Developing modern web applications with the React library and WordPress

Natalie Gustafsson

Degree Thesis
Online Media
2019

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Online Media
Identifikationsnummer:	6786
Författare:	Natalie Gustafsson
Arbetets namn:	Utveckling av moderna webbapplikationer med React biblioteket och WordPress
Handledare (Arcada):	Owen Kelly
Uppdragsgivare:	

Sammandrag:
Syftet med detta examensarbete är att utforska användningen av frånkopplad WordPress genom att använda de populära JavaScript biblioteken ReactJS och GatsbyJS. Med hjälp av dessa teknologier utvecklas en Gatsby kodmall som publiceras på GitHub för alla att använda. Syftet med kodmallen är att erbjuda ett lätt sätt att utveckla webbapplikationer som använder sig av WordPress och ReactJS. Målsättningen är att lösa följande problem: 1) WordPress sidor är väldigt detaljerade vilket kan leda till mycket onödigt innehåll vilket i sin tur leder till en långsam sida, 2) användningen av WordPress för sidor som använder en liten del av databasdrivet innehåll, men som annars är statiska, är inte optimalt. De huvudsakliga teknologierna diskuteras först varefter det förklaras hur dessa teknologier används i utvecklingen av Gatsby kodmallen. Detta examensarbete skall ge läsaren tillräckligt med information om de olika teknologierna för att kunna utföra ett liknande projekt på egen hand. Detta är inte en guide för de olika teknologierna, och endast de specifika verktygen som används presenteras och förklaras. Kodmallen som utvecklats som en del av detta examensarbete erbjuder ett seriöst alternativ till det traditionella sättet att utveckla WordPress sidor, samt löser de problem examensarbete har som mål att lösa.

Nyckelord:	Webbapplikation, Webbutveckling, ReactJS, WordPress
Sidantal:	36
Språk:	Engelska
Datum för godkännande:	29.05.2019

DEGREE THESIS	
Arcada	
Degree Programme:	Online Media
Identification number:	6786
Author:	Natalie Gustafsson
Title:	Developing modern web applications with the React library and WordPress
Supervisor (Arcada):	Owen Kelly
Commissioned by:	

Abstract:
The purpose of this thesis is to investigate the use of headless WordPress together with the React library. And to develop a Gatsby starter that will be published to GitHub for anyone to use. Trough the investigation this thesis aims to solve the following issues: 1) WordPress sites are very intricate which can lead to unnecessary bloat, ultimately making the site slow. 2) Using WordPress for sites that only use a small amount of database-driven content and are otherwise static, is excessive. First the main technologies used are discussed. Next, it is explained how the technologies are used in the development of the starter site. This thesis should give the reader enough familiarity with the technologies to develop a similar site themselves. It is not a full-fledged guide for the different technologies and only the specific tools used will be presented in the thesis. The thesis proved that the starter developed offers a serious alternative to the standard way of developing WordPress sites and gives solutions to the issues the thesis aims to solve.

Keywords:	Webapplication, Webdevelopment, ReactJS, WordPress
Number of pages:	36
Language:	English
Date of acceptance:	29.05.2019

CONTENTS

Figures

TERMS AND ABBREVIATIONS

Front-end	Part of an application that the user interacts with directly
Back-end	Part of an application that is not directly accessed by the user
CMS	Content Management System
Starter	Code that is repeated with little or no alteration i.e. code template
API	Application programming interface
REST API	Representational state transfer API
GitHub	Web-based hosting service for version control
Plugin	A software component that adds a feature to a computer program
JSON	JavaScript Object Notation
URL	Uniform resource locator i.e. web address
UI	User interface
DOM	Document Object Model i.e. application programming interface for HTML and XML documents
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
PWA	Progressive Web App
camelCase	Writing phrases without spaces where each word starts with a capital letter, e.g. eBay, iPhone
Headless	An application where the front-end is completely separated from the back-end and only communicates with it through an API
POC	Proof of concept

1 INTRODUCTION

The purpose of the thesis is to find an alternative solution to traditional WordPress sites by developing a GatsbyJS starter. Through the development of the starter I aim to find a solution for common WordPress issues like speed and bloat, as well as a solution to the problem of using WordPress for sites with only a small amount of database-driven content.

In more technical terms, the purpose of this thesis is to investigate the use of headless CMS, more particularly WordPress, together with the popular JavaScript framework React, and to develop a headless web application using these two technologies. As a part of this thesis, a Gatsby starter has been developed. It uses WordPress as the headless CMS and React based static site generator GatsbyJS for the frontend.

The original idea for this thesis came about when listening to a podcast on web development. In the episode, the hosts discuss headless CMS and in particular headless WordPress. They discuss how headless web development is a field containing many opportunities but that it is not quite user friendly enough. “The thing that is most needed in this space is unifying a lot of things and making it easy” (Syntax 2018), this episode of the podcast and specifically that quote is what sparked the interest to investigate the topic of headless development and to develop a starter in order to help other people get started more easily.

This thesis aims to give the reader enough knowledge of the different technologies to complete a similar project on their own, and knowledge about when and why this type of site might be used.

1.1 Background

When developing sites for clients, WordPress is often asked for. This is no surprise as the CMS currently powers 32% of the web, which means a lot of people will have used it at some point in their life (WordPress.org). The problem is that WordPress is not optimal for all types of projects.

Let us imagine a small business that wants to update their website. They need a page for pricing and a page for some basic information about employees and contact details. In addition, they want a blog on the site where they can post about current events. All of the other information on the site is more or less static and does not have to be updated frequently and could be built as a static site. The blog, however, will need a more complex site and a back-end CMS for it to work.

In this case, building a custom WordPress theme is too much work for what is, essentially, a static site. On the other hand, "ready-made" themes can be large and slow since they contain so much stuff of which only a fraction is used. This is where the starter comes into play. In the starter WordPress and the front-end of the site are not connected in any other way than the data being interchanged. Even here the site does not fetch all the data WordPress creates. It will only fetch the data it is told, which means there will never be unused data cluttering or slowing down the site. All of this is then deployed to a static site which makes the site even faster.

Making a WordPress site is fairly easy as one can install themes and plugins to customize the site. But without really understanding the intricacies of WordPress this type of site-building will quickly result in bloat and suddenly the site contains a lot of unwanted stuff that slows it down. That is the second big issue with WordPress sites, speed. In a world where a user expects a site to load in under 2 seconds (Patel 2011), one really cannot afford to have a slow site.

WordPress is also known for the security issues it has. This is of course something that will always be a problem as long as the internet exists. But security issues and attacks on sites can be minimized, by using a static site for example.

These irritating issues are one reason I chose to base this thesis on WordPress, it makes for a challenge and also something I know other people will have use of. It also offers developers without much experience in WordPress development a way to develop sites with the popular CMS.

Without dragging on WordPress too much, it also has good features. The actual CMS is good and user-friendly. This is the second reason for me wanting to use WordPress as the back-end software for this thesis.

Some people might also ask, why use WordPress at all when there are many headless CMSs, e.g. Contentful, available? Yes, there are many wonderful headless only CMSs out there which can be used together with Gatsby applications. I however wanted to see how the very popular and user-friendly CMS WordPress would work with this type of setup.

I am not the first person to think of this, there are other starters with the same concept. However, they usually contain lots of stuff that might be unrelated to what you are currently developing. Ultimately resulting in the same issue that WordPress has, unnecessary bloat. This is why I wanted to create a starter that is easy to pick up and tailor to any type of project. With the thesis and the starter, I wish to not only make my own life easier but also to help others find new ways to tackle the problems they face. And to offer a solution allowing developers to continue using the web's most popular CMS without hassle.

ReactJS was chosen for the development of the starter because of my personal interest in the framework as well as its popularity. I chose to use GatsbyJS for the static site generator tool because it was quite new when development of the proof of concept was started, it also offered a wide variety of plugins and built-in features that were appealing. GatsbyJS and GraphQL sort of come as a package. In the documentation GatsbyJS strongly advises developers to use GraphQL together with Gatsby, and this is the reason I chose to add GraphQL to the list of technologies used.

1.1.1 Proof of concept

Before starting to write or develop the final starter, a proof of concept was produced. The proof of concept was developed to look like the popular WordPress theme, 2017. By making it look and act like a WordPress theme it could easily be proven whether or not it could do everything a WordPress site could. The proof of concept worked successfully, and from there the development of the actual starter begun.

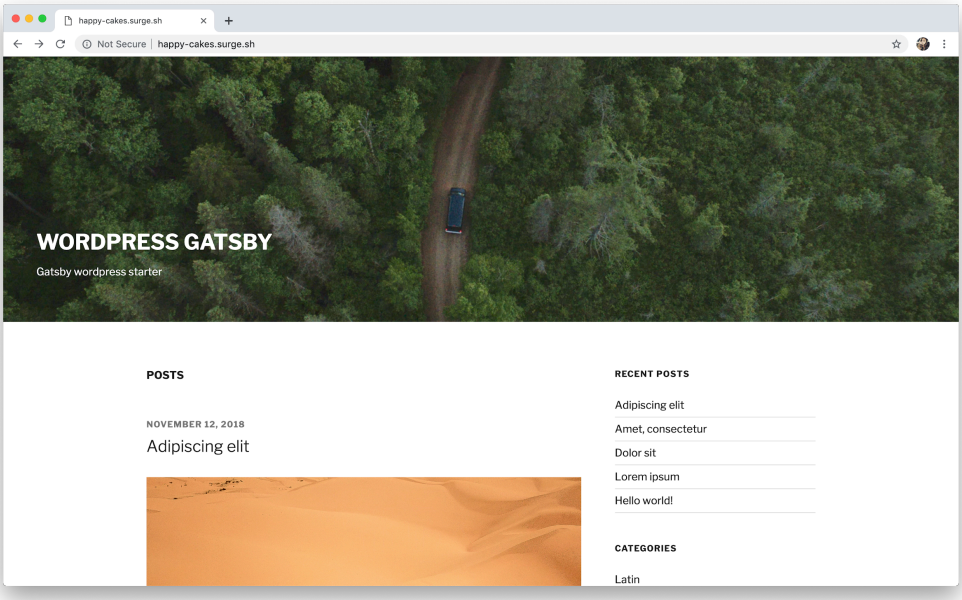

Figure 1. Screenshot of proof of concept demo site.

1.2 Goals and methods

The goal of this thesis is to produce and develop a starter site that demonstrates the use of headless WordPress together with React and to then publish the source code to GitHub for everyone to use. With this, I hope to help other people get started faster and more easily. As well as fill a gap in the current state of available information and tools regarding this way of developing sites. In the written part of this thesis, I will mainly focus on the technologies used and detailing the process of developing the starter, and also my findings on when to consider using a starter like this, and when to consider using something else.

The problems I seek to solve with this thesis are the following:

- WordPress sites are very intricate which can lead to unnecessary bloat, ultimately making the site slow.
- Using WordPress for sites that only use a small amount of database-driven content and are otherwise static, is excessive.

The primary focus of this thesis is on the development of the proof of concept as well as the development of the starter. Secondary research or desk research has been carried out

to gain a broader understanding of the technologies. The desk research method involves analyzing already existing data to gain insight into the field (Bhat 2018).

This thesis should give the reader familiarity with the different technologies used and how to use them for developing a similar site. This thesis is not a full-fledged guide on how to use the different technologies.

1.3 Delimitations

Both React and WordPress are huge frameworks that offer many different functionalities that can be used for a variety of projects. This thesis will only cover the functionalities used in the starter site. For example, this thesis will not cover WordPress theme development or plugins since these are functionalities that were not used when developing the starter site.

1.4 Structure

This thesis will be divided into three parts. First discussing the key technologies and explaining them in more detail. How they work together and for what and how they might be used. This part will also explore some minor concepts briefly for a better understanding of the whole project.

Once the technologies and other concepts are familiar, the second part will explain the development and setup of the starter site and how the previously presented technologies are being used within the site.

Lastly, I will discuss my findings when developing the site and researching the topic. And delve deeper into the pros and cons of developing sites this way.

2 KEY TECHNOLOGIES

This chapter will discuss the key technologies used in more depth.

2.1 ReactJS

React is a JavaScript library for building composable user interfaces (Hunt 2013). With it, you can build different one-page applications where the interface needs real-time updates, for example, a chat application. Development on react started in late 2011 and was released and open sourced in the summer of 2013, by Facebook Inc. (Krill 2014) It is now one of the most used JavaScript libraries and is currently being used to power Instagram, Facebook and Spotify, to name a few.

2.1.1 JSX

React components are written using JavaScript Extended, more commonly known as JSX. JSX is a way to combine both logic and markup in one file, instead of separating them. React does not require JSX to be used, but many people find it helpful when working with UI inside JavaScript since it uses HTML like syntax. JSX looks like a templating language but comes with all the features of JavaScript, this is what makes JSX very powerful. Figure 2 demonstrates a simple example of JSX. First a JavaScript variable called *name* is declared, after that, you can use it inside JSX by wrapping it in curly braces. (Facebook Inc. 2018b)

```
const name = 'Natalie';
const element = <h1>Hello, {name}</h1>;
```

Figure 2. Example of embedding expressions in JSX.

In fact, any valid JavaScript expression can be used inside JSX by simply wrapping it in curly braces wherever you need it. For example, *3+3*, *user.name* and JavaScript functions like *formatName(user)* are all valid expressions that could be used inside JSX. When compiled, JSX expressions become JavaScript function calls and are equal to ordinary JavaScript objects. This allows for JSX to be used e.g. inside if statements and for loops, assigned to variables and accepted as arguments. Below, figure 3 shows JSX being used inside a JavaScript if statement. (Facebook Inc. 2018b)

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {name}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Figure 3. Example of a JSX expression.

JSX tags may have multiple children e.g. a div can contain both an h1 and h2 element, however, all adjacent JSX elements need to be wrapped in one single parent element to work. (Facebook Inc. 2018b) Usually, all elements would be wrapped in a div. Figure 4 presents a JSX tag with two children, an *h1* and *img* element. Without the div wrapping both children the site would break.

```
const element = (
<div>
  <h1 className="greeting">Hello!</h1>
  <img src={user.avatarUrl} />;
</div>
);
```

Figure 4. Example of a JSX tag with children.

The biggest difference between writing JSX as opposed to regular HTML is the fact that JSX uses the camelCase naming convention to write out HTML attribute names. For example, the HTML attribute *class* will become *className*, *background-color* will be *backgroundColor* and so on. This is the result of JSX being more similar to JavaScript than HTML, and in JavaScript, the camelCase naming convention is used. (Facebook Inc. 2019b) The *className* attribute can be seen in use in figure 4.

2.1.2 Elements and components

React is based on components. A component is a piece of reusable code that can be used all over your application. It can define how something on your site looks and the behavior it has. (Saxena 2018) React components are made up of smaller pieces of code, which are called elements. In the chapter on JSX, figure 2 displays a react element.

React elements at their core are immutable. Meaning once created they cannot be changed. In the official React documentation an element is described as follows, "*An element is like a single frame in a movie: it represents the UI at a certain point in time.*". The UI can be updated through creating a new element and rendering that to the page, but the original element will always stay the same as elements are immutable. When an elements UI is updated React.DOM will only update the parts of it that are necessary. It will compare the new element and its children with the previous one and only update what has changed. For example, if the *user.avatarUrl* constant in figure 4 was to be changed, React.DOM would only update that part of the element. (Facebook Inc. 2019d)

With the help of these elements React components can be built. An example of a basic component would be a header navigation. A simple header navigation could be divided into four different components (see Fig. 5)

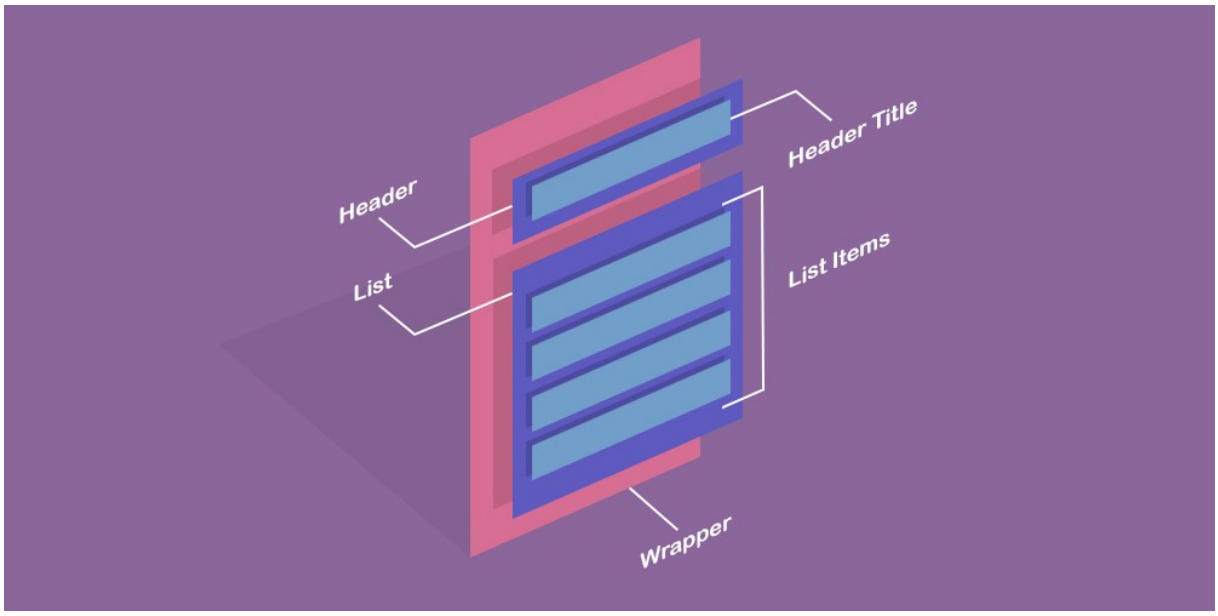

Figure 5. Component structure of header navigation. (Bilgili 2018).

The same navigation could also be written into one big component, two smaller ones or however many or few one wants. Buttons, forms, and dialogs are other pieces of code that are commonly written into components (Facebook Inc. 2018a). The possibilities are endless, which makes reacts component-based way of coding extremely flexible and suitable for all kinds of projects.

React components can be written as functions or as a class (Facebook Inc. 2018a). Figure 6 shows two different ways of writing a component, both of which are equivalent in Reacts point of view.

```
function Greeting(props) {
  return <h1>Hello, {props.name}</h1>;
}

------------------------------------------------------

class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Figure 6. Example of React component.

2.1.3 Props

To pass data between components React uses an object called props (short for properties). Components are in theory like JavaScript functions, they accept arbitrary props and return React elements describing what should appear on the screen. (Facebook Inc. 2018a)

Figure 7 is an example of props being passed to a component. This code will render *"Hello, Natalie"* to the page.

```
function Greeting(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Greeting name="Natalie" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Figure 7. Example of a component with props.

A component must never modify its own props, this is a strict rule in React. For example, a *sum* function is called a pure function since they never change their inputs and will always return the same result for those same inputs (see Fig. 8). (Facebook Inc. 2018a)

```
function sum(a, b) {
  return a + b;
```

Figure 8. Example of a pure function.

The same rule applies to components written as function as well as components written as classes. Nonetheless, most applications are dynamic and have UIs that need to be updated over time in response to user interaction, network responses and more, without the component modifying its own props. (Facebook Inc. 2018a)

2.2 Static sites

Static sites have been around since the dawn of the world wide web. The first website ever created was a static site, because back then there was no alternative. Even though the technology used for developing sites has progressed tremendously since, static sites still offer features that make them an attractive option even today. (Camden & Rinaldi 2017.)

Here are a few of the many benefits of a static site as presented by Raymond Camden and Brian Rinaldi in their book *"Working with static sites"* (Camden & Rinaldi 2017.)

Speed: Static sites are fast since the same HTML is being served to every user and is not slowed down by any dynamic rendering. Speed is extremely important since it has been proven that a user is likely to exit a site if it takes more than three seconds to load.

Security: A static site is more secure than a dynamic one. Usually, when a site is breached it is because of an outdated CMS, but since a static site is only static HTML and CSS it eliminates the risk of the CMS being a vulnerability. Of course, no site is a 100% secure but a static site definitely makes it more difficult to take advantage of.

Versatility: Since a static site is not dependent on a CMS the possibilities for how the site can look are endless. It can be completely customized to whatever the site needs to be.

Hosting: A static site also eliminates the hassle of hosting. Since it is only static files that need to be deployed there are endless possibilities for where and how the site can be hosted.

2.3 GatsbyJS

Gatsby is a react powered static PWA generator framework. It eliminates complicated and time-consuming site deploys trough building your site with "static" files. This is done by using technologies like React, GraphQL, CSS and more. Gatsby sites are known for being very fast since Gatsby fetches only the most crucial data, HTML, CSS and JavaScript for the page in question. Once the dependencies are loaded, Gatsby prefetches other resources in the background so that navigating the site is quick and painless. (Gatsby Inc. 2018)

Gatsby was made for building sites where the data can be brought from anywhere. It has a big data plugin system that allows users to easily fetch data from one or multiple sources like APIs, CMSs, databases or file system (see Fig. 9). Gatsby then pulls this data directly into your site with the help of GraphQL. (Gatsby Inc. 2018)

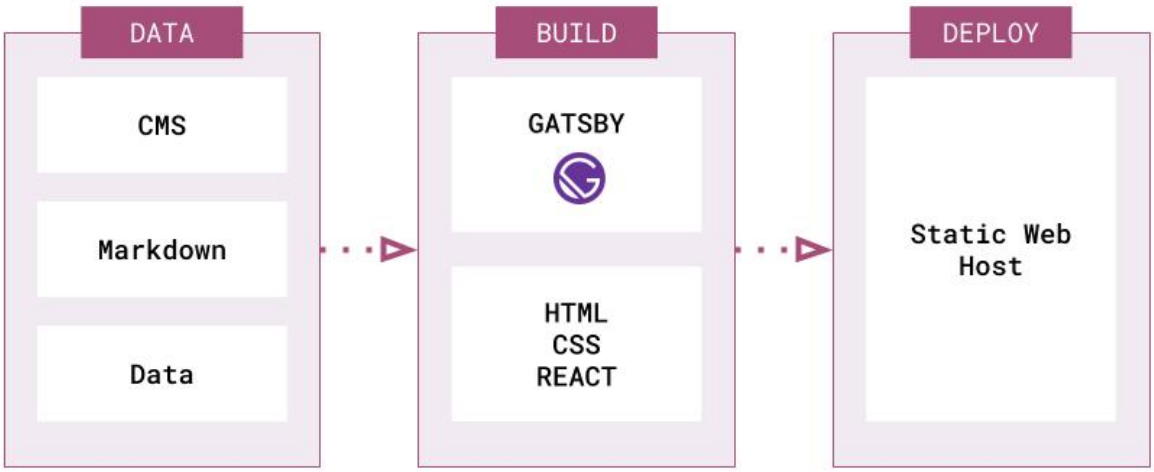

Figure 9. Map of how GatsbyJS works.

2.3.1 GraphQL

Gatsby uses a technology called GraphQL for accessing data. GraphQL is a query language for APIs that focuses on getting exactly the data that is needed, nothing more and nothing less. Where normal REST APIs need to load resources from multiple URLs GraphQL will get all the data your app needs with a single request, making data fetches incredibly fast. (Facebook Inc. 2019c) This makes GatsbyJS and GraphQL a perfect match since both technologies' priorities speed. As seen below in figure 10 GraphQL is really straightforward and always returns what you would expect.

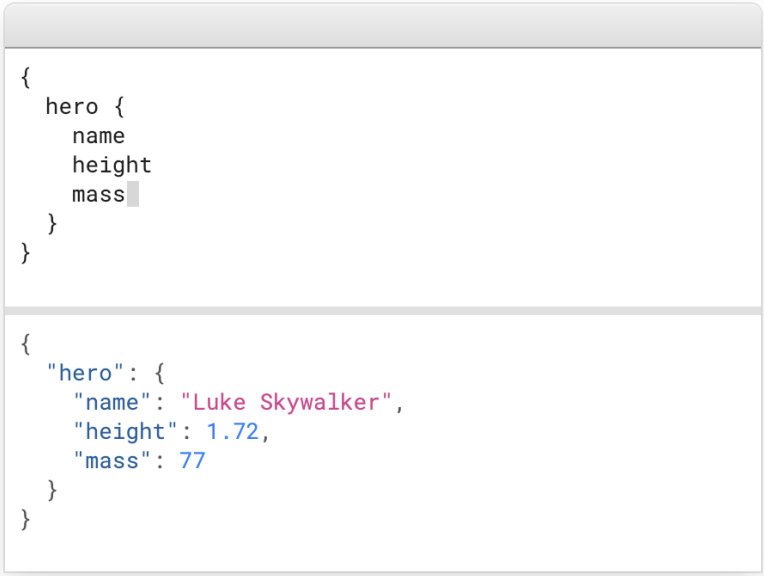

Figure 10. Example of a simple Graphql query. (Facebook Inc. 2018c)

2.3.2 Gatsby-source-WordPress

For this project, the *gatsby-source-WordPress* plugin was used. The plugin allows data to be pulled directly into the Gatsby application using the WordPress REST API. It currently will pull the data for the following entities:

- All basic WordPress entities (posts, pages, categories, tags, media, site metadata, …)
- Advanced custom fields
- Custom post types declared in WordPress' *functions.php*

(Gatsby Inc. 2019)

The plugin works so that every time the application build process runs it will automatically check the designated WordPress database for data and then pull it down.

2.4 Headless CMS

In an article for Storyblok Dominik Angerer explains headless CMS as a "back-end only content management system (CMS) that makes content accessible via a RESTful API for display on any device." (Angerer 2019). Where a traditional CMS is directly linked to the front-end, a headless CMS is not. This allows for the data to easily be distributed to many different channels (see Fig. 11). The term headless comes from the application front-end being chopped off from the back-end. Decoupled CMS is another term used for describing the same thing.

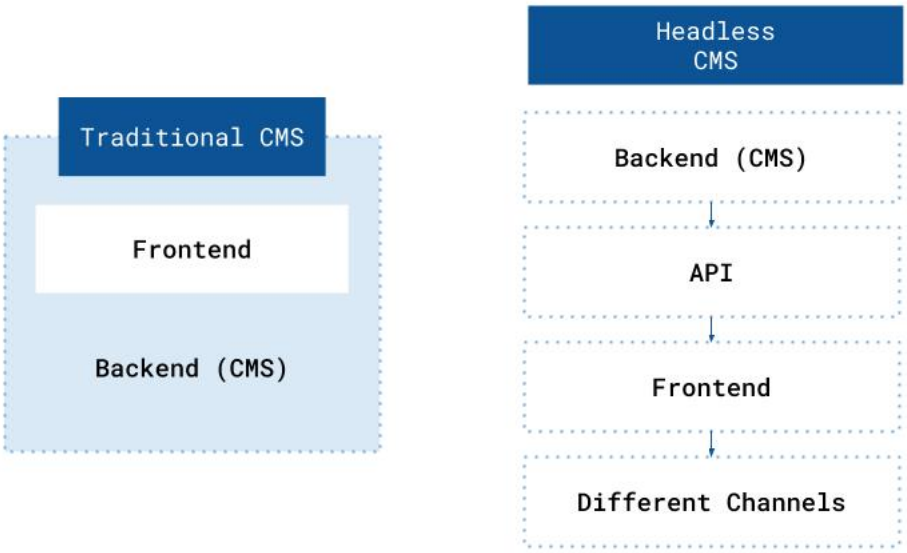

Figure 11. How a traditional and a headless CMS works.

For a typical static site, the content would be stored on the file system. A headless CMS gives the opportunity of storing those same files in the CMS and then distributing it through an API. Rather than producing content into e.g. Markdown files stored on your computer, the benefits of a CMS editor is gained. (Camden & Rinaldi 2017.)

2.5 WordPress

WordPress was initially released in 2003 as a blogging platform. During the years it grew to be one of the most popular CMSs, and today WordPress is thought of as a CMS rather than a blogging tool. (Ratnayake 2017 p.9)

WordPress is based on PHP and MySQL. It powers 32% of the web with everything from personal blogs, portfolios to corporate sites. WordPress is not only a CMS but also helps with building applications, providing translation, URL routing, HTTP requests and more. A key feature of WordPress is its simplicity, it was built to make getting sites up and running easy and fast. (WordPress Org. 2018)

2.5.1 WordPress REST API

WordPress uses its own REST API allowing developers to interact with sites by sending and receiving JSON objects. The WordPress REST API is designed to make building applications on top of WordPress painless and easy since you can use any programming language that can make HTTP requests and interpret JSON. (WordPress Org. 2019) It is automatically installed and enabled for every version of WordPress since WordPress 4.7. This can be checked by visiting *http://example.com/wp-json* for the site in question. If the API is working, you will get a long JSON string showing available endpoints. WordPress provides a large set of different endpoints for all the main data types you need for building web applications. Some of the most important ones in general, and for the starter developed as a part of this thesis are:

- **Posts:** wp/v2/posts
- **Pages:** wp/v2/pages
- **Categories:** wp/v2/categories
- **Tags:** wp/v2/tags

(Ratnayake 2017 p.393)

The API is designed to be predictable hence the very predictable endpoints. *wp/v2/posts* will return a list of all posts and the information linked to each one, like date, title, and content. *wp/v2/pages* will return a list of all pages and so on. (WordPress Org. 2019)

3 IMPLEMENTATION OF KEY TECHNOLOGIES

The starter site enables developers to easily create static web applications with GatsbyJS and the WordPress CMS. This is done by providing the basic setup of the site, allowing the developer to focus on site-specific functions and needs. This chapter will explain the development of the starter and discuss the key technologies in-depth and how they were used in practice.

3.1 Environment setup

The development process starts with installing GatsbyJS and creating a new Gatsby site. This is done by using the terminal. Because these types of sites need a lot of different dependencies to run properly a GatsbyJS starter was used to create the base for the project. The starter used is called *gatsby-starter-hello-world,* this starter comes without any additional plugins or code which allows developers to stay in control of that only the very essential things are added. Once the Gatsby site has been installed through the terminal with the command, *gatsby new my-site https://github.com/gatsbyjs/gatsby-starter-hello-world,* the site can be run. After this, the very important plugin, *gatsby-source-WordPress*, is installed. This plugin pulls the data from WordPress directly into Gatsby using the WordPress REST API (Gatsby Inc. 2019). The plugin is essential for the starter to work but not essential when developing sites this way, nonetheless it makes development considerably easier. You also need to have a WordPress site running locally, or a public WordPress site in order for the development to start.

3.2 Application structure

Once the Gatsby site is up and running it will have a default file structure that contains these files and folders (see Fig. 12).

```
/
|-- /node_modules
|-- /src
      |-- /pages
      |-- index.js
|-- /static
      |-- favicon.ico
|-- .prettierrc
|-- LICENSE
|-- package-lock.json
|-- package.json
|-- README.md
```

Figure 12. Starter default file and folder structure.

For the development of the starter site, the most important files and folders are *gatsby-node.js*, *gatsby-config.js* in which the queries for what data should be pulled from WordPress are made, and the */src* folder which will contain all the code related to what is seen on the front-end. The *gatsby-node.js* and *gatsby-config.js* files will not automatically appear when the application is first installed, these files will have to be created manually. Inside the *src* directory, folders for templates and components have to be created. The templates folder will contain all, as the name suggests, templates for the programmatically made pages e.g. WordPress posts and pages. These will not be placed in the pages folder since files in this folder automatically become pages with URLs based on their filenames e.g. *index.js* and *404.js* Inside the components folder is where smaller pieces of code that are used throughout the site e.g. the main navigation is placed.

3.3 Queries

The core of this project is the queries made to pull down the data from WordPress, without these the application would not exist. As mentioned earlier Gatsby has a plugin available specifically designed to pull down data from WordPress directly in the build process.

3.3.1 gatsby-config.js

The *gatsby-config.js* file is where the site from which Gatsby should pull the data is defined. This is a simple file yet very crucial to the application working. Below is a snippet of the JavaScript code where Gatsby is being told from what WordPress site to pull the data.

```
    plugins: [
        {
            resolve: "gatsby-source-wordpress",
            options: {
                baseUrl: "example.com",
                protocol: "http",
                hostingWPCOM: false,
                useACF: true,
                verboseOutput: true
            },
        },
    ],
};
```

Figure 13. gatsby-config.js file.

First Gatsby is told what plugin needs to be resolved, then some basic information about the WordPress site is given. All of this is then injected straight into the build process of the site. If the starter developed as a part of this thesis was to be used by anyone else this is essentially the only file that would need to be modified.

3.3.2 gatsby-node.js

For the starter, the application pulls down data for WordPress pages and posts. The queries are made in a JavaScript file called *gatsby-node*. As mentioned earlier Gatsby uses GraphQL for querying data and the *gatsby-source-WordPress* plugin is no different. The basic queries for pulling down WordPress pages and posts for the starter site are shown in the picture below.

```
const pageQuery = `
{
  allWordpressPage {
    edges {
      node {
        id
        slug
        status
      }
    }
  }
}
`

const postsQuery = `
{
  allWordpressPost {
    edges {
      node {
        id
        slug
        status
        format
        date
        categories{
          name
          slug
        }
        tags{
          name
          slug
        }
      }
    }
  }
}
`
```

Figure 14. WordPress pages and posts queries.

As previously noted, the GraphQL way of querying is very simple and straightforward. As figure 14 shows, the queries are written in plain text using words that anyone could understand. Here the basic data needed for posts and pages are queried, these are just a small amount of all the entities that can be queried.

The next step is to tell Gatsby what to do with the data once it is pulled down. This is done in the same JavaScript file.

```
graphql(pageQuery)
    .then(result => {
        if (result.errors) {
            console.log(result.errors);
            reject(result.errors);
        }

        const pageTemplate = path.resolve("./src/templates/page.jsx");

        _.each(result.data.allWordpressPage.edges, edge => {
            createPage({
                path: `/${edge.node.slug}/`,
                component: slash(pageTemplate),
                context: {
                    id: edge.node.id,
                },
            });
        });
    })
```

Figure 15. Creating unique gatsby pages for WordPress pages.

The picture above shows the code snippet that tells Gatsby what to do with the pulled data for WordPress pages. First, the query we want Gatsby to handle is mentioned. After that, some generic error handling occurs. Next, a template for what the page should look like on the front-end is provided, then Gatsby is told to create a page for each of the queried WordPress pages and put them into the defined template. Here Gatsby is also told how the routing for the pages should be handled, in this case, Gatsby is told to put all pages behind the base URL and name them by their slug. Like so: *http://example.com/this-is-a-page.*

The same type of rules is given to WordPress posts, categories and tags with a few alterations. Nonetheless, figure 15 shows the most basic application of how to tell Gatsby what to do with the queried data.

It is also worth noting that the *gatsby-node.js* file can be written in many different ways and it really depends on what type of project and what the developer is most comfortable with. For example, in the starter the queries shown in figure 14 are defined at the top of the JavaScript file, these could just as well be defined in the same block of code shown in figure 15.

3.4 Custom components

When the queries are defined, and the fetched data is displayed on the frontend the rest of the application can be built. Through custom components the site can become whatever is needed.

An important thing to consider is that the *gatsby-source-wordpress* plugin will only fetch post and site data. This means that no installed plugins or custom php will be transferred to the Gatsby application. When transferring an existing WordPress site using the Gatsby-WordPress starter, rewriting plugins and php into custom components might seem like a lot of unwanted work. While it certainly does entail a bit more work from the developer it is also worth noting that there are endless npm packages available for the react library, which make transferring common WordPress plugins, e.g. a masonry image gallery, much easier.

3.5 UI

Since the very basic *gatsby-starter-hello-world* was used as a base for the development project, the application will have very minimal to no UI out of the box. This section will explain how to get the pulled down data to show on the front-end.

3.5.1 Mapping through data

Once the *gatsby-source-WordPress* plugin has pulled down all the requested data it needs to be made available for users to see on the front-end. This is done with a basic JavaScript map function.

```
<div className="posts">
  {data.allWordpressPost.edges.map(({node}) => (
    <div key={node.slug}>

        <p>{node.date}</p>

        <Link to={'/post/' + node.slug}>
          <h4>{node.title}</h4>
        </Link>

        {node.featured_media &&
          <div>
           <img src={node.featured_media.source_url}/>
           </div>
         }

      <div dangerouslySetInnerHTML={{__html: node.content}} />
    </div>
  ))}
</div>
```

Figure 16. Mapping through posts data.

Figure 16 shows how a list of all the posts is being displayed to the front page of the application. It is a simple JavaScript map function being told what to map trough. In this example it is being told to show the date followed by the post title which should also link to the post itself, after this the featured image of the post will be displayed together with the text content of the post. For this to work Gatsby needs to be told what data to use and from where. So, the query of the data that was pulled down during the build process needs to be imported into the JSX file, like shown below.

```
export const pageQuery = graphql`
    query postsQuery{
        allWordpressPost {
            edges{
              node{
                id
                title
                content
                slug
                date(formatString: "MMMM DD, YYYY")
                featured_media{
                  source_url
                }
              }
            }
        }
    }
`
```

Figure 17. Data query for front-end.

This is the basic setup for using the queried data on the front-end. The same base can then be used to display specific posts, list of pages, categories and so on.

3.6 Publishing

Since Gatsby is a static site generator the site can be published very easily. With the terminal command *run build,* Gatsby will compile the site into static HTML and JavaScript files. The site can then be deployed like any other static site.

Adding new posts and pages to the site is done by writing a post through the WordPress backend, it can be done through the WordPress mobile application or the admin dashboard online. Once a post or page is published the WordPress database will update. Every time the WordPress database is updated with new content the build process for the site needs to run again in order for Gatsby to pull down the new data and have it show up on the frontend of the application. Getting the build process to run can be done in two ways, running it manually with the terminal command *run build* or by setting up continuous deployment. Continuous deployment is definitely preferable when working on client projects since this requires no action from the developer and enables the user to view the new content on the site immediately after publishing. Continuous deployment can be setup to automatically run the build process of a site when something happens, for example when a post is published. There are many different ways to set up continuous deployment, this however is out of scope for this thesis.

3.6.1 Surge

To deploy the starter site surge.sh was used. Surge offers free static web publishing straight from the command line. This makes deploying sites very easy and fast. But since the application will compile into static files the options for deploying the site are endless. A live demo version of the starter can now be at http://helpful-kitten.surge.sh/.

3.6.2 GitHub

Besides the application itself being deployed, the codebase was also published to GitHub for other developers to use and learn from. The code base can be found at https://github.com/natalieah/Gatsby-WordPress-starter.

3.7 Starter

The starter was published to surge and GitHub in April 2019. It contains the building blocks for showing individuals pages and posts as well as lists of all available pages and posts. A component for showing categories and tags and the posts containing them is also available. The starter is very simple and contains the basic building blocks with which one can build virtually anything.

3.7.1 Design

The starter is very minimal in its design. The goal was to create a starter that is really easy to customize for all types of projects. Because of this, no inline CSS was added. This resulting in a starter where one could simply delete the existing CSS file and not have to go through each file individually to delete classes and styles.

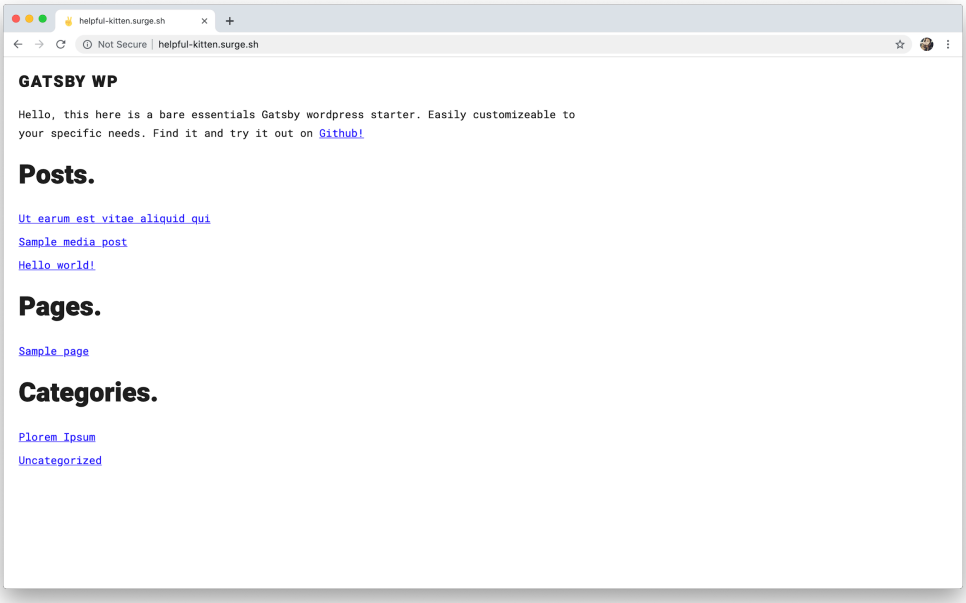

Figure 18. Screenshot of the starter demo site.

4 RESULTS

To better demonstrate how the starter offers a solution to the problems with speed and bloat, a comparison in amount of code, load time and overall performance was done between the POC and the twentyseventeen WordPress theme. The twentyseventeen theme site that was used contained the same amount of content as the POC. Because the POC and twentyseventeen theme are more similar than the starter and twentyseventeen theme, they were compared in order to get more accurate results.

4.1 Lines of code

With pretty printed code the POC has 392 lines of code on the index page while the twentyseventeen theme has 722. For a simple post like hello world the twentyseventeen theme has 645 lines of code as opposed to the POC with only 249. There is quite a huge gap, with the POC having almost half of the lines of code compared to the twentyseventeen theme.

4.2 Load time

The load time for the POC is 1.33s while the twentyseventeen themes load time is 1.64s. For loading a post from the frontpage with the twentyseventeen theme, the load time was 1,04s. Since Gatsby prefetches content in the background no loading has to occur when clicking on a new post, meaning the load time for a post was 0s.

4.3 Overall performance

Using PageSpeed Insights, the performance score for the POC on desktop was 99 of a possible 100, while the twentyseventeen themes score was 93. On mobile the performance score for the POC was 95 and the twentyseventeen scored an 86. The reason for the POC not scoring a 100 on desktop was because of the fonts it needed to load, while the biggest issue with the twentyseventeen theme both for desktop and mobile was incorrectly sized images and unused CSS. Gatsby comes with an image component that automatically resizes and lazy loads images to prevent images slowing down the site.

4.4 Conclusion

Through these tests it is clear that the POC is faster and leaner than the twentyseventeen WordPress theme. The gap between the two sites is considerable but not huge. However, if sites with more content where used I think the gap would grow exponentially, especially considering the twentyseventeen themes issue with images.

5 DISCUSSION AND CONCLUSION

The purpose of the thesis was to find an alternative solution to traditional WordPress sites by developing a GatsbyJS starter. And through the development of the starter investigate whether or not it provides solutions to some of the most common problems regarding WordPress sites. The starter provides a viable alternative to the regular way of developing WordPress sites, and is considerably faster.

From developing the proof of concept, I had already learned that adding too much CSS and custom layouts were something that made the site harder to use since you would first need to take time just to understand or delete all of the added styles. This meaning that the proof of concept had one of the same problems that WordPress sites have, bloat. As seen through the speed test it was not as big of a problem as with WordPress, but it is still noticeable. And since the aim of this thesis is to find a solution to the problems WordPress has, this was not acceptable and so I decided to go a different direction and develop a starter that only contains the bare essentials.

From what I have learned from the process of developing the starter and using it since, I would say it is a starter suitable for most types of websites. In my opinion, the starter is best used for smaller business sites and personal blogs and portfolios, as these types of sites usually contain static content as well as some database driven content. With this said I have yet to test the starter with different types of websites and this would be a good point of further research in order to conclude for what types of projects it might not work.

The starter also works well when one wants to transfer an already existing WordPress site to this type of setup. It offers developers without much experience in WordPress development a way to provide customers with completely custom websites while still having the convenience of a user-friendly CMS. As mentioned earlier, transferring already

existing sites with the starter might entail a bit more work and is something to take into consideration when contemplating to use the starter.

I have not come over any substantial issues with the starter and in the big scope of things, everything has worked really well this far. A possible issue I think could lead to some problems are the technologies used in the starter. Since the technologies are updated regularly it means the developer has to make sure everything is up to date and working together. This is a good thing as no one wants an outdated site, but this can in return lead to some issues and in the worst-case scenario an update of e.g. a plugin could lead to the starter breaking. However, the perk of the starter being a static site is that the site will never break as the development will happen locally and only when everything is working will it be pushed to deployment.

Looking back at the problems stated earlier:

- WordPress sites are very intricate which can lead to unnecessary bloat, ultimately making the site slow.
- Using WordPress for sites that only use a small amount of database-driven content and are otherwise static, is excessive.

I can confidently say I have found an answer to these issues in the form of the starter developed. Unnecessary bloat and slow sites are no longer a problem and the best part of WordPress, the actual CMS, can still be used.

Personally, I have learned so much during the process of this thesis. Before this I had never even used Gatsby, GraphQL or the WordPress REST API, nor had I ever published any of my code for the community to use. Personally, I have been somewhat insecure about my knowledge in WordPress. The experience I did have with it was not great which is what started the journey to finding a better solution. Not only have I gained heaps of invaluable knowledge, but I also have an end product of which I am really proud.

6 FURTHER RESEARCH

It would be important to try out the starter with different type of projects and websites in order to define what type of projects it works best with and where a classic WordPress site would be better.

Another interesting point of research would be to make an “input app” that allows users to input data into an application which then sends it to the WordPress database. If modified this could also be a solution for e.g. clients to input data even if you have only a local install of WordPress. This could even further help eliminate unnecessary data created by WordPress and help create only the data that you really do need for your application. This could also give developers more control to make rules for when users input and publish content, making sure that nothing the user can publish, or change is something that might break the site. Of course, this type of thing can be setup in WordPress with user roles, but a custom WordPress input app might give even more control and make the experience of using WordPress together with Gatsby even better and more optimized.

REFERENCES

Angerer, Dominik, Storyblok, *Headless CMS explained in 5 minutes,* 28.01.2019. Available from: https://www.storyblok.com/tp/headless-cms-explained Accessed 18.03.2019

Bhat, Adi, QuestionPro, *Secondary Research- Definition, Methods And Examples.* 8.2018. Available from: https://www.questionpro.com/blog/secondary-research/ Accessed 14.04.2019

Bilgili, Doğacan, LogRocket, *Building a custom dropdown menu component for React*. 4.5.2018. Available from: https://blog.logrocket.com/building-a-custom-dropdown-menu-component-for-react-e94f02ced4a1 Accessed 29.12.2018

Bos, W., Tolinski, S. 2018, Potluck EP × Vue.js × Headless WP × Typescript & Flow × Productivity × Server Side Rendering × Yeoman*, Syntax*, [podcast]. Available from: https://syntax.fm/show/042/potluck-ep-vue-js-headless-wp-typescript-and-flow-productivity-server-side-rendering-yeoman Accessed 03.04.2019

Camden, R. Rinaldi, B., 2017, *Working with static sites*, O'Reilly Media, Inc., California.

Components and Props, 2018a, Facebook Inc. Available from: https://reactjs.org/docs/components-and-props.html Accessed 29.12.2018.

Features, 2018, WordPress Org. Available from: https://WordPress.org/about/features/ Accessed 23.12.2018.

GatsbyJS, Gatsby Inc. Available from: https://www.gatsbyjs.org/ Accessed 29.12.2018.

gatsby-source-WordPress, Gatsby Inc. Available from: https://www.gatsbyjs.org/packages/gatsby-source-WordPress/?=word Accessed 26.02.2019

GraphQL, 2018c, Facebook Inc. Available from: https://graphql.org/ Accessed 29.12.2018. 3

Hunt, Pete, 2013, Facebook Inc, *Why did we build React?*. Available from: https://reactjs.org/blog/2013/06/05/why-react.html Accessed 27.12.2018

Introducing JSX, 2018b, Facebook Inc. Available from: https://reactjs.org/docs/introducing-jsx.html Accessed 29.12.2018. 2

Krill, Paul. 2014, React: Making faster, smoother UIs for data-driven Web apps, *InfoWorld*, Available from: https://www.infoworld.com/article/2608181/javascript/react--making-faster--smoother-uis-for-data-driven-web-apps.html Accessed 15.5.2014.

Patel, Neil, Neilpatel, *How Loading Time Affects Your Bottom Line.* 4.2011 Available from: https://neilpatel.com/blog/loading-time/ Accessed 02.04.2019

Ratnayake, Rakhitha Nimesh. 2017, *WordPress Web Application Development*, 3d edition., Birmingham: Packt Publishing Ltd, 536 pages.

Rendering elements, 2019d, Facebook Inc. Available from: https://reactjs.org/docs/rendering-elements.html Accessed 1.4.2019.

REST API Handbook, 2019, WordPress Org. Available from: https://developer.WordPress.org/rest-api/ Accessed 10.2.2019

Saxena, Rajat, freeCodeCamp, *Rock Solid React.js Foundations: A Beginner's Guide*. 31.1.2018. Available from: https://medium.freecodecamp.org/rock-solid-react-js-foundations-a-beginners-guide-c45c93f5a923 Accessed 29.12.2018

APPENDICE 1 - SAMMANFATTNING PÅ SVENSKA

1.1 Introduktion

Målet med detta examensarbete är att finna en alternativ lösning till traditionella WordPress sidor, genom att utveckla en GatsbyJS kodmall. Genom utvecklingen av kodmallen önskar jag finna en lösning till de vanligaste WordPress problemen som snabbhet och onödig kod, samt hitta en lösning för problemet med att använda WordPress för sidor med endast en liten del av databas drivet innehåll. Mera specifikt söker detta examensarbete lösningar till följande problem:

- WordPress sidor är väldigt detaljerade vilket kan leda till mycket onödigt innehåll vilket i sin tur leder till en långsam sida
- Användningen av WordPress för sidor som använder en liten del av databasdrivet innehåll, men som annars är statiska, är inte optimalt.

WordPress är efterfrågat då man jobbar tillsammans med kunder. Det här är ingen överraskning eftersom WordPress för tillfället driver 32% av webben vilket betyder att de allra flesta personer använt innehållshanteringssystemet någon gång. Problemet är att WordPress inte är optimalt för alla typers projekt.

Låt oss tänka ett litet företag som vill uppdatera sin hemsida. De behöver en vy för prissättning och en vy för basinformation om de anställda. Utöver detta vill de ha en blogg som de kan uppdatera med aktuell information. All data på sidan är mer eller mindre statisk och behöver inte uppdateras frekvent och kunde byggas som en statisk sida. Bloggen däremot, behöver ett mera komplext innehållshanteringssystem för att fungera. Här kommer GatsbyJS kodmallen in i bilden. Kodmallen möjliggör för WordPress och framändan (eng. front-end) av sidan att inte vara kopplade till varandra på något annat sätt än i form av datan som flödar mellan dem. Utöver detta erbjuder kodmallen utvecklare ett lätt sätt att komma igång med nya projekt eftersom grunden redan finns kodad.

Förutom kodmallen har en koceptvalidering (eng. Proof of concept) utvecklats. Konceptvalideringen utvecklades för att likna det populära WordPress temat twentyseventeen. Genom att efterapa temats utseende kunde det lätt bevisas ifall konceptvalideringen kunde göra samma saker som en WordPress sida kunde. Konceptvalideringen lyckades och efter detta började utvecklingen av den egentliga kodmallen.

Denna sammanfattning följer samma struktur som det egentliga arbetet. Inga källhänvisningar används och läsaren hänvisas till huvudtexten.

1.2 Huvudsakliga teknologier

För utvecklingen av kodmallen användes ReactJS, GatsbyJS, GraphQL och WordPress.

ReactJS är ett JavaScript bibliotek för utvecklingen av applikationer som behöver uppdateringar i real tid. React använder sig av JSX (JavaScript Exended). JSX är ett sätt att kombinera logik och HTML i samma fil och på så sätt bättre kunna se sambandet mellan dem. React är baserat på komponenter. En komponent är en bit av återanvändbar kod som kan injekteras där den behövs. Ett exempel på en vanlig komponent är en navigationsbalk. En navigationsbalk kunde byggas som 4 olika komponenter, 1 komponent eller så många eller få som en själv vill. Knappar, formulär och dialoger är andra bra exempel på React komponenter.

GatsbyJS är ett React drivet ramverk för genereringen av statiska sidor. Gatsby sammanställer applikationen till statiska filer. En statisk applikation har många fördelar så som, snabbhet, säkerhet och flexibilitet. Speciellt Gatsby sidor är kända för deras snabbhet eftersom Gatsby endast hämtar den mest centrala datan, HTML, CSS och JavaScript för sidan i fråga. Gatsby skapades för utvecklingen av sidor där data kan hämtas varifrån som helst. Ramverket har ett stort bibliotek av insticksmoduler (eng. Plugin) som låter användare hämta data från en eller flera källor så som API, innehållshanteringssystem eller filsystem. Gatsby drar sedan ner denna data direkt med hjälp av GraphQL.

GraphQL är ett frågespråk (eng. Query language) för API:er som är fokuserat på att hämta precis den data som behövs. Både GraphQL och GatsbyJS prioriterar snabbhet vilket gör att de två teknologierna passar bra ihop.

WordPress var ursprungligen en bloggplattform men har sedan dess vuxit till ett av de populäraste innehållshanteringssystemen i världen. WordPress använder sig av sin egen REST API som tillåter utvecklare att interagera med sidor genom att skicka och ta emot

JSON objekt. För kodmallen som utvecklades som en del av detta examensarbete användes WordPress REST API för att hämta data till GatsbyJS applikationen.

1.3 Implementation av de huvudsakliga teknologierna

Kodmallen underlättar utvecklingen av statiska webbapplikationer tillsammans med GatsbyJS och WordPress. Detta görs genom att erbjuda ett bas upplägg av koden som låter utvecklaren fokusera på specifika funktioner och behov för sidan. I följande del diskuteras utvecklingen av kod mallen.

Utvecklingsprocessen börjar med att installera och skapa en ny GatsbyJS sida. Efter detta installeras insticksmodulen *gatsby-source-wordpress.* Denna insticksmodul tillåter Gatsby att hämta data från WordPress. Utöver detta krävs också en WordPress sida, sidan kan vara tillgänglig lokalt eller finnas online. Kärnan av kodmallen är de förfrågningar (eng. Query) som görs för att hämta data från WordPress, utan dessa skulle applikationen inte existera. Kodmallen har två viktiga filer där data förfrågningarna förekommer, *gatsby-config.js* och *gatsby-node.js.* I *gatsby-config.js* definieras den sida som Gatsby skall hämta data från. Ifall någon annan skulle använda kodmallen är detta i huvudsak den enda filen som behövs modifieras. I *gatsby-node.js* berättar man för Gatsby vad man vill att skall hämtas från WordPress, tex. sidor och blogginlägg. Nästa steg är sedan att definiera vad Gatsby skall göra med datan då den blivit nerdragen till applikationen, detta sker i samma JavaScript fil. För att sedan få den hämtade datan synlig på sidans framända, används en simpel JavaScrpit map funktion. Genom unika React komponenter kan sidan sedan utvecklats till vad som helst.

Utseendet för kodmallen är väldigt minimalt. Målet var att skapa en kod mall som var väldigt lätt att skräddarsy för att passa många typer av olika projekt. Kodmallen publicerades på GitHub under adressen, https://github.com/natalieah/Gatsby-WordPress-starter.

1.4 Resultat

För att bättre kunna demonstrera hur kodmallen erbjuder en lösning till problemen med snabbhet och onödig kod, utfördes ett test där mängden kod, laddningstid samt generell prestation jämfördes mellan konceptvaliderings sidan och WordPress twentyseventeen temat. Nedan följer resultaten för konceptvaliderins sidan och twentyseventeen temat.

Konceptvaliderings sidan
Rader av kod: 329
Laddningstid: 1.33s
Generell prestation med hjälp av PageSpeed Insights: 99/100

Twentyseventeen tema
Rader av kod: 722
Laddningstid: 1.64s
Generell prestation med hjälp av PageSpeed Insights: 93/100

Genom dessa tester kunde man klart och tydligt se att konceptvaliderings sidan är snabbbare än twentyseventeen temat.

1.5 Diskussion

Målet med detta examensarbete var att hitta en alternativ lösning till traditionella WordPress sidor genom att utveckla en GatsbyJS kodmall. Och att genom utvecklingen av kodmallen utforska ifall den erbjuder lösningar till de vanligaste problemen med WordPress sidor. Kodmallen erbjuder ett praktiskt alternativ till det vanliga sättet att utveckla WordPress sidor, och är betydligt snabbare.

Kodmallen fungerar också bra då man vill överföra en existerande WordPress sida till detta upplägg. Kodmallen erbjuder utvecklare utan mycket erfaret inom WordPress utveckling ett sätt att förse sina kunder med unika sidor men samtidigt ha lyxen av ett användarvänligt innehållssystem. Att överföra existerande sidor kan betyda mera jobb då skräddarsydd PHP logik och WordPress insticksmoduler måste kodas om till React komponenter.

Om jag kollar tillbaka på de problemen jag ville lösa:

- WordPress sidor är väldigt detaljerade vilket kan leda till mycket onödigt innehåll vilket i sin tur leder till en långsam sida
- Användningen av WordPress för sidor som använder en liten del av databasdrivet innehåll, men som annars är statiska, är inte optimalt.

Kan jag med säkerhet säga att jag har hittat en lösning till problemen i formen av kodmallen. Onödig kod och långsamma sidor är inte längre ett problem och den bästa delen av WordPress, det egentliga innehållssystemet, kan användas.

Ytterligare forskning kunde göras genom att testa kodmallen med olika typer av sidor för att bättre kunna definiera till vad kodmallen passar bäst och i vilka fall en WordPress sida kunde vara bättre.