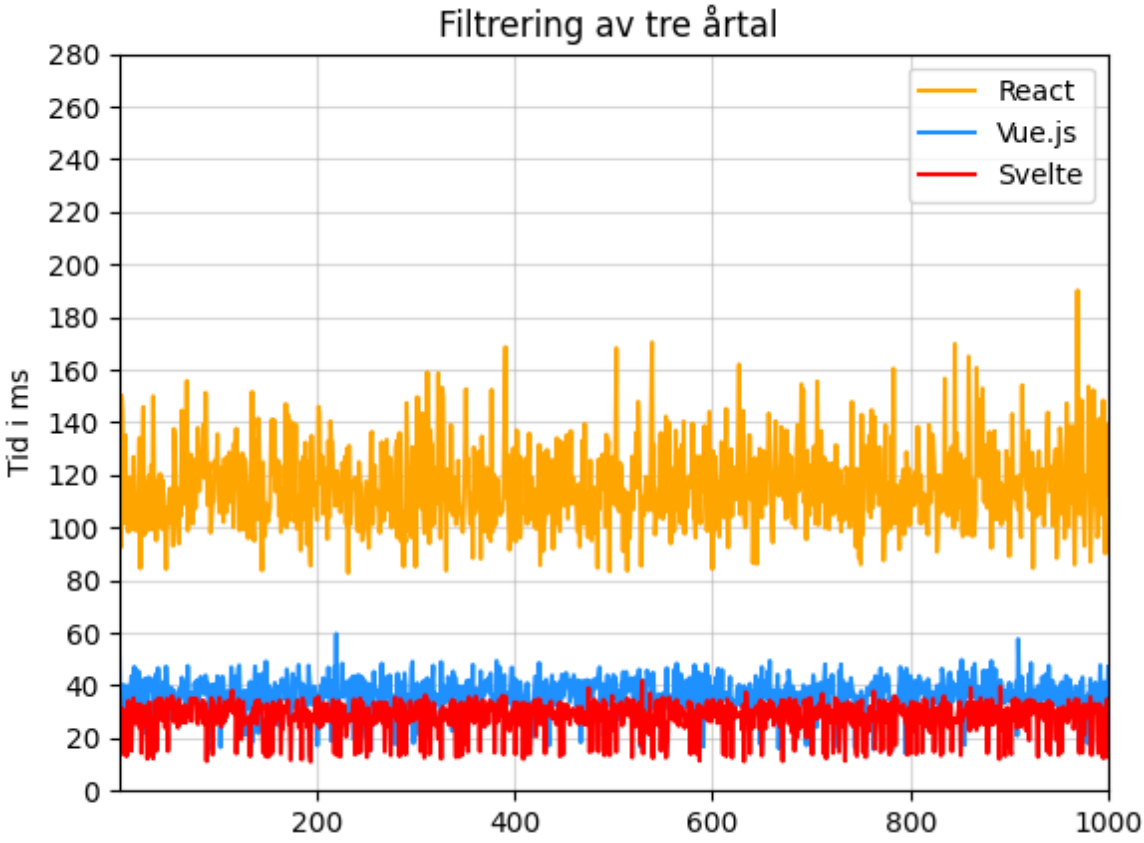
För att jämföra ramverken parvis genomförs ett tukey-test. Testet visar på att det finns skillnader i svarstider mellan samtliga ramverk, se figur 36.

TUKEY						
Grupp1	*Grupp2*	*Medeldiff*	*p-adj*	*Lägre*	*Övre*	*Avvisa*
React	Svelte	-40,1119	0	-41,0064	-39,2175	SANT
React	Vue	-34,4892	0	-35,3837	-33,5947	SANT
Svelte	Vue	5,6227	0	4,7282	6,5172	SANT

Figur 36 Tukey-test flera jämförelser av medelvärdet vid filtrering av två årtal

6.3.3 Mätserie 3

Resultatet av tredje mätserien med filtrering av tre årtal visas i figur 37.



Figur 37 Linjediagram för filtrering av tre årtal

Ett ANOVA-test genomförs för att se om det finns någon överlappning. Testet visar att det finns en statistisk signifikant skillnad mellan ramverken med p-värde 0, se figur 38.

SAMMANFATTNING

Grupper	*Antal*	*Summa*	*Medelvärde*	*Varians*
Vue	1000	36327,89868	36,32789868	44,40433771
React	1000	115431,0984	115,4310984	237,2190559
Svelte	1000	27916,59961	27,91659961	39,90138144

ANOVA

Variationsursprung	*KvS*	*fg*	*MKv*	*F*	*p-värde*	*F-krit*
Mellan grupper	4662284,551	2	2331142,275	21750,81788	0	2,998728735
Inom grupper	321203,2503	2997	107,174925			
Totalt	4983487,801	2999				

Figur 38 ANOVA-test för filtrering av tre årtal

För att jämföra ramverken parvis genomförs ett tukey-test. Testet visar på att det finns skillnader i svarstider mellan samtliga ramverk, se figur 39.

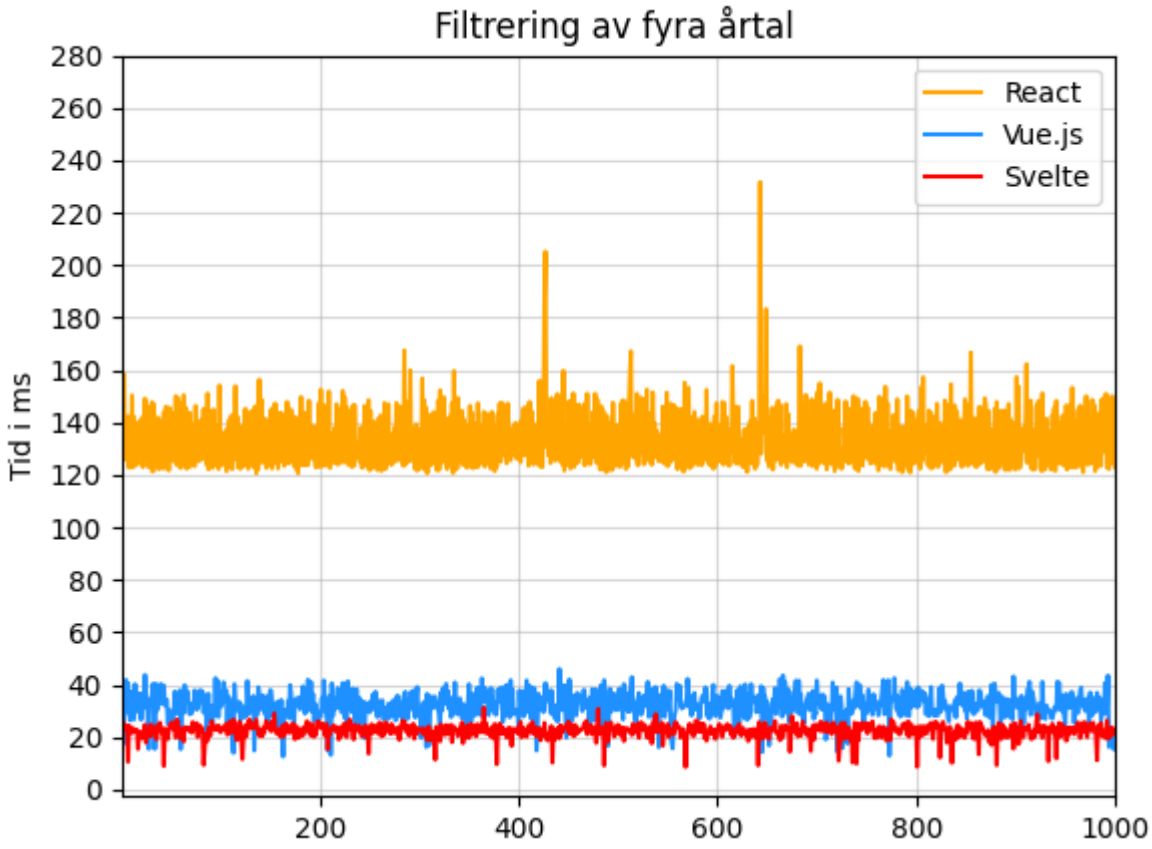
TUKEY

Grupp1	*Grupp2*	*Medeldiff*	*p-adj*	*Lägre*	*Övre*	*Avvisa*
React	Svelte	-87,5145	0	-88,6001	-86,4289	SANT
React	Vue	-79,1032	0	-80,1888	-78,0176	SANT
Svelte	Vue	8,4113	0	7,3257	9,4969	SANT

Figur 39 Tukey-test flera jämförelser av medelvärdet vid filtrering av tre årtal

6.3.4 Mätserie 4

Resultatet av fjärde mätserien med filtrering av fyra årtal visas i figur 40.



Figur 40 Linjediagram för filtrering av fyra årtal

Spikarna som syns i linjediagrammet, se figur 40, för React tyder på en slumpmässig varians. Ett ANOVA-test genomförs för att se om det finns någon överlappning. Testet visar att det finns en statistisk signifikant skillnad mellan ramverken med p-värde 0, se figur 41.

SAMMANFATTNING

Grupper	*Antal*	*Summa*	*Medelvärde*	*Varians*
Vue	1000	32336,00244	32,33600244	26,51045197
React	1000	134198,2021	134,1982021	125,5162668
Svelte	1000	22475,89722	22,47589722	7,387812327

ANOVA

Variationsursprung	*KvS*	*fg*	*MKv*	*F*	*p-värde*	*F-krit*
Mellan grupper	7651667,608	2	3825833,804	71997,83691	0	2,998728735
Inom grupper	159255,1166	2997	53,13817704			
Totalt	7810922,724	2999				

Figur 41 ANOVA-test för filtrering av fyra årtal

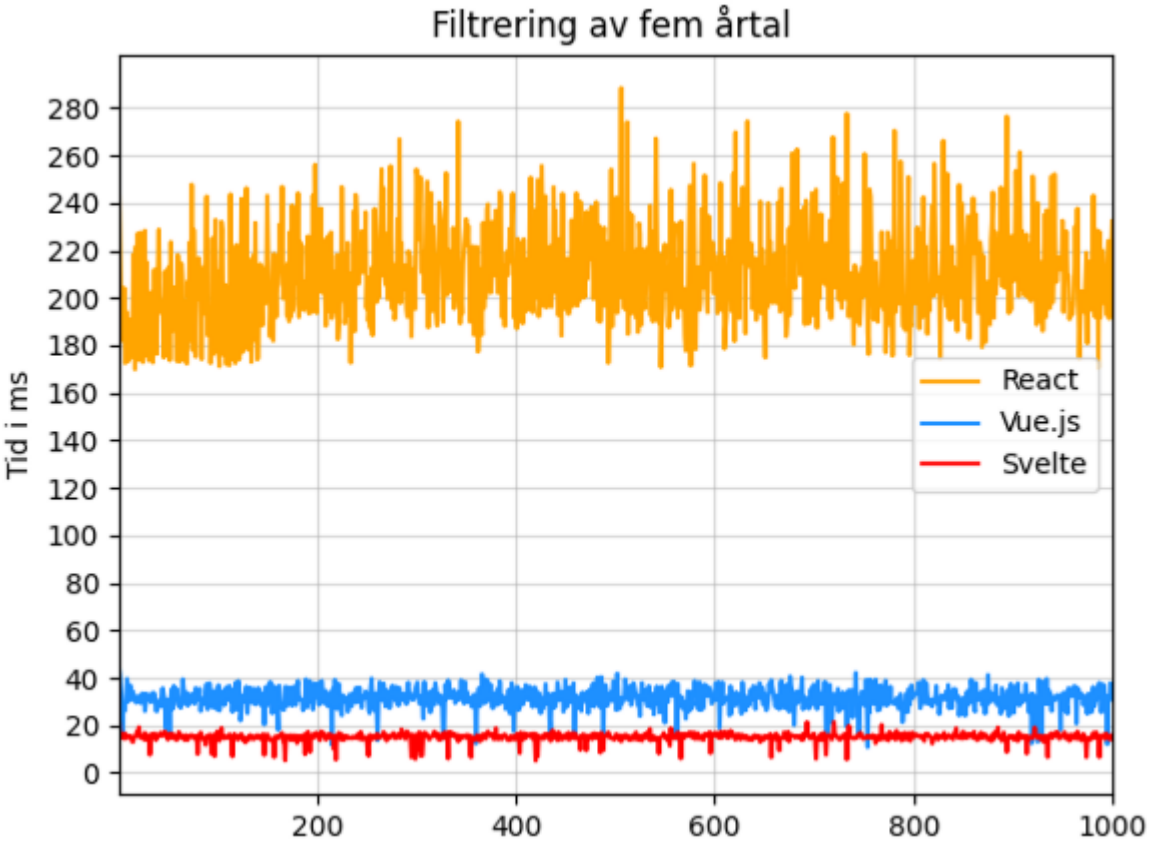
För att jämföra ramverken parvis genomförs ett tukey-test. Testet visar på att det finns skillnader i svarstider mellan samtliga ramverk, se figur 42.

TUKEY						
Grupp1	*Grupp2*	*Medeldiff*	*p-adj*	*Lägre*	*Övre*	*Avvisa*
React	Svelte	-111,7223	0	-112,4867	-110,9579	SANT
React	Vue	-101,8622	0	-102,6266	101,0978	SANT
Svelte	Vue	9,8601	0	9,0957	10,6245	SANT

Figur 42 Tukey-test flera jämförelser av medelvärdet vid filtrering av fyra årtal

6.3.5 Mätserie 5

Resultatet av femte mätserien med filtrering av fem årtal visas i figur 43.



Figur 43 Linjediagram för filtrering av fem årtal

Ett ANOVA-test genomförs för att se om det finns någon överlappning. Testet visar att det finns en statistisk signifikant skillnad mellan ramverken med p-värde 0, se figur 44.

SAMMANFATTNING

Grupper	*Antal*	*Summa*	*Medelvärde*	*Varians*
Vue	1000	31347,79956	31,34779956	20,89906267
React	1000	210260,8079	210,2608079	410,2957249
Svelte	1000	14990,60425	14,99060425	3,789526793

ANOVA

Variationsursprung	*KvS*	*fg*	*MKv*	*F*	*p-värde*	*F-krit*
Mellan grupper	23469291,6	2	11734645,8	80931,51003	0	2,998728735
Inom grupper	434549,33	2997	144,9947714			
Totalt	23903840,93	2999				

Figur 44 ANOVA-test för filtrering av fem årtal

För att jämföra ramverken parvis genomförs ett tukey-test. Testet visar på att det finns skillnader i svarstider mellan samtliga ramverk, se figur 45.

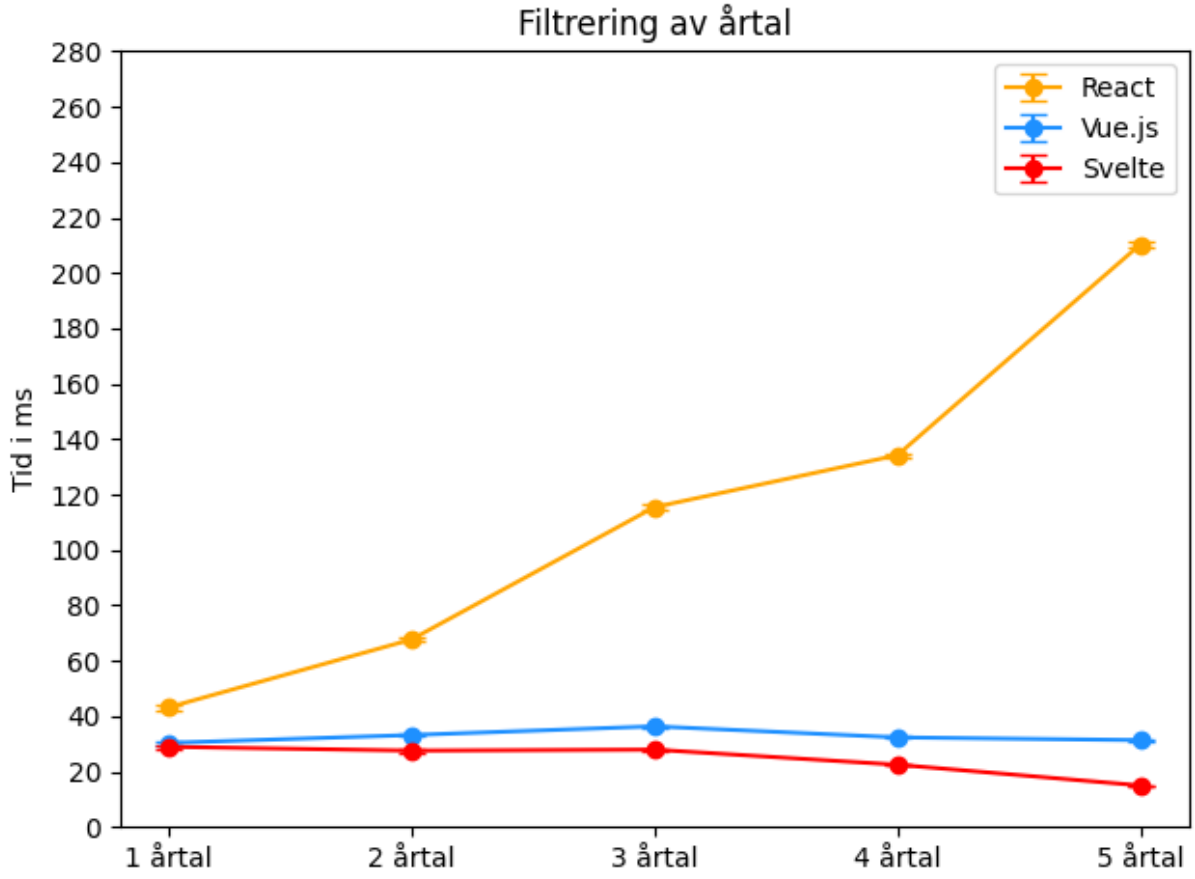
TUKEY

Grupp1	*Grupp2*	*Medeldiff*	*p-adj*	*Lägre*	*Övre*	*Avvisa*
React	Svelte	-195,2702	0	-196,5329	-194,0075	SANT
React	Vue	-178,913	0	-180,1757	-177,6503	SANT
Svelte	Vue	16,3572	0	15,0945	17,6199	SANT

Figur 45 Tukey-test flera jämförelser av medelvärdet vid filtrering av fem årtal

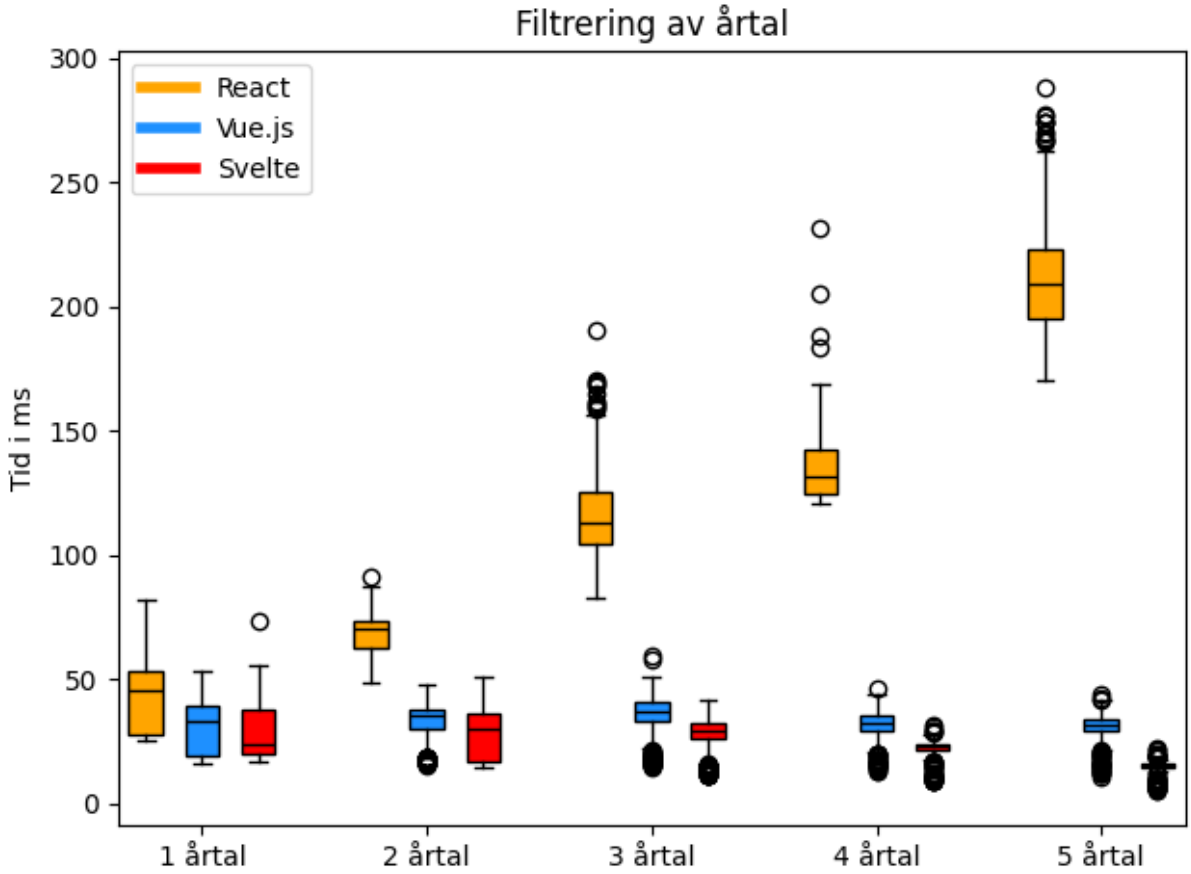
6.3.6 Skalbarhet

I figur 46 visas ett grupperat linjediagram över alla fem mätserier. Punkterna i diagrammet visar medelvärdet för ramverken i respektive mätserie.



Figur 46 Grupperat linjediagram innehållandes alla fem mätserier

Linjediagrammet (se figur 46) ger en tydlig överblick över hur ramverkens svarstider responderar när det kommer till skalbarhet. Resultatet som presenteras i diagrammet visar att det finns en skillnad i svarstider vid olika datamängder. React har betydligt högre svarstider än både Vue.js och Svelte. Vid fler filtreringval ökar Reacts kurva över svarstider medan Vue.js ligger på en relativt jämn nivå och Sveltes kurva minskar.



Figur 47 Grupperat boxplot diagram innehållandes alla mätserier

Boxplot diagrammet (se figur 47) ger en överblick över hur ramverkens svarstider responderar när det kommer till skalbarhet. I boxplot diagrammet är det lättare att se distributionen av svarstiderna för det olika mätserierna. Diagrammet visar även utstickande värden som förkommit i en del av mätningarna.

6.4 Slutsatser

Det finns en statistisk skillnad i svarstider mellan React, Vue.js och Svelte i alla fem mätserier. Utifrån de linjediagram och tester som presenterats för samtliga mätserier syns ett tydligt samband för hur ramverken presterar i svarstider med hänsyn till filtrering. Vue.js har en kurva som ligger på en relativt jämn nivå genom alla mätserier. Svelte har en kurva som minkar vid fler filtreringsval, vilket innebär större datamängder. Vid filtrering av 1–3 årtal ligger kurvan för svarstider hos Svelte på en relativt jämn nivå. React har en kurva som visar på en ständig ökning av svarstider vid fler filtreringsval. Anledningen till att Vue.js och Svelte har lägre svarstider genom alla mätserier kan bero på att ramverken har reaktiva deklarationer. Det betyder att om värdet ändras uppdateras beroende värden automatiskt. Vilket resulterar i att endast berörda funktioner påverkas. React använder sig inte av reaktiva deklarationer och det kan vara orsaken till att ramverket har en betydligt högre kurva när det kommer till svarstider vid filtrering av data. Att notera är även att all data är utskriven vid första inladdningen av applikationen innan några filtreringsval har gjorts, vilket eventuellt kan ha en påverkan när det kommer till reaktivitet.

Utifrån de resultat som experimentet visat kan slutsatser dras för att besvara frågeställningarna från kapitel 3.1 Frågeställningar:

1. Vilket JavaScript ramverk har kortast svarstider vid filtrering av data i tabellform?
2. Finns det några signifikanta skillnader i svarstider hos ramverken vid skalbarhet?

Resultatet från experimentet visar att Svelte har kortare svarstider än React och Vue.js, vilket besvarar frågeställning 1. Näst kortast svarstider har Vue.js och sist kommer React som har längst svarstider. Gällande frågeställning 2 så finns det en skillnad i svarstider för samtliga ramverk vid olika datamängder, vilket syftar till ramverkets skalbarhet. Resultatet styrker även hypoteserna:

H1.A: Det kommer att finnas en signifikant skillnad i svarstider beroende på vilket JavaScript ramverk som används.

H2.A: JavaScript ramverken kommer att ha signifikant skillnader i svarstider beroende på storleken av datamängden.

7 Avslutande diskussion

I detta kapitel sammanfattas studien som en helhet och en diskussion kring hur arbetet har genomförts presenteras. Andra ämnen som nämns i detta kapitel är etiska frågor som berörts, samhällelig nytta samt idéer för framtida arbeten.

7.1 Sammanfattning

Målet med den här studien är att jämföra olika JavaScript ramverk för att få fram vilket ramverk som ger snabbast svarstider vid filtrering av data i tabellform samt om det finns någon skillnad i svarstider hos ramverken vid skalbarhet. För att få svar på frågeställningarna sattes fyra delmål upp.

Delmål 1 ”Identifiera potentiella JavaScript ramverk och välja ut möjliga ramverk enligt tidigare identifierade urvalskriterier”, uppfylls genom att ta fram JavaScript ramverk utifrån två urval. Det första urvalet bestod av fem olika funktioner tagna från tre områden ur en modell skapad av Pano et el. (2018) och det andra urvalet bestod av anpassade kriterier för den här studien. De JavaScript ramverk som utsågs var Svelte, React och Vue.js.

Delmål 2 ” Identifiera ett sätt att mäta och jämföra svarstiden på hos de utvalda ramverken”, uppfylls genom att mäta tiden från och med att vald filtrering med checkboxar klickats i för respektive implementation. För att jämföra svarstiderna används diagram och tester.

Delmål 3 ”Genomföra jämförelser i en kontrollerad miljö och lagra resultaten”, uppfylls genom ett experiment där ett skript har skapats för att kunna lagra en starttid och en sluttid när filtrering med checkboxar genomförs. Resultaten sparas sedan ner i en extern textfil.

Delmål 4 ”Sammanställa, visualisera och analysera resultaten från delmål 3 och dra en slutsats för att besvara frågeställningarna”, uppfylls genom att omvandla data från textfilerna till diagram samt genom ANOVA och tukey tester. Diagrammen och testerna har skapats med hjälp av Python och Excell för att lättare kunna jämföra resultaten. Därefter har resultatet analyserats för att kunna ge svar till frågeställningarna:

1. Vilket JavaScript ramverk har kortast svarstider vid filtrering av data i tabellform?
2. Finns det några signifikanta skillnader i svarstider hos ramverken vid skalbarhet?

Analyseringen av mätningarnas resultat visar på att Svelte är de ramverk som ger kortast svarstider vid filtrering av data av de JavaScript ramverk som experimentet utfördes på. Näst snabbast var Vue.js och sist kom React.js. Analysen visar också att ramverken skalar olika i svarstider vid olika datamängder. Från resultatet styrks båda hypoteserna:

H1.A: Det kommer att finnas en signifikant skillnad i svarstider beroende på vilket JavaScript ramverk som används.

H2.A: JavaScript ramverken kommer att ha signifikant skillnader i svarstider beroende på storleken av datamängden.

7.2 Diskussion

Resultatet av studien visar på att Svelte ger snabbast svarstider vid filtrering av data oavsett datamängdens storlek, på andra plats hamna Vue.js och sist blev React med längst svarstider. Det som sker vid filtreringen i experimentet är att tabellens data uppdateras efter de filtreringsval som klickats i.

React, Vue.js och Svelte är uppbyggda med olika tekniska funktionaliteter som kan ha en inverkan på dess svarstider. En anledning till att React har betydligt högre svarstider än Vue.js och Svelte kan bero på att ramverket inte använder sig av reaktiva deklarationer. Att Svelte har kortare svarstider än Vue.js kan bero på att de hanterar data olika. Vue.js använder sig av virtual DOM medan Svelte använder sig av en kompilator och uppdaterar direkt till DOM. En möjlig förklaring till varför svarstiderna hos Svelte minskade när fler filtreringsval användes kan bero på att all data är utskriven i tabellen från start. Vilket betyder att vid uppdateringen av fler årtal blir det en mindre mängd data som sorteras bort och inte visas upp.

Experimentet visar på att det finns en statistisk skillnad i svarstider mellan Vue.js och Svelte. Skillnaden är endast på ett fåtal millisekunder och räknas därför inte som en skillnad i praktiken, eftersom det inte ger en märkbar påverkan för användaren. I linjediagrammet, som visas i figur 46 under kapitel 6.3.6 Skalbarhet, ser det ut som att Svelte har en mer logaritmisk utveckling medan Vue.js liknar en linjärutveckling. Utifrån de datapunkter som visas i diagrammet går det däremot inte att säkerhetsställa hur utvecklingen skulle se ut för ramverken om det fanns ännu fler filtreringsalternativ. Värt att nämna är att vid ett färre antal mätningar finns det inga statistiska skillnader mellan Vue.js och Svelte när det kommer till svarstider vid filtrering av data. I pilotstudien, kapitel 5.4, genomfördes 100 mätningar med en filtrering på två årtal. Tukey testet som genomfördes visade att det inte finns någon signifikant skillnad i svarstider mellan Vue.js och Svelte.

Likt detta arbete gör Ollila et al. (2022) en jämförelse mellan olika JavaScript ramverk. De fokuserar dock på renderingsstrategierna hos ramverken. Bland de ramverk som undersökts inkluderas React, Vue.js och Svelte. Studien mäter flera olika scenarion som kan påverka renderingstiden hos ramverken där en del av deras testscenarion riktar in sig på kostnaden för uppdateringsåtgärder. Resultatet visar att mellan ramverken React, Vue.js och Svelte är det Svelte som presterar bäst i det stora hela när det kommer till snabbast tider vid uppdateringsåtgärder. Resultatet visar även att React hade längst tider av React, Vue.js och Svelte.

Diniz-Junior et al. (2022) utför en prestandautvärdering bestående av funktionerna: skapa, redigera och ta bort ett visst antal element i en webbläsares fönster. De ramverk som jämfördes i studien var Angular, React och Vue.js. I resultatet rekommenderas Vue.js för applikationer som kräver tung DOM-manipulation och React för applikationer som kräver lättare och interaktiva miljöer.

Resultaten från de relaterade forskningsartiklarna överensstämmer med resultatet som presenterats i den här studiens experiment. Även om inriktningarna i studierna är olika så har resultaten visat sig vara likartade. Det som kan skilja sig åt från resultatet i den här studien är att Diniz-Junior et al (2022) rekommenderar React för lättare och interaktiva miljöer. Dock kan lättare och interaktiva miljöer motsvara filtreringen av endast ett årtal i den här studien. Vid filtrering av endast ett årtal är datamängden som visas upp i tabellen endast ca 365 rader. Det går tydligt att se i figur 46, under kapitel 6.3.6 Skalbarhet, att svarstiderna för React ligger

mer i linje med svarstiderna för Vue.js och Svelte vid filtrering av ett årtal än vid fler filtreringsval med en större datamängd. I den här studien används en tabell med över 1000 rader data som filtrerats och uppdaterats, vilket kan kräva en tyngre DOM-manipulation och där rekommenderar Diniz-Junior et al (2022) Vue.js.

7.2.1 Etiska aspekter

Aspekter som kan påverka studien är bland annat att mätningarna har genomförts under olika tider på dygnet vilket kan påverka resultatet. De enheter som är kopplade till samma nätverk har stängts av under tiden som mätningarna har genomförts för att säkerställa att de inte påfrestar nätverket. Under experimentet avaktiverades datorns virusskydd och de enda program som kördes var de som är relaterade till studien. För att se till så inte onödiga påfrestningar sker genom att andra program körs i bakgrunden under tiden som mätningarna pågår.

Samtliga tester genomfördes på samma dator med operativsystemet Windows 11, mer specificerad information finns under kapitel 6.2 Presentation av undersökningen, och i webbläsaren Google Chrome. Experimentet har inte varit möjligt att utföra på andra webbläsare eller operativsystem på grund av studiens tidsram. Vilket innebär ett validitetshot mot generaliserbarheten. Eftersom resultatet från den här studien inte nödvändigtvis överensstämmer vid användandet av ett annat operativsystem eller en annan webbläsare.

Studien använder sig inte av människor och innehåller inga känsliga uppgifter. Datasetet som används kommer från Bolin Centre of Climate Research öppna databas och ligger under licensen Open Data Commons Attribution License (ODC-By)[3]. All programkod som används i studien finns tillgänglig på Github[4]. Det bidrar till att studien går att återskapa av andra som vill utföra ett framtida arbete av studien eller validera studiens resultat.

I de delar av rapporten där andra texter eller arbeten har använts anges referenser för att inte ta äran för någon annans arbete. Allt arbete i studien har gjorts med gott uppsåt. Har eventuella fel inträffat är dessa inte avsiktliga.

7.2.2 Samhällelig nytta hos arbetet

Studien har genomförts för att undersöka hur olika ramverks svarstider ser ut vid filtrering av data för att kunna dra en slutsats om vilket ramverk som presterar bäst med kortast svarstid. Studien bidrar med att göra valet av JavaScript ramverk enklare för utvecklare där snabba svarstider är en viktig faktor för deras projekt. Genom att uppnå snabba svarstider kan en webbapplikation enklare leva upp till användares förväntningar och därmed erbjuda en bättre upplevelse för slutanvändare.

Med hjälp av filtrering är det möjligt för utvecklare att skapa interaktiva webbapplikationer som är mer komplexa. Filtrering skapar en struktur av data som kan användas för flera olika ändamål och leder till en förbättrad effektivitet i IT processer. Eftersom det går att dela upp och sortera data ger det företag en möjlighet att använda sig av större datamängder på ett strukturerat sätt.

[3] https://bolin.su.se/data/stockholm-historical-thermometer-1
[4] https://github.com/g21emmka/Examensarbete

Ur hållbarhetsperspektiv kan webbapplikationer med snabba svarstider vid filtrering av data bidra med reducering av energiförbrukning samt minimera behovet av resurskrävande underhåll. Om en webbapplikation är lätt att underhålla behöver utvecklare inte spendera lika mycket tid på att arbeta med applikationen, vilket reducerar utvecklartimmarna. Det går även att spara in på den energi som det tar för utvecklare att åka till arbetsplatsen för att fixa till några buggar i systemet. I slutändan kan det leda till både kostnadsbesparingar och minskad miljöpåverkan.

En annan viktig aspekt att tänka på ur ett hållbart perspektiv är inlåsningseffekter. För att uppnå hållbarutveckling bör oönskad inlåsning undvikas. Därför är det viktigt att olika ramverk kan interagera med varandra. Både React och Vue.js har skapats på ett sådant sätt att de går att lägga till delar av ramverken till andra webbapplikationer som är byggda med andra tekniska funktionaliteter eller JavaScript ramverk. Dock kräver React att den data som används ska ha unika nycklar som kan identifiera olika objekt. Finns inte unika nycklar kommer en varning att dyka upp i konsolen. Av de ramverk som använts i den här studien var React det enda ramverket som klagade på att det inte fanns något unikt som kunde identifiera objekten i JSON-filen. Även om de flesta filer använder sig av exempelvis id:n för att identifiera olika objekt är det inte alla som gör det, något som kan påverka vid integration av andra applikationer ihop med React. Svelte är ett nyare ramverk (jämfört med React och Vue.js) och det framgår inte om Svelte har kapaciteten till att interagera med andra ramverk på samma sätt som React och Vue.js. Alla tre ramverk är open-source och ligger under MIT licensen. Utvecklare har därmed möjligheten att själva utföra vissa modifieringar om de anser att det nödvändigt.

7.3 Framtida arbete

Det finns flera sätt att driva denna studie vidare på vid ett framtida arbete för att få mer kunskap om olika JavaScript ramverk vad gäller filtrering av data.

Att använda sig av flera olika filtreringval blev för komplext i den här studien och det diskuteras närmare under kapitel 5.3 Progression. Ett framtida arbete hade därför kunnat vara att lägga till fler filtreringsval för att möjligen visa på en ännu större varians och skalbarhet. Det skulle vara intressant att se om resultatet för svarstiderna följer samma kurva mellan ramverken eller om resultatet hade förändrats vid fler filtreringsval. Ett annat alternativ hade varit att använda ett större dataset för att se hur ramverken hanterar väldigt stora datamängder.

Studiens experiment genomfördes endast i en webbläsare (Google Chrome) och med samma operativsystem (Windows 11). Att genomföra studien i en annan webbläsare som Mozilla Firefox eller Microsoft Edge med en ett annat operativsystem än Windows 11 hade varit intressant, eftersom det skulle kunna leda till andra resultat.

Experimentet kan även genomföras tillsammans med Backbone som inte valdes med i denna studie. Ramverket skiljer sig från de tre ramverk som experimentet genomfördes med genom att det varken använder sig av reaktiva deklarationer eller virtual DOM. Ett fortsatt arbete av denna studie med Backbone hade bidragit med ett komplement för att se hur ramverkets svarstider hade skiljt sig åt från React, Vue.js och Svelte som antingen använder sig av reaktiva deklarationer, virtual DOM eller båda två.

Kommande studier kan inkludera att genomföra experimentet med antingen ännu fler JavaScript ramverk eller genom att bygga vidare på studien och jämföra helt andra JavaScript ramverk, för att se hur deras svarstider presterar vid filtrering av data. Det skulle vara intressant att bygga vidare på filtreringen och lägga till en sökruta som ett filtreringsval för att få fram eftersökt data.

Studien hade även kunnat genomföras med någon av de alternativa metoderna, fallstudie eller enkätundersökning som finns beskrivet under kapitel 3.5 Alternativa metoder, som ett framtida arbete. Även om ett tekniskt experiment utfördes i den här studien så har svarstiderna en stor inverkan för användarupplevelsen.

Ytterligare ett alternativ som skulle vara intressant att inkludera i ett framtida arbete är att ta en djupare inblick i ramverket React, som inte använder sig av reaktiva deklarationer. Med mer tid och resurser hade reaktiva deklarationer kunnat byggas ihop med ramverket för att se om svarstiderna för React hade minskat. Ett annat alternativ till ett större framtida arbete, ifall det finns obegränsat med både resurser och tid, skulle vara att bygga ett program som gör det enklare att kombinera olika ramverk med varandra. På så sätt kan utvecklare använda sig av både nyare och äldre ramverk för att bygga ett system eller en applikation. Programmet skulle även bidra till en hållbarutveckling eftersom det ser till att oönskade inlåsningseffekter undviks oavsett vilket ramverk som används. Det hjälper företag att enkelt och smidigt uppgradera sina system och webbsidor med nyare tekniker utan att behöva flytta över eller skriva om all kod. Programmet skulle kunna skapas som antingen ett tillägg eller som ett helt fristående program.

Referenser

Angular (2023) https://angular.io/guide/what-is-angular [2024-04-28]

Angular (2024) https://angular.io/ [2024-04-28]

Bartuskova, A., & Krejcar, O. (2015). Loading speed of modern websites and reliability of online speed test services. In *Computational Collective Intelligence: 7th International Conference, ICCCI 2015, Madrid, Spain, September 21-23, 2015, Proceedings, Part II* (pp. 65-74). Springer International Publishing. doi:10.1007/978-3-319-24306-1_7

Diniz-Junior, R. N., Figueiredo, C. C. L., Russo, G. D. S., Bahiense-Junior, M. R. G., Arbex, M. V., Dos Santos, L. M., ... & Giuntini, F. T. (2022, October). Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue. In *2022 XVLIII Latin American Computer Conference (CLEI)*. Armenia, Colombia 17-21 October 2022 (pp. 1-9). IEEE. doi:10.1109/CLEI56649.2022.9959901

Ferreira, F., Borges, H. S., & Valente, M. T. (2022). On the (un-) adoption of JavaScript front-end frameworks. *Software: Practice and Experience, 52*(4), pp. 947-966. doi:10.1002/spe.3044

GitHub Examensarbete https://github.com/g21emmka/Examensarbete [2024-05-22]

Github facebook/react (2022) https://github.com/facebook/react/blob/main/LICENSE [2024-01-21]

Github QwikDev/qwik (2024) https://github.com/QwikDev/qwik/blob/main/LICENSE [2024-04-28]

Gizas, A., Christodoulou, S., & Papatheodorou, T. (2012). Comparative evaluation of javascript frameworks. In *Proceedings of the 21st International Conference on World Wide Web*. Lyon, France 16-20 April 2012, (pp. 513-514). 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), pp. 2142-2157. doi:10.1109/TKDE.2016.2557324

Graziotin, D., & Abrahamsson, P. (2013). Making sense out of a jungle of JavaScript frameworks: towards a practitioner-friendly comparative analysis. In *Product-Focused Software Process Improvement: 14th International Conference, PROFES 2013, Paphos, Cyprus, June 12-14, 2013. Proceedings 14* (pp. 334-337). Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-39259-7_28

Jiang, Z., Zhong, H., & Meng, N. (2021). Investigating and recommending co-changed entities for JavaScript programs. *Journal of Systems and Software, 180*, 111027. doi:10.1016/j.jss.2021.111027

Moberg, A. (2019) Stockholm Historical Weather Observations — Thermometer observations since 1756. Dataset version 1. Bolin Centre Database. doi:10.17043/stockholm-historical-thermometer-1 [2024-04-11]

Nielsen, J. (1997). User interface design for the WWW. In *CHI'97 Extended Abstracts on Human Factors in Computing Systems*. Atlanta Georgia, United States 22-27 Mars 1997 (pp. 140-141). doi:10.1145/1120212.1120312

Nielsen, J. (1999). User interface directions for the web. *Communications of the ACM*, 42(1), pp. 65-72. doi:10.1145/291469.291470

Ocariza, F. S., Bajaj, K., Pattabiraman, K., & Mesbah, A. (2017). A study of causes and consequences of client-side JavaScript bugs. *IEEE Transactions on Software Engineering, 43*(2), pp. 128-144. doi:10.1109/TSE.2016.2586066

Ollila, R., Mäkitalo, N., & Mikkonen, T. (2022). Modern Web Frameworks: A Comparison of Rendering Performance. *Journal of Web Engineering, 21*(3). doi:10.13052/jwe1540-9589.21311

Pano, A., Graziotin, D., & Abrahamsson, P. (2018). Factors and actors leading to the adoption of a JavaScript framework. *Empirical Software Engineering, 23*, pp. 3503-3534. doi:10.1007/s10664-018-9613-x

Qwik (2023) https://qwik.dev/docs/ [2024-04-28]

React (2024a) https://react.dev/ [2024-04-17]

React (2024b) https://react.dev/learn [2024-04-17]

React (2024c) https://react.dev/learn/writing-markup-with-jsx [2024-04-28]

Selvidge, P. R., Chaparro, B. S., & Bender, G. T. (2002). The world wide wait: effects of delays on user performance. *International Journal of Industrial Ergonomics, 29*(1), pp. 15-20. doi:10.1016/S0169-8141(01)00045-2

Shneiderman, B. (1984). Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR), 16*(3), pp. 265-285. doi:10.1145/2514.2517

Stackoverflow (2023) *2023 Developer Survey* https://survey.stackoverflow.co/2023/#technology [2024-02-22]

Svelte (2023a) https://svelte.dev/ [2024-04-14]

Svelte (2023b) https://svelte.dev/docs/svelte-components [2024-04-14]

Vepsäläinen, J., Hellas, A., & Vuorimaa, P. (2023). The Rise of Disappearing Frameworks in Web Development. In *International Conference on Web Engineering, ICWE 2023, Alicante, Spain, June 6-9, 2023* (pp. 319-326). Springer, Cham. doi: 10.1007/978-3-031-34444-2_23

Vue.js (2024a) https://vuejs.org/ [2024-04-17]

Vue.js (2024b) https://vuejs.org/guide/introduction.html [2024-04-17]

Vue.js (2024c) https://vuejs.org/api/built-in-directives.html [2024-04-17]

Weinberg, B. D. (2000). Don't keep your internet customers waiting too long at the (virtual) front door. *Journal of interactive marketing*, 14(1), pp. 30-39. doi:10.1002/(SICI)1520-6653(200024)14:1<30::AID-DIR3>3.0.CO;2-M

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

Xu, W., Yang, X., & Shi, Y. (2010, November). Enhancing browsing experience of table and image elements in web pages. In *10: International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*. Beijing, China 8-10 November 2010 (pp. 1-8). doi:10.1145/1891903.1891935

Zari, M., Saiedian, H., & Naeem, M. (2001). Understanding and reducing web delays. *Computer, 34*(12), pp. 30-37. doi:10.1109/2.970554