Aalto University
School of Science
Bachelor's Programme in Science and Technology

Review of HTML-oriented state managers for web development

Bachelor's Thesis

April 22, 2024

Emil Ekholm

Aalto University
School of Science
Bachelor's Programme in Science and Technology

ABSTRACT OF
BACHELOR'S THESIS

Author:	Emil Ekholm
Title of thesis:	Review of HTML-oriented state managers for web development
Date:	April 22, 2024
Pages:	39
Major:	Computer Science
Code:	SCI3027
Supervisor:	Prof., Lauri Savioja
Instructor:	M.Sc., Juho Vepsäläinen (Department of Computer Science)

Web frameworks like React and Vue.js dominate the current landscape of front-end development. Frameworks like these have a high level of adoption and are flexible and efficient. However, they fail to cater to every requirement, particularly ease of use and suitability for small-scale projects. Recognizing these limitations, this thesis compares and evaluates HTML-oriented state managers with traditional frameworks. The main goal of HTML-oriented state managers is to integrate functionality directly into the HTML seamlessly.

When evaluating the frameworks the study focuses on the technologies from a developer and technological perspective. Finally, the thesis discusses the future of these technologies and how the applicable technologies align with the HTML First directive. The frameworks selected for evaluation are Alpine.js, Mavo, and Petite-vue. A basic 'todo' app was developed for each framework to showcase the features.

Numerous advantages and factors influencing the decision to not select the analyzed frameworks were identified. In a situation where you are building a Single-Page Application (SPA) and need things like feature authentication, the currently widely adopted frameworks still provide the best options. The HTML-state managers studied were found to have the best applicability when the user wanted to introduce simple interactivity to the web page. The analyzed frameworks align well with the HTML First directive. Alpine.js and Petite-vue were also found to be a great replacement for aging frameworks, like jQuery. In situations, when you need to create a CRUD application and quickly connect it to a suitable backend Mavo seems like an excellent option. Finally, it's crucial to acknowledge the lower level of expertise required to use the analyzed frameworks compared with React and Vue.

Keywords:	AJAX, Alpine.js, HTML First, JavaScript, Mavo, Petite-vue, React
Language:	English

Aalto-universitetet
Högskolan för teknikvetenskaper
Kandidatprogram i teknikvetenskap

SAMMANDRAG AV
KANDIDATARBETET

Utfört av:	Emil Ekholm
Arbetets namn:	Granskning av HTML-orienterade statushanterare för webbutveckling
Datum:	Den 22 april 2024
Sidoantal:	39
Huvudämne:	Datateknik
Kod:	SCI3027
Övervakare:	Prof., Lauri Savioja
Handledare:	M.Sc., Juho Vepsäläinen (Institutionen för datateknik)

Webbutvecklingssektorn domineras för tillfället av webbramverk så som React och Vue.js. Deras höga effektivitet och flexibilitet är en bidragande orsak. Ramverken är inga universallösningar och teknologierna är i flera fall olämpliga speciellt när det kommer till applikationer som inte behöver den höga komplexiteten som erbjuds. Arbetet jämför och utvärderar dessa lösningar gentemot HTML-orienterade statushanterare. Det mest centrala målet med en HTML-orienterad statushanterare är att integrera funktionalitet direkt i HTML på ett smidigt sätt. Ramverken utvärderas från ett teknologiskt perspektiv och en användares perspektiv. Utöver detta behandlar arbetet även möjliga framtidsutsikter för de analyserade teknologierna och deras relation till HTML Först-principen. De ramverk som är valda för utvärdering är Alpine.js, Mavo och Petite-vue. För att demonstrera ramverkens syntax och implementering konkret görs en ”att-göra-applikation” för alla ramverk.

Många faktorer påverkar i vilka situationer man inte ska välja de analyserade ramverkena. Främst skall man undvika ramverkena då man bygger komplicerade SPA-applikationer. Applikationer som behöver speciellt hög säkerhet eller komplicerad identifiering av användaren kräver SPA-teknologier. Vid utveckling av applikationer som kräver stora mängder visuell framställning bör man använda ramverk som har en virtuell DOM.

HTML-orienterade statushanterare fungerar bäst när utvecklaren vill introducera funktionalitet till en existerande eller enkel HTML-sida. Detta möjliggör ersättning av åldrande ramverk som jQuery med Alpine.js och Petite-vue. HTML-orienterade statushanterarna är också kompatibla med HTML Först- direktiven. Om man behöver skapa en CRUD-applikation är Mavo ett utmärkt alternativ. En av de mest centrala fördelarna med HTML-baserade statushanterare är ändå den lägre kunskapsnivå som krävs för att förstå och kunna programmera funktionalitet i webbsidor.

Nyckelord:	AJAX, Alpine.js, HTML First, JavaScript, Mavo, Petite-vue
Språk:	Engelska

Contents

Abbreviations

AJAX	Asynchronous JavaScript and XML
AMP	Accelerated Mobile Pages (Open source framework)
CSS	Cascading Style Sheets
CMS	Content Management System
CRUD	Create, Read, Update, Delete applications
DSL	Domain-Specific Language
DOM	Document Object Model
HTML	Hyper Text Markup Language
JAVAX	Java Extension
MPA	Multi-Page Application
SPA	Single-Page Application
WYSIWYG	What You See Is What You Get

1 Introduction

The current web development landscape is reliant on Single-Page Applications (SPAs) [24]. A SPA uses one HTML web page as the main foundation of the web page. Interactions on the web page are generally implemented using JavaScript [14]. Technologies like JavaScript enable the movement of logic and computation from the server to the client side. When invented this was a breakthrough that resulted in dynamic and non-server-centered websites, and in general, has led to the current environment where the industry standard is JavaScript frameworks like React, Vue and Angular [14]. Although the current mainstream frameworks offer many advantages there are situations where other technologies might be more beneficial to use. Instead of using the previously mentioned frameworks state managers existing in HTML itself can be used. This thesis will compare and evaluate these state managers.

The HTML First guidelines emphasize using vanilla HTML to develop whenever possible [13] and avoiding defaulting to a JavaScript framework when the functionality can be achieved using vanilla HTML. The idea with HTML First is to decrease the friction in writing web software and use the capabilities of modern browsers instead of using JavaScript frameworks by default [13]. The research evaluates how HTML-oriented state managers integrate the practice of HTML First development. Furthermore, the research investigates how these types of development might interfere with traditional software practices[1].

The goal of the research is to review HTML-oriented state managers as an alternative to mainstream frameworks and answer the question: **Q: How can HTML-markup-oriented state managers improve the web application development landscape?** To further scope the research I decided to define these questions based on the previous question.

1. How can HTML-markup-oriented state managers improve the development practices when it comes to HTML first development?
2. What technological advantages/disadvantages do HTML-markup-oriented state managers have compared with current solutions?

The work begins by explaining how frontend development has evolved and it describes the way that the currently widely adopted frameworks work. The HTML-oriented state managers selected to limit the scope of the research are Alpine.js, Mavo, and Petite-vue based on the arguments made in Section 2.2.6. These are subsequently discussed and examined in the chapters 3, 4 and 5. The focus lies on development practices and technological advantages/disadvantages. Lastly, the work compares the selected frameworks between each other and contemporary frameworks in chapter 6.

[1] One popular practice is the separation of concerns [13]

2 Frontend development

This chapter will discuss the history of frontend development and how the landscape has evolved to solutions like Single Page Applications (SPA)[2] and Multi-Page Applications (MPA)[3]. Current popular frameworks used to create SPAs, like React, are introduced. Finally, the chapter touches on HTML markup state managers generally and defines the ones chosen for closer evaluation in the thesis.

2.1 Frontend development from the 90s to 2020s

Web technologies in 1990 began as static web pages. The static web pages meant that application changes and refreshes required full-page updates because of the logic being concentrated on the server side [14]. Applications structured this way are called server-centric applications [14]. Such an architectural choice led to a system where the front-end technology only handled rendering. Furthermore, even session and application states were stored on the backend. Therefore, the system was when compared to subsequent inventions highly unresponsive. Due to full-page refreshes being the standard MPAs established their presence and became the norm for creating web applications [14].

MPAs operate traditionally, where any browser change requires fetching new pages from the server. Routes are server-side registered, and each client request involves fetching a new HTML page, resulting in either the requested view or an error [16]. Application logic primarily resides on the server, with the client acting solely as a recipient of fetched pages [16]. Asynchronous JavaScript would later improve the MPAs and allow for only partial refreshes [16].

2.1.1 AJAX revolution (mid 2000s)

The Asynchronous JavaScript (AJAX) revolution wasn't new technology but rather existing development techniques paired with an asynchronous mindset (`XMLHttpRequest` objects) that allowed the implementation of partly rendered HTML pages. Asynchronous requests meant that other functionality would be responsive when previous actions were processed [14]. From a developer standpoint, this change initially improved code readability but as the website's mechanics got more complex the need for frameworks to handle the client-side logic emerged [16].

[2]Web application that dynamically changes content on a single page.

[3]Web application where the different pages represent different sections and functionality of the application

2.1.2 The rise of the Single Page Applications (early 2010s)

A Single-Page Application (SPA) uses one HTML web page as the main foundation of the web page [24]. The JAVAX revolution facilitated the growth of these web pages. Thus allowing logic to be implemented on the client side. The asynchronous requests only return partial pages in JSON [4] format. Both MPA's and SPA's allow Cross-platform functionality and no installation of other libraries and technologies [24]. However, SPA's enabled Client-State Management (CSM). CSM is the technique of storing data on the client's browser in the form of for example cookies [5]. When discussing different frameworks and their functionality it is important to distinguish between SPA and MPA as they (among other things) address state management differently [14]. This thesis will primarily focus on SPAs as they have become ubiquitous in today's development landscape.

2.2 Current landscape of frontend development

From the latest surveys (2022), it is quite clear that React dominates the development landscape [26]. In a study that compared the proficiency of frameworks, React had a usage of around 82% [26]. The other two big frameworks Angular and Vue.js have a usage percentage of 49% and 46% respectively [26]. No matter which of the three popular options is chosen the developer can create smooth SPAs based on the principles of hydration, templating, and component orientation [28]. These three practices fit perfectly into JavaScript-based development.

2.2.1 Hydration

Hydration is used when initializing web apps efficiently. The process starts with the server sending the requested HTML to the client-side [20]. The website will, to the user, appear interactive but is when it comes to functionality inactive at this stage. When the HTML is sent the JavaScript files containing the necessary functionality follow and the scripts are executed with the chosen framework taking over and rebuilding the application on the client side and restoring functionality [20]. Subsequently, the application will work more or less as a client-side rendered application and the hydration process will only be done when initially rendering the application. Hydration is in a sense a hybrid solution between client-side rendering and Server-side rendering [20], but it still has deficiencies.

[4]JavaScript Object Notation) is a data-interchange format used for sending structured data between a server and a client.

2.2.2 Templating

Templating is the practice of using reusable structures in libraries like Backbone.js that define the layout of the page with the data or functionality of the page. It is an example of the practice of ”Separation of Concerns” as it keeps the presentation layer (HTML and CSS) and logic layer separate [14].

The interactivity in a web page is handled by the DOM (Document Object Model) [2]. The DOM is an object-based representation of the HTML file as a JS object. Changing the DOM subsequently changes the HTML. This is an ingenious way to update the HTML because it allows for dynamic updates. However, the DOM does have performance limitations when the application becomes more complex. For example, ponder the application created in 3.4. If we use traditional DOM manipulation adding one todo or completing a todo will result in rebuilding the entire list. If the list grows large it results in unnecessary work. This is especially true if we have multiple components inside the list are manipulated.

Modern web frameworks like React have addressed this by using a programming concept known as Virtual DOM [7]. The Virtual DOM is an in-memory representation of the actual DOM rendered by the browser. However, when changes are made in the UI rather than the framework making corresponding changes to the DOM it registers the changes in the virtual DOM. After the update, React compares this with the real DOM to identify the differences and only updates the necessary parts of the DOM [7]. In the todo example, this would decrease the DOM manipulation needed if small changes are made to a large DOM. Vue.js has a similar implementation to React. Angular however uses something called change detection due to it being designed in a tree-like structure however it still performs efficient DOM updates [1].

2.2.3 Component orientation

Component orientation development is an approach that focuses on creating separate independent components thus encouraging modularity. This practice has a lot of advantages including testability and enhanced collaboration [21].

2.2.4 Current popular libraries

Vue.js and React are libraries (not frameworks) because they don’t offer all the critical applications by default. However, These ”libraries” will become the main driver when running web applications. They require different libraries to implement features like routing, animation, and state management [4]. When it comes to React, examples of libraries that handle this are React Router for routing, React-spring for animations, and

React Redux for state management [22]. This implementation keeps React minimal to its nature but makes the application rely on other libraries that might differ substantially from each other [4]. It requires the developer to have a fundamental understanding of web development and make decisions on scalability and maintainability early in the development process. The same is true for Vue.js. Angular however provides a fully-fledged working solution by default [28].

2.2.5 HTML First

HTML First principles prioritize the capabilities of modern web browsers and HTML's attribute syntax to facilitate the easier, faster, and more maintainable development of web software [13]. By prioritizing HTML, these principles aim to broaden accessibility, thus making it easier for people to begin developing [13]. The key practices used to achieve this are; preferring Vanilla approaches, using libraries that support HTML attributes, avoiding build steps, and allowing the user to see how the source code is built [13].

2.2.6 HTML markup oriented state managers

An HTML markup-oriented state manager is a framework that allows the functionality of traditional frameworks such as state management, templating, and component creation to happen inside the HTML file. I will select the most suitable state managers for this evaluation based on a comprehensive list in the Sidewind documentation [25]. The options are provided in Table 1.

2.3 Selection of HTML-oriented state managers

According to the criteria stated in the introduction, I want to evaluate more frontend-focused libraries. htmx is more about enhancing existing server-rendered HTML pages with dynamic behavior in a non-intrusive manner [30], rather than completely overhauling the frontend architecture. Therefore I will not be picking htmx as it has a server-centric solution.

Amp-bind is an interesting state management solution. There are two main reasons for not selecting amp-bind. Firstly, amp-bind is a component of Accelerated Mobile Pages (AMP) [10]. Because I want to evaluate this framework individually it is hard to evaluate amp-bind. Secondly, the AMP technology might be dying. While Google, the creator of AMP, officially continues to endorse them especially news sites and platforms like X are moving away from AMP [17]. Google has also discontinued the ranking system that supported AMP [17].

Name	Description
Alpine.js	Lightweight JavaScript framework designed for building dynamic web applications with minimal overhead. It provides a declarative syntax for handling common frontend tasks.[9]
amp-bind	feature in the AMP project which implements state management, data binding and mutation in the HTML. [10]
htmx	Offers AJAX, CSS and Server sent events in HTML. Has similar functionality as Alpine.js but also server integration. [30]
Mavo	Introducing functionality to HTML using a DSL. This library does not use any JavaScript and all functionality enhancement is done using the DSL. [18]
petite-vue	Similar syntax to Vue.js but allows some functionality/interactivity to be built in HTML [31]
Nue JS	Similar to Alpine. However has a larger file size due to its broader feature set, including reactive data binding, component-based architecture, state management, and routing [3].

Table 1: HTML markup oriented state managers

Mavo is selected because of a multitude of factors. It is the only state manager on the list that does not support JavaScript in HTML. Functionality wise everything is done by the attributes [29]. The target group of Mavo is narrow which makes the architectural decisions interesting. Mavo is designed to make web development more accessible to people who are comfortable with HTML and CSS but may not have experience with JavaScript or backend programming, especially targeting the replacement of CMS solutions [29]. The review also focuses on development practices improvements made by state managers and as Mavo is very accessible to beginners it will be interesting to analyze the studies related to beginner satisfaction of Mavo [23].

Based on the requirements outlined in the introduction and the brief descriptions of these frameworks, I have decided to select Alpine.js, Petite-vue, and Mavo. They align more closely with the criteria of enhancing interactivity within HTML markup while maintaining a frontend-focused approach. Alpine.js and Nue.js align well with the requirements outlined in the introduction. However, they are very similar frameworks and I have decided to select the more popular one. According to the study referenced in 2.2 Alpine.js has a usage percentage of around 6% [26]. Nue.js is also currently in a very early stage of development [3].

3 Alpine.js - Lightweight framework for state management in HTML

Alpine.js is a lightweight (7 kB gzipped) JavaScript framework designed for building dynamic web applications with minimal overhead. It enables the user to add behavior and interactivity to the static HTML pages rendered on the server [9]. Alpine.js provides a declarative syntax for handling common frontend tasks, such as DOM manipulation, event handling, and data binding [9].

3.1 History

Alpine.js was created by Caleb Porzio in late 2019. Porzio aimed to construct an application where it was possible to create simple reactive applications written directly into HTML without the use of JavaScript [19]. While jQuery[5] existed as a solution for adding interactivity, he felt that it was too imperative when it came to state management and wanted something like jQuery but with the reactive aspects that modern solutions like React offer.

3.2 HTML First

The goal is to address how effectively frameworks like Alpine improve the development practices when it comes to development from an HTML First perspective. Alpine breaks the rule of ”Separation of concerns” which the HTML First manifest endorses to increase readability and improve the coherence of the application [13]. Nevertheless, consolidating all functionality within the HTML may lead to decreased readability, as it could become cluttered with numerous operations.

HTML First also discourages build steps as it often results in compatibility and usability issues when it comes to development and deployment. Alpine.js does not have a build step. Taking these things into account Alpine.js aligns well with the original aspirations of HTML First development [13]. From a developer's perspective this results in more understandable code and thus a lower barrier to entry into frameworks like this. Furthermore, the development experience without the build step is more seamless and should when employed successfully decrease the cost and complexity of building software. Syntax is also borrowed from Vue.js and Angular directive which for experienced developers brings a certain level of familiarity [32].

[5]jQuery is a JavaScript library that can handle events and AJAX interactions using its own syntax [6].

3.3 Use cases

Alpine.js takes a minimalist approach to web development and is a great substitute for aging frameworks like jQuery [8]. jQuery is still the third most used web framework among professional developers according to a Stack Overflow study [6] done in 2023. The main reasons that developers are selecting jQuery are related to backward compatibility, extensive availability of plugins and libraries, simplified DOM manipulation, AJAX support, and jQuery Core [15]. For developers using this framework for these reasons and similar ones, Alpine.js provides solutions for almost all of these use cases. Alpine.js allows the developers to add custom directives by using Alpine.js directive [9], however, it doesn't offer as many plugins as jQuery. Additionally, discussions are underway regarding the potential implementation of custom directives and "core events" from other libraries [15]. The main difference is jQuery works using JavaScript and Alpine.js works on a HTML template.

jQuery core provides the interface for manipulating the DOM. The DOM manipulation in jQuery is inherently imperative and Alpine.js has a declarative way to achieve similar functionality. instead of directly manipulating the DOM imperatively, you declare the desired state of the DOM based on the data you have. This is done using x-bind in Alpine.js [9], which binds data to DOM elements. The x-ref attribute provides a way to reference DOM elements directly from Alpine.js components [9], similar to how you might use jQuery to directly access DOM elements [15]. Alpine.js also has AJAX support which is discussed in 3.7.

Alpine.js does not provide a traditional SPA solution. This is by design because SPA's involve a trade-off between initial page load time and subsequent navigation speed. SPAs load all the necessary assets upfront, which can lead to longer initial load times, especially for larger applications. Alpine.js, on the other hand, focuses on delivering a fast and lightweight solution that doesn't require a significant initial load overhead [9]. Alpine.js doesn't need hydration because it directly manipulates the static HTML served by the server, adding interactivity without the need for client-side rendering or hydration. This lightweight approach contributes to its simplicity and performance benefits.

Alpine.js enables developers to handle user interactions and dynamic content directly within the frontend code, reducing the need for frequent communication with the backend [8]. This can be advantageous in situations where backend access is limited. The other qualities already discussed make it usable for applications where backend accessibility is limited [8].

3.4 Installation

Alpine.js can be installed by including a script tag or importing it as a module (installing Alpine via npm and importing it). You are encouraged to use the Content Delivery Network (CDN) and therefore eliminate the traditional build step of web applications [32]. A simple todo application with the following features is implemented to demonstrate the Alpine.js library. The todo application can be located in Appendix A and the essential parts of the code are broken down when it is relevant.

- The user should be able to add todos
- The user should be able to check/uncheck todos
- The user should be able to perform additional functionality: clearing all, selecting all and clearing selected todos.

In the example application Alpine.js will be installed by including it in the script tag.

```
<script defer
    src="https://cdn.jsdelivr.net/npm/alpinejs@3.x.x/dist/cdn.min.js">
</script>
```

3.5 State management

Alpine allows you to declare local and global states. The local states can be declared using the `x-data` tag directly into the HTML. `x-data` works as a normal JavaScript object and thus the tag allows for the usage of methods [9]. The tag also allows other `x-data` tags to be nested inside the elements. In the case of the todo app, we will need a variable that handles the todos, an input variable for the text input, and a boolean value for displaying additional functionality [9].

```
<div x-data="{ todos:[], input: '', additionaldata:false}">
...
</div>
```

Any Alpine code that is run inside the `div` will have access to these variables. Everything will react and automatically update the DOM when the data changes, thus no further state management will be necessary. However, state management for globally declared variables is different. For global state management, the `$store` method should be utilized. For example, global state management could be necessary when implementing dark mode into a web application [9].

3.6 Templating

Every directive in Alpine.js can parse JavaScript expressions. The following are the most useful when manipulating DOM. In the todo application `x-for` is used to display the todos.

Name	Description
`x-html`	Parses html data based on the defined variables
`x-show`	Makes it possible to show and hide DOM elements by changing the CSS
`x-if`	Similar to x-show but completely removes the html
`x-for`	Can be used to display a list of variables

Table 2: Caption

```
<template x-for="todo in todos">
    <div>
        <input type="checkbox" x-model="todo.completed">
        <span
        x-text="todo.name" x-bind:class="{'completed': todo.completed}"
        ></span>
    </div>
</template>
```

3.7 Events

Event listeners are implemented using the `x-on` attribute followed by the event you want to invoke [9]. Popular events include `onclick` and keyboard invocations. Alpine allows you to use the JavaScript event object `$event`. Furthermore Alpine permits the creation of custom DOM events with customized dispatchers. This is for example done in the todo application when a todo is added.

```

<form x-on:submit.prevent>
    <input x-model="input" placeholder="Add Todo.." type="text">
    <button x-on:click= "if(input.length>0){todos.push({name:input,
   completed:false}); input=''}">
        Add
    </button>
</form>
```

Alpine has child-parent communication that allows the child of a certain component to inherit the values from the parent [9]. However, we often want our components to be able to communicate with each other in a more general fashion. This can be done in three

ways. Either by using a custom JavaScript event or using the dispatch attribute [32]. The HTML First manifesto advises against the latter option, advocating instead for leveraging HTML attributes from external libraries whenever feasible[13]. The dispatch attributes allow us to easily register events in other components or in related applications [32]. In the case of our todo application, If we want an added ”todo” to be sent to a database it could be done by using the dispatch attribute.

```
<!-- Adding dispatch to the todoinput -->
$dispatch('todoAdded', { todo: input })
<script>
    window.addEventListener('todoAdded', function(event) {
        console.log('Todo added event detected!');
        console.log('Event Detail:', event.detail);
    });
</script>
```

3.8 Additional limitations of Alpine

There are architectural and technological limitations to Alpine.js which could make development more expensive if it isn't addressed and evaluated before beginning development. Alpine.js does not provide an SPA solution and thus it has severe limitations when it comes to more complex projects [9]. For a large-scale project with intricate functionality on the front end, you may benefit by choosing a traditional framework like React or Vue.js. These frameworks come equipped with features such as client-side routing, state management libraries, server-side rendering (SSR), and code splitting that Alpine.js does not have. If your application requires advanced user interactions, extensive routing, and state management requirements you should probably not pick Alpine.js [8]. Traditional frameworks also often provide extensive performance optimization [8].

4 Mavo - Simplifying Web Application Development with HTML Extensions

Mavo is an HTML-markup-oriented state manager focused on creating web applications using only HTML. One of the key architectural features that separates Mavo from other frameworks is that it only utilizes HTML attributes to create reactivity [18]. If developers want functionality that isn't available in Mavo the Mavos JS API can be utilized to construct plugins [29].

Mavo is intended by its developers as a solution for creating web pages that interact with datasets [29] Simple applications that interact with datasets are more broadly known as CRUD (Create, Read, Update, Delete) applications [29].

Mavo aligns well with the main HTML First ideas by utilizing built-in features of modern web browsers, taking advantage of existing HTML attributes, and by allowing the users to view the source code [13]. The creators specifically state their desire to create a framework that allows for the reusability of the existing HTML functionalities [29]. The ViewSource affordance permits other developers to look at other code straight from the webpage thus improving the code shareability [13].

4.1 Replacing Content Management Systems

Mavo is a solution designed to replace Content Management Systems (CMS) in some situations. A CMS is a solution where the user can graphically edit or create for instance web applications. The idea with CMS is that the user should not need to write any code themselves. Mavo merges the capabilities of CMS applications with regular front-end applications when it comes to adding, editing, and removing data. This is done by editing widgets that are available as loadable components directly into the HTML [29]. Based on these principles "Mavo Create" was extended on this framework and designed as a "What You See Is What You Get Editor" (WYSIWYG) built on top of GrapesJS [11].

Connell's study on CMS usage in academic libraries [12] found varying results when considering user satisfaction. They found that the most common CMS solutions were Drupal and WordPress [12]. The study found that CMS users are slightly more satisfied than non-CMS users (54% vs 47%). There are still however a large level of complaints with current CMSs due to their limited customization and the graphical interfaces that can be illogical in their design [12]. The level of people who understand HTML and CSS has as discussed in 2.2.6 increased and Mavo could therefore help to solve some of these complaints. The CMS solutions also provide a heavy-weight solution in situations where the developer only wants to do something simple [29].

4.2 Installation

Mavo is installed by downloading or linking to its CSS and JS files. A simple to-do application with the following features is implemented to demonstrate the Mavo library with the same features as in 3.4. The to-do application can be located in Appendix B and the essential parts of the code are broken down when it is relevant. Mavo can also store data locally or be connected to a cloud service such as Dropbox. Below is an example of installing Mavo into a project.

```
<head>
    <link rel="stylesheet" href="https://get.mavo.io/mavo.css"/>
    <script src="https://get.mavo.io/mavo.js"></script>
</head>
```

4.3 Templating

In Mavo you can declare variables inside the `mv-app`. The `mv-app` attribute is needed to create a Mavo application [18]. Defining variables is done by adding the `property` or `itemprop` as an HTML attribute[29]. These elements are variables that can be used inside the `mv-app` and stored in the data store. Using the data-store one can select if the variables should be connected to any BE system. In the example application, the completed variable is created by adding the tag `property="completed"` [18]. Below is a table that explains the main templating features of Mavo.

Name	Description
`mv-app`	Initiates the creation of a Mavo application and handles connections and storage.
`property`	Defines variables as an HTML attribute that can be used inside the mv-app
`itemprop`	Another attribute option for defining variables to be used within the mv-app

Table 3: Templating tags for Mavo [18]

4.4 State management and rendering

As previously stated, The `mv-app` attribute is needed to create a Mavo application [18]. However, the `mv-app` attribute by itself only initiates the creation of an app and then the connections and way of storage are handled by the attributes in the table.

```
<div mv-app="todoApp" mv-storage="local" mv-mode="edit" mv-bar="none">
```

```
    ...
</div>
```

Name	Description
`mv-bar`	A simple bar that is displayed at the top with save and clear options by default
`mv-storage`	Handles connection to any BE database and how elements should be added/re-read
`mv-format`	Declares the data format
`mv-autosave`	Extension of mv-storage where time until save can be added
`mv-uploads`	facilitating the management and handling of uploaded files

Table 4: State management attributes [18]

Mavo provides editing widgets and customizable editors by default. The configuration and design of the editor depend on the HTML object chosen. [29]. Mavo improves on how you edit things on a webpage by making smart use of HTML tags. For example, if you're editing a date shown as plain text (like with `<span>`), you'll get a basic text box. But if it's shown using the `<time>` tag, you'll get a nice date or time picker. This makes it more likely for people to use the right HTML tags for their content [29]. In the case of the todo application, I didn't want to display the bar as it adds unnecessary functionality for the user. I enabled the edit to allow the todos to have the standard functionality associated with it. I wanted this application to be similar to the applications created by Petite-vue and Alpine.js and I therefore selected storage to local [18].

Initial rendering is done in Mavo by recursively fetching the data that should be displayed [29]. During editing or rendering, whenever an object is created, it maintains a reference to its corresponding node in the Mavo tree, improving performance by avoiding redundant operations. When data within an object changes due to rendering, editing, or expression evaluation, Mavo re-evaluates expressions within it or refers to it to reflect the updated values [29]. This re-evaluation occurs within a special execution context where current object data and Mavo functions appear as if they were global scope.

Mavo employs ECMAScript 2015 Proxies to selectively fetch descendant or ancestor properties only when necessary, enabling efficient handling of DOM manipulation [29]. Figure 1 explains how the DOM-handling tree looks for the todo application [29].

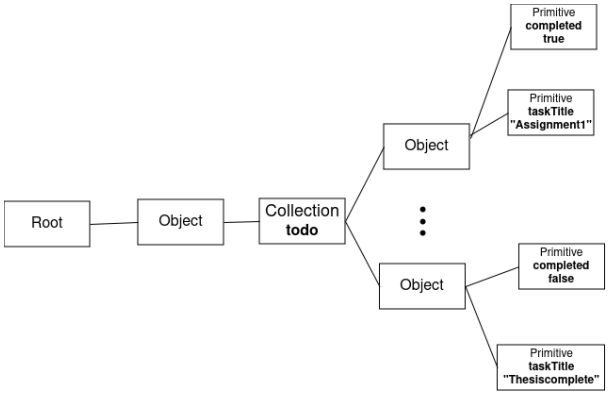

Figure 1: Mavo rendering for the todo app

4.5 Events

The `mv-action` is used to define actions or behaviors that can be triggered by user interactions, such as clicking a button, and these actions manipulate the data and DOM in the Mavo application, enabling functionalities like saving data, updating values, or doing specific actions [18]. Actions that can be performed are adding, setting moving, or deleting the variables that are defined inside the mv-app [18]. The `mv-action` is used to add the additional features to the todo application

```
<button class="clear-completed" mv-action="delete(todo where completed)
    ">
     Clear completed
</button>
<button class="clear-all" mv-action="delete(todo)">Clear all</button>
<button mv-action="set(todo.completed, true)">Complete all</button>
```

4.6 Current limitations

This technology is supposed to offer more customization and accessibility than contemporary WYSIWYG editors and CMS solutions. While it succeeds in being a viable HTML-oriented state manager, users still need to grasp HTML fundamentals for effective utilization [29]. Consequently, Mavo is not accessible to all users.

A study [29] done by the developers of Mavo is promising with some caveats. They recruited 20 participants from local web design groups where 19/20 identified as ”Intermediate or below in JS”, 13/20 as ”beginners or worse in any programming language” and 7/20 ”considered themselves intermediate or better in any programming language”. They gave the participants a 45-minute introduction to Mavo [29]. They had the participants create two simple applications and the success rate for the basic CRUD functionality was 100%. The reason for this was attributed to the similarities with regular

HTML.

The main issue that this study found is that the participants had a hard time understanding events and implementing expressions correctly [29]. This part of the Mavo framework currently bears the most resemblance to JS. A participant said the following *"That's some math and logic which are not my strong points. Just seeing those if statements...I did a little bit of Java and I remember those always screwed me over in that class. No surprise that that also tripped me up here"* [29]. According to this study in the cases where the existing editor can not be utilized the `mavo-action` attribute for creating events is still feared by beginners.

Mavo does not provide an SPA solution, it has severe limitations when it comes to more complex projects and computationally more intensive projects [29]. SPA frameworks like React, Vue.js, or Angular offer robust solutions. The calculations and objects are rendered each time they are needed. This might not be a problem for small apps. Even for big apps, Mavo could be handy if servers are utilized to show only the small bit of data each user needs at a time [29].

5 Petite-vue - Lightweight Vue.js library for HTML-Centric Development

Petite-vue is a lightweight JavaScript framework designed for building dynamic web applications with minimal overhead. It enables the user to add behavior and interactivity to the static HTML pages rendered on the server [31]. Petite-vue provides a declarative syntax for handling common frontend tasks, such as DOM manipulation, event handling, and data binding [31]. In this way, it is very similar to Alpine.js. It is however half the size of Alpine.js and has a similar syntax to Vue.js [31].

5.1 HTML First

Petite-vue aligns with the rule of ”Separation of concerns”. It encourages that the functionality should be in the HTML. This is shown by the todo application located in Appendix C. The syntax and structure of the Petite-vue are very similar to the Alpine.js.

Petite-vue does not have a build step when developing web applications. Petite-vue therefore aligns well with the original aspirations of HTML First development. From a developer's perspective this results in more understandable code and thus a lower barrier to entry into frameworks like this. Additionally, the development experience without the build step is more seamless and should decrease the complexity of building software.

5.2 Installation

Petite-vue can be installed by including a script tag or importing it as a module (installing Petite-vue via npm and importing it). You are encouraged to use the Content delivery network (CDN) and therefore eliminate the traditional build step of web applications [31]. A simple to-do application with the same features as in (2.2.6) is implemented to demonstrate the Petite-vue library. The to-do application can be located in Appendix C and the essential parts of the code are broken down when it is relevant.

```
1 <script src="https://unpkg.com/petite-vue" defer init></script>
```

5.3 State management

Petite-vue allows the developer to declare global and local states. Local states can be declared by using the `v-scope` attribute. The v-scope attribute works in a similar way to `x-data` in Alpine.js. Global state management uses the reactive command. That can be inserted in the script part or in Vue files to signify that a certain variable should

be recognized inside the HTML. The reactive method always declares the singletons as globally accessible variables independent of the v-scope they may be called in [31].

```
<div v-scope="{ todos: [], input: '', showcompleted:false}"></div>
    ...
</div>
```

5.4 Templating and Events

In Petite-vue events and templating can be handled in a similar way to Alpine.js. The main difference is that the attribute names are different. An example from the todo app below will demonstrate this.

```
<input
    placeholder="Add Todo.."
    v-model="input"
    @keyup.enter="if(input.length>0){todos.push({name:input, completed:
   false}); input=''}"
  />
  <button @click="if(input.length>0){todos.push({name:input, completed:
   false}); input=''}">
    Add
  </button>
```

```
<ul v-for="todo in todos">
  <input type="checkbox" v-model="todo.completed">
  <span v-html="todo.completed ? `<s>${todo.name}</s>` : todo.name">
  </span>
</ul>
```

Petite-vue also supports special lifecycle events for each element. Listening to the `vue:mounted` and `vue:unmounted` events for each element, enables event handling for the specific elements in the DOM [31]. When using the `createApp()` function, these events can be utilized to manage the lifecycle of the components and their associated events. Additionally, the `v-effect` directive can be used to define reactive effects for specific DOM elements, providing further control over event implementation within Petite-vue [31].

5.5 Similarity to Vue

Petite-vue is similar to Vue.js especially when it comes to template syntax, reactivity, and ecosystem. Both utilize a reactivity system, which allows for the automatic updating of the DOM when the data changes [31]. This provides a similar mental model for handling data and interactions. Petite-vue provides some Vue-compatible template

syntax, enabling developers to use familiar Vue template constructs. Petite-vue is intended to be part of the Vue ecosystem, aligning with standard Vue behavior whenever possible. This allows for a smooth transition to standard Vue if needed [31].

Petite-vue was created by Evan You and the idea was to develop an HTML-state manager that was compatible with the Vue syntax. Similarly to Alpine.js, the main use case marketed by its creator is adding JS functionality directly into HTML. The goal of the creator was to create a framework where that made it possible to progressively enhance HTML web pages [31].

Petite-vue is partly compatible from a syntax standpoint but it differs from Vue.js in some ways. Compatible features such as text bindings, event handling, conditional rendering with `v-if/v-else`, iterative rendering with `v-for`, and others. However, it isn't compatible with certain Vue.js functionalities like `ref()` and `computed()` methods, render functions due to the absence of a virtual DOM, reactivity for collection types such as `Map` and `Set`, and components like Transition, keep-alive, `<teleport>`, and `<suspense>` Vue is adaptable and can be used either with or without a build step. Petite-vue is designed to be used without a build step. However, all applications not using build-steps are from a performance and technical standpoint not equal. Petite-vue avoids overhead by attaching effects directly to DOM elements in a similar way to Alpine.js. Standard Vue without the build step is less optimal, both from a speed and size perspective (13kB) due to template compiler overhead especially for smaller applications [31].

5.6 Current limitations

The framework is currently not receiving updates as the last change in the GitHub repository was made over two years ago [31]. This may introduce potential security risks as well as slow adoption of changes in the web development landscape. Furthermore, it does not have as many active contributors or stars on GitHub as something like Alpine. The author also emphasizes that users encountering bugs are advised to find workarounds or contribute fixes [31]. Feature requests are unlikely to be considered at this time, as the project aims to remain minimal in scope [31].

No tool is perfect for every job and as Vue.js does not provide SPA solution, it has severe limitations when it comes to more complex projects [31]. For large-scale applications with intricate UI requirements, extensive data management needs, and complex routing logic, SPA frameworks like React, Vue.js, or Angular offer robust solutions.

6 Comparison

All three analyzed solutions are HTML-markup-oriented state managers and they are therefore fundamentally similar. All frameworks develop a solution on how to add functionality to HTML. However, the implementation and scope of the solution are different. Below is an overview that mentions the main areas of the analyzed framework.

Area	Alpine.js	Mavo	Petite-vue
Syntax	Declarative for adding interactivity JS support	Declarative attributes define data properties, actions, and controls.	Declarative for adding interactivity Vue.js support
Features	Reactive data binding (`x-on`), DOM manipulation (`x-show`) JS expressions support	Declarative data-driven web applications (No JS support) automatic data storage and sync,	Reactive data binding. Similar to Alpine.js
Community support	26k stars on GitHub	2.8k stars	8.7k stars
Learning curve	JavaScript for events	Only HTML needed	Familiarity with Vue needed
Flexibility	Replaces existing solutions like jQuery. Very flexible	Aimed at solving the CRUD problem	Same as Alpine. Supports Vue.js syntax
Size	7 kB (gzip)	12 kB (gzip)	4 kB (gzip)
Maintenance	Last release $<$ 1 week	Monthly changes in GitHub	Two years since the last change in GitHub (2024)

Table 5: The sources for Alpine [9], Mavo [18] and Petite-vue [31]

6.1 HTML First development

The frameworks analyzed break the common rule of ”separation of concerns”, allowing reactivity to be written directly into the HTML. Petite-vue and Alpine.js have different syntaxes but are declarative [9], aligning with HTML practices. Even though it is encouraged to leverage the default capabilities of the browser Alpine.js and Vue.js support JavaScript and Vue.js syntax respectively [8]. This allows the user to use the provided language to accomplish features that the browser supports by default. The main idea of this implementation is to allow users to use the benefits of languages like Vue.js

and JavaScript while remaining committed to the ideals of HTML First. Mavo is completely different in this regard. Mavo does not allow other scripting languages like JavaScript to be written inside the HTML. This makes Mavo very limiting and users might feel that certain features are lacking. Mavo is on the other hand a great choice for developers who want to use HTML and design applications that align with the CRUD mentality. Furthermore, Mavo allows seasoned developers to write JavaScript classes to add attributes [18]. All the frameworks support code shareability because attributes can be viewed by using web developer tools. Mavo takes shareability one step forward as disabling JavaScript allows the complete view of the exact implementation straight from the web developer tools.

6.2 Technological similarities and differences

All the analyzed frameworks can be built without a complex build step. This makes them great for quick prototyping [8] of web pages. They are all designed to be lightweight HTML-state managers and they therefore lack some features. For example, none of the analyzed frameworks provide an SPA solution and by that extension, the frameworks have severe limitations especially when it comes to features needed in more complex projects. The analyzed frameworks aren't suitable for large-scale applications where features like intricate UI requirements, extensive data management needs, and complex routing logic are needed.

All the analyzed frameworks support HTML attributes to some extent. When using Alpine.js or Petite-vue JavaScrip is needed for doing reactivity or data manipulation [8]. Mavo focuses on creating web applications using only HTML and without writing JavaScript or JavaScript-like code [29].

In summary, while these state managers share the goal of enhancing interactivity within HTML markup, they differ in their approach, features, use of technologies, and intended audience. These differences should be considered when making a choice based on specific project requirements and development preferences.

6.3 Comparing performance of modern frameworks with Alpine.js

In this section, Alpine.js is compared with Vue.js and React. The reason for not comparing the other analyzed frameworks is a lack of reliable data. Petite-vue might have similar results to Alpine.js as they are technologically similar frameworks [31].

The performance of modern frameworks deepened on a multitude of factors. A framework benchmark study [27] on reactivity and performance done by Krause compares the most common frameworks with each other. The benchmark does large table operations and

records the time in milliseconds needed to complete it. The most recent benchmark is selected for evaluation. The benchmark is rum on a MacBook Pro 14 (32 GB RAM, 8/14 Cores, OSX 14.3.1) running Chrome 122.0.6261.69 (arm64). The benchmark compares frameworks that are keyed and non-keyed. In the keyed mode, using a unique identifier for each data item ensures that any update to the data reflects in the associated DOM node, even if the list is reordered. In non-keyed mode, changes to data items may modify DOM nodes associated with other data, potentially improving performance by avoiding costly DOM operations [27]. However, depending on requirements, the non-keyed mode can either enhance performance or cause issues, so careful consideration and compatibility checks with the chosen framework are necessary. The benchmark has multiple libraries and variations of Vue, and React. Selected for evaluation are react-classes-v18.2.0, vue-v3.4.18 and Alpine.js. From the benchmark it is found that the modern frameworks have

Benchmark	Vue (ms)	React (ms)	Alpine.js (ms)
`create rows 1000`	43.4	46.2	106.2
`replace all rows 1000`	47	50.9	126.6
`partial update every 10 row`	19	21.4	22.3
`highlight selected row`	4	5.1	34.9
`swap 2 rows`	20.1	164.7	33.5
`removing one row`	18.3	16.6	24.3
`clear rows 1000`	14.9	16.1	52.5
`append rows 1000 new`	48.1	51	119.1

Table 6: Benchmark [27] created by Krause.

similar performance. React is according to this benchmark inherently bad at swapping rows in a list with a result of 164.7ms compared with the 20±0.1 ms from Vue.js. The performance of swapping rows in a list depends on various factors, including the specific implementation, the complexity of the components, and the efficiency of the underlying rendering mechanism. One reason why React is different from Vue and Angular when it comes to DOM updates is that in addition to having the concept of the virtual DOM React handles changes in the DOM by using a Diffing Algorithm [7]. React's Virtual DOM reconciliation process involves comparing the previous and current representations of the Virtual DOM to determine the minimal set of changes needed to update the real DOM [7]. While this approach is generally efficient, in scenarios involving frequent updates to large lists, React may struggle to optimize the diffing process efficiently. While Vue.js also employs a similar diffing algorithm to React, it uses a more aggressive approach called "Virtual DOM Patching" that applies patches directly to the real DOM as soon as changes are detected in the virtual DOM [8]. In the case of Alpine.js, it is slower than React or Vue.js.

The directives of Alpine.js do directly interact with the DOM which makes it simple for the developer to understand [8] but is one of the causes of these performance limitations shown in the benchmark[27]. According to this benchmark, Alpine.js demonstrates lower responsiveness, particularly under heavy DOM manipulation. Especially in the experiment, where 1000 rows were manipulated. For basic tasks, Alpine.js may offer quicker and more efficient performance compared to Vue.js because it doesn't utilize a virtual DOM thus reducing overhead. In this case, the benchmark was done on such a large application that this possible effect was not visible. The conclusion is that Alpine works well but cannot match the performance of Vue.js or React in complex applications [8].

7 Discussion

The analyzed frameworks[6] generally excel when it comes to projects with a lower level of complexity. The previous chapters have examined the frameworks more closely and this chapter addresses the main findings and compares the findings with contemporary frameworks. Furthermore, alternative technologies like Mavo Create are discussed.

7.1 Compatibility of the analyzed frameworks with the practice of HTML First development

The analyzed frameworks follow HTML First requirements well and have the majority of features outlined in the HTML First manifesto [13]. For example, HTML-state managers avoid the build steps to support the ability of the user to view the source code in the browser. All the analyzed frameworks leverage HTML attributes and Mavo uses only HTML attributes [18] when implementing functionality.

React on the other hand utilizes JSX for templating, which allows developers to write HTML-like syntax within JavaScript files [4]. JSX provides the opportunity to write HTML while incorporating functionality. One problem with this implementation is that it often discourages using HTML functionality that is part of the ”Vanilla” experience [13]. Because developers think in a JavaScript fashion this can result in unnecessary code being written in JSX and therefore lead to less use of the vanilla features of HTML. React follows the rule of ”separations of concerns”. This might be great for larger projects, but can make code more unreadable for beginners. React also has a build step that increases complexity when developing applications.

Alpine.js aligns well with the main HTML First requirements. This alignment does present a challenge in some scenarios of development. From a development perspective if there are a lot of developers working on the same project the practice of ”separation of concerns” is often a good thing. It allows the developers to independently work on different parts of the application and decreases the number of merge conflicts when working. Alpine.js's way of having the functionality and HTML in the same document increases the risk of conflict when having multi-developer teams working on related tasks. Thus Alpine.js should be preferred when the team and project complexity is smaller or templating components should be used.

[6] Alpine.js Mavo and Petite-vue

7.2 The steep learning curve of React versus analyzed frameworks

React has a steep learning curve, especially for developers new to component-based architectures and concepts like JSX and virtual DOM [7]. However, once understood, React's patterns and practices can be applied to large-scale applications effectively. The analyzed HTML-oriented state managers have a more gentle learning curve than React. For example, only HTML, CSS, and basic knowledge of JavaScript are needed to be able to use Alpine.js. Petite-vue is similar and the main differentiator is its compatibility with the Vue.js syntax. To develop with Mavo only HTML and CSS are needed. This is a conscious decision of its creators as they wanted a CMS-like solution for people who understand HTML.

7.3 Level of community support for frameworks

The analyzed framework with the biggest and growing community is Alpine.js. While it may lack some advanced features and integrations available in React, it offers lightweight interactivity out of the box. The second framework analyzed, Mavo was originally developed by MIT and it still has an active community. Petite-vue on the other hand no longer has an active community as the last major update was two years ago. In my opinion, this is unfortunately often the case with new HTML-oriented state managers and frameworks in general. If they lack large support from big groups of people or massive tech conglomerates they often have a hard time getting a significant market share. Furthermore, a large amount of frameworks are created every year intensifying competition for adoption and recognition.

7.4 Performance analysis of Vue.js, React, and Alpine.js

Currently one of the main limitations of the analysis was the failure to find performance data on Petite-vue and Mavo. Comparisons to React were therefore not possible but because Petite-vue has similar DOM manipulation to Alpine.js the performance might be similar. In this thesis, Alpine.js was compared with React and Vue.js. Alpine.js had less optimal performance compared to React and Vue, especially in scenarios with frequent updates or complex UI interactions. The expected slower initial loading times of Alpine.js and Petite-vue due to their smaller overhead compared to React and Vue.js were not confirmed in the thesis. Thus, for simpler UIs and smaller projects, the performance impact may be negligible.

Alpine.js is a lightweight JavaScript framework designed for simpler interactions and behaviors within the DOM. It's focused on adding interactivity directly to HTML using declarative syntax. Alpine.js doesn't require a build step or complex setup, making it

more straightforward for small to medium-sized projects. The analyzed frameworks are generally great for adding interactivity to static and server-rendered websites. They can also be a great tool for prototyping a solution quickly.

In summary, when it comes to technological requirements the choice between analyzed solutions or something different depends on a multitude of factors; like project requirements, developer preference, and features needed. The analyzed frameworks excel in creating interactivity for smaller projects or enhancing existing web pages.

7.5 Mavo Create as a substitute for HTML-oriented state managers

Mavo Create might be an appropriate solution for developers who want something like a CMS solution while still being able to utilize all the benefits of regular Mavo. Mavo Create is a WYSIWYG (What You See Is What You Get) editor similar to WordPress. Mavo Create is built on on the existing GrapeJS editor [11]. The main benefit of Mavo Create is that it still allows the users to develop the applications further in Mavo. A user study [11] found that four out of five users would use Mavo Create if they needed to create an application but they felt like Mavo Create lacked the possibility of creating more advanced applications. Lastly, solutions like this could be a substitute for existing templating technologies.

8 Conclusion

In this thesis, I reviewed HTML-oriented state managers and assessed their impact on development practices, technological advantages, and their alignment with the HTML First development approach. The reviewed state managers were compared with mainstream JavaScript frameworks. Based on this background, I sought to find answers to the following questions: *How can HTML-markup-oriented state managers improve the development practices when it comes to HTML First development?* and *What technological advantages/disadvantages do HTML-markup-oriented state managers have compared with current solutions?*.

A severe limitation of the thesis is the lack of scientific research due to the rapid change in the computer science field. Subsequently, the sources are often documentation and potential opinion pieces.

For the purposes of this study, HTML-oriented state managers Mavo, Alpine.js, and Petite-vue were selected. While the selected frameworks are HTML-oriented state managers they focus on inherently different features and should be used in different situations. To illustrate the differences between the frameworks, a *todo app* was created.

Alpine.js was analyzed and compared with React and Vue.js. The evaluation of Alpine.js highlighted its minimalist approach to web development, However for larger applications React and Vue.js reign supreme from an efficiency standpoint due to their extensive optimization technologies like Virtual DOM.

Mavo was examined for its potential to enhance development practices in an HTML First approach and especially compete with current CMS solutions. Studies[29] showed that people familiar with CMS solutions were able to create simple CRUD applications using Mavo.

Petite-vue was determined to be a solution that is similar to Alpine.js. The main differentiation factor of Petite-vue is its compatibility with the Vue.js syntax. The largest problem with Petite-vue is the inactivity of its maintainers.

Although I was able to analyze the suitability of these frameworks for enhancing interactivity within HTML markup, their implications for traditional software practices, and the performance of Alpine.js several open questions remain:

- What is the performance of Mavo and Petite-vue?
- How usable are Alpine.js and Petite-vue for beginners? (Usability study)

References

[1] "Angular - angular.io," https://angular.io/docs, [Accessed 17-04-2024].

[2] "Document Object Model (DOM) — ionos.com," https://www.ionos.com/digitalguide/websites/web-development/an-introduction-to-the-document-object-model-dom/, [Accessed 20-04-2024].

[3] "Nue: The content-first web framework — nuejs.org," https://nuejs.org/, [Accessed 17-04-2024].

[4] "React — react.dev," https://react.dev/, [Accessed 13-04-2024].

[5] "Single-page application vs. multiple-page application," https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58, [Accessed 17-04-2024].

[6] "Stack Overflow Developer Survey 2023," https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe-prof, [Accessed 13-03-2024].

[7] "Virtual DOM and Internals – React," https://legacy.reactjs.org/docs/faq-internals.html, [Accessed 17-04-2024].

[8] Alex, "Vue.js vs Alpine.js: The Ultimate Javascript Framework Battle," https://codingshortcuts.com/vue-js-vs-alpine-js/, [Accessed 13-03-2024].

[9] Alpine.js. [accessed 24.01.2024]. [Online]. Available: https://alpinejs.dev/

[10] "amp-bind - documentation," https://amp.dev/documentation/components/amp-bind, [Accessed 07-02-2024].

[11] F. R. Cicileo, "Mavo crete: A wysiwyg editor for mavo," Massachusetts Institute of Technology, Tech. Rep., 2018.

[12] R. S. Connell, "Content management systems: Trends in academic libraries," *Information technology and libraries.*, vol. 32, no. 2, 2013-06-09.

[13] T. Ennis. HTML First. [Accessed 15.01.2024]. [Online]. Available: https://html-first.com/

[14] G. Fink and F. Ido., *Pro Single Page Application Development Using Backbone.js and ASP.NET*. Apress Media, 2014, information taken from the first chapter.

[15] D. F. Hugo, "Alpine.js: The JavaScript Framework That's Used Like jQuery, Written Like Vue, and Inspired by TailwindCSS," https://css-tricks.com/alpine-js-the-javascript-framework-thats-used-like-jquery-written-like-vue-and-inspired-by-tailwindcss/, [Accessed 22-02-2024].

[16] M. Kaluza and V. Bernard, "Comparasion of front-end frameworks for web applications development," Zbornik Veleučilišta u Rijeci 6, Tech. Rep., 2013.

[17] S. Karst, "Are Google Accelerated Mobile Pages (AMP) Dead in 2024?" https://hurrdatmarketing.com/web-design-news/whats-happening-to-google-amps/, [Accessed 17-04-2024].

[18] "Mavo: A new, approachable way to create Web applications — mavo.io," https://mavo.io/, [Accessed 07-02-2024].

[19] M. Melanson, "Alpine.js Brings JavaScript Interactivity without Complexity to HTML — thenewstack.io," https://thenewstack.io/alpine-js-brings-javascript-interactivity-without-complexity-to-html/, [Accessed 31-01-2024].

[20] J. Miller and A. Osmani. (2019) Rendering on the web. [Accessed 29.01.2023]. [Online]. Available: https://web.dev/articles/rendering-on-the-web#client-side_rendering

[21] O. Nierstrasz, S. Gibbs, and D. Tsichritzis, "Component-oriented software developmment," *Association for Computing Machinery*, 1992.

[22] Reactrouter. [Accessed 24.01.2024]. [Online]. Available: https://reactrouter.com/en/main

[23] M. B. Rosson and J. Ballin, "Who, what, and how: A survey of informal and professional web developers," *IEEE*, 2005.

[24] E. Scott, *SPA Design and Architecture: Understanding Single Page Web Applications.* Shelter Island: Manning, 2016.

[25] "Sidewind - related approaches," https://sidewind.js.org/#related-approaches, [Accessed 07-02-2024].

[26] State of js. [Accessed 26.01.2023]. [Online]. Available: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

[27] K. Stefan, "GitHub A comparison of the performance of a few popular javascript frameworks," https://github.com/krausest/js-framework-benchmark, [Accessed 13-03-2024].

[28] J. Vepsäläinen, A. Hellas, and P. Vuorimaa, "The state of disappearing frameworks in 2023," 2023.

[29] L. Verou, A. Zhang, and D. Krager, "Mavo: Creating interactive data-driven web applications by authoring html," 2016-16-10.

[30] w3ird0h, “htmx: Big Wins for State Management - htmx,” https://medium.com/@w3ird0h/htmx-big-wins-for-state-management-3f8f5bf327f7, [Accessed 07-02-2024].

[31] E. You, “Github - petite-vue,” https://github.com/vuejs/petite-vue/tree/main, [Accessed 06-02-2024].

[32] K. Zagoris, “Cross-component Communication Patterns in AlpineJs - Witty Programming — wittyprogramming.dev,” https://www.wittyprogramming.dev/articles/cross-component-communication-pattern-in-alpinejs/, [Accessed 12-02-2024].

A Todo application in Alpine.js

```
<head>
<script defer src="https://cdn.jsdelivr.net/npm/alpinejs@3.x.x/dist/cdn.min.js"></script>
<style>
    .completed{text-decoration: line-through;}
</style>
</head>
<body>
<div x-data="{ todos:[], input: '', additionaldata:false}">
    <h3>Things to do!</h3>
    <template x-for="todo in todos">
        <div>
            <input type="checkbox" x-model="todo.completed">
            <span
                x-text="todo.name"
                x-bind:class="{'completed': todo.completed}"
            ></span>
        </div>
    </template>
    <form x-on:submit.prevent>
        <input x-model="input" placeholder="Add Todo.." type="text">
        <button x-on:click=  "if(input.length>0){todos.push({name:input, completed:false}); input=''}">
            Add
        </button>
    </form>

    <button x-on:click="additionaldata = ! additionaldata">
        Additional functionality
    </button>
    <div x-show="additionaldata">
        <button x-on:click= "todos= todos.filter((todo)=>!todo.completed)">
            Clear completed
        </button>
        <button x-on:click="todos= []">Clear all</button>
        <button x-on:click="todos.forEach((todo)=>todo.completed=true)"
        >Complete all
        </button>
    </div>
</div>
</body>
```

B Todo application in Mavo

```
<head>
    <link rel="stylesheet" href="https://get.mavo.io/mavo.css"/>
    <script src="https://get.mavo.io/mavo.js"></script>
</head>
<body>
    <h1> My tasks: </h1>
    <div mv-app="todoApp" mv-storage="local" mv-mode="edit" mv-bar="
    none">
        Things to do:
        <ul mv-list >
            <li property="todo" mv-list-item >
                <label>
                    <input property="completed" type="checkbox" />
                    <span property="taskTitle"></span>
                </label>
            </li>
        </ul>
        <fieldset>
            <legend>Additional options</legend>
            <button class="clear-completed" mv-action="delete(todo
    where completed)">Clear completed</button>
            <button class="clear-all" mv-action="delete(todo)">Clear
    all</button>
            <button mv-action="set(todo.completed, true)">Complete all<
    /button>
            <meta property="height" content="50" mv-mode="read">
        </fieldset>
    </div>
</body>
```

C Todo application in Petite-vue

```
<script src="https://unpkg.com/petite-vue" defer init></script>
<div v-scope="{ todos: [], input: '', showcompleted:false}">
  <h3>Things to do! </h3>
  <ul v-for="todo in todos">
    <input type="checkbox" v-model="todo.completed">
    <span v-html="todo.completed ? `<s>${todo.name}</s>` : todo.name"></
   /span>
  </ul>
  <input
    placeholder="Add Todo.."
    v-model="input"
    @keyup.enter="if(input.length>0){todos.push({name:input, completed:
   false}); input=''}"
  />
  <button @click="if(input.length>0){todos.push({name:input, completed:
   false}); input=''}">Add</button>
  <button @click="if(showcompleted){showcompleted=false} else {
   showcompleted=true}">Additional functionality</button>
    <div v-show="showcompleted">
        <button @click="todos= todos.filter((todo)=>!todo.completed)">
   Clear completed</button>
        <button @click="todos= []">Clear all</button>
        <button @click="todos.forEach((todo) => todo.completed = true)"
   >Completed all</button>
    </div>

</div>
```