Independent degree project - second cycle

Masters' thesis AV, 30 higher credits

Master of science in Computer Engineering

Computer Engineering

Isomorphic web applications
Depends on how you react

Eric Mathiasson
MID SWEDEN UNIVERSITY
Department of Information and Communication Systems

Examiner: Mikael Gidlund, mikael.gidlund@miun.se
Supervisor: Stefan Forsström, stefan.forsstrom@miun.se
Author: Eric Mathiasson, erma1007@student.miun.se

Degree programme: Master of science in Computer Engineering, 300 credits
Main field of study: Computer Engineering
Semester, year: VT, 2015
Abstract

Websites are today one of the most common ways for humans to interact with each other. Evolving from just static pages for companies to highly dynamic social medias, websites are built upon complex techniques and architectures to meet end-user demands. One of the latest techniques are single page applications which makes use of the web browsers capabilities to execute application code. One drawback is that search engines cannot visit the site, nor can script-disabled web browsers. For a single page application to function on all devices a new concept called isomorphism needs to be considered. The aim of this thesis have been to analyze, design and implement an isomorphic web application using modern technologies. The practical work have been applied using an agile project method, while theoretical works are based on previous knowledge and an internet based research. The implementation went through several iterations of trial and error but was finally completed with Facebook’s React framework together with a modified version of the Flux architecture. The resulting application was implemented in both Ember and the proposed isomorphic solution. Benchmarks of both implementations were carried out and compared with each other to see how the thesis’ isomorphic solution performed. The results shows an performance increase in the isomorphic solution. In conclusion the full application was not implemented due to time limits, but the most performance critical parts were. While only one framework was compared, it would be of interest to compare with more frameworks to give a better performance overview.
Acknowledgements

Thank you internet, Red Bull and vodka for making this thesis possible.
Table of Contents

Terminology ........................................................................................................................................ vi
Acronyms and abbreviations .................................................................................................................. vi

1 Introduction ........................................................................................................................................ 1
1.1 Background and problem motivation ................................................................................................. 1
1.2 Overall aim .......................................................................................................................................... 2
1.3 Concrete and verifiable goals .............................................................................................................. 3
1.4 Scope ................................................................................................................................................ 3
1.5 Outline .............................................................................................................................................. 3
1.6 Contributions ................................................................................................................................... 4

2 Theory ................................................................................................................................................. 5
2.1 Web development ................................................................................................................................. 5
2.1.1 HTML .......................................................................................................................................... 5
2.1.2 CSS ............................................................................................................................................ 7
2.1.3 JavaScript .................................................................................................................................. 7
2.1.4 Communication ............................................................................................................................. 8
2.1.5 JSON ........................................................................................................................................... 9
2.1.6 Dynamic languages ....................................................................................................................... 9
2.1.7 MVC frameworks .......................................................................................................................... 9
2.1.8 Responsive web design .................................................................................................................. 10
2.2 Single Page Applications ................................................................................................................... 11
2.2.1 Frameworks ................................................................................................................................. 12
2.2.2 Data services ............................................................................................................................... 12
2.3 Isomorphic web applications using React ......................................................................................... 13
2.3.1 Virtual DOM ............................................................................................................................... 14
2.3.2 JSX ............................................................................................................................................ 15
2.3.3 State and properties ..................................................................................................................... 15
2.3.4 Rendering .................................................................................................................................. 16
2.4 Related work .................................................................................................................................... 16

3 Method ............................................................................................................................................... 17
3.1 Theoretical study ................................................................................................................................. 17
3.2 Technical solution ............................................................................................................................... 17
3.3 Developing the isomorphic web application ...................................................................................... 17
3.4 Verification of thesis goals ............................................................................................................... 18

4 Implementation .................................................................................................................................. 19
4.1 Analysis of application requirements ............................................................................................... 19
## Table of Contents

4.2 The isomorphic design ................................................................. 20
4.2.1 Actions .................................................................................. 21
4.2.2 Dispatcher ............................................................................. 22
4.2.3 Stores ...................................................................................... 22
4.2.4 Views ...................................................................................... 23
4.2.5 Dehydration and rehydration .................................................. 23
4.3 Frontend .................................................................................... 25
4.3.1 Browser compability ............................................................... 25
4.3.2 HTML structure ...................................................................... 25
4.3.3 CSS styles ............................................................................... 27
4.3.4 Image assets ............................................................................ 27
4.3.5 Optimization strategies ........................................................... 27

5 Result ............................................................................................. 29
5.1 Resulting application ................................................................. 29
5.2 Rendering initial page load ....................................................... 31
5.3 Rendering sub pages ................................................................. 32
5.4 Client workload ........................................................................... 33
5.5 Comparison of implementations ............................................... 34

6 Conclusions .................................................................................. 36
6.1 Ethical discussion ....................................................................... 37
6.1.1 Technical aspects ................................................................. 37
6.2 Future work ................................................................................ 38

References ....................................................................................... 39
## Terminology

### Acronyms and abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>CSS3</td>
<td>CSS version 3</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unit</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTML5</td>
<td>HTML version 5</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>PNG</td>
<td>Portable Network Graphics</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SPA</td>
<td>Single Page Application</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1 Introduction

This thesis study was conducted on the Mid Sweden University for the company Hypervalley AB, located in Stockholm.

1.1 Background and problem motivation

The internet with its web pages have, since their first public appearance in August 1991, evolved a lot in terms of visual design, standards and programming techniques. Starting out as just a digital replacement for text publishing with static content, web pages today serves highly dynamic content and is one of the most common ways for humans to interact and share information with each other.

As the internet expands with faster up-link connections, smartphones becomes more powerful and technology constantly evolves, end-users sets higher demands on the products and services. Websites are no exception. To give a good user experience, a website does not only need to be visually appealing but also responsive, have a quick initial load time and process its user’s actions quickly.

Various programming frameworks have been developed lately to aid user experience and lowering developing costs, so called frontend frameworks. Frontend frameworks have the ability to process user actions more swiftly and lower the response times, but are not supported by all users nor search engines such as Google and Microsoft Bing. To support all parties of the internet a new strategy called isomorphism needs to be implemented.

An isomorphic web application classifies as a frontend application, but can also be executed at the server, enables both the client and the server to execute the application code. Advantages such as code reusage, sharing of common libraries and lowering the number of developers are some core advantages of isomorphic web applications. However, there are common pitfalls and new ways of thinking that needs to be taken into account, which can turn against the developers instead of aiding them. To successfully develop an full-scale isomorphic application, a good technical design and theoretical knowledge needs to be acquired.
This thesis has had its focus on isomorphic web applications and has analyzed the benefits and disadvantages of developing such an application. The outcome of the implementation have been compared to other implementations to see if there exists any performance gain in the isomorphic approach. If there exists notable performance gains in this thesis implementation, the problem of developing an full-scale isomorphic web application might get solved.

1.2 Overall aim

The overall aim of this thesis is to determine if the time spent developing an isomorphic web application is worth its performance gain. That is, if the performance increase is satisfying enough to abandon current reliable ways of developing a web application. To verify the aim in a practical manner a full-scale web application will be completely rewritten using the isomorphic approach. A successful implementation of the website should result in an increased initial load time, swift response to user interactions and a more stable and scalable codebase that is bound to the laws of isomorphism. Using an isomorphic technique forces the developer to follow designated code patterns. This will ease the future development by other programmers since the code is maintainable and consistent throughout the code.

Regardless of a successful or unsuccessful implementation of the web application, this thesis contains theoretical analyses and information about isomorphic web applications that will likely be of interest to other developers and researchers. Therefore, this master thesis study might solve the problem of developing a full scalable isomorphic web application.
1.3 Concrete and verifiable goals
From the problem statement defined above, a list of goals that needs to be fulfilled have been derived:

1. Study web applications and isomorphism to acquire theoretical knowledge about them.

2. Design a technical solution for an isomorphic web application.

3. Implement the technical design proposal for an isomorphic web application to build a full-scale application.

4. Compare the isomorphic solution with another framework in terms of rendering times and system memory usage.

1.4 Scope
The theoretical purpose of this thesis is to give a brief introduction to what web development is and explain the differences between a regular web application and an isomorphic web application. Thus, no deep dive into web development is covered, only the basics. However, a full survey covering modern web applications and how to deal with isomorphic web applications is covered in the theoretical chapter.

The practical focus of this thesis has been to implement the most vital part of the full-scale application, therefore only a subset of the web application have been implemented. The parts implemented are the most performance critical parts of the website, which was required to prove the isomorphic design solution liable.

1.5 Outline
Chapter 2 gives a brief introduction to web development and the technology behind it. Further down the section, the reader is introduced to single page applications and isomorphic web applications.

Chapter 3 presents the methodology used for this thesis study. The methodology covers both how the theoretical and the practical work have been done and what tools that have been used.

Chapter 4 is the implementation chapter which describes how two of the main thesis goals were solved and implemented.
Chapter 5 contains the results of this thesis. The results are an effect of the designed implementation in section 4 and aims to give the reader a overview of how the implementation performs.

Chapter 6 contains the conclusions about the work that has been done, an ethical discussion about the thesis work and proposed future work.

1.6 Contributions

This master thesis is exploring new grounds of web development. The techniques being used are merely used globally, besides at Facebook and Instagram. The contents of this thesis is therefore of interest to developers that are eager to learn modern and new ways of developing web applications. This thesis explains the difficulties and provides the solutions to isomorphism that can be difficult to come by, due to the face isomorphic web applications still are in its early stages.

There exists only a few scientific research papers on the subject which makes this thesis a good candidate amongst them. The results in this thesis could contribute to hold the technique behind isomorphic web applications liable and trustworthy, since it shows positive measurements compared to equal implementation in another common framework.

The specific implementation in this thesis is an improved version built upon a proposed technical design (Flux) from Facebook. This implementation however successfully solves the problem of creating a true isomorphic web application, in contrast to other various suggestions.

The technical design and implementation was done solely by the author, while all graphical works on the web application was provided by Hypervalley.
2 Theory

In chapters 2.1 - 2.4 this thesis theoretical study is presented in the form of a brief introduction to web development, followed by a description of single page applications, isomorphic web applications and related work.

2.1 Web development

The development process of a website varies a lot depending of its functionality and user interaction, but the basics are the same for all of them. A website requires an internal structure for its text and images, some graphical and dimensional interpretation and finally some interaction processing and handling. In development terms these parts are referred to as Hypertext Markup Language (HTML), Cascading Style Sheet (CSS) and JavaScript.

2.1.1 HTML

HTML is not a programming language like C++ or Java for example, but instead a markup language which means it describes the structure of an object (in this case a website) and not how the object operates or functions. The building blocks in HTML are referred to as elements and are of different types with different properties and attributes [1], see figure 1.

```
</div>
</div><!-- panel -->
</div>

<div class="col-md-6">
  <div class="panel panel-default">
    <div class="panel-heading">
      <h4 class="panel-title">Nyhetstopp</h4>
      <p>Justera information för nyhetstopp</p>
    </div><!-- panel-heading -->
    <div class="panel-body nopadding">
      <div class="form-horizontal form-bordered">
      
      </div>
    </div><!-- panel-body -->
  </div><!-- panel -->
</div>
```

Figure 1: Example of HTML code.

The HTML language is identical to extensive markup language (XML) in terms of parsing and structure, but differs in terms of how and where it is used.
The name of the elements in HTML indicates how the block should be interpreted in the web browser and has its origins from newspapers, which was one of the websites first intentions to replace. Element names such as headline, divider, list and image are commonly used and describes exactly what they are called.

To be able to describe an element further, they support the feature of having special attributes and properties. A common set of default attributes are id, class, width and height. These attributes can in turn be parsed by the web browser or other third party services to determine how the element should be shown. This is called meta data, that is, data that describes the actual raw data.

The elements are stacked into each other, creating a parent-child relation. This is more commonly known as a tree. The HTML tree is called the Document Object Mapping (DOM) tree [2]. A tree structure is good for performance reasons since not all elements needs to update if one element change; only the element that changes and all it successive children (nodes).

The first draft version of HTML became public in the early 1990's and has evolved a lot since [3]. Today's standard is version 5 (HTML5) and features a large variety of HTML elements that enables highly dynamic and interactive websites.
2.1.2 CSS

CSS is a language that describes the graphical appearance (color, dimension, position etcetera) of each element on a website, see figure 2. Its purpose is to style each element to give a uniform representation in all web browsers and a custom appearance of the target element [4]. These styles are stored in a separate document called style sheet.

```
.btnWhite:hover {
    background: #fff;
    box-shadow: 0 1px 1px 0 rgba(0,0,0,.1);
}
.btnDrop:active {
    background: #fff;
}
/* icons for buttons ***/
.btnWhite .icon use {
    fill: #797979;
}
.btnWhite:hover .icon use {
    fill: #353535;
}
```

Figure 2: Example of CSS styles

CSS operates by defining a list of styles that should target one or more elements on a website. The targeting is done by defining a selector that matches the target element(s). This targeting can be performed against a HTML element's class, id or type with optional properties as well. The selector is executed right-to-left, which includes several performance aspects into consideration when writing efficient CSS styles [5].

2.1.3 JavaScript

While CSS can target specific styles when hovering an element for example, neither CSS nor HTML can make a website interactive and perform logical operations. JavaScript is intended to act as a middleman between HTML and CSS, alternating both in runtime and performs routines in response to user interactions (such as clicking on a button for example).

JavaScript has direct access to the DOM-tree making it capable to insert, delete and update elements in the tree. This feature enables JavaScript to alter the web applications behavior in real-time.
JavaScript is the language most commonly used in the web browser, but
can also be executed as server-side language. Despite its name it does
not share many features with the programming language Java. Its first
release appeared in the Netscape 2.0 browser and the Internet Explorer
3.0 browser [6]. The language introduced the ECMA standard which
was stated in late 1996.

Since the web pages follow many standards and common techniques,
several large pieces of commonly used code are combined into pack-
ages. These packages are referred to as libraries and are actively being
developed and maintained by developers across the world.

2.1.4 Communication

In cases where user actions on a website needs to be reflected in a
database and/or to other users in real-time, the clients does not suffice.
In those cases the server, headmaster of websites, needs to be taken into
account and respond to these kind of actions.

The server runs special designed software that serves web pages with
their appropriate HTML markup, JavaScript files and CSS documents.
The server does also communicate with databases and other servers (or
clients) and is therefore the middleman for all web browsers. For
example, in the case of a user submitting a form, a message is sent to the
server about the submission, then the message is processed and a
response message (new HTML markup) is sent back to the web browser.
In this way, the website can dynamically change layout in response to
user actions that the server treats.

The messages sent are commonly referred to as requests and responses,
where the requests are the originating messages which the responses
replies to. There are standard formatting types available for these
messages such as HTML, XML and JavaScript object notation (JSON)
amongst others.
2.1.5 JSON

The JSON message standard is one of the most common message types when communicating on the web. JSON is not directly comparable to XML, but can be used in a similar manner. The standard is inspired by how JavaScript defines its objects and arrays of data [7], see figure 3.

```json
{
  "userId": 5,
  "name": "Eric Mathiasson",
  "location": "Sundsvall"
}
```

Figure 3: A JSON message describing a person

JSON does not have any pre-defined names or conventions. The developer is free to name all the properties of the message object to whatever suits the application’s needs. Objects can also be stacked inside one an another, just like XML, creating tree-like structures that easily can be parsed by dynamic and high-level programming languages.

2.1.6 Dynamic languages

Dynamic websites needs yet another language at the server side to operate, a so called backend programming language. Though it is possible to use JavaScript for this task, other common languages includes PHP, python, C#, golang and Java amongst others. All of these languages supports individual frameworks used to aid the development of a dynamic website. A common framework pattern is a so called Model View Controller (MVC) framework.

2.1.7 MVC frameworks

A MVC framework is built upon three concepts, a model, a view and a controller [8]. Together they make up a complete framework for creating robust and easy-to-maintain websites.

The controller acts as an input and output mechanism that keeps track of how data flows in the application. The controller handles the request, determines what action to do and send an appropriate response back to the origin [8]. This part of the framework is also responsible for controlling which HTML that is to be handled at different part in the application.
The view is the part which handles the HTML markup, in the form of so called templates. A HTML template is a set of HTML code with placeholders for dynamic data. The controller picks up the appropriate view and replaces the placeholder blocks with dynamic data from the models [8]. Hence the name template since the view can dynamically switch out some parts and replace them with other content.

A model is responsible for holding application data, often representing a real world object such as an user for example. The models are often directly bound to the database where each table in a relational database correspond to a model with the same name [8]. In that manner an instance of a model represents a singular item, called record.

Due to the nature of models and how they are bound in MVC frameworks, an update in a model often leads to cascading updates. This is due to the fact that the frameworks often use a two-way data binding, that is, a property is bound to another property somewhere else in the application. As one of the updates, so does the other. This quickly goes out of hand and can create unpredictable cascading effects, where several different places of an application updates in response to user interactions.

2.1.8 Responsive web design

Since the smartphone revolution occurred, websites have evolved in terms of design and programming approaches. A common term is responsive design which means that a website adapts itself to the device currently being viewed on. Whether it is a smartphone, tablet or laptop the website should adopt itself to fit the target screen and resolution, giving a good user experience across all devices. Since April 2015, Google has started to take a websites mobile friendly implementation into account when ranking them in the search results [9]. This introduces a common phrase mobile first design, which means websites should be designed from a mobile point of view instead of desktop [10].

Not only the graphical design of an application needs to utilize the mobile first approach, but also the technical HTML part. Programmers needs to write efficient HTML markup and CSS styles in order to write reusable components that easily can be adapted to other appearances. The source code gets even more complex as well since code that are
aimed for a desktop version of the site, must not be executed in a mobile environment and vice versa.

To target CSS styles at specific resolutions, a technique called CSS Media Queries is used [11]. These queries matches the current viewport resolution and executes the describing style sheet. Thus, several style sheets can be loaded into one web page, and some only applies to certain resolution and/or devices.

JavaScript can also be targeted using a similar technique as in CSS Media Queries. JavaScript has access to the current viewport dimensions and can thus write code that only executes when the viewport matches specific dimensions.

HTML elements does not have any standard way of modifying themselves in the DOM-tree, instead, JavaScript is used to accomplish this. To simplify the process of dynamically dealing with the DOM-tree, and bind data together, there are JavaScript based frameworks available to solve this problem. Such frameworks are getting more popular and are used to make so called Single Page Applications (SPA).

### 2.2 Single Page Applications

A single page application differs from web applications previously described in chapter 2.1 [12]. This type of applications does not rely on the server to determine what a state change (that is, navigating to a sub-page) should result in. In other words, the web browser already know about the views and how the data is bound throughout the application. This is possible due to the fact that SPAs are written in JavaScript and are retrieved as a library at the initial page load. Once downloaded in the client, it unpacks and generates all HTML markup which is dynamically inserted into the DOM-tree. This requires that the web browser has JavaScript enabled, otherwise a blank web page would be displayed.

Since a SPA is client based and keeps all HTML structure stored, it supports websites to be accessed offline, with some limitations. The application would function to the point it needs to interact with the server, at that point it would need internet access. While being offline, all the user actions could be recorded and sent to the server.
2.2.1 Frameworks

There are several frameworks available to ease the development of a SPA, some of them more common than others. Three examples of such frameworks are Backbone, Angular and Ember. All of the frameworks provide similar functionality and can be used to create single page applications.

Backbone was created before the other two frameworks [13]. Backbone is a MVC framework that is highly modular and can use almost any suitable library for its model, controller or view components. The aim of Backbone is to ease the complicated work with models, collections and views.

Angular is developed and maintained by Google and had its first stable release in 2009 [14]. The framework features a two-way data binding which connects two or more elements on a website together, making them depend on each other and can efficiently share data. In terms of HTML structuring, Angular uses a declarative approach which allows the developer to extend the HTML language and create custom and reusable components. This aids larger applications since code re-usage minimizes the size of the application and downloads quicker in the web browser, which minimizes the initial load time for the end user.

Ember is a similar framework to Angular, except it does not use the declarative approach for HTML templates, nor does it use the same data-binding technique as Angular or Backbone does [15]. Ember is also a MVC framework with Handlebars as view layer. Whenever a HTML element changes in Ember it picks up that change and re-render that specific element only. Angular on the other hand needs to check the complete DOM-tree on the page to determine which element has changed. This approach can hurt performance on a page with many elements, and is only suitable for small scale applications. Ember on the other hand does not suffer this limitation and is more scalable.

2.2.2 Data services

Since a SPA never refreshes the page the server cannot bind data in the HTML code, thus the web browser is not dependent on the server for updating the layout. With this approach, the web browser cannot access any databases either since credentials to the server would be exposed to the user and exploited as a security flaw. This is solved by making the...
server act as a middleman for these kinds of request, operating a so called web application programming interface (API).

There are many different types of web APIs. A commonly used API is the representational state transfer services API (REST-API) [16]. A REST-API often utilizes JSON or XML messages, making it easy for languages to parse and construct messages. JSON is often used in favor to XML since JSON has lower overhead data in the payload. The API works by listening for request on a pre-defined uniform resource identifier (URI) with a corresponding Hypertext Transfer Protocol (HTTP) request type. Once the server receives a HTTP request, it parses the incoming data and executes defined functions that corresponds to the request URI being made.

In single page applications, these kinds of APIs are highly utilized. The applications only needs to setup rules for the basic data types they require, such as users, forum posts, news etcetera. The REST-API only serves that portion of data and the rest can be calculated in the web browser when needed. This keeps the server code consistent since it only focuses on raw data types, instead of HTML templates and other business logic.

2.3 Isomorphic web applications using React

An isomorphic web application is classified as a regular SPA, but differs in terms of how data and business logic is handled. In an isomorphic web application the code in the web browser and on the server is the same, that is they share the same application code. Sharing the code base gives a couple of advantages:

- Developers only needs to maintain a single code base
- Clients can visit the website without JavaScript enabled
- Make use of same libraries on both server and client

The main criteria for an isomorphic web application is that it needs to be written in the JavaScript language; since it needs to be supported in the web browser. There are several server software available for parsing and executing JavaScript, one of them is NodeJS[17], which efficiently runs the application code and supports multi-core technology.
Sharing the application code between client and server does not affect performance, it rather optimizes it. At the building phase of the project, the source code is compiled and minified into packages; one package for the client and one package for the server. Each package is optimized for the specific environment, which results in no redundant code at the server and/or the client.

Another criteria for making an isomorphic web application is that the server needs to be able to generate a string representation of the DOM-tree. This is due to the fact that search engines (such as Google) and clients with JavaScript disabled needs an HTML representation when they visit the site, since they cannot execute the code and generate markup by themselves. This also results in greater user experience since the HTML markup is available in the initial page load, which is directly rendered in the web browser.

There exists various libraries for creating isomorphic web applications, some of them more used than others. Facebook released such a library they currently are using in the development process of Facebook itself and Instagram [18]. The library is called React and introduces a new approach to writing web applications.

### 2.3.1 Virtual DOM

When React was released to public, one of its core features was a virtual DOM-tree. A virtual DOM is a DOM-tree that relies solely in the memory and can be converted into a string representation when needed. While this seems pretty useful it plays the most significant part of supporting development of isomorphic web applications, namely to be supported on the server side as well as the client site.

The server environment does not have access to any DOM-tree as the browser does, therefore it requires a DOM-tree structure in order to serve the initial HTML markup to the client. The virtual DOM-tree is used for this purpose, giving the server possibility to act as an HTML parser and generator. Other than just generating HTML markup, the virtual DOM tree serves another key feature for isomorphism - HTML difference calculation and patching.

The biggest performance loss in web applications is when a modification to the DOM-tree is being made. Transitioning between pages in a single page application requires that almost the whole DOM-tree needs to be
cleared and new elements to represent the new page will be inserted. The virtual DOM solves this issue efficiently by comparing the current DOM structure with the one to transition to \[19\][20]. It looks for differences and calculates the least amount of deletions, insertions and updates that needs to be performed in order for the DOM-tree to look like the new state. In other words, the whole DOM-tree is not cleared and re-rendered, instead only portions of it is. This is a big deal for mobile devices where CPU power is less than desktop browsers, and thus, needs more carefully optimized algorithms.

2.3.2 JSX

JSX is a declarative way of creating web components with React. It is not required to use JSX when developing applications with React, however it is similar to XML and HTML and provides extensions for running JavaScript expressions inside HTML tags \[21\]. It is used in preprocessors to generate HTML code and is thus suitable for the common HTML developer.

While JSX is similar to regular HTML it has some core differences. JSX cannot include escaped HTML characters by default, since an built-in security mechanism is applied that escapes unknown characters \[21\]. Neither does it support custom attributes on elements. The use of custom attributes must be prefixed with the data attribute.

2.3.3 State and properties

React is built as a component-based framework. Each component is meant to act as a simple state machine, meaning the user interface (UI) is built with several components that are in a specific state. By following this pattern the application UI stays consistent and is easy to follow.

The component state should only contain the least amount of data to represent the component. Computed values and other React components does not belong in the component state, but data that changes in response to user interactions do. For example when an user types text into a text field, the input value should go into state for processing. Or when a button is clicked and toggles it state.

The idea in React is to have as many stateless components as possible that acts as a rendering canvas, and have one or many stateful
components higher up in the HTML hierarchy that are stateful. The state is then passed further down by using properties.

A property in React is the same as attributes in HTML, see section 2.1.1. Properties acts as meta data for the component and should not be confused with component state [22]. The common pattern is to pass data from a stateful component down to a stateless component by properties. This keeps application state central and can easily be modified without cascading effects throughout the application.

2.3.4 Rendering
The React components are rendered using a function named render. It is the render function that structures the JSX code, computes properties based on the state and returns the complete component structure. The render method is automatically invoked whenever a component state changes. The render method should only read application state and properties.

2.4 Related work
Isomorphic web applications are, at the time of writing, a new approach to developing web applications and are not widely used. The React framework however is used at both Facebook and Instagram, as well as the Flux architecture. These two application are the best related work since they are as well full-scale web applications built as an isomorphic web application. The exact implementation details at Facebook and Instagram may still differ, but the main technologies used are React together with Flux.
3 **Method**

This master thesis study was mainly done at Mid Sweden University and partly at the company Hypervalley AB in Stockholm. Various project methods have been combined, but mainly the agile approach have been used with recurring meetings over Skype and real world visits in Stockholm.

3.1 **Theoretical study**

The first goal was to study web applications and isomorphism to acquire theoretical knowledge about them. Previous knowledge about web development and single page applications have been applied, together with an internet-based research on articles describing web development and isomorphic web applications.

The theoretical study have been focusing on common frameworks for building single page applications. The frameworks covered in the theoretical study are Ember, Angular, Backbone and React, which all of them are written in JavaScript and are used to create single page applications. Based on the theoretical research performed, these was the most common frameworks used to create a SPA, hence the choice of them.

3.2 **Technical solution**

The second goal was to design a technical solution for an isomorphic web application. The proposed design solution have been inspired by various other isomorphic implementations found by other developers. The most crucial part in the technical solution was the usage of the React framework, developed and maintained by Facebook. React with its features was used as the main framework for designing a robust and reliable data structure for an isomorphic web application.

3.3 **Developing the isomorphic web application**

The third goal for this master thesis was to implement the technical design proposal for an isomorphic web application to build a full-scale application. As previously stated the React framework version 0.13.1 was used as the main framework for the application, resulting in that
JavaScript was used as language on both client and server side. The server software used was Node.js version 0.12.1 together with the Express framework. All source code was written using the Sublime Text 2 IDE together with various JSON parsing tools online. One of the requirements was to have good documentation and source control which was solved using YUIDoc [23] as documentation and GitHub as source control.

To automate the building process when an edit have been made, the building software Jenkins [24] was used. The Jenkins software was connected with the GitHub project which aided the development since Jenkins was re-building the application for every commit being made to GitHub. Two environments was used in Jenkins, one for staging environment and one for the production environment of the application.

3.4 Verification of thesis goals

The fourth and final goal was to compare the isomorphic solution with another framework in terms of rendering times and system memory usage. To solve this goal the isomorphic solution was measured in rendering time of the initial page load, when navigating to sub pages and how much memory was used. The rendering time is the time it takes from when the user enters the website address and starts loading the website, until everything is rendered and the user can interact with the website. The same actions and environment was measured using the Ember framework, which was compared to React framework.
4 Implementation

To determine if there exist such a performance gain in isomorphic web applications that other single page applications can be abandoned, a real-world application needs to be implemented and evaluated. In this chapter the system design considerations, isomorphic implementation and database design is described.

4.1 Analysis of application requirements

As previously described in this master thesis in chapter 1.2, one of the primary goals was to rewrite a full-scale application using available isomorphic technology. The responsible company stated a list of requirements the application needs to fulfill. The application should:

- Be a responsive web application that supports modern devices
- Be swift in user interactions
- Be implemented using an isomorphic approach
- Be optimized in loading and rendering times

In terms of isomorphism, the use of a virtual DOM is required for the algorithm to function server-side. Because of this, there is only need for a framework to support virtual DOM algorithms to create an isomorphic web application. However most of libraries like virtual-dom[25] and virtualdom[26] does not provide any functionality to create custom web components, neither do they have a big developer community. Additional libraries can be combined to solve the lack of custom components, like Polymer [27], but the struggle getting them to cooperate is not worth the time being spent.

In order to create a scalable isomorphic web application, knowledge about the common pitfalls, strengths and weaknesses in isomorphism is required, as described in chapter 2.3. A suitable framework that fulfills the company’s application requirements is Facebook’s React. The React framework was chosen as the framework for development of the new version of the website, due to its strong virtual DOM functionality and the ability to write custom re-usable web components.
4.2 The isomorphic design

The React framework has the ability to create custom and re-usable components, but lacks default support for data handling (such as models in a MVC framework). React's focus is in the view layer of the application which opens up the possibility for developers to create their own model layer and customize the application for its needs.

The first big feature the isomorphic pattern needs to support, is the ability to render the website at server-side, send it to the web browser and let the browser take on from there. This requires the web server to be able to execute the same application code, render the HTML markup and send the markup to the client together with the application state. The concept of dehydration and rehydration is introduced, which basically serializes the application state on server-side and later de-serializes the same code in the web browser, making the web browser able to catch up where the server ended its processing.

To solve common issues in MVC frameworks, such as the two-way data bindings described in section 2.1.7, a new algorithm for data flow that does not make use of a two-way data binding has to be invented. The new algorithm should be split into decoupled component to aid abstraction and transparency for the developer. Such an algorithm is necessary for isomorphic web applications since React by nature is a component driven framework that favors modularization.

A proposal for such an algorithm, or architecture rather, have been provided by Facebook called Flux. The architecture utilizes a so called unidirectional data flow which solves the issue of two way data bindings, see figure 4. This thesis has implemented a modified version of Flux to fit the isomorphic pattern with the dehydration and rehydration process. Chapters 4.2 - 4.3 describes how the implementation of this thesis has been committed.
In the Flux architecture there are only four main decoupled components; the actions, dispatcher, stores and views. The latter one is the component that the React framework provides, the other parts were implemented and designed by the author solely. However, the Flux architecture only solves the issue of data flow, not the dehydration nor the rehydration stage are provided, which is key to an isomorphic web application.

Using an unidirectional data flow keeps the red line throughout the web application [28]. It reminds of flow diagrams where information is traveled along one line and does not take any short cuts. The application state is held by the stores solely which allows the other parts in the application to be decoupled and abstracted from each other.

While Flux has advantages it comes with disadvantages as well. It has a slower learning curve for the novice web developer(s) and does not feature the same routing and model architecture as found in MVC frameworks. However, the last two can be solved using external third-party libraries.

4.2.1 Actions

The action component is what triggers a change in the application [28]. Actions may be invoked by the views, in response when an user clicks on a button for example, or by other occasions when data is asynchronously loaded for example. The action can hold optional payload data that is used for describing what the action actually does. The action is created (dispatched) by the dispatcher which in turn trigger an event that stores are listening to. Thus the data flows from an action, through the dispatcher and further to the stores.
4.2.2 Dispatcher

For comparison to MVC frameworks, the dispatcher works as one central controller that all data flows through [28]. The dispatcher is responsible for dispatching actions and calling the concerned stores with the action’s payload data. At initialization of the application the stores are connected to the dispatcher, exposing their events and which actions they respond to. Once an action is dispatched in the dispatcher only the affecting stores would be notified, and thus, isolating the action's payload and providing transparency.

It is also the dispatcher’s job to manage dependencies between stores. This is solved by setting up constraints that describes in which order stores should be notified by actions. This provides transparency for stores, since they only need to worry about getting data, not how or when they get the data. In other words, the data in stores will always be available when needed, thanks to the dispatcher.

4.2.3 Stores

The stores are responsible for holding application state and business logic. It is not to be confused with the traditional models in a MVC framework since stores manages state of all objects, not a single record as in traditional models. It’s therefore not directly comparable with Ember’s model or Backbone’s collection object. The idea is to let stores manage state of a complete part on a website, rather than just collections of records.

Much like traditional models, stores does internally keep a collection of records but does not acquire them by their own [28]. As mentioned previously in section 4.2.2, stores registers them self in the dispatcher and listens for specific actions. Each action is bound to a callback handler which gets called with the provided payload, which in turn executes business logic and updates itself. Once an update is being made the store emits an event to the views which queries the stores for their state. The store is therefore dependent of dispatcher actions which holds the application data. This is a crucial part of the unidirectional data flow since stores cannot access data by themselves, but need to acquire them from the dispatcher.
4.2.4 Views

The view layer is what the React framework provides out of the box. The component is used to create custom and reusable components to aid full scale applications where consistency and code re-usage is a big concern [28]. The components can be nested inside each other just like HTML markup, in fact, the React components are custom made HTML elements that are used in exactly the same way native HTML elements are used. However, since the React framework does not support the Flux architecture, thus a technique needs to be designed a technique for views to connect to stores and getting the stores’ data. To solve the problem of connecting views to stores, a new layer called controller-view is designed and implemented.

The controller-views are created using a mixin named `connectToStores` which exposes the stores’ public methods for retrieving their state. When the mixin is applied, the view functions acts as a controller-view and can access the state of the connected stores. When a top level controller-view is connected to a specific store, all the descendants of the components are also available to fetch the stores’ state. This minimizes misuse of the mixin, plus providing transparency for child components since they automatically gets the data without knowing how the data is fetched.

Typically a controller-view is used for top level components in an application [28]. The developer needs to carefully plan the application structure to identify these kind of decoupled application parts. This is due to the Flux architecture where a website should be modularized into decoupled independent parts, where they only share information through stores and dispatched actions. Stacking multiple controller-views inside one another breaks the data flow since data then can flow from several directions into the DOM hierarchy, creating inefficient debugging possibilities and multiple render calls.

4.2.5 Dehydration and rehydration

The execution of an isomorphic web application is performed in two steps, first at the server and then at the client. The exact same application code is executed at both ends, providing transparency for both non-JavaScript enabled clients and those who have JavaScript enabled. For this to work in practice the server needs to calculate the application state from which end point the client is viewing the site,
render the HTML markup with React and send that to the client together with the state. The client picks up the HTML, renders the site and boots the React framework into the DOM-tree, applying the provided state and re-renders the application once more. The last render step would not affect the layout since the server rendered markup and the client rendered markup is the same, assuming application state is correct. However, if the markup should differ React will stop execution with an error and notify the user that an error has occurred.

In this isomorphic design solution the application state is kept inside the stores, see section 4.2.3. Using this approach, dehydrating the application state from the stores and rehydrate it on the client is done using a single JSON object, which is inlined as a script element together with the other HTML markup sent from the server. While this gives more markup for the web browser to download in the initial request, it is a safe method for JavaScript-disabled browsers since they will ignore any script element in the HTML source.
4.3 Frontend
The isomorphic web application was designed as a responsive website from the ground up. It adopts the common mobile first technique and is intended to run across all modern browsers and devices, see figure 5. The key focus in the application is quick response and rendering times, which requires the use of modern web technologies such as HTML version 5 (HTML5) and CSS version 3 (CSS3).

![Mockup of the website in desktop mode](image)

**Figure 5: Mockup of the website in desktop mode**

4.3.1 Browser compability
The application relies on modern web browser techniques and thus does not support elder browsers such as Internet Explorer 8. The support for this implementation in desktop browsers are Internet Explorer 10 or higher, Firefox 4 or higher and Google Chrome 12 or higher. Mobile browsers with Apple iOS 6 or higher and Google Android 4 or higher is also supported. Elder browser may still be supported, but are not intended to be and are therefore untested.

4.3.2 HTML structure
The website make use of several new elements introduced in the HTML5 standard. Elements of the types section, article, aside, header and footer is a few of them to mention. The reason for this is to aid HTML-parsing and to give better semantics in terms of what the element actually represents.
A common used element is the canvas-element, for drawing high performance graphics and animations. The popular parallax effect (which can be seen at various places in the application) is rendered using the canvas element. This element has the advantage of being graphically processing unit (GPU) accelerated using WebGL technique, which minimizes the central processing unit (CPU) overhead and boosts overall performance.

Implementing a responsive website puts requirements on how the HTML markup is structured. The ideology is to reuse the same elements on both desktop and mobile, but change their appearance with CSS styles. This application utilizes this common strategy but does also insert and remove some elements between the mobile and the desktop version of the site. An example is the top header in the mobile where the layout changes depending on which subpage it renders, see figure 6. This is done by the React framework and self-written helper classes where it renders separate HTML markup for desktop and mobile viewports.

Figure 6: The mobile header changes HTML structure when navigating
4.3.3 CSS styles
The latest CSS standard, CSS3, introduces the ability to animate properties in response to user interactions, such as hovering an image. The technique is called transitioning between properties. Previously this was done using JavaScript but can now be done natively using just CSS styles. Also the ability to manually be able to GPU-accelerate certain elements was introduced, which enhances performance throughout the application.

The implementation in this master thesis uses CSS3 transitions and GPU-accelerated elements at various places. Especially on the mobile version of the site. Mobiles today is not as powerful as a desktop computer, which needs to be taken into account when developing a responsive website. All animations on the mobile implementation of the website are done using CSS3 transitions with GPU-acceleration enabled on them, to enhance the user experience.

Just like adopting the HTML structure for different devices, the CSS styles needs to change in order to make up the mobile layout. This is done by using CSS Media Queries, which targets specific CSS styles to a given resolution.

4.3.4 Image assets
There are several ways of adopting images to fit all devices. The biggest issue on mobile devices is their dots per inch (DPI), which varies across all devices. Either the application serves different rasterized images for the appropriate resolution, or use a resolution independent image format such as scalable vector graphics (SVG). This application use SVG images for all icon-based graphics, but portable network graphics (PNG) or joint photographic experts group (JPEG) based images for all other pictures.

4.3.5 Optimization strategies
One of the requirements for the implementation was to serve an optimized site that is lightweight in terms of physical size and does not take long time to render.

The physical size in bytes of the application can be optimized in several steps. The main part is to minimize the size of the JavaScript code, which was done using minification tools such as UglifyJS[29]. Using such a tool
analyzes the code and rewrites it in a more lightweight manner, which can save up to 75 percent of document size. The same technique is used for optimizing CSS documents.

All image assets (SVG files and rasterized images) are also optimized for optimal performance. Various tools such as ImageMin[30] in the building process takes all rasterized images and shrinks them as much as possible without losing visual quality. All the SVG images are inlined into the HTML structure to create a sprite sheet. In this manner, all SVG icons are loaded together with the initial HTML request, and thus, does not require fetching all SVG files separately which saves bandwidth.
5 Result

The chapters 5.1 - 5.4 will present the resulting application developed in React, followed by performance measurements and comparisons.

5.1 Resulting application

The resulting application is a fully responsive website built upon modern web technologies. The main purpose of this thesis' web application is to aid online advertising and marketing for publishers and news groups. The first page consists of an overview of websites, social profiles and publishing groups available in the application, see figure 7.

![Figure 7: Overview of the website in desktop mode](image)

The overview can be filtered using a real-time filter that does partly filtering on client side, but also sends several asynchronous JavaScript and XML (AJAX) requests to server side to fetch new entities to filter out.

The response time is quick due to the fact React uses partially loaded data. That means all data is not loaded on the initial request, but only the most vital parts. The rest of the data are asynchronously loaded at a later point in time. This result in better visual feedback for the end-users since they can interact with the website directly on page load.
The mobile layout is equal to the desktop version, but minified to suit the device, see figure 8. The minimal resolution supported is 320x480 which corresponds to an iPhone 4. There is no maximum resolution limit since the CSS Media Queries only are applied to resolutions below 1200 pixels in width. Resolutions higher than that simply centers the web page on screen, providing space on the sides on the main content.

Figure 8: The website adapts itself to fit mobile devices

The mobile layout hides and shows elements in a different way than on desktop. For example the header changes to a search field with a touch button to the left, for showing the main navigation.
5.2 Rendering initial page load

The overview page is made up of several boxes representing either a website, social profile or publishing group, see figure 9. Each box has to be sequentially rendered by the web browser, where the time it takes depends on the HTML structure, CSS styles and JavaScript execution. The website initial page load was measured in the range between 10 to 200 boxes, where the time it takes is measured from where the browser begins parsing the site until it is completely rendered, see figure 9.

![Figure 9: Comparison of initial rendering time between React and Ember](image)

The frameworks compared was React and Ember. The result shows that React is slightly faster to serve the initial page load in the web browser. This is partly due to the fact React does not need to build the HTML structure from scratch since it is generated on server side first.

The time measured is an aggregated time of the scripting time, layout time, painting time and miscellaneous other web browser actions performed during the initial page load.
5.3 Rendering sub pages

The time it takes to render a new page when navigating on the site was also measured, see figure 10.

Comparing transitions between sub pages is where the performance gain a virtual DOM-tree kicks in. The result in figure 10 shows that React is faster than Ember to render a new sub page on the website. This is due to the fact React does not need to clear the whole DOM-tree and render a new one, see section 2.3.1.
5.4 **Client workload**

When a website is displayed the web browser consumes some amount of memory to render it and execute business logic. The more memory are used the slower the web browser will respond to user interactions. Memory usage is therefore a big concern when developing single page applications, since the whole application resides in the computer memory. The applications memory usage has been measured and compared, see figure 11.

![Figure 11: Memory usage in the application](image)

The measurements shows no significant differences between React and Ember, but React is more efficient in the long run. But websites containing a small amount of elements, or a fairly large amount of elements, the memory usage is similar in both frameworks. The peak noticed at 200 boxes is due to internal optimizations done by the web browser itself, above this line the frameworks converges.

The difference at 50 to 100 boxes is due to a more memory-optimized algorithm in React, compared to Ember. React does not need to keep as much overhead information for each element as Ember does, and thus, minimizes the memory consumption.
5.5 Comparison of implementations

The choice of which implementation or framework to use when developing a website does not only depend on how the framework performs. To find a suitable framework that covers the applications demands, the functionalities of the framework needs to be analyzed. Table 1 shows a comparison between the features that the React framework and the Ember framework offers.

Table 1: Comparison of the React framework and the Ember framework

<table>
<thead>
<tr>
<th>Feature</th>
<th>React</th>
<th>Ember</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Page Application</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Supports script-disabled browsers</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Uses a common templating language</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Fastest average rendering time</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>REST interface</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Unidirectional data-flow algorithm</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

At first glance React seems to solve all problems there are, but nothing comes for free. While it supports search engines (script-disabled browsers) and has the fastest rendering time, it requires the developer to think in other terms of how to build a web application. The unidirectional data-flow can be difficult to handle at because it disallows the components in a web application to directly affect another, as it is in a SPA written in Ember. The templating language in React is JSX which is common to HTML, but has limitations in the way data is bound in the HTML elements.
Isomorphic web applications -
Depends on how you react. Eric Mathiasson

Ember uses a more common approach with its MVC-architecture, but lacks support for all browsers and renders slower. The framework does not support an unidirectional data-flow either, which leads to the cascading update problem as described in section 2.1.7. However, Ember has a shorter learning curve than React and may be a better choice for novice web developers. The React framework is not suitable for a regular web application. React is aimed at a UI-intensive website that reuses components and serves highly dynamic content, not a personal blog site for example.

The choice of which framework to use depends on the application itself. Larger web applications requires more reliable and dynamic framework whilst small websites can suffice with an server-generated website. A single page application must not be abused, but handled with care. The developer(s) needs to be aware of the advantages and disadvantages of a SPA and what the outcome can be.
6 **Conclusions**

Developing an isomorphic web application requires a more complicated architecture in contrast to a regular SPA. There are several frameworks available for solving common issues with the isomorphic design, such as Facebook’s React for example. The most difficult problem to solve in an isomorphic implementation is how to dehydrate and rehydrate the application state in a transparent way between server and client. The second biggest problem to implement was the data flow architecture which must not use a two-way data binding, as other single page application frameworks use.

The two most difficult design goals were solved theoretically using the knowledge acquired about isomorphic web applications, as stated in the first thesis goal. By solving these problems, the second thesis goal of designing an isomorphic solution for web applications was completed using the theoretical study. The design stage went through several iterations of trial and error where different modifications to the common Flux architecture was tested. The first design proposal did use a similar design pattern to MVC frameworks, but was later abandoned due to a too complex data flow algorithm. It was decided that a modified version of the Flux architecture was an optimal solution for the implementation.

Implementing the design proposal and develop a full-scale web application, as stated in the third thesis goal, have been partially solved by the author. The design proposal was completed as the first step in order to later use that library to implement the full-scale application. The full web application was not completed, only the most crucial parts such as the overview page. This decision was made by the author due to time limits and due to the fact only the most performance critic parts needed to be implemented to prove the isomorphic solution liable and to verify that it performed as expected.

To verify that the isomorphic implementation performed as expected, that is at least as good as other single page application frameworks, the application had to be benchmarked, compared to and implemented in another framework. The benchmarks done in chapter 5 shows that the authors isomorphic implementation using React, performs better than the same website implemented in the Ember framework. The
performance increase is mainly because React use a virtual DOM-tree technology which minimizes the time spent modifying the website layout. The increase is also due to carefully implementing custom components that are solely designed for this application. The results confirms the fourth and final thesis goal to verify the application performance.

In conclusion using the React framework together with the isomorphic design described in this thesis shows that the proposed isomorphic implementation performs better compared to the Ember framework. It is worth to notice that the author had previous knowledge in the subject, and thus did not need to spend excess time learning the new way of developing SPAs. However, the choice of which technology to use is up the the developer; this master thesis shows that isomorphic web applications is a worthy opponent in the modern web application development area.

6.1 Ethical discussion
This master thesis' resulting application is used by people to market themself and their websites to find companies that wants to display advertisement on their site. Advertising on websites becomes more and more common these days and are frustrating to some people. This implementation of the new website might lead to an increase of online-based advertisement, and thus upset more people browsing the internet on their daily basis.

Another aspect is that some people's work may be obsolete due to this application, since it will in practis replace the salesman that needs to contact the companies and offer them a deal on adverisment. The advertising companies themselves can use this website and find appropriate advertising spots themself, skipping the need for a salesman.

6.1.1 Technical aspects
The website holds sensitive information about real world persons and companies. Information and credentials such as names, address', e-mail address', credit card numbers and passwords etc. To give a trustworthy impression the website has to offer secure channels and encrypt sensible data for the users. If this information would be leaked, due to a hacker
attack for example, it could lead to devastating consequences for the users since their credit card credentials could be abused.

The website does also make a great use of cookies and page tracking to enhance the user experience and serve directed content. This basically gives the website information about what type of habits the user follows on the application, which can be analyzed and stored for further use. The user can disable cookies if so, but will reduce the user experience.

6.2 Future work
First and foremost the complete web application should be implemented to really see the potential in isomorphic web applications and its performance. While not covered in this report, database optimizations and architecture was also implemented for the website. It would be of interest to make benchmarks and performance analysis of database connections and architecture as well. For a single page application to glance it needs a quick database connection since all data are fetched using asynchronous calls to the database.

In terms of comparison to other frameworks it would be of keen interest to compare several more frameworks to give a better overview of how React performs in contrast to the others. Due to time limit and implementation constraints it was chosen by the author to only compare with one framework (Ember). The frameworks Angular, Backbone and Knockout would be of interest to compare against as well, since they all are frameworks for creating a SPA. Angular is one of the top-leading frameworks that uses different technologies to handle DOM-tree, which would be of interest to see how it performs against Ember for example.
References


Isomorphic web applications -
 depends on how you react. Eric Mathiasson

Retrieved 2015-04-27


http://www.w3.org/TR/css3-mediaqueries/
Retrieved 2015-03-22


http://backbonejs.org/
Retrieved 2015-04-27

https://angularjs.org/
Retrieved 2015-04-27

http://emberjs.com/
Retrieved 2015-05-20


[17] Node.js, ”About” [www]
https://nodejs.org/about/
Retrieved 2015-05-20

[18] React, ”Why did we build React?” [www]
https://facebook.github.io/react/blog/2013/06/05/why-react.html
Retrieved 2015-05-20
Isomorphic web applications -
Depends on how you react. Eric Mathiasson

References
2015-06-16

[19] React, "React (Virtual) DOM Terminology" [www]
    https://facebook.github.io/react/docs/glossary.html
    Retrieved 2015-05-20

[20] React, "Reconciliation" [www]
    https://facebook.github.io/react/docs/reconciliation.html
    Retrieved 2015-05-20

[21] React, "JSX Gotchas" [www]
    https://facebook.github.io/react/docs/jsx-gotchas.html
    Retrieved 2015-05-20

[22] React, "Interactivity and Dynamic UIs" [www]
    Retrieved 2015-05-20

[23] YUIDoc, "JavaScript Documentation Tool" [www]
    http://yui.github.io/yuidoc/
    Retrieved 2015-05-20

[24] Jenkins CI, "Welcome to Jenkins CI!" [www]
    https://jenkins-ci.org/
    Retrieved 2015-05-20

    https://github.com/Matt-Esch/virtual-dom
    Retrieved 2015-05-25

[26] GitHub, "virtualdom" [www]
    https://github.com/Swatinem/virtualdom
    Retrieved 2015-05-25

[27] Polymer [www]
    https://www.polymer-project.org/0.5/
    Retrieved 2015-05-25

[28] Flux, "Application Architecture for Building User Interfaces"
    [www]
    https://facebook.github.io/flux/docs/overview.html
    Retrieved 2015-05-07
Isomorphic web applications -
Depends on how you react. Eric
Mathiasson

References
2015-06-16

[29] GitHub, "UglifyJS 2" [www]
https://github.com/mishoo/UglifyJS2
Retrieved 2015-04-27

https://github.com/gruntjs/grunt-contrib-imagemin
Retrieved 2015-04-27