Proprietors Assembly / Hausversammlung

Final Year Project

Alexander Gogl

Supervisor: Chris Casey

Course: BSc (Hons) Computing

University of Central Lancashire

19th 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 [24] and used as a term for describing webinterfaces with a smooth almost native-like user experience [11], 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 [25]. 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" [9]. Instead of directly updating a sites DOM which is an expensive operation Vue utilizes a virtual DOM [17] - the representation of the actual DOM as a JavaScript object [22]. 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 [22].

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.

```
1  <template>
2  <div>
3  <h1 v-if="condition" class="custom-title"> Hello {{ user }} </h1>
4  <h1 v-else> "Hello Stranger" </h1>
5  </div>
6  </template>
7
8  </script>
9  export default {
10     data: {
11         return() {
12             user: "Alex"
13         }
14     },
15     methods: {/*execute code*/},
16     computed: {/*return data when data changes*/},
17     watch: {/*execute code when data changes*/}
18 }
19 </script>
20
21 </style scoped>
22 .custom-title {
23     size: 30px;
24 }
25 </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 [17]. 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 [15, 17]. 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 [17].

Performance & Build Size

The source size of Vue with the additional dependencies of vuex (state management) and vue-router is about 30KB [17]. 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 [23]. 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 [17]. 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 [16] 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 [17]. 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) [16, 17].

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 [17]. 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 [23]. 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 [16]. 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 [27]. As opposed to JavaScript, TypeScript comes with static type

checking which naturally makes applications less prone to errors [12] 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 [23] 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 [17].

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 [17].

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 [17]. 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 [15, 17] 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 [12, 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 [18]. 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 [17].

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.

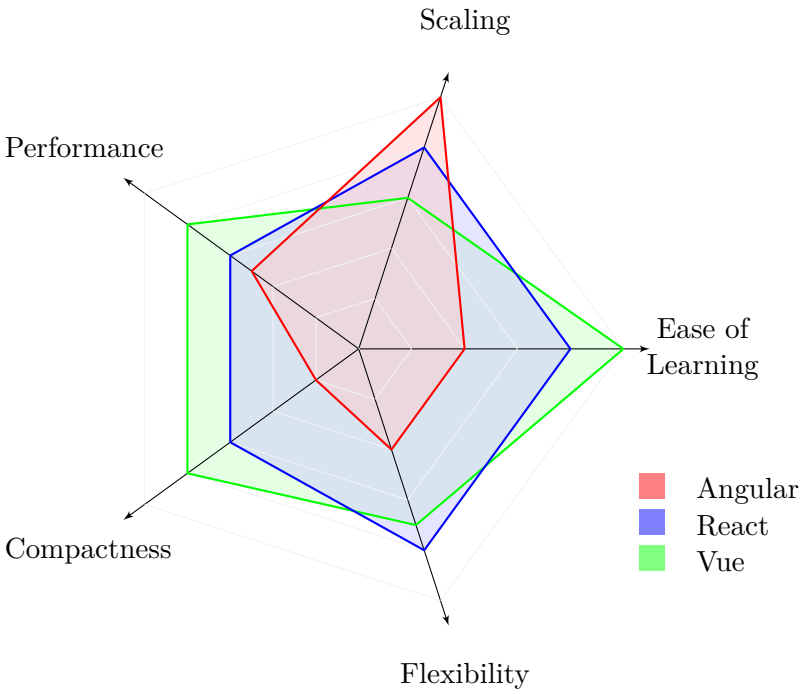

Figure 1.1: Comparison of Vue, React and Angular

1.4 SEO for Single Page Applications

1.4.1 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 [13]. 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, 10]. 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 [10]. 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 [10]. 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].

1.4.2 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 [20]. 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 [20]. 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.

1.4.3 Possible Solutions

The problem is to be solved by offering crawlers a fully rendered page instead of having them load additional resources. This concept is called Server Side Rendering (SSR) in broad terms. Depending on a web application's usage several variatons are feasible:

Prerendering

The first is to implement prerendering: a process in which static HTML-files are created for specific routes at build time [19]. 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 [19]. 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.

Server Side Rendering

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" [19].

Dynamic Rendering

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 [21]. 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

The project was to develop a front-end to an existing system and had to be planned accordingly. This chapter will give an overview about which parts of the system already exist 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 create polls in order to provide an easy way for decision-making. The requirements are not "hard" requirements in a sense that a customer requirements document or something similar exists, rather they are written as informal user stories. It is the developer's responsiblity to develop feasible use cases and flows. It is also noteworthy to mention that this project will initially be realeased in Austria.

2.2 Provided Services

A Database, Exchange Email Server and API are 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 provide 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

As mentioned in the introduction, several functional requirements exist. Furthermore, there are some non-functional requirements for the project which have to be taken into account. Table 2.1 depicts these general requirements.

Functional Requirements	**Non-Functional Requirements**
Digital Noticeboard	High Flexibility
Polls	Easy to maintain
Forum	Easy way of creating apps for mobile phones from frontend
Issue Management	Low learning curve for future developers
Bilinguality	

Table 2.1: Project Requirements

These very high functional requirements can be further broken down into user stories. The most important of which are illustrated along with additional information in the next few sections. For a full list of user stories please see (appendix user stories) The word "user" is used for referring to every user-group of the system. The specific user-groups are: Tenant, Proprietor, Facility Management.

2.3.1 Digital Noticeboard

Description	A unidirectional way of communication between house parties
User Story	As a user I want to create noticeboard entries so other users can see my messages
Example	Announcing a get-together everyone is invited to

2.3.2 Polls

Description	A method for decision-making
User Story	As a proprietor I want to create polls other proprietors can vote on so that we as a group can make decisions
Example	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
Notes	Polls should not be editable so proprietors cannot change their contents after someone voted

2.3.3 Forum

Description	A way of discussing topics related to a specific house
User Story	As a user I want to ask questions so other users can answer them
Example	A tenant wants to know when garbage cans are emptied

2.3.4 Issue Management

Description	A way of submitting and track the status of maintenance related issues
User Story	As a user I want to submit an issue so that it can be taken care of
Example	A user has a water leak and submits an issue, the facility management can access these issues and take action

2.3.5 Bilinguality

Description	Changing the language
User Story	As a user I want to change the language of the application so I can understand the content
Notes	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

According to these requirements, a high-level use-case diagram as illustrated in Figure 2.1 can be generated. Additional information like basic flows of these use cases can be found in the respective section of Chapter 4 which is in accordance with the "Decide as late as possible" principle found in lean development projects.

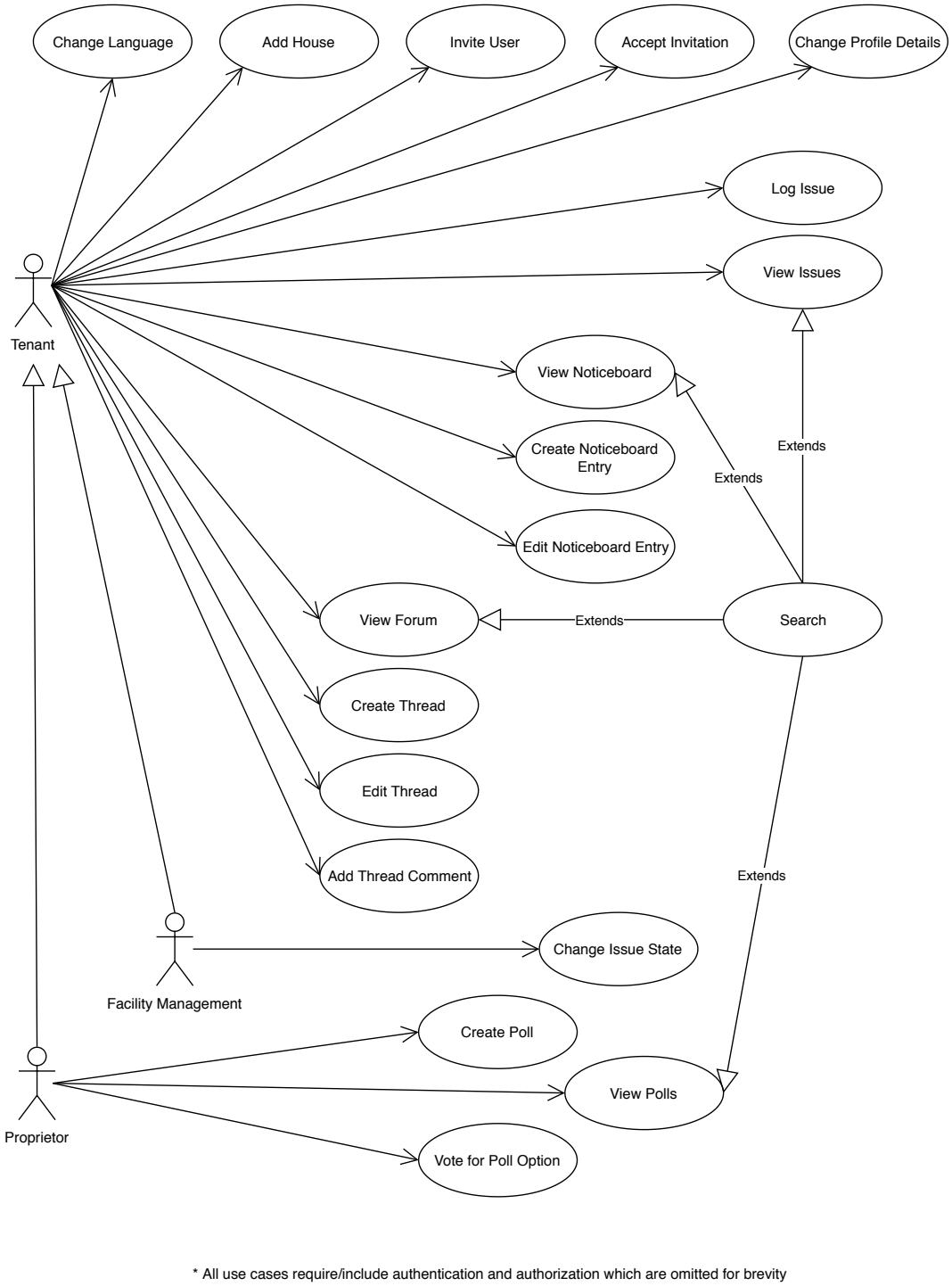

Figure 2.1: Proprietors Assembly Use Cases

2.4 Considered Technologies

2.4.1 SPA Framework

To meet both functional and non-functional requirements a Single Page Application (SPA) is 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. PWA's are a fairly new type of application that use traditional web-browser technology but can be run almost as if they are native applications. As illustrated in Figure 1.1, an especially appropriate framework for developing this project can be found in Vue, which perfectly suits the needs of the non-functional requirements: as opposed to React or Angular, 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 implicate the necessity of implementing any type of Server Side Rendering (SSR), it is possible that rapidly changing content such as publicly available blog posts with comments will be added in a future release. Section 1.4.3 identifies three ways of rendering SPA's on the server side and explains in great detail why this is important. Developing the project from the start while keeping this information in mind could reduce long-term complexity.

Server Side Rendering as defined in Section 1.4.3 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 (Not yet written) .

2.4.3 CSS Framework

CSS frameworks can drastically decrease developing time for frontend projects as they provide predefined tools and components for creating user interfaces. 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 to be used as widely on the web. Furthermore, the vue library "Buefy" builds on top of Bulma's CSS and provides predefined vue 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

(monolith) to using virtual machines for each individual service. A much better and future-proof 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 Webserver

The two main webserver technologies are NGINX and Apache httpd. According to various statistics and benchmarks, NGINX can handle a vastly greater number of concurrent connections and can serve static content more efficiently, whereas httpd is better suited for handling dynamic content (e.g. PHP-generated sites). These two technologies however, are often used in tandem: NGINX used as a reverse proxy handling a great number of connections and httpd used to serve dynamic content. With the finished Single Page Application (SPA) consisting only of static content NGINX clearly is more suitable for this project.

2.4.6 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. An NGINX-based webserver is used to serve the static frontend.

2.5 Additional Tools and Techniques

GitLab

GitLab not only serves as a Git-based Version Control System (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 a redeployment is initiated.

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 should always be kept in a production-ready

state, wheras 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 an iterative approach.

Circle CI

Automated testing is important when developing applications. 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 runs its tests upon every change made to the develop or master branch or a pull request. 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

As identified in the Technical Report (see Appendix ...) there are two main areas of concern regarding ethical and legal issues: privacy and visual impairment. In order to use the service, users have to provide information about them, including name, email and home address which could lead to GDPR related issues. Providing access to the service to visually impaired users also poses an ethical issue as adaptions have to be carefully thought about.

Chapter 3
System Design

3.1 System Design

In the previous chapter the requirements of the project were illustrated. From these an initial high level overview of the system can be generated as depicted in Figure 3.1.

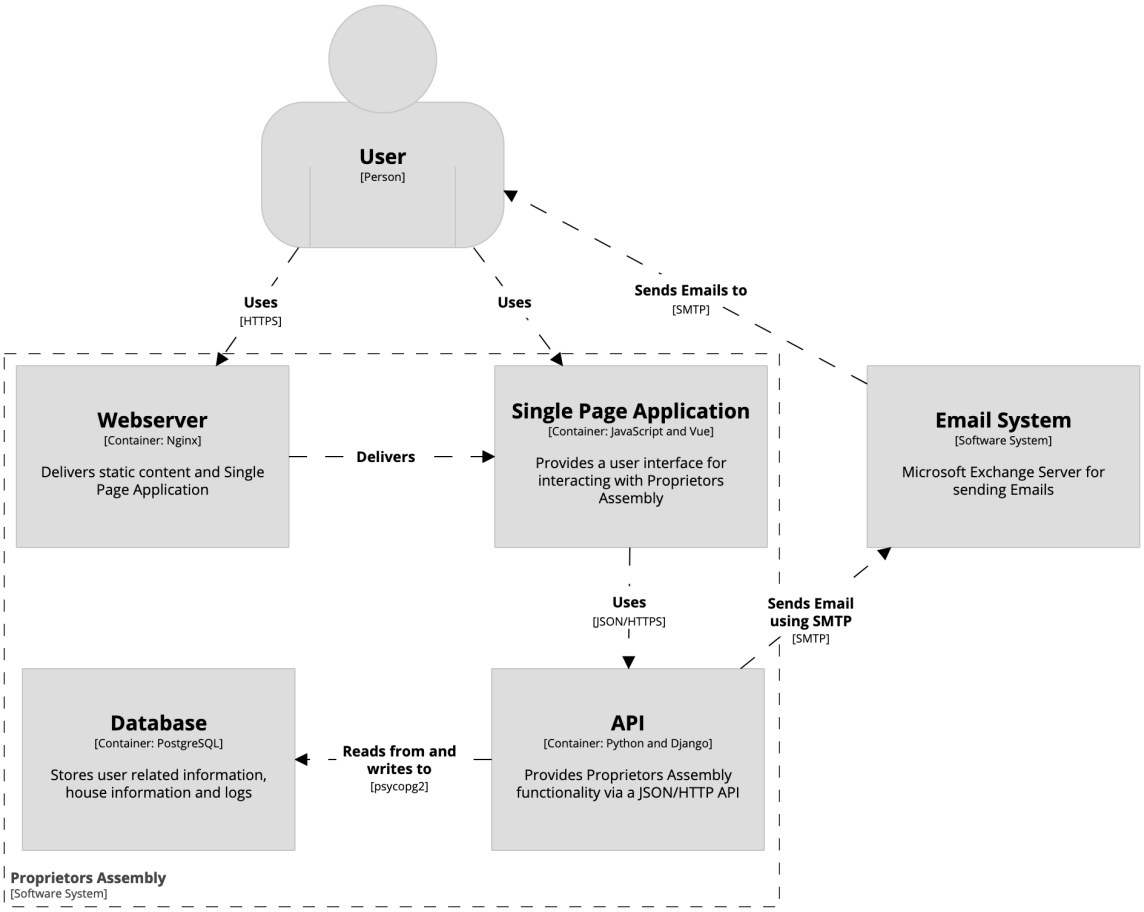

Figure 3.1: System Design

In its basic structure, Proprietors Assembly as a software system consists of an NGINX webserver, a Vue Single Page Application, a Django API and a PostgreSQL database. When a user visits the domain `https://hausversammlung.at`, the webserver will serve the static frontend which communicates with the API. Depending on the use-case, the API will retrieve data from or add to the Database. An additional third-party Exchange server is used to send emails regarding authentication and house details. The diagram intentionally omitts certain parts of the system for brevity. One of these is a reverse-proxy service which is the first point of contact from a user's viewpoint. A reverse proxy in this case routes between containers which are

hosted on the same machine, accessible through unique ports. This topic is discussed in depth in Chapter 6.

3.2 Identifying UI Components & Use Case Flows

Interactive components of the frontend are strongly dependant on the required functionality. This section aims to define what is minimally needed on a certain page and to translate functional requirements into identifiable UI elements. It is intended to be an extension to the requirements of the frontend (see Section 2.3) with regards on how to implement these. This makes it possible to identify elements that are used in multiple places and clearly shows the base structure of the frontend. Furthermore, as Vue uses a component-based approach, these elements serve as a very good starting point for defining which vue-components are required.

Additionally, as mentioned in Section 2.1 it is the developer's responsibility to derive all use cases and flows from the requirements. The approach taken is as following: 1) the thought process of how a site should look like is laid out 2) use case flows are derived 3) the site's elements are be mapped to a flow 4) UI elements are identified. **Note:** as too many use case flows would quickly result in a bloated document, only the most important flow per page/category will be shown, additional flows are depicted in Appendix B.

3.2.1 Login Page

The Login / Register page handles user authentication. A user is required to log into an existing account or create a new one before they can use the system. Traditionally, this is done with a plain HTML-Form. It is sensible to create separate forms for login and register and let the user decide which form to use depending on their needs. Additionally, a forgot-password form should provide a uniform way of requesting a password change.

Identified Elements: HTML forms for login, register and password-reset

3.2.2 Home

On the home site various data sources should composed into a consolidated view - a dashboard. It shows the most recent forum items, polls and noticeboard entries in addition to the current number of issues, house parties and squaremeters. A user should be able to switch between a list of houses they are associated with. A navigation bar should contain links to all subpages: Home, Noticeboard, Polls, Forum, Issue Management and the Profile.

Identified Elements: Noticeboard Item, Poll Item, Forum Item, House Switcher, Navigation Bar

3.2.3 Noticeboard

The Noticeboard page contains a list of all noticeboard entries associated with a house. An input field should let the user search for a specific item. A navigation bar as described for

the previous page should allow for page changes. Finally, a button that links to a page where noticeboard entries can be created should exist. If the user is the owner of an entry an edit button should be shown. It is sensible to consolidate creating and editing an item on the same site by using a single form which is either initially empty or populated with the existing data should be edited. Creating a noticeboard requires a title and a message.

Key use case according to user stories depicted in Table 3.1. Supporting use cases can be found in

The user opens the noticeboard page by using the navigation bar
The user clicks a "create" button
The user fills a from with a title and a message
The system checks for appropriate authorization
The system adds a new entry to the database
The user is redirected to the main noticeboard page

Table 3.1: Use Case Flow: Create Noticeboard

Identified Elements: Noticeboard Item List, Noticeboard Item, Search Bar, Navigation Bar, Create Button, Create / Edit Noticeboard Entry Form

3.2.4 Polls

On the Polls page every poll associated with a house is displayed. Users should be provided with a way of searching and filtering these items. A navigation bar should allow for quick page changes. In order to create a poll, users should be shown a button which takes them to a subpage where they can create new polls or edit existing ones. To create a poll, a title and at least two options have to be provided by the user.

Identified Items: Poll Item List, Poll Item, Search Bar, Navigation Bar, Create Button, Create / Edit Poll Form

3.2.5 Forum

The Forum page shows users a list of all thread items of a house which are searchable. A subpage which is accessible by clicking a button should contain a single form for creating and editing items. A thread needs to have a title, category and content. Additionally, a navigation bar links to the other pages.

Identified Items: Thread Item List, Thread Item, Search Bar, Navigation Bar, Create Button, Create / Edit Thread Form

3.2.6 Issue Management

Similar to the other pages, the issue management page should have a navigation bar, a search bar and a button which links to a subpage where items can be created or edited. Its main purpose is to show a list of all house-related issues. An issue has a reference, date, type, location, title and status.

Identified Items: Issue Item List, Issue Item, Search Bar, Navigation Bar, Create Button, Create / Edit Issue Form

3.2.7 Profile Page

The profile page provides a way of making changes to a user's account. Initially, they should be able to change their name. Additionally, they should be able to switch languages or house and to invite additional users to their currently active house.

Identified Elements: Language Switcher, Invite Form, House Switcher

3.2.8 Add-House Page

Users need a way of creating a new house or adding an existing house to their account by using an invitation link. To create a new house a form should be provided which takes the address, total squaremeters and role of the user as input variables. An invitation should be accepted by providing an invitation id and the intended role of the user. After creating or accepting an invitation users should be given the possibility to invite additional users to their newly added house.

Identified Elements: Create House Form, Accept Invite Form, Invite Form

3.3 Summary

Some elements are very distinct and are only used once whereas others are used across multiple pages. One element that stands out and is used in almost every single page is the "Navigation Bar". It would be very sensible to not only create a reusable component of this element but to also include it in some kind of base-layout that can be used as a starting point when creating sub-pages. Some other widely used elements are the "Search Bar" and "Create Button". Elements which might not be used very often but reused nevertheless are the "Language Switcher" and "House Switcher". The various forms, lists and items mentioned previously are used only once most of the time and therefore not reused, however, it still makes sense to create reusable Vue components for these to make site composition cleaner and easier.

3.4 Site Structure

Another important aspect of frontend-applications is the question of how sites are related to eachother. This concept can best be described as a sitemap. Well designed sitemaps which are often reflected as links in a navigation bar not only give users a clear understanding of the applications structure but can also be turned into machine-readable files which can be accessed by crawlers. From the previous defined pages it is possible to interpolate the complete structure of the frontend-application as depicted in Figure 3.2.

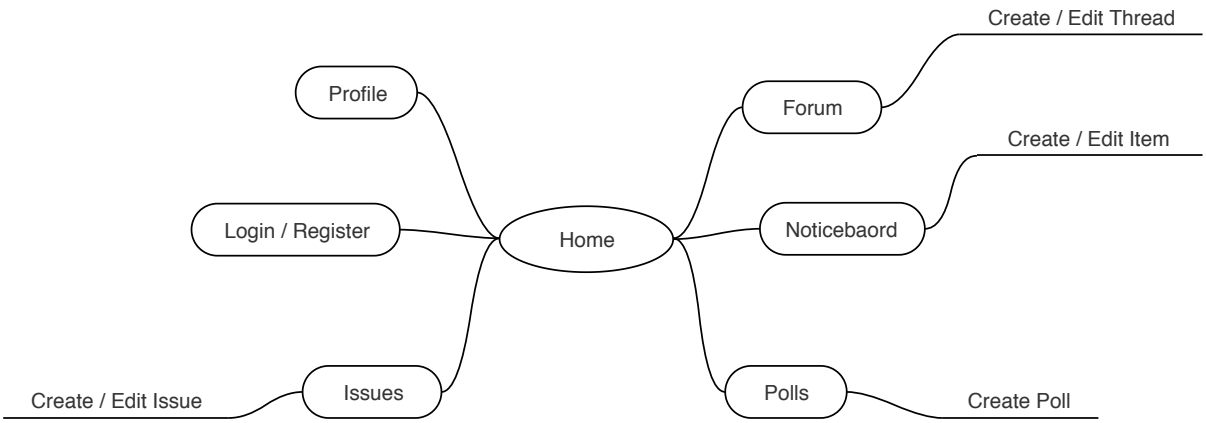

Figure 3.2: Sitemap

Chapter 4
Implementation

4.1 Introduction

Based on the evaluation carried out in Section 2.4 the Single Page Application was implemented with Vue and Nuxt. To give the reader a better understanding of how the application was built, the first few sections of this chapter will discuss which additional tools and functionalities Nuxt provides, how Vue is used and Separation of Concerns (SoC) is achieved and briefly illustrates how the frontend and the API communicate with each other.

4.2 Nuxt.js

Nuxt's primary use case is to provide Server Side Rendering for Vue-based applications. It also advertises itself as a framework which "presets all the configuration needed to make your development of a Vue.js application enjoyable" [14]. While this blurry description is not very meaningful on its own, it indicates that Nuxt pre-configures certain parts of a Vue application. The most important of which are: Server Side Rendering, automatic setup of routing and store management.

4.2.1 Rendering modes

Nuxt offers three rendering modes: SSR, "Prerendering" and "Single Page Application". For descriptions of the first two please see Section 1.4.3. The "Single Page Application" mode does not offer server-rendering capabilities while still pre-configuring routing and state management. This mode is especially useful for applications that require users to be authenticated before being able to use them. The reason why is simple: Server Side Rendering can only improve the SEO ranking of a page if crawlers are able to look at the full contents of it. Fully implementing SSR for an application that only shows a login form would therefore be pointless.

Then why should this project use Nuxt, which is primarily used for SSR if the application cannot even profit from it because authentication prevents crawlers from seeing anything but a login form? The answer is because 1) state management and routing are auto-configured as well and 2) because in the future, the frontend might expose parts of the system in a publicly accessible way without the need of authentication. By using Nuxt from the beginning, the transition to this is much easier.

4.2.2 Usage

When using Nuxt, different folders contain different parts of the frontend application. Behind the scenes, Webpack - one of the most popular bundler tools consolidates, minifies, transforms and transpiles the code of these folders into browser-readable plain JavaScript code.

Folder	Description
assets	contains un-compiled assets such as images, global css or fonts
components	contains reusable Vue Single File Components
layouts	contains Vue files that define the applications theme
middleware	defines files which are run before a route changes, e.g check if user is authenticated before redirecting
pages	contains the pages of the application, Nuxt automatically creates routes corresponding to a page's name
plugins	contains plain JavaScript files which are run before instantiating the application, e.g used for inject adding an internationalization plugin
static	files in this is directory are directly mapped to server root and are not compiled or renamed in any way, e.g robots.txt
store	contains Vuex store files, every file added in this directory is automatically mapped to the global store
nuxt.config.js	contains the Nuxt configuration for the application, e.g which plugins and middleware to use
package.json	is not Nuxt specific but added by npm and contains application dependencies and scripts

Table 4.1: Nuxt File Structure

Typically, functional components are defined inside the components folder. These components might include other components in order to compose more complex architectures. Pages and layouts are also nothing more than Vue components, however, they are used in very specific ways. The file structure of the pages folder is automatically read by Nuxt and routes are generated from it. For example the file pages/index.vue is shown when a user visits `hausversammlung.at`, whereas the file pages/issues/index.vue translates to the url `hausversammlung.at/issues/`. Layouts are used to change the look and feel of the application. For example it is possible to create a layout file which includes a navigation bar on the top of the page. Nuxt then allows developers to set a layout property inside a page which corresponds to a layout file, resulting in easily reusable themes.

4.3 Vue

This section aims to extend the basic description of Vue's usage as described in Section 1.3.1 so that code samples of this chapter are more easily understood. For the sake of brevity, some parts will be omitted if they are not crucially relevent (e.g transitions).

Discusses: Single File Components separated into HTML, JS and CSS v-bind and shorthand, v-model ternary expressions events how they are emitted and caught props methods computed properties v-if conditional classes vuex store routing

4.4 Separation of Concerns

A very important aspect of modern software architectures is Separation of Concerns (SoC) which is often regarded to as principles of modularization of code and object-oriented design [8]. In the case of traditional web design HTML, CSS and JavaScript are often physically separated

(separation of file types) leading to increased maintainability of code, a less tightly coupled system and decreased probability of violating DRY principles. Furthermore, it helps building complex layered systems which can be actively worked on by multiple developers.

Vue however, takes different approaches on SoC depending on the way it is used in a project: either Vue.js is added to existing sites by using a standard HTML script tag or by using a complete bundling tool like Webpack or Browserify to add it as a dependency. Using Vue by adding a script tag has some disadvantages: it only allows developers to target a specific HTML container element on the page Vue is embedded in, and also brings certain disadvantages such as no CSS support inside the Vue component itself and no build steps which requires the developer to fall back to plain HTML and ES5 JavaScript rather than using various preprocessors.

```
Vue.component('todo-component', {
    template: '<h1> {{ newItem }} </h1>',
    data() {
        return {
            items: [
                {
                    id: '1',
                    title: 'Buy milk',
                    completed: false
                },
            ],
            newItem: ''
        };
    },
    methods: {
        addItem() {
            // LOGIC OMITTED FOR READABILITY
        }
```

Listing 4.1: Vue component without bundler

The more preferable approach would be to use a so called “Single File Component”, a file with a “.vue” extension. These files however have to be compiled with build tools like Webpack or Browserify to extract the corresponding concerns. A sample file is depicted in Listing 4.2.

```
<template>
  //HTML
  <p>{{ text }}</p>
</template>

<script>
//JS
export default {
  data() {
    return {
      text: 'Hello World!'
    }
  }
}
</script>

<style scoped>
//CSS
  p {
    color: red;
  }
</style>
```

Listing 4.2: Vue Single File Component

By defining .vue files developers can utilize syntax highlighting, linting and component-scoped CSS. Furthermore preprocessors like Pug, Babel or SASS can be used to further improve the development process. This however comes with an important tradeoff: as Single File Components rely on the use of build tools, they are not very suitable projects which are not completely vue-based

Clearly HTML, CSS and JavaScript are not separated into their own respective files even when using Single File Components which leaves us with the question of how Separation of Concerns is achieved. As stated in the Vue Documentation:

> "... separation of concerns is not equal to separation of file types. In modern UI development, we have found that instead of dividing the codebase into three huge layers that interweave with one another, it makes much more sense to divide them into loosely-coupled components and compose them. Inside a component, its template, logic and styles are inherently coupled, and collocating them actually makes the component more cohesive and maintainable [26]."

Instead of creating different files in which logic, presentation and design reside in, Vue consolidates these tiers into a single file, hence the name "Single File Component", making it more maintainable and editable in the process.

4.5 API Endpoints

4.6 Authentication / Authorization

4.7 Forum

4.8 Polls

4.9 Noticeboard

4.10 Issue Log System

4.11 State Management

4.12 Internationalization

4.13 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

VCS Version Control System. 14

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

Appendix A
User Stories

A.0.1 Noticeboard

As a user I want to create noticeboard entries so other users can see my messages
As a user I want to edit my noticeboard entries so that I fix mistakes
As a user I want to view all noticeboard items so that I have a complete overview
As a user I want to search for specific items so that I can find an item faster

Table A.1: Noticeboard related User Stories

A.0.2 Polls

As a proprietor I want to create polls other proprietors can vote on so that we as a group can make decisions
As a proprietor I want to view all polls so that I have a complete overview
As a proprietor I want to vote for a poll option so I my decision is accounted for

Table A.2: Poll related User Stories

A.0.3 Forum

As a user I want to create threads so other users can see my question or discussion
As a user I want to edit my threads so that I can fix mistakes
As a user I want to view all noticeboard items so that I have a complete overview
As a user I want to search for specific items so that I can find an item faster

Table A.3: Forum related User Stories

A.0.4 Issue Management

As a user I want to submit an issue so that it can be taken care of
As a user I want to edit my issues so that I can fix mistakes
As a user I want to view all logged issues so that I have a complete overview
As a user I want to search for specific issues so that I can find an item faster
As the facility management I want to change an issues state so I can let users know their issues are taken care of

Table A.4: Issue related User Stories

A.0.5 Profile

As a user I want to change my profile details so they reflect the right data

Table A.5: Profile related User Stories

A.0.6 Authentication

As a user I want to create an account so I can use the system
As a user I want to login to an existing account so I can use the system
As a user I want to change my password so that I can access my account
As a user I want to validate my email address so that emails are sent to the right address

Table A.6: Authentication related User Stories

A.0.7 Bilinguality

As a user I want to change the language so I can understand the site's content

Table A.7: Bilinguality related User Stories

A.0.8 House

As a user I want to create a new house so I can manage this property
As a user I want to invite other users to my house so that we can manage it together
As a user I want to accept an invitation
As a user I want to switch between houses

Table A.8: Property Management related User Stories

Appendix B
Use Case Flows

B.0.1 Noticeboard

The user opens the noticeboard page by using the navigation bar
The user clicks a "create" button
The user fills a from with a title and a message
The system checks for appropriate authorization
The system adds a new entry to the database
The user is redirected to the main noticeboard page

Table B.1: Use Case Flow: Create Noticeboard

B.0.2 Polls

B.0.3 Forum

B.0.4 Issue Management

B.0.5 Profile Page

B.0.6 Add-House Page

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, 29).

[4] Olga Filipova. *Learning Vue. js 2*. Packt Publishing Ltd, 2016 (cit. on pp. 1, 2, 29).

[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] Phillip A Laplante. *What every engineer should know about software engineering*. CRC Press, 2007 (cit. on p. 22).

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

[10] 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).

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

[12] 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).

[13] 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

[14] Nuxt.js contributors. *Nuxt.js - The Vue.js Framework*. 2019. URL: https://nuxtjs.org/ (visited on 16/03/2019) (cit. on p. 21).

[15] 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).

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

[17] 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).

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

[19] 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 pp. 8, 9).

[20] 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).

[21] *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).

[22] 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).

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

[24] 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).

[25] 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).

[26] *Single File Components — Vue.js*. URL: https://vuejs.org/v2/guide/single-file-components.html#What-About-Separation-of-Concerns (visited on 17/03/2019) (cit. on p. 24).

[27] *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