# 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.

April 24, 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 | 2016 | `18.2.0` |
| Vue | 2014 | `3.4.15` |
| Svelte | 2016 | `4.2.12` |
| Angular | 2016 | `17.3.0` |
| Lit | 2019 | `3.1.2` |
| Hilla | 2022 | `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.2.2 and A.2.3 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 React

### 2.1.1 Overview

React is a library [4] created by Meta *(originally known as Facebook)*. Its recommended usage is in combination with JSX [5] *(see 2.1.2)*. 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 [6] *(see 2.1.4)* to manage content instead of directly manipulating the DOM. It attempts to detect changes using the reconciliation algorithm *(see 2.1.5)* in the browser at runtime [7].

### 2.1.2 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 [8] 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 [9], [10].

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 [11] 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 [5], [12]

```
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

### 2.1.3 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 [13].

### 2.1.4 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 *(2.1.5, [7], [14])*.

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)* [14]. 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 [13]. Moreover, it allows the renders to be batched instead of each one being its own operation.

```
{
  type: "button",
  props: {
    className: "button button-blue",
    children: {
      type: "b",
      props: {
        children: "OK!"
      }
    }
  }
}
```

Listing 2.3: JSON representation of VDOM element [15], [16]

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

Listing 2.4: HTML equivalent of Listing 2.3

### 2.1.5 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 $O(n^3)$ [17]. Because this is too expensive for a web framework, the react reconciler implements a $O(n)$ algorithm based on two assumptions [7].

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."

### 2.1.6 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 [9].

#### Sharing State

In React, there are various methods for sharing state, with one popular approach being to lift the state up *(Figure 2.2)* [18]. This method aligns with the principle of a "*single source of truth*" [19], 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)* [20].

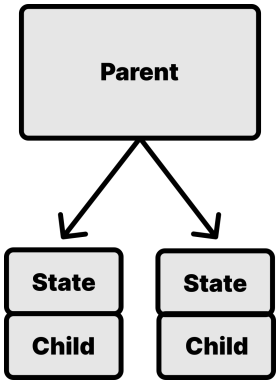


Figure 2.1: per component state

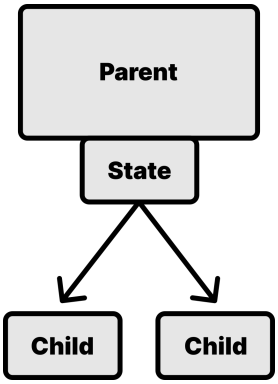


Figure 2.2: state lifted up *(shared state)*

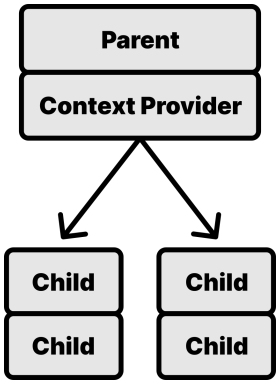


Figure 2.3: context provides state to all

### 2.1.7 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)* [21]. This setup ensures that each state maintains its own logic and adheres to consistent, standardized definitions.

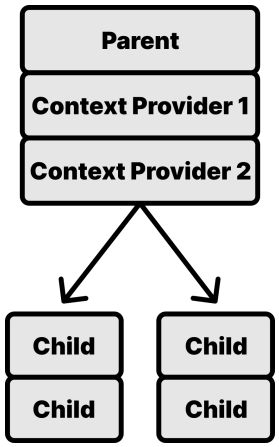


Figure 2.4: serveral providers provide state to all

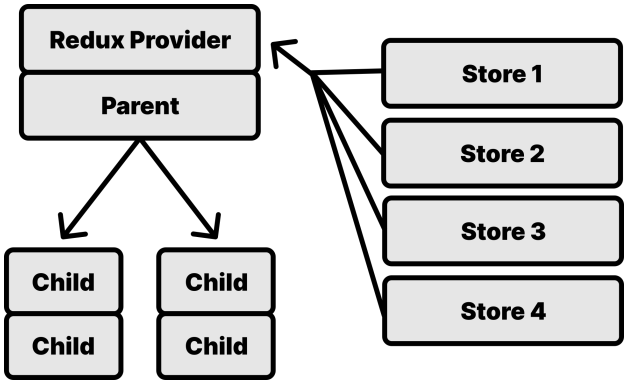


Figure 2.5: several Redux stores provide state to all

### 2.1.8 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 [22]
- not that performant compared to native JS, but a one-click addon called *Million* [23] makes this problem go away
- professional support is widely available on top of the community support

### 2.1.9 Weaknesses

- VDOM overhead
- client-side rendered *(solution: Next.js or canary "use server" directive [24])*
- State management can be complex. Even if you are familiar with bigger components, managing their state can become difficult quickly.
- 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

### 2.1.10 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 2.1: complexity

relative performance

| **Method** | **Score** |
|---|---|
| Community | 1 *(Table A.1)* |
| Professional Support | 1 |
| Documentation (interactive?) | 0.5 |
| Ecosystem | 0.9 *(Table A.4)* |
| Usage by other enterprises | 1 |
| *(added)* Size of solution | 0.5 |
| Relative performance of solution | - |
| Complexity | 0.5 *(Table 2.1)* |
| Server Side Rendered (SSR) | 0.5 |

Table 2.2: React general scores

## 2.2 Vue

### 2.2.1 Overview

write this

### 2.2.2 Strengths

- strength!

write strengths

### 2.2.3 Weaknesses

- weakness! :(

write weaknesses

This uses the VDOM!

Estimated outline

1. What will we build and how?
2. React short overview (what, by who, how, strengths & weaknesses)
3. Explaining the VDOM
4. React Reconciliation Algorithm
5. Vue short overview (what, by who, how, strengths & weaknesses)
6. Svelte short overview (what, by who, how, strengths & weaknesses)
7. Angular short overview (what, by who, how, strengths & weaknesses)
8. Hilla short overview (what, by who, how, strengths & weaknesses)
9. Do we even need a framework?
10. Don't write JS/TS solutions (HTMX, Hyperscript, ...)
11. For enterprises (release cycles, stability, support, licenses, scalability, ...)

better title for this

Write the experiment

In this part of your bachelor's dissertation, you describe your experiment. Or put another way, how did you gain knowledge to give your TEDTalk/final presentation. This is the body of your report, where you make the difference compared to existing literature. Also make sure it is clear what you have made/added compared to existing material. Make sure the following issues are covered throughout the different chapters:

- the methodology
- justified choices of technology/software/procedure etc.,
- results,

- critical analysis of the results.

The above items are not literal chapters but should be interwoven throughout the body of your documentation report. Make sufficient use of figures and visualisations (do not forget citations). Make sure you have a nice coherent whole, with clear structure and a smooth readability. Don't be afraid to use AI tools for this. Do not forget to include your AI prompts at the back of this report. Too many details that would distract from the story of your bachelor's dissertation should be relegated to an appendix. (E.g. installation procedure, pieces of code.)

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.

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

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

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

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

[8] 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.

[9] 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.

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

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

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

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

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

[15] 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].

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

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

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

[19] 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.

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

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

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

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

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

APPENDIX A

# Metrics

## A.1 Community Size Rating

The largest statistic gets chosen out of: GitHub stars or Discord members.

| **Range** | **Score** |
|---|---|
| $< 500$ | 0 |
| $< 1{,}000$ | 1 |
| $< 3{,}000$ | 2 |
| $< 6{,}000$ | 3 |
| $< 12{,}000$ | 4 |
| $< 25{,}000$ | 5 |
| $< 50{,}000$ | 6 |
| $< 100{,}000$ | 7 |
| $< 150{,}000$ | 8 |
| $< 200{,}000$ | 9 |
| $> 200{,}000$ | 10 |

Table A.1: scores for size of community

## A.2 Ecosystem Ratings

### A.2.1 GitHub Stars

| Range | Score |
|---|---|
| $< 500$ | 0 |
| $< 1{,}000$ | 1 |
| $< 3{,}000$ | 2 |
| $< 6{,}000$ | 3 |
| $< 12{,}000$ | 4 |
| $< 25{,}000$ | 5 |
| $< 50{,}000$ | 6 |
| $< 100{,}000$ | 7 |
| $< 150{,}000$ | 8 |
| $< 200{,}000$ | 9 |
| $> 200{,}000$ | 10 |

Table A.2: scores for amount of stars

### A.2.2 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.3: special symbols

**React**

| nr | **Tag** | **Stars**$^{+}$ | **Documentation**$^{*}$ | **Tests**$^{*}$ |
|---|---|---|---|---|
| 1 | facebook/react | 10 | yes | yes |
| 2 | vercel/next.js | 8 | yes | yes |
| 3 | facebook/react-native | 8 | yes | yes |
| 4 | facebook/create-react-app | 8 | yes | yes |
| 5 | mui/material-ui | 7 | yes | yes |
| 6 | ant-design/ant-design | 7 | yes | yes |
| 7 | storybookjs/storybook | 7 | yes | yes |
| 8 | enaqx/awesome-react | 7 | yes | - |
| 9 | leonardomso/33-js-concepts | 7 | yes | - |
| 10 | shadcn-ui/ui | 7 | yes | yes |
| 11 | gatsbyjs/gatsby | 7 | yes | yes |
| 12 | facebook/docusaurus | 7 | yes | yes |
| 13 | remix-run/react-router | 7 | yes | yes |
| 14 | ionic-team/ionic-framework | 7 | yes | yes |
| 15 | meteor/meteor | 6 | yes | yes |
| 16 | pmndrs/zustand | 6 | yes | yes |
| 17 | appwrite/appwrite | 6 | yes | yes |
| 18 | streamich/react-use | 6 | yes | yes |
| 19 | brillout/awesome-react-components | 6 | yes | - |
| 20 | styled-components/styled-components | 6 | yes | yes |
| 21 | TanStack/query | 6 | yes | yes |
| 22 | react-hook-form/react-hook-form | 6 | yes | yes |
| 23 | chakra-ui/chakra-ui | 6 | yes | no |
| 24 | preactjs/preact | 6 | yes | yes |
| 25 | jaredpalmer/formik | 6 | yes | yes |
| - | **Average** | **6.8** | **1** | **0.956** |

Table A.4: react ecosystem ratings

Score: $(6.8/10 + 1 + 0.956)/3 = 0.878666\cdots \approx 0.9$

### A.2.3 Size

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

| **Range** | **Score** |
|---|---|
| $< 10{,}000$ | 0 |
| $< 20{,}000$ | 1 |
| $< 50{,}000$ | 2 |
| $< 100{,}000$ | 3 |
| $< 250{,}000$ | 4 |
| $< 500{,}000$ | 5 |
| $< 750{,}000$ | 6 |
| $< 1{,}000{,}000$ | 7 |
| $< 2{,}500{,}000$ | 8 |
| $< 5{,}000{,}000$ | 9 |
| $> 5{,}000{,}000$ | 10 |

Table A.5: scores for size

APPENDIX B

# Project Requirements

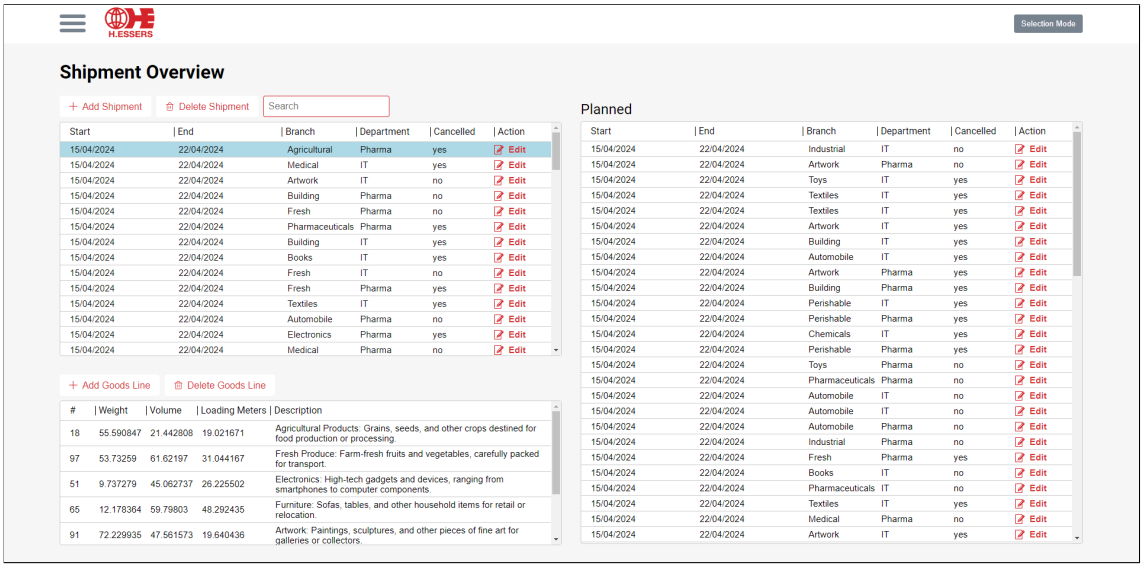


Figure B.1: general layout - `https://link.arthurdw.com/ngws-layout`

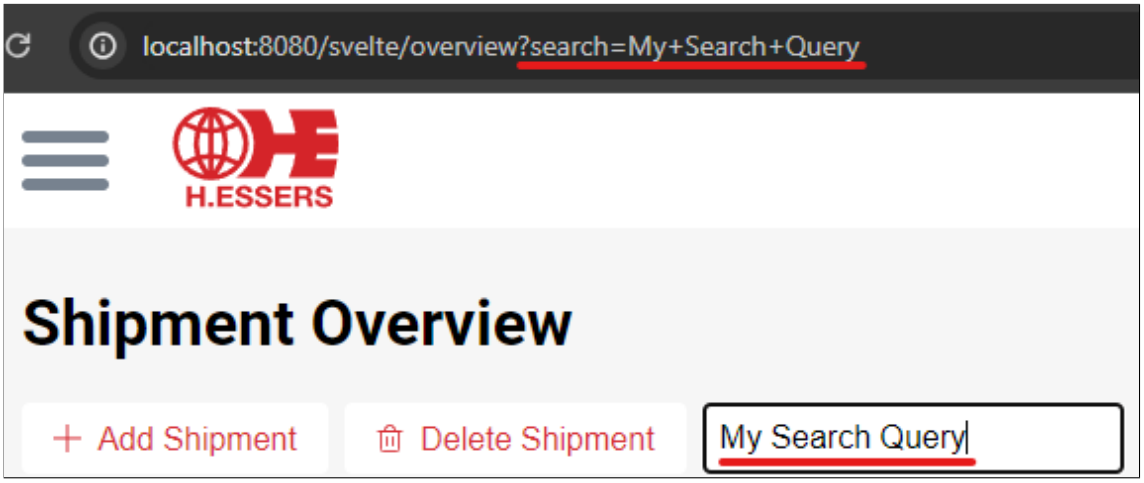


Figure B.2: interactive search - `https://link.arthurdw.com/ngws-search`

| 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. | |
| 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. | |
| 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`

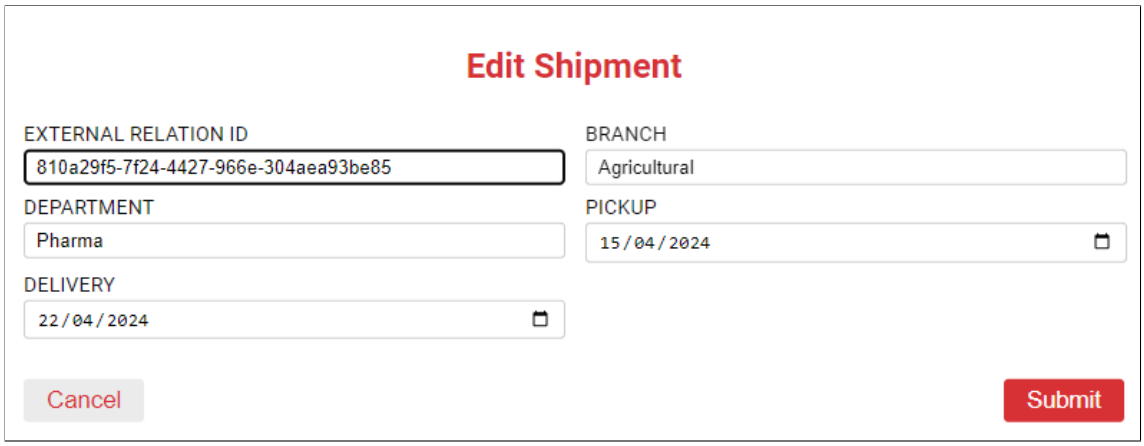


Figure B.5: form: `https://link.arthurdw.com/ngws-form_1`

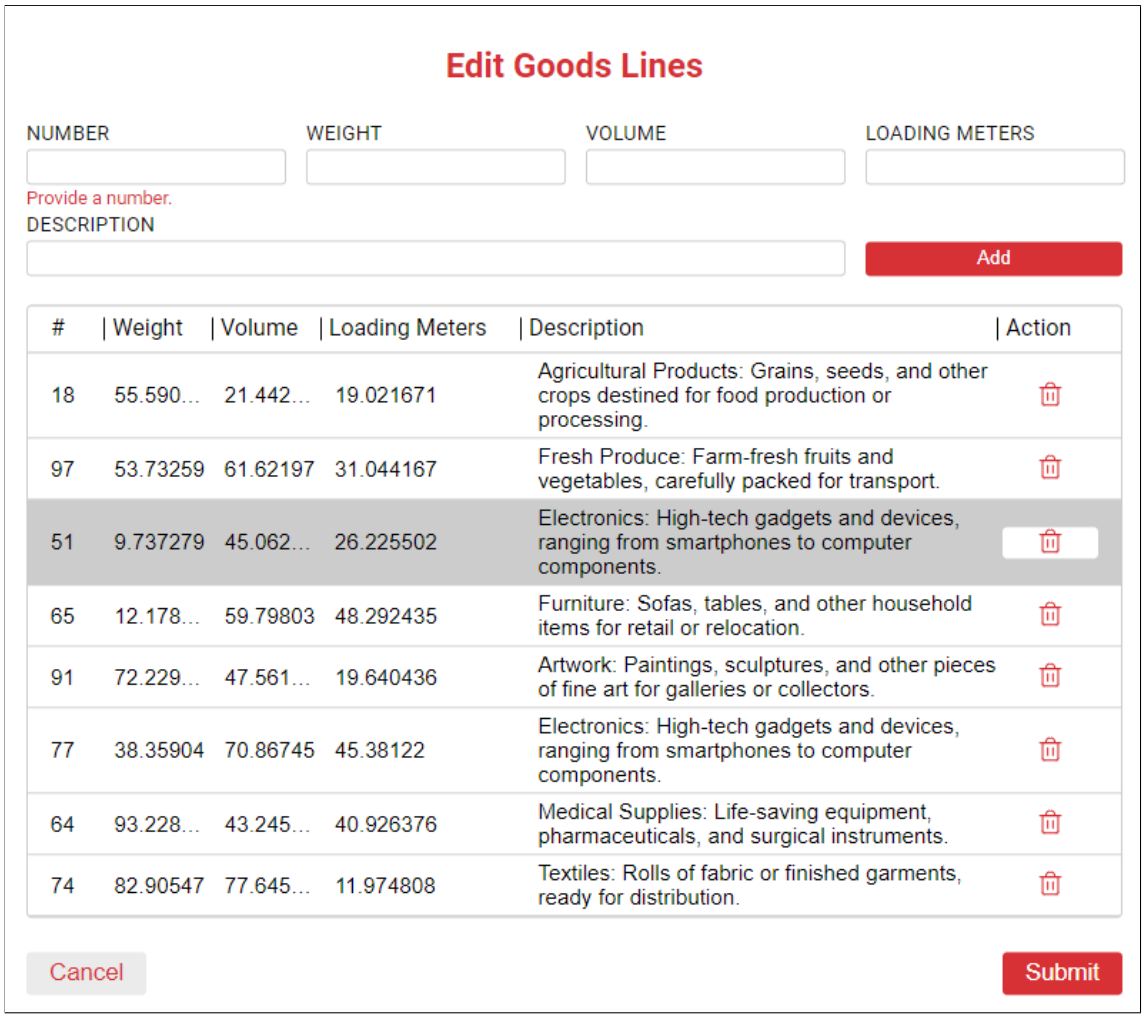


Figure B.6: form with grid - `https://link.arthurdw.com/ngws-form_2`

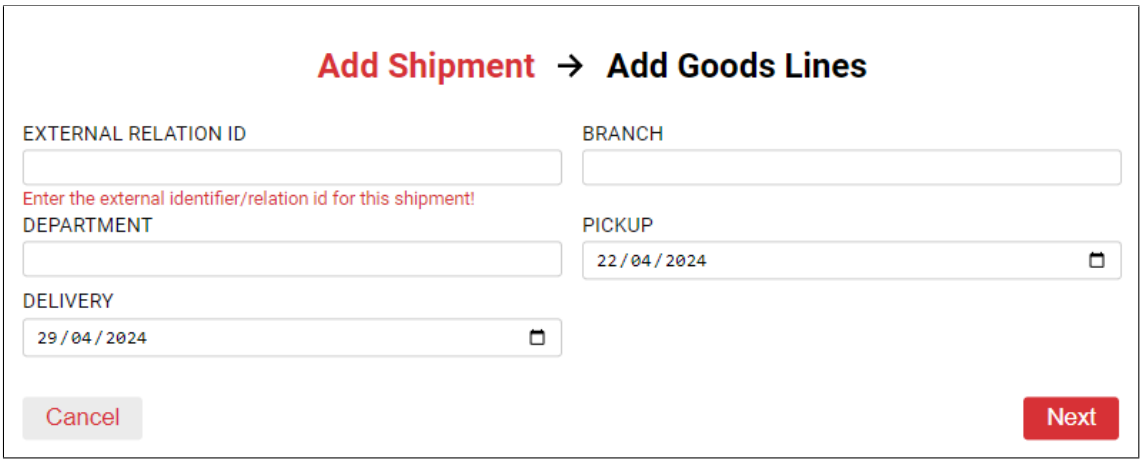


Figure B.7: wizard form: `https://link.arthurdw.com/ngws-wizard_1`

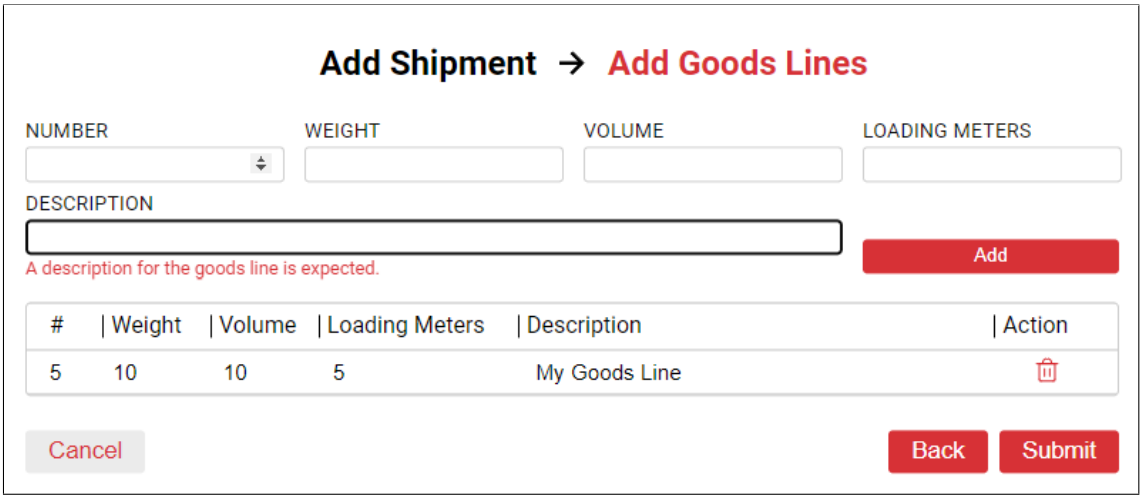


Figure B.8: wizard form with grid - `https://link.arthurdw.com/ngws-wizard_2`

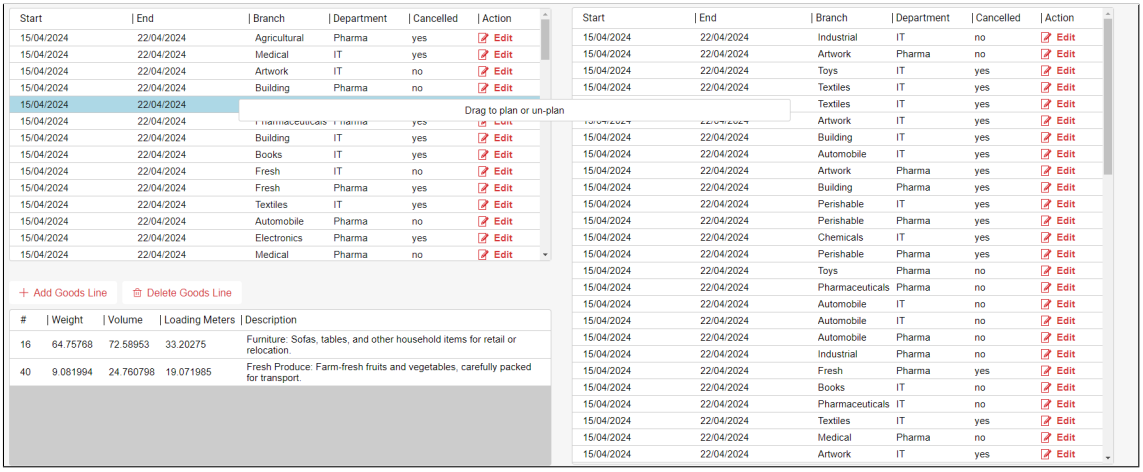


Figure B.9: drag and drop - `https://link.arthurdw.com/ngws-dnd`

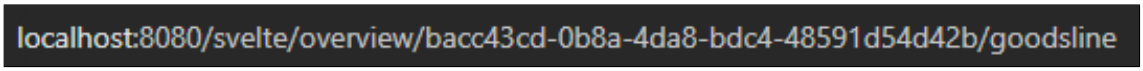


Figure B.10: reflective routing - `https://link.arthurdw.com/ngws-routing_1`


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


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

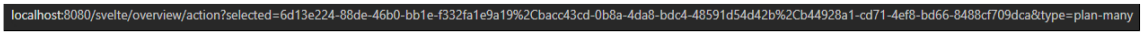


Figure B.13: reflective routing - `https://link.arthurdw.com/ngws-routing_4`

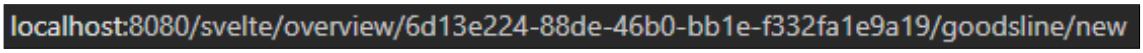


Figure B.14: reflective routing - `https://link.arthurdw.com/ngws-routing_5`