Master's thesis in Computer Engineering
Degree Programme in Information Technology
Software Engineering
Faculty of Science and Engineering
2019

Ekaterina Petukhova

Sitecore JavaScript Services Framework Comparison

Åbo Akademi University

MASTER'S THESIS | ABSTRACT
ÅBO AKADEMI UNIVERSITY

Degree Programme in Information Technology | Software Engineering
May 2019 | 80

Supervisor: Dragos Truscan

Ekaterina Petukhova

SITECORE JAVASCRIPT SERVICES FRAMEWORK COMPARISON

The purpose of this thesis is to compare three modern JavaScript frameworks used in the context of JSS (Sitecore JavaScript Services) and define the most suitable one for a particular project case. The target project is built on Sitecore 9.1 Content Management System; therefore, usage of JSS by JavaScript developers allows efficient maintenance of front-end development with backend performed by Sitecore.

In this work, we will implement the same JSS application within three JS front-end frameworks and libraries, Angular, React, and Vue.js, and compare them between each other in order to determine what framework is more relevant in current conditions for creating a desired application by certain software metrics such as cycle time, source lines of code (SLOC) and active days.

KEYWORDS:
Sitecore, JSS, front-end development, web development, JavaScript, Angular, React, Vue, frontend, JS framework.

CONTENTS

LIST OF ABBREVIATIONS

API	Application Programming Interface
CLI	Command-Line Interface
CMS	Content Management System
DOM	Document Object Model
DSL	Domain-Specific Language
IDE	Integrated Development Environment
IT	Information Technology
GUI	Graphical User Interface
JSS	Sitecore JavaScript Services
JS	JavaScript
JSON	JavaScript Object Notation
MPA	Multi Page Application
MVC	Model-View-Controller pattern
MVVM	Model-View-ViewModel pattern
OS	Operating System
PWA	Progressive Web Application
SASS	Syntactically Awesome StyleSheets
SDK	Software Development Kit
SEO	Search Engine Optimization
SLOC	Source Lines Of Code
SPA	Single Page Application
UI	User Interface

1 INTRODUCTION

Web applications are used as an effective solution for a wide range of business tasks. During recent years, the development of web applications has rapidly grown, gradually crowding out desktop solutions and becoming the most important component of the business in the modern world.

Single page applications are web applications, the components of which are loaded once on a single page and the content is updated when necessary. For instance, on the button click “read more” the block of content appears with the detailed information without sending a request to the web server. The obvious advantage would be the rich user experience (UX) of such web applications. Due to the fact that we have one web page, it is much easier to build a responsive and functional user interface (UI).

Developing single page applications, websites and web applications using a Content Management System (CMS) has many advantages because it usually includes not only user-friendly content management and editing but also Search Engine Optimization (SEO), email and social media marketing, etc. In the past few years, there are more and more businesses choosing commercial CMSs (also called proprietary) over open source ones as long as purchasing paid CMS provides maintenance, security and reliable support [1].

In Figure 1, the four most popular CMS systems are presented: Drupal, Joomla, Sitefinity and Sitecore. The last two represent the commercial type of CMS restricting the usage of the third party plugins that provide additional functionality to the software, therefore, increasing the level of security.

Features	Drupal	Joomla	Sitefinity	Sitecore
Website	https://www.drupal.org/	https://www.joomla.org/	www.sitefinity.com/	www.sitecore.net/
Vendors	Acquia	Joomla	Telerik	Sitecore
Type of CMS	Open	Open	Commercial	Commercial
Latest Version	Drupal 8	Joomla 3.4	Sitefinity 8.2	Sitecore XP 8.1
3rd Party Plug-ins	Yes	Yes	No	No

Figure 1. Open and commercial CMS [1].

Sitecore Experience Platform (Sitecore XP) is a commercial high-performance CMS based on the .NET technology. It is a powerful and easily scalable tool that combines web content management, marketing automation, and analytics, as well as Search Engine Optimization and social media interaction. With the latest version of Sitecore 9.1, there comes Sitecore JavaScript Services (JSS) client Software Development Kit (SDK), which allows the JS programmer to perform front-end development using Sitecore XP as a headless CMS [2].

The headless CMS, in the concept of head to be front-end and body to be the back-end, implements only the back-end side built as a content repository without the front-end part. Headless CMS also allows interaction using RESTful API or GraphQL, therefore, developing the project on the headless CMS, there is no way to present the content; therefore, it supports the use of any suitable front-end tools.

Sitecore JavaScript Services (Sitecore JSS) toolkit gives the possibility to develop applications and web sites with Sitecore XP being disconnected using the code-first approach instead of Sitecore-first. This means front-end developers deal with JS frameworks and libraries for building fully-fledged software without a Sitecore instance.

Sitecore JavaScript Services provide capabilities of Sitecore Experience Platform power and at the same time Sitecore as a headless CMS. Front-end developers can implement the client-side of the web application (or SPA) being disconnected and completely independent from Sitecore installed on a local machine. This approach results in more productive and flexible project implementation, as developers can work with the JS framework of their choice, whether it is Angular, React or Vue.js.

For this thesis, I have chosen the three most popular JS frameworks and libraries [3], Angular, React, and Vue.js with the purpose of comparing them to the chosen software metrics and determining the most relevant one for a specific project. We go through them in detail in the MVC Frameworks and Libraries chapter.

In this thesis, we discuss the principles of JSS applications built with Angular, React and Vue.js JS frameworks based on the same requirements. First, we go through the basic structure and features of each framework, and then we apply the gained knowledge and develop three independent JSS applications.

As the main purpose of this thesis is to define the most suitable and easy-to-use front-end JS framework, we compare the implementation process and the end result of a

provided solution by metrics such as cycle time, source lines of code (SLOC) and active days. On the basis of collected data, we select the one framework or library which is high performing, lightweight and relatively fast in development for this use case.

We will create two different applications in each framework. The first test application aims to present the code structure, in practice helping to understand the workflow of each framework and library. The test application is a photography equipment web application consisting of a certain number of product items, which we will render and perform certain actions with. The second web application will be of the same topic but implemented based on Sitecore JSS and software metrics will be applied for comparison and further conclusions. All data including images is taken from the official Canon website[1].

2 WEB APPLICATIONS

2.1 What is web application

Web applications are used for business purposes such as promoting services and products. Comparing to websites [4], web applications do not just render the content, but also provide business logic. The websites are more informational, whereas web applications aim to increase interactivity with the end user. Most of the applications are hybrids because they convey information and advanced user interactions, such as a call to actions, data manipulation, filling the form, navigation between the tabs or sections and registration.

In comparison with the desktop or mobile applications, web applications do not require installation on a local computer or mobile phone [5]. Web applications do not depend on the operating system (OS), whether it is Microsoft Windows, macOS or Linux, and they are independent of any mobile platforms, such as Android, iOS and Windows. Web applications are cross-platform services.

A Web application runs on a remote web server, and the interactive user interface (UI) is displayed as web pages in the browser. By the very structure of the application, all

[1] canon.com

program logic is located on the web server as one copy, and the user interface is available to any person via the browser. Server-side is implemented using different technologies and programming languages such as ASP, ASP.NET, C/C++, Java, PHP, etc. For the client side, HTML is used for views and CSS for styling. For the formation and processing of requests, JavaScript comes along. Back in the days JavaScript was used for adding interactivity to the web site, for instance, on mouse moving, hover effects, etc [6]. Today JavaScript is a mature scripting language and is used for performing certain actions on the client side without unnecessary requests to the web server. The third key component of the web application is the database where all the data is stored. The database is also located on the remote server.

There are two design types of web application: single page application (SPA) and multi-page application (MPA) [7].

Single page applications do not require reloading of the web page or switching to additional web pages during the use, therefore exclude constant requests to the web server to retrieve the same subsequent content when a user navigates throughout the web site. This is because SPAs use the web caching, so the content is kept during the user session. On the first opening of the web application, the browser receives a bit more data from the server comparing to regular web sites. SPA works by updating parts of the page using Asynchronous JavaScript and XML (AJAX) calls. Not reloading of the page improves the speed of content update and reduces delays between certain actions.

A poor Search Engine Optimization (SEO) [8] can be deliberated as a significant disadvantage of SPA. SEO aims to improve the website's visibility on the Internet and increase organic (non-paid) website traffic. In SPAs, the data is loaded dynamically via AJAX calls, thus, no page refreshing, as there is one page to work with. In addition, SEO is based on the sustainability of the content in every single page of the web site. SPA is a JS based programme, so, in case the search engine spiders and crawlers are unable to render the scripts, the application cannot be indexed. Therefore, search engines such as Bing, Yahoo, Baidu perceive SPA pages as empty. GoogleBot is able to run the JS scripts, handle crawling and indexing, but the possibilities are still limited. There are certain paths for improving the SEO of the SPAs, some changes can be made on the server side, and some companies such as Google are continuing the creation of new solutions.

Multi-page applications provide a more classic way of website development. Navigation between traditional separate web pages simplifies the SEO and has more natural application navigation for users. AJAX technology simplifies the MPA so that only a sufficient change of the content or requests to the server results in the rendering of a separate page.

As a conclusion, web applications provide the benefits, over desktop applications or websites, by the key parameters such as business logic supply. They retain the functionality on all devices, are based in the cloud, and indeed much more convenient to use. Single page applications improve user experience and interactivity.

2.2 TypeScript

Typescript [9] is a superset of JS and a powerful toolkit with Integrated Development Environment (IDE further) support. The Typescript was developed by Microsoft C# lead architect Anders Hejlsberg. The open-source scripting language compiling into JS was created for the purpose of empowering capabilities of JS and code readability of the program. Typescript is a smooth transition from static development to dynamic, providing a simpler more understandable way of writing code.

As a great benefit of TypeScript can be considered its scalability and easiness in the browser and device portability. It works on all platforms on which JavaScript works. Therefore, the main difference from analogous technologies lies in the fact that TypeScript does not need a special virtual machine or the runtime environment. TypeScript supports type definitions for currently existing JS libraries by having a .b.ts file extension, therefore containing libraries in itself.

An obvious benefit is optional static typing added in the TypeScript structure; the coder directly specifies to the system the type of each variable declared, which is a preventive measure against getting errors in further project development.

Benefits:

- Early detection of errors;
- Easy code analysis;
- Improved code readability;
- Improved IDE support;
- Promoted dependable refactoring.

For the last few years, TypeScript support community grows continuously, as does the acceptance of this scripting solution and assistance of its syntax in Microsoft Visual Studio software, as well as text editors plugins such as Sublime Text and Atom, Visual Studio Code, etc.

2.3 Virtual DOM

The Document Object Model (DOM) is an in-memory representation formulated based on the HTML document taking into account JS scripts, which can manipulate the DOM structure. DOM content can be checked through the View Source option in the Browser. According to the DOM, the HTML document is the hierarchy tree consisting of HTML tags, which create element nodes in that tree.

Initially, the real DOM was not designed for dynamic UI updates, and changing its structure via JS scripts appears to be slow and decreases the speed of the page load. On the other hand, there is Virtual DOM rendering allowing dynamically update parts of the code on state change.

Virtual DOM [10] is the lightweight copy of the Real DOM, so all changes are applied to the copy at first, then both DOMs are compared to each other and only changed elements are updated. Elements of Virtual DOM are simply JS objects. For instance, React creates objects such as React.div instead of div element. This supports quick manipulation of those without affecting the Real DOM or going through the DOM API. Therefore, practically, after specific changes, we as developers send a signal to the Real DOM to update the changed parts of the code, without reloading of the whole node tree. This can be done by two methods: dirty checking, meaning regular updates of the DOM within a certain period of time regardless of whether the nodes were manipulated or not, and observable method, which attaches the event listener to the elements and updates the DOM on elements state change, thus, as a result: no changes in the code - no DOM tree reload needed.

The usage of Virtual DOM created and manipulated by JS objects increases the client-side web application performance in times and improves the user experience.

2.4 NODE.JS

Node.js [11] is an asynchronous JS server framework and has been created by Ryan Dahl in 2009 and. Node.js is event-driven and allows network full-stack web application development taking the role of a web server.

NPM stands for a Node Package Manager that helps to manipulate the libraries and tools used for the project wrapped into packages. NPM is included in the Node.js, so after finishing of Node.js platform installation, both tools with be set up. Main commands of the Node.js can be seen in Appendix 1.

We will use node.js in the projects for server-side development and for access to the file system and different kinds of APIs. Node.js allows quick installation of frameworks and libraries, as well as including them to the project. It is easier to scale creating a project based on Node.js. When thousands of users connect to the server at the same time, Node works asynchronously, setting the priorities and allocating the resources in an intelligent way.

2.5 JSX

JSX is a syntax extension for JS language similar to Extensible Markup Language (XML) [12]. JSX is used in React library. It is is not obligated to use with React, but it makes the code more polished and visually it is more concise to read HTML-like lines of code compared to JS functions and object literals, especially for designers.

By default, React does not contain installed JSX, so in order to include it in the project, JS compiler is needed, for instance, open-source next-generation compiler Babel [13]. Therefore, this utility transliterates the code written in ES6 or ES7 (ES-2015 or ECMAScript-2015) into ES5 standard.

The principle of JSX work is simple: HTML / XML code structure, for instance, DOM tree elements, which comes along with JavaScript code in the same file. JSX expressions define the UI elements, they can include any JS code inside the curly brackets. After compilation JSX becomes a regular JS code, which allows using of if-else and loop statements or assigning its value to a constant variable.

JSX:

```
<div>
```

```
  <input type="text" placeholder="Search" onChange={
this.filterList }/>
  <List items = { this.state.items } />
</div>
```

The same code after Babel compiles JSX into React:

```
React.createElement(
  "div",
  null,
  React.createElement("input", {type: "text", placeholder:
"Search", onChange: undefined.filterList }),
  React.createElement(Öist, {items: undefined.state.items })
 );
```

3 MVC FRAMEWORKS AND LIBRARIES

JavaScript is one of the most known and trendy programming languages fully interacting with HTML and CSS, manipulating with the web page, interacting with the user and, to some extent, with the server. The main advantages of JS are flexibility, scalability and reliability. Currently based on that language, there are many frontend frameworks and libraries created that allow developing dynamic reusable web applications and clarifying project patterns by well-defined application architecture. The most common patterns are MVC (Model-View-Controller) [14], MVP (Model-View-Presenter) and MVVM (Model-View-ViewModel). In this thesis, we concentrate on MVC pattern frameworks and discuss their benefits.

The frameworks themselves are represented as an abstraction providing generic functionality that is fulfilled by user-typed code. Using a framework developer implements a specific type of application, relying on platform precoded standardized structure. Compared to the JS library, there is an inversion of control, which means that not the application code calls for the framework, but the framework calls for particular parts of user code when needed.

Modern MVC-based JS applications have fewer lines of code compared to the classic MVC, which allows faster maintenance and clearer readability, positively affecting development speed and the ability to work on modules by different developers operating on the same project. One of the following benefits of creating a JS application is the ability to quickly create a mobile and/or desktop cross-platform application out of a web/desktop version, for instance, using a platform for mobile

application development with an open source code, such as Apache Cordova [15] or PhoneGap (platform based on Apache Cordova, also known as Apache callback).

With the advent of the MVC pattern by Trygve Reenskaug, the developers' workflow has been simplified and become more constructive and organized in the way of responsibility area separation. The original idea of creating a Model-View-Controller architectural pattern is for the purpose of user interface development [16] and decoupling of the presentation part from model and controller. This allows manipulating each layer without affecting another as well as the possibility to have a few screen presentations for a single model [17].

The user interacts with the MVC application in accordance with the natural cycle: the user takes an action, in response to which the application changes its data model and delivers an updated view to the visitor. Then the processing cycle is repeated accordingly. This fits well with the web application (and SPA) schema provided in the form of HTTP request and HTTP response sequences.

Web applications that need to combine several technologies (for example, databases, HTML markup, and executable code) are usually divided into a number of layers. The resulting patterns naturally fit into the concept of MVC (See Figure 2).

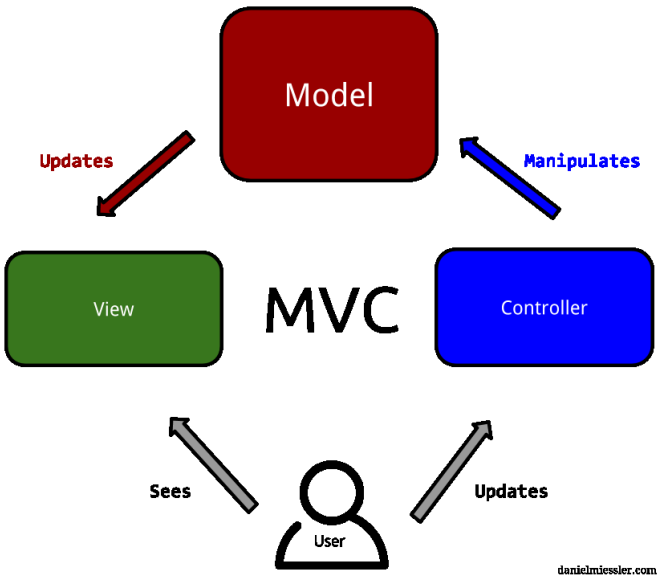

Figure 2. MVC pattern.

- Model (Business Logic), consists of application functional business logic, so-called application logic. Should be absolutely independent of other parts, a central component. Model on change sends a notification to the View to update.
- View (Client, UI), has access only to read data from Model and update the screen presentation; interacts with the user. Responsible for the look and feel of the web application.
- Controller (Server), reacts on outside requests and actions and changes and updates Model.

Developing a heavy lifting, a dynamically complex client-side project requires much coding effort and makes it crucial to choose MVC framework to work with. There are many MVC frameworks and libraries [18] managing the code architecture and making it easy to maintain and test.

Front-end side is one of the most dynamically developing areas of modern web development. Thus, it has overgrown with a variety of tools such as libraries and frameworks designed to support the work of front-end developers. As a result, there is a question raised: what tool would be the most suitable for a certain project?

Within the latest trends [19] of the years 2018 and 2019 [20] the three most outstanding front-end MVC frameworks Angular, React and Vue.js are at the top of the list respectively. In addition, as long as JSS provides client SDKs for all of them we discuss the structure and principles, advantages and disadvantages of each. Figure 3 represents the statistics in per cent of the JS frameworks by the level of popularity. According to the results, React has the highest rating in usage.

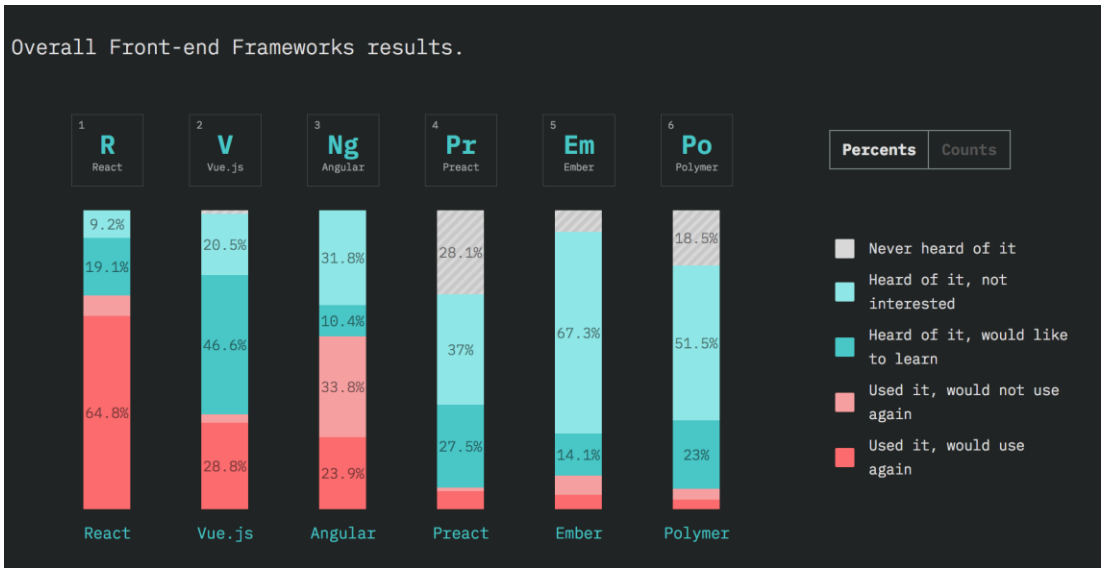

Figure 3. The most notable front-end MVC Frameworks and libraries [19].

In practice, choosing the preferred front-end framework should be done based on web application scale, resources and project goals. Comparing these three frameworks, we will understand which one will be the most appropriate for the particular presented project.

According to Google Trends statistical data [21] on the time period of the full year 2018, the leading position belongs to Angular, even though React is tagging along and second half a year React tops among the three participants of comparison. The results are calculated out of search queries. Over the last year, the usage of Angular and React in application development stays up on a high level comparing to Vue.js. This is because Vue is comparatively new (see Figure 4).

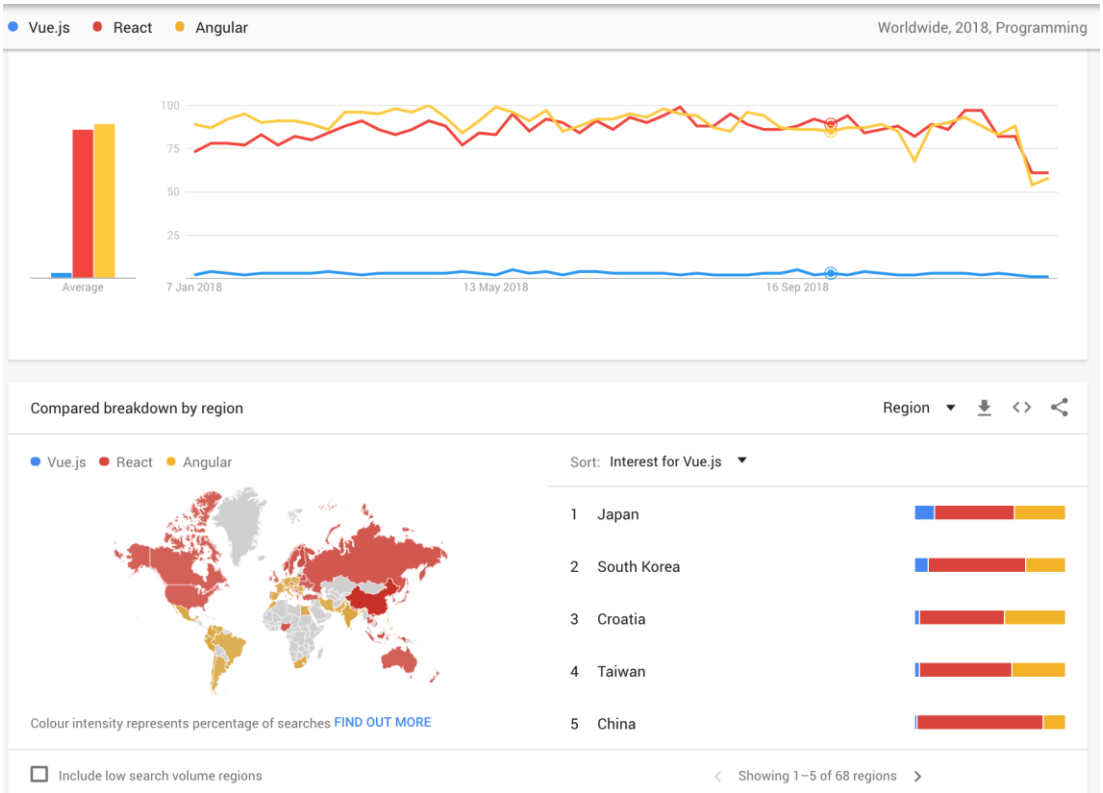

Figure 4. JS frameworks and libraries comparison [21].

At this point, Vue.js is used in Japan and South Korea for 9-13% only, which is understandable, taking into account that the framework development started only in 2013. Based on AngularJS, Vue.js is a progressive framework suitable for UI interface development, gaining popularity for the latest 4 years.

By the region breakdown, China, Indonesia and Nordic countries have the highest interest in React framework comparing to Angular, while in the Peru, India and Columbia Angular is involved much more. See Figures 5 and 6.

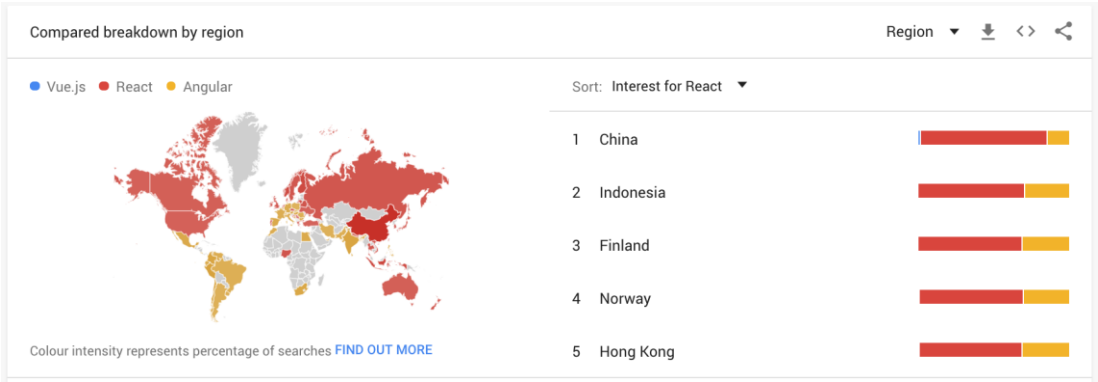

Figure 5. Interest in React [21].

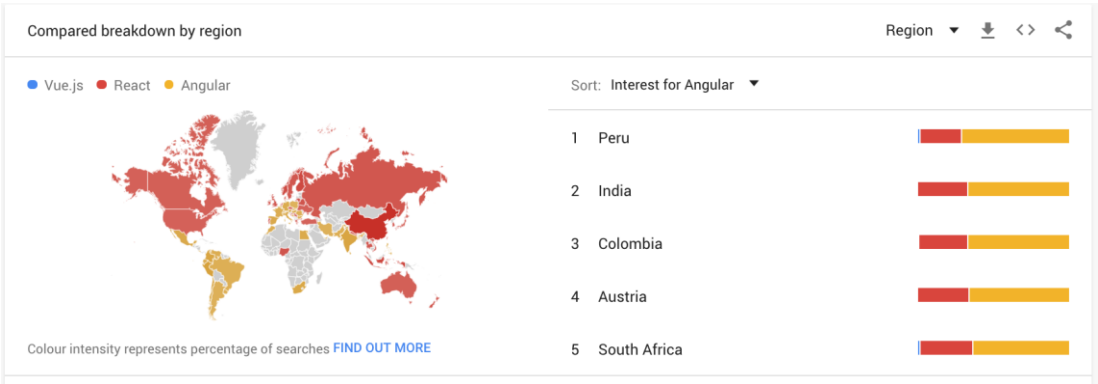

Figure 6. Interest in Angular [21].

As a conclusion, React and Vue.js continuously gain their popularity due to the fast development and growing community support. On account of the information provided by Google Trends, we will perform the comparison of these frameworks and libraries during the application development and observe their benefits and drawbacks.

3.1 ANGULAR

Angular [22] is a rich client-side MVC framework backed by Google since the year 2009. It was designed for the development of web and SPAs, providing reusable UI components and extending HTML with two-way data binding with the simultaneous separation of DOM manipulations and application logic.

Original AngularJS is an open-source framework and is supported not only by Google team but also by many freelance developers who helped to create new packages allowing web development process go much faster. Therefore, as a result, in September 2016, a very new framework appeared called Angular 2. This version is rewritten from the scratch and is not backwards compatible, so as advised by programmers, web applications based on AngularJS (or Angular 1.x) should be migrated to Angular 2 or latest versions because as long as developers concentrate on newest versions support, there is less support for AngularJS.

One of the key differences between the original AngularJS and Angular 2 is that the first version is based on JS language, and from the second version and on developers switched to TypeScript as the main programming language. The structure of this programming language is discussed in the following chapter. In this thesis, we concentrate on Angular 2+ versions of the framework and consider the practical strengths and weaknesses.

Angular version 3 [23] was skipped in order to synchronize versions of angular and angular route. Angular 4 was released 23 of March 2017 and introduced HttpClient and a new router life cycle events. Angular provides its own support policy and schedule for releases. The support consists of two phases: 6 months of active support and 12 months on long-term support from the release date (see Figure 7 for examples). Long-term support for Angular 4 has ended in September 2018.

VERSION	STATUS	RELEASED	ACTIVE ENDS	LTS ENDS
^7.0.0	Active	Oct 18, 2018	Apr 18, 2019	Apr 18, 2020
^6.0.0	LTS	May 3, 2018	Nov 3, 2018	Nov 3, 2019
^5.0.0	LTS	Nov 1, 2017	May 1, 2018	May 1, 2019

Figure 7. Support plan for Angular 5, 6, and 7 [23].

Angular 5 was released in November 2017 and includes Progressive Web Application (PWA) support, and updates to Material Design (visual language for design). PWAs are the applications created with specific tools for certain purposes. Among tools there are technologies and purposes are reliable, fast and engaging application. Angular 6 [24] release was in May 2018 and is more concentrated on a toolchain of the framework and such attribute such as ng update, ng add.

The latest current version of this front-end framework is Angular 7, released in October 2018. It is backwards compatible with Angular 6 and has a row of benefits such as a new ng-compiler allowing reducing the big application size almost by two times as well as the eight-phase compilation; 418 modules are available for a separate use preventing multipurposeness.

New major versions are released every 6 months, so Angular 8 stable release date comes to March/April 2019.

Projects such as Udacity, YouTubeTV, and Freelancer are based on Angular 2+ (4.4.4, 2.4.8, 5.1.0 respectively). Large applications as InVisionApp and iStockPhoto are still comfortable using the AngularJS framework for their business. Therefore, there is no such a strong opinion of full migration of all AngularJS project to Angular 2+.

AngularJS has been changing the pattern during these years from Model-View-Controller to Model-View-ViewModel back and forth so, in the end, they have decided to create a paradigm MVW, which stands for Model-View-Whatever [25]. The reason for implementing that was the desire to develop better applications and solution not concentrating on design pattern.

According to GitHub stars [26], on November 2018, AngularJS has 59 300 stars and has been forked for 29 000 times, and Angular newer version has 42 700 stars and has been forked almost three times less - 10 800 times. GitHub stars are used for the purpose of the rating of the most interesting projects and topics, simplifying the user access to them later. The higher the rating is the more appreciation has been given to the project or topic. Also starring the repositories allows discovering other similar projects. As a result, even if AngularJS is older than Angular and has bigger numbers because of that as well, we conclude that there is this interest in the development of AngularJS based applications (see Figure 8).

Figure 8. GitHub stars of Angular and AngularJS.

Advantages (see Figure 9):

- Reliable huge support community consisting of permanent development group and of freelance developers who contribute to the deployment of the improvements of the platform;
- Dealing with data binding based on Angular elements decreases encountering errors in the code;
- Component-based approach, therefore higher reusability of code;
- Support of different MVC pattern elements;
- HTML is operated by Angular as a programming dialect, simplifying user interface representation, therefore making it more intuitive and different comparing to other interfaces based on JS.
- Angular CLI - a command-line interface that allows creating Angular projects fast, maintain files and perform specific tasks such as testing, simplifying the development process especially for beginners.

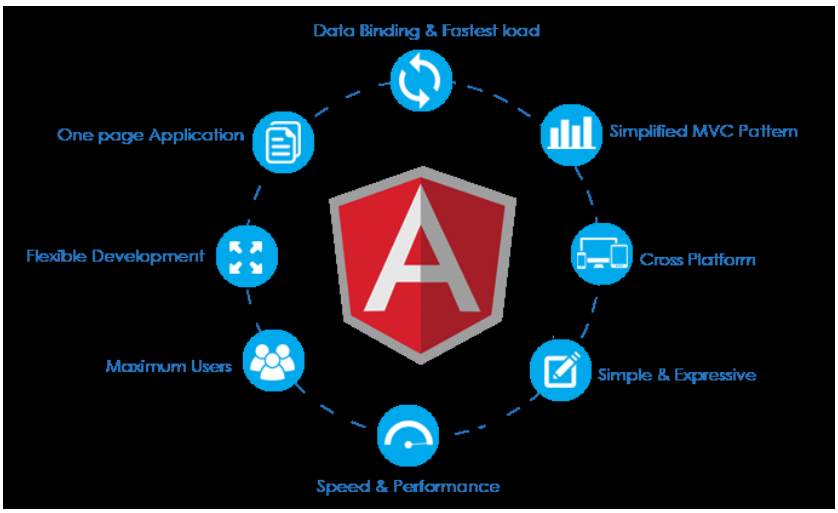

Figure 9. Features of Angular framework [medium.com].

Disadvantages:

- Most suitable for the development of SPAs, and possibly do not meet the requirements of a programmer who works on a more complex project;
- Angular API is extremely large, so it consumes time to deal with its concepts, learn TypeScript and read all the documentation, though latest versions have lower learn curve;
- Angular uses DOM, therefore the speed and efficiency are getting lower comparing to React.

3.2.1 Angular application

3.2.1.1 Setup

We will create an application based on Angular version 7, so far the latest one. Angular requires Node.js version 8 or 10 installed on the machine (Chapter 2.1 Node.js). In order to install Angular globally, we need to run a certain command at the command prompt in the terminal [27]. Next, we create a workspace and the app itself, move to the workplace and serve the application. See Appendix 2.1 and Figure 10 as a result of the installation.

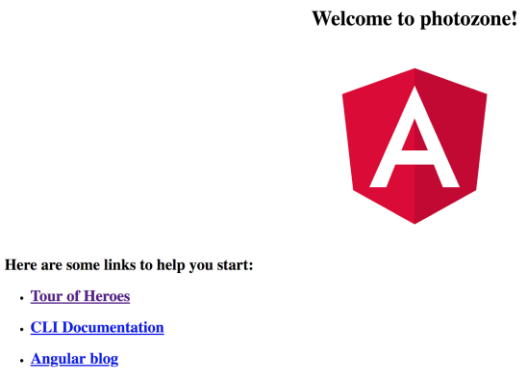

Figure 10. Application Shell. Start page after Angular application is created.

In all our projects, we are going to use Bootstrap 4 frontend library, which allows fast prototyping and developing due to ready-made components and styles. Among the advantages of this library are responsive grids and Sass variables and mixins. In order to use Bootstrap with our Angular application, we have to install ngx-bootstrap [28], which is a realization of all Bootstrap components powered by Angular. For instructions on the install of the Bootstrap and module import, see Appendix 2.2.

3.2.1.2 Components

After the opening of the project in the editor, moving to src folder and then the app folder brings up four important component files [30]:

- app.component.ts - the component Typescript class code;
- app.component.html - the component HTML template;
- app.component.css - the component SCSS styles;

- app.component.specs.ts - the component test spec file;
- app.module.ts - all imports and generated files of the project are stored.

Angular CLI provides easy and fast creation of the new component by typing the following command in the terminal inside the workplace:

```
ng generate component products
```

This request generates a new folder inside app folder with the name of the component and will consist of four files (all except the last one as above in the list), though the 'app.' is replaced by 'products.' in this case. The app.module.ts file is automatically updated with importing the ProductsComponent and declaring it in @NgModule.

```
import { ProductsComponent } from
'./products/products.component';

@NgModule({
 declarations: [
   AppComponent,
   ProductsComponent
 ],
…
```

First, we create a class for product item calling it product.ts. This file defines the properties of the product:

```
export class Product {
   id: number;
   title: string;
   type: string;
   image: string;
   trademark: string;
   price: number;
   description: string;
}
```

Now we create a mock file that contains all products data by declaring an array that is wrapped into the constant. The file is located inside the app folder and called products.ts. In this file, we import Product class. See Appendix 2.2.

All images are located inside the src/assets/images folder. Next step is to display the list of products in products component. In products.component.ts we import products array and declare a variable 'products' out of it. See Appendix 2.4.

In the component template file (products.component.html) we insert the HTML structure and add Angular loop *ngFor. See Appendix 2.5.

In order to render the component on the page, we add to app.component.html tags <app-products></app-products> that import the component template with generated data in it. The final look of the main template is:

```
<div class="jumbotron">
 <div class="container">
   <h1>{{title | titlecase}}</h1>
   <p class="lead">Photography equipment</p>
 </div>
</div>
<div class="container">
 <app-products></app-products>
</div>
```

The creation of the basic components in the Angular application is straightforward and relatively fast (see the final result in Figure 11). Wide documentation provides useful solutions and practical advice.

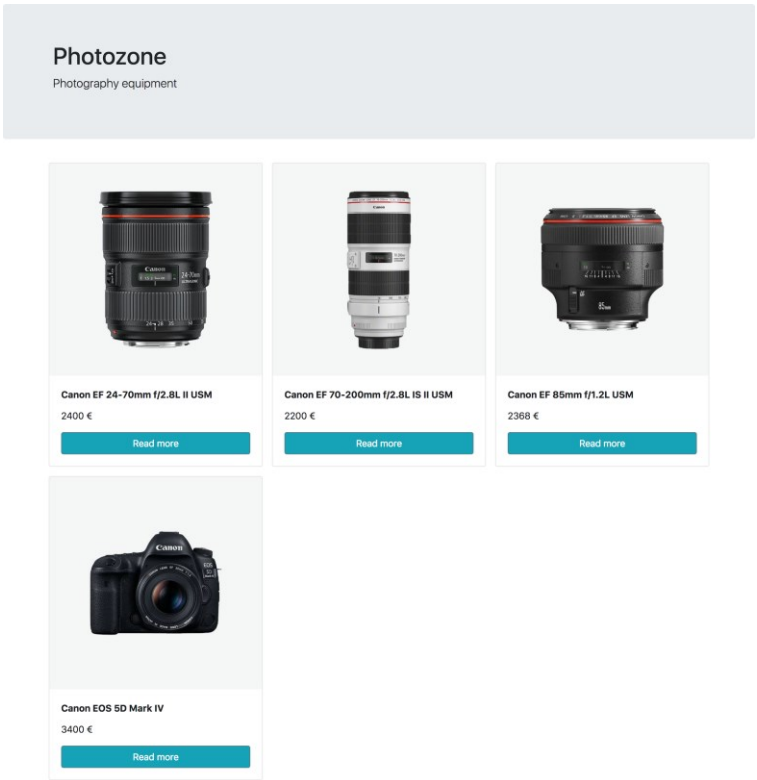

Figure 11. The final look of the Angular application.

3.3 REACT

The banner on the main page of official React JS website states: “React is a JavaScript library for building user interfaces” [31]. It is clear that there is no sense to compare Angular with React as far as they are the framework and the library respectively. Therefore, React is the rendering of representation and Angular as a framework includes the representation layer. The library itself is a collection of functions with the well-defined interface by which the desired behaviour is invoked. As a benefit: reuse of behaviour and modularity, simplified creation and maintenance of UI components. The only thing that is specific for particularly React library is JSX (discussed in chapter 2.5).

React was developed by a software engineer of Facebook Jordan Walke and deployed in 2011; in May 2013, it was open-sourced at JSconf US (Conferences for the Javascript community in the United States). Originally, React is coming from the prototype of Facebook Ads application, which has to be refactored in a short period of time. During summer 2013 React becomes available on JSFiddle (online text editor for texting and sharing code snippets), Ruby on Rails (a server-side web application framework), and in Python applications. The latest version of the framework currently is 16.8.

Large size companies such as Facebook, Netflix, PayPal, Airbnb, etc have their websites built using React, and the amount of projects grows consistently. In GitHub, the amount of stars is nearly three times more than for Angular, as well as two times more forks. This results in expanding of the React use as a JS library chosen for projects.

Advantages:

- The high speed of work due to Virtual DOM and JSX;
- Declarative in a way of creating interactive UIs, so the developer can be sure how the component will be presented, based on the source code;
- The component-based structure allows reusing of them and combing into more complex UIs;
- Possibility to render on a server accelerates the website workload.

Disadvantages:

- The rapid development speed of ReactJS environment, which enforces coders to relearn ways of programming [32]; this temp does not fit all the developers;
- As a result of the previous statement - poor documentation that is not updated as fast as development updates arrive;
- JSX needs to be learned by developers requiring time to get all the complexity of it.

3.3.1 React Developer Tools

In January 2014, Facebook offered to Chrome DevTools [33] a new extension for React. React Developer Tools [34] adds the possibility of React debugging through the browser, enabling inspection of the components hierarchy. Using this extension it is easy to change the props and state of the specific component in the React tab located next to the main tabs such as Elements, Console and Sources, as well as explore components themselves and their subcomponents (children) in the tree (see Figure 12).

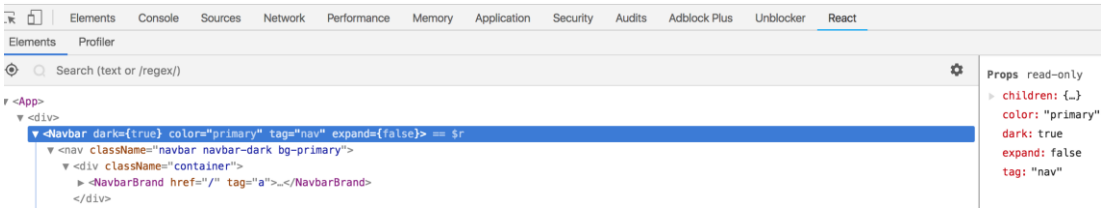

Figure 12. React Developer Tools extension.

3.3.2 React application

3.3.2.1 Setup

In order to install React application package globally we need to run certain commands at the command prompt in the terminal, and in the desired folder create a new application using ‘create-react-app’. See Appendix 3.1 and Figure 13.

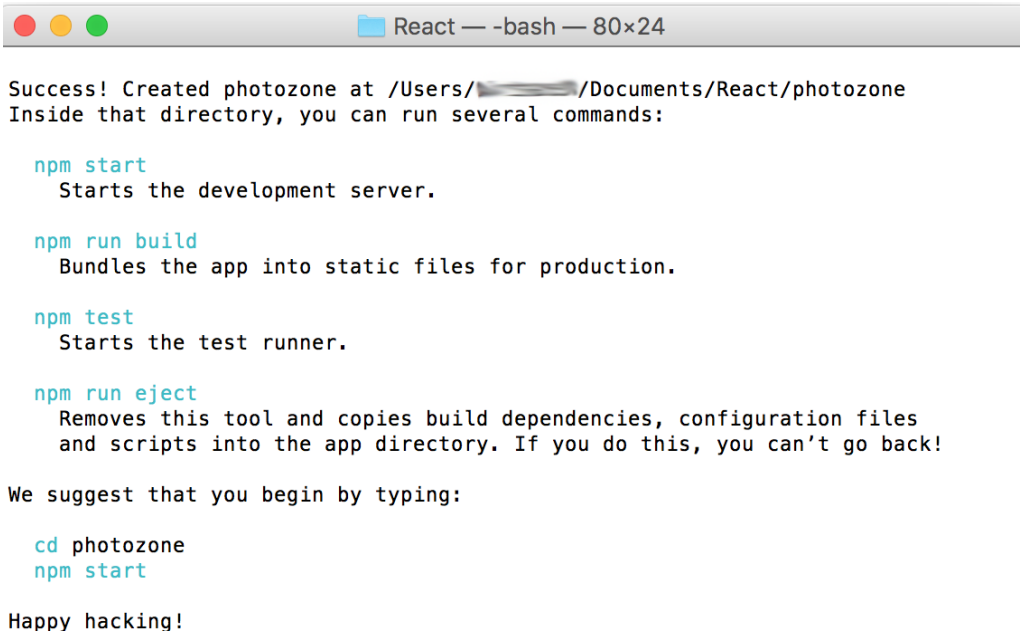

Figure 13. React application is successfully created. Terminal.

Starting the built-in server, rendering the web application in the browser using command npm start:

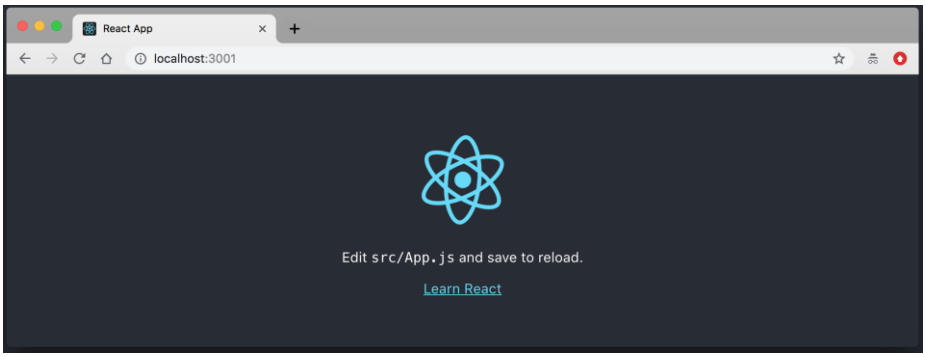

Figure 14. Start page after React application is created.

As it was mentioned in the Angular application, we are using Bootstrap 4. In order to use Bootstrap components in the React application, a reactstrap package should be installed as a project dependency. React-popper is another package that we are going to use during project development. React-popper is an advantage of the render-props React pattern.

3.3.2.2 Components

In the src folder of the application, we create a subfolder named components where all the components will be located. The main component will be responsible for rendering of all other subcomponents and is imported in App.js file.

At this point, we render a jumbotron within the header and a paragraph (See Appendix 3.3). Importing jumbotron from reactstrap allows usage of this class as a tag, simplifying the process of coding. Instead of a class attribute, developer should use className as long as the class has been reserved by JS. Passing props to super() allows accessing this.props inside the constructor.

Next step is to import MainComponent into App.js file by inserting Main class as a tag into an empty div. This is done by adding tag <Main /> and importing that file. See Appendix 3.4.

Next component is rendering of all product items that are located in a separate JavaScript file. The structure of the file is a list containing item type, title, image, trademark, price, and description. Data file itself called products.js and is stored in a folder shared on the same level as folder components (See Appendix 3.5). In the public folder (the same level as src folder) we create a new assets folder which contains images folder. All images are retrieved from this location.

This products.js file and ProductComponent are imported in MainComponent as following (See Appendix 3.6 for the file code):

```
import { PRODUCTS } from '../shared/products';
```

Now in the same file, we expand constructor initializing the state component, which stores and reflects its internal state. In our case, we add products key and assign products array value.

```
this.state = {
      products: PRODUCTS,
  };
```

In addition, in the same file in order to dynamically render all items we add Products as a tag sending the data in products props, to the render method class right after Jumbotron component:

```
<Products products={this.state.products} />
```

The new component named ProductComponent has been created (See Appendix 3.6). As a result, React dynamically renders all the items existing in the data file, which simplifies the process of adding or removing of the products:

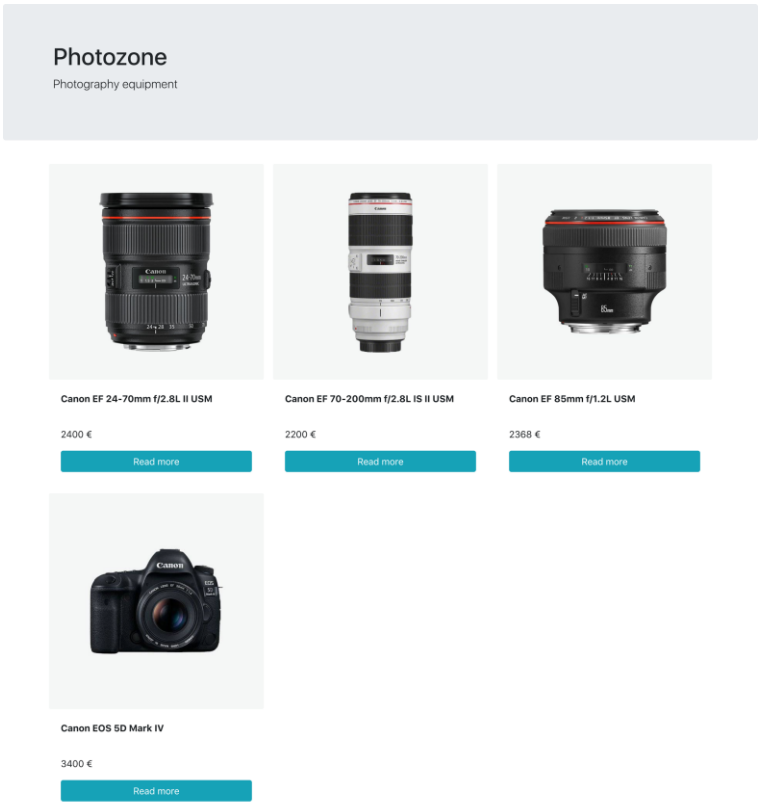

Figure 15. The final look of React application.

3.4 VUE.JS

Vue.js [35] is a relatively new progressive MVC framework, created by Evan You, a former Google company developer. He started development of this framework in 2013 and presented the first public release in February 2014. Currently, the latest stable version is 2.6 [36], and it was released in year 2016. The version 3.0 release is planned for late 2019.

The Vue.js is a lightweight and remarkably fast JS framework for user interface creation suitable for gradual implementation. Companies such as Alibaba, Behance, Facebook, Netflix, and GitLab [37] have implemented their web applications utilizing Vue.js framework.

The main concepts of Vue.js are constructor, components, directives and transitions, which is a simpler and lighter architecture compared to other JS presented frameworks. Practically, it is closer to React, but templates are similar to Angular (HTML + Mustache). Vue.js focuses on the root library, placing other issues into additional

libraries, for instance, routing and managing the global state of the application (Vue-router and Vuex respectively). Vue.js is approachable and uses ECMAScript 5, thus supports the compliant browsers. High performance and flexibility are the necessary attributes of the framework.

3.4.1 Vue CLI

Vue Command Line Interface is a complete system for quick development on Vue.js, aiming to become a standard tool for Vue.js environment [38]. Projects created based on Vue CLI already have some ready-made initial template code and file structure, and an installed configuration, for instance, a configuration for a Webpack compiler and module bundler [39], as well as a number of basic files.

Vue CLI allows prototyping, working with plugins and customization of new project development. Vue CLI works on Node.js version 8.9 and higher. Easy NPM installation is followed by clear selection of project features, such as use TypeScript, build PWA, work with Router, Vuex and CSS preprocessors. Based on selected features, additional parameters can be specified.

Vue CLI is used for scaffolding of a new project and fast prototyping (commands vue create and vue serve respectively). In addition, it is possible to start a new project through the Graphical User Interface (GUI) using command vue ui.

3.4.2 Vuex

As stated in the official documentation of Vuex: it is a ‘state management pattern + library for Vue.js applications’ [40]. The idea of Vuex is based on Flux [41]: an application pattern for client-side coding created by Facebook in order to be able to share a single state among multiple views. Vuex does not implement all Flux, but simply a subset. Therefore, as a centralized global state store, Vuex allows keeping all states, changing and reactively updating them in views.

Vuex provides communication between distant sibling components distributing the shared state without parsing the props through child/parent references. This approach saves the coding time, improves the application structure and becomes a necessary tool in Vue.js environment while the web application grows in complexity.

Vuex store is called a single source of truth, because SPA components that refer to a shared state are not granted to store the state copy locally. This prevents conflicts that may occur due to different copies allocation. However, components are able to keep local states that are not used in other sibling components. Components are restricted to update the state values directly in the Vuex store. This process goes through the mutation functions, that record all the changes received from different views and keep tracking of commits (see Figure 16).

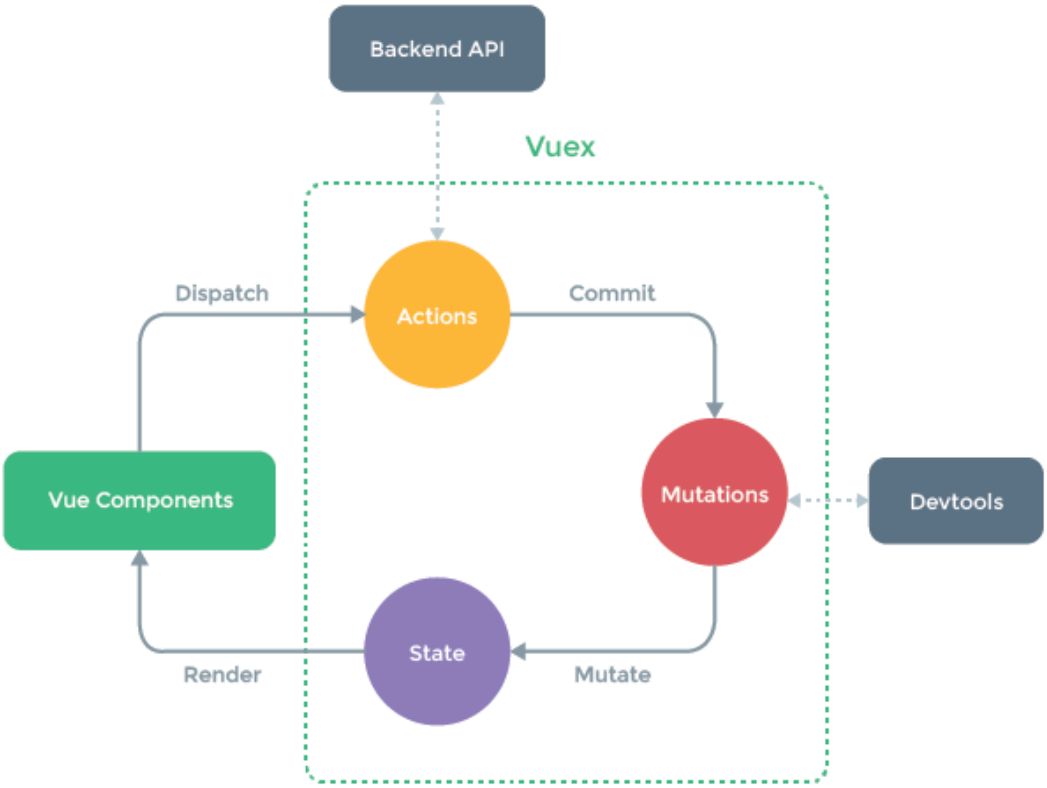

Figure 16. The Vuex principle [40].

3.4.3 Vue.js Developer Tools

Vue.js has provided the Chrome and Firefox browsers with a possibility to inspect and debug Vue.js applications through the Vue.je devtools extension. After installation, the extension automatically detects the use of Vue.js on the website and if it is present, a new Vue tab is added to the Developers Tools (see Figure 17).

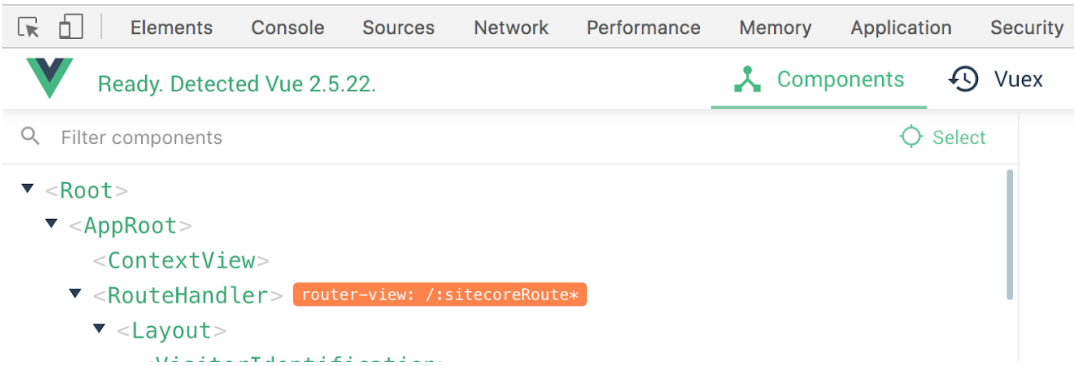

Figure 17. Vue.js devtools.

3.4.4 Vue.js application

Creating an application based on Vue.js is simple enough, and documentation provides the total guidance with understandable examples of the code.

3.4.4.1 Setup

First, we need to install Vue CLI globally with NPM. In order to ensure that installation was successful check the version of Vue CLI.

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

Vue CLI command vue create photozone creates a new project inside the folder where the terminal is open. After running this command, CLI suggests two preset of the application: default with babel and eslint (pluggable linting utility for JS and JSX [42]) enabled and manually select the features of SPA. For this simple application, I have chosen to have babel, CSS pre-processor and eslint options. In case of the application that is more complex, I could choose Vuex and Router options as well. After selection, Vue CLI asks for additional parameters of the project (See Figure 18). During the creation of the project, CLI initializes GIT (free and open source version control system [43]) repository and CLI plugins (npm packages, for this project it is babel and eslint). In order to run the project, we move to the project folder and run npm run serve command (see Appendix 4.1 for all commands used).

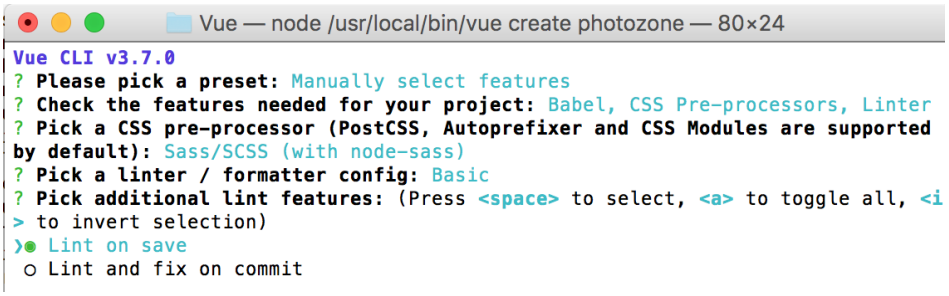

Figure 18. Vue project parameters in Vue CLI.

The project is running and now we install BootstrapVue [44] by running the command in terminal and import Bootstrap V4 framework into the main.js file located in the src folder of the project (See Appendix 4.2).

Any work in Vue.js starts within the creation of a new instance:

```
new Vue({
          // options
})
```

3.4.4.2 Components

The .vue extension indicates that this file is a single-file component of Vue. The component consists of three parts: <template>, <script> and not mandatory <style>. In our project, we define styles in a separate file. Now we will create a new component called Jumbotron.vue using BootstrapVue. We locate it in the components folder. After updating the App.vue file to include a new component (See Appendix 4.3) we have a new component rendered.

The second component is Products.vue renders the list of products located in separate JS file products.js. All images are located in the public/assets/images folder. For the data to be rendered v-for directive is used (See Appendix 4.4). The final look of the Vue web application including Jumbotron and Products components can be seen in Figure 19.

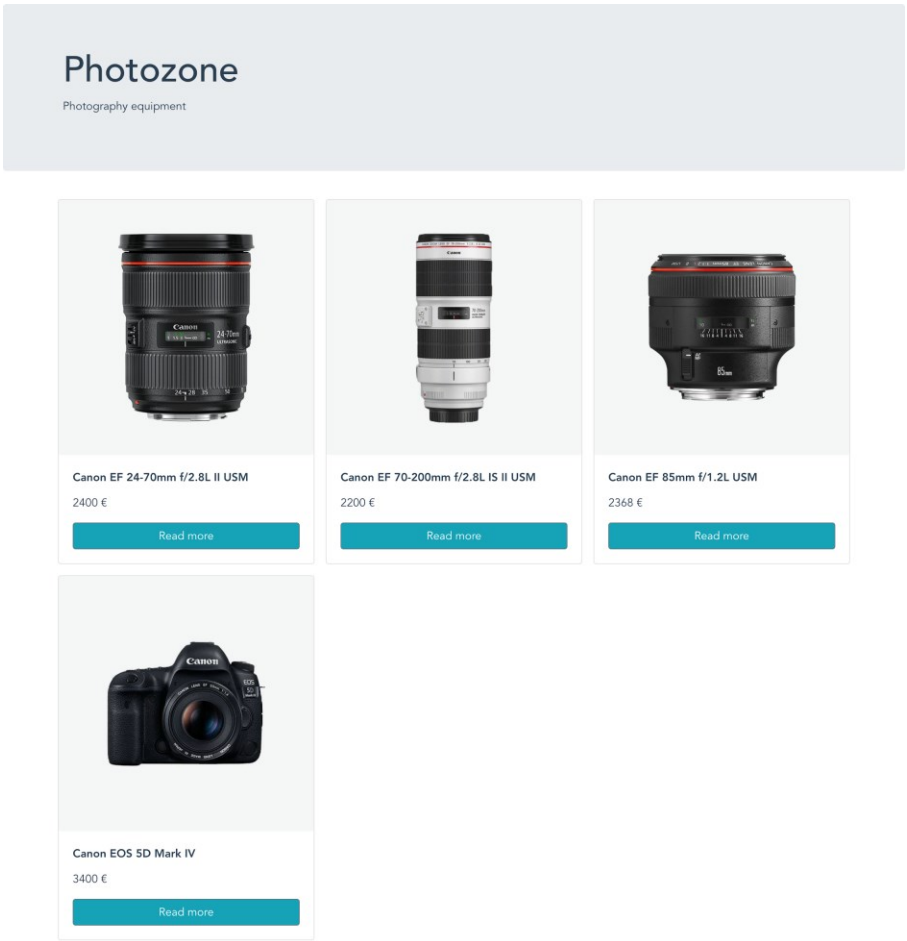

Figure 19. Final look of Vue application.

3.5 Framework comparison

The enormous benefit of the frameworks is the ability to absorb all successful technologies from other frameworks, and at the same time remain clean and logical. We compare the frameworks in the following chapter to reveal the similarities and differences [45] also considering three simple applications created and discussed above.

3.5.1 Vue.js vs React

Both the framework and the library are creating Virtual DOM and synchronizing it with the real DOM as soon as the data in it changes (described in chapter 2.3.1). Moreover, both of them provide a reactive view structure, high efficiency and routing.

In React the components implement their UI using JSX (described in chapter 2.3.2) by render functions. In addition, Vue.js has the same possibilities but goes with a classic way of using templates by default. Templates are HTML based on very intuitive domain-specific language (DSL) directives and are considered to be more intuitive for developers. They also simplify the migration of the applications for utilizing the Vue.js framework.

Another difference is that CSS styles in Vue are located in the same file as the component itself whereas in React styles are located in a separate file. However, React's ecosystem is bigger and there are more modules comparing to Vue.js, while the last one is easier to use and maintain reducing the time to develop and support the project, as well as descriptive documentation with plentiful examples.

3.5.2 Vue.js vs AngularJS

As it is stated in the comparison documentation of Vue.js [45]: "there were a lot of things that AngularJS got right and these were an inspiration for Vue". In the case of simplicity, Vue.js is much more straightforward compared to AngularJS. The Angular learning curve is steeper, requires time and effort to learn and understand the concept of the framework API. In a sense of year on the market, AngularJS is a stable long time existing framework with tested solutions to offer, while Vue.js is relatively young.

AngularJS provides a relatively strict structure to follow, while Vue.js is more flexible in that case and provides more modular solutions. In Vue.js there is no two-way data binding (one-way) and no dirty-checking, which increases the runtime performance of Vue.js.

3.5.3 Vue.js vs Angular

Also known as Angular2+, Angular is based on TypeScript (chapter 2.2.1). Vue.js is not restricted to that but has this possibility. As declared in the official Vue.js blog [46], Vue.js 3.0 will present improved TypeScript support with TSX (JSX under TypeScript). Both frameworks are efficient, but the size of Vue.js bundle is much smaller than Angular's. Regarding flexibility and modular structure, Vue.js has the same differences with Angular as with AngularJS.

4 SITECORE

Sitecore is a Danish IT company that introduces a few products to the market including Sitecore Experience Platform (CMS), Sitecore Experience Commerce (shopping experiences delivery), Sitecore Content Hub (assets management), Sitecore Experience Manager (content management and scaling), and different modules for e-commerce [47].

In the current thesis, we concentrate on the Sitecore Experience Platform (XP). This product effectively combines content and commerce solutions. Sitecore CMS is based on ASP.NET platform and is driven by Internet Information Services (IIS, extensible web server). This grants developers the creation of practically all kinds of applications using XHTML, HTML5, CSS, jQuery, JSON, XML, and other technologies.

A global software engineering services company Epam presents itself as a platinum partner of Sitecore for over 10 years and states the following in their press release from February 5, 2019 [48]: ”The Sitecore Experience Platform™ combines web content management, omnichannel digital delivery, insights into customer activity and engagement, and strategic digital marketing tools into a single, unified platform”. Sitecore offers a full complex package for running e-marketing business perfectly fitting big enterprises with mature marketing target.

Among Sitecore marketing modules such as media analytics and content modules, there are separate optimization and survey modules. These options help in better understanding of potential customers and are able to notice changes faster, therefore, to adjust the content for them more accurately. Personalization of content is achieved by, for instance, tracking of the page from where the user has been directed.

Sitecore CMS is a paid platform for a website and portal development backing integration possibilities such as an intranet and ERP-systems, deeply customizable and very extensible. The user interface is convenient for nontechnical users such as editors and marketers because the CMS development focus is on customer intelligence. As one of the benefits, Sitecore supports content management from mobile devices.

One of the difficult things about Sitecore is the learning curve. Sitecore recommends that new developers take developer training classes so that the base is explained in the

curriculum. In this training, beginners master architecture and the connection of code to it. The training includes short lectures and manual work on the architecture in the content tree, as well as practical coding.

From the point of view of the content editor, Sitecore elements in most cases are simply dragged, but in order to actually turn this content into a web page, the developer needs to implement layouts, sublevels, etc.

Sitecore and MVC are complementary technologies especially when web applications with a structured stream and multiple data presentations take place. In these cases, Sitecore creates an excellent level of data for the data model.

In this thesis, we are going to use Sitecore 9.1, which was announced in November 2018. In the latest release, Sitecore developers concentrated not only on platform and infrastructure improvements, but also on important aspects such as Sitecore Identity, Sitecore Accelerator Framework (SxA), and JSS. Therefore, Sitecore has a new separate service for authorization, which becomes the default Single Sign-On (SSO) system. SxA provides an improved speed up of website production and component reuse. This version empowers front-end development as far as the Sitecore Experience Platform is used as a headless CMS.

Headless CMS represents a decoupled architecture system [49] separating the Content Presentation (rendered view) from Content Management (CRUD, create-read-update-delete) and Content Delivery (a way to store the data). With the latest version, Sitecore CMS presents JSS providing a division of front-end and back-end development, which did not have a clear separation before. Sitecore JSS supports JS frameworks such as Angular, React, and Vue.js. Headless CMS is API-driven and releases front-end developers from dealing with server-side programming languages, enabling to take proper care of front-end look and feel, as well as functionality [47].

5 JSS

JSS is Sitecore JavaScript Services and as stated on the official website [50]: “JSS is a complete SDK for JavaScript developers that enables you to build full-fledged solutions using Sitecore and modern JavaScript UI libraries and frameworks.” Therefore JSS supports the development of modern web applications based on Angular, React or Vue front-end frameworks (and libraries) extended by Sitecore.

With JSS tool kit developers can create web applications without Sitecore instance installed on the local machine and in a disconnected from Sitecore CMS mode. The data from Sitecore is received as JSON files via API calls.

5.1 JSS application modes

There are four different methods of JSS application development process that should be chosen before any JSS project installation [50]:

- Disconnected dev (code-first) mode allows front-end developers are disconnected from Sitecore and Sitecore instance on the local machine, front-end structure can be quickly implemented and the necessary Sitecore artefacts created. The data is retrieved from the local mock data file. After a client-side part of the application has been coded, the project can be deployed and connected to the Sitecore;

- Connected dev mode (Sitecore-first) retrieves the content data directly from the Sitecore databases and is hosted locally. The application development is similar to non-JSS project coding where the Sitecore information architecture created. The Sitecore artefacts such as templates, items, renderings are running by the Sitecore instance. For the Sitecore-first approach the Sitecore JSS-enabled license is required;

- The integrated mode also requires the Sitecore JSS license and is hosted on the Sitecore instance. Also renders an application by Node.js on a server-side (server-side rendered, SSR);

- API-only mode provides the content data from the headless Sitecore API. Choosing this option restricts the benefits of the routes and components. This mode does not require JSS npm packages installed, but obligates for the Sitecore license.

5.2 JSS installation

The steps for the JSS installation are described in Appendix 5. As a result, after successful installation, the browser opens a new window with the default JSS page

containing the links to the JSS tasks documentation, the style guide for JSS usage, and ways of interaction with GraphQL as a default API (see Figure 20).

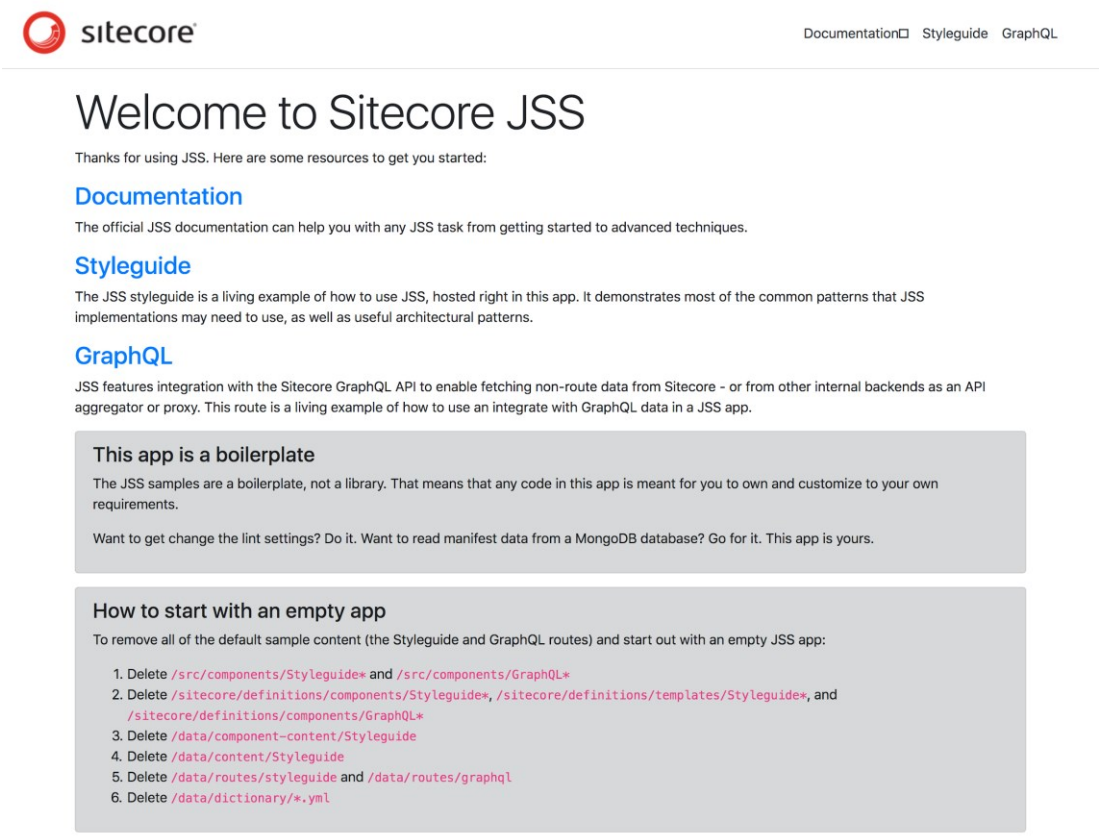

Figure 20. Start page after JSS project installation.

JSS CLI gives an option of JSS server installation via JSS setup command. This command deploys the application to Sitecore or connects to the Sitecore instance in order to request the data.

Before proceeding the development of the application and adding new components to it the project should be emptied by removal of the default sample content, which is style guide and GraphQL files.

6 GRAPHQL

JSS applications are able to use GraphQL. As stated on the official website of GraphQL: “GraphQL is a query language for your API, and a server-side runtime for executing queries by using a type system you define for your data” [51]. One of the

key features is that the structure and the amount of data are determined by the client application. It means that if the user requests for only three fields of data, GraphQL produces a JSON result containing these three fields of data, even though there can be more fields that the data consists of. This approach reduces the amount of data at the transport level. The query example:

```
{
   product {
      title,
      type,
      price
   }
}
```

Returns:

```
{
  "data": {
    "product": {
      "title": "Canon EF 70-200mm f/2.8L IS II USM",
      "type": "lense",
      "price": 2200
    }
  }
}
```

GraphQL was developed by Facebook in the year 2015 as a more successful alternative to REST API (developed in 2000), which is an architectural style based on representational state transfer [52]. REST API are the general principles for organizing the interaction of a web application or a website with a server using the HTTP protocol and using its own HTTP request method for each type of operation: GET, POST, PUT, and DELETE.

GraphQL API and REST API are both capable of sending requests via URL and returning the data in JSON format [53]. Compared to GraphQL, REST can return the data in different formats (JSON, XML). While REST is based on and depends on the HTTP protocol, GraphQL is independent of the data transfer protocol, being able to use any, and can retrieve data from different databases in one request.

7 SOFTWARE METRICS

In the fast-growing software market, the requirements for software product development speed and its quality are significantly increasing. Though project

development based on frameworks can structure the code, increase its quality, and avoid typical mistakes, formal quality criteria still remain relevant. Here is where tracking and analyzing of software metrics measuring the quality and reliability of the project are applied.

A software metric is a measure of characteristics that can be computed and evaluated based on certain conditions. The use of software metrics helps in evaluating the developed project or software process complexity assessment [54], estimation of the scope of work, the style of the program being developed and the effort spent by each developer on implementation of a particular solution. Metrics allow making weighed and objective decisions on improvement.

Software measurement has become a key aspect of good software engineering practice (Farooq and Quadri, 2011) [55]. Software metrics are important components of project management, but occasionally are used as recommendations, as far as the project indicators can non-linearly be dependent on the scope of work. For instance, if one of the work criteria would be the number of code lines, some developers could increase lines without a need, and, vice versa, if the requirement is less line of code, it can affect the performance. Based on my IT practice, I conclude that the measurement also depends on the complexity of the project, because sometimes the search of the error can take hours or days, and the fix would be one line.

At present, my experience as of software developer shows that many software metrics are used in the IT world, though some part of them is never applied in practice, either because of the impossibility of further use of the results, the hopelessness of automatization or because of too narrow specialization.

The main objective of software metrics is to improve the software value, increase the return on investment (ROI, a measure of the gain comparing to the cost) [56] and reduce the costs and overtime. The proper methodologies and metrics chosen for the project improve the quality of the product, its development and customer service more precisely.

There are different categories of software metrics: software sizing metrics, performance metrics, software complexity metrics, etc. For each project, software metrics are chosen based on specific requirements. If during the process of tracking of software metrics there is no progress, they should be changed. In this thesis, we concentrate on software metrics such as cycle time, source lines of code (SLOC) and active days.

7.1 Cycle time

Cycle time is a software metric that measures the time period from the beginning of the work on the product feature until its development completion. Thus, tracking this metric provides the time elapsed between the start and the end of the development process.

Cycle time metric is a part of Agile metrics. Agile Project Management is a methodology for software project management and planning [57]. Based on that approach, the software product development happens through small sprint iterations (usually two weeks). This means that the product is delivered feature by feature (tasks) positioned on the tasks board by priority. The agile approach simplifies the work on the project and allows learning how to manage it, thereby increasing the efficiency of the team.

Sometimes the cycle time metric is confused with lead time. The main difference between these metrics is that lead time tracks the time from scheduling of the idea, creating a task to the customer delivery, while cycle time represents the actual timing for task (product or a service) implementation. The lead time value includes cycle time value (see Figure 21).

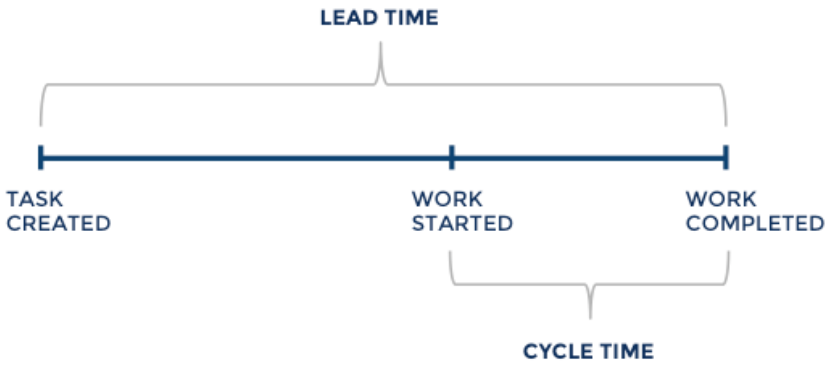

Figure 21. Cycle time and lead time metrics [58].

7.2 Lines of code

One of the most notable quantitative size-oriented measurements is Lines Of Code (LOC) or Source Lines Of Code (SLOC). As they are declared in kilo lines of code, KLOC abbreviation is also used. This simple and widely used software metric originally was developed for a project effort estimation, as well as for error and bugs resolution. However, due to the fact that the same functionality can be split into several lines or written into one line instead of multiple the metric has become practically inapplicable. For instance, with the advent of arrow functions which are the compact version of standard functions used in ES6. This approach decreases the number of code lines significantly.

ES5:

```
var regularFunction = function () {
     console.log('Regular function';)
};
```

ES6:

```
const arrowFunction = () => {console.log('Arrow function') };
```

LOC software metric is differentiated into 2 types: physical and logical lines of code. The physical LOC is defined as the total number of lines of the project source code, including comments and blank lines. The logical LOC defined as the number of software logical commands or statements, and the abbreviation for this metric is lLOC [59]. In logical LOC, a line is accepted if it is an operator of the programming language, though lines such as a comment or an empty line are not considered. Based on that numeric value the metric complexity can be built.

Both physical and logical LOC are used to assess development productivity after the software was finished returning some numeric value of a certain software property. In my opinion, it is wrong to reduce the assessment of a person's work to several numerical parameters and judge performance by it. In addition to SLOC, quantitative characteristics can also include other metrics such as a number of blank lines or comments, the percentage of comments in the project source code, average number of lines for functions, etc.

7.3 Active days

Active days are considered to be a part of production metrics, that measure the productivity of the development team. Active days are the days that a developer spent

on the project code contribution excluding non-engineering tasks such as the time spent on planning and meeting hours. This metric helps to calculate the so-called hidden costs of coding interruptions. Compared to cycle time, active days are measured in days or minutes representing the day-to-day activity and include only development actions performed by the coder.

8 USE CASE

In this chapter, we will create three JSS applications based on three different JS frameworks: Angular, React and Vue.js. We will work in disconnected from Sitecore mode. The purpose of this is to create the basic components, define them for Sitecore, and compare the final project between each other.

8.1 ANGULAR

To create a new JSS project based on Angular framework the following command is typed in the terminal:

```
jss create photozone-angular-jss angular.
```

After the successful installation of all npm packages and application start, we can begin to work on our first component Jumbotron:

```
jss scaffold Jumbotron
```

In Figure 22, the above command will produce several files in different folders. According to the JSS CLI guide, the developer needs to specify the component's data inside the jumbotron.sitecore.ts file. All other files are the same as standard Angular component consists of.

```
Component Jumbotron is scaffolding.
Next steps:
* Define the component's data in /sitecore/definitions/components/Jumbotron.sitecore.ts
* Implement the Angular component in /src/app/components/Jumbotron
* Add the component to a route layout (/data/routes) and test it with jss start

CREATE sitecore/definitions/components/jumbotron.sitecore.ts (494 bytes)
CREATE src/app/components/jumbotron/jumbotron.component.css (0 bytes)
CREATE src/app/components/jumbotron/jumbotron.component.html (96 bytes)
CREATE src/app/components/jumbotron/jumbotron.component.spec.ts (778 bytes)
CREATE src/app/components/jumbotron/jumbotron.component.ts (556 bytes)
```

Figure 22. Component files in JSS application.

In the jumbotron.sitecore.ts we specify the dynamically changed fields of the component, disconnected component definition, for jumbotron we have two fields: heading and lead:

```
fields: [
      { name: 'heading', type:
CommonFieldTypes.SingleLineText },
      { name: 'lead', type:
CommonFieldTypes.SingleLineText },
]
```

In the jumbotron.component.html we add a template for jumbotron adding directive *scRichText inside the header so that this data is dynamically retrieved, the same structure is done the jumbotron lead text (See Appendix 6.1 for all files for Jumbotron component):

```
<h1 *scRichText="rendering.fields.heading"></h1>
```

After the component content is finalized, it must be added to the routes located in /data/routes/en.yml file, route definition for the home route. This component will be defined under the main placeholder titled jss-main. Here we specify the fields data, for instance, for jumbotron it will be the heading and the lead text:

```
- componentName: Jumbotron
    fields:
      heading: Photozone,
       lead: Photography equipment
```

The content layout, which is the main file that renders the first index page, is located in src/app/routing/layout/layout.component.html. In this file, we can update the structure of the main page, also as stated in JSS documentation, it is in charge of UI handling 404 and 500 errors and updating of the route-level state (metadata). Figure 23 displays the look of the application after the Jumbotron component has been added:

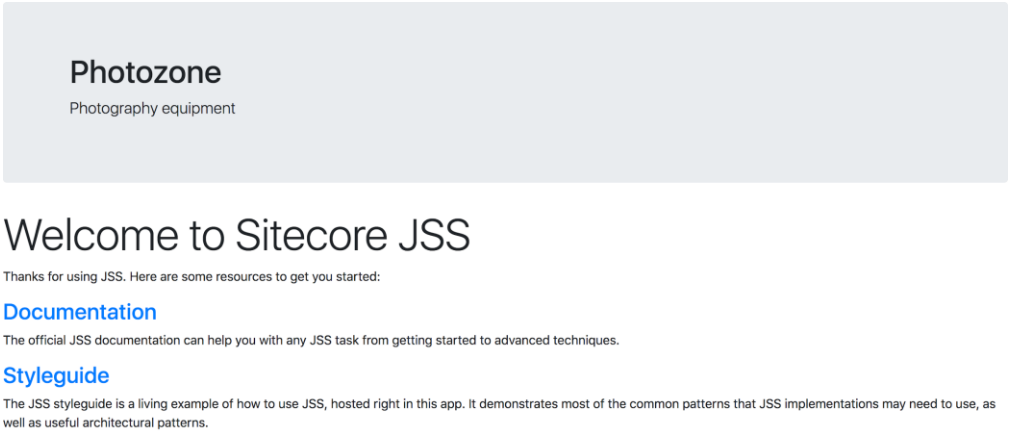

Figure 23. Jumbotron component rendered.

The next component called Products will render the list of product items. The component is scaffolded by the same way as the first component but extended within the list of shared products and an additional template (See Appendix 6.2 for all files and changes of the Products component).

We start with updating of the component's .html file, that should include a standard Angular *ngFor directive that enables going through the items in the products list. In the Sitecore directive file products.sitecore.ts the field type is specified as a ContentList and includes the source where the items are located globally. The list of items is located in the data/content/Products folder with the following structure as presented in Figure 24. Each item is separated by folder and determined via .yml configuration file. In this file, all fields of the product item are specified. The list requires the definition template for Sitecore, where fields for product items are defined by the type. The Image is characterized as Image type, and the price is a Number type. All media are placed in data/media folder.

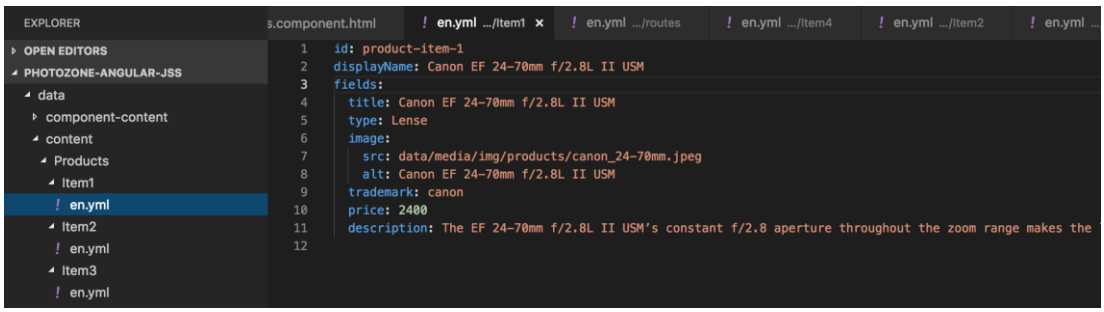

Figure 24. The product item structure.

The last step is to provide the routing by specifying the component inside the main .yml file:

```
- componentName: Products
    fields:
```

```
products:
- id: product-item-1
- id: product-item-2
```

The final result gives the same rendering as a regular application based on Angular JS framework. In the console of Chrome Developers tools, there are logs indicating what components are included in the project (See Figure 25).

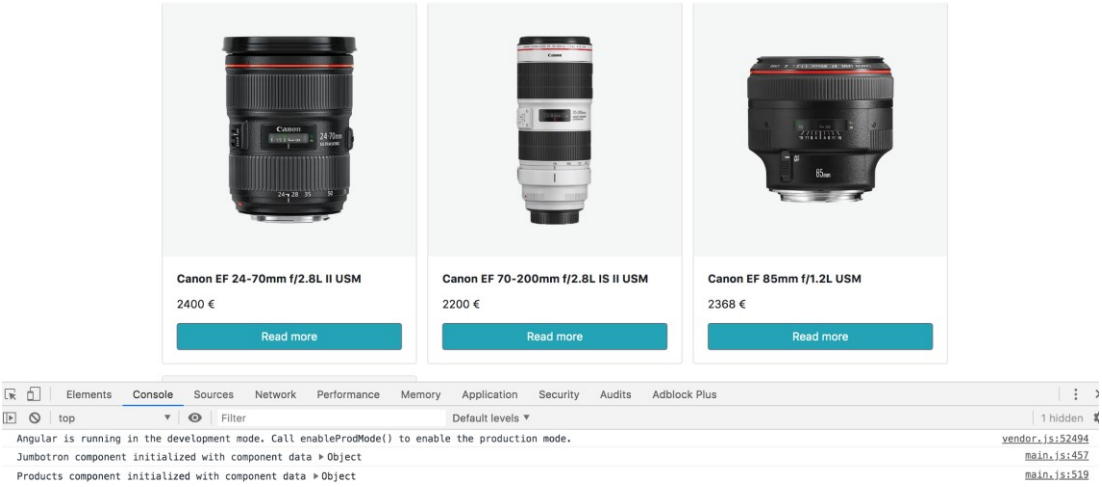

Figure 25. JSS application based on Angular.

8.2 REACT

To create the new JSS project based on React library the following command is typed in the terminal: jss create photozone-react-jss react. Scaffolding of a new component results in Figure 26 indicating that the component can be implemented in a single Jumbotron/index.js file. The Layout.js file is responsible for rendering of global components of JSS application and root placeholder.

```
Component Jumbotron has been scaffolded.
Next steps:
* Define the component's data in sitecore/definitions/components/Jumbotron.sitecore.js
* Implement the React component in src/components/Jumbotron/index.js
* Add the component to a route layout (/data/routes) and test it with jss start
```

Figure 26. New React JSS component is created.

In React JSS application for component development, sitecore-jss-react module is imported to support the use of React components and to integrate it for JSS. For the Jumbotron component we import Text from @sitecore-jss and use it for the header rendering in the following way:

```
<Text tag="h1" field={fields.heading} />
```

Similar to Angular JSS application the component fields must be defined for Sitecore in Jumbotron.Sitecore.js. The difference is that in Angular application the definition file extension of Typescript, while React uses JS files. After the fields and their types are defined, the en.yml file of the application should be updated to include the component into the main placeholder (See Appendix 7.1 for Jumbotron component files).

The products component consists of the same as in Angular JSS application structured product item files and data/routes/en.yml component routing, as well as products template. The main difference with Angular based JSS application is in the component structure implementation. In React the component content is exported as JS constant (See Appendix 7.2 Products component). The product items are accessed via map function, requiring the unique key for each item inside of an array. The final result of the application home page can be seen in Figure 26.

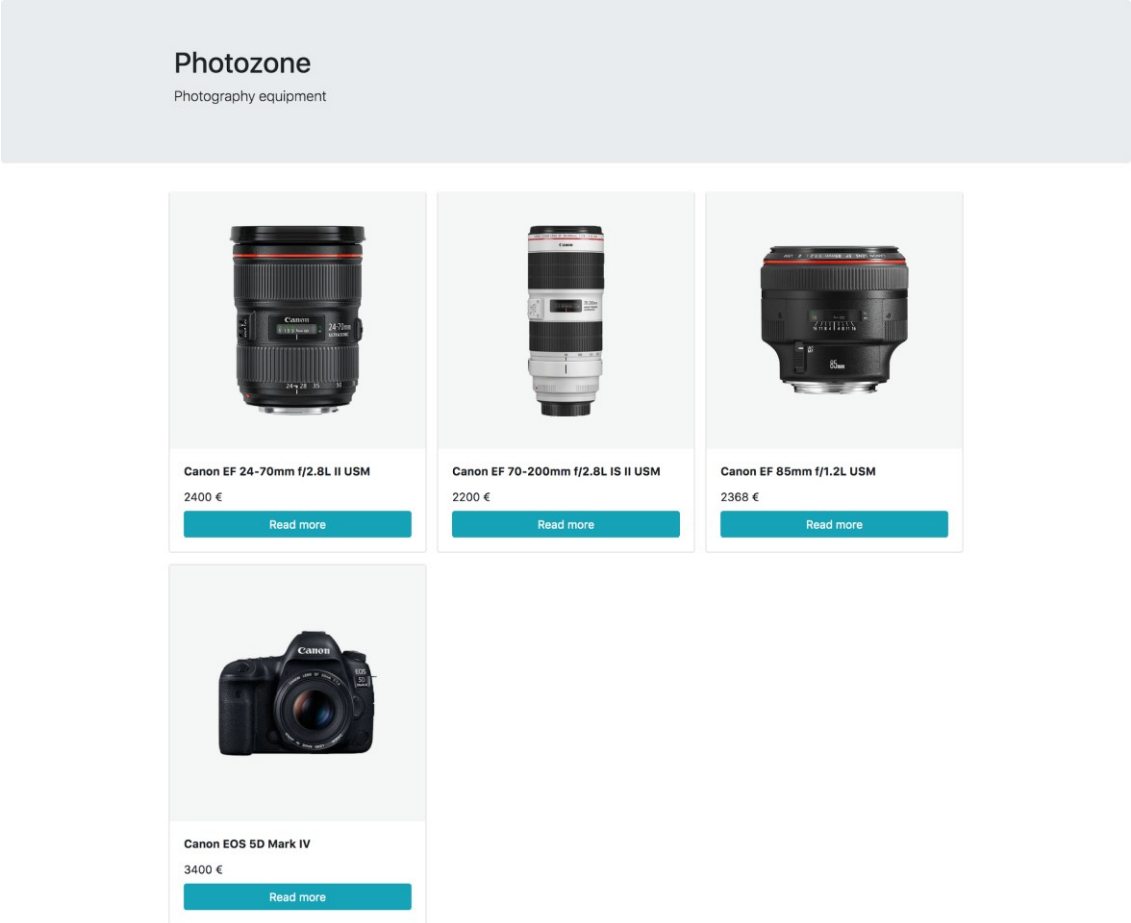

Figure 26. JSS application based on React.

8.3 VUE.JS

To create the new JSS project based on Vue framework the following command is typed in the terminal: jss create photozone-vue-jss vue. The Layout.vue is a shell for home page component rendering where all global components can be defined (such as navigation), as well as the main placeholder. For this project, the navigation and sample component are removed. Creation of a new component results in a new .vue file (see Figure 27), where the component itself should be implemented.

```
Component Jumbotron has been scaffolded.
Next steps:
* Define the component's data in sitecore/definitions/components/Jumbotron.sitecore.js
* Implement the Vue component in src/components/Jumbotron.vue
* Add the component to a route layout (/data/routes) and test it with 'jss start'
```

Figure 27. New Vue JSS component is created.

Compared to Angular and React file structure, Vue components are not located in separate folders containing the structure file, while in Vue JS framework components are created as separate files inside the main components folder. This simplifies the total file structure inside the project.

The Sitecore definition and routing (en.yml file) for Jumbotron component is exactly the same as for JSS applications based on Angular and React. The main difference comes in component implementation in .vue file. As in a standard Vue.js based web application, this file includes the template, the script and the styles. Therefore, what is defined here is HTML structure and export of the name, components (such as ScText: Text), and properties as fields (See Appendix 8.1 for the file structure). Here is an example of how to bind the heading of the Jumbotron:

```
<sc-text tag="h1" :field="fields.heading" />
```

The second products component has the same product items structure, component routing, Sitecore definition and products template as both previous JSS web applications. The Products.vue file, where the component is defined, contains the v-if directive for checking of the product list existence inside the template tag (See Appendix 8.2 for Products component).

The final look and feel of the web application are similar to the previous JSS projects. The Vue.js devtools allows tracking of the components included in the placeholder

providing the properties such as name, rendering and metadata. In Figure 28 Vue.js devtools displays the tree of the application structure. The two anonymous components inside the Jumbotron are simply two fields stored as properties with the value of text displayed.

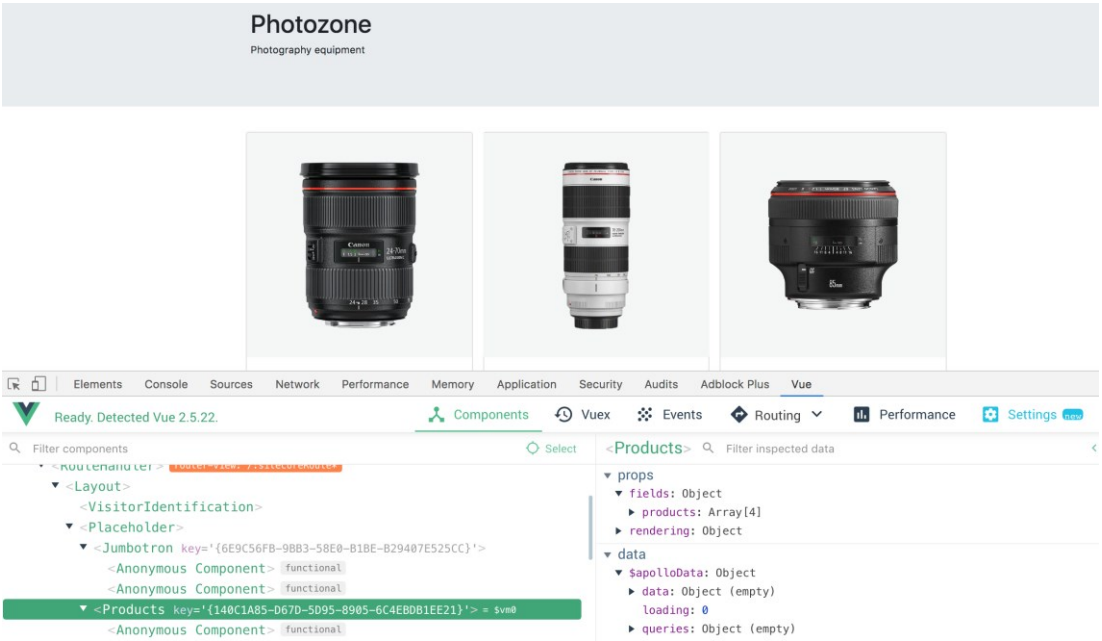

Figure 28. JSS application based on Vue.js with Vue.js devtools in action.

8 COMPARISON

In the following chapter, we will compare three implemented JSS applications based on Angular, React and Vue.js frameworks and libraries between each other based on software metrics that are chosen and described in the previous chapters: cycle time, lines of code, and active days.

Overall, all JSS projects work fast and reactive. The source sizes of projects are 364MB, 246MB, and 136MB (Angular, React and Vue.js respectively), though the amount of implemented components is the same. When a new component is scaffolded, Angular creates four files for component implementation, while React and Vue.js are using just one file for declaration of the component. Both React and Vue.js use virtual DOM compared to Angular, which speeds up the page loading and rendering. Therefore, the volume of traffic is reduced; this is especially important for mobile or tablet device users.

Cycle time software metric measures the time spent on implementation of the task from start to finish. As far as all of the frameworks are new to me as for developer, the familiarity can be set at the same level for all JS frameworks and libraries used. As Jumbotron component can be considered as a simple basic one, including only two fields of data, we compare products component. This component includes not only a Sitecore definition but also product item configuration, products template and setup for media files. The initial scaffold of the component includes the creation of the files where the component structure is implemented. In this aspect, Angular requires more time to scaffold a new component because it consists of four files, while React and Vue.js produce only one file. Therefore, the final size of products component can be seen in Table 1.

Table 1. The size and time spent on products component

JSS project	**Size in bytes**	**Time spent in minutes**
Angular	2094	35
React	1074	40
Vue.js	1027	30

The reason why the React implementation was slightly longer is that it requires time to understand the JSX basic structure and use that in component implementation. For the statistic conclusions, these data cannot be considered as meaningful, though helps to understand what general time effort is required for implementation based on that particular component structure and functionality.

Lines of code software metric, as was described before in this thesis, becomes one of the simplest quantitative metrics. In order to count lines of code in files in the root directory of the project, we run wc -l * command. This result is in the following Table 2, which mostly includes the lines from package-lock.json file. This file is a snapshot of the entire dependency tree, including all downloaded npm packages and their installed versions.

Table 2. LOC count of the root directory

JSS project	**LOC**	**LOC (package-lock.json)**
Angular	13405	12498
React	20840	18140

Vue.js	13024	12613

Using a simple tool to count lines of code called sloc [60], we run a simple command for each project sloc /src and formulate the following Table 3 based on output. Numbers show that the most compact project by both physical and source LOC is based on React, though it has more comments and empty lines than Vue.js based application.

Table 3. LOC count of the total amount of files in all directories

JSS project	**Physical LOC**	**Source LOC**	**Comment LOC**	**Empty lines LOC**
Angular	1759	1241	367	163
React	783	462	222	116
Vue.js	1034	819	126	100

Active days software metrics in this particular case are less than one working day for each application (See Table 4). Thus, this value can be considered as less meaningful for a particular project rather than cycle time and LOC information. During the Angular project development, code contribution was higher compared to React and Vue.js. Active days are an important measure of the productivity of the coder during the certain story or the overall project, while cycle time is the quantitative measure of the specific task or story.

Table 4. Cycle time of each JSS project

JSS project	Active days
Angular	0.3
React	0.25
Vue.js	0.2

The implemented JSS projects represent an identical UI mostly because Bootstrap 4 was used as UI framework, which remains the classes and styles to be the same independently of the JS frameworks used.

9 CONCLUSION

One of the fundamental things in the development of modern web applications is the need for high-quality and competent code creation. Among the variety of modern front-end technologies, JS frameworks and libraries are strongly staying on the top positions. The reason for that is the reusability of the code achieved by application components and the project structure, which simplifies the development process.

Sitecore version 9 has provided an excellent tool for front-end developers and clarified the decoupling of the client- and server-side development. The possibility to deploy and publish the project to Sitecore after the client-side is completed supplies more functionality and intelligibility to the project.

Comparing the top three JS frameworks between each other reveals the benefits and drawbacks of each, which contributes to a more accurate choice of framework for a certain project. Requirements for an application and additional conditions are a valuable part of the final solution developed. This thesis described a simple application development using JSS as a meaningful toolkit for Sitecore CMS.

Starting a new project the developers' team is faced with a choice of which JS framework to choose: Angular, React or Vue.js. There are significant differences between them, and we have described these in this thesis. The best way to find an answer to what front-end framework or library to use is to test them independently. As the results show, all these three SDKs have a place in application development, therefore, the main question is the performance.

During the comparison of JSS projects, including subjective personal experience and software metrics data, I have concluded that React library is preferable for this kind of application because it provides a clear straightforward code architecture and the actual performance is high. The community support is strong and multilingual, as well as has a gradual learning curve. Even though it is a library and requires integrations with

other libraries and tools for project development, it is a more lightweight approach, though Angular and Vue.js are comparable in terms of performance.

For the current project requirements, the preference is given to the React JavaScript library, though the Vue.js framework has enormous prospects due to the ease of learning, minimal basic configuration and effortlessly extendable project architecture. React benefits from a longer-term presence on the IT market. In case the project to be developed is complex and functionally rich, the Angular framework should be considered as the main contender. It provides flexibility, speed of rendering and heavy documentation support.

As a conclusion, this thesis proves that front-end JavaScript frameworks are simplifying and structuring the code architecture. We can see that in the code structure in any project we have created, regardless of JS framework used, there is a certain file and code architecture. Therefore, rich web application functionality and reusability of the code is improved. These features are crucial for present-day products and services presented on the Internet.

REFERENCES

[1] Open Source Vs Commercial Content Management Systems. https://www.cygnet-infotech.com/blog/open-source-vs-commercial-content-management-systems. Read 06.04.2019.

[2] About HeadlessCMS. https://headlesscms.org/about. Read 09.04.2019.

[3] Front-end Frameworks - Overview. https://2018.stateofjs.com/front-end-frameworks/overview/. Read 06.04.2019.

[4] Should Your WordPress Site be an App? https://modeeffect.com/key-differences-between-website-web-app/. Read 09.04.2019.

[5] Gillian Nowlan. Web vs. Native Applications: Best Practices and Considerations in the Development and Design of Web Applications. https://ourspace.uregina.ca/bitstream/handle/10294/6760/Web%20vs%20Native%20Applications.pdf?sequence=1. Read 09.04.2019.

[6] Keith J., Sambells J. (2010) A Brief History of JavaScript. In: DOM Scripting. Apress, Berkeley, CA. Online ISBN 978-1-4302-3390-9.

[7] Single-page application vs. multiple-page application. https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. Read 22.04.2019.

[8] Matthias Kunnen. SEO for Single Page Applications. https://medium.com/forcit/seo-for-single-page-applications-8543619e1d0c. Read 28.04.2019.

[9] Typescript Official Website. https://www.typescriptlang.org/. Read 24.03.2019.

[10] Virtual DOM and Internals. https://reactjs.org/docs/faq-internals.html. Read 30.03.2019.

[11] Node.js Official Website. https://nodejs.org/. Read 23.03.2019.

[12] Introducing JSX. https://reactjs.org/docs/introducing-jsx.html. Read 23.03.2019.

[13] BabelJS next-generation compiler. https://babeljs.io. Read 20.11.2018.

[14] Trygve Reenskaug. Models-Views-Controllers. http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf. Read 23.03.2019.

[15] Apache Cordova / Overview. https://cordova.apache.org/docs/en/latest/guide/overview/index.html. Read 01.05.2019.

[16] Gamma, Erich, Helm, Richard, Johnson, Ralph ja Vlissides, John: Design Patterns: Elements of Reusable Object-oriented Software. Addison75 Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995, ISBN 0-201-63361-2.

[17] Ivan Bodyagin. Model-View-Controller in .Net. RSDN Magazine, No. 2, 2006, Version 1.1.

[18] MVC: Model, View, Controller. https://www.codecademy.com/articles/mvc. Read 09.10.2018.

[19] Top 5 Trends in Front-End Development framework for 2018. https://www.websoptimization.com/blog/front-end-development-trends/. Read 09.10.2018.

[20] Top JavaScript Frameworks and Topics to Learn in 2019. https://medium.com/javascript-scene/top-javascript-frameworks-and-topics-to-learn-in-2019-b4142f38df20. Read 23.03.2019.

[21] Google Trends / compare JS libraries: Vue.js, React, Angular. 2018. https://trends.google.com/trends/explore?cat=31&date=2018-01-01%202018-12-31&q=Vue.js,React,Angular. Read 23.03.2019.

[22] Angular Official Website. https://angular.io/docs. Read 23.03.2019.

[23] Angular Versioning and Releases. https://angular.io/guide/releases. Read 23.03.2019.

[24] Version 6 of Angular Now Available. https://blog.angular.io/version-6-of-angular-now-available-cc56b0efa7a4. Read 23.03.2019.

[25] MVVM architectural pattern with AngularJS. https://meritsolutions.com/mvvm-architectural-pattern-angularjs/. Read 01.05.2019.

[26] Github stars. https://help.github.com/en/articles/about-stars. Read 27.03.2019.

[27] Angular Documentation. https://angular.io/guide/. Read 24.03.2019.

[28] Ngx-bootstrap. https://valor-software.com/ngx-bootstrap/#/documentation. Read 25.03.2019.

[29] Ngx-bootstrap. Getting started with angular-cli. https://github.com/valor-software/ngx-bootstrap/blob/development/docs/getting-started/ng-cli.md. Read 25.03.2019.

[30] Using Bootstrap with Angular. https://codeburst.io/getting-started-with-angular-7-and-bootstrap-4-styling-6011b206080. Read 25.03.2019.

[31] Samer Buna. React.js Succinctly. 2016. http://pepa.holla.cz/wp-content/uploads/2016/08/Reactjs_Succinctly.pdf. Read 20.11.2018.

[32] The Good and the Bad of ReactJS and React Native. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/. Read 24.03.2019.

[33] Chrome DevTools. https://developers.google.com/web/tools/chrome-devtools/. Read 30.03.2019.

[34] React Developer Tools extension offered by the Facebook team. https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi. Read 30.03.2019.

[35] Official Vuejs website. https://vuejs.org/. Read 30.03.2019.

[36] Evan You. Vue 2.6 released. https://medium.com/the-vue-point/vue-2-6-released-66aa6c8e785e. Read 30.03.2019.

[37] 13 Top Companies That Have Trusted Vue.js – Examples of Applications. https://www.netguru.com/blog/13-top-companies-that-have-trusted-vue.js-examples-of-applications. Read 30.03.2019.

[38] Vue CLI / Overview. https://cli.vuejs.org/guide/. Read 01.05.2019.

[39] Webpack concepts. https://webpack.js.org/concepts. Read 01.05.2019.

[40] What is Vuex? https://vuex.vuejs.org/. Read 02.05.2019.

[41] Flux. https://facebook.github.io/flux/. Read 02.05.2019.

[42] ESlint Official Website. https://eslint.org/. Read 03.05.2019.

[43] Git Official Website. https://git-scm.com/. Read 03.05.2019.

[44] BootstrapVue / Getting Started. https://bootstrap-vue.js.org/docs. Read 03.05.2019.

[45] Vue.js. Comparison with Other Frameworks. https://vuejs.org/v2/guide/comparison.html. Read 31.03.2019.

[46] Gregg Pollack. Evan You Previews Vue.js 3.0. https://medium.com/vue-mastery/evan-you-previews-vue-js-3-0-ab063dec3547. Read 31.03.2019.

[47] Sitecore Official Website / Products. https://www.sitecore.com/products. Read 01.05.2019.

[48] Twelve EPAM Experts Named Sitecore Most Valuable Professional. https://www.epam.com/about/newsroom/press-releases/2019/twelve-epam-experts-named-sitecore-most-valuable-professional. Read 04.05.2019.

[49] Ayoush Kohli. Getting Started with Sitecore JSS. http://thinking.edynamic.net/getting-started-with-sitecore-jss. Read 04.05.2019.

[50] Sitecore JSS. https://jss.sitecore.com/. Read 04.05.2019.

[51] Introduction to GraphQL. https://graphql.org/learn/. Read 05.05.2019.

[52] Rest API. https://restfulapi.net/. Read 06.05.2019.

[53] Sashko Stubailo. GraphQL vs. REST. https://blog.apollographql.com/graphql-vs-rest-5d425123e34b. Read 06.05.2019.

[54] Lee, Ming-Chang. (2013). Software measurement and software metrics in software quality. International Journal of software engineering and its application. 7. 15-33.

[55] S. U. Farooq and A. M. K. Quadri, “Software measurements and metrics: Role in effective software testing”, International Journal of Engineering Science and Technology, vol. 3, no. 1, (2011), pp. 671-680.

[56] What Are Software Metrics and How Can You Track Them? https://stackify.com/track-software-metrics/. Read 04.05.2019.

[57] Dybå, Tore & Dingsøyr, Torgeir & Moe, Nils. (2014). Agile Project Management. 10.1007/978-3-642-55035-5_11.

[58] Sami Linnanvuo. Agile & Lean Metrics: Cycle Time. https://screenful.com/blog/software-development-metrics-cycle-time. Read 05.05.2019.

[59] Lines of code metrics (LOC). https://www.aivosto.com/project/help/pm-loc.html. Read 05.05.2019.

[60] Npm package sloc. https://www.npmjs.com/package/sloc. Read 19.05.2019.

APPENDIX

Appendix 1. Main commands of the Node.js

- npm init initializes and fills up the content of package.json;
- npm install installs the dependencies in the local node_modules folder;
- npm install <package name> downloads and installs the certain package in the project, locating it in node_modules folder;
- npm start starts the development server;
- node -v displays the current version installed.

Appendix 2. Angular application

Appendix 2.1 Angular and project setup

- npm install -g @angular/cli
- ng new photozone
  - Choose ‘yes’ on adding Angular router;
  - Choose ‘SCSS’ for stylesheet format;
- cd photozone
- ng serve --open

Appendix 2.2 Bootstrap installation

First, install ngx-bootstrap and bootstrap with Angular CLI [14]:

- npm install ngx-bootstrap bootstrap --save

In the folder src/app/app.module.ts we import:

- import { AlertModule } from 'ngx-bootstrap'
- AlertModule.forRoot()

The app.module.ts looks like this:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AlertModule } from 'ngx-bootstrap';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
      declarations: [
        AppComponent
```

```
      ],
      imports: [
        BrowserModule,
        AppRoutingModule,
        AlertModule.forRoot(),
      ],
      providers: [],
      bootstrap: [AppComponent]
})
export class AppModule { }
```

In the file .angular.json we import the styles:

```
"styles": [
"node_modules/bootstrap/dist/css/bootstrap.min.css",
"src/styles.scss"
],
```

In the folder src/app/app.component.ts we insert an alert:

```
<alert type="success">Bootstrap works!</alert>
```

After saving of all the changes, the project needs to be recompiled because .json file was changed.

Appendix 2.3 Products.js

```
import { Product } from './product';

export const PRODUCTS: Product[] =
   [
       {
       id: 0,
       title: 'Canon EF 24-70mm f/2.8L II USM',
       type: 'Lense',
       image: 'assets/images/canon_24-70mm.jpeg',
       trademark: 'canon',
       price: 2400,
     description:'The EF 24-70mm f/2.8L II USM's constant  f/2.8
aperture throughout the zoom range makes the lens a great low
light performer that's able to make the most of any available
ambient light. ',
       },
       {
       id: 1,
       title: 'Canon EF 70-200mm f/2.8L IS II USM',
       type: 'Lense',
       image: 'assets/images/canon_70-200mm.jpeg',
       trademark: 'canon',
       price: 2200,
```

```
    description:'An essential telezoom lens for sports, wildlife or portraits. The EF 70-200mm f/2.8L IS II USM is a workhorse telephoto zoom lens designed for professional use. It has a rugged durable design, a four-stop Image Stabilizer and specialised lens elements.',
       },
       {
       id: 2,
       title: 'Canon EF 85mm f/1.2L USM',
       type: 'Lense',
       image: 'assets/images/canon_85mm.jpeg',
       trademark: 'canon',
       price: 2368,
      description:'A professional short-telephoto lens, precision-made for low-light shooting and those situations where extremely shallow depth of field is required. Perfect for creative portraiture.',
       },
       {
       id: 3,
       title: 'Canon EOS 5D Mark IV',
       type: 'Camera',
       image: 'assets/images/canon_5d-markiv.jpeg',
       trademark: 'canon',
       price: 3400,
      description:'Designed to perform in every situation, the EOS 5D Mark IV is beautifully engineered and a thoroughly accomplished all-rounder.',
       }
   ];
```

Appendix 2.4 Import Products component

```
import { Component, OnInit } from '@angular/core';
import { PRODUCTS } from '../products';

@Component({
 selector: 'app-products',
 templateUrl: './products.component.html',
 styleUrls: ['./products.component.scss']
})
export class ProductsComponent implements OnInit {
 products = PRODUCTS;
 constructor() { }

 ngOnInit() {
 }
}
```

Appendix 2.5 Products Component

```
<div class="row">
 <div class="col-12 col-md-4 p-2" *ngFor="let product of products">
  <div class="h-100 card" id={{product.id}}>
```

```
 <img class="card-img" width="100%" src={{product.image}}
alt={{product.title}}>
 <div class="d-flex justify-content-between flex-column card-
body">
           <div class="font-weight-bold card-
     title">{{product.title}}</div>
       <p class="card-text">{{product.price}} €</p>
       <button class="bg-info btn btn-secondary">Read
more</button>
      </div>
   </div>
 </div>
</div>
```

Appendix 3. React application

Appendix 3.1 React and project setup

- npm install -g create-react-app
- create-react-app <project title>

Appendix 3.2 Bootstrap, reactstrap and react-popper installation

- npm install bootstrap --save
- npm install reactstrap --save
- npm install react-popper --save

Next step is to include the Bootstrap into the project. This is done by importing it into index.js file located in the rootFolder/src/:

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bootstrap/dist/css/bootstrap.min.css';
import './index.css';
```

Appendix 3.3 MainComponent.js

```
import React, { Component } from 'react';
import { Jumbotron } from 'reactstrap';

class Main extends Component {

   constructor(props) {
       super(props);
   }

   render() {
       return (
           <div>
```

```
                <Jumbotron>
                    <h1>Photozone</h1>
                    <p className="lead">Photography
equipment</p>
                </Jumbotron>
            </div>
        )
    }
}

export default Main;
```

Appendix 3.4 Import the main component into the application

```
import React, { Component } from 'react';
import Main from './components/MainComponent';

class App extends Component {
 render() {
   return (
       <div>
         <Main />
       </div>
   );
 }
}

export default App;
```

Appendix 3.5 Product.js

```
export const PRODUCTS =
      [
             {
             id: 0,
             title: 'Canon EF 24-70mm f/2.8L II USM',
             type: 'lense',
             image: 'assets/images/canon_24-70mm.jpeg',
             trademark: 'canon',
             price:'2400',
             description:'The EF 24-70mm f/2.8L II USM’s
constant f/2.8 aperture throughout the zoom range makes the
lens a great low light performer that’s able to make the most
of any available ambient light. ',
             },
             {
             id: 1,
             title: 'Canon EF 70-200mm f/2.8L IS II USM',
             type: 'lense',
             image: 'assets/images/canon_70mm-200.jpeg',
             trademark: 'canon',
             price:'2200',
             description:'An essential telezoom lens for
sports, wildlife or portraits. The EF 70-200mm f/2.8L IS II
```

```
USM is a workhorse telephoto zoom lens designed for
professional use. It has a rugged durable design, a four-stop
Image Stabilizer and specialised lens elements.',
            },
            {
            id: 2,
            title: 'Canon EF 85mm f/1.2L USM',
            type: 'lense',
            image: 'assets/images/canon_85mm.jpeg',
            trademark: 'canon',
            price:'2368',
            description: 'A professional short-telephoto
lens, precision-made for low-light shooting and those
situations where extremely shallow depth of field is required.
Perfect for creative portraiture.',
            }
     ];
```

Appendix 3.5 Final content of MainComponent.js

```
import React, { Component } from 'react';
import { Jumbotron } from 'reactstrap';
import Products from './ProductComponent';
import { PRODUCTS } from '../shared/products';

class Main extends Component {

    constructor(props) {
        super(props);

        this.state = {
            products: PRODUCTS,
        };
    }

    render() {
        return (
            <div>
                <Jumbotron>
                    <div class="container">
                        <h1>Photozone</h1>
                        <p className="lead">Photography
equipment</p>
                    </div>
                </Jumbotron>
                <Products products={this.state.products} />
            </div>
        )
    }
}

export default Main;
```

Appendix 3.6 Final content of ProductComponent.js

```
import React, { Component } from 'react';
import { Card, CardImg, CardBody, CardTitle, CardText, Button
} from 'reactstrap';

class Products extends Component {

      constructor(props) {
        super(props);
      }

      render () {
        const products = this.props.products.map (product => {
            return (
       <div className="col-12 col-md-3 p-2">
          <Card key={product.id} className="h-100">
            <CardImg width="100%" src={product.image}
alt={product.title} />
           <CardBody className="d-flex justify-content-between
flex-column">
              <CardTitle className="font-weight-bold">
{product.title}</CardTitle>
              <CardText>{product.price} €</CardText>
              <Button className="bg-info">Read more</Button>
             </CardBody>
           </Card>
        </div>
                  );
      });
      return (
         <div className="container">
           <div className="row">{products}</div>
         </div>
      )
     }
}
export default Products;
```

Appendix 4. Vue application

Appendix 4.1 Vue and project setup

- npm install -g @vue/cli
- vue --version
- vue create photozone
- npm run serve

Appendix 4.2 Bootstrap installation

- npm i bootstrap-vue

Next step is to include the Bootstrap into the Vue project. This is done by importing it into main.js file:

```
import BootstrapVue from 'bootstrap-vue'
import './styles/main.scss'
Vue.use(BootstrapVue)
```

All scss styles are located in the separate folder named styles inside the src folder. The main scss file imports the bootstrap files and other scss files. Adding style rules will be automatically converted into css styles and the project page in browser will reload on a change.

Appendix 4.3 First component

In components folder new file created titled Jumbotron.vue:

```
<template>
    <b-jumbotron>
      <div class="container">
      <h1>{{title}}</h1>
      <p>{{lead}}</p>
    </div>
    </b-jumbotron>
</template>

<script>
export default {
  name: 'Jumbotron',
  props: {
    title: String,
    lead: String
  }
}
</script>
```

By adding title property to props we can control the title of the jumbotron from the App.vue file (which is the main component constructed based on sub components). Next step is to update that file to include jumbotron component and render it in the view. Thus, App.vue file has this structure:

```
<template>
  <div id="app">
    <Jumbotron title="Photozone" lead=”Photography equipment”/>
  </div>
</template>

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

export default {
  name: 'app',
  components: {
    Jumbotron
  }
```

```
}
</script>

<style lang="scss">
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  color: #2c3e50;
}
</style>
```

Appendix 4.4 Products component

Content of products.js located in src/shared folder:

```
const PRODUCTS =
    [
        {
        id: 0,
        title: 'Canon EF 24-70mm f/2.8L II USM',
        type: 'Lense',
        image: 'assets/images/canon_24-70mm.jpeg',
        trademark: 'canon',
        price:'2400',
        description:'The EF 24-70mm f/2.8L II USM's constant
f/2.8 aperture throughout the zoom range makes the lens a great
low light performer that's able to make the most of any available
ambient light. ',
        },
        {
        id: 1,
        title: 'Canon EF 70-200mm f/2.8L IS II USM',
        type: 'Lense',
        image: 'assets/images/canon_70-200mm.jpeg',
        trademark: 'canon',
        price:'2200',
        description:'An essential telezoom lens for sports,
wildlife or portraits. The EF 70-200mm f/2.8L IS II USM is a
workhorse telephoto zoom lens designed for professional use. It
has a rugged durable design, a four-stop Image Stabilizer and
specialised lens elements.',
        },
        {
        id: 2,
        title: 'Canon EF 85mm f/1.2L USM',
        type: 'Lense',
        image: 'assets/images/canon_85mm.jpeg',
        trademark: 'canon',
        price:'2368',
        description:'A professional short-telephoto lens,
precision-made for low-light shooting and those situations where
extremely shallow depth of field is required. Perfect for
creative portraiture.',
        },
```

```
        {
        id: 3,
        title: 'Canon EOS 5D Mark IV',
        type: 'Camera',
        image: 'assets/images/canon_5d-markiv.jpeg',
        trademark: 'canon',
        price:'3400',
        description:'Designed to perform in every situation,
the EOS 5D Mark IV is beautifully engineered and a thoroughly
accomplished all-rounder.',
        }
    ];

export default PRODUCTS;
```

To display the list of product v-for directive is used. The directive :key is a shortcut for v-bind:key. A text coming from the data source is wrapped into double curly braces. The structure of new Products.vue component:

```
<template>
    <div class="row">
      <div  class="col-12  col-md-4  p-2"  v-for="product  in
products" :key="product.id">
        <div class="h-100 card" :id="product.id">
            <img         class="card-img"         width="100%"
:src="product.image" :alt="product.title">
          <div  class="d-flex  justify-content-between  flex-
column card-body">
            <div         class="font-weight-bold         card-
title">{{product.title}}</div>
            <p class="card-text">{{product.price}} €</p>
            <button  class="bg-info  btn  btn-secondary">Read
more</button>
          </div>
        </div>
      </div>
    </div>
</template>

<script>
import Products from './../shared/products.js'

export default {
  name: 'Products',
  data () {
        return {
        products: Products
        }
    }
}
</script>
```

The final structure of App.vue component:

```
<template>
```

```
  <div id="app">
    <Jumbotron title="Photozone" lead="Photography equipment"/>
    <div class="container">
      <Products />
    </div>
  </div>
</template>

<script>
import Jumbotron from './components/Jumbotron.vue'
import Products from './components/Products.vue'

export default {
  name: 'app',
  components: {
    Jumbotron,
    Products
  }
}
</script>

<style lang="scss">
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  color: #2c3e50;
}
</style>
```

Appendix 5. JSS application installation

As a prerequisite for JSS installation, the latest Node.js version should be installed, as well as the Sitecore version must be at least 9.0. In order to simplify the workflow in JSS, there is a JSS CLI tool which scripts are invoked during the project development. The JSS CLI can be installed and executed globally via running this command in the terminal:

```
npm install -g @sitecore-jss/sitecore-jss-cli
```

The project itself is created by a command containing the project title and the framework of use: Angular, React, or Vue. Then moving to the project folder and starting jss.

- jss create <app-title> <framework>
- cd <app-title>

- jss start

Appendix 6. Angular JSS application

Appendix 6.1 Jumbotron component

Jumbotron.sitecore.ts:

```
import { CommonFieldTypes, SitecoreIcon, Manifest } from
'@sitecore-jss/sitecore-jss-manifest';

export default function(manifest: Manifest) {
  manifest.addComponent({
    name: 'Jumbotron',
    icon: SitecoreIcon.DocumentTag,
    fields: [
      { name: 'heading', type: CommonFieldTypes.SingleLineText
},
      { name: 'lead', type: CommonFieldTypes.SingleLineText },
    ],
  });
}
```

Jumbotron.component.html:

```
<div class="jumbotron">
  <div class="container">
    <h1 *scRichText="rendering.fields.heading"></h1>
    <p class="lead" *scRichText="rendering.fields.lead"></p>
  </div>
</div>
```

data/routes/en.yml (part):

```
placeholders:
  jss-main:
  - componentName: Jumbotron
    fields:
      heading: Photozone
      lead: Photography equipment
  - componentName: ContentBlock
    fields:
      heading: Welcome to Sitecore JSS
```

src/app/routing/layout/layout.component.html:

```
  <ng-container *ngIf="state === LayoutState.Layout">
    <app-visitor-identification></app-visitor-identification>
    <sc-placeholder  name="jss-main" [rendering]="route">></sc-
placeholder>
  </ng-container>

  <app-not-found *ngIf="state === LayoutState.NotFound"
[errorContextData]="errorContextData"></app-not-found>
```

```
  <app-server-error *ngIf="state === LayoutState.Error"></app-
server-error>
```

Appendix 6.2 Products component

Products.sitecore.ts:

```
import { CommonFieldTypes, SitecoreIcon, Manifest } from
'@sitecore-jss/sitecore-jss-manifest';
const packageJson = require('../../../package.json');

export default function(manifest: Manifest) {
  manifest.addComponent({
    name: 'Products',
    icon: SitecoreIcon.DocumentTag,
    fields: [
      { name: 'products',
        type: CommonFieldTypes.ContentList,
        source: `dataSource=/sitecore/content/${(packageJson
as any).config.appName}/Content/Products`
      }
    ],
  });
}
```

Products.component.html:

```
<div class="container">
  <ng-container *ngIf="rendering.fields.products">
    <div class="row">
      <div class="col-12 col-md-4 p-2" *ngFor="let product of
rendering.fields.products">
        <div class="h-100 card">
          <img class="card-img" width="100%"
*scImage="product.fields.image">
          <div class="d-flex justify-content-between flex-
column card-body">
            <div class="font-weight-bold card-title"
*scText="product.fields.title"></div>
            <p class="card-text"><span
*scText="product.fields.price"></span> €</p>
            <button class="bg-info btn btn-secondary">Read
more</button>
          </div>
        </div>
      </div>
    </div>
  </ng-container>
</div>
```

products-template.sitecore.ts:

```
import { CommonFieldTypes, Manifest } from '@sitecore-
jss/sitecore-jss-manifest';

export default function(manifest: Manifest) {
```

```
  manifest.addTemplate({
    name: 'Products-Item-Template',
    fields: [
      { name: 'title', type: CommonFieldTypes.SingleLineText
},
      { name: 'type', type: CommonFieldTypes.SingleLineText },
      { name: 'image', type: CommonFieldTypes.Image },
      { name: 'trademark', type:
CommonFieldTypes.SingleLineText },
      { name: 'price', type: CommonFieldTypes.Number },
      { name: 'description', type: CommonFieldTypes.RichText
},
    ],
  });
}
```

data/routes/en.yml:

```
id: home-page

fields:
  pageTitle: Photozone
placeholders:
  jss-main:
  - componentName: Jumbotron
    fields:
      heading: Photozone
      lead: Photography equipment
  - componentName: Products
    fields:
      products:
      - id: product-item-1
      - id: product-item-2
      - id: product-item-3
      - id: product-item-4
```

Appendix 7. React JSS application

Appendix 7.1 Jumbotron component

Jumbotron.sitecore.js:

```
// eslint-disable-next-line no-unused-vars
import { CommonFieldTypes, SitecoreIcon, Manifest } from
'@sitecore-jss/sitecore-jss-manifest';

export default function(manifest) {
  manifest.addComponent({
    name: 'Jumbotron',
    icon: SitecoreIcon.DocumentTag,
    fields: [
      { name: 'heading', type: CommonFieldTypes.SingleLineText
},
```

```
      { name: 'lead', type: CommonFieldTypes.SingleLineText },
    ],
  });
}
```

Jumbotron.component.js:

```
import React from 'react';
import { Text } from '@sitecore-jss/sitecore-jss-react';

const Jumbotron = ({ fields }) => (
   <div class="jumbotron">
     <div class="container">
         <Text tag="h1" field={fields.heading} />
         <Text tag="p" className="lead" field={fields.lead} />
     </div>
   </div>
);

export default Jumbotron;
```

data/routes/en.yml (part):

```
placeholders:
  jss-main:
  - componentName: Jumbotron
    fields:
      heading: Photozone
      lead: Photography equipment
```

Appendix 7.2 Products component

Products.sitecore.js:

```
import { CommonFieldTypes, SitecoreIcon, Manifest } from
'@sitecore-jss/sitecore-jss-manifest';
import packageJson from '../../../package.json';

export default function(manifest) {
  manifest.addComponent({
    name: 'Products',
    icon: SitecoreIcon.DocumentTag,
    fields: [
      { name: 'products',
        type: CommonFieldTypes.ContentList,
        source: `dataSource=/sitecore/content/${
          packageJson.config.appName
        }/Content/Products`, },
    ],
  });
}
```

products/index.js:

```
import React from 'react';
import { Text, Image } from '@sitecore-jss/sitecore-jss-
react';

const Products = (props) => {
  const { products } = props.fields;

  return (
    <div className="container">
      <div className="row">
        {products && products.map((productItem, index) => (
            <div className="col-12 col-md-4 p-2"
key={`product-item-${index}`}>
                <div className="card h-100">
                    <Image className="card-image w-100"
media={productItem.fields.image}></Image>
                    <div className="card-body d-flex justify-
content-between flex-column">
                        <Text className="card-title font-
weight-bold" tag="div"
field={productItem.fields.title}></Text>
                        <div><Text className="card-text"
field={productItem.fields.price}></Text><span> €</span></div>
                        <div className="btn mt-2 text-white
bg-info">Read more</div>
                    </div>
                </div>
            </div>
        ))}
      </div>
    </div>
  );

};

export default Products;
```

products-template.sitecore.js:

```
import { CommonFieldTypes, Manifest } from '@sitecore-
jss/sitecore-jss-manifest';

export default function(manifest) {
  manifest.addTemplate({
    name: 'Products-Item-Template',
    fields: [
      { name: 'title', type: CommonFieldTypes.SingleLineText
},
      { name: 'type', type: CommonFieldTypes.SingleLineText },
      { name: 'image', type: CommonFieldTypes.Image },
      { name: 'trademark', type:
CommonFieldTypes.SingleLineText },
      { name: 'price', type: CommonFieldTypes.Number },
      { name: 'description', type: CommonFieldTypes.RichText
},
```

```
    ],
  });
}
```

Appendix 8. Vue JSS application

Appendix 8.1 Jumbotron component

Jumbotron.vue:

```
<template>
  <div class="jumbotron">
    <div class="container">
      <sc-text tag="h1" :field="fields.heading" />
      <sc-text tag="p" :field="fields.lead" />
    </div>
  </div>
</template>

<script>
import { Text } from '@sitecore-jss/sitecore-jss-vue';
export default {
  name: 'Jumbotron',
  components: {
    ScText: Text,
  },
  props: {
    fields: {
      type: Object,
      default: () => ({}),
    },
  }
}
</script>
```

Appendix 8.2 Products component

Products.vue:

```
<template v-if="fields.products">
  <div class="container">
    <div class="row">
      <div class="col-12 col-md-4 p-2" v-for="(productItem,
index) in fields.products" :key="`productItem-${index}`" >
        <div class="h-100 card">
          <sc-image class="card-img w-100"
:media="productItem.fields.image" />
          <div class="d-flex justify-content-between flex-
column card-body">
            <sc-text tag="div" class="font-weight-bold card-
title" :field="productItem.fields.title" />
            <div><sc-text class="card-text"
:field="productItem.fields.price" /> € </div>
```

```
            <button class="bg-info btn btn-secondary mt-
2">Read more</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>

<script>
import { Text, Image } from '@sitecore-jss/sitecore-jss-vue';

export default {
  name: 'Products',
  components: {
    ScText: Text,
    ScImage: Image,
  },
  props: {
    fields: {
      type: Object,
    },
    rendering: {
      type: Object,
    },
  },
};
</script>
```