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 25, 2024

Foreword

As I reflect on the culmination of my internship at H.Essers and the journey towards completing this dissertation, I am filled with a profound sense of accomplishment and gratitude. Throughout my internship, I had the privilege to explore the dynamic world of next-generation web solutions, delving into innovative technologies and strategies aimed at enhancing the digital landscape.

The inspiration for this dissertation stemmed from my fascination with the ever-evolving nature of web technologies and their profound impact on developers, businesses, and society. Guided by H.Essers' vision as a frontrunner in logistics and transport solutions, I embarked on a journey to investigate how next-generation web solutions could revolutionize the industry, optimizing processes and enhancing efficiency.

I extend my heartfelt gratitude to my mentor, Tom Stijnen, and colleagues at H.Essers, whose expertise and guidance were invaluable in shaping the direction of this research. I would like to specifically thank Aaron Lardenoije for providing me with great insights and assisting me when I was stuck.

Furthermore, I am indebted to Gert Bervoets and Thierry De Block for granting me the opportunity to undertake this internship, and to the entire team at H.Essers for their unwavering support and collaboration. The collaborative environment within the organization fueled my curiosity, enabling me to engage in meaningful discussions, exchange ideas, and leverage collective expertise.

This dissertation would not have been possible without the support of my academic advisor, Mattias De Wael, who helped me refine my work.

As I conclude this journey, I am confident that the findings presented in this dissertation will contribute to the ongoing discourse surrounding next-generation web solutions, particularly in non-SSR environments. It is my hope that this research will serve as a catalyst for further innovation and inspire future endeavors in the field.

I am deeply grateful for the opportunity to undertake this research and for the invaluable experiences gained during my internship at H.Essers. I extend my sincere appreciation to everyone who has supported and contributed to this endeavor.

Thank you for your interest in this dissertation. I hope the insights presented herein will inspire continued exploration and innovation in the realm of next-generation web solutions.

Arthur De Witte
24/05/2024

Contents

Foreword **V**

1 Introduction **1**
1.1 General . . . 1
1.2 The problem . . . 1
1.3 Research question . . . 2
1.3.1 Web Solutions . . . 2
1.4 Experiment . . . 2
1.4.1 Project Requirements . . . 2
1.4.2 Evaluation . . . 3

2 Experiment **4**
2.1 Angular . . . 4
2.1.1 Overview . . . 4
2.1.2 Update Process . . . 4
2.1.3 Unique Features . . . 5
2.1.4 Strengths . . . 5
2.1.5 Weaknesses . . . 5
2.1.6 Scores . . . 6
2.2 Lit . . . 6
2.2.1 Overview . . . 6
2.2.2 Shadow DOM . . . 6
2.2.3 Strengths . . . 7
2.2.4 Weaknesses . . . 7
2.2.5 Scores . . . 7
2.3 React . . . 8
2.3.1 Overview . . . 8
2.3.2 JSX . . . 8
2.3.3 Update Process . . . 9
2.3.4 Virtual DOM . . . 9
2.3.5 Reconciliation Algorithm . . . 9
2.3.6 State Management . . . 10
2.3.7 Redux . . . 10
2.3.8 Strengths . . . 11
2.3.9 Weaknesses . . . 11
2.3.10 Scores . . . 12
2.4 Hilla . . . 12
2.4.1 Overview . . . 12

2.4.2 Strengths . . . 13
2.4.3 Weaknesses . . . 13
2.4.4 Scores . . . 14
2.5 Svelte . . . 14
2.5.1 Overview . . . 14
2.5.2 Strengths . . . 15
2.5.3 Weaknesses . . . 15
2.5.4 Scores . . . 15
2.6 Vue . . . 16
2.6.1 Overview . . . 16
2.6.2 Strengths . . . 16
2.6.3 Weaknesses . . . 17
2.6.4 Scores . . . 17

3 Conclusion **18**
3.1 Summary of Findings . . . 18
3.2 Concluding Scores . . . 18
3.2.1 General Complexity . . . 18
3.2.2 General Scores . . . 19
3.3 Limitations . . . 19
3.4 Future Research . . . 19
3.5 Concluding Remarks . . . 19

Acronyms **20**

Glossary **21**

AI Engineering Prompts **22**
3.6 Rewriting of text . . . 22
3.7 Small Text Block Rewrites . . . 23

Bibliography **24**

A Metrics **28**
A.1 Community Size Rating . . . 28
A.2 Ecosystem Ratings . . . 29
A.2.1 GitHub Stars . . . 29
A.2.2 Size . . . 29
A.2.3 Quality . . . 29

B Project Requirements **36**

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]
- Documentation *(interactive?)* [2]
- Ecosystem [3]
- Usage by other enterprises [2]
- Complexity [4]
- Server Side Rendered (SSR) [5, 6]

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] Likert Scale
[3] Points are 70% by quality and 30% by size, see A.2.3 and A.2.2 respectively
[4] Evaluated using the Likert Scale by easiness/speed to learn, state management, boilerplate, and API integration
[5] Likert Scale, MEDIUM/OK being available through a well-supported and known extension
[6] 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/smoothy 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

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.5 Table A.3)*
Usage by other enterprises	1
Complexity	0.25 *(Table 2.1)*
Server Side Rendered (SSR)	1

Table 2.2: Angular general scores

Total score: 5.66 Percentile score: 0.809

2.2 Lit

2.2.1 Overview

Lit, developed by Google, is a library for browser native web components, emphasizing simplicity and versatility. Unlike libraries employing VDOM, Lit is lightweight and adaptable, compatible with any web solution, including plain HTML. It minimizes boilerplate and utilizes straightforward HTML alongside JS or TS. Key features include reactive state management, scoped styles, and a declarative templating system [19].

Its primary objective is to facilitate the creation of shareable components and design systems for seamless integration across different solutions. Additionally, Lit can enhance basic HTML sites progressively.

Challenges and considerations

Web components, unlike other solutions, aren't widely adopted, leading to a smaller ecosystem for Lit. The documentation also needs improvement [20]. Additionally, Lit currently lacks SSR capability and relies on Shadow DOM *(see 2.2.2)*. Integration with IDEs is subpar, offering inaccurate linting and suggestions [21]–[23].

2.2.2 Shadow DOM

The Shadow DOM is a web standard that creates encapsulated components in web applications. It hides HTML, CSS, and JavaScript within a scoped boundary. By

attaching a separate DOM tree to an element, it shields internal structure and styles, preventing conflicts and unintended manipulation from external sources. This isolation fosters reusable, modular components, enhancing maintainability and code reusability in web development. [24] However, challenges exist with assistive technologies, developer experience, and SEO integration. [22], [25]

2.2.3 Strengths

- no VDOM overhead
- uses browser native API's *(web components)*
- not much boilerplate
- uses plain HTML, JS/TS
- compatible with any web solution
- backed by Google
- Shadow DOM ensures that components behave consistent

2.2.4 Weaknesses

- SSR is experimental
- mediocre ecosystem
- mediocre documentation
- Shadow DOM drawbacks can interfere with development simplicity
- IDE integration lacks compared to other solutions
- calls are by reference which changes the JS `this` context

2.2.5 Scores

Method	**Score**
Easiness/speed to learn	0.5
State management	0.5
Boilerplate	0.75 *(components: 0.5, state management: 1)*
API integration	0 *(default browser fetch API)*

Table 2.3: complexity

Method	Score
Community	0.3 *(Table A.1)*
Professional Support	0
Documentation (interactive walkthrough?)	1
Ecosystem	0.465 *(quality: 0.63, size: 0.3; Table A.6 Table A.3)*
Usage by other enterprises	0.5
Complexity	0.44 *(Table 2.3)*
Server Side Rendered (SSR)	0.5

Table 2.4: lit general scores

Total score: 3.205 Percentile score: 0.458

2.3 React

2.3.1 Overview

React is a library [26] created by Meta *(originally known as Facebook)*. Its recommended usage is in combination with JSX [27] *(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 [28] *(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 [29].

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 [30] 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 [31], [32].

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 [33] 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 [27], [34]

```
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 [35].

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, [29], [36])*.

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)* [36]. 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 [35]. 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 [37], [38]

```
<button class="button button-blue">
    <b>OK!</b>
</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 $O(n^3)$ [39]. Because this is too expensive for a web framework, the react reconciler implements a $O(n)$ algorithm based on two assumptions [29].

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 [31].

Sharing State

In React, there are various methods for sharing state, with one popular approach being to lift the state up *(Figure 2.2)* [40]. This method aligns with the principle of a "*single source of truth*" [41], 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)* [42].

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

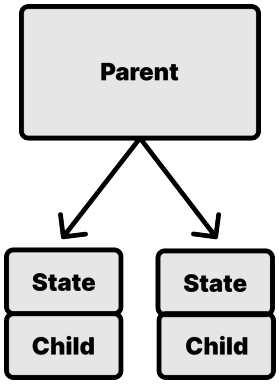


Figure 2.1: per component state

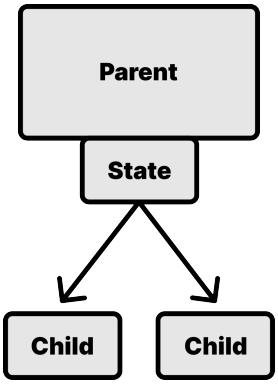


Figure 2.2: state lifted up *(shared state)*

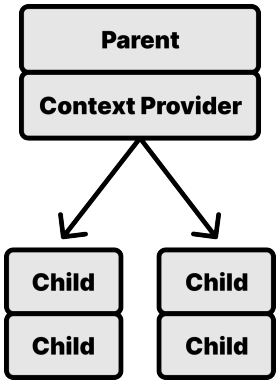


Figure 2.3: context provides state to all

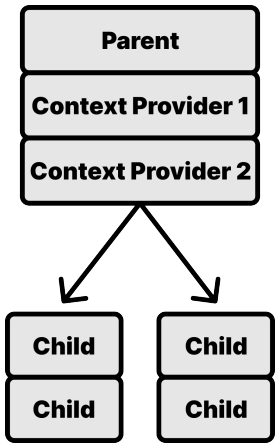


Figure 2.4: several providers provide state to all

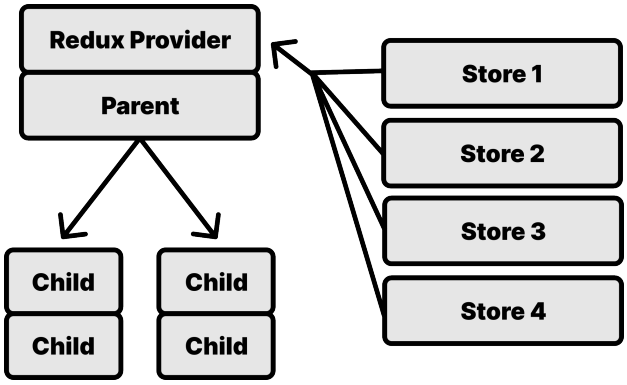


Figure 2.5: several Redux stores provide state to all

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 [44]
- not that performant compared to native JS, but a one-click addon called *Million* [45] 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 [46])*
- 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

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.7 Table A.3)*
Usage by other enterprises	1
Complexity	0.5 *(Table 2.5)*
Server Side Rendered (SSR)	0.5

Table 2.6: React general scores

Total score: 5.44 Percentile score: 0.777

2.4 Hilla

2.4.1 Overview

Hilla, created by Vaadin, is a framework that builds on top of other JS libraries, primarily focusing on the communication between the backend and frontend. This framework generates endpoints and types from the backend, which can be easily used within the frontend. The supported backend is in Java with Spring Boot, and the supported frontends are Lit *(2.2)* [47] or React *(2.3)* [48]. Hilla aims to streamline the development process by automatically handling much of the boilerplate code associated with connecting frontend components to backend services. This automation enables developers to focus on implementing business logic rather than dealing with the intricacies of communication between the two layers. By leveraging the robustness of Spring Boot for backend operations and the flexibility of modern frontend frameworks like Lit and React, Hilla provides a cohesive development environment that promotes productivity and consistency. However, the choice of technologies

and architectural decisions inherent to Hilla also introduce certain challenges that need to be carefully considered, as discussed in the following section.

Challenges and Considerations

The integration of our system poses several notable challenges and considerations that must be carefully addressed. One significant issue is the reliance on RPC calls rather than REST. This architectural choice necessitates the exclusive use of POST requests, which inherently limits the potential for caching optimizations. As a result, the system may experience inefficiencies in data retrieval and increased server load due to the inability to leverage more effective caching mechanisms typically available with other HTTP methods. [49], [50]

Another considerable challenge is the technological stack constraint, as the system is tightly coupled with Java Spring Boot for the backend and React or Lit for the frontend. This lock-in restricts flexibility in choosing alternative technologies and may hinder the adoption of more suitable or emerging frameworks that could better meet evolving project requirements.

Additionally, the system does not support the conversion of record generics, which limits the versatility and robustness of data handling and manipulation. This limitation can lead to increased complexity in the codebase and potential difficulties in maintaining and extending the system.

Method overloading for endpoints presents another area of difficulty. The system struggles with managing multiple methods that share the same name but differ in parameters, which can lead to ambiguous endpoint definitions and increased potential for errors. This issue complicates the API design and can result in less intuitive and harder-to-maintain code.

Furthermore, Java object types within the system are prone to bloat due to the custom Hilla "@Notnull" decorator. This decorator, while intended to ensure non-null constraints, adds additional overhead and complexity to the object types. The increased verbosity and clutter in the code can detract from readability and maintainability, making it more challenging for developers to navigate and understand the system.

2.4.2 Strengths

- only define the types on the backend and automatically synchronize them to the frontend
- automatic CSRF protection
- easy to use Vaadin components *(for both Lit and React)* included out of the box

2.4.3 Weaknesses

- uses RPC calls instead of REST, meaning it always uses POST requests, which hinders caching optimizations
- locks the stack into Java Spring Boot with React or Lit

- doesn't convert record generics
- struggles with method overloading for endpoints
- Java objects types get bloated with a custom Hilla "@Notnull" decorator

2.4.4 Scores

Method	**Score**
Easiness/speed to learn	1
State management	-
Boilerplate	0.5 *(components: 0.5, state management: -)*
API integration	0.5 *(implementation is lacking for features)*

Table 2.7: complexity

Method	**Score**
Community	0.3 *(Table A.1)*
Professional Support	1
Documentation (interactive walkthrough?)	0.5
Ecosystem	0.115 *(quality: 0.23, size: 0; Table A.8 Table A.3)*
Usage by other enterprises	0.5
Complexity	0.666... *(Table 2.7)*
Server Side Rendered (SSR)	-

Table 2.8: Hilla general scores

Total score: 3.081 Percentile score: 0.514

2.5 Svelte

2.5.1 Overview

Svelte, created by Rich Harris and maintained by the open-source community with support from Vercel [51], stands out for its compiled approach, speed, and simplicity, removing the need for boilerplate code. This framework has seen significant adoption, including in enterprise settings, and has developed a large and vibrant ecosystem.

Components in Svelte are contained within a custom file type that is both readable and accessible, even for beginners. This file type is compiled into JS, HTML, and CSS, ensuring efficient and optimized performance. Unlike many other frameworks, Svelte does not use the VDOM, thereby avoiding its associated drawbacks [52]. Additionally, Svelte automatically scopes styles, simplifying the management of component-specific styles and reducing the risk of style conflicts. [53]

Svelte's popularity is reflected in its acclaim, having been voted the most loved and admired JS framework in the Stack Overflow survey [54]. Besides its technical strengths, Svelte offers an in-depth interactive walkthrough and excellent documentation, making it accessible to developers of all skill levels and easing the learning

curve. Moreover, Svelte includes an SSR extension that offers numerous out-of-the-box features for full-stack development and emphasizes accessibility by providing linter warnings [55], [56].

Challenges and Considerations

Due to its anti-boilerplate philosophy, Svelte presents some challenges when writing advanced state manipulation logic. However, this issue is addressed in Svelte 5 with the introduction of Runes *(Svelte 5 is, at the time of writing, not yet released nor does it have a planned release date)* [57]. Using Svelte without its supported SSR extension, called SvelteKit, can feel immature.

2.5.2 Strengths

- officially supported SSR solution *(SvelteKit)*
- custom file type which is easily readable and understandable
- great documentation with interactive walkthrough
- lots of usage including in enterprises
- big ecosystem
- compile time optimizations
- style is automatically scoped and can be bypassed using the ":global" selector
- backed by Vercel
- most loved/admired JS framework according to the Stack Overflow survey [54]
- emphasizes accessibility using linter warnings

2.5.3 Weaknesses

- anti boilerplate mindset can make state management more complex than needed *(resolved in Svelte 5)*
- usage without SvelteKit doesn't feel mature

2.5.4 Scores

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

Table 2.9: complexity

Method	Score
Community	0.7 *(Table A.1)*
Professional Support	1
Documentation (interactive walkthrough?)	1
Ecosystem	0.565 *(quality: 0.83, size: 0.3; Table A.9 Table A.3)*
Usage by other enterprises	1
Complexity	0.625 *(Table 2.9)*
Server Side Rendered (SSR)	1

Table 2.10: Svelte general scores

Total score: 5.89 Percentile score: 0.841

2.6 Vue

2.6.1 Overview

Vue, created by Evan You and maintained by the open-source community, has been financially sustainable since 2016 through sponsorships [58], [59]. It is built to be approachable, utilizing standard HTML, CSS, and JS, and is extensively documented. Its updates are performant due to compile-time optimizations [60].

Like Svelte *(2.5)*, Vue uses its own file type, which is similar in structure. The primary difference is that in Vue, unlike Svelte, you need to wrap your HTML content with a template tag [53], [61]. This approach almost eliminates boilerplate code.

Similar to React *(2.3)*, Vue uses the VDOM *(2.3.4)* to manage its reactivity logic [28], [62]. Additionally, the Vue team maintains two packages beneficial for large applications: Vue Router and Pinia. Pinia serves as a global state management system, comparable to Redux *(2.3.7)*, but retains the same state management logic as standard Vue processes, making it extremely fast to learn and easy to use [63], [64].

Furthermore, Vue's state management system is easy to use and scalable, without the confusing side effects seen in React.

Challenges and Considerations

The VDOM introduces a memory overhead in the browser, which is a concern not present in other solutions that do not utilize the VDOM. Despite several compile-time optimizations, this overhead remains an issue. Additionally, there are no officially supported SSR solutions for VDOM. However, popular and stable alternatives like Nuxt.js and VuePress are available. Lastly, there have been issues with nested generics passing through slots not correctly detecting types, but this is a narrow problem that will likely be resolved with a future patch.

2.6.2 Strengths

- great official global state management library *(Pinia)*
- great official router library *(Vue Router)*

- custom file type that is easy to learn and readable
- great documentation with an interactive walkthrough
- lots of usage, including in enterprises
- big ecosystem
- performant out of the box due to compile time optimizations
- easy to learn and maintainable state management system
- financially stable and backed by Vercel
- boilerplate free

2.6.3 Weaknesses

- VDOM memory overhead
- no out of the box SSR solution
- issues with nested generics to child params

2.6.4 Scores

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

Table 2.11: complexity

Method	**Score**
Community	0.8 *(Table A.1)*
Professional Support	1
Documentation (interactive walkthrough?)	1
Ecosystem	0.775 *(quality: 0.85, size: 0.7; Table A.10 Table A.3)*
Usage by other enterprises	1
Complexity	0.75 *(Table 2.11)*
Server Side Rendered (SSR)	0.5

Table 2.12: Vue general scores

Total score: 5.825 Percentile score: 0.832

CHAPTER 3

Conclusion

This study examined various methods for building web applications, evaluating their strengths and weaknesses to identify the most suitable solution for different types of projects. The usability of each solution was also tested to provide a comprehensive evaluation.

3.1 Summary of Findings

The research indicates that Svelte, Vue, Angular, and React are top choices for general use, with each solution offering unique advantages.

1. Svelte scored the highest overall, being straightforward to use for simple applications but requiring improvements in state management for more complex projects.

2. Vue is particularly suitable for non-SSR applications and offers a seamless DX.

3. React has the largest ecosystem and community support, making it versatile for various projects.

4. Angular offers a unique features but lags behind its competitors in DX.

5. Lit excels in cross-solution components and enhancing plain HTML sites.

6. Hilla introduces a novel frontend-backend communication concept and could pair well with React if current issues are resolved.

3.2 Concluding Scores

The tables provide a quick summary of the ratings for each web solution with the best score highlighted in bold.

3.2.1 General Complexity

The sources for each complexity score are derived from the following tables: Table 2.1, Table 2.3, Table 2.5, Table 2.7, Table 2.9, Table 2.11

Method	Angular	Lit	React	Hilla	Svelte	Vue
Easiness/speed to learn	0.5	0.5	**1**	**1**	**1**	**1**
State management	0	0.5	0.5	-	0.5	**1**
Boilerplate	0	0.75	0.5	0.5	**1**	**1**
API integration	**0.5**	0	0	**0.5**	0	0

Table 3.1: General complexity overview

3.2.2 General Scores

The sources for each general score are derived from the following tables: Table 2.2, Table 2.4, Table 2.6, Table 2.8, Table 2.10, Table 2.12

Method	Angular	Lit	React	Hilla	Svelte	Vue
Community	0.6	0.3	**1**	0.3	0.7	0.8
Professional Support	**1**	0	**1**	**1**	**1**	**1**
Documentation (interactive walkthrough?)	**1**	**1**	0.5	0.5	**1**	**1**
Ecosystem	0.81	0.465	**0.94**	0.115	0.565	0.775
Usage by other enterprises	**1**	0.5	**1**	0.5	**1**	**1**
Complexity	0.25	0.44	0.5	0.666...	0.625	**0.75**
Server Side Rendered (SSR)	**1**	0.5	0.5	-	**1**	0.5
General Score	0.809	0.458	0.777	0.514	**0.841**	0.832

Table 3.2: General scores overview

3.3 Limitations

This study focused on popular web solutions and general use cases. The rapidly evolving web development landscape means that new features could change the current standings soon. Additionally, DX can be subjective and may vary among individuals and teams.

3.4 Future Research

Future research should explore the long-term performance and scalability of these solutions in different environments. Investigating the impact of new features and updates on usability and efficiency would be beneficial. A deeper analysis of specific use cases, such as e-commerce or content management systems, could provide more targeted recommendations.

3.5 Concluding Remarks

Selecting a web development solution should consider the specific needs and goals of the project. By understanding the specific advantages and limitations of each solution developers can make informed decisions that enhance their development experience. This study highlights the importance of continuous evaluation and adaptation in the fast-paced field of web development.

Acronyms

API Application Programming Interface. 6, 7, 12, 14, 15, 17, 19

CLI Command Line Interface. 4, 5

CSS Cascading Style Sheets. 5, 6, 14, 16

DOM document object model. 6–9, 20

DX developer experience. 1, 18, 19, 21

HMR Hot Module Replacement [65]. 5

HTML Hypertext Markup Language. 4–9, 14, 16, 18

HTTP Hypertext Transfer Protocol. 13

I18N internationalisation. 5, 8, 12

IDE Integrated Development Environment. 6, 7

JS JavaScript. 6–8, 12, 14–16

JSON JavaScript Object Notation. 9

JSX JavaScript XML. 8, 11

REST Representational State Transfer. 13

RPC Remote Procedure Call. 13

SEO Search Engine Optimalisation. 3, 7

SSR Server Side Rendered. 3–7, 15–18

TS TypeScript. 6, 7

UI user interface. 8

UX user experience. 1, 21

VDOM Virtual DOM. 5–9, 11, 14, 16, 17

XML Extensible Markup Language. 8, 20

Glossary

boolean a value which is either true or false *(0 or 1)*. 21

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. 20

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" [66]. 20

framework a supporting structure that requires specificity. You must follow the pattern of the framework. It is essentially a skeleton that is in control of your code. With a library, your code is in control [67]. 1, 4, 9, 12, 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. 20

library a collection of pre-existing code that can be utilized to create new code [67]. 8, 21

predicate something that results in a boolean. 30

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. 20

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.

3.6 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>

3.7 Small Text Block Rewrites

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.
- 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
- Do not use a listing/enumeration/bullet points
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] Mozilla - MDN, *Using shadow DOM*, https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM, [Online; accessed 10-May-2024].

[25] Manuel Matuzović, *Pros and cons of using Shadow DOM and style encapsulation*, https://www.matuzo.at/blog/2023/pros-and-cons-of-shadow-dom, [Online; accessed 10-May-2024].

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

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

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

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

[30] 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.

[31] 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.

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

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

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

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

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

[37] 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].

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

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

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

[41] 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.

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

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

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

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

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

[47] Hilla, *Documentation - Lit*, https://hilla.dev/docs/lit, [Online; accessed 17-May-2024].

[48] Hilla, *Documentation - React*, https://hilla.dev/docs/react, [Online; accessed 17-May-2024].

[49] Lokesh Gupta, *Caching REST API Response*, https://restfulapi.net/caching/, [Online; accessed 18-May-2024].

[50] kiewic, *Is Chrome ignoring Cache-Control: max-age?* https://stackoverflow.com/a/14842553/12668716, [Online; accessed 18-May-2024].

[51] Guillermo Rauch, CEO and Founder of Vercel, *Vercel welcomes Rich Harris, creator of Svelte*, https://vercel.com/blog/vercel-welcomes-rich-harris-creator-of-svelte, [Online; accessed 18-May-2024].

[52] Svelte, *Virtual DOM is pure overhead*, https://svelte.dev/blog/virtual-dom-is-pure-overhead, [Online; accessed 18-May-2024].

[53] Svelte, *Svelte components*, https://svelte.dev/docs/svelte-components, [Online; accessed 18-May-2024].

[54] Stack Overflow, *Developer Survey 2023*, https://survey.stackoverflow.co/2023/, [Online; accessed 18-May-2024].

[55] SvelteKit, *Web development, streamlined*, https://kit.svelte.dev/, [Online; accessed 18-May-2024].

[56] Svelte, *Accessibility warnings*, https://svelte.dev/docs/accessibility-warnings, [Online; accessed 18-May-2024].

[57] Svelte, *Introducing runes*, https://svelte.dev/blog/runes, [Online; accessed 18-May-2024].

[58] Vue.js, *Become a Vue.js sponsor*, https://vuejs.org/sponsor/, [Online; accessed 20-May-2024].

[59] Vue.js, *Frequently Asked Questions*, https://vuejs.org/about/faq, [Online; accessed 20-May-2024].

[60] Vue.js, *The Progressive JavaScript Framework*, https://vuejs.org/, [Online; accessed 19-May-2024].

[61] Vue.js, *Template Syntax*, https://vuejs.org/guide/essentials/template-syntax.html, [Online; accessed 19-May-2024].

[62] Vue.js, *Rendering Mechanism*, https://vuejs.org/guide/extras/rendering-mechanism.html, [Online; accessed 19-May-2024].

[63] Vue Router, *Getting Started*, https://router.vuejs.org/guide/, [Online; accessed 19-May-2024].

[64] Pinia, *Introduction*, https://pinia.vuejs.org/introduction.html, [Online; accessed 19-May-2024].

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

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

[67] 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

Angular

nr	Tag	Stars^{+}	Documentation*	Tests*
1	angular/angular	0.7	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**	**0.508**	**1**	**0.957**

Table A.5: Angular ecosystem ratings - 10/05/2024

Score: $\frac{0.508+1+0.957}{3} = 0.822 \approx 0.82$

Lit

nr	**Tag**	**Stars**$^{+}$	**Documentation***	**Tests***
1	lit/lit	0.5	yes	yes
2	material-components/material-web	0.3	yes	yes
3	home-assistant/frontend	0.3	yes	yes
4	TanStack/form	0.3	yes	yes
5	daybrush/selecto	0.3	yes	yes
6	Trendyol/baklava	0.2	yes	yes
7	ascorbic/unpic-img	0.2	yes	yes
8	pwa-builder/pwa-starter	0.2	yes	yes
9	carbon-design-system/carbon-for-ibm-dotcom	0	yes	yes
10	micro-lc/micro-lc	0	yes	yes
11	runem/lit-analyzer	0	yes	yes
12	giscus/giscus-component	0	yes	yes
13	kor-ui/kor	0	yes	no
14	adobe/lit-mobx	0	yes	yes
15	microsoft/typescript-lit-html-plugin	0	yes	yes
16	IBM/pwa-lit-template	0	yes	no
17	motss/app-datepicker	0	yes	yes
18	hydrogenjs/hydrogen	0	yes	yes
19	steeze-ui/icons	0	yes	no
20	andreasbm/lit-translate	0	yes	yes
21	phase2/outline	0	yes	no
22	fernandopasik/lit-redux-router	0	yes	yes
23	PolymerLabs/lit-ssr	0	yes	yes
24	kitze/copy-lite	0	yes	no
25	Festify/fit-html	0	yes	yes
-	**Average**	**0.092**	**1**	**0.8**

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

Score: $\frac{0.092+1+0.8}{3} = 0.630666\cdots \approx 0.63$

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.7: React ecosystem ratings - 24/04/2024

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

Hilla

nr	**Tag**	**Stars**$^{+}$	**Documentation***	**Tests***
1	spring-projects/spring-boot	0.7	yes	yes
2	spring-projects/spring-framework	0.7	yes	yes
3	vaadin/hilla	0.1	yes	yes
4	vaadin/docs	0	yes	-
5	TatuLund/hilla-demo	0	no	yes
6	miliariadnane/moroccan-cooking-companion	0	no	yes
7	spring-petclinic/spring-petclinic-hilla	0	yes	yes
8	mcollovati/quarkus-hilla	0	yes	yes
9	TatuLund/hilla-react-demo	0	no	no
10	Thanh25102/chat-rocket	0	no	no
11	DurjoyAcharya/todo-application	0	no	no
12	nisha8c/Hilla-React-TodoWithAddedFunctions	0	no	no
13	simasch/sakila	0	no	yes
14	pavelklecansky/recipedb-hilla	0	no	yes
15	vaadin/directory	0	no	no
16	TatuLund/hilla-v2-demo	0	no	yes
17	ShubhamkumarAnand/todo-Hilla-framework	0	no	no
18	exouciam/ProjektbasiertesArbeiten_2WS-23-34	0	no	no
19	pavelklecansky/note	0	no	yes
20	ashutoshsahoo/spring-boot-react-hilla-crm	0	no	no
21	TatuLund/hilla-react-ui5	0	no	no
22	teggr/hilla-dev-test-project	0	no	no
23	nisha8c/HILLA-React-Todo-Practice	0	no	no
24	Inmoresentum/ai-powered-quiz-app	0	no	no
25	nisha8c/Authentication-with-Spring-Security	0	no	no
-	**Average**	**0.006**	**0.24**	**0.458**

Table A.8: Hilla ecosystem ratings - 18/05/2024

Score: $\frac{0.006+0.24+0.458}{3} = 0.234666\cdots \approx 0.23$

Svelte

nr	**Tag**	**Stars**$^{+}$	**Documentation***	**Tests***
1	storybookjs/storybook	0.7	yes	yes
2	sveltejs/svelte	0.7	yes	yes
3	TanStack/query	0.6	yes	yes
4	immich-app/immich	0.6	yes	yes
5	saadeghi/daisyui	0.6	yes	yes
6	TanStack/table	0.5	yes	yes
7	NativeScript/NativeScript	0.5	yes	yes
8	wailsapp/wails	0.5	yes	yes
9	xyflow/xyflow	0.5	yes	yes
10	sveltejs/kit	0.5	yes	yes
11	tabler/tabler-icons	0.5	yes	yes
12	coollabsio/coolify	0.5	yes	yes
13	codesandbox/codesandbox-client	0.5	yes	yes
14	yoavbls/pretty-ts-errors	0.5	yes	yes
15	BuilderIO/mitosis	0.4	yes	yes
16	gitbutlerapp/gitbutler	0.4	yes	yes
17	openreplay/openreplay	0.4	yes	yes
18	vercel/ai	0.4	yes	yes
19	jsplumb/jsplumb	0.4	-	-
20	TanStack/router	0.4	yes	yes
21	tsparticles/tsparticles	0.4	yes	yes
22	BuilderIO/builder	0.4	yes	yes
23	windicss/windicss	0.4	yes	yes
24	huggingface/chat-ui	0.4	yes	yes
25	inertiajs/inertia	0.3	yes	yes
-	**Average**	**0.48**	**1**	**1**

Table A.9: Svelte ecosystem ratings - 18/05/2024

Score: $\frac{0.48+1+1}{3} = 0.82666\cdots \approx 0.83$

Vue

nr	**Tag**	**Stars**$^{+}$	**Documentation***	**Tests***
1	vuejs/vue	1	yes	yes
2	PanJiaChen/vue-element-admin	0.7	yes	yes
3	storybookjs/storybook	0.7	yes	yes
4	vuejs/awesome-vue	0.7	yes	-
5	hoppscotch/hoppscotch	0.7	yes	yes
6	https://github.com/ElemeFE/element	0.7	yes	yes
7	nuxt/nuxt	0.7	yes	yes
8	abi/screenshot-to-code	0.7	yes	no
9	ionic-team/ionic-framework	0.7	yes	yes
10	prettier/prettier	0.6	yes	yes
11	marktext/marktext	0.6	yes	yes
12	vuejs/core	0.6	yes	yes
13	go-gitea/gitea	0.6	yes	yes
14	Asabeneh/30-Days-Of-JavaScript	0.6	yes	-
15	bailicangdu/vue2-elm	0.6	yes	no
16	dcloudio/uni-app	0.6	yes	yes
17	vuetifyjs/vuetify	0.6	yes	yes
18	jeecgboot/jeecg-boot	0.6	yes	yes
19	lyswhut/lx-music-desktop	0.6	yes	yes
20	NervJS/taro	0.6	yes	yes
21	SheetJS/sheetjs	0.6	yes	yes
22	slidevjs/slidev	0.6	yes	yes
23	vuejs/vue-cli	0.6	yes	yes
24	vuejs/vuex	0.6	yes	yes
25	Advanced-Frontend/Daily-Interview-Question	0.6	yes	-
-	**Average**	**0.648**	**1**	**0.91**

Table A.10: Vue ecosystem ratings - 21/05/2024

Score: $\frac{0.648+1+1}{3} = 0.852666\cdots \approx 0.85$

APPENDIX B

Project Requirements

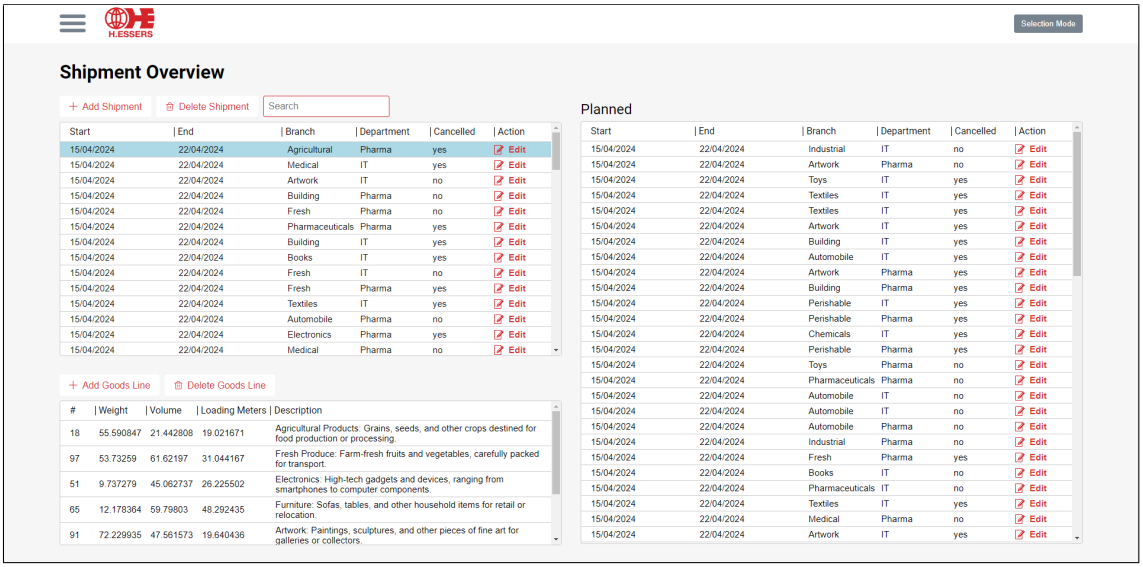


Figure B.1: general layout - `https://link.arthurdw.com/ngws-layout`

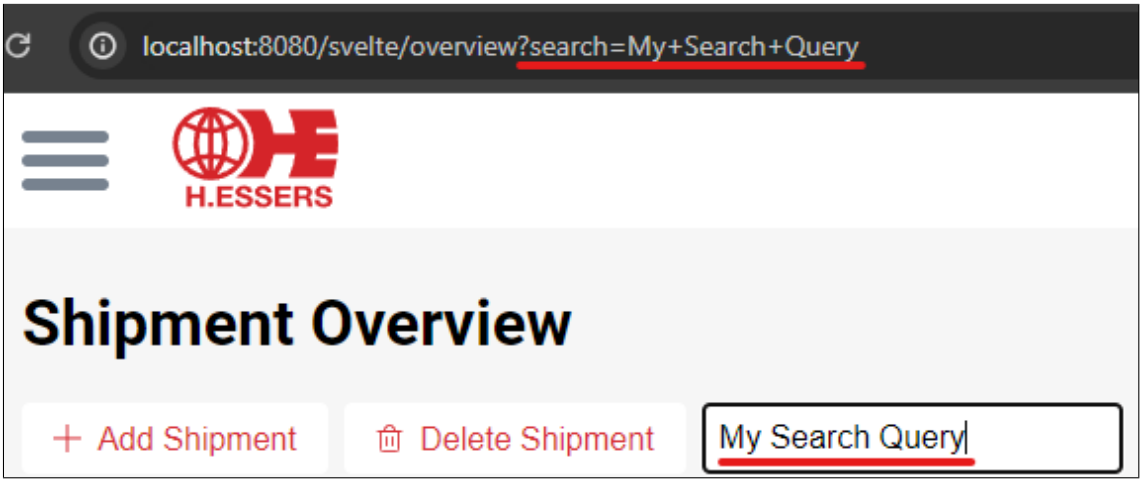
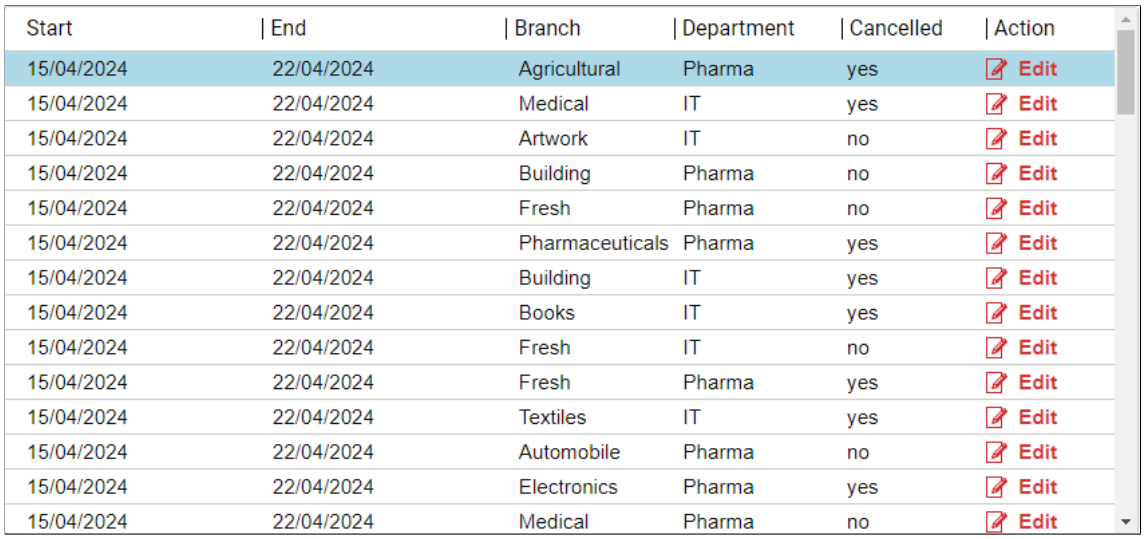
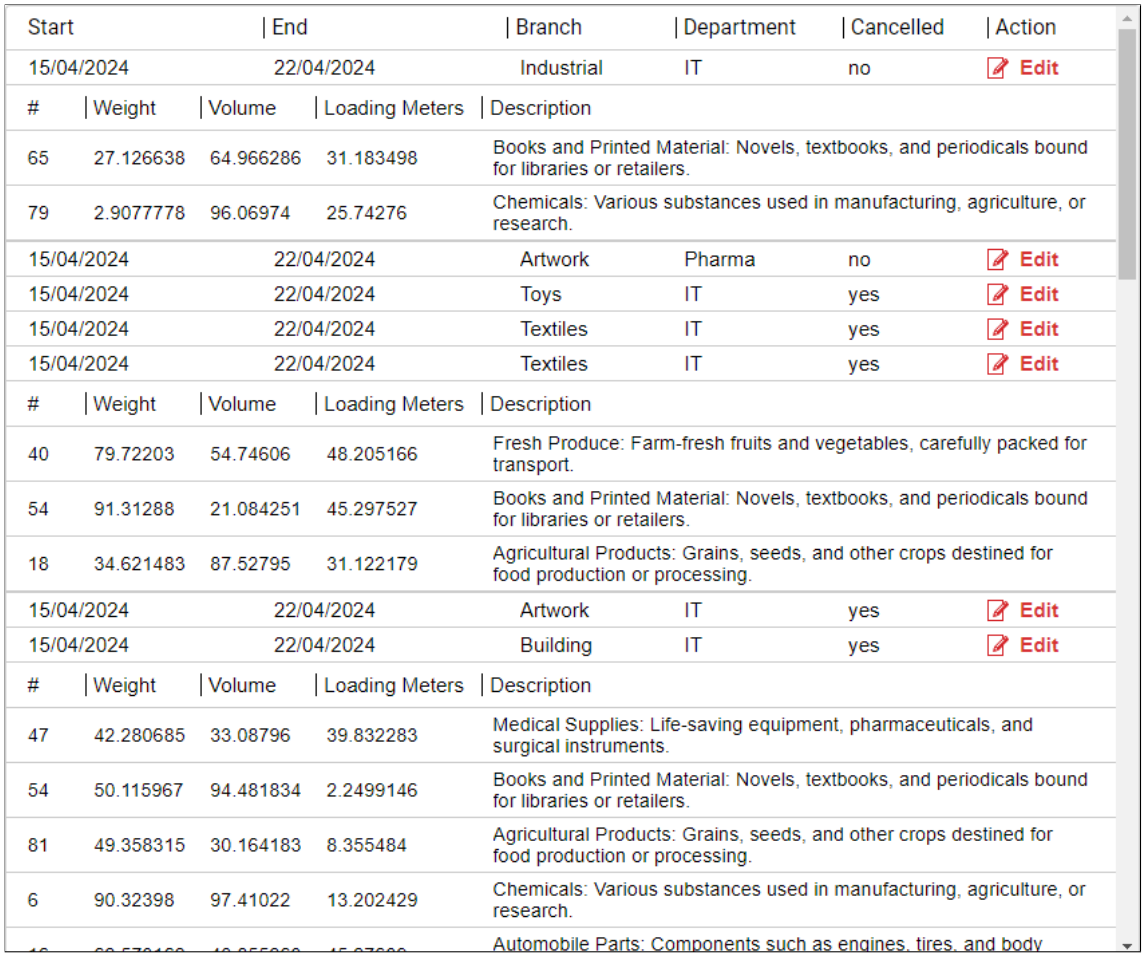


Figure B.2: interactive search - `https://link.arthurdw.com/ngws-search`



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`

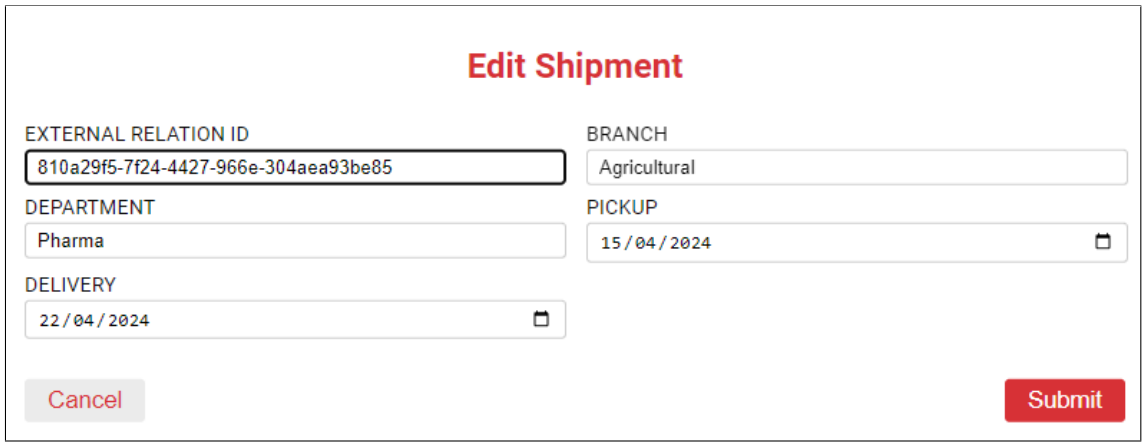


Figure B.5: form: `https://link.arthurdw.com/ngws-form_1`

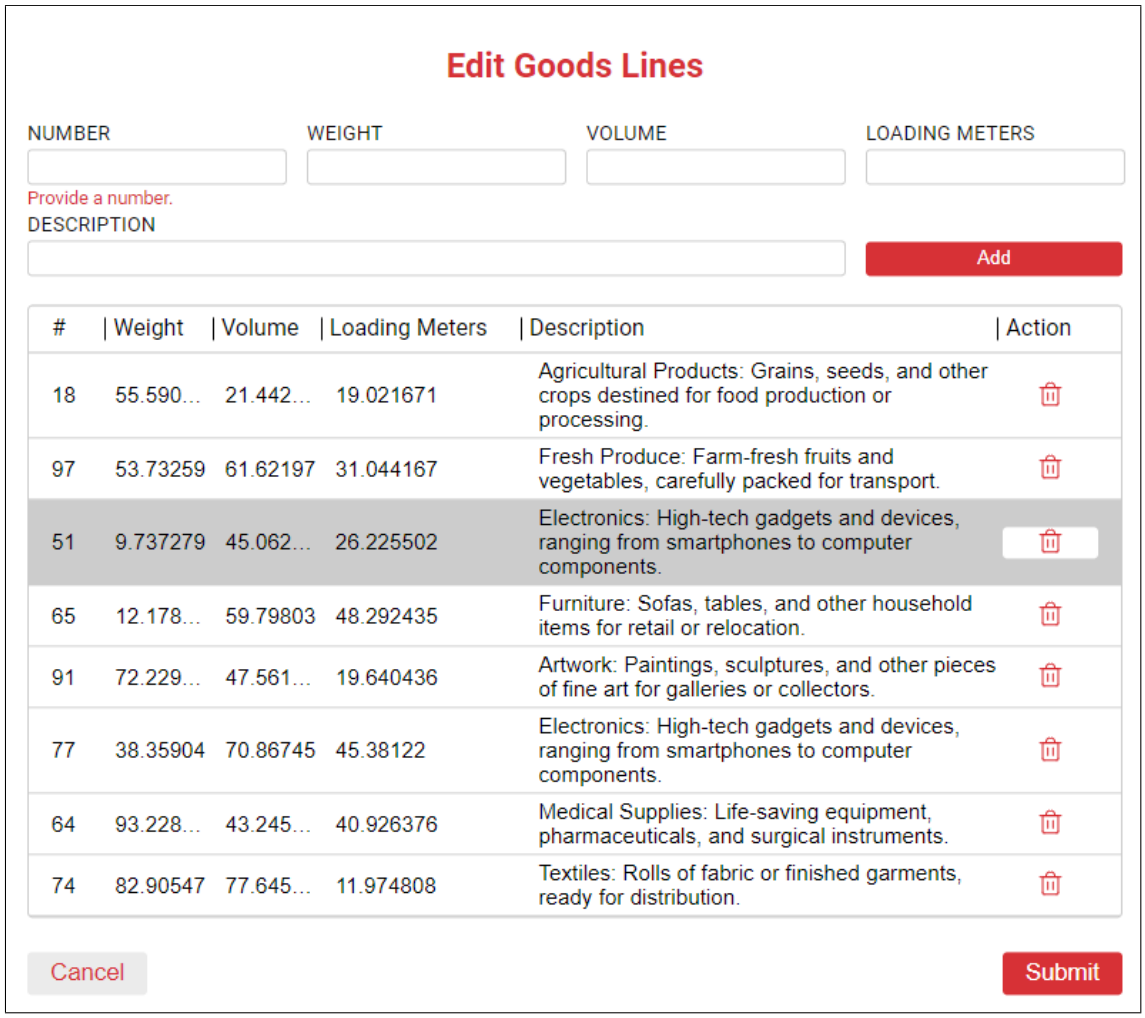


Figure B.6: form with grid - `https://link.arthurdw.com/ngws-form_2`

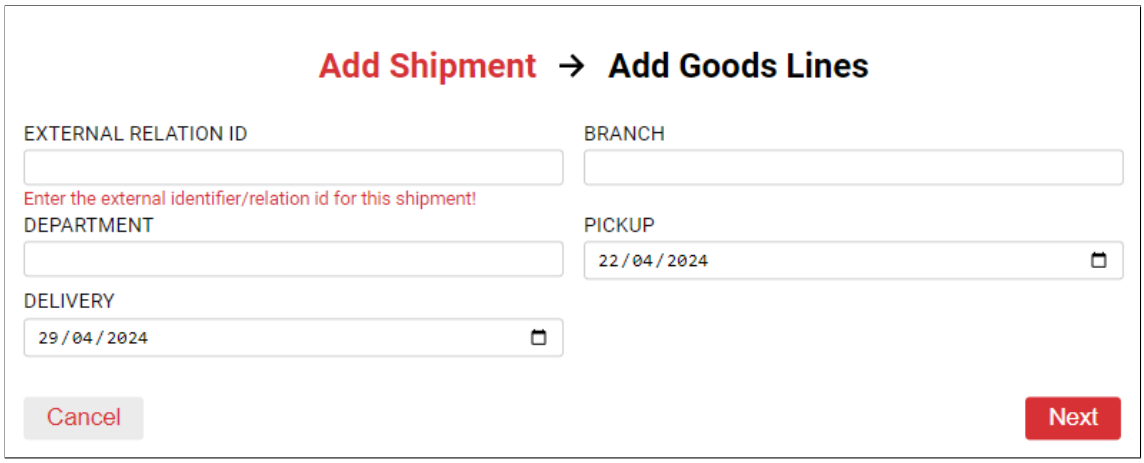


Figure B.7: wizard form: `https://link.arthurdw.com/ngws-wizard_1`

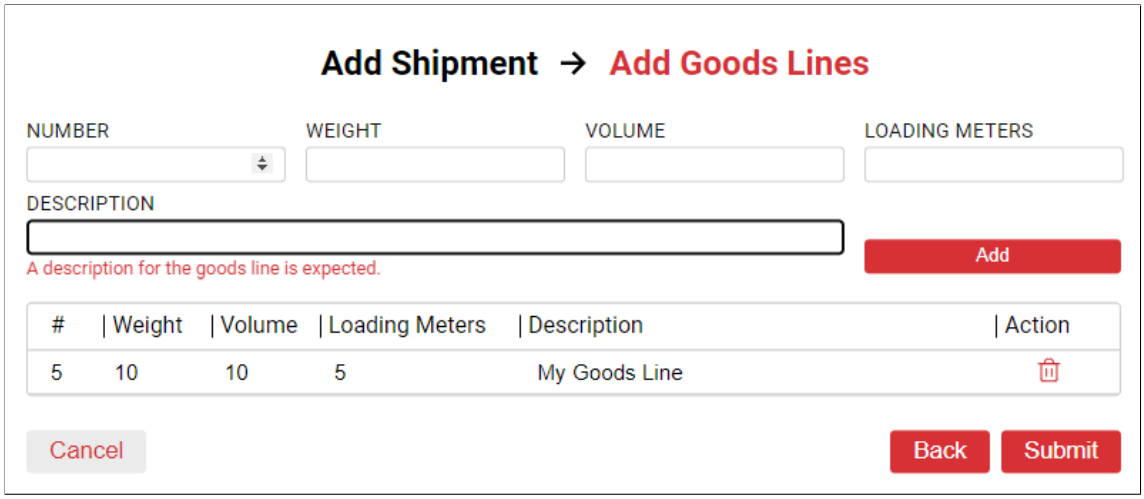


Figure B.8: wizard form with grid - `https://link.arthurdw.com/ngws-wizard_2`

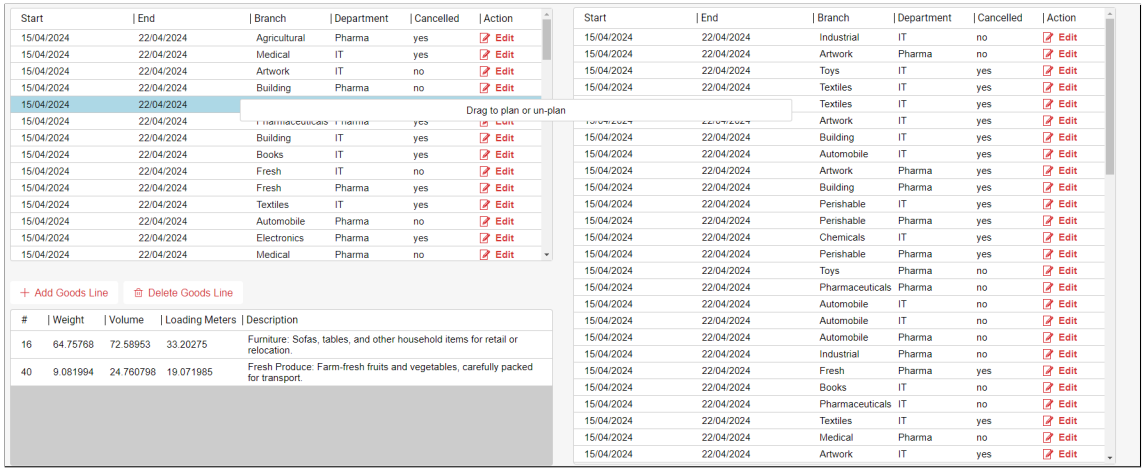


Figure B.9: drag and drop - `https://link.arthurdw.com/ngws-dnd`

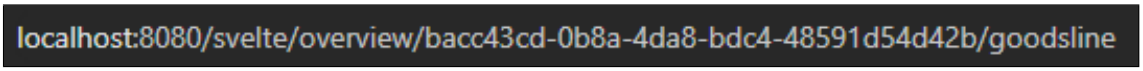


Figure B.10: reflective routing - `https://link.arthurdw.com/ngws-routing_1`

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

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

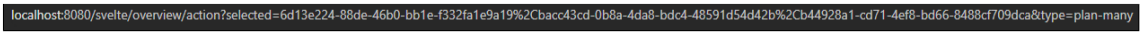


Figure B.13: reflective routing - `https://link.arthurdw.com/ngws-routing_4`

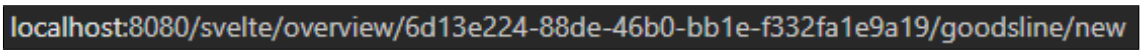


Figure B.14: reflective routing - `https://link.arthurdw.com/ngws-routing_5`