Proprietors Assembly / Hausversammlung

Final Year Project

Alexander Gogl

Supervisor: Chris Casey

Course: BSc (Hons) Computing

University of Central Lancashire

8th March 2019

Contents

Abstract

Chapter 1
Literature Review

1.1 Introduction

As Vue.js was primarily used to develop this project, this chapter aims to illustrate why Vue was created in the first place and how it compares to other frameworks. Finally, it introduces the reader to Search Engine Optimisation (SEO) for Single Page Application (SPA)'s, why it is such a problem to get a high search engine rank and discusses various solutions to this problem.

1.2 Background of Vue.js

Although existing frameworks were already used in order to create Single Page Application, a concept which was discussed as early as 2003 [22] and used as a term for describing webinterfaces with a smooth almost native-like user experience [10], none of these were designed with the purpose of rapidly prototyping UI interfaces. Angular, already widely used by developers and initially created by Google was too big and bloated of a framework to be sensible for small applications. React, a fairly new framework at the time also proved being too complex just like Backbone.js which was used for large-scale enterprise applications. There was something missing: a lightweight framework flexible enough to quickly build prototypes while not being too hard to master. As existing solutions did not seem adequate for this exact purpose [4, p. 10] new frameworks emerged, with Vue being one of them. Its lightweight approach of reactive data-binding and reusable HTML-based components helped fill that niche. Rising in popularity over the years Vue is now utilized for complex enterprise-grade applications and small prototpyes alike and has since been adopted by many developers and companies around the world. The most noteworthy of which are: Facebook, Netflix, Adobe, Xiaomi, Alibaba and GitLab [23]. With more than 130,000 stars on GitHub at the time of this writing, Vue is more popular than both React (122,000 stars) and Angular (45,000 stars).

1.3 Comparison to other Frameworks

As there is an excessive amount of libraries and frameworks available for creating web based applications this section will be limited to comparing Vue to the other most popular frameworks React and Angular. The following aspects of these will be evaluated: basic usage, scaling, performance, build size and ease of learning. As some terms are prone to ambiguity further clarification is necassary: basic usage describes how a framework is typically used when integrated into an environment that allows some kind of build process. For example Webpack is most commonly used as a bundler, providing a build process which main purpose it is to consolidate, transform and transpile code for usage in a browser, allowing developers to use next-gen JavaScript syntax that is not yet widely available in browsers among other things. Scaling discusses the capabilities of adding functionality to a framework. Flexibility describes if a framework enforces specific conventions in other words, how opionionated it is.

1.3.1 Vue 2.6.X

Vue.js is considered a very performant JavaScript framework that "makes it easier for developers to create rich, interactive websites" [8]. Instead of directly updating a sites DOM which is an expensive operation Vue utilizes a virtual DOM [15] - the representation of the actual DOM as a JavaScript object [20]. When updating the UI, changes are made to this object which is a far cheaper operation. To get the real DOM in sync an efficient updating function is called, resulting in reduced ineffeciency especially in case many nodes have to be updated [20].

Basic Usage

By default, any valid HTML can be used to define the basic structure of Vue components, which is also referred to as a "template" in Vue terminology. Component-scoped data and logic are defined within script tags and are complemented by functional directives such as "v-if" inside the template. Style sheets are defined within style tags, separating style and logic. By optionally using the scoped keyword, CSS is bound to a specific component, reducing the risk of polluting global style sheets. The composition of template, script and style tags is called a "Single File Component", indicating that only a single file is needed to create a functional and styled component. Additionally, props provide a uni-directional way of passing data from a parent component to a child component, whereas events are commonly used to pass data in the opposite direction. "Methods" can be defined to execute repeatable code, computed properties "are calculations that will be cached based on their dependencies and will only update when needed" [4] and watchers which "watch" developer-defined properties and run a function everytime the property's value changes.

```
<template>
<div>
<h1 v-if="condition" class="custom-title"> Hello {{ user }} </h1>
<h1 v-else> "Hello Stranger" </h1>
</div>
</template>

</script>
export default {
    data: {
        return() {
            user: "Alex"
        }
    },
    methods: {/*execute code*/},
    computed: {/*return data when data changes*/},
    watch: {/*execute code when data changes*/}
}
</script>

</style scoped>
.custom-title {
    size: 30px;
}
</style>
```

Listing 1.1: Vue Single File Component

Scaling

Vue applications can easily be scaled up or down in terms of application functionality depending on the requirements. In order to create sophisticated and large applications, three parts have to be incorporated in general: The core library, routing and state management, of which all are officially provided by Vue as supporting libraries [15]. This typically goes along with bundling tools such as Webpack or Browserify, which make for a much more powerful development environment. In addition, Vue offers an optional CLI generator interface for scaffolding projects, leaving the choice to the developer which building system and plugins to use. Using this interface, sophisticated applications along with routing and state management can be created. If the goal is to add reactive and interactive elements to an existing webpage, Vue can be used by adding a script tag to any valid site where necessary [13, 15]. In doing so, no bundler is needed for code to function but at the same time developers are deprived from being able to use plugins, preprocessors, and various other tools and are most commonly left with a larger bundle size [15].

Performance & Build Size

The source size of Vue with the additional dependencies of vuex (state management) and vue-router is about 30KB [15]. As for performance, Vue is a very performant and efficient framework in terms of startup time, memory allocation and rendering time when compared to other technologies [21]. Early beta versions of Vue 3.0 even reduce memory allocation and rendering time by about 50%.

Flexibility & Learning Curve

Among other things, Single File Components are easy to use when coming from an HTML background, make transitioning existing applications to Vue smoother, do not have a steep learning curve resulting in faster adoption by beginners and can be further enhanced with various preprocessors [15]. As Vue supports various bundlers and build systems while not enforcing specific ways of usage, you could argue that it is less opinionated than some other technologies.

1.3.2 React 16.8.X

Basic Usage

React describes itself as a "library for building user interfaces" [1, p. 2]. By using the word "library" it is implied that less functionality is shipped as opposed to using a traditional framework [1, p. 2]. Like Vue it also utilizes a virtual DOM [1, p. 81]. In React everything is defined in terms of JavaScript, meaning HTML often coupled with CSS are directly embedded into so called render functions. JavaScript XML (JSX), which is essentially a syntax extension to JavaScript with XML-like features [14] is most commonly used (even though not necessary) to define these render functions and is capable of mixing the full power of a programming language with rendering and UI logic [15]. Styling is most commonly achieved by CSS-in-JS solutions provided by additional libraries, effectively consolidating logic and styling in the same place [6].

```
1  class Welcome extends React.Component {
2      render() {
3          return <h1>Hello, {this.props.name}</h1>;
4      }
5  }
```

Listing 1.2: Usage of JSX Render Function

In addition, JSX render functions provide full leverage of JavaScript, including temporary variables and direct references to these and good tooling support (linting, type checking, autocompletion) [14, 15].

Scaling

Like Vue React can either be used with a bundler or added to a single site by using a script tag. React outsources routing and state management to the community, fragmenting its ecosysytem in the process [15]. More than eleven well-known routing libraries are available for React with similar statistics for state management and styling which gives developers a vast variety of available options but also makes choosing the right library for a projects requirements a much more tedious task. Reacts cli "create-react-app" works in a similar fashion like Vues but is much more limited: instead of letting the user choose a variety of options, it assumes that a single page application is to be created with always the same dependencies. However, an existing project can be migrated to a more customized environment with a command provided by React.

Performance & Build Size

The React source itself has about 4.7KB gzipped which makes sense, given that React advertises itself as a library rather than a framework. However, including React DOM (34KB), routing with react-router (6.9KB) and state management with redux (2.4KB) the total size grows considerably to 48KB. Performancewise, React is also very efficient when compared to other technologies, in fact, very similar to Vue [21]. The reason for this, is mainly due to the usage of a virtual DOM.

Flexibility & Learning Curve

To take advantage of online resources and documentation JSX is almost a requirement. As it is an extension to JavaScript, developers who are familiar with this programming language can easily learn the additional syntax [14]. Without prior knowledge of JavaScript however, the learning curve rises considerably. At the same time, the availability of hundreds, perhaps even thousands of supporting libraries makes React very flexible, given that its core is so slim and additional functionality can easily be added by leveraging these.

1.3.3 Angular 7.2.X

Basic Usage

Even though JavaScript could be used, Angular essentially requires the usage of TypeScript, a typed superset of JavaScript [24]. As opposed to JavaScript, TypeScript comes with static type

checking which naturally makes applications less prone to errors [11] and therefore is often used within very large corporate projects. Like Vue Angular also uses a component-based approach for composing user interfaces, but rather than using a single file, multiple files for HTML, CSS and JavaScript logic are typically created.

```
1  //app.component.html
2  <button (click)="show = !show">{{show ? 'hide' : 'show'}}</button> show = {{
       show}}
3  <br>
4  <div *ngIf="show; else elseBlock">Text to show</div>
5  <ng-template #elseBlock>Alternate text while primary text is hidden</
       ng-template>
6
7  // app.component.ts
8  import { Component } from '@angular/core';
9
10 @Component({
11   selector: 'app-root',
12   templateUrl: './app.component.html',
13   styleUrls: ['./app.component.css']
14 })
15
16 export class SampleComponent {
17   show: boolean = true;
18 }
```

Listing 1.3: Angular Basic Usage Example

Scaling

Designed with the purpose of building large and complex applications Angulars API exposes a lot of functionality which is most commonly only necessary for exactly these types of applications. It includes everything from routing, state management, http calls to complete testing suites, however, Angular can also be added to existing sites using a script tag, only providing core functionality.

Performance & Build Size

Angular applications built with the angular project scaffolding interface angular-cli have around 65KB gzipped, double the space as the other two frameworks. As for performance, Angular is also a performant framework [21] but becomes slow under certain circusmstances: For instance if a project utilizes a lot of watchers and data in the scope changes, all watchers are re-evaluated again [15].

Flexibility & Learning Curve

Angulars very big ecosystem and API provide most needed functionality out of the box but at the same time limit its flexibility. What is more, by prodiving pre-defined ways of interacting with Angular, it inherently becomes opinionated and more difficult to master [15].

1.3.4 Conclusion

All three frameworks solve similar problems but are used in different ways. Vues Single File Components consolidate logic, styles and HTML in the same place, React can and is most often used in a very resemblant way if used with a CSS-in-JS approach, whereas the "Angular way" is to split these parts into separate files.

Applications built with React or Vue can be easily scaled up or down depending on the application-requirements. Large applications typically need the core library, routing and state management. While all of these are officially provided by the Vue core team for Vue applications, React leaves routing and state management to the community [15]. Vue and React both offer an optional CLI generator interface to scaffold projects, however, Vue offers more options, leaving the choice to the developer which building system and plugins to use. React's more limiting approach with create-react-app makes it easier to start a project as it only needs a single dependency but limits the user to a given setup, which can however, be moved to a more customized environment with little effort whereas Angular is completely set up for the development of very large projects and provides most functionality out of the box, scaling up is therefore not a big problem because developers can rely on most of the functionality already existing.

All three frameworks can be scaled down by adding script tags to any site [13, 15] which however removes the powerful building layer.

Vue and React both are similar in terms of runtime performance, are based on a virtual DOM and are used for similar use cases while Angular is a much more heavy-weight framework regarding performance and size.

JSX as well as TypeScript are additional learning steps and can lead to decreased productivity in smaller projects. Being a dynamic language, JavaScript's ability to address quickly changing requirements makes it especially suitable for rapid prototyping [11, p. 72]. By using very HTML-like components Vue is often easier to master than the other two frameworks as most developers have at least basic knowledge of HTML. Nonetheless, Vue makes it possible to use TypeScript if the need arises, paving the way for bigger projects [16]. Angular is not very flexible in the sense that users can pick whatever supporting library they deem best and is said to be an opinionated framework which enforces the "Angular way". Such frameworks are "pragmatic, with a strong sense of direction" [2] often forcing very specific conventions upon its users, effectively restricting what a developer can to with a framework. This also means there is a steeper learning curve but once overcome, can lead to increased productivity [15].

The diagram shown in Figure 1.1 does by no means reflect the properties of the given framework / library 100% accurately as some of them are somewhat subjective (e.g flexibility) but shall rather illustrate the strengths and weaknesses in relation to another when used in a typical manner.

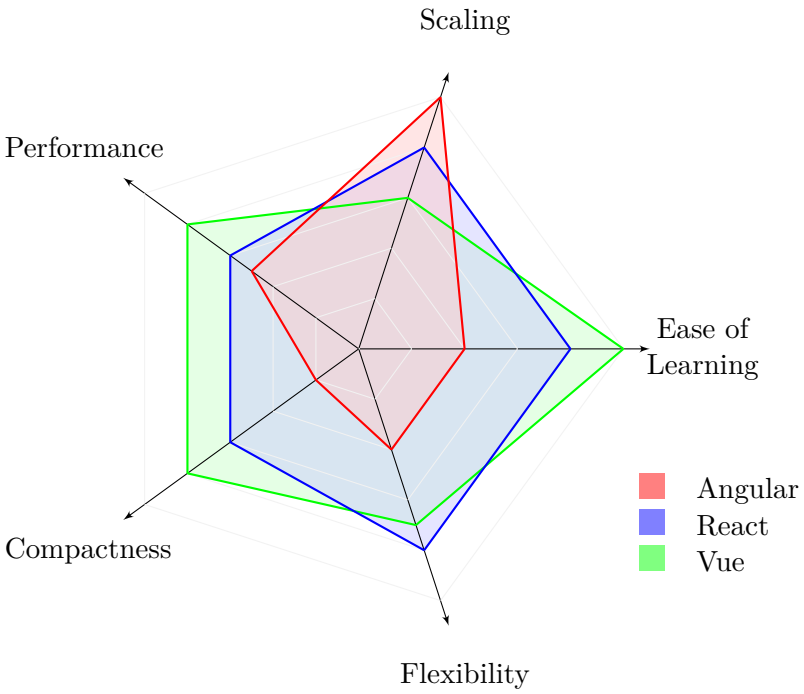


Figure 1.1: Comparison of Vue, React and Angular

1.4 SEO for Single Page Applications

Introduction

There is no doubt that competing companies all over the world want to be found first in the biggest market of all: the internet. As 67.60% of organic clicks are accounted by the first five pages returned by the SERP [7], these same companies are highly incentivized to invest into an improved search engine rank which was found to have a positive impact on an organizations performance [12]. Therefore, the process of Search Engine Optimisation has become ubiquitous in our modern era as it promises to attract more users to a certain website [7]. In other words: more users mean there is a greater probability of conversions, thus generating more revenue.

Unfortunately, successfully implementing SEO is a very delicate matter. In reality it is a set of techniques geared towards software rather than humans. These programs are commonly referred to as crawlers or bots which traverse the Web by exploiting the Web's hyperlinked structure and add their findings to the search engine's index [5, 9]. They start on a websites root page and try to find any available links on that site [5]. The process of choosing which links to follow, itself is dependant on complex algorithms as different crawlers will have to make different decisions [9]. For example a crawler trying to index the web as comprehensively as possible will have different underlying instructions than one that tries to find product reviews [9]. After being added to the index, a site's content is destructured and with additional metrics such as speed, size of images and mobile-friendliness its search engine rank is calculated [5].

Search Engine Optimisation helps crawlers better understand a site's content and "serves as a solution to figure out the nature of each webpage and determine how it can be made worthwhile to the users" [7].

The Problem

In general there are two ways to improve any type of site's rank: on-site optimisation and off-site optimisation [7]. The first pertains to techniques that can be implemented on a site itself, such as efficiency of a webserver, compression of data, deferred loading of images, keywoards or the actual content, whereas the latter is a set of techniques which primarily drive more traffic to a certain site, such as social media marketing or the creation of backlinks [7].

Improving a Single Page Application's ranking score proves difficult when using on-site optimisation techniques. When Google first started crawling websites in 1998 JavaScript was not a big problem as it was not widely used, so it simply was not executed [18]. The rapidly changing landscape of webdesign however, demanded a more sophisticated approach. Things started to change drastically in 2014 when Google's crawler began executing JavaScript [18]. However, as websites increasingly rely on AJAX [3, p. 97], more specifically the deferred loading of resources, crawlers faced with a problem. The research of Brunelle et al. [3, p. 97] illustrates it as following:

> When a crawler fetches a web page (1), it waits for the page and all of the embedded resources to load to consider it fully rendered. At this point, the crawler preserves (2) all of the content on the web page (A). After the page has loaded, the page can request further resources (3), even without user interaction. The resource is then returned to the web page to be rendered (4) producing the final intended result (Ab). Because the crawler preserved the page prior to the page being fully loaded, the secondary content is not preserved and the archived version of the web page is not complete.

This can be seen as one of the biggest disadvantages for SPA's as they heavily rely on AJAX and client-side rendering. If crawlers are not aware of their contents they will in turn suffer from a decreased search engine rank.

Possible Solutions

The problem is to be solved by offering crawlers a fully rendered page instead of having them load additional resources. Depending on a web application's usage several options can be feasible:

The first is to implement prerendering: a process in which static HTML-files are created for specific routes at build time [17]. This technique is especially useful for sites with content that does not change very often. Typical scenarios would be to create static versions of marketing or about sites [17]. One of the main advantages is a rather simple setup process since creating a static version happens only once during building. At the same time, it becomes obvious that this solution is not very suitable for pages with rapidly changing content - rebuilding a project after every data change is not very efficient after all.

A more sensible approach for sites with often changing content is Server Side Rendering (SSR). As opposed to prerendering, SSR is a lot more sophisticated and complex to set up as it requires a tight coupling to the used framework. Upon every request by a client, the framework is responsible for creating a static version of the site which reflects the current available data. In technical terms, a static HTML site is rendered on the server by the framework, sent to the browser and hydrated "into a fully interactive app on the client" [17].

Another possible solution which is still relevant today is to "create a static representation of your web site/application and direct crawlers to use it" and was identifed by Fink et al. in 2014 [5, p. 270]. This technique has been given the name "Dynamic Rendering" by Google in 2018 [19]. It differs from prerendering and SSR in on key aspect: real users and crawlers are served alternative content, hence the term "dynamic". When a crawler requests a site, an intermediate service transforms HTML and JavaScript to a static, prerendered version. This intermediate step allows for the creation of prerendered sites whenever needed - it builds upon prerendering but removes its build time constraint.

Evaluation

Prerendering is fairly easy to implement but is not very suitable for web applications with content that changes rapidly. As prerendering happens at build time, a project would have to be built every time content changes - an automated environment could be set up which builds the project after a given time interval, in this case however, it is noteworthy to mention that building is a rather expensive operation which takes longer the bigger the project is and deployment could result in downtime which accumulates over time. A better solution would be to use dynamic rendering, which builds upon prerendering but is run as a separate service, therefore coming with the benefits but not the disadvantages of normal prerendering. The most sophisticated approach and most complex to implement is SSR which requires intimiate knowledge of a frameworks internals. However, there are solutions that take away some of that overhead for specific frameworks. Examples would be "Next.js" for React and "Nuxt.js" for Vue but in general, the previous statements apply.

Chapter 2
Project Planning

2.1 Introduction

As this project was initially the idea of its project partner who needed an interactive frontend application, planning the project had to be adapted accordingly. This chapter will give an overview about which parts of the whole system are already provided for and identifies the requirements and potential solutions for the frontend. Additionally, it depicts considered technologies and techniques and evaluates the most appropriate development stack.

The main idea is to give tenants, proprietors and facility management a way to communicate over a single point of access. In its first version, they should be able to discuss topics they care about, easily make announcements, submit and track the state of maintenance issues and to create polls in order to provide an easy way for decision-making. The frontend is to be designed as a Minimum viable product (MVP) rather than adhering to exact requirements, meaning deviations are desired if it is sensible (Example found in chapter ...). This MVP will initially be realeased in Austria.

2.2 Provided Services

The Database and API are both provided by the project partner. The first is a PostgreSQL instance, the latter an API built on top of Django and Django Rest Framework. These two technologies reflect the backend of the system as a whole. The API is documented with Postman, describing each endpoint in terms of what type of data has to be sent and what data will be returned. Apart from making these compatible with a containerized environment (see Chapter 6), no further action has to be taken in order to use these services.

2.3 Requirements

The functional and non-functional requirements are primarily defined by the project partner. The following lists describe the functional and non-functional aspects of the frontend on the left and right, respectively.

- Digital Noticeboard
- Polls
- Forum
- Issue Management
- Bilinguality
- Custom Authentication

- High Flexibility
- Easy to maintain
- Low learning curve for future developers
- Easy way of creating apps for mobile phones from frontend

2.3.1 Digital Noticeboard

The digital noticeboard is what it says: a noticeboard accessible online. It is a unidirectional way of communication between house parties. A typical example would be to announce a get-together everyone is invited to.
Allowed user roles: Proprietors, Tenants, Facility Managament

2.3.2 Polls

Polls are intended to remove the friction of decision-making. An example would be a facade that has to be repainted with a new color. To get an initial idea which color it should be, Proprietors can create polls and let other proprietors choose between given options.
Allowed user roles: Proprietors

2.3.3 Forum

The Forum is a way of discussing topics related to a specific house. E.g. a tenant wants to know when garbage cans are emptied. Proprietors as well as facility management and other tenants can answer such questions.
Allowed user roles: Proprietors, Tenants, Facility Managament

2.3.4 Issue Management

With an issue management system tenants or proprietors can submit and track the status of maintenance related issues. The facility management can access these issues and take action. When an issue's state changes they can mark it accordingly in the system.
Allowed user roles: Proprietors, Tenants, Facility Managament

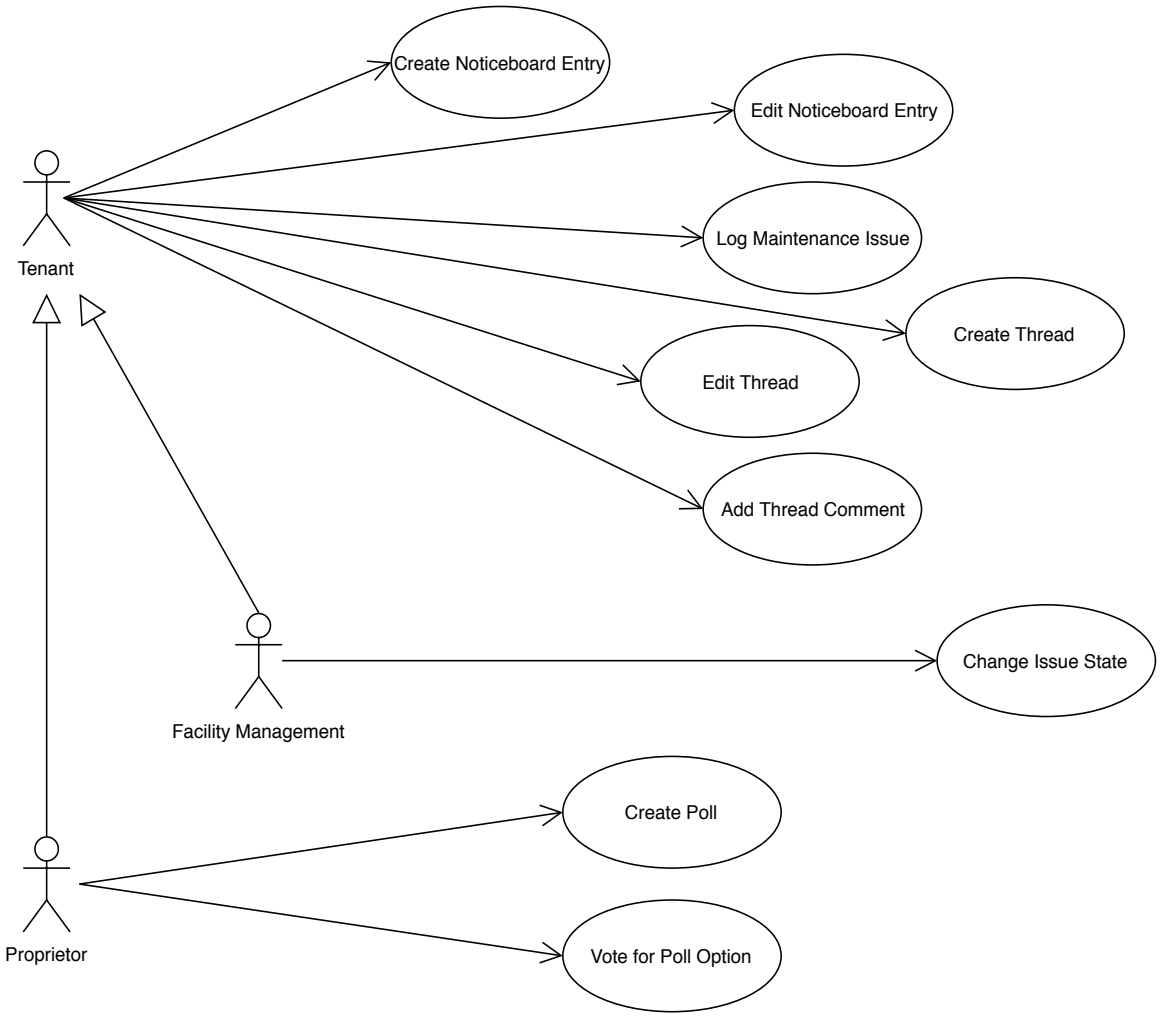
2.3.5 Bilinguality

It is important that users are able to choose between at least two languages. German is the most obvious choice as it is Austria's primarily spoken language. English, the second most well known language in Austria is obvious as well.

Authentication

Users shall be authenticated via a custom endpoint defined in the API. Authentication-providers such as Google or Facebook are not needed.

According to these requirements, a use-case diagram as illustrated in Figure 2.1 can be generated.



* All use cases require/include authentication and authorization which are omitted for brevity

Figure 2.1: Proprietors Assembly Use Cases

2.4 Considered Technologies

2.4.1 SPA Framework

To meet both functional and non-functional requirements SPA's are a potential solution. They provide an interactive, smooth native-like user interface and can easily be transformed to a Progressive Web App (PWA) if the need arises. As illustrated in Figure 1.1, an especially appropriate framework can be found in Vue, which perfectly suits the needs of the non-functional requirements for this project. Its relatively high flexibility in addition to its shallow learning curve allow for a rapid development process.

2.4.2 SSR Framework

Although the project's requirements do not state any need for SSR it is possible that rapidly changing content such as publicicly available blog posts with comments will be added in a future release. Section 1.4 identifies three ways of rendering SPA's on the server side. Server Side Rendering would be the most viable option as it comes with the biggest advantages.

It can be utilized in one of two ways: By using vue as a framework and the supporting library "vue-server-renderer" or by using an additional framework. The first way requires the developer to configure everything themselves whereas the latter provides pre-configured SSR functionality. One of these frameworks is Nuxt.js which is the most popular SSR framework within the vue community. In addition, it configures automatic routing and state management, which are described in more detail in chapter ... .

2.4.3 CSS Framework

There are various CSS frameworks available which differ greatly in design. In addition to the widely used Bootstrap and Foundation frameworks which have a very basic and familiar design, there are smaller ones which provide different design elements which could help distinguish the project from others. Such frameworks are "Bulma" or "Materialize". Bulma comes with a very distinct design, that does not seem as ubiquitous on the web as the others. Furthermore, the vue library "Buefy" builds on top of Bulma's CSS and provides predefined components to use. This is very important as it makes it much easier and quicker to build user interfaces.

2.4.4 Deployment

There are various techniques of deploying a technology stack to a production ready environment ranging from installing a webserver, database and additional services directly onto a server to using virtual machines for each individual service. A much better and more recent approach is the usage of "containers": Each service is encapsulated in its own linux environment but hosted on the same machine. It is important to note that some services are bundled together in the same container if it makes sense: E.g a backend framework which defines REST endpoints and a webserver which exposes these endpoints. A more appropriate word in this case is "microservice". The most popular containerization technology is Docker which gives developers the ability to write "compose" files that define a composition of services. One of the biggest advantages of Docker is that the same services can be run in a development as well as production environment, greatly reducing overhead.

2.4.5 Summary

Vue is the perfect framework for this project due to its flexibility and ease of learning. Nuxt will be used to lay the groundwork for SSR. The design of the frontend will be based on Bulma and Buefy giving the project a distinct and distinguishable look. Docker will provide a way of easily deploying the application on any server and development environment.

2.5 Additional Tools and Techniques

GitLab

GitLab is used not only as a VCS but also as a container registry. The registry exposes an interface for uploading and downloading container images. This is especially useful for creating images on a local machine, uploading them to the registry and downloading them again onto a remote server where they should be deployed. Doing it this way completely removes the need of storing the source code for the whole project on the remote server and building it from source. Instead the version number of the used image is increased in a compose file and the container gets updated accordingly when redeploying.

Git Flow

Git Flow is a VCS methodology which is based around "features". In its most basic structure there is one master and develop branch accompanied by multiple feature branches (usually one feature per branch). The master branch serves as the single source of truth and should always be kept in a production ready state. The develop branch contains the actively developed project-source in which every finished feature branch is merged into. Additionally there are hotfix and release branches which support the others. Git Flow adds an abstraction layer to the development flow and greatly helps building applications by using an iterative approach.

Circle CI

Automated testing is an important aspect when developing applications. When done right, it can reassure the developer that nothing will break and result in unexpected errors. Circle CI is a continuous integration platform which automatically tests code when certain rules are met. It works very well when used in addition to Git Flow and Docker: it can be set up to create a container with the projects source, builds it and run its tests upon every change made to the develop or master branch or pull requests. Circle CI notifies its users if any step during this process failed preventing faulty code to ever reach a production state.

2.6 Legal and Ethical Issues

Chapter 3
System Design

3.1 System Design

3.2 User Interface Design

3.3 Summary

Chapter 4
Implementation

4.1 Nuxt.js

4.2 Basic File Structure

4.3 Separation of Concerns

4.4 API Endpoints

4.5 Authentication / Authorization

4.6 Forum

4.7 Polls

4.8 Noticeboard

4.9 Issue Log System

4.10 State Management

4.11 Internationalization

4.12 Linting

Chapter 5
Test Strategy

5.1 Introduction

5.2 Testing Frontends

Describes which possibilites there are regarding frontend testing: Mainly unit and integration

5.3 Test Setup

Talks about vue-test-utils and jest integration

5.4 Continuous Integration

Gitlab CI / CD

Chapter 6
Deployment with Docker

6.1 Applicability of Containers

6.2 Docker

6.3 Docker Swarm

6.4 Microservices

6.5 Reverse Proxy

Chapter 7
Evaluation

7.1 Project Objectives

7.2 Self Evaluation

7.3 Project Evaluation

7.4 Commercial Applicability

7.5 Conclusions

7.6 Future Work

Acronyms

JSX JavaScript XML. 3, 4

MVP Minimum viable product. 10

PWA Progressive Web App. 12

SEO Search Engine Optimisation. 1

SPA Single Page Application. i, 1, 8, 12, 13

SSR Server Side Rendering. i, 8, 9, 13

Glossary

computed properties "Are calculations that will be cached based on their dependencies and will only update when needed" [4]. 2

DOM Defines the logical structure of HTML-based documents and the way a document is accessed and manipulated. 2

routing Is responsible for coordinating page changes on a website. More specifcally when used as part of a single page application it allows for switching pages without changing the top level Universal Resource Identifier [3]. 3, 4

state management Provides a centralized store for all the components in a single page application. For example makes it possible to store a username in such a way, that any component can retrieve and use it. 3, 4

watchers A watcher tracks a property of the component state and runs a function when that property value changes. 2

References

Literature

[1] Alex Banks and Eve Porcello. *Learning React: Functional Web Development with React and Redux*. " O'Reilly Media, Inc.", 2017 (cit. on p. 3).

[2] Kevin Bedell. 'Opinions on Opinionated Software'. In: *Linux J.* 2006.147 (July 2006), pp. 1–. URL: http://dl.acm.org/citation.cfm?id=1145562.1145563 (cit. on p. 6).

[3] Justin F. Brunelle et al. 'The impact of JavaScript on archivability'. In: *International Journal on Digital Libraries* 17.2 (June 2016), pp. 95–117. URL: https://doi.org/10.1007/s00799-015-0140-8 (cit. on pp. 8, 21).

[4] Olga Filipova. *Learning Vue. js 2*. Packt Publishing Ltd, 2016 (cit. on pp. 1, 2, 21).

[5] Gil Fink and Ido Flatow. 'Search Engine Optimization for SPAs'. In: *Pro Single Page Application Development: Using Backbone.js and ASP.NET*. Berkeley, CA: Apress, 2014, pp. 267–276. URL: https://doi.org/10.1007/978-1-4302-6674-7_12 (cit. on pp. 7, 9).

[6] Naimul Islam Naim. 'ReactJS: An Open Source JavaScript Library for Front-end Development'. In: (2017) (cit. on p. 3).

[7] M. N. A. Khan and A. Mahmood. 'A distinctive approach to obtain higher page rank through search engine optimization'. In: *Sādhanā* 43.3 (Mar. 2018), p. 43. URL: https://doi.org/10.1007/s12046-018-0812-3 (cit. on pp. 7, 8).

[8] Callum Macrae. *Vue. js: Up and Running: Building Accessible and Performant Web Apps*. " O'Reilly Media, Inc.", 2018 (cit. on p. 2).

[9] Filippo Menczer et al. 'Evaluating topic-driven web crawlers'. In: *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2001, pp. 241–249 (cit. on p. 7).

[10] Michael Mikowski and Josh Powell. *Single Page Web Applications: JavaScript end-to-end*. 1st. Generic publisher, 2013 (cit. on p. 1).

[11] Francisco Ortin and Miguel Garcia. 'A Programming Language That Combines the Benefits of Static and Dynamic Typing'. In: *Software and Data Technologies*. Ed. by José Cordeiro, Maria Virvou and Boris Shishkov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 72–87 (cit. on pp. 5, 6).

[12] Zhuofan Yang, Yong Shi and Bo Wang. 'Search engine marketing, financing ability and firm performance in E-commerce'. In: *Procedia Computer Science* 55 (2015), pp. 1106–1112 (cit. on p. 7).

Online sources

[13] React contributors. *Add React to a Website – React*. URL: https://reactjs.org/docs/add-react-to-a-website.html (visited on 18/02/2019) (cit. on pp. 3, 6).

[14] React contributors. *Rendering Elements – React*. URL: https://reactjs.org/docs/rendering-elements.html (visited on 18/02/2019) (cit. on pp. 3, 4).

[15] Vue contributors. *Comparison with Other Frameworks — Vue.js*. URL: https://vuejs.org/v2/guide/comparison.html (visited on 18/02/2019) (cit. on pp. 2–6).

[16] Vue contributors. *TypeScript Support — Vue.js*. URL: https://vuejs.org/v2/guide/typescript.html (visited on 18/02/2019) (cit. on p. 6).

[17] Vue.js contributors. *Vue.js Server-Side Rendering Guide | Vue SSR Guide*. URL: https://ssr.vuejs.org/#why-ssr (visited on 03/03/2019) (cit. on p. 8).

[18] Michael Xu Erik Hendriks. *Official Google Webmaster Central Blog: Understanding web pages better*. May 2014. URL: https://webmasters.googleblog.com/2014/05/understanding-web-pages-better.html (visited on 01/03/2019) (cit. on p. 8).

[19] *Get started with dynamic rendering | Search | Google Developers*. Feb. 2019. URL: https://developers.google.com/search/docs/guides/dynamic-rendering (visited on 03/03/2019) (cit. on p. 9).

[20] Anthony Gore. *What's The Deal With Vue's Virtual DOM? – Vue.js Developers – Medium*. Dec. 2017. URL: https://medium.com/js-dojo/whats-the-deal-with-vue-s-virtual-dom-3ed4fc0dbb20 (visited on 01/03/2019) (cit. on p. 2).

[21] Stefan Krause. *Interactive Results*. URL: https://stefankrause.net/js-frameworks-benchmark8/table.html (visited on 20/02/2019) (cit. on pp. 3–5).

[22] Ian Oeschger Marcio Galli Roger Soares. *Inner-browsing extending the browser navigation paradigm - Archive of obsolete content | MDN*. URL: https://developer.mozilla.org/en-US/docs/Archive/Inner-browsing_extending_the_browser_navigation_paradigm (visited on 28/02/2019) (cit. on p. 1).

[23] Michał Sajnóg. *13 Top Companies That Have Trusted Vue.js – Examples of Applications | Netguru Blog on Vue*. Mar. 2018. URL: https://www.netguru.com/blog/13-top-companies-that-have-trusted-vue.js-examples-of-applications;%20http://gfs.sf.net/gerris.pdf (visited on 18/02/2019) (cit. on p. 1).

[24] *TypeScript - JavaScript that scales*. URL: https://www.typescriptlang.org/ (visited on 19/02/2019) (cit. on p. 4).

List of Figures

Listings

List of Tables