Master's thesis

Master's Programme in Computer Science

Benefits and Challenges of Isomorphism in Single-Page Applications: A Case Study and Review of Gray Literature

Aleksi Huotala

May 18, 2021

Faculty of Science

University of Helsinki

Supervisor(s)

Dr. M. Luukkainen, Prof. T. Mikkonen

Examiner(s)

Dr. M. Luukkainen, Prof. T. Mikkonen

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki, Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Aleksi Huotala

Työn nimi — Arbetets titel — Title

Benefits and Challenges of Isomorphism in Single-Page Applications: A Case Study and Review of Gray Literature

Ohjaajat — Handledare — Supervisors

Dr. M. Luukkainen, Prof. T. Mikkonen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	May 18, 2021	42 pages

Tiivistelmä — Referat — Abstract

Isomorfiset web-sovellukset yhdistävät staattisten Hypertext Markup Language (HTML) -sivujen ja Single-page application (SPA) -sovellusten parhaat puolet. Isomorfinen web-sovellus jakaa koodia palvelimen ja käyttöliittymän välillä.

Isomorfisista web-sovelluksista ei ole tehty kovinkaan paljoa tieteellistä tutkimusta. Web-sovellusten suorituskyvyn, käyttökokemuksen ja kehittäjäkokemuksen parantaminen on suosittu tietojenkäsittelytieteen tutkimusaihe. Tässä tutkielmassa tutkitaan isomorfismin hyötyjä ja haasteita SPA-sovelluksissa.

Isomorfismin hyötyjen ja haasteiden tutkimiseksi suoritettiin kirjallisuuskatsaus ja tapaustutkimus. Kirjallisuuskatsauksessa käytettiin lähteenä internetistä löytyviä artikkeleita, joita etsittiin neljältä eri verkkosivustolta. Kirjallisuuskatsausta varten kerättyihin artikkeleihin suoritettiin laaduntarkastus, jotta niitä voitaisiin käyttää osana tieteellistä tutkimusta. Tapaustutkimus suoritettiin kehittäjähaastatteluna, jossa haastateltiin isomorfisia web-sovelluksia tuntevia kehittäjiä. Sekä kirjallisuuskatsauksen että tapaustutkimuksen tuloksia vertailtiin, ja niistä muodostettiin yhteenveto.

Kirjallisuuskatsauksen ja tapaustutkimuksen tuloksena isomorfismi SPA-sovelluksissa tuo etuja sekä käyttäjille että kehittäjille. Isomorfismi SPA-sovelluksissa on haastavaa toteuttaa. Isomorfismilla on myös muita, lähinnä kehittäjiä koskevia haittoja. Isomorfismi parantaa sovelluksen suorituskykyä ja hakukonekyvykkyyttä. Isomorfismin käyttö mahdollistaa ohjelmakoodin jakamisen palvelimen ja käyttöliittymän välillä, mutta se lisää sovelluksen monimutkaisuutta. Sovelluskirjastojen tuki on yksi isomorfismin ongelmista, joka kehittäjien tulee huomioida. Tämän tutkimuksen tulokset antavat kehittäjille syitä ja motivaatiota kehittää isomorfisia web-sovelluksia ja siirtää olemassa olevia sovelluksia käyttämään isomorfismia.

ACM Computing Classification System (CCS)
Information systems → World Wide Web → Web applications

Avainsanat — Nyckelord — Keywords

isomorphism,javascript,server,web

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software study track

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Aleksi Huotala

Työn nimi — Arbetets titel — Title

Benefits and Challenges of Isomorphism in Single-Page Applications: A Case Study and Review of Gray Literature

Ohjaajat — Handledare — Supervisors

Dr. M. Luukkainen, Prof. T. Mikkonen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	May 18, 2021	42 pages

Tiivistelmä — Referat — Abstract

Isomorphic web applications combine the best parts of static Hypertext Markup Language (HTML) pages and single-page applications. An isomorphic web application shares code between the server and the client.

However, there is not much existing research on isomorphic web applications. Improving the performance, user experience and development experience of web applications are popular research topics in computer science. This thesis studies the benefits and challenges of isomorphism in single-page applications.

To study the benefits and challenges of isomorphism in single-page applications, a gray literature review and a case study were conducted. The articles used in the gray literature review were searched from four different websites. To make sure the gray literature could be used in this study, a quality assessment process was conducted. The case study was conducted as a developer survey, where developers familiar with isomorphic web applications were interviewed. The results of both studies are then compared and the key findings are compared together.

The results of this study show that isomorphism in single-page applications brings benefits to both the developers and the end-users. Isomorphism in single-page applications is challenging to implement and has some downsides, but they mostly affect developers. The performance and search engine optimization of the application are improved. Implementing isomorphism makes it possible to share code between the server and the client, but it increases the complexity of the application. Framework and library compatibility are issues that must be addressed by the developers. The findings of this thesis give motivation for developers to implement isomorphism when starting a new project or transforming existing single-page applications to use isomorphism.

ACM Computing Classification System (CCS)
Information systems → World Wide Web → Web applications

Avainsanat — Nyckelord — Keywords

isomorphism,javascript,server,web

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software study track

Contents

List of Acronyms

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CPU Central Processing Unit

CSS Cascading StyleSheets

DOM Document Object Model

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

NPM Node.js Package Manager

REST Representational State Transfer

SPA Single-page application

SSR Server-side rendering

SEO Search engine optimization

TTI Time To Interactive

TTFB Time To First Byte

UI User interface

UX User experience

V8 Google's open-source high-performance JavaScript and WebAssembly engine

WWW World Wide Web

XHR XMLHttpRequest

1 Introduction

The increased popularity of the World Wide Web (WWW) has lead to the development of software for the web (Taivalsaari et al., 2008). Asynchronous JavaScript and XML (AJAX) started to gain traction almost 20 years ago, which lead to the introduction of XMLHttp-Request (XHR) Application Programming Interface (API) (Scott, 2015). The XHR API can be used to fetch data asynchronously on the client-side (Scott, 2015). The interest to merge XHR with HTML and JavaScript made web applications evolve from static HTML web pages into complex JavaScript applications. Developers are continuously looking for ways to improve the performance and user experience of their web applications. They also value the developer experience, which they aim to improve by using new software libraries and tools.

The Single-page application (SPA) was introduced in 2002 (Birdeau et al., 2002). A SPA is a JavaScript application, which does not reload the page if the user navigates in it (Mikowski and Powell, 2014). With the release of Angular.js [1] and Backbone.js [2] in 2010, web applications have become more dynamic and can serve more richful content to the end-users. This, however, has introduced some problems regarding site performance and Search engine optimization (SEO) (Silva and Farah, 2018). In some cases, web crawlers fail to index SPAs correctly (Silva and Farah, 2018). SPAs might also be slow to load on mobile devices because the page is rendered with JavaScript code. Older devices, which do not support JavaScript, cannot run JavaScript SPAs. The proposed solution for the performance and SEO-related issues of SPAs is the isomorphic web application architecture (Silva and Farah, 2018).

Isomorphic web applications combine the best parts of statically generated websites and SPAs (Gordon, 2018). An isomorphic web application first renders the initial page server-side and sends it to the browser. The browser then attaches event handlers and Document Object Model (DOM) handlers to the initial page markup, which is called hydration (Facebook Inc., 2013). After hydrating the application in the client, the application behaves like a SPA (Silva and Farah, 2018). This all happens in full transparency for the client, where the user browsing the site might not even notice that the application has switched

[1] https://angularjs.org/
[2] https://backbonejs.org/

from a static markup webpage to an interactive JavaScript application.

Isomorphic web applications share code between the server and the client (Silva and Farah, 2018). The code that runs in both environments is called Universal JavaScript (Rauschmayer, 2015). Isomorphic web applications provide good performance and user experience for the end user. From the developers' point of view, the isomorphic web application architecture leads to increased complexity. The isomorphic web application architecture also has a steep learning curve.

Software architecture requirements and the lack of proper toolkits for making a web application that uses the same programming language in both the server and the client are the reasons why these problems haven't been solved before. JavaScript is nowadays a well-supported programming language across the server and the client, and it is being continuously developed.

In this thesis, we studied the benefits and challenges of isomorphism in single-page applications. To study the benefits and challenges, we conducted a gray literature review and a case study that was constructed as a developer survey. The results of the gray literature review are compared to the case study. The results of this joint study are then used to create an overview of the benefits and challenges of isomorphism in single-page applications.

The results of the study show that isomorphic web applications improve the performance, user experience, and SEO of single-page applications. The ability to share code between the server and the client improves the maintainability of the application. The complexity of the application increases and code might get duplicated. The framework and library support for isomorphic web applications has a lot of catching up to do. Data fetching and state management are architectural challenges of isomorphic web applications. Isomorphic web applications can be hard to grasp in the beginning, and it might be not worth the trade-off of increased complexity to implement isomorphism in single-page applications.

The structure of the study is as follows. First, in Chapter 2, we introduce prerequisite information, which is important for the base of this thesis. Next, in Chapter 3, we talk about our research approach and introduce the research questions for this study. Chapter 4 presents the results of this study. In Chapter 5, we discuss the results of this study. In the last chapter, Chapter 6, we conclude the study as a whole.

2 Background

In this chapter, we introduce isomorphic web applications, SPAs, Server-side rendering (SSR), State management, and SEO, which are all important for the base of this thesis.

2.1 Isomorphic web application

An isomorphic web application is a web application that shares the application's code between the server and the client (NR, 2015). Executing the same code in different environments require polyfills for the environment-specific dependencies. A polyfill is a piece of code that emulates an API, so it can function in a different environment (Sons et al., 2013).

Isomorphic web applications combine the best parts of single-page applications and statically generated websites (Gordon, 2018). In Figure 2.1, the isomorphic web application architecture is shown. From Figure 2.1 we see that the application's state is shared between the server and the client. The code of the application is bundled into a JavaScript file, which will be downloaded by the browser.

The server constructs the initial HTML markup by using the application's state and User interface (UI) code. The server can also do Hypertext Transfer Protocol (HTTP) requests to external APIs. The data from HTTP responses are then processed and inserted into the initial HTML markup. The application's state is encoded and injected into the HTML template, which can later be accessed by the client.

The final operation the client needs to do is to hydrate the application (Gordon, 2018). Hydration means attaching existing event handlers and DOM handlers to the HTML document (Facebook Inc., 2013). In this phase, the application's state generated by the server will be decoded. After hydration has been done, the isomorphic web application functions like any other single-page application (Gordon, 2018). If the hydration fails, the application might not function correctly; the UI might flicker, output errors in the browser's developer console or be otherwise completely unusable.

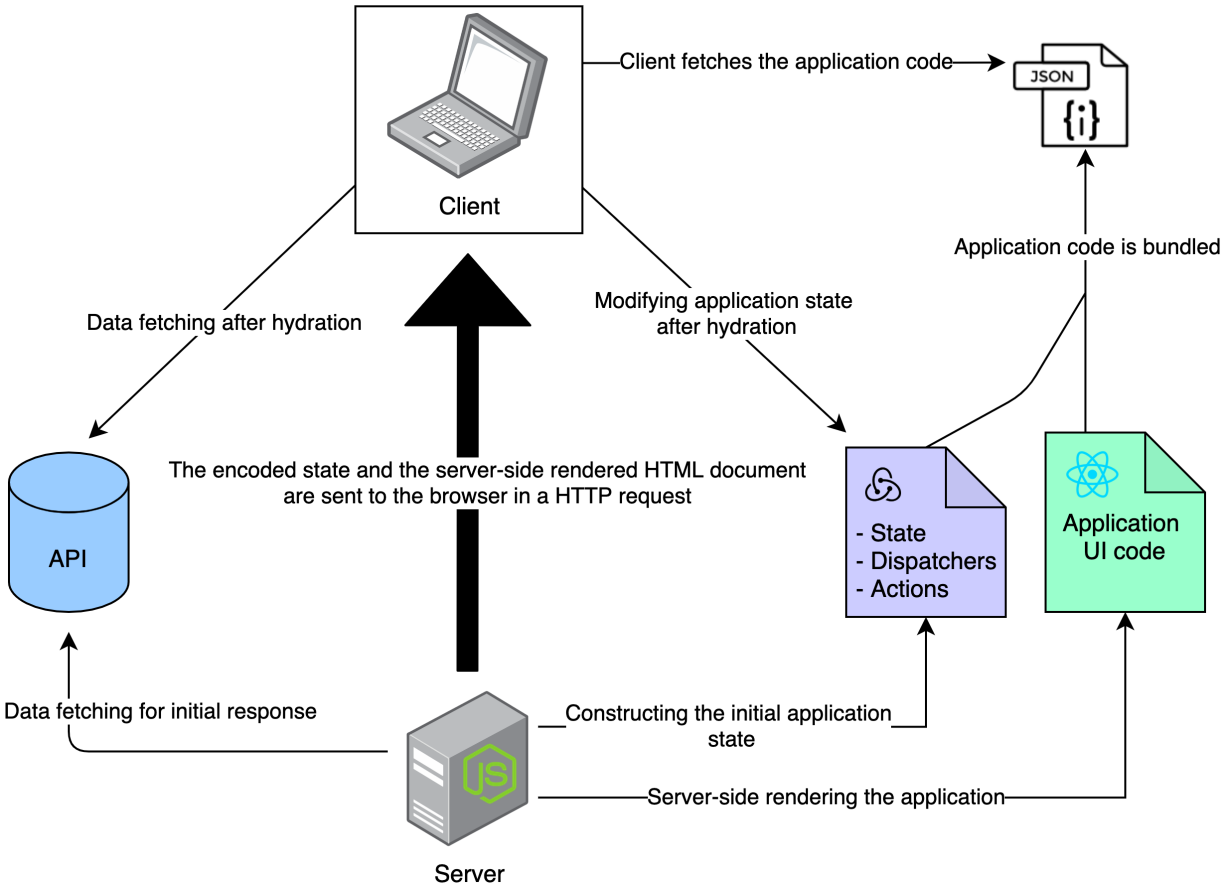

Figure 2.1: Isomorphic web application architecture.

2.2 Single-page application

A single-page application is a web application that does not reload the page, if the user navigates in it (Mikowski and Powell, 2014). Single-page applications work by embedding the source code of the application into a JavaScript file, which is downloaded by the browser for execution. Single-page applications adopt AJAX to the structure of the entire application (Scott, 2015).

AJAX was introduced in the 1990s to make web applications behave more like desktop applications (Paulson, 2005). AJAX can be used to fetch data without actually reloading the webpage (Paulson, 2005). It makes desktop applications more dynamic and responsive (Paulson, 2005).

The single-page application functions as follows:

1) The server responds to the user's request with the initial HTML page markup, as seen in Figure 2.2.

2) The user's browser downloads other assets, such as executable JavaScript code and Cascading StyleSheets (CSS) files.

3) After the assets have been downloaded, the JavaScript code is executed. The JavaScript code will mount the application into the initial page markup.

4) After all AJAX requests for fetching data have been completed, the page has finished loading.

There are many JavaScript front-end frameworks available, which can be used to develop single-page applications. The most popular front-end frameworks of today are listed in Table 2.1.

React.js [3] is an open-source front-end framework developed by Facebook. It was first released in 2013 and is in active development (Facebook Inc., 2013). With over ten million downloads to date (Table 2.1), React.js is currently the most popular JavaScript front-end application of the three. **Vue.js** [4] and **Angular** [5] are front-end frameworks that compete with React.js, which both have good developer backing with tens of thousands of stars in GitHub.

[3] https://reactjs.org
[4] https://vuejs.org/
[5] https://angular.io/

Table 2.1: Comparison of downloads and GitHub stars of front-end frameworks, as of 5th May 2021 (*React vs Vue vs Angular* 2021).

	Downloads (all time)	GitHub stars
React.js	10,854,762	167,872
Vue.js	2,303,841	182,805
Angular	577,932	59,610

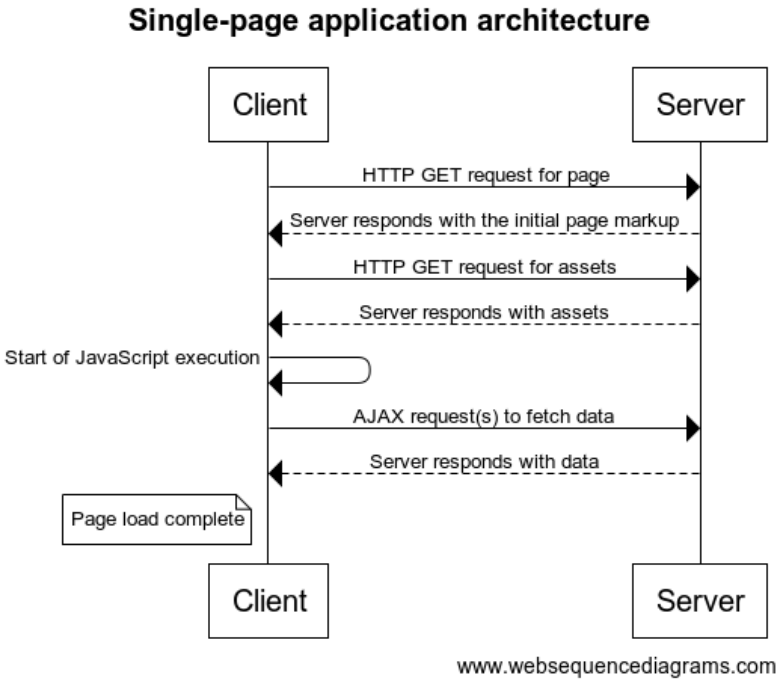

Figure 2.2: Single-page application architecture.

2.3 Server-side rendering

Server-side rendering (SSR) is a technique used to render web pages on the server, rather than on the client (Sun, 2019). Single-page applications render the initial page on the client using JavaScript, which requires JavaScript support from the web browser. Backbone.js, React.js, Angular, and Vue.js supports server-side rendering out of the box, meaning that an existing single-page application could be transformed into a server-side rendered application (Mukkamala, 2018; Airbnb, 2012).

When the user makes an HTTP request to the server-side rendered application, the server starts to generate static HTML. For example, server-side rendering of the React.js component in Listing 2.1 outputs a static HTML representation seen in Listing 2.2. The event handlers and component identifiers, which React.js uses internally are also inserted as element attributes in the HTML elements.

```
// App.js
import React from "react";

const App = () => (
  <div className="application">
    <ul id="list">
      <li onClick={(_e) => console.log("Clicked first")}>First</li>
      <li onClick={(_e) => console.log("Clicked second")}>Second</li>
    </ul>
  </div>
);

export default App;
```

Listing 2.1: A React.js component that will be server-side rendered.

```
// Server.js
import React from "react";
import ReactDOMServer from "react-dom/server";
import App from "./App";

ReactDOMServer.renderToString(<App />);
/*
<div
  class="application"
  data-reactid=".0"
  data-react-checksum="-1358679199"
>
  <ul id="list" data-reactid=".0.0">
    <li data-reactid=".0.0.0">First</li>
    <li data-reactid=".0.0.1">Second</li>
  </ul>
</div>
*/
```

Listing 2.2: Server-side rendering the "App" React.js component.

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

ReactDOM.hydrate(<App />, document.getElementById("root"));
```

Listing 2.3: Hydrating the React.js application in the client.

2.4 State management

State management in isomorphic web applications is a mandatory practice for synchronizing the state between the server and the client. The server hosting an isomorphic web application needs to send the application's state to the client at the end of the request. The server must also ensure that the state between the server and the client is coherent,

or the hydration process will not work properly. By using the same state management solution in the server and the client, the developer can reuse the same state management code. For example, the actions and dispatchers could be used in both environments.

2.5 Search engine optimization

Search engine optimization (SEO) is the process of improving the search performance of a website, by using crawlers and spiders to collect data from web pages (Cui and Hu, 2011; Killoran, 2013). The collected data can include keywords, meta tags, and links to other pages on the website (Killoran, 2013).

Tools such as Google Webmaster Tools [6], Google Analytics [7], and Bing Webmaster Tools [8] give valuable insight into websites' search performance (Barbar and Ismail, 2019). These tools gather statistics about user interactions, click counts, and search queries, which were used to find the page in the search results (Barbar and Ismail, 2019). That information can be used as a base to optimize the website's search performance, either by On-page or Off-Page SEO techniques (Barbar and Ismail, 2019).

On-page SEO is made on the website and its markup (Barbar and Ismail, 2019). In Listing 2.4, an example HTML document is shown where meta tags and special HTML elements are inserted to promote page content to the crawlers. Meta tags are located in the HTML document's head section, and hyperlinks can appear anywhere on the webpage (Killoran, 2013). Articles, headers, sections, and other semantic elements tell the crawlers, which kind of content the page contains.

In contrast, Off-page SEOs are used outside the website (Barbar and Ismail, 2019). Off-page SEO is usually made after the On-page SEO, to further enhance the SEO (Barbar and Ismail, 2019). Examples of Off-page SEO include backlinks, creating meaningful content, which boosts the page's reputation, page views, and archiving the page to some online directory, such as the Internet Archive [9] (Barbar and Ismail, 2019).

[6] https://www.google.com/webmasters/tools/siteoverview
[7] https://analytics.google.com/
[8] https://www.bing.com/webmasters
[9] https://archive.org/

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Page description" />
    <meta name="keywords" content="technology,computers" />
    <title>Page title</title>
  </head>
  <body>
    <article>
      <h1>Header</h1>
      <section>
        Hello world
      </section>
      <h2>Subheader</h2>
      <section>
        Lorem Ipsum
      </section>
    </article>
  </body>
</html>

```

Listing 2.4: A HTML document may contain meta tags and other semantical elements to make crawlers and spiders index the page better.

3 Research Approach

In this thesis, we conducted a concept-centric gray literature review described by Webster & Watson (Webster and Watson, 2002). In the process of conducting the gray literature review, we followed the gray literature review guidelines set by Garousi et al. (Garousi et al., 2019). We also conducted a case study where we interviewed a group of developers. Together, the results of the gray literature review and the case study are compared to identify the key benefits and challenges of isomorphism in SPAs.

We decided to conduct a gray literature review because there is not that much scientific literature available about isomorphic web applications. Many books about isomorphism and isomorphic web applications were found, but the books mostly focused on the technical implementation of isomorphic web applications. In this thesis we wanted to study the benefits and challenges developers say about isomorphism in SPAs, and how these benefits and challenges affect the end-users.

3.1 Research Questions

In this thesis, we answer the following research questions:

RQ1. What are the benefits of isomorphic web applications?

RQ2. What are the challenges of adopting isomorphism in single-page applications?

3.2 Gray Literature Review

In this section, we conduct a gray literature review on the benefits and challenges of isomorphism in SPAs. We have chosen four websites for the literature review, which are Reddit [10], Medium [11], Dev.to [12] and Hackernoon [13].

[10] https://reddit.com
[11] https://medium.com
[12] https://dev.to
[13] https://hackernoon.com

3.2.1 Search Query

The search query used to search for the gray literature was crafted in a way that it focuses on experiences, opinions, or thoughts about isomorphic web applications and universal JavaScript. We included articles, which contain *javascript*, *web*, and either *isomorphic* or *universal*. To focus on articles about experiences, opinions, or thoughts, we included articles, which contain either *experience*, *opinion*, or *thoughts*. To stop the search process, we used the evidence exhaustion stopping criteria described by Garousi et al. (Garousi et al., 2019).

site: website_url

javascript

AND

(isomorphic OR universal)

AND

web

AND

(experience OR opinion OR thoughts)

3.2.2 Inclusion and Exclusion Criteria

The inclusion criteria for the gray literature was to include all relevant articles to the topic, which contained personal experiences, opinions, or thoughts about isomorphic web applications and universal JavaScript. We also conducted a quality assessment for the gray literature.

In the gray literature quality assessment process, we ranked each article based on the authority of the producer, methodologies, objectivity, novelty, impact, and outlet type (Garousi et al., 2019). To form the list of quality assessment metrics, we looked at the *Quality assessment checklist of grey literature for software engineering* by Garousi et al. (Garousi et al., 2019). We modified the assessment checklist to suit our needs and the purpose of this thesis and ended up with 22 metrics that we used to rank the articles. These metrics are listed in Table 3.1. Each metric has a value between $[0.0, 1.0]$ on how well the article matches the criteria. The formula we used to compute the final score for

Table 3.1: Quality assessment metrics used in the inclusion and exclusion phase of the gray literature review.

No.	Metric
1	The reputation of the author's organization
2	Author's association to a reputable organization
3	Author's other articles in the field
4	Author's expertise
5	Stating the aim in the article
6	Stating the methodology in the article
7	Count of high-quality references
8	Stating the limits of the work
9	Covering a specific question in the article
10	Referring to a particular population
11	Balance of the author's article
12	Objectivity and subjectivity
13	Self-advertising or the amount of bias in the work
14	Data that support the conclusions
15	Visible date in the article
16	Linking to, or discussion about other gray literature
17	Unique or enriching research
18	Strengthening or weakening of the current position
19	Number of backlinks
20	Number of social media shares
21	Number of comments
22	The credibility of the work (blog, video, thesis, article, etc.)

Table 3.2: The data extraction format, which was used in the data extraction process of this study.

Article ID
Concept
Experience, opinion, or thought

the article was

$$\frac{\text{Sum of all metric values}}{\text{Total number of metrics}}$$

For each gray literature article candidate, we used the above formula to calculate a final score between $[0.0, 1.0]$. A score of ≥ 0.50 was required for the gray literature article to pass the validation. Not all articles contained personal experiences, opinions, or thoughts or were relevant to the topic or passed the gray literature quality assessment process.

After we performed the search on the four websites and going through the results, a total of 42 articles were selected in the initial phase. When we filtered out the articles, which did not contain experiences, opinions, or thoughts, we were left with 26 articles. Six out of 26 articles did not pass the gray literature quality assessment process, so we were left with 20 articles to be used in the gray literature review. The articles selected for the gray literature review are listed in Table 3.3.

3.2.3 Extracting Data

As the approach of this study was to conduct a concept-centric literature review, we call each unit of data a concept (Webster and Watson, 2002). The format of the concept used in this thesis is shown in Table 3.2.

To extract data from the articles, each article was carefully read through and key points made by the article's author(s) were written down. After the reading process was complete, we grouped the key points by their category and made a note, if the key point is expressed as a reason, benefit, or challenge. The final results were compiled into a concept matrix, which shows each concept and the corresponding articles where they were mentioned in.

3.3 Case Study

In the second part of this research, we conducted a case study to compare how the gray literature review and the developers from the case study tell us about the benefits and

Table 3.3: List of articles selected to be used in the gray literature review.

Identifier	Article
A 1	Why Everyone is Talking About Isomorphic / Universal JavaScript and Why it Matters (March 21, 2016) https://medium.com/capital-one-tech/why-everyone-is-talking-about-isomorphic-universal-javascript-and-why-it-matters-38c07c87905
A 2	Isomorphic JavaScript: The Future of Web Apps (November 11, 2013) https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc
A 3	Isomorphic (Universal) JavaScript (January 29, 2017) https://medium.com/commencis/isomorphic-universal-javascript-496dc8c4341a
A 4	Why You'll Always Need Isomorphic JavaScript (July 9, 2016) https://medium.com/@asyncmax/why-youll-always-need-isomorphic-javascript-bd310596d203
A 5	Next-gen Web Apps with Universal JavaScript (May 1, 2015) https://medium.com/@ghengeveld/on-the-road-to-isomorphism-eb0742a9305f
A 6	An Introduction to Isomorphic Web Application Architecture (November 29, 2016) https://medium.com/@ElyseKoGo/an-introduction-to-isomorphic-web-application-architecture-a8c81c42f59
A 7	Isomorphic Web Applications (July 9, 2020) https://medium.com/@elavarasi/isomorphic-web-applications-5fbd3118eba9
A 8	Why and How Coursera Does Isomorphic Javascript: A Fast and Snappy Quiz (August 18, 2015) https://medium.com/coursera-engineering/why-and-how-coursera-does-isomorphic-javascript-a-fast-and-snappy-quiz-a42acdd59ef8
A 9	JavaScript SEO: Server Side Rendering vs. Client Side Rendering (July 6, 2018) https://medium.com/@benjburkholder/javascript-seo-server-side-rendering-vs-client-side-rendering-bc06b8ca2383
A 10	What on earth is an isomorphic application? (January 21, 2018) https://medium.com/@christianiacullo/what-on-earth-is-an-isomorphic-application-cd13d8fb87d5
A 11	Client Side Rendering Vs Server Side Rendering in React.js & Next js (April 6, 2020) https://yudhajitadhikary.medium.com/client-side-rendering-vs-server-side-rendering-in-react-js-next-js-b74b909c7c51
A 12	Why do we love Isomorphic/Universal rendering? (September 28, 2019) https://medium.com/@sujankanwar/why-do-we-love-isomorphic-universal-rendering-988c22933933
A 13	You Might Not Need Server Side Rendering (January 30, 2019) https://hackernoon.com/you-might-not-need-server-side-rendering-f2681e02e4e
A 14	An Overview of Server Side and Isomorphic Async Rendering (February 23, 2020) https://hackernoon.com/an-overview-of-server-side-and-isomorphic-async-rendering-hr743bsa
A 15	Isomorphic JavaScript is Dead... Long Live Isomorphic JavaScript! (December 12, 2017) https://hackernoon.com/isomorphic-javascript-is-dead-long-live-isomorphic-javascript-743e1b7b181c
A 16	Break Down Isomorphic and Universal Boilerplate: React-Redux server rendering (March 29, 2017) https://hackernoon.com/isomorphic-universal-boilerplate-react-redux-server-rendering-tutorial-example-webpack-compenent-6e22106ae285
A 17	The benefits and origins of Server Side Rendering (January 20, 2019) https://dev.to/sunnysingh/the-benefits-and-origins-of-server-side-rendering-4doh
A 18	Isomorphic JavaScript: What is it and what can I do with it? (August 29, 2019) https://dev.to/roelofjanelsinga/isomorphic-javascript-what-is-it-and-what-can-i-do-with-it-3mkp
A 19	Why you should render React on the server side (July 8, 2019) https://dev.to/bnevilleoneill/why-you-should-render-react-on-the-server-side-hb3
A 20	Server-side rendering (SSR) for an SPA project (July 17, 2020) https://dev.to/daiyanze/server-side-rendering-ssr-for-an-spa-project-2ogl

challenges of isomorphism in single-page web applications. A group of developers from a Finnish technology consulting company were selected to participate in this study. The five developers who participated in this study are listed in Table 3.5.

3.3.1 Case Description

To search for developers to take part in the case study, we sent an interview invitation to the Finnish technology consulting company's internal message board. In the message, we shortly described the thesis, its goals, and our research approach for the case study. The choosing criteria for taking part in the interview was that the developer had worked with isomorphism and isomorphic web applications before. We did not narrow down the developers by their experience of the technology. After a while, we contacted the developers who had expressed their interest in taking part in the study and scheduled the video interviews.

3.3.2 Data Collection and Units of Analysis

The data was collected as a developer survey. The interview questions were formed in a way that covers both research questions of this paper. The questions selected for the first interview are listed in Table 3.4.

After the first interview was held, we noticed some overlap in the questions. For example, when asking about the benefits of isomorphic web applications, the same answer could be applied to the reasons for adopting isomorphic web applications as well. Because of this, we removed question **#10** from the interview questions.

To get a better understanding of the downsides of isomorphic web applications, we added a question "What are the downsides of isomorphic web applications?". Questions **#7** and **#8** were asked from both the developer's and end user's points of view.

The final interview questions, which were used for developers **(I 2)** — **(I 5)** are shown in Table 3.6.

3.3.3 Data Analysis

Each interview was held in English on the Google Meets -platform and recorded for transcribing. Before we started recording the interview, consent from the developer was asked.

Table 3.4: Initial developer interview questions.

#	Question
1	When did you first hear about isomorphic web applications?
2	How did you first learn about isomorphic web applications?
3	How much experience do you have in developing isomorphic web applications?
4	Describe isomorphic web applications with your own terms.
5	How much experience do you have in developing isomorphic web applications?
6	Describe isomorphic web applications with your own terms.
7	What would be the main challenges in transitioning to isomorphic web applications?
8	What do you feel about isomorphic web applications overall?
9	What are the benefits of isomorphic web applications?
10	What are the reasons for adopting isomorphic web application architecture?
11	What are the challenges of developing isomorphic web applications?
12	What is the developer experience of developing and working with isomorphic web applications?
13	What effects have isomorphic web applications had?
14	Have there been any surprises?
15	Would you develop isomorphic web applications in another case?
16	Is there a topic that was not addressed in the questions?

We also gave an introduction about how and where the interview will be used.

Questions **Q5** — **Q11** were the main interview questions that focused on answering the two research questions of this thesis.

In addition, questions **Q1** — **Q4** and **Q12** — **Q13** from the final interview questions were used to get an overview of the developers' experience and a general feeling of working with isomorphic web applications. The last interview question (**Q14**) was asked at the end of the interview to catch any topics the developers would have liked to discuss in more detail. This question also allowed us to ask question-specific things, if we felt like we wanted to

Table 3.5: List of developers who were interviewed in the second phase of the study.

ID	Role	Experience working with isomorphic web applications
I 1	Developer	4.5 years
I 2	Developer	3 years
I 3	Developer	5.5 years
I 4	Developer	6 years
I 5	Developer	6 years

Table 3.6: The final interview questions, which were used in the last four interviews.

#	Question
Q 1	When did you first hear about isomorphic web applications?
Q 2	How did you first learn about isomorphic web applications?
Q 3	How much experience do you have about developing isomorphic web applications?
Q 4	Please describe isomorphic web applications with your own terms.
Q 5	What would be the main challenges in transitioning to isomorphism from an existing application?
Q 6	What other challenges exist in developing isomorphic web applications?
Q 7	What are the benefits of isomorphic web applications?
Q 8	What are the downsides of isomorphic web applications?
Q 9	What is the developer experience of developing and working with isomorphic web applications?
Q 10	Have there been any surprises?
Q 11	What effects have isomorphic web applications had?
Q 12	What do you feel about isomorphic web applications overall?
Q 13	Would you develop isomorphic web applications in another case?
Q 14	Is there a topic that was not addressed in the questions?

know more about specific topics.

At the end of every interview session, the developers had time to give feedback about the interview. After the interview session was over, we transcribed the video interviews into English. We then sent the transcribed interview text to the developers for them to check for any spelling or contextual errors, which might have occurred when transcribing the video interview.

Some answers were refined in another format, which was more suitable for this thesis's context. In some parts of the interviews, we asked focusing questions because there was a lot of variation in the developers' answers and their formats.

4 Results

The results of both the gray literature review and the case study are included in this chapter. First, we go through the gray literature review. Then, we go through the results of the case study. Finally, we compare the results of the gray literature review and the developer survey.

4.1 Gray Literature Review

In this section, we show the results of the gray literature review. We answer the two research questions, which we introduced in Section 3.1.

4.1.1 What are the benefits of isomorphic web applications?

16 articles out of 20 mentioned that isomorphism in single-page web applications **improves the performance of the application**. This was one of the most observed benefits in this gray literature review.

> "Pages are served as rendered and faster initial page loading (less JavaScript files and file sizes)" (A3)

One article stated that the overall performance of the application is still improved, although the initial page response is slower:

> "Although an isomorphic application will be slower in responding an initial page compared to a client rendering application (as it fetches data and renders more markups on the server side before responding), overall completion time to the final user interrace is faster because data access on the server side is much faster than client-side API requests." (A4)

In addition to rendering the page server-side, a method called "above-the-fold stylesheets" was mentioned as one method of improving the performance of the application even more:

> "To achieve better initial page load times your best option is to render the application on the server. This way you can simply have the server

> return the fully compiled initial view as an HTML document, so a user doesn't have to wait for the JavaScript to download and execute before displaying useful content. We can even enhance this further by inlining above-the-fold stylesheets (so called Critical CSS) so we don't have to wait for CSS files to load" (A5)

Improved SEO was the other most mentioned benefit, with 16 articles out of 20 mentioning it.

Googlebot [14] has an isolated environment for network requests, which is a reason why isomorphic web applications improve SEO:

> "In order to index content that JavaScript apps render on client-side, Googlebot must run those JavaScript apps inside a full browser environment and capture the rendered DOM. This involves complex browser compatibility issues. Because JavaScript apps can make AJAX requests for further rendering, Googlebot must have a policy that controls apps' network access. These are why getting your JavaScript app indexed properly by Googlebot is still challenging." (A4)

In case the search crawlers can crawl the page directly, it also leads to improved SEO:

> "If the server renders the whole page, search engine crawlers won't have to resort to executing JavaScript but can crawl your pages directly. This should drastically improve your PageRank." (A5)

For some search crawlers, the crawlers run their own version of the browser engine. In case of isomorphic web applications, it reduces the amount of polyfills the application is required to send in the JavaScript bundle:

> "Search crawlers are now able to parse and execute JavaScript. Googlebot, in particular, is a PhantomJS [15] script that runs as Chrome 41. This is why SSR / Isomorphic makes sense since you won't be sending down the unnecessary polyfills to all your users just so that Google bot can render your page." (A12)

Better user experience was mentioned in nine articles, being the third most mentioned benefit of isomorphic web applications:

[14] https://developers.google.com/search/docs/advanced/crawling/googlebot
[15] https://phantomjs.org/

> "By rendering important parts of the application with the real data on the server side, an isomorphic application can show a meaningful initial page. On the other hand, client rendering application can't show any meaningful information until it fetches all external data it needs. In the meantime, only thing a user will see is a loading indicator." (A4)

Along with improved performance and search engine optimization, **code sharing** was the fourth most common benefit mentioned six times in the gray literature review:

> "...using the same code on both platforms. This is commonly referred to as Universal JavaScript and can be achieved by creating a set of abstractions that decouple our application logic from the underlying implementation. Effectively you can write your application logic in such a way that it won't matter if it runs in the browser or on the server." (A5)

An isomorphic web application is **easier to maintain**, a benefit mentioned in four articles:

> "Having the same library on both the server and browser allows for better development and code reuse, which leads to happier software engineers and less time spent maintaining the code." (A1)

Only one article stated that isomorphic web applications have better **old device support** than traditional SPA:

> "As rendered pages are served, it is not a problem to support old devices" (A3)

4.1.2 What are the challenges of adopting isomorphism in single-page applications?

In the 20 articles we reviewed, there were much fewer challenges than benefits mentioned in the articles.

Increased complexity was mentioned in nine out of 20 articles stating it as a challenge of isomorphic web application development.

Both the browser and the server have to execute the same code identically:

> "In order to build a Universal JavaScript application, each part of the stack has to support it. The challenge lies in getting the same code to

Table 4.1: The concept matrix, which mentions the benefits and challenges of developing isomorphic web applications. The gray background separates the challenges from the benefits.

Article	Improved performance	Improved SEO	Better user experience	Code sharing	Easier to maintain	Old device support	Increased complexity	Harder to maintain	Steep learning curve	Slower initial response	Increased server load	Code duplication	Bundling assets is harder	State management
A1		+	+	+	+									
A2	+	+			+		−					−		
A3	+	+	+	+	+	+	−							
A4	+	+	+							−				
A5	+	+		+			−							
A6	+						−	−						−
A7	+	+			+									
A8	+													
A9	+	+					−							
A10	+	+		+			−							
A11	+	+												
A12		+	+							−	−			
A13	+	+					−							
A14	+		+										−	
A15	+		+						−					
A16	+	+						−	−					
A17	+	+	+	+					−					
A18	+	+	+				−							
A19		+	+	+										
A20		+					−	−						
Total	16	16	9	6	4	1	9	3	3	2	1	1	1	1

> execute identically in the browser context and in Node.js. While the Node environment is quite flexible and supports most of what you'd expect of a back-end system, the browser is a much more restricted environment. To achieve isomorphism in terms of limiting code duplication, we need wrapper libraries so we can use the same API to talk to both the browser and Node.js" (A5)

One article stated that some application code will end up duplicated:

> "While the ideal case can lead to a nice, clean separation of concerns, inevitably some bits of application logic or view logic end up duplicated between client and server, often in different languages" (A2)

Implementing isomorphism requires additional logic, which increases the complexity of the application:

> "Lots of logic should be handled in server-side. Handling HTTP requests, routing, rendering, styling and module loading can be difficult and external libraries would be needed in server-side rendering. They should be handled in the same way both in server-side and client-side." (A3)

Three articles stated that an isomorphic web application is **harder to maintain**. The increased amount of source code was one reason, which makes isomorphic web applications harder to maintain:

> "It's possible to server-render your site with Java or Ruby and then transition to a single page app, but it isn't commonly done because it requires duplicating large portions of code in two languages. This has a high cost in the maintenance of an app." (A6)

In another article, isomorphic web applications ended up causing more problems than solutions:

> "I had to grow and maintain a large codebase over time. In many cases, the "tricks" I used to make my code "isomorphic" ended up causing more problems than they solved." (A15)

Adopting SSR made the maintenance of the application more time-consuming:

> "Adopting SSR means that we are adding a new service layer into front-end clusters. The position could be right after the proxy server and in

> front of our Representational State Transfer (REST) API servers. This made the architecture a bit more complex and the maintenance a bit time-consuming." (A20)

The **steep learning curve** of developing isomorphic web applications was mentioned in three articles:

> "...the learning curve is distressingly steep" (A16)

Two sources stated that isomorphic web applications have a **slower initial response**. In one article, Time To First Byte (TTFB) is said to be higher, but the perceived page load is faster:

> "TTFB is higher though perceived page load time is faster." (A12)

When pre-rendering applications on the server, it leads to **increased server load**. This was mentioned by a single article:

> "In Isomorphic / SSR with React.js, the server throughput impact is large. renderToString is synchronous Central Processing Unit (CPU) bound-call, which holds the event loop, which means the server will not be able to process any other request till the function call completes." (A12)

Code duplication was a challenge mentioned only by one article:

> "While the ideal case can lead to a nice, clean separation of concerns, inevitably some bits of application logic or view logic end up duplicated between client and server, often in different languages" (A2)

Concerns regarding asset bundling in isomorphic web applications were mentioned in two articles, saying that **bundling assets is harder**:

> "In using Webpack, achieving "parity" between server and client-side bundles can get messy." (A13)

State management is important for the isomorphic web application to function properly. It was mentioned by one article:

> "On the server, you save the state of the application and then provide this state to the browser. The browser uses this state to bootstrap the SPA version of the application. Without this isomorphic handoff,

> the user must wait for the server-rendered page to load, and then wait longer for a complete re-render of the content in the browser." (A6)

4.2 Case Study

The results of the case study are shown in this section. We apply the same result formalization from the gray literature review, which was presented above.

4.2.1 How the developers got familiar with isomorphic web applications

Most of the interviewed developers have been in a project that used isomorphism. One developer told that they first heard about isomorphism when they were looking to improve the performance of their application:

> "As the application grew, we were looking at how to improve performance for the initial load. It involved a great deal of data and showing a lot of it. I found out that it's possible to render the whole thing on the back-end, just serve the HTML and later on tell React.js to hook into this data" (I4)

One developer discovered isomorphism when they were going through the application's code:

> "I was going through code and I came across a solution, which I hadn't seen before." (I2)

Blog posts were the source for one developer where they first heard about isomorphism:

> "I didn't know about isomorphism until I saw a blog post about it." (I3)

4.2.2 How much experience the developers have about isomorphic web applications

All developers who we interviewed have years of development experience in isomorphic web applications. The experience each developer has about isomorphic web applications is listed above in Table 3.5.

4.2.3 How the developers described isomorphic web applications

Developers gave different definitions for isomorphic web applications. One developer answered that isomorphic web applications are about reusing the same code between the server and the client:

> "It's an approach where you reuse the same rendering code on the client-side and the server-side." (I1)

One developer stated that isomorphic web applications are applications that are not tied to the browser environment:

> "It's more about having the application not being tied into the browser. You have an application that can run without the browser (server-side) and it produces a state that you can transfer somewhere else and hydrate it." (I3)

4.2.4 Benefits of isomorphic web applications

The improved performance was mentioned by all five developers. Time To Interactive (TTI) was mentioned as one metric for measuring improved performance:

> "We monitored and saw an immediate performance increase. TTI dropped dramatically when we shipped ready HTML." (I4)

There was one mention that performance is given with isomorphic web applications:

> "Performance is kind of given with isomorphism." (I1)

Improved search engine optimization is one of the benefits of isomorphic web applications that was mentioned by all five developers:

> "All kinds of crawlers will have an easier time crawling what's on your page. We know Googlebot does JavaScript execution, but other bots like Facebook and Twitter will include a card to show a preview of the page. These crawlers don't support JavaScript execution. They see the content if the page is server-side rendered." (I5)

In addition, one developer mentioned that SSR was initially used as a SEO tool:

> "Initially, this was used as a search engine optimization tool, where you have the content on the page as the crawlers could access the dynamically/ asynchronously loaded content" (I4)

The **improved user experience** that isomorphic web applications give was mentioned by four developers. One developer stated that a SPA tends to be slow when the page is loaded for the first time:

> "It's the snappiness. You get directly to a useful state. Sadly quite a lot of SPAs end up being a bit janky in the start." (I3)

Flickering of the UI was mentioned to be a problem that isomorphic web applications solve:

> "There was no flickering in the UI. Getting back to the good old days of HTML-only when you have complete web pages served rather than a shell with five spinners." (I4)

Code sharing was mentioned by two developers. One developer didn't directly say it is a benefit, but rather a test that you can share code between the server and the client:

> "One of these advantages is that it is basically a check that are you able to share code efficiently between these projects." (I4)

One developer stated that if you share code, it simplifies the changes needed to implement isomorphic web applications:

> "The good side of doing an isomorphic application is that if you are using the exact same rendering code on both sides, it simplifies some of the changes." (I2)

The **easier maintainability** of isomorphic web applications was stated as a benefit by two developers. However, it requires the whole application to be set up, before actual maintainability improvements can be observed:

> "Once you have set up the isomorphic web application and the whole team recognized this is the way, it is easier to maintain than a regular SPA." (I4)

4.2.5 Downsides and challenges of isomorphic web applications

Framework and library support for building isomorphic web applications was mentioned by all five developers. For example, the developers claimed that there is a lot of

boilerplate around isomorphism, and it is up to the development team how much work they are willing to do to implement isomorphism:

> "In React.js, you are left very much on your own; with other frameworks as well. There is a lot of boilerplate you have to do, and there is a lot of gotchas you have to know." (I5)

Libraries that are installed using Node.js Package Manager (NPM) [16] might be incompatible in an isomorphic environment:

> "When installing Node.js packages, you might end up with code that's not suitable for an isomorphic environment" (I3)

The isomorphic web application architecture can be **hard to understand**. This fact was mentioned by three developers:

> "Many developers did not fully understand it. They did not understand what it actually means." (I4)

Increased complexity was mentioned by three developers. For instance, Developer 4 mentioned that isomorphic web applications introduce additional build tooling configuration:

> "There is additional build tooling configuration required to set it up." (I4)

State management was mentioned by three developers. It is challenging to keep the state in sync between the server and the browser, and how the state is transferred to the client:

> "State synchronization is a big challenge; to be able to maintain the initial rendering state from the server to the client without any issues.
>
> You need to somehow fetch the initial state of the app from the back-end, and transport that to the client-side for hydration. You need to make sure that the states don't get out of sync in the meantime." " (I5)

Data fetching was mentioned by two developers. If the data fetching code is shared between the server and the client, the developer needs to make sure it works on both server and client environments:

[16] https://www.npmjs.com/

> "I earlier mentioned data loading. That can get more complex because the rendering code is shared between the server and the client. All the supporting code often might not be shared, so you would use different code for fetching data in the initial load than the rest of the application." (I1)

Additional requirements are a challenge mentioned by two developers. For instance, there could be some implementation-specific behavior on different browsers:

> "When serving an isomorphic web application with Node, you are using Google's open-source high-performance JavaScript and WebAssembly engine (V8) as the JavaScript engine. With Safari, you might be using JavaScriptCore on the client-side. Different JavaScript engines have different implementations, and there is some implementation specific behavior that you can observe - especially when sorting arrays. Some operations happen in a different order, and if you use only Chrome and always use Node, you will never notice as they both use V8. Some people are using Safari a lot, especially on the mobile use case. Many developers are in a Chrome monoculture, and they fail to understand that the JavaScript code will be executed by a different engine." (I4)

In addition to the browser-specific behavior, extra data transfer was mentioned by two developers:

> "Over a Server-side rendered app there's always going to be extra data transfer because you have the JavaScript that needs to be downloaded. You have usually the JavaScript Object Notation (JSON) blob of initial data that has been generated on the back-end that needs to be transferred alongside HTML." (I5)

If the user has strict data caps, isomorphic web applications might not be ideal for them:

> "If we lived in an environment with very strict data caps; first you download the pre-rendered stuff, and then you are executing all the JavaScript code again after the initial load. We are basically serving the same data twice. If you really had to monitor your data usage, this could be something. Caching can solve this issue." (I4)

The choice of **back-end language** was mentioned by two developers. JavaScript is one of

the scripting languages that can be run both on the browser and server. There also exists some other web-oriented programming languages, which can be transpiled into JavaScript:

> "If you want to create an isomorphic application, you either have to fully use JavaScript, so you have a Node.js back-end and JavaScript front-end, or you use some language that can be compiled to JavaScript." (I5)

Code duplication was mentioned to happen if the server's and the client's programming language was not the same:

> "If you go with a different kind of back-end (for example Python), then you have duplicate code because you have to have Django [17] templates and the React.js templates for the front-end. Those have to match." (I5)

4.2.6 Surprises of isomorphic web application development

The developers shared both positive and negative surprises when developing isomorphic web applications. Library support was a negative surprise told by one developer:

> "Library support has been a negative surprise. In the React.js ecosystem, this has been a supported feature for five to six years, maybe even longer. There are still many popular libraries that don't support server-side rendering at all. It causes problems, and you have to build things around it to make them work. You have to defer some components to make them run only on the client-side. You cannot work around these issues with simple if-statements because the outputs of server and client-side rendered pages have to match exactly to ensure that things work. Those are some negative surprises." (I1)

One developer shared that isomorphic code was hard to understand in the beginning:

> "In the beginning, it was hard to understand that the code can be run on the server and the client. I sometimes stumble on it when developing isomorphic web applications." (I2)

One developer told in the interview that there was only little information about isomorphic web applications available:

[17] https://www.djangoproject.com/

> "I was surprised that there was so very little information. I kept Googling for information, and I kept ending up with the same contrive, simple solutions" (I3)

The mindset of developers was surprising for one interviewed developer:

> "The resistance. Other developers might disagree and say that this complicates things." (I4)

4.2.7 Effects of isomorphic web applications for the developers

Isomorphic web applications complicate the development process, stated by one developer:

> "It can complicate some aspects of the development because you have to consider the code running in both client and server side." (I1)

The introduction of isomorphic web applications have lead to the development of isomorphic web application frameworks, such as Next.js [18]:

> "There are good frameworks for building isomorphic web applications, take Next.js for example." (I2)

Isomorphic web applications can be tricky for developers, who have worked only on the front-end side of the technology:

> "It can come off as cumbersome if you're only worked with React.js front-end heavy applications.
>
> Development might take longer for some things" (I3)

4.2.8 The general feeling about isomorphic web applications

The general feeling about isomorphic web applications was very positive. The first developer who was interviewed thought isomorphism about "fixing a problem that shouldn't exist in the first place":

> "They are a good thing that they exist, but they are fixing a problem that shouldn't exist in the first place. Many of these problems are only present because of the way our current HTML and Web applications

[18] https://nextjs.org/

> work and how they are designed. How they are "hacked" around the DOM" (I1)

The increased needs of web applications were the general feeling of isomorphism for one developer:

> "It's something that growing web apps need to implement at some point. Rendering the whole DOM on the client is going to be slow, when the application grows in size" (I2)

4.2.9 Developing isomorphic web applications in another case

Almost every developer stated that they would develop isomorphic web applications when starting a new project or transforming an existing project to use isomorphism:

> "Yes. If I start a new project I usually go with an isomorphic web application setup" (I2)

One developer would develop an isomorphic web application to a certain point if the complexity does not overcome the benefits:

> "Maybe. For modern applications and certain use cases the initial load time is important to get down. Users don't want to wait too much for the page to load and for the page to become interactive. In e-commerce applications, I would pretty much always go with isomorphism. If the project is more like a management UI or an admin UI where performance matters, but the initial load doesn't matter (because the user usually goes to the page and spends a few minutes and then leaves), optimizing for initial load is in my opinion not worth the complexity." (I1)

4.3 Comparison of results

In this chapter, we compare the results of the gray literature review and the case study. The comparison table is presented in Table 4.3. To create the comparison table, we used Table 4.1 and Table 4.2 that contain the results for the two parts of the study. The comparison table includes all the key points found during the gray literature review and the case study.

Table 4.2: Results of the case study.

Developer	Improved performance	Improved SEO	Improved User experience (UX)	Code sharing	Easier maintainability	Framework and library support	Hard to understand	Increased complexity	State management	Data fetching	Additional requirements	Back-end language	Code duplication
I1	+	+	+	+	+	−	−	−		−		−	
I2	+	+				−	−	−	−				
I3	+	+	+			−			−	−			
I4	+	+	+	+	+	−	−				−		
I5	+	+	+			−		−	−		−	−	−
Total	5	5	4	2	2	5	3	3	3	2	2	2	1

We did not force the key concepts of either part of the study to match the table; the table contains all the key concepts, which were found during the study. If no match was found for both the gray literature review and the case study, we marked the section as empty.

Table 4.3: Comparison of the results of the gray literature review and developer interview.

Concept	Gray literature review	Case study
Performance	The performance of the application improves. Devices, which have a slower internet correction load the application faster. The TTFB of an isomorphic web application is higher because the pages are server-side rendered. Therefore, the initial response is slower.	The performance of the application improved, dropping TTI dramatically. However, the initial response might be slower. On the other hand, performance is given with isomorphism. Google's Lighthouse tool will detect isomorphic web application slowdowns easier than the user.
Search engine optimization (SEO)	Search engine optimization improves with isomorphism. Page crawlers have an easier time indexing the pages.	Page crawlers have an easier time indexing the pages, and crawlers from Facebook and Twitter correctly show a card preview of the page.
User experience (UX)	Isomorphic web applications have a better user experience because no initial loading indicators or blank page flickering are observed.	The UI is generally faster and no blank page flickering is observed.
Code sharing and duplication	Code can be shared in an isomorphic web application, but using isomorphism can also introduce duplicate code.	Code sharing simplifies the changes needed to implement isomorphic web application architecture. Using a different programming language between environments leads to duplicate code.
Maintainability	Maintainability is improved because code can be reused and the server and the client code is separated.	Once the application is set up, it is easier to maintain than a regular SPA.
Complexity	Code complexity is increased, as the browser and the server have to execute the same code identically. Additional infrastructure requirements are also needed.	Additional build tooling configuration is needed.
Back-end language	No mention.	Using a different programming language between the server and the client makes the code more complex.
Framework and library support	It is a challenge to execute the same code in both environments. To be able to use the same library between two different environments, a wrapper library might be needed.	Some functionality needs to be polyfilled. Not all libraries support server-side rendering out of the box. Isomorphism adds a lot of boilerplate code.
State management	An "isomorphic handoff" needs to be done, which complicates things.	State management is challenging with isomorphism because it has to be kept in sync between the server and the client.
Developer experience	Isomorphic web application architecture has a steep learning curve.	Isomorphic web applications are hard to understand in the beginning and require more experience than traditional SPAs.

5 Discussion

In this thesis, we studied the benefits and challenges of isomorphism in single-page web applications. We first go through the research questions and list the key points we found out while conducting this study. Then, we discuss each topic a bit more in detail.

In the first research question, we asked about the benefits of isomorphic web applications. From the gray literature review and the case study, we found out that the most mentioned benefit was **improved performance**. The second most mentioned benefit was **improved SEO**. The increased performance of the application leads to **better UX**. Fourth on the list was **code sharing**, which makes the application **easier to maintain**. The least mentioned benefit of all was **old device support**.

The second research question asked about the challenges of adopting isomorphism in single-page applications. **Increased complexity** was mentioned the most. Isomorphic web applications can be **hard to understand** and the **framework and library support** of isomorphic web applications is something the developers need to address. Even though isomorphic web applications are said to have increased performance, **a slower initial response** might be observed, because the server needs to pre-render the page. Pre-rendering the page server-side causes **increased server load**. The state of the isomorphic web application needs to be synchronized between the server and the client, so **state management** opposes a challenge.

5.1 How isomorphic web applications affect the end-users?

Increased performance was mentioned many times, both in the gray literature review and the case study. The increased performance of the application was observed in the TTI of the application. However, the TTFB increases with isomorphism because the server has to pre-render the page. This can be solved with caching, but it adds additional complexity to the application. UX is improved with isomorphism because no blank page flickering or loading indicators can be observed in the UI. On devices with slower internet connections, the increased performance of the application can be noticed more easily.

5.2 What does isomorphism bring for developers?

SEO is a benefit of isomorphic web applications for developers, mentioned in both parts of the study. Even though most search crawlers can index modern web pages that use JavaScript, there are certain limitations on the crawling process. The environment where the search crawlers run on is limited, which means polyfills might be needed to be able to crawl the page correctly. Crawlers that generate previews for Facebook and Twitter work well with isomorphic web applications because the HTML meta tags are inserted into the page in the first response.

Code sharing and duplication is a topic that the articles and developers had many opinions about. The application code can be shared in an isomorphic web application, but code can still get duplicated in some scenarios, for example, when choosing a different programming language for the server of the application.

The ability to re-use code between environments improves the maintainability of isomorphic web applications. The maintainability benefit can be observed after the application has been set up.

Isomorphic web applications introduce additional boilerplate to the application. Because the same code must work on both the server and the client environments, additional abstraction and polyfills might be required. Additional server infrastructure is needed because requests to isomorphic web applications must be handled by a server. Google Lighthouse [19] is a tool that developers can use to measure the speed of their isomorphic web application.

In most of the articles and mentioned by developers, JavaScript is the most popular programming language when talking about isomorphic web applications. It is beneficial to choose the same programming language for the server and the client to be able to share code. However, when choosing a different programming language for the server, it will make the application's code more complex.

Framework and library support was a challenge that was mentioned many times during the gray literature review and the case study. Not all libraries support running the same code in the server and the client environments. Additionally, writing extra code and creating polyfills to make certain libraries function in an isomorphic environment might be mandatory. The differing environments, where the server and the client run, can in the

[19] https://developers.google.com/web/tools/lighthouse

worst-case scenario introduce browser compatibility issues, bugs, and inconsistencies.

The application's state in isomorphic web applications must be kept in sync between the server and the client. In the gray literature review, this process was called the "isomorphic handoff". If the states between the server and the client are mismatched, flickering or weird behavior can be observed.

The developer experience mentioned in the articles and by the developers was positive. All interviewed developers would develop isomorphic web applications in another case (for example, when starting a new project). In most cases, the benefits of isomorphism give outweigh the challenges the technology has. With careful planning, the challenges can be minimized.

5.3 Validity of the results

We chose to use gray literature in this thesis because there is not that much scientific literature available about isomorphic web applications. Most of the resources about isomorphic web applications that exist on the internet are blog posts. By definition, low credibility sources include blogs, emails, and tweets (Garousi et al., 2019). The blog posts, which we selected as candidates in the literature review, are either implementation guides or experiences working with isomorphism.

We took great caution in the process of choosing the gray literature for this thesis. First, we identified any existing reviews. Then, we proceeded with the planning of the gray literature review and started to construct our research questions. The research questions we chose met our requirements for the target audience and review goals. With the two selected research questions, we were able to compare the results of the gray literature review and the case study. The two research questions, which we chose, belong to the exploratory research question category (Garousi et al., 2019). To be able to use the internet articles in the gray literature review, we conducted a gray literature quality assessment (Garousi et al., 2019). Based on the results of the quality assessment, we formed the final list of gray literature and could continue on the literature review process.

Both in the gray literature review and the case study, we assumed that the results are positively biased. The gray literature review resulted in a total of 52 positive and 21 negative points for isomorphic web applications, which supports our theory that gray literature might be positively biased. On the other hand, the results of the case study

were much more balanced; a total of 18 positive and 21 negative points were observed in the case study. The possible reason why we see a much more balanced outcome in the case study might be because we interviewed only five developers. With a small sample size, not all possible answers and opinions are taken into count.

6 Conclusions

In this thesis, we studied the benefits and challenges of isomorphism in single-page applications. To study these benefits and challenges, we conducted a two-part study, which included a gray literature review and a case study. A total of 20 gray literature articles were reviewed. We also interviewed five developers, who have experience working with isomorphic web applications before. The interviews were transcribed and a comparison to the gray literature review was made.

The results of this thesis show that isomorphic web applications have benefits for both the end-users and the developers, bringing trade-offs that the developers must be aware of. The downsides and challenges of isomorphic web applications mostly focus on the developer side of the technology.

The benefit of isomorphic web applications, which we observed, was the increased performance of the application. The TTI of the application is reduced, but because the page needs to be rendered server-side, TTFB is increased. Compared to SPAs, an isomorphic web application has a better UX, because it does not have loading spinners or blank-page flickering. Isomorphic web applications also support older devices, which do not have JavaScript support. The isomorphic web application architecture improves the SEO of the application, making all kinds of search crawlers crawl the content of the page correctly. The developers can reuse some application code between the server and the client, to increase the maintainability of the application. In an isomorphic web application, the application infrastructure is more complex because the server pre-rendering the pages has to be set up. Different browsers have different kinds of environments they use, so developers might have to polyfill browser-dependent features to make the application work across different versions of web browsers. Some libraries do not work with isomorphism out of the box, which requires additional work for the libraries to be functional in an isomorphic environment. State management is a big challenge of isomorphism because the state needs to be kept in sync between the server and the client.

The main motivation to research more about isomorphic web applications is the lack of scientific literature around it. For further research, it would be interesting to see how a specific implementation of isomorphic web applications would solve the challenges of isomorphic web application development that we found out when conducting this study.

Bibliography

Airbnb (Oct. 2012). *Render your Backbone.js apps on the client and the server, using Node.js*. [Accessed 5th May 2021]. URL: https://github.com/rendrjs/rendr.

Barbar, A. and Ismail, A. (2019). "Search Engine Optimization (SEO) for Websites". In: *Proceedings of the 2019 5th International Conference on Computer and Technology Applications*. ICCTA 2019. Istanbul, Turkey: Association for Computing Machinery, pp. 51–55. ISBN: 9781450371810. DOI: 10.1145/3323933.3324072. URL: https://doi-org.libproxy.helsinki.fi/10.1145/3323933.3324072.

Birdeau, L., Yeh, C., Peachey, M., and Hakman, K. (2002). "Delivery of data and formatting information to allow client-side manipulation". U.S. patent 8136109B1.

Cui, M. and Hu, S. (2011). "Search Engine Optimization Research for Website Promotion". In: *2011 International Conference of Information Technology, Computer Engineering and Management Sciences*. Vol. 4, pp. 100–103. DOI: 10.1109/ICM.2011.308.

Facebook Inc. (2013). *React – A JavaScript library for building user interfaces*. [Accessed 6th Feb 2021]. URL: https://reactjs.org/.

Garousi, V., Felderer, M., and Mäntylä, M. V. (2019). "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering". In: *Information and Software Technology* 106, pp. 101–121. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.09.006. URL: https://www.sciencedirect.com/science/article/pii/S0950584918301939.

Gordon, E. (2018). *Isomorphic Web Applications: Universal Development with React*. Manning Publications. ISBN: 9781617294396. URL: https://books.google.fi/books?id=DvThAQAACAAJ.

Killoran, J. B. (2013). "How to Use Search Engine Optimization Techniques to Increase Website Visibility". In: *IEEE Transactions on Professional Communication* 56.1, pp. 50–66. DOI: 10.1109/TPC.2012.2237255.

Mikowski, M. S. and Powell, J. C. (2014). *Single page web applications: JavaScript end-to-end*. Manning.

Mukkamala, K. (Sept. 2018). *A comparison of Server Side Rendering in React and Angular applications*. [Accessed 5th May 2021]. URL: https://levelup.gitconnected.com/a-comparison-of-server-side-rendering-in-react-and-angular-applications-fb95285fb716.

NR, J. (June 2015). *What is an isomorphic application?* [Accessed 2nd Feb 2021]. URL: https://www.lullabot.com/articles/what-is-an-isomorphic-application.

Paulson, L. D. (2005). "Building rich web applications with Ajax". In: *Computer* 38.10, pp. 14–17. DOI: 10.1109/MC.2005.330.

Rauschmayer, A. (Aug. 2015). *Is "Isomorphic JavaScript" a good term?* [Accessed 16th Apr 2021]. URL: https://2ality.com/2015/08/isomorphic-javascript.html.

React vs Vue vs Angular (2021). [Accessed 5th May 2021]. URL: https://www.npmtrends.com/react-vs-vue-vs-angular.

Scott, E. (2015). *SPA Design and Architecture: Understanding Single Page Web Applications*. 1st. USA: Manning Publications Co. ISBN: 1617292435.

Silva, W. O. da and Farah, P. R. (2018). "Characteristics and Performance Assessment of Approaches Pre-Rendering and Isomorphic Javascript as a Complement to SPA Architecture". In: *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse*. New York, NY, USA: Association for Computing Machinery, pp. 63–72. ISBN: 9781450365543. DOI: 10.1145/3267183.3267190. URL: https://doi-org.libproxy.helsinki.fi/10.1145/3267183.3267190.

Sons, K., Schlinkmann, C., Klein, F., Rubinstein, D., and Slusallek, P. (2013). "xml3d.js: Architecture of a Polyfill implementation of XML3D". In: *2013 6th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, pp. 17–24. DOI: 10.1109/SEARIS.2013.6798104.

Sun, Y. (2019). "Server-Side Rendering". In: *Practical Application Development with AppRun: Building Reliable, High-Performance Web Apps Using Elm-Inspired Architecture, Event Pub-Sub, and Components*. Berkeley, CA: Apress, pp. 191–217. ISBN: 978-1-4842-4069-4. DOI: 10.1007/978-1-4842-4069-4_9. URL: https://doi.org/10.1007/978-1-4842-4069-4_9.

Taivalsaari, A., Mikkonen, T., Ingalls, D., and Palacz, K. (2008). "Web Browser as an Application Platform". In: *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pp. 293–302. DOI: 10.1109/SEAA.2008.17.

Webster, J. and Watson, R. T. (June 2002). "Analyzing the Past to Prepare for the Future: Writing a Literature Review". In: *MIS Quarterly* 26.2, pp. xiii–xxiii. ISSN: 0276-7783.