Hello Framework! A heuristic method for choosing front-end JavaScript frameworks

Mitra Bauer
Abstract

Hello Framework! A heuristic method for choosing front-end JavaScript frameworks

Mitra Bauer

This thesis investigates how to decide what JavaScript framework to use when the front-end world is constantly changing and new technologies are added. Choosing the right framework can save a developer or company time, money and reduce complexity. An interview study with developers was done to extract what is important in a framework. The interview study concluded in six themes regarding what to investigate when choosing a framework along with a checklist and in-depth questions. Based on the results from the interview study together with a test application design, four web applications were implemented in the JavaScript frameworks React, Angular, Vue and Svelte. This was followed by a qualitative evaluation to see if this investigative approach provided meaningful insights.

The test application results show that the combination of in-depth questions and design specification gave a basic technical understanding of the strengths and weaknesses of the frameworks together with an understanding of their history and current development. The goal of this thesis is to provide more ways of comparing frameworks in the front-end communities and academic world.
1 Introduction

In the world of JavaScript there has been many new frameworks created for the purpose of helping developers create better applications. Each new framework comes with their own background, set of rules and structures. How do we know which framework to choose in this vast array of frameworks? As a developer, what are the different criteria we need to be aware of when making this decision? Why are there new frameworks constantly on the rise and what makes a framework die or live?

While some projects are still using pure JavaScript, it might not always work for the given user requirements in a project. Advanced topics such as state management, routing, data handling and manipulation are some of the problems being solved for developers by using frameworks. Many of today’s frameworks are adopting single-page application principles which enables users to navigate easily between different web pages without waiting for them to reload whole new pages.

There has been quite a bit of work done in benchmarking the performance of JavaScript frameworks but not as much investigation into what other aspects of a framework is interesting to look at from a developer’s perspective. This thesis is about how to think when choosing a front-end framework for a project and is set up in two parts. Part one is an interview study done with experienced developers to extract what their experiences with different frameworks are and what qualities are important. The interviews are followed up by an analysis resulting in listed criteria and in-depth questions. In the second part a test application was built in the top four most used JavaScript frameworks to see how well the frameworks answers some of the criteria and in-depth questions, as well as understanding their similarities and differences. The test application and frameworks are evaluated and followed by discussion about the applications, frameworks and the interview study. The work will result in more ways to evaluate a framework and a checklist, in-depth questions and specification are available in the appendices.

1.1 Purpose

The goal of this thesis is to give insight into what is sought after in a JavaScript front-end framework, mainly from a developer’s perspective, and to suggest which framework of today’s top 4 most used frameworks matches the developers’ expectations. The test application together with the interviews resulted in a checklist, list of in-depth questions and test application specification that is provided by the end. Another purpose is to add a hypothetical way to inspect and choose frameworks, contributing to the front-end communities and other academic works related to this field.
1.2 Delimitations

Delimitations for this thesis is that the tested frameworks are JavaScript focused and the test applications were developed for the web platform. The versions of the frameworks were the latest stable releases as of May 2021.
2 Background

Front-end development is a fast-paced field of work where all types of helpful libraries are constantly being developed. Many of the frameworks today are written for JavaScript, today’s most used scripting or programming language that developers use to create interactive web pages and applications. Different frameworks and libraries have different purposes. Some are for structuring the styling in a more efficient way. Some are just to help developers structure their JavaScript code in a set of rules with functions abstracted and working in the background. For small to mid-sized projects one might not have to use any libraries but they turn out useful when there are user requirements involved and a more complex architecture planned out. There are also options for cross-platform development meaning one code base will work on multiple platforms and operating systems.

2.1 Concepts within JavaScript development

Component architecture. Components provide a way to organize an application. Different frameworks may come with different ways to set up a component’s logic, HTML and styling.

Document Object Model (DOM). The DOM is the data representation of the objects that comprise the structure and content of a document on the web. It represents the document as nodes and object in a way so programming languages can connect to the page, serving as a programming interface for HTML and XML documents. Programs can change the document structure, style and content.

Single-page application (SPA). SPA works in the way that parts of a web application are being rerendered on user input instead of reloading entire pages, which results in faster transitions and a more smooth application experience.

State management. An application that allows users to show, add and remove content needs to track and update the underlying data. This underlying data is known as state. State can for example live locally in a component or kept in some kind of global state that components or pages can access in certain ways.

Reactivity. Every time state or data changes the user interface (UI) needs to be updated and match the data. Different frameworks have different syntax and methods to make the UI reactive.
Reusability. Code reuse has many benefits and allows you to write less code. Other benefits are better readability of code, better structure of code and reduced testing effort. Ways of reusing code could be extracting classes and methods when opportunities arise, resulting in less errors or inefficiencies. The avoidance of duplication through extracting methods makes code more maintainable as any future modifications only need to be applied in one place and prevents inconsistency when there are multiple sections of code that needs to synchronize changes. [9]

Routing. There are several ways of navigating in a web application. Server-side routing lets your browser communicate with a server and fetch new content to display, which makes the URL in the address bar change. The browser remembers the navigation history and allows for navigation back and forth.

Modern web applications or SPA load a single HTML shell and continually update the DOM without navigating users to new addresses on the web. When an SPA is complex and renders enough unique pseudo-webpage views, it’s important to bring routing functionality. When routing is handled in the client application, it’s called client-side routing. [8]

2.2 Front-end frameworks

While you can still write JavaScript code without any helping frameworks, many developers today are working with frameworks as a tool to help them manage code structure, performance and reusability. From State of JS surveys [5], a survey answered by 23,765 developers the top four used JavaScript frameworks are React, Angular, Vue and Svelte. They are all adopting single-page application (SPA) principles.

2.2.1 React

The first version of React was released in 2013 and is owned by the social media company Facebook. [10] It is an open-source library and is maintained by both Facebook and its community of developers and companies. React includes state management and rendering state to DOM but does not include routing by default as well as certain client-side functionalities. It is the top most used framework for building JavaScript user interfaces in 2020 and ranked second place in satisfaction among users. [5]

React’s key feature is the composition of components and that components written by different people should work well together. Other design principles are common abstraction meaning they avoid to add features that can be implemented by users hence less useless library code. Stability, interoperability and good developer experience is also at the core of the React framework development. [11]
2.2.2 Angular

AngularJS started out as a side project of a Google employee[12] and was the second most used JavaScript framework in 2020 and ranked eighth place in satisfaction among users[5]. It was first released 2010 but was scrapped and completely rewritten in 2014-2015. The latter version was renamed to just Angular and is in fact a TypeScript based open-source library. TypeScript is a superset of JavaScript.

Angular suggests applying the single responsibility principle to all components, services and other symbols. One example is defining one service or component per file and making sure to define small functions. Angular has its suggested style guide to follow and their design principles can be understood by implementing the style guide."[13]

2.2.3 Vue

Vue.js or Vue is an open-source library made to be adoptable and is able to scale between a library and a framework depending on different use cases. The initial release was in 2013 and is maintained by the creator and a team of core members. It was the third most used framework for building JavaScript user interfaces in 2020 and also ranked third in satisfaction among users. [5]

2.2.4 Svelte

One of the most recent JavaScript frameworks is Svelte which was first released in 2016 and rewritten in TypeScript 2019. Svelte converts an application into JavaScript at build time rather than interpreting it at run time. In the State of JS 2020 survey it quickly got the highest satisfaction score, ranking number one over React, Angular and Vue. It is still on the rise, ranking as the fourth most used framework for building JavaScript user interfaces. [5]

2.3 Related work

There has been a few studies done on the performance of different JavaScript frameworks but so far not many that are including Svelte. A study on React, Angular and Vue had the research question Which JavaScript framework is most suitable in terms of performance, modularity and usability for a single-page application? and came to the conclusion that React performed best due to its startup time but was very similar in terms of performance. React also stood out in modularity, but Vue was the suggested framework when it came to documentation and learning curve. [4]
Yet another study from 2019 was investigating React, Angular and Vue. In terms of popularity and community React ranked highest. Vue took first place when it came to performance and learning curve. Angular was suggested as the most fitting framework when it comes to building large scale applications. [16]

Going back one more year to 2018, another study looking at React, Angular and Vue suggested that the three do not stand far apart from each other. Angular was raised having a high TypeScript appreciation, emphasising guidance and structure and being object-oriented. Vue had a shallow initial learning process, emphasis on new popular technology and separation of concerns in one file. React was suggested to have a strong focus on using JavaScript, a large ecosystem and high importance of flexibility. [17]
3 Interview study

There is not much research done on how to choose frameworks holistically. Depending on what kind of project it is or what competencies there are in a team, choosing technologies can be tough - sometimes simply the more familiar framework is chosen because of initial efficiency and ease of use. There might not be time to look into a certain framework or spend time on learning it. But if there is time and interest, what approach can one take and what should one look for?

The goal of the interviews was to find some general questions to answer and topics to check when researching frameworks for both personal and company use.

3.1 Methodology

The interviews were qualitative semi-structured 30 minutes long sessions. Semi-structured interviews allow for more flexibility where the researcher can follow topics of interest during the interview without sticking to a structured set of questions, rather using the prepared questions as guidance to keep the interview on track. [18] The participants were interviewed one at a time totaling in four interviewees, held at the interviewees’ workplace. Most questions were prepared in advance but there was also space for general discussions and following up on statements. Questions can be found in appendix A.

3.1.1 Participants

The interview study was done with four developers between the age of 24 and 42 with several years of programming experience. All worked at the same company, two of which had previous experience from other companies. All were front-end development oriented. Gender is not relevant for these in-depth questions and hence is not reported.

3.1.2 Analysis method

The interview results were analysed using Thematic Analysis which emphasizes identifying, analysing and interpreting themes within qualitative data. [19] There are six phases of Thematic Analysis:

1. Familiarising yourself with your data: Transcribing, reading and noting down initial ideas.

2. Generating initial codes: Systematic coding of interesting features across the entire data set.
3. Searching for themes: Collating codes into potential themes and group the data.

4. Reviewing themes: Checking if the themes work and generating a thematic map.


6. Producing the report: Extracting examples, analysing them and relating back to research questions.

3.2 Results

The thematic analysis resulted in six themes. They are reported here in the term of what was said and questioned about each theme. Most of the questions are also applicable when researching other technologies.

Community. Whether you are stuck or having difficulties solving a problem or just want to learn more about a certain matter, it is helpful to have a well-established community - this is one of the most important factors. If you are looking to learn the framework properly it might be worth to look for good tutorials or online courses.

• How much support can someone find, both from official documentation and developers’ experiences?

• If the framework is available on GitHub, what does the activity and history look like? What can you see in the comments?

• What has been said on Stack Overflow [20] or other articles?

Quotes from interview:

... I would say the most important is community. To a certain extent also who is creating the framework, there’s a difference to take a small company’s framework or a smart person’s idea compared to Facebook’s framework.

... if there are many users you could kind of assume that the community is big enough and that’s a sign of that it (the framework) is being maintained.
Maturity. To check how developed a framework is might be a good idea - choosing a framework that is new or not entirely stable could end up being more work than expected. Some new frameworks do not have big communities, so it is up to you whether you like a challenge or want to have as much support as possible.

- Is the framework mature?
- Is it still being developed?
- Are there existing bugs?
- How many are working on the framework and are they developing it in the same direction?
- What is the code’s philosophy?
- How is the documentation and GitHub history or activity?

Quote from interview:

*I don’t invest a lot of time in something that hasn’t been around for a while, with React you understand it’s groundbreaking and here to stay.*

Technical aspects. Writing down your technical requirements is a good way to summarize what qualities you are looking for. In the interviews multiple people expressed that a good framework should save time, remove complexity and help them get things done.

- What are the functions and features of the framework?
- How is the code built?
- What platforms does it support?
- How are the debugging options?
- What is the size of the framework and does it matter?
- What plugins are there and is the framework expandable?

Quotes from interview:

*It’s more important to have the opportunity to do anything rather than using a certain framework, if it has restrictions you lose a lot.*
A framework that is good at everything isn’t the best at something. Then it might be better to use a few smaller libraries and create your own kind of framework.

A good framework is a small one with big possibilities. Efficient. It is flexible, easy to create own things instead of just following a track and if you don’t it all breaks.

A powerful framework is used by people and companies I can relate to. It solves a problem I’ve had for a long time in my career and that I never understood how easy it was to do. It is powerful and very few succeed in doing so, so it’s like "I’ll bring a new concept to do these things and it will save you time and make your life easier."

**Practical use.** Hands-on experience with a framework might be the deciding factor as a developer gets to see how it actually works. Maybe it suits them really well or they end up being more confused. A way to test it is to just try implementing something small. Sometimes you do not have a lot of time to spend on learning a framework and if a basic task feels too complicated, it might not be worth looking into the framework any more depending on what you need it for.

- Is it easy to work with and understand?
- Is it fast to work with?
- What disadvantages and advantages can you find?
- Is it supported by your current development environment?

Quotes from interview:

... after comparing the two frameworks we chose the one we were used to.

*Use whatever is easiest to use. The technology is secondary to the user experience if the frameworks aren’t completely useless, but when they are equally good, it feels like the most important is to choose whatever you can work fast with.*

*Is it easy to understand? I read the documentation in two hours, now I can start building. I find those kind of frameworks really great, those that are easy to understand and gets things done.*
I usually just take whatever seems coolest or most interesting after reading the documentation, but if you don’t understand the documentation on the first read it’s like - forget it, I will never test this. ... There can’t be magical stuff and if the most important parts lack documentation, then you know, this will be troublesome.

The most important part (in a framework) for me is how much time I win, in experienced time or real time, in a so called developer iteration. ... All tools that remove loads of troublesome difficulties like reloading, including new files, everything really that removes complexity and makes things go faster - that’s the tools I have adapted and worked with and makes me a more efficient developer. It’s the most important.

**Project requirements.** It is quite important to prepare requirements beforehand to see the size of the project and what resources it needs. You may end up finding that you need to do more than expected. If a change is needed after starting the project implementation, you might need to be flexible and change technology or requirements as needed.

- What will the project be used for?
- How will the framework help and what problems will it solve?
- Does it support your current technologies?
- If you are contemplating adding a framework to an existing project, is it easy to add and apply?

Quote from interview:

_I tend to argue for those that are, instead of these huge frameworks, smaller libraries. That are easier to switch, easy to add, because often you don’t start from scratch and things already exist. You have some code, or a system that already exists and if you add new functionality - you can’t just rewrite everything. However it easy to add a small part in a system._

**Company questions.** Choosing the right technologies can improve the product delivery time and make the product easier to maintain. A project often involves more than one person, which makes choosing technology not only a technical question but a personnel question. It might also be important to check whether the framework is open-source and what license it has to avoid future complications.
• How many in the team know the framework and how many would need to learn it?

• Can one work productively with the framework and will it affect production speed?

• What will it cost to use this technology?

• Will the code be reusable or shareable between projects?

• What current problems will it solve and does it support the current technologies that are being used?

**Summarized checklist of topics to investigate**

- Community
- Documentation
- GitHub history
- GitHub activity
- Functions/features of the framework
- Support for multiple platforms
- Plugins
- Tutorials
- Existing bugs
- Size of framework
- Possibility to expand framework
- Stack Overflow questions/activity
- Articles
- Support for your development environment, tools, IDE
- Debugging options
4 Test application design

To further look into the Technical Aspects, Practical Use and Community part of the interview study results a test application was built. The test application and other research resulted in a final checklist, in-depth questions and specification to use when investigating frameworks, available in appendices B, C and D.

The inspiration for the test application design is based on a combination of the results from the interview study and today’s most used social media applications which normally consists of some feed page and lists of data. The application design needs to address some in-depth questions from the interview study but also let the developer get enough experience with the framework to feel confident in making a technical decision. For this application the scope will be:

- Handling of libraries - How do we manage project libraries?
- Reactivity - In what way does this framework support reactivity?
- Reusability - What components are reusable, in what way?
- Component architecture - Is there a component architecture setup or recommended?
- Data handling - How do we fetch and handle data in this framework? How is data passed?
- State management - Is state management included and how does it work?
- Styling - How does styling in terms of scoping and conditionals work?

The base application was implemented in React, Vue, Angular and Svelte. All variants were built for the web platform in mind. The first application was implemented in React as a template app for the other frameworks. I chose to start with React because of previous work experience. As for guidelines and best practices, as much as possible was followed from official documentation and otherwise from other community resources such as Stack Overflow or GitHub threads. Setting up the projects was also done by official recommendations and documentation. The coding paradigm was functional if applicable. Since some frameworks were new in terms of my own experience I also drew inspiration from community websites.

Another way to test the test application design is to give a simple specification to a developer and see how well they manage to implement it. As for the application built in Vue, the specification was handed to another experienced developer to see how the proposed design was built and afterwards I gathered feedback for more input on the specification.
Finally, these are the questions from the interview study that should be answered by implementing the specification:

- Is the framework mature?
- Is it still being developed?
- Are there existing bugs?
- How is the documentation and GitHub history or activity?
- What are the functions and features of the framework?
- What platforms does it support?
- How are the debugging options?
- What plugins are there and is the framework expandable?
- Is it easy to work with and understand?
- Is it fast to work with?
- What disadvantages and advantages can you find?
- Is it supported by your current development environment?
- What will it cost to use this technology?

4.1 Suggested application specification

- Navigation bar:
  - Links to Home and Feed page where Home is shown at root path (/)
  - Theme toggle with theme value in a global state, available themes should be default and dark theme

- Home Page:
  - Title
  - Logo of some sort
  - Link to framework homepage

- Feed Page:
  - Feed component consisting of image, title and id or likes
– Simple error handling, showing a text if data fetching request fails
– Data fetching from some open-source REST API should be paginated and kept in a local state
– List of feed items with infinite scrolling and a loading spinner at the bottom, spinner should be from a third-party library
– Routing/navigation library if not built-in

Other guidelines:

• Coding style as functional as possible
• Follow project setup from official documentation
• Feel free to use inspiration from community websites
5 Evaluation

This section includes the evaluation of the test applications built in React, Angular, Vue and Svelte. Both similarities and differences will be presented and the test application design will be scrutinized. The way that this evaluation is done could be one way for developers to investigate frameworks in the future, which is one of the purposes of this thesis.

5.1 Methodology

A systematic evaluation will be done on the four test applications by looking at the application criteria from Subsection 4 one framework at a time. That includes handling of libraries, reactivity, reusability, component architecture, data handling, state management, styling and finally the interview study in-depth questions.

Since the Vue implementation was outsourced to another developer some questions can not be answered in the same way as the other three that I implemented myself. However outsourcing the implementation is important to answer questions about how clear the specification is for someone who has not yet implemented it and how well one gets to know a certain framework, evaluating whether the specification is valid enough for testing a framework.

5.2 Results

Before diving into the separate framework evaluations I will share what was common or the same in all the frameworks given the application criterias. This is to avoid repetition.

- Data handling: Used JavaScript’s Fetch API in all applications for fetching data from the open-source API and it worked fine. The API provided images, title and id.

- Handling of libraries: All frameworks used Node package manager (npm) for managing libraries.

- Building the theme toggle I used community resources for inspiration and all frameworks had articles on how to handle themes.

- Following the official documentations on setting up the application, each framework project came with a landing page from which I was able to reuse title, logo and link to framework website.

- As for debugging, I did not make use of any framework specific ways to debug. I only used console logging and a network tab in the browser.
All frameworks supported having an error state variable that could conditionally show an error text instead of the infinite scrolling list when a data request failed.

All frameworks are still being developed and have existing bugs but are still mature enough to handle the requirements of the test application.

All frameworks are free to use and under MIT license.

All frameworks only support the web platform by default.

5.2.1 React

Following the official documentation it was an easy project setup with Create React App, an officially supported way to create single-page React applications which offers a build setup with no configuration. I added a third-party library called react-router-dom for routing and navigation. For the infinite scroll I took inspiration from community resources and found a third-party library for the loading spinner using npm. Altogether the React application was a successful implementation since all specification requirements were met. After finishing the implementation the built bundle size was 235 KB and took 4.0 seconds to build. For a visual presentation of the application, see figure [1] and [2].

![Home page, default theme](image1)

![Feed page, dark theme](image2)

**Reactivity.** Using React’s state and prop variables allows components to update whenever some state has changed. In this test application states were held locally at page level and passed down as props to the smaller components.

**Reusability.** Components were reusable and mapped to data in the feed page. A separate file was created for the list item component and imported in the page file.
Component architecture. From own experience I decided to do a simple component architecture with a folder for pages and one for components. This was not set up from the start and had to be done manually.

Data handling. The data flow is unidirectional. The data was fetched from an open-source API, put in a local state for the Feed page and updated when new data was fetched since it was a paginated request.

State management. Local states were held in Feed page for the fetched data, loading and error handling. Global state for the theme was done with React’s Context API.

Styling. Create React App came with some default styling, I left some of it and added some new styles in a global CSS file that came with the project setup.

In-depth Questions

- **How is the documentation and GitHub history or activity?**
  The documentation and GitHub history is vast and activities like pull requests are regularly showing up. There was also a lot of community resources for the theme handling I researched. Currently there are 542 GitHub open issues and 192 open pull requests. In Stack Overflow, searching for questions with a react tag shows 313 294 results.

- **What are the functions and features of the framework?**
  In the test application I got to make use of React’s state handling, lifecycle methods, Context API for creating a theme provider and custom hooks.

- **What plugins are there and is the framework expandable?**
  Yes, there are a lot of third-party libraries and I added one for the loading spinner and one for routing. Searching for react in the npm package library shows 162 771 results.

- **Is it easy to work with and understand?**
  Yes but I have previous experience.

- **Is it fast to work with?**
  Yes because of previous experience.

- **What disadvantages and advantages can you find?**
  Advantages: Easy to split up components and creating custom hooks and providers to separate features.
Disadvantages: Could be that there is not a lot of suggestions on project architecture.

- Is it supported by your current development environment?
  Yes, I already had everything set up for React projects to work.

5.2.2 Angular

The official documentation for Angular proposed to use the Angular CLI (command-line interface) to setup a project. When creating the project one gets prompted if they want to use routing and what stylesheet format they would prefer out of CSS, SCSS, LESS and Stylus. I chose to go with the built-in routing and CSS. The Angular CLI generates a lot of boilerplate code when creating components, such as recommended architecture, base tests and CSS files. I used the `ng generate` command for creating modules, components and services.

Due to limited implementation time and difficulty to find community resources on how to build a infinite scroll list I decided to use the third-party library `ngx-infinite-scroll`. As for the loader spinner I could not find a library that supported Angular 12 which was the version setup by the Angular CLI. I found that the community resources were sometimes limited to a certain Angular version. In my experience there was not as many resources as for React and Vue. Since Angular was new to me I used a lot of community resources to get an understanding for best practices and common design patterns. After finishing the implementation the built bundle size was 709 KB and took 19.4 seconds to build. For a visual presentation of the application, see figure 3 and 4.

Figure 3: Home page, default theme  
Figure 4: Feed page, dark theme

Reactivity. The Angular CLI set up the project with RxJS by default, a library for reactive programming using Observables that makes it easier to compose
asynchronous or callback-based code. Whenever new data was fetched the components were updated.

**Reusability.** I could reuse and map components using the `*ngFor` keyword but the file structure is different from React and Svelte in the sense that I created the list item in the same template file as the feed page.

**Component architecture.** With the Angular CLI the whole architecture is set up for you and when creating new components or modules it organizes the whole structure automatically. The component is separated with logics in a TypeScript file, view in a HTML template file and styling can be separate or inside the template file.

**Data handling.** Angular has a two-way data binding syntax, data can flow in both directions. I did not make use of the two-way data binding syntax in the test application. Data for the Feed page was stored in the component TypeScript file and mapped in the HTML template file.

**State management.** Local states such as data, loading and errors states were handled in the Feed TypeScript file and inserted in the HTML template file.

**Styling.** The styling was set up to use separate CSS files but there is also the option to keep it directly in the template file. The styling is encapsulated into the component’s view and does not affect the rest.

**In-depth Questions**

- *How is the documentation and GitHub history or activity?*
  The documentation and GitHub history is vast and activities like pull requests are regularly showing up. You just have to make sure you are looking at the right Angular version. Currently there are 2075 GitHub open issues and 172 open pull requests. In Stack Overflow, searching for questions with a `angular` tag shows 255,558 results.

- *What are the functions and features of the framework?*
  A few of the functions and features I got use in the test application were Angular’s CLI, lifecycle methods, directives, dependency injections and Angular’s first party library for routing.

- *What plugins are there and is the framework expandable?*
  Yes, there are a lot of third-party libraries but a lot of them are bound to
a certain Angular version. Unfortunately I could not find a loading spinner for Angular 12, but I did add a library for the infinite scroll list that worked well. Searching for `angular` in the npm package library shows 50 845 results.

- **Is it easy to work with and understand?**
  Yes after some time but it took some rereading to grasp certain concepts. To me the learning curve felt steep. I had to go back to the documentation multiple times to grasp certain concepts, check syntax and how to structure my code.

- **Is it fast to work with?**
  Quite fast to work with when you get used to the rules and using the Angular CLI.

- **What disadvantages and advantages can you find?**
  **Advantages:** Angular CLI sets everything up for you and creates boilerplate code when adding components. Clear directions on code structure and how to split logic and view.
  **Disadvantages:** If you do not use the Angular CLI it will be a lot of boilerplate code to write. There is also some extra work to add custom elements.

- **Is it supported by your current development environment?**
  Yes, I already had everything set up for Angular projects to work.

### 5.2.3 Svelte

Following the official documentation and setting up a new project was easy. For Svelte I had to install a plugin for my editor VS Code to syntax highlight the code. To me the syntax was the most different among the four frameworks. I added a third-party library for routing called `svelte-routing`. Even though Svelte is the newest of the four there is many packages available and it was easy to find one for the loading spinner. After finishing the implementation the built bundle size was 156 KB and took 1.7 seconds to build. For a visual presentation of the application, see figure 5 and 6.

**Reactivity.** Svelte is using the dollar sign ($) for reactive declarations to indicate dynamic values that will rerender the component when state has changed.

**Reusability.** Components were easy to create, import and use in other files. For me it was a similar mindset to React.
Component architecture. The Svelte documentation had some recommendations - each component file can have a script tag containing the variables, lifecycle methods and other logic. This is followed by HTML code and CSS styling in a style tag, kept in the same file. Some logical functions such as contexts and stores are separated into their own files. This structure seems similar to Vue, but in Svelte you do not have to use the template tag to write HTML in the Svelte files as you do in Vue.

Data handling. As a general rule, data flow in Svelte is top-down - a parent component can set props on a child component, and a component can set attributes on an element, but not the other way around.

State management. State management is included in the framework by using stores and contexts. Contexts and stores seem similar but they differ in that stores are available to any part of an app, while a context is only available to a component and its descendants.

Styling. CSS rules are scoped to the component. However in my experience some computed styling did not change with the global tag used for creating or overriding global styles.

In-depth Questions

- How is the documentation and GitHub history or activity?
  The documentation is simplistic but covers most functions and features through a thorough tutorial. In GitHub there are pull requests regularly showing up but compared to Vue, Angular and React there was less activity.
Currently there are 628 GitHub open issues and 141 open pull requests. In Stack Overflow, searching for questions with a svelte tag shows 1870 results.

- **What are the functions and features of the framework?**
  In the test application I got to make use of Svelte’s lifecycle methods, reactive declarations, syntax for conditional rendering, stores and contexts.

- **What plugins are there and is the framework expandable?**
  Yes, there are a lot of third-party libraries and I added one for the loading spinner and one for routing. Searching for svelte in the npm package library shows 2929 results.

- **Is it easy to work with and understand?**
  Even though Svelte was quite different from my previous experiences the official tutorial made it easy to understand and learn. Learning curve was gentle. I found it easy to get the project running and implementing the specification.

- **Is it fast to work with?**
  Yes. Developer experience was good but the website crashed from time to time when reloading the page so I needed to rebuild the project.

- **What disadvantages and advantages can you find?**
  Advantages: Easy and fast to learn. Small bundle size. Good examples and tutorial on their official website. Simplistic and clear coding style.
  Disadvantages: Still quite new and less resources than the other frameworks.

- **Is it supported by your current development environment?**
  I had to install a plugin for syntax highlighting but other than that everything worked fine.

### 5.2.4 Vue

This application was outsourced to another experienced developer. The Vue CLI was used to initiate the project and you choose whether to use state management, routing, stylesheet framework, linting, compiler and more. Folder structure was also created by the CLI. The specification requirements were met, although the styling looked a little different from my implementations the functionality was the same. The built bundle size was 180 KB and took 1.2 seconds to build. For a visual presentation of the application, see figure 7 and 8.

The feedback from the developer was that they felt it was a clear enough specification to get a feel for the framework. It would be good to avoid unnecessary styling and the instructions for the error handling felt a bit unclear. The time it took to build the application was a bit longer than expected.
Reactivity. Using Vue’s state and prop variables allows components to update whenever some state has changed. In this test application states were held locally at page level and passed down as props to the smaller components.

Reusability. A few components were made and easily imported and reused in the page files.

Component architecture. The component architecture is similar to Svelte in the sense that logic, view and styling live in the same file. However the component setup looks quite different in the way methods, data, components and lifecycle methods are instantiated or injected into the template.

Data handling. Two-way data binding is supported for input components. Data was fetched in Feed page and kept in a local state.

State management. A store containing the states for the theme toggle was created in this project.

Styling. Vue has a scoped attribute one can use to scope the CSS styling to the file.

In-depth Questions

- How is the documentation and GitHub history or activity?
  The documentation and GitHub history is vast and activities like pull requests are regularly showing up. However since there is a new Vue version rolling out (3.0) the version used in the application (2.0) does not have the same frequency of pull requests opened. Currently there are 327 GitHub
open issues and 203 open pull requests. In Stack Overflow, searching for questions with a **vue** tag shows 79 642 results.

- **What are the functions and features of the framework?**
  State management, routing, lifecycle hooks were used in this project.

- **What plugins are there and is the framework expandable?**
  There are many plugins and libraries for Vue, for this project **vue-spinner** library was added. A routing library was already included in the ecosystem. Searching for **vue** in the npm package library shows 50 381 results.

- **Is it easy to work with and understand?**
  Not possible to assess due to outsourcing the implementation.

- **Is it fast to work with?**
  Not possible to assess due to outsourcing the implementation.

- **What disadvantages and advantages can you find?**
  Advantages: Allows partial adoption of functionality. Possible to build pure HTML and CSS pages.
  Disadvantages: Vanilla JS can be used but may affect reactivity. Using global state complicates the use of two-way data binding.

- **Is it supported by your current development environment?**
  I had to install a plugin for syntax highlighting.

### 5.3 Framework differences

Some of the differences between the frameworks were that Svelte and Vue has a similar component architecture in the sense that logic, HTML and styling live in the same file. An Angular component is split up into three different files, one for HTML, one for styling and one for logics. React gives you the freedom to choose whatever component architecture you want but in this application logics and HTML were in the same file and CSS was separated.

Svelte had scoped CSS styling in each file by default, while Vue has a **scoped** attribute one can use to scope the CSS styling to the file. React does not have scoped styling by default and the application was just using one global CSS file for all components and pages. Angular CLI set up the components with separate component-scoped CSS files.

Two-way data binding syntax were supported in Vue and Angular while Svelte has a **bind** directive for it.

Out of the two new frameworks I got to learn, Angular had a rather steep learning curve in comparison with Svelte that had a gentle one.
React and Angular projects were already supported by my editor while I had to install syntax highlighting plugins for Vue and Svelte.

Svelte had the smallest online community and React had the largest out of the four frameworks. Svelte also had the fewest available libraries and packages through npm, while React had the most. The Angular application was the only one that did not use a third-party library for the loading spinner because of version incompatibility.
6 Discussion

In these subsections I will discuss the results from the Interview study, Test application design and Evaluation and finally some discourse on which framework could be the better choice.

6.1 Interview study

First of all I would like to highlight some points that could affect the results. There might have been some bias in the responses since the interviewees all worked together at the same company, which could result in similar answers. Doing semi-structured interviews could sometimes result in discussions that had leading questions, which could also affect the answers and maybe something would be brought up in just one interview. In those cases it can be difficult to validate answers as something of heavy weight. Another bias could be the interviewees’ tendency to raise React as an example because of their strong experience with it. This could mean their responses and preferences already leaned towards something similar to the features and mindset of React.

I found the common theme to be that a framework should improve the developer experience with environments and tooling, in addition to the main point which is improving code quality and adapting functions and code patterns that feel easy and efficient to use. There was a big emphasis on community and having good documentation along with other community resources. It was interesting that also the philosophy and founder of frameworks were brought up, an ethical aspect of using technology. It could be a good idea to check who or what is behind a framework to know what we support by using it.

Outside of the technical aspects project requirements and company questions are important in choosing a framework. This might not matter too much if the project is personal but in a company setting choosing something that might not live up to expectations can cost time and money.

6.2 Test application design and Evaluation

Since I already had experience with React it was quite easy to implement the first application. I thought it would be mostly copying and pasting the code to create the other applications but each framework had their specific syntax and structures to follow so I had to spend more time learning about them than expected. React was quick in building the production build but had the second largest production build size. The production build sizes for Vue (180 KB) and Svelte (156 KB) were quite close though. The one that stood out the most was Angular with a production build size of 709 KB, even after optimizing the build command with
options to reduce the size. In terms of production build speed Vue was the fastest with 1.2 seconds.

With Angular I experienced it was more difficult to adapt their principles and also finding the right information, I wonder if that is because of my previous experiences being used to finding information related to React. I did also encounter a few community resource solutions that would not work in Angular because of versioning. Overall, when you got used to their style guide it got easier but still felt a little clumsy in comparison to React and Svelte. It was slow in building the production build (19.4 seconds) and had the largest production build size. Angular CLI solved a lot of boilerplate code which was quite nice though, it seems like it could scale well when a project is growing.

Svelte was very easy to learn and fast to work with, it will be interesting to see how it grows in the developer community. Even though Svelte is rather new and does not have a big community like the other three frameworks I found all the resources I needed to complete the test application design. Svelte was fast in building the production build and was the build size was smaller than the other three frameworks. For Svelte it would have been interesting to try SvelteKit, their latest framework for building web applications, but it is in beta stage. Maybe using SvelteKit would have resolved some of the reloading issues I encountered.

Since I did not implement the Vue application it was quite difficult to evaluate it more than saying that it met the requirements. I did not get to know the functions and features of the framework, but, I got to see the specification requirements met and that could be a sign of the test application design being successful in the end.

For the specification, two-way data binding was not compatible with the theme toggle input that was dependent on a global state. It could be a good idea to add something to the specification that makes use of two-way data binding, perhaps a form.

I chose not to add any code examples in the evaluation because of two major reasons. The first reason is that I felt like the actual code is not the focus of the thesis, rather more of a high-level evaluation of certain topics. The second reason is that with front-end development there usually are many ways to solve the same problems and with framework standards constantly changing it could mean my solutions would be obsolete in the near future.

To me the test application design was enough to get a basic understanding of the frameworks but there was definitely functions and features that I did not get to implement. In combination with the in-depth questions from the interview study I got a good understanding of their features, history and development activity. For a bigger study, a more elaborate specification that implements more of the framework features would be interesting.
6.3 Choosing a framework

Short summaries of the frameworks:

- React - Has the largest online community and strong backing. There are loads of libraries, plugins and a lot of GitHub and Stack Overflow activity. It ranked second place in satisfaction and first place in usage according to the State of JS survey.

- Svelte - Has the smallest community. The framework is new and fast, produces a small bundle size and tutorials are thorough. Learning curve is stable. Good for smaller projects but how will it scale in larger applications? It ranked first place in satisfaction and fourth place in usage according to the State of JS survey.

- Angular - Has a large community but ranked ninth place in satisfaction and second place in usage according to the State of JS survey. It has a strict set of rules and out of the four frameworks the only framework that enforces the use of TypeScript. Angular has the most open GitHub issues.

- Vue - Has a medium sized community. It ranked third place in satisfaction and usage according to the State of JS survey. Vue has the fewest open issues on GitHub out of the four but the most open pull requests which could be a sign of mature framework in development.

Most of the developers in the interviews wanted the freedom to have a smaller framework with the opportunity to expand it with other plugins or libraries. Considering that, maybe Angular would not be the best choice as it comes with a lot of strict rules. We can also see that it has lost popularity in the State of JS 2020 survey. Vue and Angular both have similar directives and component architecture, maybe Vue could be the more up-to-date choice between the two. React wins in the race of community support and available resources and could be the safest choice ranking high on usage and satisfaction. Svelte is a new contender, maybe more appropriate for personal or smaller projects since the community and online resources still are rather small. Open issues on GitHub is rather high compared to React and Vue that have been around for a longer time. The high satisfaction score could indicate that it will grow larger and more stable in the coming years.

Comparing the results with related work in section 2.3 they seem similar. According to the studies, React was best in the performance but Vue suggested as better when it came to documentation and learning curve. Another study claimed Vue had a better performance and raised React for being more modular. In comparison Angular did not stand out in any of these areas which supports the drop in satisfaction rankings. It will be interesting to see if Svelte will be included in more JavaScript framework comparisons in the future.
7 Conclusion

This thesis has provided a set of questions, checklist and specification to investigate frameworks. The suggested way to evaluate a framework would be to implement the specification in appendix D together with researching topics in the checklist and in-depth questions in appendices B and C.

Building the test application in the four frameworks React, Angular, Vue and Svelte gave knowledge about the functions, features and developer experience. The checklist and in-depth questions resulting from the interview study added to the understanding of non-technical parts such as community resources and activity. React seems to be a good and safe choice in regards to its community and online resources size, in combination with the high ranking in developer usage and satisfaction. Svelte has a gentle learning curve and is on the rise ranking at the top of the satisfaction score but still needs more community support to grow in usage and resources. Angular is still widely used but losing popularity in satisfaction rankings and has a lot of open GitHub issues. Vue has some elements similar to Angular but is ranking high in satisfaction and usage which could indicate that it is a more appropriate choice if you want to use a similar framework with less strict rules.

In the end, choosing a framework is about preference, experience and requirements. Each project needs to be evaluated on an individual basis. Hopefully this thesis can aid in the choosing of frameworks.

7.1 Future work

What could be further tested or researched?

- More frameworks could be included in the test application evaluation, maybe even testing cross-platform or native app frameworks.
- Investigate if there are other features that are important to test in addition to the ones in this thesis.
- Create an even more complex specification to be able to answer more in-depth questions and result in a more in-depth comparison of differences between frameworks.
- Outsource coding to more developers and collect more feedback about the test application specification.
- Extend the size of the interview study and get more data from developers with different experiences.
References


Appendices

A Interview Questions

About you

• How old are you?
• How long have you worked or done programming?
• What type of programming do you like the most?
• How long have you been at Company X?
• Do you have development experience another company?

General questions

• How do you choose new framework?
• What are you looking for in a new framework? What areas and what do you want to work for you?
• If you find two or more similar frameworks, how do you compare them?
• What is it that finally makes you choose a new framework?

What are you thinking about when you hear these words:

• Powerful framework
• Fidelity
• Developer experience
• Versatility

How important are these for you when you choose a technology from 1-5?

• Versatility
• Tools and IDE
• Debugging
• OS
• Developer Experience
• App quality
• Possible users
• App production cost
• Security
• Platform compatibility
• Easy to update
• Time to market
• Third party plugins
• Performance
• Community
• Documentation
• Easy to implement
• Fast to implement
• Reusability
• Easy to implement in existing project

B Checklist

Based on the interview study and results from the test app, these are some of the following things to check when deciding on what framework to use:

☐ Community
☐ Documentation
☐ GitHub history
☐ GitHub activity
☐ Functions/features of the framework
Support for multiple platforms

Plugins

Tutorials

Existing bugs

Size of framework

Possibility to expand framework

Stack Overflow questions/activity

Articles

Support for your development environment, tools, IDE

Debugging options

C In-depth Questions

• Is the framework mature? Are there no more major bugs?

• Is the framework still being developed? How many are working on it? What kind of people are developing it? Are they close or dragging the framework in different directions?

• What do people say about the framework?

• How is the code built? What is its philosophy?

• How is the documentation/GitHub history and activity?

• What are the functions and features of the specific framework?

• Is it easy to create own plugins? Is the framework limited somehow?

• Is the framework modular?

• What kind of project will the framework be used for?

• Will the code be shareable between projects?

• Is the framework easy to add and apply to an existing project?
• Try implementing something, what works well? Is it fast to use? Easy to work with?
• Is the code easy to understand?
• How much time will be saved using the framework?
• Is it open source? Licensed?
• Does it support your current technologies?

Company focused questions:
• How many (in the team) know the framework?
• How many would need to learn the framework?
• Can one work productively with the framework?
• Will the code be reusable and can it be shared between projects?
• What will it cost to use this technology?
• How is the framework going to help? What problems will it solve?
• Does it support your current technologies/libraries/frameworks?
• Will it affect production speed?

D Test Application Specification

• Navigation bar:
  – Links to Home and Feed page where Home is shown at root path (/)
  – Theme toggle with theme value in a global state, available themes should be default and dark theme

• Home Page:
  – Title
  – Logo of some sort
  – Link to framework homepage

• Feed Page:
– Feed component consisting of image, title and id or likes
– Simple error handling, showing a text if data fetching request fails
– Data fetching from some open-source REST API should be paginated and kept in a local state
– List of feed items with infinite scrolling and a loading spinner at the bottom, spinner should be from a third-party library
– Routing/navigation library if not built-in

Other guidelines:

• Coding style as functional as possible
• Follow project setup from official documentation
• Feel free to use inspiration from community websites