JOHANNES KEPLER
UNIVERSITY LINZ

Author
Katrin A. Kern, 11774714

Submission
Institute of
Networks and Security

Thesis Supervisor
Dr. **Michael Roland**

March 2022

Comparing Modern Front-End Frameworks

Bachelor's Thesis

to confer the academic degree of

Bachelor of Science

in the Bachelor's Program

Computer Science

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Abstract

Web technologies have evolved rapidly in the last couple of years and applications have gotten significantly bigger. Common patterns and tasks have been extracted into numerous frameworks and libraries, and especially JavaScript frameworks seem to be recreated daily. This poses a challenge to many developers who have to choose between the frameworks, as a wrong decision can negatively influence the path of a project.

In this thesis, the three most popular front-end frameworks Angular, React and Vue are compared by extracting relevant criteria from the literature and evaluating the frameworks against these criteria. Angular is then used to develop a web application for displaying data from the Android Device Security Rating.

Acknowledgements

This work has been carried out within the scope of ONCE (FFG grant FO999887054) in the program "IKT der Zukunft" and has partially been supported by Digidow (Christian Doppler Laboratory for Private Digital Authentication in the Physical World) and the LIT Secure and Correct Systems Lab. We gratefully acknowledge financial support by the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility,Innovation and Technology (BMK), the Austrian Federal Ministry for Digital and Economic Affairs (BMDW), the National Foundation for Research, Technology and Development, the Christian Doppler Research Association, 3 Banken IT GmbH, ekey biometric systems GmbH, Kepler Universitätsklinikum GmbH, NXP Semiconductors Austria GmbH & Co KG, Österreichische Staatsdruckerei GmbH, and the State of Upper Austria.

Contents

Chapter 1

Introduction

Since the early 1990s, web content evolved from text-based, static websites to the highly dynamic and interactive web applications we are used to today; especially since the rise of AJAX (Asynchronous JavaScript + XML). Common code pieces, patterns and structures were collected into frameworks and libraries to increase efficiency and maintainability. Naturally, with the acceleration of web development, lots of new frameworks, libraries and web technologies were created to accommodate the needs of both users and developers. In particular JavaScript-based frameworks and libraries have been entering the market at such a frequency that jokes (e.g. a website called "Days since last JavaScript framework" [5]) and blog entries with titles similar to "100+ JavaScript frameworks" [30] emerged in the developer community, but also serious criticism of the status quo (e.g. "YAFS" – Yet Another Framework Syndrome [40]). Additionally, developers and team leaders must take extra care when selecting a tech stack for a product. Picking the wrong framework might waste time (and therefore money) if the developers are not used to it or its underlying language or if changing to another one becomes necessary. The later this happens in the development process, the higher the costs will be. It is therefore necessary to take some time to review the advantages and disadvantages of various frameworks as well as the requirements for the desired application to find a good fit and save time and money.

This thesis aims to help with that process by evaluating the three most widely spread front-end frameworks – Angular, React and Vue. One of the frameworks is then used for developing a website displaying data from the Android Device Security Rating.

A few requirements were given for the website:

- The data from the security rating should be listed in tabular form.
- The interface should provide common filter functionality, e.g. filtering by vendors or models.
- The applied filters should be represented in the URL to allow the user to share particular views.
- The website should be fully responsive.

Another requirement that arises implicitly: The website should be able to handle large amounts of data, as the data set will continuously grow, meaning that the user should not notice any delays while filtering or navigating.

In the first chapter of the thesis, a quick overview of the frameworks is given, and the evaluation process and its results are described. The second chapter contains a bottom-up description of the most important implementation aspects of the practical part of this thesis.

Chapter 2

Evaluation

2.1 Methodology

To collect relevant criteria, literature on (web) applications and their comparison methods were studied. The idea was to identify common criteria throughout the studies, consolidate them and use them for evaluating the frameworks. The following section contains brief descriptions of the literature including the found attributes and the resulting selection of criteria along with their definitions and rationale.

2.2 Literature

Mairiza et al. [22] conducted a literature study on non-functional requirements (NFRs), resulting in an extensive catalogue of NFRs. The complete list spans over 114 NFR types, which is why only the "most commonly considered NFRs" are listed here:

- Performance (e.g. response times, resource utilization),
- Reliability (consistency, maturity, fault tolerance),
- Usability (user friendliness, learnability),
- Security (authentication, confidentiality), and
- Maintainability (testability, analysability).

Villamor et al. [9] defined a comparison model for agile web frameworks (full stack web frameworks). As the selected front-end frameworks do not operate on the full stack, not all of the criteria are applicable. Relevant criteria are:

- Presentation (automatic generation of basic representation, internationalization supported, etc.),
- Security (static typing, user authentication, escaping mechanisms, etc.),
- Usability (development experience),
- Testing (support for various types of testing),
- Service Orientation (e.g. REST support), and
- Adoption (community, maturity, etc.).

Shan and Hua [37] categorized Java web frameworks by their type (e.g. component framework) and noted a few basic design principles a web application framework should follow. Additionally, they noted some reasons why a few of them have become more dominant on the market than others.

- Simplicity – usage of the framework should require less and simpler code
- Consistency (e.g. of components and conventions)
- Efficiency – applications should be performant and scalable
- Integration – with existing solutions
- Reusability – constructs (e.g. components) should be reusable
- Non-intrusive – HTML/markup should not be polluted with other semantics
- Diagnosis – diagnostics and debugging information
- Development tools – “maximum tool support with minimum dependency on special tools”
- Additional factors: Community, Standardization, Documentation, Tool integration, Vendor endorsement, Ease of use

Gizas et al. [11] evaluated JavaScript frameworks using code quality metrics (e.g. lines of code and complexity metrics) and performed validation and performance tests with various tools:

- Active community (mentioned, but not analyzed)
- Quality (i.e. size, complexity and maintainability of the framework code)
- Validity (errors)
- Performance

In contrast to other comparisons, the focus lies on the code quality of the framework itself and not on the functionality or non-functional properties it provides to its users. Nonetheless, a complex framework that is hard to maintain is error-prone and its development more likely to be slower or even discontinued—which in turn affects the quality of the application developed with it.

2.3 Evaluation Criteria

Mairiza et al. [22] provided a good basis for the selection with the most common NFRs they had identified in their literature review. In fact, most of the criteria found in other literature are either the same, fall into the five categories or are in between. The five categories were then used as a rough classification and some others were used as “sub-criteria” or metrics. In the following sections, all categories and their metrics are described in detail. It should be noted that not all metrics can be determined irreproachably and that the aim is not to rank frameworks from best to worst, but merely to give an overview of the main factors that might influence the selection of a framework for a project.

Performance Both **speed (P1)** and **application size (P2)** determine performance, which is a decisive factor in whether people use an application or not. For the comparison of speed and (transfer) size, the results of the “js-framework-benchmark” [21] were used, as it measures the performance of frameworks using data table operations—which corresponds well to the intended use case.

Reliability Frameworks that are **supported by large organisations (R1)** are less likely to be discontinued, especially if they use them for their own applications (e.g. Facebook using React [20]). **Regular updates, fixes, security patches (R2)** and new features keep the framework secure and up to date. A dedicated **release policy** describing release cycles and versioning practices makes updates more predictable. The last point is **maturity (R3)**: a framework that has been developed for multiple years and is used in major applications can be considered as "mature" and more reliable than newcomers.

Maintainability **Documentation (M1)** is the basis for developing and maintaining an application with a framework. Besides the regular documentation on the website, this also includes tutorials, demo applications and other resources such as books and courses. Since it would go beyond the scope to assess the quality of all these resources, this criterion is assessed on the basis of quantity. **Tooling (M2)** includes every tool and feature that facilitate the development and maintenance of an application, e.g.:

- support for various testing types and frameworks,
- IDE support, e.g. syntax highlighting and plugins for common IDEs (Eclipse, IntelliJ, Visual Studio), and
- other forms of supporting development tasks, e.g. browser extensions.

Usability The steepness of the **learning curve (U1)** influences how quickly developers can familiarise themselves with a framework. A steep curve can make it harder to find developers, especially experts, or prolong the onboarding process. Even though learning something is a highly subjective matter, some things can be observed:

- Is it necessary to learn additional, possibly unusual technologies to use the framework?
- Does the framework provide information about best practices, project structure, etc.; anything that facilitates the learning process and leaves less room for error?
- Documentation (M1) and community (U2) are two other factors that influence the learning curve.

A large **community (U2)** makes it easier to get answers on specific questions and find resources such as tutorials, projects, etc. This criterion is measured by the number of GitHub stars and StackOverflow questions to get a sense of the number of people interested in or working with the framework, as well as the presence of other forms of community such as events, conferences, and forums.

It should be noted that maintainability and usability are closely related in this context. For example, good documentation and good developer tools improve the maintainability, but also the usability of a framework. Conversely, a large community improves developer experience, but also makes it easier to find solutions to problems; or a moderate learning curve makes the framework comfortable to use at the beginning, but also makes it easier to hire and train new developers for existing projects.

Security A **security policy (S1)** describes how the framework maintainers handle security issues, how developers can report vulnerabilities, and it could even provide a financial incentive for finding vulnerabilities. The framework could (and should) also have **built-in protection (S2)** against common threats or at least **guidelines (S3)** for developers on how to secure their application.

2.4 Frameworks

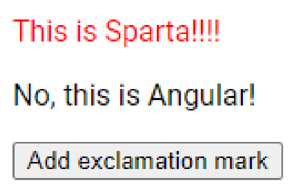
Before the frameworks are evaluated in detail, a brief introduction is given for each framework, highlighting the key concepts.

2.4.1 Angular

Angular or Angular 2+ is a cross-platform TypeScript-based front-end framework maintained and developed by Google and its community. Its predecessor AngularJS was released back in 2010, but the framework was rewritten from scratch and released again in 2016 [3]. Its basic building blocks are components consisting of a TypeScript class, an HTML template and CSS styles. The template can be extended with special syntax that enables multiple features:

- Interpolation: Fields from the class can be inserted into the template and are automatically update when changes are made.
- Property Binding: HTML attributes, e.g. id or class, can be bound to a field in the TypeScript class, which also allows for dynamic changes.
- Event Binding: similar to Property Binding, except that a specific event (e.g. click, keystrokes) triggers the execution of the specified method in the component.
- Directives: special classes that change the behaviour or appearance of HTML elements (attribute directives, e.g. NgClass, NgModel) or add and remove DOM elements (structural directives, e.g. NgIf, NgFor)

Figure 2.1 and the code in listings 2.1, 2.2 and 2.3 show a minimal example including the above-mentioned features. The small application provides a button that dynamically appends exclamation marks to the displayed sentence. If the number of exclamation marks exceeds 3, a second sentence appears.


This is Sparta!!!!

No, this is Angular!

Figure 2.1: Angular example application.

Listing 2.1: app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html',
})
export class AppComponent {
  name: string = 'Sparta';
  color: string = 'red';
  exclamationCount: number = 0;

  addExclMark() {
    this.name += '!';
    this.exclamationCount++;
  }
}
```

Listing 2.2: app.component.html

```
<p [style.color]="color">
  This is {{ name }}
</p>
<p *ngIf="exclamationCount > 3"
  >No, this is Angular!
</p>
<button (click)="addExclMark()">
  Add exclamation mark
</button>
```

Listing 2.3: app.component.css

```
p {
  color: black;
}
```

Listing 2.4: React example code.

```
import React from "react";
import ReactDOM from "react-dom";

class App extends React.Component {
  state = {
    name: "Sparta",
    exclCount: 0,
    nameColor: "red"
  };

  addExclMark() {
    this.setState((prevState) => ({
      name: (prevState.name += "!"),
      exclCount: prevState.exclCount + 1
    }));
    console.log(this.state.exclCount);
  }

  render() {
    return (
      <div>
        <p style={{ color: this.state.nameColor }}>
          This is {this.state.name}
        </p>
        {this.state.exclCount > 3 && (
          <p>
            No, this is React!
            <br />
          </p>
        )}
        <button onClick={this.addExclMark.bind(this)}>
          Add exclamation mark
        </button>
      </div>
    );
  }
}

ReactDOM.render(<App />, document.getElementById("container"));
```

2.4.2 React

Like Angular, React is supported by a large corporation (Facebook) and its community. It was first used in 2011 and released as an open source project in 2013 [34]. The difference to the other two frameworks is that React is “only” a front-end JavaScript library responsible for rendering UI elements. Nevertheless, it was still considered for this comparison because its large ecosystem makes it equally powerful.

React uses so-called React elements to build the user interface. These are created with JSX (JavaScript syntax extension) and are rendered into the DOM nodes. Components in React are not a composition of logic, template and style files like in Angular, but essentially JavaScript functions that return React elements. To illustrate this, listing 2.4 contains the sample application from before, written in React. Instead of manipulating the DOM directly, React uses a concept called virtual DOM. If something changes, the previous virtual DOM and the changed one are compared using an efficient algorithm so that React only needs to update the changed elements and not the entire DOM tree, which speeds up the whole process significantly.

Listing 2.5: Vue example code.

```
<template>
  <div id="app">
    <p v-bind:style="{ color: nameColor }">This is {{ name }}</p>
    <p v-if="exclCount > 3">No, this is Vue!</p>
    <button v-on:click="addExclMark()">Add exclamation mark</button>
  </div>
</template>

<script>
export default {
  el: "#app",
  data() {
    return {
      name: "Sparta",
      exclCount: 0,
      nameColor: "red",
    };
  },
  methods: {
    addExclMark: function () {
      this.name += "!";
      this.exclCount++;
    },
  },
};
</script>

<style>
p {
  color: black;
}
</style>
```

2.4.3 Vue

Vue is another JavaScript framework that was released in 2014 [41]. Unlike the other two frameworks, it is not supported by an organisation and is still developed by its creator Evan You and his core team members. Vue incorporates concepts from both Angular and React. It uses a virtual DOM approach and supports JSX, but also supports TypeScript and has directives similar to those in Angular. Components are usually defined as single file components consisting of a template, script and style part, as shown in listing 2.5.

2.5 Evaluation

This section evaluates the frameworks against the criteria defined in section 2.3, followed by a summary and rationale for choosing Angular for the implementation part of this thesis.

2.5.1 Performance

The following framework versions from the “js-framework-benchmark” [21] were considered for the comparison: react-v16.8.6-non-keyed, angular-v8.0.1-non-keyed and vue-v2.6.2-non-keyed.

Speed (P1): Overall, no significant difference in application speed could be identified. Only Vue performs worse than the other two for the "select row" implementation, and Angular has a slightly worse boot up time and memory usage after page load.

Size (P2): The "total kilobyte weight" metric—the size of all resources loaded into the page—is also more or less the same, with Vue being slightly more lightweight than React and Angular.

2.5.2 Reliability

Supported by large organisations (R1): Both Angular and React are supported by two large technology companies: Google and Facebook. Vue is developed and maintained by a former Google employee and his core team and is funded by sponsors [44].

Regular updates, release policy (R2): Unsurprisingly, all three frameworks receive regular updates including new features and bug fixes.

Angular's documentation contains a detailed release policy that describes the release cycle, support policy and even deprecation practices [2]. There is also a roadmap that lists the features that are already in development or at least planned.

React states its release policy in the FAQs and informs its users about new releases and planned features in a blog [32, 33].

The Vue documentation on their website does not provide a special section about releases and versioning, but the section "Release management" in a GitHub read-me describes their schedule and LTS policy [42]. Planned features and open tasks are published via GitHub Projects (a Kanban-like to-do list) [45].

Maturity (R3): React is the oldest framework, with its first public release in 2013. Vue followed a little later, with the first commits to the official repository at the end of 2013 and the first "named" release in 2014. With its first release in 2016, Angular is the youngest of the three frameworks. Its predecessor AngularJS is not considered here, as it is fundamentally different from the new Angular.

All three frameworks are used by major companies in enterprise-grade applications, for example Alibaba, GitLab (Vue) [6, 36], Facebook (React), Google and Microsoft (Angular) [27].

Measured by their age and their usage, all of the frameworks can be considered as mature.

2.5.3 Maintainability

Documentation (M1): All three frameworks provide extensive documentation on their websites, with Angular and React also providing sample applications in a sandbox environment and Vue linking to external online course platforms (some of which require a subscription).

For researching books, the search functionality on the website of the well-known publisher O'Reilly Media was used [13]. As of April 2020, there were 47 books on Angular, 34 on React and 22 on Vue in their database. The database also includes resources from other publishers such as Manning and Packt.

A similar search for courses was conducted on the websites of Coursera, one of the biggest education platforms. Here, the results had to be filtered manually as the search function returned a lot of irrelevant results. In April 2020, 10 courses were offered on Angular, 8 on React and none on Vue.

Tooling (M2): All three frameworks have a section on testing in their documentation. While Vue mainly gives recommendations on which framework to use for which testing type, both Angular and React offer additional tips and tricks for testing. Angular also includes the testing framework Jasmine in its downloads (but using other frameworks is also possible).

For the evaluation of IDE support, three popular IDEs (Eclipse, IntelliJ and Visual Studio) were considered. Several third-party tools for all three frameworks are offered on the Eclipse and Visual Studio marketplaces. Jetbrains provides official tools for Angular and Vue for its IDE IntelliJ, and third-party tools are available for React.

Similarly, when researching browser extensions, three popular browsers (Chrome, Firefox, Edge) were considered. For Chrome, there are official extensions developed by Google, Facebook and the Vue team, as well as some unofficial tools. In the Firefox marketplace, there are official tools for React and Vue and some unofficial ones for Angular. The situation is different on Edge, where there is only one official tool for React, the rest are supported by third-party tools. Overall, all three frameworks seem to be well supported by both official and third-party tools.

2.5.4 Usability

Learning curve (U1): For React and Vue, knowledge of HTML, CSS and JavaScript is sufficient to get started. Angular uses TypeScript for its dynamic parts, which may require some time to get used to.

Since React is only a UI rendering library, other libraries will most likely need to be added to develop a full application. Vue is also stripped down to a minimum, but provides a few official core libraries to handle common tasks like state management and routing. Angular can be considered a "batteries included" framework that comes with everything needed for development. While the opt-in approach of React and Vue makes no difference in the learning time required—learning about routing in Angular or learning about an external library will take about the same amount of time—it requires an additional research and selection process.

Community (U2): According to the number of GitHub stars and StackOverflow questions in the table 2.1, React is the most popular framework overall among the three selected. The numbers for Angular and Vue were a little surprising. Vue has a lot of GitHub stars, but only a quarter of the questions of the other two frameworks. This suggests that the user base is still quite small, despite its familiarity and the fact that many developers have "saved it for later". Angular's numbers are the other way around: less than 60k GitHub stars, but over 200k questions on StackOverflow might suggest a larger user base that just doesn't

Table 2.1: Number of GitHub stars and StackOverflow questions, retrieved in Feb. and Mar. 2020.

	Angular	**React**	**Vue**
GitHub stars	57.9k	144k	157k
StackOverflow questions	205k	194k	51k

use GitHub as much—possibly enterprise users who manage their repositories elsewhere. Another possibility is that the numbers include (incorrectly tagged) questions about the older AngularJS framework.

There are events, conferences and meetings for all three frameworks, with Angular and React having slightly more to offer than Vue—not surprising given the R1 criterion.

2.5.5 Security

Both Vue and Angular have a separate section in their documentation dedicated to security, describing where to contact them, best practices **(S3, guidelines)** and what measures the maintainers have already taken against common threats **(S2, built-in protection)** [1, 43]. Angular also refers to Google's security philosophy **(S1, security policy)**, which includes a bug bounty program. For React, no official policy or guidelines could be found, only a reference to Facebook's bug bounty program [31].

2.5.6 Summary

Table 2.2 summarizes the results of the comparison. The selected criteria highlights the subtle differences between the frameworks well, and overall, all three can be considered mature, stable and well suited for the task at hand. Ultimately, Angular was chosen for developing the application, as it seems to offer the clearest structure and documentation from the perspective of a new developer (there was no previous noteworthy experience with any of the frameworks) with minor downsides in terms of size and speed compared to the other frameworks.

Table 2.2: Overview of the results (++ very good, + good, o partial, – no support).

		Angular	React	Vue
P1	Speed	+	++	+
P2	Size	+	+	++
R1	Supported by large org.	++	++	o
R2	Release policy	++	+	o
R3	Maturity	++	++	++
M1	Documentation	++	++	+
M2	Tooling	+	+	+
U1	Learning curve	+	+	+
U2	Community	++	++	+
S1	Security policy	+	o	–
S2	Built-in protection	+	–	+
S3	Guidelines	+	–	+

Chapter 3

Implementation of a Database View in Angular

In the first section of this chapter, the idea and overall architecture of the application is explained. In the second section, any additional technologies used in the project besides Angular are briefly introduced, along with the used version and a few notes on their usage. In the remaining sections, the implementation is explained step by step in a bottom-up way, focusing on the most important parts.

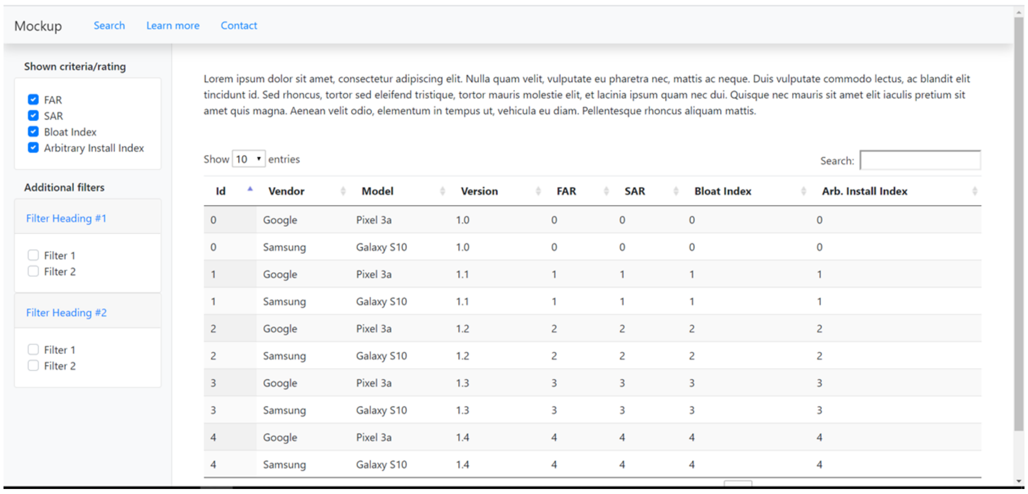
3.1 Overview

The application needed to include the following parts:

1. A table showing the device data and the corresponding security rating results, and
2. functionality for filtering by vendors, models, and various attributes from the rating.

Figure 3.1 shows a first mockup of those two parts. Additionally, the filter settings should be represented in the URL to enable users to share views.

The application structure (Figure 3.2) is simple: a PostgreSQL database provides the data that can be accessed by the Express API running on a Node.js server. The Angular application consists of a data service that accesses the API and is injected into the data table component, to which it passes the data.



✓ FAR
✓ SAR
✓ Bloat Index
✓ Arbitrary Install Index

Filter Heading #1: ☐ Filter 1 ☐ Filter 2

Filter Heading #2: ☐ Filter 1 ☐ Filter 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis.

Id	Vendor	Model	Version	FAR	SAR	Bloat Index	Arb. Install Index
0	Google	Pixel 3a	1.0	0	0	0	0
0	Samsung	Galaxy S10	1.0	0	0	0	0
1	Google	Pixel 3a	1.1	1	1	1	1
1	Samsung	Galaxy S10	1.1	1	1	1	1
2	Google	Pixel 3a	1.2	2	2	2	2
2	Samsung	Galaxy S10	1.2	2	2	2	2
3	Google	Pixel 3a	1.3	3	3	3	3
3	Samsung	Galaxy S10	1.3	3	3	3	3
4	Google	Pixel 3a	1.4	4	4	4	4
4	Samsung	Galaxy S10	1.4	4	4	4	4

Figure 3.1: First mockup of the user interface.

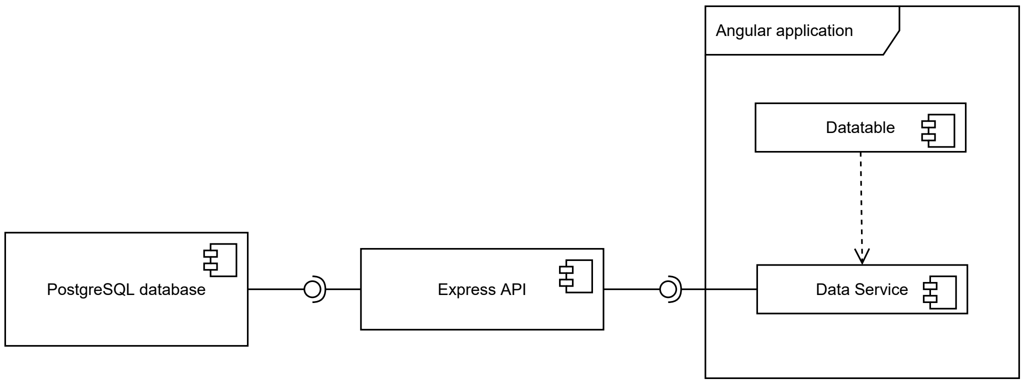


Figure 3.2: Application overview.

3.2 Complementary Technologies

3.2.1 PostgreSQL

PostgreSQL is a free and well-known DBMS released in 1996 [28]. The setup and schema is described in the section 3.3. In this implementation, version 12.3 was used.

3.2.2 Node.js

Node.js [23] is a JavaScript runtime environment used for creating Web servers, allowing developers to write a complete web application in a single language. Version 12.6.3 was used, which was the LTS version when the application was implemented.

3.2.3 NPM

NPM [25] is a package manager included by the Node installation that automates installing and updating the libraries and frameworks used in a project. The website npmjs.com allows searching for packages and provides information about dependencies, versions, download numbers, links to the repositories etc. Version 6.14.4 was used in the project. Although it is a practical and widely used tool, there are some security issues that have come up in the research for this thesis that developers should be aware of:

Typo-squatting: Packages containing malicious code were named similar after popular packages, which led to accidental usage of the wrong package [24].

Module Hijacking: Lots of maintainers have or had very weak credentials, which affected thousands of packages—including top packages like react, react-dom, express or mysql [38]. In another incident, malicious code was released in a popular package after the account of its maintainer was hijacked [35].

Social Engineering: In a blog post, a developer described a way that would have allowed him to steal data from websites by contributing to open source projects with seemingly harmless code (not an actual attack) [10]. In a different case, the original author of a popular library (over two million weekly downloads) did not have the time and resources anymore to maintain it and gave it away to another person. This person turned out to be a hacker and added code that stole information from users' cryptocurrency wallets [4, 39].

To mitigate those threats, there are some things that can be done by the developer or have been done by NPM:

General: Automatic updates, especially major releases, should be prevented—this might lower the risk of module hijacking and social engineering, as the malicious version of a package could be detected and removed before it even reaches the application through an update. Additionally, it ensures that all developers on a team use the exact same versions of packages. This can be achieved by either specifying versions without the typical semantic versioning range prefixes or by using the package-lock.json, which is automatically generated after modifying the dependency tree and locks packages to a specific version [26]. At the same time, updates should be installed regularly—especially security patches. The total number of dependencies should be kept to a minimum—which is often easier said than done, especially when using frameworks and libraries that depend on lots of other packages [17].

Typo-squatting: Skovoroda [38] showed that even well-known and well-maintained packages are not safe from malicious intent. Still, using packages from trustworthy sources and organizations that are regularly checked and maintained is the safer bet. Special care should also be taken when installing packages—ideally they should be looked up on npmjs.com, where the command to install them can be copy-pasted.

Module hijacking: Concerning the weak credentials, NPM introduced token authentication instead of basic authentication headers, password rules and 2-factor authentication [38].

Social Engineering Besides the above mentioned measures like preventing automatic updates and using less packages if possible, there is also the possibility of creating a private npm repository and checking packages thoroughly before adding them [17].

3.2.4 Express.js

Express [8] is a small web application framework written for Node.js and typically used for writing REST APIs (see section 3.4). Version 4.17.1 was used.

3.2.5 Knex.js

Knex.js [18] is a query builder for various database technologies, mainly intended for usage with Node.js. It adds an additional layer of abstraction to SQL queries, with two main advantages:

- Dynamic queries with conditions can easily be written in JavaScript syntax.
- Knex makes it harder to write unsafe queries. The developer would have to explicitly use the raw functions to do this.

A disadvantage is, of course, the extra layer that is not absolutely necessary, especially not for simple queries (e.g. SELECT * from XY). A good compromise would be using Knex for complicated queries, where the layer of abstraction makes it easier to write and read and using template strings for simple queries. In this project, Knex (version 0.21.5) was used throughout the whole API, both for consistency and because the API is quite small.

3.2.6 PrimeNG

PrimeNG [14] is a large UI component library for Angular (also available for React, Vue and JSF). At the beginning, the intention was to just try it and quickly pull up a UI prototype with it, but it turned out to be useful and integrates well with Angular. The components and templates can be viewed and tried out on the showcase website [15]. It also contains the documentation and the source code of the examples. In this project, version 10.0.0 was used.

3.3 Database

3.3.1 Data & Scheme

The data can be split into two parts:

- Device data, e.g. manufacturer, model, Android version, firmware version etc.
- Attributes, e.g. False Acceptance Rate, Bloat Index etc.

In the beginning, the measurement and analysis of attributes just started, meaning that not a lot of data was available and more attributes were added later. Because of this, the front-end had to be as dynamic as possible. Attributes can be added to the database and will be displayed without touching the front-end code. This will be explained in detail in section 3.5. During development, the scheme of the test database looked like figure 3.3.

3.3.2 Setup

A full SQL-script for recreating the table and populating it with test data can be found in the project files (config/database.sql). This section aims to explain the creation of the data view that is sent to the front-end.

View The goal was to create a view similar to figure 3.4—displaying the attributes and values for each device horizontally—which is already more or less the (unfiltered) table that should be displayed on the front-end.

To achieve this, the function “crosstab” [29] was used. It “tips” or “pivots” the data, as shown in figures 3.5 and 3.6.

Listing 3.1 shows the inner part of the view creation query (and actual crosstab query).

Table 3.1: Result of the query using the test data.

devid	far	frr	sar	bi	aii	ssti	psr	kbb	sba	fbe	mus	sus
1	0.1								true	false	false	file
2	0.2								false	false	false	file
5	0.3								false	false	false	file

The first parameter of the crosstab function is a query string selecting the device ID, attribute name and measurement value from the respective joined tables. The ORDER BY 1 is important, as it makes sure that only values with the same device ID are brought together. The second parameter contains a string (dollar-quoted, which allows using unescaped single quotes within the string) that defines the categories (i.e. the columns) of the resulting table. This could also be written as “SELECT attname from attribute”. Here, the columns were specified explicitly to ensure that the order is correct. Note that the order has to be exactly the same as the one in the column definition below (AS t(...)). If the second parameter was omitted and some measurements were missing, the values would be filled up from left to right, as shown in figures 3.7 and 3.8.

Using the test data, executing the query would result in the data shown in table 3.1.

To create the final view (Listing 3.2), this data is then joined with the result of a query selecting various device information. Both queries required selecting the device ID; to avoid the duplicate “USING(devid)” is used instead of “ON devid” in the join.

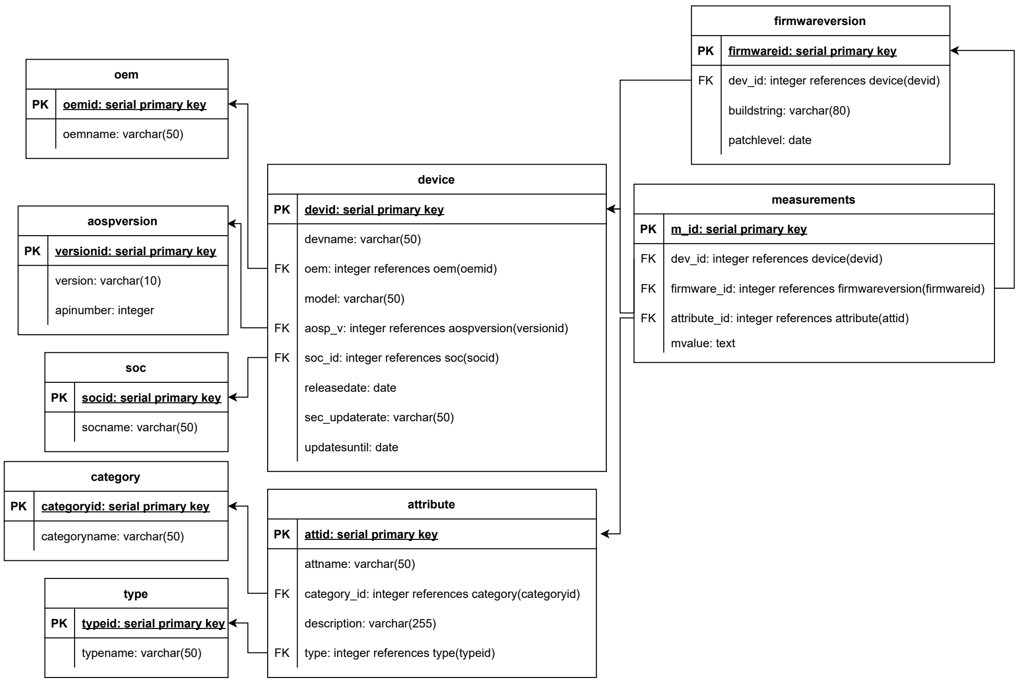

oem
PK oemid: serial primary key
oemname: varchar(50)
aospversion
PK versionid: serial primary key
version: varchar(10)
apinumber: integer
soc
PK socid: serial primary key
socname: varchar(50)
category
PK categoryid: serial primary key
categoryname: varchar(50)
type
PK typeid: serial primary key
typename: varchar(50)
device
PK devid: serial primary key
devname: varchar(50)
FK oem: integer references oem(oemid)
model: varchar(50)
FK aosp_v: integer references aospversion(versionid)
FK soc_id: integer references soc(socid)
releasedate: date
sec_updaterate: varchar(50)
updatesuntil: date
attribute
PK attid: serial primary key
attname: varchar(50)
FK category_id: integer references category(categoryid)
description: varchar(255)
FK type: integer references type(typeid)
firmwareversion
PK firmwareid: serial primary key
FK dev_id: integer references device(devid)
buildstring: varchar(80)
patchlevel: date
measurements
PK m_id: serial primary key
FK dev_id: integer references device(devid)
FK firmware_id: integer references firmwareversion(firmwareid)
FK attribute_id: integer references attribute(attid)
mvalue: text

Figure 3.3: Database scheme.

device 1 id	device 1 data 1	device 1 data 2	attribute 1	attribute 2	...
device 2 id	device 2 data 1	device 2 data 2	attribute 1	attribute 2	...

Figure 3.4: Intended layout of the view.

device 1	attribute 1
device 1	attribute 2
device 2	attribute 1
device 2	attribute 2

Figure 3.5: Original table.

device 1	attribute 1	attribute 2
device 2	attribute 1	attribute 2

Figure 3.6: Pivoted table.

Listing 3.1: Part of the view creation query.

```
...
SELECT * FROM crosstab(
    'SELECT f.dev_id, a.attname, m.mvalue
     FROM firmwareversion as f
     JOIN measurements as m
     ON f.firmwareid = m.firmware_id
     JOIN attribute as a
     ON a.attid = m.attribute_id
     ORDER BY 1',
    $$values ('far'), ('frr'), ('sar'), ('bi'), ('aii'), ('ssti'),
    ('psr'), ('kbb'), ('sba'), ('fbe'), ('mus'), ('sus')$$
)
AS t(devid int,
     far text,
     frr text,
     sar text,
     bi text,
     aii text,
     ssti text,
     psr text,
     kbb text,
     sba text,
     fbe text,
     mus text,
     sus text
   )
...
```

devid	attribute 1	attribute 2
1	0.1	0.2
2		0.3

Figure 3.7: Exemplary query result with correct column definition.

devid	attribute 1	attribute 2
1	0.1	0.2
2	0.3	

Figure 3.8: Exemplary query result with incorrect column definition.

Listing 3.2: Final view query.

```
create view data as
WITH ct AS (-- crosstab query from listing 3.1 --),
device_data as (
    SELECT d.devid, o.oemname, d.model, d.devname, to_char(d.releasedate,
    'YYYY-MM') as releasedate, s.socname, a.version,
    to_char(f.patchlevel, 'YYYY-MM-DD') as patchlevel
    FROM device d
    JOIN firmwareversion f
    ON d.devid = f.dev_id
    JOIN soc s
    ON d.soc_id = s.socid
    JOIN oem o
    ON o.oemid = d.oem
    JOIN aospversion a
    ON d.aosp_v = a.versionid
)
SELECT *
FROM device_data dd
JOIN ct
USING(devid);
```

3.4 API

This section describes the purpose of the REST API endpoints (see routes/api.js).

/totalRecords Since the table is lazy-loaded, this query fetches the total data count for displaying the correct number of pages at the bottom of the table. It is executed once while initializing the page in the `ngOnInit()` method, one of Angular's lifecycle hooks.

/models This query returns all device models for initializing the model drop-down filter after selecting one or more manufacturers. Multiple manufacturers must be separated by a dot.

/attributes All currently available attributes are fetched and used for generating the table header as well as the advanced filters (see section 3.5.3 for details). The response consists of the attribute name (which is actually the abbreviation of the attribute, e.g. 'far'), the description (the full name) and the type of the attribute (here 'boolean', 'numeric' or 'other').

/data/:column This query returns the values for a specific column. It is used for initializing the manufacturer dropdown filter and the non-numeric attribute filters.

/data This is the main query that actually fetches the device and measurement data. First, the basic parameters not related to the data are processed (['page', 'rows', 'sortBy', 'order', 'show']). If some of them are not specified, it will fall back to the default values specified at the top of api.js. Then, a Knex database query builder is created and for every attribute filter, a corresponding 'WHERE' clause depending on the filter's properties is added. The query should return both the total count of the resulting data and the data itself. For this to work, the created query has to be cloned and executed twice: once as a count query and once as a select query.

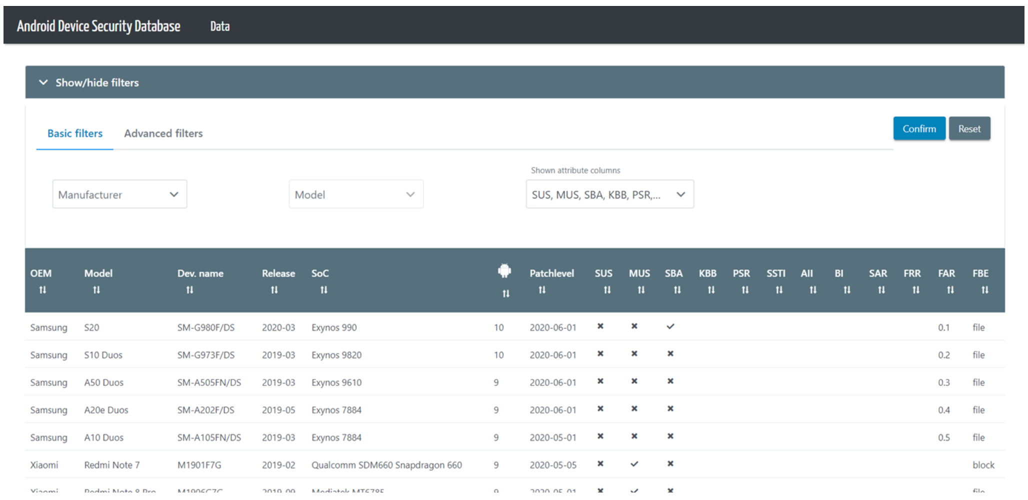
3.5 Front-End

3.5.1 User Interface

Layout The interface is obviously quite simple: a table and a place for the filter functionality. Two variations were tried out:

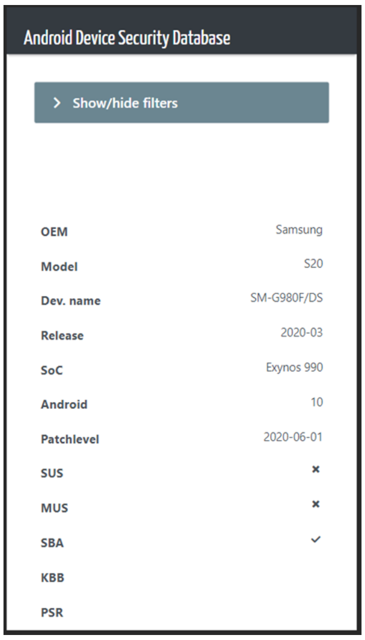
1. Sidebar to the left of the table/on the left page border: Similar to the sidebar on e.g. Amazon, where it works well – but takes up too much space in combination with a table that is likely to grow in width. (see Fig. 3.1 in section 3.1)
2. Filter box above the table: The version that was chosen (see Fig. 3.9), as it mitigates the width problem.

The colors and the navigation bar font were taken from the Android Device Security website. Only sans-serif fonts were chosen, as these tend to generate less "visual noise". For the same reason—and because they do not significantly increase the speed or accuracy when using the table [7]—the often seen "zebra stripes" in tables were omitted. Instead, a highlighting effect when hovering over the rows should help the user navigate and orientate in the table. Icons were kept to a minimum, also to avoid visual clutter: arrows to indicate sortable columns and dropdowns, an Android symbol to save some space in the Android version column as well as crosses and check marks for boolean values. As the names of the attributes tend to be quite long, abbreviations were used for the table header. In the "Advanced filters" tab, the full name is displayed. Still, this



OEM	Model	Dev. name	Release	SoC		Patchlevel	SUS	MUS	SBA	KBB	PSR	SSTI	AII	BI	SAR	FRR	FAR	FBE
Samsung	S20	SM-G980F/DS	2020-03	Exynos 990	10	2020-06-01	✗	✗	✓								0.1	file
Samsung	S10 Duos	SM-G973F/DS	2019-03	Exynos 9820	10	2020-06-01	✗	✗	✗								0.2	file
Samsung	A50 Duos	SM-A505FN/DS	2019-03	Exynos 9610	9	2020-06-01	✗	✗	✗								0.3	file
Samsung	A20e Duos	SM-A202F/DS	2019-05	Exynos 7884	9	2020-06-01	✗	✗	✗								0.4	file
Samsung	A10 Duos	SM-A105FN/DS	2019-03	Exynos 7884	9	2020-05-01	✗	✗	✗								0.5	file
Xiaomi	Redmi Note 7	M1901F7G	2019-02	Qualcomm SDM660 Snapdragon 660	9	2020-05-05	✗	✓	✗									block
Xiaomi	Redmi Note 8 Pro	M1906G7G	2019-09	Mediatek MT6785	9	2020-05-01	✗	✓	✗									file

Figure 3.9: Final interface showing the collapsible filter box and the data table.



OEM	Samsung
Model	S20
Dev. name	SM-G980F/DS
Release	2020-03
SoC	Exynos 990
Android	10
Patchlevel	2020-06-01
SUS	×
MUS	×
SBA	✓
KBB	
PSR	

Figure 3.10: Mobile UI.

leaves some space for improvement: To help with the abbreviations, tool tips for the table header could be implemented to avoid unnecessary tab switching in the filter section. Because the height of the table might require scrolling, especially if the number of shown entries per page is increased, the table header is fixed.

Filter functionality The filters were divided into two groups:

- Basic filters: Filtering by manufacturer and model; also the option to toggle attribute columns.
- Advanced filters: divided into three groups, "boolean" (is attribute X present or not), "numeric" (min-max filters) and "other" (drop-down menu). This filter type is, as mentioned before, a field in the database table of attributes and is used for dynamically generating the UI filter components, in case more attributes are added to the database (see section 3.5.3).

Responsiveness Displaying tabular data on mobile screens poses a particularly difficult challenge and the solution depends on the underlying data and use case. A few possibilities are [16, 19]:

- Downsizing the table by leaving out some data: not applicable here, because all the data should be available on mobile devices.
- Horizontal scrolling: possible, but not optimal since the data is likely to grow in width.
- Flipped table (header as the left fixed column) combined with horizontal scrolling: makes it possible to compare about two entries in this case.
- Transformed: the option that was chosen – the rows are transformed into blocks (see Fig. 3.10) stacked on top of each other, which does not require horizontal scrolling.

3.5.2 Data Service

The data service (`data.service.ts`) is an additional layer between the datatable component and the API, providing functions to fetch and pre-process data. The service is injected into the datatable component using Angular's dependency injection system. Instead of directly making HTTP requests, the datatable component delegates this task to the data service, which separates presentation and data access a bit more.

3.5.3 Datatable Component

The datatable component (`datatable.component.ts`, `datatable.component.html`) contains the filter functionality and the actual table displaying the data. The main challenge was to make the generation of the filters and the table as dynamic as possible, as it was not known how many attributes might be added in the future. To achieve this, the component makes use of Angular directives for dynamic generation of HTML elements.

To aid understanding, this section describes the individual parts of the component and their relationships mostly by example.

Columns

The columns are separated into two categories:

1. `baseCols`, which contain the basic information about a device (manufacturer, model, release date, etc.) and are statically defined in the component (assuming that they will not change much in the future), and
2. `attributes`, which are initialized by fetching the relevant data from the database in the `ngOnInit()` method. This method is an Angular lifecycle hook and should be used for initialization tasks that go beyond setting a few properties (like in a constructor).

Filters

For the `baseCols`, two dropdown filters were implemented: one for selecting a manufacturer and one for selecting a specific phone model (depending on the manufacturer). Additionally—as the table was already quite broad and will probably grow—a menu for selecting the displayed attribute columns was implemented.

The advanced filters are represented by the attributes array in the component, which is initialized in the `ngOnInit()` method. With the `ngFor` structural directive, a set of HTML elements is generated for each attribute. Within that container, a `div` element containing the corresponding form elements is generated using the ngIf directive for each filter type (boolean, numeric, other).

Dropdown filters Listing 3.5 shows the tag for the manufacturer filter. All other dropdown filters work in the same way and will not be explained separately. The `p-multiSelect` tag is a PrimeNG component for creating dropdown menus. It is defined as follows:

Listing 3.3: The surrounding container with the ngFor directive. Also visible here: the usage of ngIf for filtering out columns that should not be displayed.

```
<ng-container *ngFor="let a of attributes">
  <div class="p-col-12 p-lg-3 p-md-6"
    *ngIf="selectedColFields.includes(a.attname)">
  ...
```

Listing 3.4: Part of the div element for the boolean filter.

```
<div *ngIf="a.typename === 'boolean'">
  <p-selectButton ...
```

- The `#dropdownFilter` selector is applied to all dropdown filters and makes it possible to query all elements with that selector from the DOM and programmatically change them. This is used for resetting the filters in the function `resetTable()`.
- Via the `[options]` directive, the options of the dropdown have to be provided as an array. As there are multiple dropdowns in the application, an object named `filterOptions` with the field names as the keys and the options arrays as the values was created in the TypeScript file.
- The option `[filter]` specifies if the dropdown items can be filtered via a search bar—here it was deactivated.
- `(onChange)` calls the function provided to it after the selected value(s) of the dropdown have changed. Here, the function `fetchModels()` is called, which fetches the phone models for the selected manufacturer(s) from the database, provides them to the `filterOptions['model']` array and also activates the (initially disabled) dropdown filter for the models.
- The provided `[(ngModel)]` contains the result: the selected manufacturers. Similar to the `filterOptions` object, a `filterNgModel` object was created, which contains the field names and the respective selected filters. *Two-way data binding* is applied to that object with the `[()]` syntax (“banana in a box”), meaning that it will be updated when changes are applied to the DOM element and, vice-versa, the DOM element will be updated when the underlying model changes. This is especially relevant for sharing views, as the dropdown model will be set based on the values in the URL and not based on user input.

Boolean and numeric filters For the boolean filters, the PrimeNG select button is used, allowing the user to select true, false or both. The usage of the options and two-way data binding is analogous to the dropdown filter(s). The `multiple`option allows selecting multiple values.

Listing 3.5: Manufacturer filter.

```
<p-multiSelect #dropdownFilter id="oemname"
    [options]="filterOptions['oemname']" appendTo="body"
    [filter]="false"
    (onChange)="fetchModels()"
    [(ngModel)]="filterNgModel['oemname']">
</p-multiSelect>
```

Listing 3.6: Boolean filter.

```
<p-selectButton
    [options]="filterOptions[a.attname]"
    multiple="multiple" appendTo="body"
    [(ngModel)]="filterNgModel[a.attname]">
</p-selectButton>
```

Listing 3.7: Numeric filter.

```
<p-inputNumber
    [(ngModel)]="numNgModel['min-' + a.attname]"
    [minFractionDigits]="2"
    placeholder="Min"
    mode="decimal">
</p-inputNumber> -
<p-inputNumber [(ngModel)]="numNgModel['max-' + a.attname]"
    [minFractionDigits]="2"
    placeholder="Max"
    mode="decimal">
</p-inputNumber>
```

For the numeric filters, two form fields (only allowing numerical input) for the minimum and the maximum value are provided. Again, a `ngModel` is used for handling the data input. The option `[minFractionDigits]="2"` makes sure that always at least two fraction digits are displayed. The mode `decimal` defines the type of number input (the other option would be currency). All other filters are represented as a drop-down menu. If further filter types are required, they can be added to the “type” table in the database and then used to create further `div` elements containing the required elements similarly to the existing elements.

Fetching data Instead of loading data on every change, which could be slightly annoying for the user (especially during typing in the numeric filters), the data is only loaded in the function `fetchRows()` after clicking the “Confirm” button.

If the page was not just reloaded (otherwise the ngModels would be empty), the function puts the data from the two ngModels into the query parameters and uses `router.navigate` from the Angular Router to navigate to the URL (staying on the /datatable page) including these parameters. After that, the function `getData()` from the injected data service is called, which queries the data from the API with the provided parameters. `getData()` will return the following (also see section 3.4): the number of total rows and the data for the first page, depending on the selected rows per page.

Sharing views Passing the filter via the URL enables users to share particular views of the table. For this to work, multiple functions are called from `ngOnInit()` that access the query parameters and set the corresponding fields and ngModels based on those parameters.

3.6 Deployment

To set up a CI/CD pipeline for GitLab, the following has to be done: a runner has to be registered for the project, and a `.gitlab-ci.yml` file has to be created

Listing 3.8: .gitlab-ci.yml

```yaml
stages:
  - build

build:
  stage: build
  image: node:12
  script:
    - npm install -g @angular/cli
    - npm install
    - npm run build-prod
  artifacts:
    paths:
    - dist
```

and put into the root location of the repository. This section describes the content of the .yml file that was used in the project. More details about the CI/CD functionality of GitLab can be found in [12]. The meaning of the commands in the file in listing 3.8 is as follows:

- **stages:** With that keyword, groups of tasks can be defined. In this case, only the build stage was defined, but it could, for example, also contain a test or deploy stage.
- The word **build** is the name of the task. The keyword *stage* states to which stage it belongs and *image* defines the Docker image the CI/CD job will run on – in this case, an official Node.js image with version 12 is used.
- The **script** part defines the commands that should be executed after a commit. Here, the NPM commands for installing packages and creating a production build are listed.
- The **artifacts** keyword tells the pipeline which files should be saved and sent to GitLab after finishing the job. Here, the “dist” folder, which contains the final build, is defined as an artifact.

Chapter 4

Conclusion and Outlook

The thesis gave an overview over the three most popular front end frameworks Angular, React and Vue, and demonstrated the implementation of a small web application with Angular. While there are some differences in terms of features, syntax, and structure, overall, all three frameworks would have been suitable for implementing the given application.

Angular on its own proved to be a fine choice, especially with regard to the learning process and the clear structure. However, its "batteries-included" philosophy also means it is shipping tools that are not always needed. For the given implementation task, a much smaller framework or even using Vanilla JS would have been enough.

Additionally, a few other technologies and libraries were used during development, which quickly added up and resulted in a larger application size and a long list of dependencies. The problem were not so much the "primary" dependencies, but the number of libraries that *they* again depended on. The thought that a lot of web applications have a similar number of or even way more dependencies, combined with the security issues of the package manager NPM mentioned in section 3.2, is unsettling, to say the least, and could be compared to a house of cards.

In future work, it would be interesting to see a comparison between the three analysed frameworks and younger, less established ones, especially regarding their concepts and features. The application could, for example, be further improved by refining the UI/UX of the filters (see section 3.5.1) and also by integrating the data table component into the existing Android Device Security website.

Bibliography

[1] Angular. 2020. Angular - Security. Retrieved 03/17/2020 from https://angular.io/guide/security.

[2] Angular. 2021. Angular versioning and releases. Retrieved 12/14/2021 from https://angular.io/guide/releases.

[3] Angular. 2016. chore(release): v2.0.0 proprioception-reinforcement. (September 2016). Retrieved 12/14/2021 from https://github.com/angular/angular/commit/ffe5c49c3ebb51d534a339e0d85a0aa7967923dc.

[4] Catalin Cimpanu. 2018. Hacker backdoors popular JavaScript library to steal Bitcoin funds. *ZDNet*, (November 2018).

[5] 2021. Days Since Last JavaScript Framework. Retrieved 09/07/2021 from https://dayssincelastjavascriptframework.com/.

[6] Nifty Software e.U. 2020. Alibaba - Made with Vue.js. Retrieved 08/07/2020 from https://madewithvuejs.com/alibaba.

[7] Jessica Enders. 2008. Zebra Striping: Does it Really Help? *A List Apart*, (May 2008).

[8] Express.js. 2020. Express.js. Retrieved 03/29/2020 from https://expressjs.com/de/.

[9] José Ignacio Fernández-Villamor, Laura Díaz-Casillas, and Carlos Angel Iglesias. 2008. A comparison model for agile web frameworks. In *Proceedings of the 2008 Euro American conference on Telematics and Information Systems, EATIS 2008, Aracaju, Brazil, September 10-12, 2008*. DOI: 10.1145/1621087.1621101.

[10] David Gilbertson. 2018. I'm harvesting credit card numbers and passwords from your site. Here's how. *Hackernoon*, (January 2018).

[11] Andreas Gizas, Sotiris P. Christodoulou, and Theodore S. Papatheodorou. 2012. Comparative evaluation of javascript frameworks. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*. ACM, pp. 513–514. DOI: 10.1145/2187980.2188103.

[12] GitLab Inc. 2020. GitLab CI/CD. Retrieved 09/04/2020 from https://docs.gitlab.com/ee/ci/.

[13] O'Reilly Media Inc. 2020. O'Reilly - Search. (April 2020). Retrieved 04/04/2020 from https://www.oreilly.com/search.

[14] PrimeTek Informatics. 2020. PrimeNG - The Most Powerful Angular UI Component Library. Retrieved 03/29/2020 from https://www.primefaces.org/primeng/.

[15] PrimeTek Informatics. 2020. PrimeNG Showcase. Retrieved 03/29/2020 from https://www.primefaces.org/primeng/showcase/.

[16] Michał Jarosz. 2018. 5 Practical Solutions to Make Responsive Data Tables. *Appnroll Publication*, (September 2018).

[17] Patricia Johnson. 2019. 4 Steps Developers Should Take To Use npm Securely. (December 2019). Retrieved 03/29/2020 from https://www.whitesourcesoftware.com/resources/blog/npm-security/.

[18] Knex.js. 2020. Knex.js. Retrieved 03/29/2020 from https://knexjs.org/.

[19] James Knutila. 2014. Web Archive - Responsive Tables: Best Practices and Examples. *DevKit (Web Archive)*, (April 2014).

[20] Rafał Kostrzewski and Matt Warcholinski. 2021. 10 Famous Apps Using ReactJS Nowadays. *Brainhub*, (March 2021).

[21] Stefan Krause. 2020. js-framework-benchmark. (February 2020). Retrieved 02/02/2020 from https://github.com/krausest/js-framework-benchmark.

[22] Dewi Mairiza, Didar Zowghi, and Nur Nurmuliani. 2010. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*. ACM, pp. 311–317. DOI: 10.1145/1774088.1774153.

[23] Node.js. 2020. Node.js. Retrieved 03/29/2020 from https://nodejs.org/en/.

[24] NPM. 2017. 'crossenv' malware on the npm registry. (August 2017). Retrieved 03/29/2020 from https://blog.npmjs.org/post/163723642530/crossenv-malware-on-the-npm-registry.html.

[25] NPM. 2020. NPM. Retrieved 03/29/2020 from https://www.npmjs.com/.

[26] NPM. 2020. package-lock.json - A manifestation of the manifest. Retrieved 03/29/2020 from https://docs.npmjs.com/cli/v7/configuring-npm/package-lock-json.

[27] Lalith Polepeddi. 2019. Made with Angular. (September 2019). Retrieved 08/07/2020 from https://www.madewithangular.com/categories/angular/.

[28] PostgreSQL. 2020. PostgreSQL. Retrieved 03/29/2020 from https://www.postgresql.org/about/.

[29] PostgreSQL. 2020. tablefunc. Retrieved 03/29/2020 from https://www.postgresql.org/docs/9.2/tablefunc.html.

[30] G. Prasad. 2014. 100+ JavaScript Frameworks for Web Developers. *CSS Author*, (June 2014).

[31] React. 2020. How to Contribute - React. Retrieved 03/17/2020 from https://reactjs.org/docs/how-to-contribute.html#security-bugs.

[32] React. 2021. React blog. Retrieved 12/14/2021 from https://reactjs.org/blog/all.html/.

[33] React. 2021. Versioning Policy - React. Retrieved 12/14/2021 from https://reactjs.org/docs/faq-versioning.html.

[34] RisingStack. 2021. The History of React.js on a Timeline. (October 2021). Retrieved 12/14/2021 from https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/.

[35] Ryadel. 2019. YARN vs NPM (vs pnpm) in 2019: comparison and verdict. (July 2019). Retrieved 03/29/2020 from https://www.ryadel.com/en/yarn-vs-npm-pnpm-2019/.

[36] Jacob Schatz. 2017. How we do Vue: one year later. *GitLab Blog*, (November 2017).

[37] Tony Chao Shan and Winnie W. Hua. 2006. Taxonomy of Java Web Application Frameworks. In *2006 IEEE International Conference on e-Business Engineering (ICEBE 2006), 24-26 October 2006, Shanghai, China*. IEEE Computer Society, pp. 378–385. DOI: 10.1109/ICEBE.2006.98.

[38] Nikita Skovoroda. 2018. Gathering weak npm credentials. (May 2018). Retrieved 03/29/2020 from https://github.com/ChALkeR/notes/commit/5b867f10302a677e63ac31ec37515c6d732ab3ad.

[39] Ayrton Sparling. 2018. I don't know what to say. (November 2018). Retrieved 03/29/2020 from https://github.com/dominictarr/event-stream/issues/116.

[40] TasteJS. 2015. Yet Another Framework Syndrome (YAFS). (January 2015). Retrieved 09/07/2021 from https://medium.com/tastejs-blog/yet-another-framework-syndrome-yafs-cf5f694ee070.

[41] VueJS. 2014. Releases - vuejs/vue. (February 2014). Retrieved 12/14/2021 from https://github.com/vuejs/vue/releases?page=20.

[42] VueJS. 2021. Roadmap for the Vue.js project. Retrieved 12/14/2021 from https://github.com/vuejs/roadmap#release-management.

[43] VueJS. 2020. Security | Vue.js. Retrieved 03/17/2020 from https://v3.vuejs.org/guide/security.html#reporting-vulnerabilities.

[44] VueJS. 2021. Sponsor Vue.js Development. Retrieved 10/13/2021 from https://vuejs.org/support-vuejs/.

[45] VueJS. 2021. Vue.js 3.2.0 - GitHub Project. Retrieved 12/14/2021 from https://github.com/vuejs/core/projects/4.