Todo list

DOCUMENTATION REPORT
Part of the BACHELOR DISSERTATION

Next-Gen Web Solutions
A comprehensive analysis of enterprise-focused web solutions, unveiling strengths and weaknesses.

Bachelor	Applied Computer Science
Elective track Graduation	Software Engineer
Academic year	2023 - 2024
Student	Arthur De Witte
Internal coach	Mattias De Wael *(HOWEST)*
External promoter	Tom Stijnen *(H.Essers)*

howest.be

Availability for consultation

The author(s) gives (give) permission to make this documentation report (part of the bachelor dissertation) available for consultation and to copy parts of this report for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this report.

May 10, 2024

Foreword

> write the foreword

The foreword contains the usual thanks. All those who helped with the final paper are thanked. The persons who made the most significant contribution are thanked first. Write the name, position and title of persons correctly. Indicate your name, place and date at the bottom (optional). A signature is not appropriate here. Because the word in advance is strongly personal, it is often written in the I form.

Contents

CHAPTER 1

Introduction

1.1 General

In web development, there are numerous solutions for creating websites. These solutions streamline website creation by simplifying complex tasks, standardizing and abstracting away common tasks, increasing DX and ultimately UX.

The Paradox of Choice

The abundance of web solutions/frameworks presents a challenge known as the paradox of choice [1]. This challenge is particularly pertinent for enterprises, which face additional criteria such as release cycles, licensing, support, state management complexities, backing, and longevity — all crucial factors considering the long-term maintenance requirements of the software.

Common Challenge

Whether you're a *(frontend)* Software Engineer, Project Manager, Technical Architect, Startup Founder, or simply someone intrigued by the web, you've likely encountered this dilemma.

Collaborative Project

This dissertation is a collaboration with the IT department of H. Essers, a major European Transport and Logistics company headquartered in Genk, Belgium. Their objective is to develop custom software solutions to optimize business processes. The department mainly comprises Java and IBM AS/400 *(now called "IBM i" [2])* developers. They utilize the Vaadin full-stack framework for Java. Despite its advantages, the team has faced limitations within Vaadin that require more effort than initially anticipated.

1.2 The problem

User interfaces are crucial components of any application, and although websites have been around for many years, the industry is constantly evolving. Nowadays, development teams have a variety of web frameworks, architectures, and principles to choose from, making it difficult to decide which one is the best for a given task. This research aims to provide an answer to this difficult question.

1.3 Research question

What is the most suitable web solution for what kind of application?

1.3.1 Web Solutions

Research has been conducted into:

Solution	Year Released	Version Reviewed
React	2022	`18.2.0`
Vue	2024	`3.4.15`
Svelte	2024	`4.2.12`
Angular	2024	`17.3.0`
Lit	2024	`3.1.2`
Hilla	2024	`2.5.7`

Table 1.1: Researched Solutions

1.4 Experiment

The same project was built in each solution to ensure equal and objective evaluation.

1.4.1 Project Requirements

These requirements will provide valuable insights. The assessment is conducted objectively using the details stated in 1.4.2.

- General Layout *(see Figure B.1)*

- Interactive Search *(with URL query reflection) (see Figure B.2)*

- *(data)* Grid *(see Figure B.3)*

- *(data)* Grid in *(data)* Grid *(see Figure B.4)*

- Normal Forms *(with validation) (see Figure B.5 and Figure B.6)*

- Wizard Forms *(see Figure B.7 and Figure B.8)*

- Internationalization

- Drag and Drop *(see Figure B.9)*

- Progressive Loading

- Global State Management and Reactions

- Reflective Routing *(see Figure B.10, Figure B.11, Figure B.12, Figure B.13, and Figure B.14)*

1.4.2 Evaluation

Because the research question is broad, it will be answered by dividing it into smaller evaluation points, which are ranked objectively using a suitable method.

- Community [1]

- Professional Support [2]

- Documentation *(interactive?)* [3]

- Ecosystem [4]

- Usage by other enterprises [3]

- *(added)* Size of solution [5]

- Relative performance of solution [3]

- Complexity [6]

- Server Side Rendered (SSR) [7, 8]

Likert Scale

Some evaluation will be done using a Likert Scale [3], with the values:

$$\text{BAD/NOT PRESENT} < \text{MEDIUM/OK} < \text{GREAT}.$$

---

[1] Points by size, see A.1

[2] Predicate

[3] Likert Scale

[4] Points are 70% by quality and 30% by size, see A.2.3 and A.2.2 respectively

[5] Likert Scale, but smaller is better

[6] Evaluated using the Likert Scale by easiness/speed to learn, state management, boilerplate, and API integration

[7] Likert Scale, MEDIUM/OK being available through a well-supported and known extension

[8] SSR is better for SEO dependent applications

CHAPTER 2

Experiment

2.1 Angular

2.1.1 Overview

Angular, a web framework developed by Google, is esteemed for its flexibility and extensive toolset. This section delves into its primary features and considerations. With numerous pre-built dependencies, Angular streamlines development and reduces setup time. Notably, it supports Server Side Rendered (SSR) [4], expanding deployment possibilities. The framework boasts comprehensive documentation, catering to developers of all skill levels. Leveraging standard TypeScript and HTML files, Angular seamlessly integrates a custom templating language into HTML [5].

Styling and Project Setup

Angular's CSS scoping mechanism ensures clarity in styling scope, thus enhancing code maintainability [6]. Additionally, its CLI application accelerates project setup by efficiently generating boilerplate code [7].

Challenges and Considerations

Angular faces challenges, particularly in state management. It provides two methods for managing state: through two-way bindings and signals, which can be confusing and are not interoperable [8], [9].

While Angular offers a powerful templating language, it may fall short in certain areas compared to other solutions. For instance, passing arguments to slots/content projection can be less intuitive for new developers, potentially increasing the learning curve [10]. Moreover, Angular's approach to internationalization requires separate builds for each language, limiting flexibility during development since it lacks built-in support for language switching [11].

Angular entails a significant amount of boilerplate, primarily due to legacy considerations [12]. Moreover, it lacks consistency in utilizing type systems, necessitating explicit declarations for certain features like input validation [13], [14].

2.1.2 Update Process

Angular boasts a reliable updating process, supported by detailed migration guides for smooth transitions across major versions. Major updates occur every six months,

typically accompanied by 1 to 3 minor releases. Furthermore, patch releases are rolled out nearly every week, enhancing the platform's stability and functionality [15].

2.1.3   Unique Features

Angular employs CSS selectors instead of tags, granting developers more control over component behavior and implementations [16]. Additionally, it implements dependency injection, a beneficial design pattern [17], [18].

2.1.4   Strengths

- no VDOM overhead

- great documentation

- standard typescript and HTML files (custom templating language in HTML)

- CSS scope can be chosen with ease

- big community *(over 40,000 members on Discord)*

- CLI application to help generate boilerplate

- stable standardized approach/libraries for many features

- uses CSS selectors instead of tags for components, providing more flexibility to the developer component usage

- baked in SSR support

- you can easily set the scope for styles

- stable and consistent release process [15]

- lots of enterprise usage

- consistent and stable updating process

2.1.5   Weaknesses

- uses different terminology than all other frameworks which makes switching more difficult

- HMR doesn't work great/smoothly out of the box

- you are unable to change the language for I18N in development mode as it requires a separate build for each language

- state management is complex *(subsection 2.1.1)*

- requires a lot of boilerplate

- templating language is less powerful compared to other solutions *(subsection 2.1.1)*

- requires explicit declarations that could be derived from TypeScript

2.1.6 Scores

Method	Score
Easiness/speed to learn	0.5
State management	0
Boilerplate	0 *(components: 0, state management: 0)*
API integration	0.5 *(has custom HTTP handler)*

Table 2.1: complexity

angular: relative performance

angular: calculate added size

Method	Score
Community	0.6 *(Table A.1)*
Professional Support	1
Documentation (interactive walkthrough?)	1
Ecosystem	0.81 *(quality: 0.82, size: 0.8; Table A.6 Table A.3)*
Usage by other enterprises	1
(added) Size of solution	-
Relative performance of solution	-
Complexity	0.25 *(Table 2.1)*
Server Side Rendered (SSR)	1

Table 2.2: Angular general scores

2.2 Lit

2.2.1 Overview

Lit is a library developed by Google on the browser native web components. This makes it a lightweight solution without VDOM that can be used with any web solution, even with plain HTML. It also does not require much boilerplate and uses plain HTML in combination with JS or TS. Out of the box it provides reactive state, scoped styles and a declarative templating system. [19]

It's main goal is to develop sharable components or design systems for its cross solution usage ability. It can also be used to progressively enhance basic HTML sites.

Challenges and considerations

Due to web components not being as popular as other solutions, lit does not enjoy a big ecosystem. The documentation is also mediocre and has room for improvement [20]. On top of that does it not currently have the ability to do SSR, and makes use of the Shadow Dom *(see 2.2.2)*. IDE integration is also suboptimal providing inaccurate linting and suggestions. [21]–[23]

2.2.2 Shadow DOM

READ article https://www.matuzo.at/blog/2023/pros-and-cons-of-shadow-dom

bla bla [22]

2.2.3 Strengths

- strength! ——————————————————— write strengths

2.2.4 Weaknesses

- weakness! :(——————————————— write weaknesses

2.2.5 Scores

lit: fill in all

Method	Score
Easiness/speed to learn	0.5
State management	0
Boilerplate	0 *(components: 0, state management: 0)*
API integration	0.5 *(has custom HTTP handler)*

Table 2.3: complexity

lit: do all

lit: relative performance

lit: calculate added size

Method	Score
Community	0.6 *(Table A.1)*
Professional Support	1
Documentation (interactive walkthrough?)	1
Ecosystem	0.81 *(quality: 0.82, size: 0.8;* **??** *Table A.3)*
Usage by other enterprises	1
(added) Size of solution	-
Relative performance of solution	-
Complexity	0.25 *(Table 2.3)*
Server Side Rendered (SSR)	1

Table 2.4: lit general scores

2.3 React

2.3.1 Overview

React is a library [24] created by Meta *(originally known as Facebook)*. Its recommended usage is in combination with JSX [25] *(see 2.3.2)*. The library is

primarily intended for the render layer and does not include native support for features such as routing and I18N. However, this does not mean that you cannot easily incorporate these features, as the library is part of a vast community ecosystem that includes many high-quality packages that specialize in various areas.

React uses a different approach than plain JavaScript by utilizing the VDOM [26] *(see 2.3.4)* to manage content instead of directly manipulating the DOM. It attempts to detect changes using the reconciliation algorithm *(see 2.3.5)* in the browser at runtime [27].

2.3.2   JSX

The rendering logic often gets tightly coupled with other UI logic. Instead of separating things by putting markup and logic in separate files, we can achieve our separation of concerns [28] by creating loosely coupled units called *components*. These components should ideally be pure, making the logic predictable, testable, and allowing us to make render optimizations like memoization [29], [30].

We can use JavaScript XML (JSX), which is neither JS nor a string. Instead, it combines *(as the name implies)* XML/HTML syntax with JS capabilities *(demonstrated in Listing 2.1)*. We can easily create components by defining a method that returns JSX, essentially currying [31] its context *(arguments, state, and more; demonstrated in Listing 2.2)*.

```
1  const example = "NGWS";
2  const element = <p>This variable is interpolated {example}</p>
```
Listing 2.1: Simple JSX interpolation [25], [32]

```
1  function AnswerToTheUniverse() {
2      const theAnswer = 42;
3      return <p>The answer to the universe is: {theAnser}</p>;
4  }
```
Listing 2.2: Simple JSX component

2.3.3   Update Process

The update process of React is highly reliable. It involves testing the entire React-meta codebase, which comprises over 50,000 components, to determine if deprecating a method requires many changes. Only after this testing, does the React team decide if deprecation is necessary. If it is, they release a warning to the open-source community, which remains for one version. After that, the deprecated item is completely removed. In case many changes are needed to address the deprecation warning, scripts are built to make the migration as automatic as possible [33].

2.3.4   Virtual DOM

The Virtual DOM (VDOM) is a mirrored version of the real DOM. Represented as in-memory objects *(eg. Listing 2.3)* which can easily be traversed *(as no DOM needs to be parsed)*, checked for changes, and used for other optimizations. For example, if a type of a VDOM element is changed it will tear down the old tree and rebuild the tree from scratch, but if the type is the same it will only update the attributes.

Or if a key is set the reconciler can easily detect what items need to update *(2.3.5, [27], [34])*.

Although the VDOM incurs more overhead as the browser has to keep the entirety in memory, it offers greater flexibility for the reconciler. For instance, the React reconciler can not only process the DOM but also native iOS and Android displays *(with React Native)* [34]. Additionally, the VDOM enables more unique optimizations such as the pull technique instead of push, which allows the prioritization of user interactions over background tasks [33]. Moreover, it allows the renders to be batched instead of each one being its own operation.

```
1  {
2    type: "button",
3    props: {
4      className: "button button-blue",
5      children: {
6        type: "b",
7        props: {
8          children: "OK!"
9        }
10     }
11   }
12 }
```

Listing 2.3: JSON representation of VDOM element [35], [36]

```
1  <button class="button button-blue">
2      <b>OK!</b>
3  </button>
```

Listing 2.4: HTML equivalent of Listing 2.3

2.3.5 Reconciliation Algorithm

There are generic solutions to the algorithmic problem of diffing and transforming one tree into another. However, the existing algorithms are expensive at $O(n^3)$ [37]. Because this is too expensive for a web framework, the react reconciler implements a $O(n)$ algorithm based on two assumptions [27].

1. "Two elements of different types will produce different tries." which is why if the type is different the tear will be torn down.

2. "The developer can hint at which child elements may be stable across different renders with a *key* prop."

2.3.6 State Management

State management in React relies on hooks, which are specialized functions. These hooks serve specific purposes within React components. Unlike traditional JavaScript assignments, manipulating state in React requires the use of hooks and their associated methods. Additionally, React enforces immutability, meaning once state is set, it cannot be directly changed. This immutability adds complexity to learning React, as it imposes restrictions on how actions can be performed.

When a state is updated in React, the reconciler detects these changes and initiates a refresh of the entire component in the next tick. To optimize performance and avoid unnecessary refreshes, it is recommended to:

- Split components into smaller, more manageable pieces.

- Minimize side effects within components.

- Utilize memoization where applicable.

By following these practices, developers can ensure efficient state management and enhance the performance of React applications [29].

Sharing State

In React, there are various methods for sharing state, with one popular approach being to lift the state up *(Figure 2.2)* [38]. This method aligns with the principle of a "*single source of truth*" [39], meaning that while the state isn't confined to just one place, there's a central component responsible for managing it. This eliminates the need for duplicating state across components, thus reducing error-prone practices.

However, this approach has its drawbacks. For nested components, the state must be passed down through each level, leading to code bloat and unnecessary dependencies between components. To address this issue, a context provider can be employed. This provider enables all nested children, regardless of depth, to access and respond to the state without requiring explicit prop drilling or predefined component structures *(Figure 2.3)* [40].
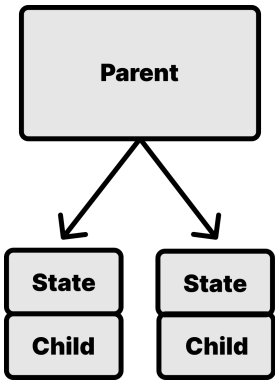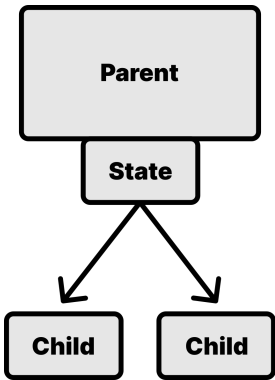
Figure 2.1:   per
component state

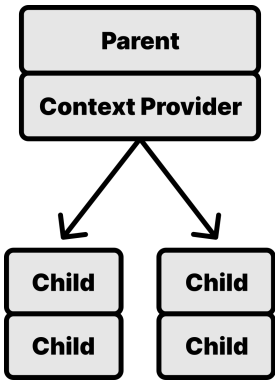Figure 2.2: state lifted
up *(shared state)*

Figure 2.3:   context
provides state to all

2.3.7   Redux

Incorporating context to manage state is useful, but it's typically confined to the parent component. In most projects, developers opt to place the global state in the root *(app)* component *(Figure 2.4)*. While this approach suffices for a few states, it becomes unwieldy when multiple states need to be shared.

To mitigate this challenge, developers often turn to Redux, a state management library. Redux operates akin to a global context, with each state, termed a "store", linked directly to the Redux provider *(Figure 2.5)* [41]. This setup ensures that each state maintains its own logic and adheres to consistent, standardized definitions.
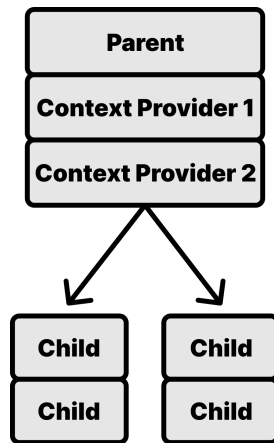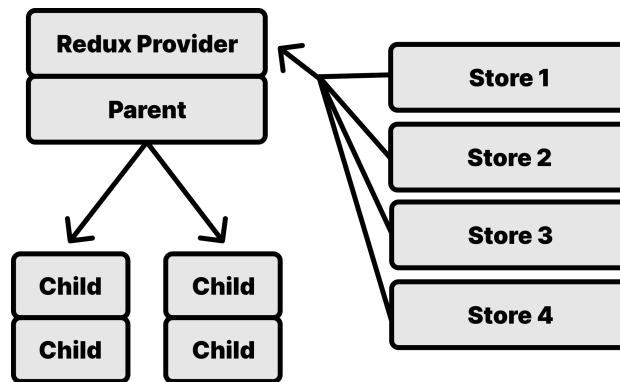
Figure 2.4: several providers provide state to all

Figure 2.5: several Redux stores provide state to all

2.3.8   Strengths

- JSX is intuitive and easy to write

- big ecosystem

- well-known and used among developers

- documentation is great *(includes interactive examples)*

- big community *(over 230,000 members on Discord)*

- lots of enterprise usage, even in the Fortune 500 companies [42]

- not that performant compared to native JS, but a one-click addon called *Million* [43] makes this problem go away

- professional support is widely available through 3rd parties

2.3.9   Weaknesses

- VDOM overhead

- Component tries can cause unexpected behavior

- client-side rendered *(solution: Next.js or canary "use server" directive [44])*

- State management can easily become complex in bigger components.

- basic dependencies *(React, Redux, I18N)* that get sent to the client are about 12MB which is relatively big and can slow down the initial load

2.3.10   Scores

Method	Score
Easiness/speed to learn	1
State management	0.5
Boilerplate	0.5 *(components: 1, state management: 0)*
API integration	0 *(default browser fetch API)*

Table 2.5: complexity

> react: relative performance

> react: calculate added size

Method	Score
Community	1 *(Table A.1)*
Professional Support	1
Documentation (interactive walkthrough?)	0.5
Ecosystem	0.94 *(quality: 0.88, size: 1; Table A.5 Table A.3)*
Usage by other enterprises	1
(added) Size of solution	-
Relative performance of solution	-
Complexity	0.5 *(Table 2.5)*
Server Side Rendered (SSR)	0.5

Table 2.6: React general scores

2.4   Hilla

2.4.1   Overview

> write this

2.4.2   Strengths

> write strengths

- strength!

2.4.3   Weaknesses

> write weaknesses

- weakness! :(

2.5   Svelte

2.5.1   Overview

> write this

2.5.2 Strengths

- strength! _____ **write strengths**

2.5.3 Weaknesses

- weakness! :(_____ **write weaknesses**

2.6 Vue

2.6.1 Overview

write this

2.6.2 Strengths

- strength! _____ **write strengths**

2.6.3 Weaknesses

- weakness! :(_____ **write weaknesses**

CHAPTER 3

Conclusion

conclusion and recommendation

Here you formulate the answer to the research question. This does not include any new results that you have not previously cited. Do not use subsections here. Conclude your conclusion with a powerful closing sentence that briefly summarises your conclusion in one sentence.

Acronyms

API Application Programming Interface. 6, 7, 12

CLI Command Line Interface. 4, 5

CSS Cascading Style Sheets. 5

DOM document object model. 8, 9, 15

DX developer experience. 1, 16

HMR Hot Module Replacement [45]. 5

HTML Hypertext Markup Language. 4–6, 8, 9

I18N internationalisation. 5, 8, 11

IDE Integrated Development Environment. 6

JS JavaScript. 6, 8

JSON JavaScript Object Notation. 9

JSX JavaScript XML. 7, 8, 11

SEO Search Engine Optimalisation. 3

SSR Server Side Rendered. 3–6

TS TypeScript. 6

UI user interface. 8

UX user experience. 1, 16

VDOM Virtual DOM. 5, 6, 8, 9, 11

XML Extensible Markup Language. 8, 15

Glossary

boolean a value which is either true or false *(0 or 1)*. 16

developer experience refers to the overall quality of interaction and satisfaction developers encounter while using tools, libraries, frameworks, or platforms to build software solutions. It encompasses various aspects such as ease of use, clarity of documentation, efficiency of workflows, availability of support, and the overall enjoyment of the development process. A positive DX contributes to increased productivity, reduced frustration, and greater motivation among developers, ultimately creating higher-quality software products. 15

document object model "connects web pages to scripts or programming languages by representing the structure of a document—such as the HTML representing a web page—in memory" [46]. 15

full-stack Encompasses the complete spectrum of web development, including both frontend and backend components, typically unified within a single codebase and language. 1

internationalisation is the process of designing applications to support multiple languages and cultures without altering the core code. It involves enabling language support, considering regional formats, supporting diverse character sets, managing content for localization, and adapting the user interface. Techniques include using resource bundles, language negotiation, dynamic content rendering, and ensuring Unicode compliance. Internationalization expands reach, improves user experience, and ensures regulatory compliance, making applications accessible to diverse global audiences. 15

library a collection of pre-existing code that can be utilized to create new code [47]. 7

predicate something that results in a boolean. 3, 23

user experience user experience refers to how people feel and interact with a product, service, or system. It encompasses aspects such as ease of use, visual appeal, and the emotions it evokes. A positive UX keeps people satisfied, interested, and returning for more. 15

AI Engineering Prompts

Note: While LLMs are excellent, they are not flawless, so the prompts have been used as a guide rather than a definitive one. There has been no copy-pasting, but rather a rewriting process to match the AI's output.

3.1 Rewriting of text

Role: English Language Expert
Language: English
Context: Computer Science Thesis
Task: Rewrite the following text slightly according to the notes
Notes:
- Clarity: Ensure that the main ideas are clearly expressed and easy to understand.
- Simplicity: Use plain language and avoid jargon or complex terminology whenever possible.
- Conciseness: Trim unnecessary words or phrases to make the text more concise and to the point.
- Structure: Organize the text logically with clear headings, subheadings, and transitions between paragraphs.
- Consistency: Maintain consistent formatting, tone, and style throughout the text.
- Active Voice: Use active voice to make sentences more direct and engaging.
- Variety: Vary sentence length and structure to keep the reader's attention and avoid monotony.
- Clarity of Purpose: Ensure that each section or paragraph serves a clear purpose and contributes to
the overall message.
- Audience Awareness: Consider the needs and knowledge level of the target audience when
choosing language and examples.
- Visual Elements: Incorporate bullet points, lists, and visuals where appropriate to break up dense
text and improve readability.
- Transition Words: Use transition words and phrases to guide the reader through the text and
connect ideas smoothly.
- Contextualization: Provide context or background information where necessary to help readers
understand unfamiliar concepts or terms.
- Parallelism: Use parallel structure for lists or series of items to improve clarity and readability.
- Avoidance of Ambiguity: Clarify ambiguous terms or phrases to prevent confusion or
misinterpretation.
- Proofreading: Correct any grammatical errors, typos, or inconsistencies.
- Output: The text is formatted using LaTeX, if there are any terms or acronyms that should be
defined, define them and put them at the top separated from the text.
- Do not include LaTeX document boilerplate
Text:
<text here>

Bibliography

[1]   The Decision Lab, *The Paradox of Choice*, `https://thedecisionlab.com/reference-guide/economics/the-paradox-of-choice`, [Online; accessed 17-April-2024].

[2]   Wikipedia contributors, *IBM i — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=IBM_i&oldid=1210410964`, [Online; accessed 17-April-2024], 2024.

[3]   Wikipedia contributors, *Likert scale — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Likert_scale&oldid=1212329755`, [Online; accessed 22-April-2024], 2024.

[4]   Angular, *Server-side rendering*, `https://angular.dev/guide/ssr`, [Online; accessed 08-May-2024].

[5]   Angular, *Template Syntax*, `https://angular.dev/guide/templates`, [Online; accessed 08-May-2024].

[6]   Angular, *Styling components*, `https://angular.dev/guide/components/styling`, [Online; accessed 08-May-2024].

[7]   Angular, *The Angular CLI*, `https://angular.dev/tools/cli`, [Online; accessed 08-May-2024].

[8]   Angular, *Managing Dynamic Data*, `https://angular.dev/essentials/managing-dynamic-data`, [Online; accessed 08-May-2024].

[9]   Angular, *Signals*, `https://angular.dev/guide/signals`, [Online; accessed 08-May-2024].

[10] Angular, *ng-container*, `https://angular.dev/api/core/ng-container`, [Online; accessed 08-May-2024].

[11] Angular, *Internationalization*, `https://angular.dev/guide/i18n`, [Online; accessed 08-May-2024].

[12] Angular, *Directive composition API*, `https://angular.dev/guide/directives/directive-composition-api`, [Online; accessed 08-May-2024].

[13] Angular, *Accepting data with input properties*, `https://angular.dev/guide/components/inputs`, [Online; accessed 08-May-2024].

[14] TypeScript Language, *Object Types*, `https://www.typescriptlang.org/docs/handbook/2/objects.html`, [Online; accessed 08-May-2024].

[15] Angular, *Versioning and releases*, `https://angular.dev/reference/releases`, [Online; accessed 08-May-2024].

[16] Angular, *Component selectors*, `https://angular.dev/guide/components/selectors`, [Online; accessed 08-May-2024].

[17] Stackify, *Dependency Injection*, `https://stackify.com/dependency-injection/`, [Online; accessed 08-May-2024].

[18] Angular, *Dependency Injection*, `https://angular.dev/guide/di`, [Online; accessed 08-May-2024].

[19] Lit, *What is Lit*, `https://lit.dev/docs/`, [Online; accessed 10-May-2024].

[20] Arthurdw - Github, *Context not being passed to parent slot consumer*, `https://github.com/lit/lit/issues/4618`, [Online; accessed 10-May-2024].

[21] Lit, *Server-side rendering (SSR)*, `https://lit.dev/docs/ssr/overview/`, [Online; accessed 10-May-2024].

[22] DesignSystemCentral, *Why Web Components Aren't the Design System Silver Bullet (Just Yet)*, `https://designsystemcentral.com/why-web-components-arent-the-design-system-silver-bullet-just-yet/`, [Online; accessed 10-May-2024].

[23] Mitosis, *Overview*, `https://mitosis.builder.io/docs/overview/`, [Online; accessed 10-May-2024].

[24] Ritu Lagad, *5 Big Limitations of React*, `https://medium.com/@LagadRitu/5-big-limitations-of-react-e6c0a54fedd0`, [Online; accessed 4-March-2024], 2023.

[25] Meta, *Introducing JSX - React*, `https://legacy.reactjs.org/docs/introducing-jsx.html`, [Online; accessed 23-April-2024].

[26] Meta, *Virtual DOM and Internals - React*, `https://legacy.reactjs.org/docs/faq-internals.html`, [Online; accessed 4-March-2024].

[27] Meta, *Reconciliation - React*, `https://legacy.reactjs.org/docs/reconciliation.html`, [Online; accessed 4-March-2024].

[28] Wikipedia contributors, *Separation of concerns — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=1214678067`, [Online; accessed 23-April-2024], 2024.

[29] Wikipedia contributors, *Pure function — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Pure_function&oldid=1215011412`, [Online; accessed 23-April-2024], 2024.

[30] Meta, *Keeping components Pure - React*, `https://react.dev/learn/keeping-components-pure`, [Online; accessed 23-April-2024].

[31] Wikipedia contributors, *Currying — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Currying&oldid=1214432439`, [Online; accessed 23-April-2024], 2024.

[32] Meta, *JSX*, `https://facebook.github.io/jsx/`, [Online; accessed 4-March-2024].

[33] Meta, *Design Principles - React*, `https://legacy.reactjs.org/docs/design-principles.html`, [Online; accessed 4-March-2024].

[34] Andrew Clark, *React Fiber ARchitecture*, `https://github.com/acdlite/`
      `react-fiber-architecture`, [Online; accessed 4-March-2024].

[35] Meta, *React Commponents, Elements and Instances - React*, `https://`
      `legacy.reactjs.org/blog/2015/12/18/react-components-elements-`
      `and-instances.html`, [Online; accessed 4-March-2024].

[36] Meta, *React - Basic Theoretical Concepts*, `https://github.com/reactjs/`
      `react-basic`, [Online; accessed 4-March-2024].

[37] P. Bille, "A survey on tree edit distance and related problems," *Theoretical
      computer science*, vol. 337, no. 1-3, pp. 217–239, 2005.

[38] Meta, *Sharing State Between Components - React*, `https://react.dev/`
      `learn/sharing-state-between-components`, [Online; accessed 5-March-
      2024].

[39] Wikipedia contributors, *Single source of truth — Wikipedia, the free
      encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Single_`
      `source_of_truth&oldid=1213115955`, [Online; accessed 24-April-2024],
      2024.

[40] Meta, *Passing Data Deeply with Context - React*, `https://react.dev/learn/`
      `passing-data-deeply-with-context`, [Online; accessed 24-April-2024].

[41] Redux, *Redux Fundamentals, Part 1: Redux Overview*, `https://redux.js.`
      `org/tutorials/fundamentals/part-1-overview`, [Online; accessed 24-
      April-2024].

[42] Built In, *Explore Top Tech Companies*, `https://builtin.com/companies/`
      `tech/react-companies`, [Online; accessed 24-April-2024].

[43] Million, *Million Beyond 'Speed'*, `https://million.dev/blog/million-`
      `beyond-speed`, [Online; accessed 24-April-2024].

[44] Meta, *'use server' directive - React*, `https://react.dev/reference/react/`
      `use-server`, [Online; accessed 24-April-2024].

[45] Webpack, *Hot Module Replacement*, `https://webpack.js.org/concepts/`
      `hot-module-replacement/`, [Online; accessed 06-May-2024].

[46] Mozilla Developer Network, *Document Object Model (DOM)*, `https://`
      `developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model`,
      [Online; accessed 22-April-2024].

[47] Codeacademy Team, *What is a Framework?* `https://www.codecademy.com/`
      `resources/blog/what-is-a-framework/`, [Online; accessed 22-April-2024].

APPENDIX   A

Metrics

A.1   Community Size Rating

Range	Score
< 500	0
< 1,000	0.1
< 3,000	0.2
< 6,000	0.3
< 12,000	0.4
< 25,000	0.5
< 50,000	0.6
< 100,000	0.7
< 150,000	0.8
< 200,000	0.9
> 200,000	1

Table A.1: scores for size of community based on amount of Discord members

A.2 Ecosystem Ratings

A.2.1 GitHub Stars

Range	Score
< 500	0
< 1,000	0.1
< 3,000	0.2
< 6,000	0.3
< 12,000	0.4
< 25,000	0.5
< 50,000	0.6
< 100,000	0.7
< 150,000	0.8
< 200,000	0.9
> 200,000	1

Table A.2: scores for amount of stars

A.2.2 Size

The size is based on the amount of GitHub results for the solution name.

Range	Score
< 10,000	0
< 20,000	0.1
< 50,000	0.2
< 100,000	0.3
< 250,000	0.4
< 500,000	0.5
< 750,000	0.6
< 1,000,000	0.7
< 2,500,000	0.8
< 5,000,000	0.9
> 5,000,000	1

Table A.3: scores for size

A.2.3 Quality

Quality is determined by stars, documentation, and testing coverage. It is assumed that libraries with many users/stars and great documentation coverage have high quality. For feasibility reasons, this is checked for the top 25 downloaded dependencies.

Symbol	Note
*	predicate
-	not applicable
+	refers to the scores from Table A.2

Table A.4: special symbols

React

nr	Tag	Stars$^+$	Documentation*	Tests*
1	facebook/react	1	yes	yes
2	vercel/next.js	0.8	yes	yes
3	facebook/react-native	0.8	yes	yes
4	facebook/create-react-app	0.8	yes	yes
5	mui/material-ui	0.7	yes	yes
6	ant-design/ant-design	0.7	yes	yes
7	storybookjs/storybook	0.7	yes	yes
8	enaqx/awesome-react	0.7	yes	-
9	leonardomso/33-js-concepts	0.7	yes	-
10	shadcn-ui/ui	0.7	yes	yes
11	gatsbyjs/gatsby	0.7	yes	yes
12	facebook/docusaurus	0.7	yes	yes
13	remix-run/react-router	0.7	yes	yes
14	ionic-team/ionic-framework	0.7	yes	yes
15	meteor/meteor	0.6	yes	yes
16	pmndrs/zustand	0.6	yes	yes
17	appwrite/appwrite	0.6	yes	yes
18	streamich/react-use	0.6	yes	yes
19	brillout/awesome-react-components	0.6	yes	-
20	styled-components/styled-components	0.6	yes	yes
21	TanStack/query	0.6	yes	yes
22	react-hook-form/react-hook-form	0.6	yes	yes
23	chakra-ui/chakra-ui	0.6	yes	no
24	preactjs/preact	0.6	yes	yes
25	jaredpalmer/formik	0.6	yes	yes
-	**Average**	**0.68**	**1**	**0.956**

Table A.5: React ecosystem ratings - 24/04/2024

Score: $\frac{0.68+1+0.956}{3} = 0.878666\cdots \approx 0.88$

Angular

nr	Tag	Stars[+]	Documentation[*]	Tests[*]
1	angular/angular	0.8	yes	yes
2	storybookjs/storybook	0.7	yes	yes
3	ionic-team/ionic-framework	0.7	yes	yes
4	tastejs/todomvc	0.6	yes	yes
5	angular/angular-cli	0.6	yes	yes
6	akveo/ngx-admin	0.6	yes	yes
7	angular/components	0.5	yes	yes
8	johnpapa/angular-styleguide	0.5	yes	-
9	nrwl/nx	0.5	yes	yes
10	pubkey/rxdb	0.5	yes	yes
11	handsontable/handsontable	0.5	yes	yes
12	teambit/bit	0.5	yes	yes
13	paperless-ngx/paperless-ngx	0.5	yes	yes
14	sweetalert2/sweetalert2	0.5	yes	yes
15	tinymce/tinymce	0.5	yes	yes
16	Chocobozzz/PeerTube	0.5	yes	yes
17	ionic-team/stencil	0.5	yes	yes
18	linnovate/mean	0.5	yes	yes
19	ag-grid/ag-grid	0.4	yes	yes
20	BuilderIO/mitosis	0.4	yes	yes
21	PatrickJS/PatrickJS-starter	0.4	yes	no
22	primefaces/primeng	0.4	yes	yes
23	PatrickJS/awesome-angular	0.4	yes	-
24	openreplay/openreplay	0.4	yes	yes
25	NG-ZORRO/ng-zorro-antd	0.4	yes	yes
-	**Average**	**0.512**	**1**	**0.957**

Table A.6: Angular ecosystem ratings - 10/05/2024

Score: $\frac{0.512+1+0.957}{3} = 0.823 \approx 0.82$

APPENDIX   B

Project Requirements
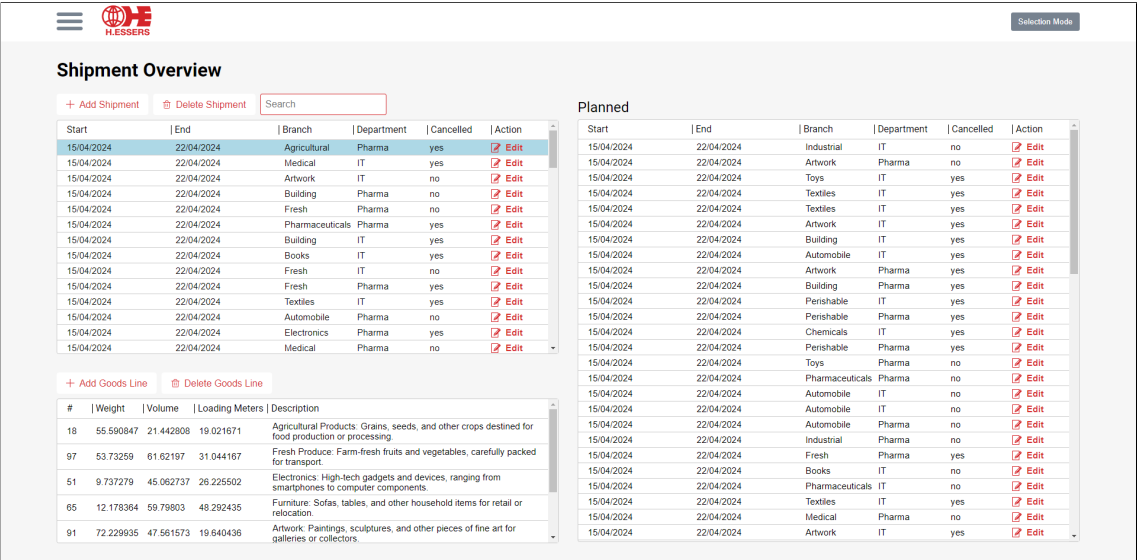
Figure B.1: general layout - `https://link.arthurdw.com/ngws-layout`
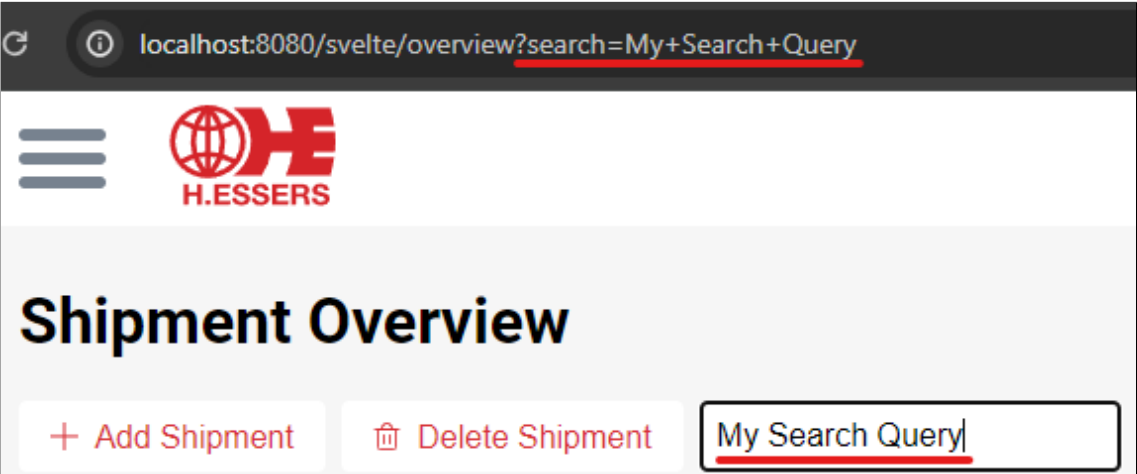
Figure B.2: interactive search - `https://link.arthurdw.com/ngws-search`

Start		End		Branch		Department		Cancelled		Action
15/04/2024		22/04/2024		Agricultural		Pharma		yes		✏ Edit
15/04/2024		22/04/2024		Medical		IT		yes		✏ Edit
15/04/2024		22/04/2024		Artwork		IT		no		✏ Edit
15/04/2024		22/04/2024		Building		Pharma		no		✏ Edit
15/04/2024		22/04/2024		Fresh		Pharma		no		✏ Edit
15/04/2024		22/04/2024		Pharmaceuticals		Pharma		yes		✏ Edit
15/04/2024		22/04/2024		Building		IT		yes		✏ Edit
15/04/2024		22/04/2024		Books		IT		yes		✏ Edit
15/04/2024		22/04/2024		Fresh		IT		no		✏ Edit
15/04/2024		22/04/2024		Fresh		Pharma		yes		✏ Edit
15/04/2024		22/04/2024		Textiles		IT		yes		✏ Edit
15/04/2024		22/04/2024		Automobile		Pharma		no		✏ Edit
15/04/2024		22/04/2024		Electronics		Pharma		yes		✏ Edit
15/04/2024		22/04/2024		Medical		Pharma		no		✏ Edit

Figure B.3: grid - `https://link.arthurdw.com/ngws-grid`

Start		End		Branch		Department		Cancelled		Action
15/04/2024		22/04/2024		Industrial		IT		no		✏ Edit

#		Weight		Volume		Loading Meters		Description
65		27.126638		64.966286		31.183498		Books and Printed Material: Novels, textbooks, and periodicals bound for libraries or retailers.
79		2.9077778		96.06974		25.74276		Chemicals: Various substances used in manufacturing, agriculture, or research.

Start		End		Branch		Department		Cancelled		Action
15/04/2024		22/04/2024		Artwork		Pharma		no		✏ Edit
15/04/2024		22/04/2024		Toys		IT		yes		✏ Edit
15/04/2024		22/04/2024		Textiles		IT		yes		✏ Edit
15/04/2024		22/04/2024		Textiles		IT		yes		✏ Edit

#		Weight		Volume		Loading Meters		Description
40		79.72203		54.74606		48.205166		Fresh Produce: Farm-fresh fruits and vegetables, carefully packed for transport.
54		91.31288		21.084251		45.297527		Books and Printed Material: Novels, textbooks, and periodicals bound for libraries or retailers.
18		34.621483		87.52795		31.122179		Agricultural Products: Grains, seeds, and other crops destined for food production or processing.

Start		End		Branch		Department		Cancelled		Action
15/04/2024		22/04/2024		Artwork		IT		yes		✏ Edit
15/04/2024		22/04/2024		Building		IT		yes		✏ Edit

#		Weight		Volume		Loading Meters		Description
47		42.280685		33.08796		39.832283		Medical Supplies: Life-saving equipment, pharmaceuticals, and surgical instruments.
54		50.115967		94.481834		2.2499146		Books and Printed Material: Novels, textbooks, and periodicals bound for libraries or retailers.
81		49.358315		30.164183		8.355484		Agricultural Products: Grains, seeds, and other crops destined for food production or processing.
6		90.32398		97.41022		13.202429		Chemicals: Various substances used in manufacturing, agriculture, or research.
								Automobile Parts: Components such as engines, tires, and body

Figure B.4: grid in grid - `https://link.arthurdw.com/ngws-grid_in_grid`

Figure B.5: form: `https://link.arthurdw.com/ngws-form_1`

Figure B.6: form with grid - `https://link.arthurdw.com/ngws-form_2`

Figure B.7: wizard form: `https://link.arthurdw.com/ngws-wizard_1`

Figure B.8: wizard form with grid - `https://link.arthurdw.com/ngws-wizard_2`

Figure B.9: drag and drop - `https://link.arthurdw.com/ngws-dnd`

Figure B.10: reflective routing - `https://link.arthurdw.com/ngws-routing_1`

Figure B.11: reflective routing - `https://link.arthurdw.com/ngws-routing_2`

Figure B.12: reflective routing - `https://link.arthurdw.com/ngws-routing_3`

Figure B.13: reflective routing - `https://link.arthurdw.com/ngws-routing_4`

Figure B.14: reflective routing - `https://link.arthurdw.com/ngws-routing_5`