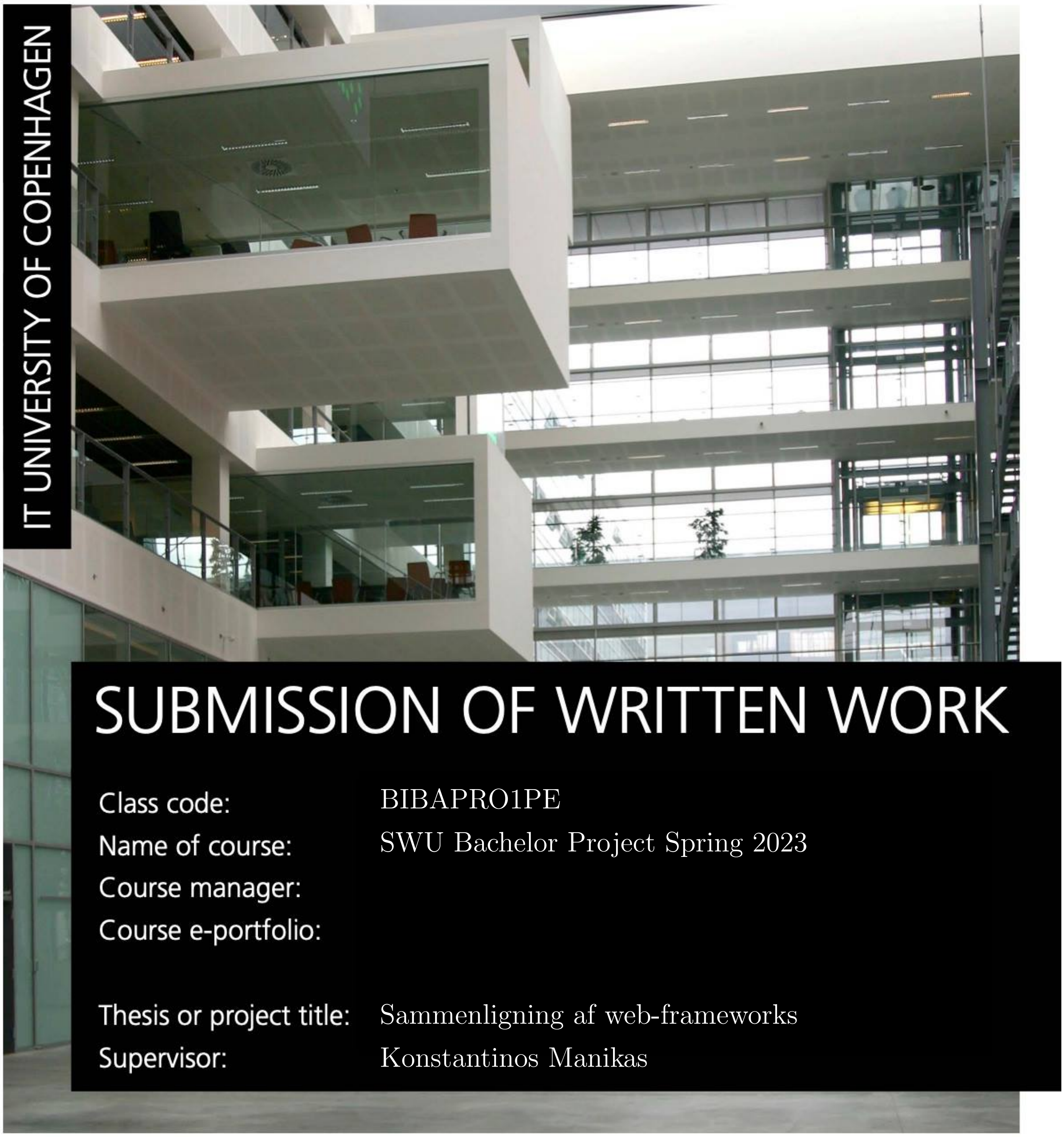


SUBMISSION OF WRITTEN WORK

Class code:	BIBAPRO1PE
Name of course:	SWU Bachelor Project Spring 2023
Course manager:	
Course e-portfolio:	
Thesis or project title:	Sammenligning af web-frameworks
Supervisor:	Konstantinos Manikas

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Deniz Isik	22/02-2001	deni@itu.dk
2.	Deniz Yildirim	02/05-2000	deyi@itu.dk
3.	Michael Hieu Tran	21/08-1999	mhtr@itu.dk
4.	Mikkel Lindgreen Bech	15/06-1999	milb@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

1 Abstract

This research paper dives into a detailed examination of four distinct JavaScript frontend frameworks. The main goal is to understand and evaluate these frameworks using several critical metrics. These include the learning curves associated with each framework, the speed of development, the available extensions, their performance capabilities, the surrounding community, and how easily the frameworks can be modified or extended.

In order to conduct this investigation, we designed and carried out four unique experiments for each of the frameworks. This amounted to a total of 16 individual experiments carried out over the course of our study. We paid careful attention to ensure that each experiment was executed using identical procedures by all four developers involved in the research. This approach was taken to maintain a high level of consistency across all tests and to guarantee the authenticity of our results.

After the completion of our experiments, we conducted an analysis of the results. Our intention was to determine if there was a particular framework that outperformed the others based on the chosen metrics. However, following an in-depth examination, it became clear that there is no one-size-fits-all solution. Each of the four frameworks we studied had its own set of strengths and weaknesses. We conclude based on our findings that the choice of a frontend JavaScript framework is dependent on the specific requirements and constraints of the project at hand. Each framework has its unique advantages and disadvantages, and understanding these can result in a more informed decision-making process.

All source code used during our tests for this project is available for access and can be found here:

- HTML and CSS Experiment Prototype
- All Experiments Conducted
- The Deployed Application, The API and API GitHub

Contents

2 Introduction

2.1 Introduction

In the world of software development, choosing the right framework can make a significant difference in the outcome of a project. The wrong framework can lead to wasted time, increased costs, and subpar results [1]. On the other hand, selecting the right framework can result in more efficient development, better code quality, and a successful project outcome. With so many frameworks available for developers to choose from, it can be challenging to determine which one is the best fit for a particular project.

In this report, we aim to explore the factors that developers should consider when choosing a framework and identify the most suitable framework for a given situation based on project scope and timeline. This leads up to our research question:

What is the most ideal framework for a developer to use in a given situation based on project scope and timeline?

2.2 Terminology

- **API**, or Application Programming Interface, is a set of protocols and tools that allow different software applications to communicate with each other. It defines a set of rules for, how two or more applications can interact, allowing developers to create complex and integrated software systems [2].
- **Endpoint** is a digital location on an API that allows the world to communicate with the API to send and retrieve data [3].
- **DOM (Document Object Model)** is a programming interface that represents a web page as a structured tree of objects. It allows programmers to interact with it and manipulate the content, structure, and style of a web page dynamically [4].
- **Virtual DOM** is a programming concept where an additional version of the DOM is kept in memory (the virtual DOM). This version is synced and compared to the real DOM, allowing changes only to be made where the two versions differ. This process helps make updates faster and more efficient [5].

- **Diffing** in the context of virtual DOM, is the process of comparing two virtual DOM trees to identify the minimal number of changes required to update the real DOM efficiently. It optimizes rendering by updating only the parts of the DOM that have changed, improving performance in web applications [6].
- **Tree Shaking** in the context of virtual DOM refers to the process of eliminating dead or unused code from the application bundle, resulting in a smaller and more efficient deployment package. It helps optimize performance by only including the necessary components and reducing unnecessary computations during rendering [7].
- **SCRUM** is an agile framework commonly used for software development that emphasizes short, iterative development cycles called sprints, during which a cross-functional team works to complete items from a product backlog [8].
- **Backlog item** is a single element of work that adds value to a product being developed [9].
- **React** is the most popular JavaScript frontend framework [10].
- **TypeScript** is a strongly typed programming language that is built on JavaScript. It compiles code to JavaScript while providing complete type safety [11].
- **Benchmark** is a test used to evaluate and quantify the performance of a computer, hardware, or software system. These tests provide the option to compare how well the performance compares to other products [12].

2.3 Problem definition

In today's fast-paced world, the landscape of technology and software is evolving at an unprecedented rate, making it increasingly challenging for developers to choose the best tools and technologies for their next project. This challenge is particularly evident in the web development space, where new frameworks are emerging at a rapid pace [13]. These frameworks, predominantly written in JavaScript, the most popular language in the world, have taken the web development world by storm [14].

The primary objective of a framework is to enable developers to build sophisticated and interactive web applications with greater efficiency. By providing a collection of prewritten code and tools, a framework helps developers save time and effort. The framework offers a set of predefined rules, structures, and patterns that developers can follow while developing an application [15].

The selection of a framework can be dependent on several factors such as performance, ease of use, security, scalability, familiarity, and compatibility with other technologies [16]. Balancing these factors and choosing the framework that meets all project requirements is a challenging task, as some frameworks might excel in some areas, but lack in others. It is crucial to select the right framework from the beginning, as it sets the foundation for the project scope and limitations. Changing the framework during development can be time-consuming, expensive, and not easily feasible. Eoin Woods, a software engineer with over 30 years of experience, is well known for his quote "Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be canceled [1]."

This poses a significant challenge as there are, to our knowledge, no existing tools or technologies that aid developers in selecting the optimal framework. Consequently, developers may resort to choosing a framework that they are most familiar with and understand its capabilities, even if it is not the best fit for their project [17]. Given the potential risks associated with choosing an inappropriate framework, it is imperative to select an ideal framework that is easy to learn and use, provides control over the application, offers relevant features, and is cost-effective.

This project seeks to evaluate and compare existing frontend frameworks based on various important metrics. The project will consider essential factors such as performance, learning curve, development speed, community, and extensions to provide an in-depth comparative study of the trade-offs between different frameworks. This will enable developers to understand how different metrics relate to each other and how the selection of a frontend framework can impact their project's success.

The ultimate goal of this evaluation is to help developers make informed and qualified decisions when selecting a frontend framework that aligns with their project scope.

2.4 Problem scope

The diagram below represents the entire cycle of product development [18] from the end user all the way to the part of our scope where developers internally select a framework to work with.

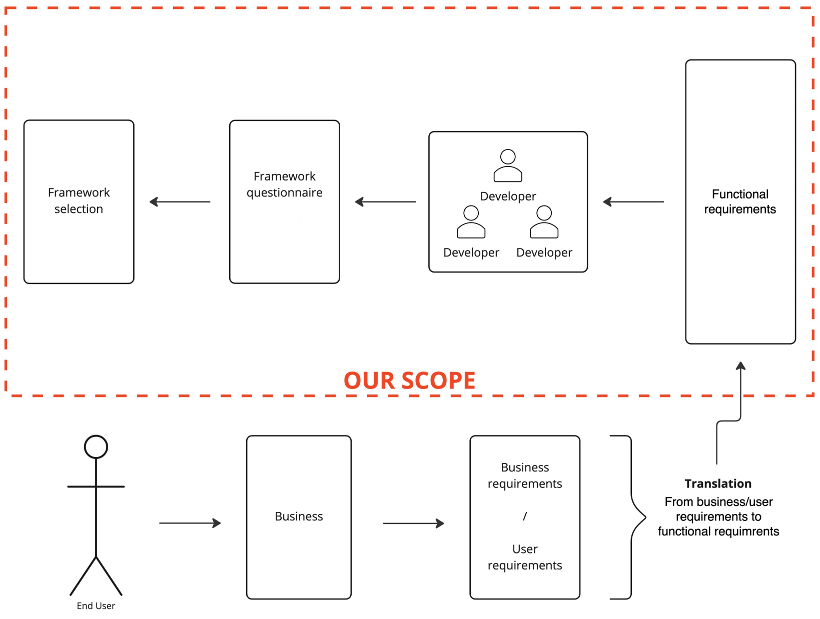


Figure 1: Our project scope in a product development cycle

The diagram divides the process into two sections. The lower section focuses on the business aspect, while the highlighted upper section centers on the development part.

Examining the initial stage of the diagram, the end user initiates a request to the business, or the business identifies a business opportunity in the current market. Subsequently, the business assesses the request and generates a business-oriented list of requirements that includes the intended functionality of the product and its target audience.

The list of business requirements needs to be translated into functional requirements before developers and technical experts can start working on the project [18]. The translation step has been found to be very challenging and crucial for the success of a new project [19]. The translation step could be a case study on its own, hence the reason why we are not focusing on it in this project scope.

The highlighted area is the main focus of our case study, where developers internally find the most appropriate framework to use for a given project based on the functional requirements.

2.4.1 Frameworks

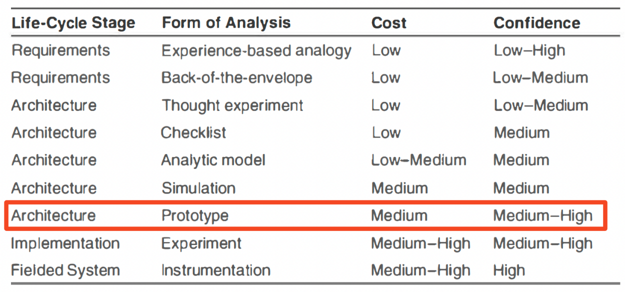
In this project, we will be testing and comparing a multitude of metrics for the following four frontend frameworks:

- Angular
- Vue
- Svelte
- Solid

3 Theoretical Background

3.1 Architectural Prototyping

In this project, architectural prototyping is the chosen method of analysing and comparing the frontend frameworks with each other. Architectural prototyping is a medium costly way of analysing, while still giving a medium to high confidence in the results.



Life-Cycle Stage	Form of Analysis	Cost	Confidence
Requirements	Experience-based analogy	Low	Low–High
Requirements	Back-of-the-envelope	Low	Low–Medium
Architecture	Thought experiment	Low	Low–Medium
Architecture	Checklist	Low	Medium
Architecture	Analytic model	Low–Medium	Medium
Architecture	Simulation	Medium	Medium
Architecture	Prototype	Medium	Medium–High
Implementation	Experiment	Medium–High	Medium–High
Fielded System	Instrumentation	Medium–High	High

Figure 2: Forms of analysis, their life-cycle stage, cost, and confidence in their outputs [20]

At its core, Architectural Prototyping consists of the process of designing, creating, and evaluating a multitude of architectural prototypes. According to the

University of Aarhus' Computer Science Department, the definition of an architectural prototype is when prototypes consist of a set of executables created to investigate, monitor and evaluate the results of architectural qualities [21]. These qualities are often qualities related to the concerns of the stakeholders for the given project.

Within IT, the word "architecture" is used to describe the design or structure of a system. It is the foundation upon which the entire system is built and controls how the system behaves [22]. Choosing an architecture is an important decision that needs to be made at the beginning of any project, and it is therefore important that the right choice is made. An incorrect choice could result in delays, issues, and complications - especially as the requirements grow.

Prototyping is the experimental process of turning ideas and designs into tangible products that can be monitored, investigated, and measured. Prototyping is often cheaper to do compared to launching the final product right away and in most cases provides great clarity and value to the teams making use of the prototype [23].

Furthermore, prototyping is an alternative way of analyzing architecture that gives valuable insights by helping turn unknown architectural parameters into constants that can be used for decision making [24].

Architectural prototyping is therefore an extremely useful and important concept to make use of when deciding on an architecture. A variety of prototypes can be made, all using different architectures, and then they can be evaluated. During these prototypes, obstacles can be caught early in the process and qualities such as performance, scalability, usability etc. can be used to evaluate the prototype.

3.1.1 How does this project apply Architectural Prototyping?

We have used Architectural Prototyping in this project by making eight different prototypes. These prototypes were investigated and monitored by evaluating their scoring for the given software qualities. These software qualities are important when it comes to choosing a frontend framework for a given project.

3.2 Software Qualities

Software qualities are in essence characteristics or attributes of software systems that determine their effectiveness, usability, and overall quality. In the book "Software Architecture in Practice", the authors define software quality as a property of a system that can be measured or tested to determine how effectively it meets the requirements of its stakeholders [25]. Essentially, the authors describe it as "measuring the "goodness" of a product along some dimensions of interest to the stakeholder" [25].

The focus and implementation of these software qualities are crucial to the real-world adoption and usage of the software. For instance, if a system is not easily modifiable, it may become obsolete as new technologies and requirements emerge. Therefore, understanding and prioritizing the right software qualities when choosing a frontend framework is essential for a successful future.

Below are the software qualities that this project and thesis find important to prioritize when choosing a frontend framework;

- Learning curve & development speed
- Community & extensions
- Modifiability

3.2.1 Learning curve & development speed

3.2.1.1 Definition and background

The learning curve measures how quickly a learner acquires new knowledge or skills in relation to the time required to complete a task. The learning curve is often represented visually as a graph of a power function [26].

The learning curve is a software quality because it assists businesses and new learners to predict their performance and evaluate how long it takes to learn a new framework or how many tasks they will complete in a given timeframe. This can help decide whether it is worth investing resources into learning or switching to a new framework.

The steepness or shallowness of the learning curve indicates the difficulty in acquiring new information. A steep or slow learning curve signifies that the new information is challenging to comprehend and apply, and requires a longer period to master. On the other hand, a shallow or fast learning curve suggests that the information is easier to acquire, and each task requires a similar completion time, implying a quicker acquisition of the new information.

The learning curve can be visualized and calculated with the following formula [26]:

$$y = a * x^b$$

where

- y is the average time it takes to complete task x.
- a is the time taken to complete the initial task.
- x is the number of tasks to be completed.
- LR is the learning rate percentage as decimal, which is the average time change on each task completed. LR can is calculated with the following formula:

  $$LR = \frac{(\sum_{n=1}^{k} \frac{time(n+1)}{time(n)})}{k-1}$$

  k is the total tasks
  n is the current iterated task
  time(n) is the total time in minutes for task n

- b is the learning rate. It is calculated with the following formula:

  $$b = log_{10}(LR)/log_{10}(2)$$

3.2.1.2 What to measure

The developers will measure the total time it takes to complete each backlog item. Moreover, each developer evaluates on a subjective basis, how it felt to complete each task. This rating, on a scale from 1-10, reflects the level of difficulty experienced by the developers in completing the task within the given framework. A rating of 1 signifies that the task was straightforward, presenting no challenges and offering no further learning opportunities. A higher rating indicates that the task was more difficult, pushing the developer to face challenges and gain new knowledge.

3.2.1.3 How to measure

The developers will set up two timers. One timer for documentation and research. Another timer for the time spent coding. These timers will be activated and paused individually, corresponding to the time spent reviewing documentation and writing code. This timing process will be implemented for each backlog item. Furthermore, the subjective assessment of each backlog item will be a rating on a scale from 1-10, as previously mentioned.

3.2.2 Community & Extensions

3.2.2.1 Definition and background

Community and extensions is a software quality that describes a framework's community size, activity, the number of extensions that exists for that framework, and how often those extensions are being used.

The community and availability of extensions can be key factors to consider when making a decision about which framework to use. A small and inactive community surrounding a framework can raise concerns about the framework's future development, as the limited number of contributors may lead to slow progress and potential abandonment of the project. Therefore, assessing the level of community engagement and support, as well as the availability of extensions, can provide valuable insights into the long term viability and potential of a framework.

3.2.2.2 What to measure

In order to evaluate community and extensions, the main information will be gathered before and during development. Before development, information will be gathered for each framework's community size and extensions. This includes how many extensions there exist, how active the community is, and other statistics that will give an indication of the framework's future. Additionally, the developers will record their experiences and any difficulties encountered while using the extensions. These recordings will be transformed into a rating from 1 to 10, where 1 indicates that the extension is easy to use and fits perfectly for the needs of the developer, and 10 indicates that the extension is extremely complicated to use and may not be what the developer is exactly looking for. This provides valuable insights into the range of extensions offered by the framework and the functionalities.

3.2.2.3 How to measure

The metrics for the community and extensions will be measured using a variety of data points as seen in the list below. In addition, through the experiments, developers will provide a subjective rating from 1 to 10, as previously discussed, reflecting their opinion on the number of available extensions and their overall experience using the extensions.

When measuring this software quality, the following questions will be asked during the experiment. This will give insight into how the framework is performing.

- What is the user-friendliness of the framework's extension on a scale from 1 to 10? Did the extension fit the needs perfectly or was it not exactly what was looked for?
- How active is the Stack Overflow community for the framework?
- How many GitHub stars has the framework received?
- What is the number of npm packages and extensions available for this framework?
- What is the weekly download rate for this framework?
- What is the count of GitHub issues and pull requests opened and closed for this framework?

3.2.3 Modifiability

3.2.3.1 Definition and background

Modifiability is a software quality attribute that refers to the ease with which changes can be made to a software system [27]. A highly modifiable system is one that is designed in a way that allows developers to easily modify or extend its functionality.

The modifiability of frontend frameworks is becoming an increasingly important software quality, with a growing emphasis on agile development methodologies and continuous delivery. With these approaches, software systems are expected to be able to evolve and adapt quickly to changing requirements and customer needs, which can only be achieved if the underlying codebase is highly modifiable [28]. Therefore, measuring modifiability can help development teams select frameworks that are more suited to agile development and continuous delivery practices, resulting in lower costs and development time.

Additionally, by measuring the modifiability of frontend frameworks, the project aims to evaluate whether it is easier for developers to learn a new framework by building the base from scratch, or if it is easier to learn by modifying or extending existing code.

Building the base from scratch requires a deeper understanding of the framework's architecture and can provide a better overall understanding of the framework, but can also be time-consuming and error-prone. On the other hand, modifying or extending existing code can provide a more hands-on and practical experience, but can also lead to a narrower understanding of the framework's underlying principles and architecture.

3.2.3.2 What to measure

In order to evaluate the modifiability of the frontend framework, the development team will be divided into pairs consisting of individuals with varying levels of expertise, one experienced and one inexperienced. The teams will be assigned tasks such that one pair focuses on the foundational base of a experiment, while the other will be tasked with extending the code. This methodology will facilitate an assessment of the modifiability, during the process of extending the already existing code base.

3.2.3.3 How to measure

In order to measure how modifiable each framework is, different metrics and methods can be used. The software quality will focus on the following:

1. **Time spent to complete the backlog item**
   What is the actual amount of minutes spent, in order to finish the modifiability of the program?
2. **Lines of code**
   How many lines of code has been produced for framework related elements?
3. **Subjective opinion**
   What is the subjective opinion of the developer after finishing the modifiability of the program?

While the first two metrics (1 & 2) can provide objective measurements of the modifiability of a given framework, the subjective opinion is important to gain a deeper understanding of the developer's experience and perspective. This metric can provide valuable insights into factors such as ease of use, learnability, and overall user experience, which cannot be measured solely by numerical data. Therefore, the combination of both objective and subjective metrics is necessary to provide a comprehensive evaluation.

4 Existing Studies

Multiple studies have already been conducted to evaluate and compare different frameworks, but all with performance in mind.

Performance is an important software quality when choosing the right framework. Hence, why we include the most thorough and well-documented existing studies to back up our final findings of what framework to choose in what situation.

4.1 Performance - Krausest

Krausest's performance metrics [29] is a project that benchmarks performance metrics for different JavaScript frontend frameworks. It is open source, allowing anyone to add their results through their own benchmarking. These results have to be approved by Krausest, who will confirm and validate the results.

All of Krausest's performance results are based on the same Google Chrome environment that the benchmark tests run in. The community of the project is regularly updating the results as new Google Chrome versions get released. Krausest's performance benchmark measures performance in terms of memory management, CPU management, and rendering time [30].

This project is built on top of Krausest's performance metrics. A small selection of benchmarks for our frameworks can be seen in table 1 below. All the data below are geometric means [1]. The lower the number, the better performance.

Row management benchmarks

Row management refers to the process of adding, removing, or updating rows in

[1] Geometric mean is a mean or average which indicates a central tendency in a set of numbers of multiple benchmark tests [31].

a table or grid-based layout dynamically using JavaScript. It is a critical part of many web applications and requires careful management to ensure efficient rendering and optimal user experience.

Startup metrics

Startup metrics are performance indicators used to measure the effectiveness of a web page's startup. This is accomplished using Lighthouse's mobile simulation. Lighthouse is an open-source tool that audits the performance, accessibility, and best practices of a web page [32].

Memory management

Memory management refers to the efficient allocation and deallocation of memory resources in a computer system. In the context of frontend development, it involves optimizing the use of memory by web applications to improve performance and ensure the best possible user experience.

Framework	**Row management**	**Startup metrics**	**Memory management**
Solid	1.11	1.05	1.34
Vue	1.23	1.25	1.87
Svelte	1.29	1.93	1.37
Angular	1.59	1.75	2.66

Table 1: Krausest results as geometric means - Chrome version 113

Table 1 is visualized with a graph to make the comparison easier.

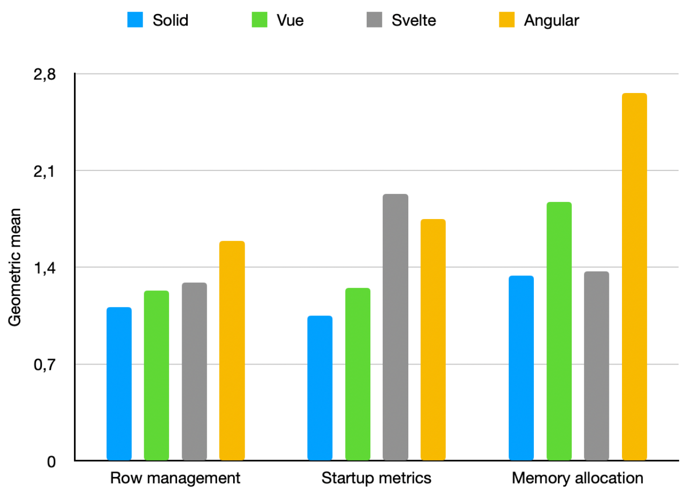


Figure 3: Krausest's results visualized

According to Figure 3, Solid emerged as the top performer in the performance test, followed by Vue in second place, Svelte in third place, and Angular in fourth place.

5 Experiments

5.1 Experiment scope

The experiment is based on building a simple and modern webshop where customers can buy a variety of designer bags. The scope of the webshop aims to touch many different features and functionalities of each framework, to support the validity of our experiments.

The webshop includes the following features and functionalities:

- Displaying interactive web pages
- Navigation between web pages
- HTTP requests to external APIs
- Browser cookie and local storage management
- Authentication and authorization

- State management and global variables

The webshop consists of a front page, an all products page, a cart page, a purchase history page, individual order page and a login/register page. A more detailed description of the requirements of each page, and how they achieve the functionality above is described in the backlog items section.

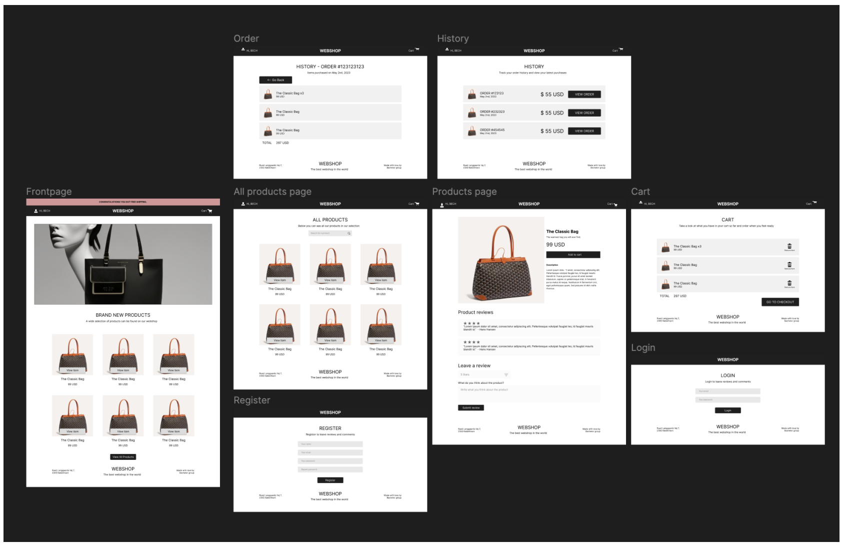


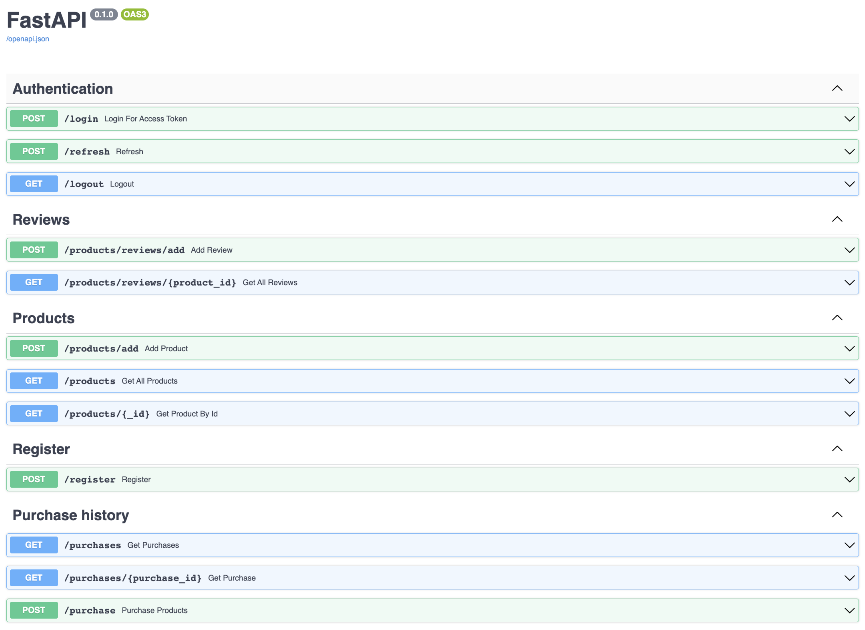
Figure 4: Figma prototype and design of experiment scope

To ensure a fair assessment and comparison of each framework, we implemented a standardized approach by excluding CSS and HTML styling and the backend functionality of the webshop from the experiment scope. Instead, we have utilized predetermined CSS styling and HTML templates for each page of the Figma prototype. Additionally, the backend API is not considered a variable in the experiment and has instead been made accessible for frontend utilization.

5.1.1 The backend API

The backend API serves as a simulation of a genuine webshop, facilitating communication between our selected frontend frameworks to retrieve and store data within the database. Additionally, the backend API plays a important role in managing authentication and authorization protocols, thereby enabling user registration and login functionalities.

The figure below shows all available endpoints of our API.



FastAPI 0.1.0 OAS3
/openapi.json

Authentication

```
POST /login Login For Access Token
POST /refresh Refresh
GET  /logout Logout
```

Reviews

```
POST /products/reviews/add Add Review
GET  /products/reviews/{product_id} Get All Reviews
```

Products

```
POST /products/add Add Product
GET  /products Get All Products
GET  /products/{_id} Get Product By Id
```

Register

```
POST /register Register
```

Purchase history

```
GET  /purchases Get Purchases
GET  /purchases/{purchase_id} Get Purchase
POST /purchase Purchase Products
```

Figure 5: Our API documentation

5.2 Experiment setup

The logistics of the experiments in this project are set up with inspiration from the SCRUM framework. There is a total of four experiments, one for each of the chosen frameworks. Each experiment will run inside a SCRUM sprint with a length of one week. The details of the experiment setup can be seen in the project plan located in the appendix section 12.

Each experiment is divided into two sub-experiments; a base experiment and an addon experiment. The goal of the addon experiment is to extend the code from the base experiment such that the modifiability of each framework can be measured. The developers will split into two groups, each with one experienced developer and one inexperienced developer. Both of the groups will conduct the base experiment for sprint one and sprint three. In sprint two and sprint four both of the groups will conduct the addon experiment by building on top of the base experiment from the previous sprint.

In the addon experiments, the experienced developer will build on top of the inexperienced developer's base and the inexperienced developer will build on top of the experienced developer's base. The purpose of doing it this way is to gain a deeper understanding of how knowledge transfer and learning occur between developers with different levels of expertise. Such insights are crucial for enhancing the professional development and productivity of individual developers, as well as for improving the overall performance of software development projects.

The tasks of the experiments are defined with a SCRUM approach, by creating backlog items that are carefully described to ensure that all developers will reach the same outcome in each experiment. This is to ensure that our results and data are comparable, decreasing the potential for misunderstandings when developing.

Lastly, each experiment has a time frame of 10 hours to be completed. If the base experiment is not completed within those 10 hours, the experiment will stop and the rest of the backlog items will be completed outside the experiment. This will allow the addon experiment to start from the same base each time, avoiding irregularities, while still keeping the same base experiment scope.

- Group 1 will consist of Mikkel Bech (experienced) and Deniz Isik (inexperienced).
- Group 2 will consist of Deniz Yildirim (experienced) and Michael Tran (inexperienced).

The experiment plan for each group, outlining the specific frameworks to be utilized, as well as the corresponding base and addon experiments can be seen in the table below.

Frameworks	**Group 1**	**Group 2**	**Week 1**	**Week 2**	**Week 3**	**Week 4**
Angular & Vue	Base	Addon	Team 1 Angular - Base	Team 1 Vue - Addon		
Vue & Angular	Addon	Base	Team 2 Vue - Base	Team 2 Angular - Addon		
Svelte & Solid	Base	Addon			Team 1 Svelte - Base	Team 1 Solid - Addon
Solid & Svelte	Addon	Base			Team 2 Solid - Base	Team 2 Svelte - Addon

Table 2: The experiment plan

5.2.0.1 The process of the experiment

The process of the experiments follows the list below, where the developers will:

1. Answer a set of ”before” questions.
2. Setup timers for measuring documentation and coding time.
3. Answer a set of ”during” questions while completing the backlog items in each experiment.
4. Answer a set of ”after” questions. Only for the addon experiment.

Before the experiments begin, the developers answer a set of ”before” questions, that has the purpose of getting to know the developers' skill level and previous work with that experiment's framework.

While the experiments are being conducted, the developers must answer a set of ”during” questions. These questions are primarily focused on how the developer felt while developing, as well as the time measured for documentation reading and coding time.

When the experiment has been conducted, the developers must answer a set of “after” questions. These questions are focused on the modifiability of each framework.

All the aforementioned questions for the experiments can be found in the appendix section 11.2.

5.2.0.2 Framework metrics assessment

Part of the before questions includes making an assessment for each framework. This gives an overview of the framework's community and activity level, which gives an insight into how future-proof the framework is. These are the following assessment metrics for each framework:

- **GitHub stars:** How many Github stars [2] does each framework have? This gives a general indication of how liked the framework is.
- **Weekly downloads:** How many times is each framework downloaded on a weekly basis? This can be measured on npm (node package manager), a website which the majority of web developers download their packages from. This gives a general indication of how used the framework is. [3]

[2]GitHub stars are a way for users to bookmark and show appreciation for a repository.

[3]The number of weekly downloads might be inflated due to automated processes that redownload the packages.

- **Framework Extensions:** How many extensions are there for each framework? This number can be found on npm. This gives a general indication of how active the community is and how many extensions exist that a developer can use in their project.
- **GitHub Issues & Pull Requests:** How many issues and pull requests are open and closed for each framework? These numbers are extracted from the GitHub repository. This shows how active the development of a framework is.
- **Stack Overflow Questions:** How many questions have been asked on Stack Overflow [4] for each framework? This number can be found on Stack Overflow using their search functionality. This gives an indication of how active the community is.

5.2.1 Backlog items

The backlog items are split into "base" and "addon" items. These items ensure that the developers gradually build the desired goal of a webshop and tests software qualities by completing each task and noting down the results discussed in the previous section[5].

All items have the requirement of matching the Figma prototype visually.

5.2.2 Base backlog items

The base backlog items are the tasks for the base experiment.

1. **Set up project:** Set up the project with the given framework and make sure that it is usable moving forward. In order to complete this task, the following has to be implemented:
    - Read the documentation on how to get started and set up the project using TypeScript.
    - Ensure that the application can be run in development mode and that it can be built.
    - Ensure that all dependencies installed are apparent and located in the `package.json` file.

[4]Stack Overflow is a forum website for programming related questions.
[5]The backlog items can be seen in detail in the appendix 11.7.

- Add a reset and normalize CSS file to all pages. This is to ensure that the website looks the same on all browsers.

2. **Frontpage:** Implementation of the website's first page; the frontpage. The page requires a call to the API to retrieve a set of webshop items to display. Users will be able to visit each item as well as navigate to all products. This task is considered complete when the following steps have been completed:

   - Redirect the user to a product's information page if they choose to view the item.
   - Redirect the user to the all products page if they click on the "view all products" button.
   - Retrieve data from the API and display it accordingly.

3. **All products page:** Implementation of the all products page, which visually looks like the frontpage but has additional functionality such as search. This task is considered complete when the following steps have been completed:

   - Add search functionality to filter between the items. The list of products should be updated on each keypress.
   - Retrieve data from the API and display it accordingly.
   - Redirect the user to the product information page when viewing the item.

4. **Product page:** Add a page that has in-depth information for any given product. This task requires dynamic URL parameters to be set up. This task is considered complete when the following steps have been completed:

   - Retrieve data from the API and display it accordingly.
   - Add support for dynamic parameters in the URL, for example, `/product/{id}`

5. **Register:** Implement functionality for users to create an account. This task is considered complete when the following steps have been completed:

   - Add form validation to the form. Communication with the API should only occur if the validation succeeds.

- Form validation should either come from framework built-in validation, a validation extension library for the given framework, or a simple validation library that is not framework specific.
- If the user does not meet the validation, red borders on the textfield should occur to notify the user.
- User should be informed through alerts if the registration succeeded or failed.

6. **Main and free shipping header:** Implement two different headers on the website. The header is visible across all pages. The main header is a simple navigation bar and the website's logo. The other header is a "free shipping header" that, based on a coin flip, determines if the user is eligible for free shipping or not. The "free shipping header" is visible across all pages. This task is considered complete when the following steps have been completed:

   - The webshop title on the header should redirect the user to the front-page when clicked on.
   - The left side of the header should display "register login" if the user is not logged in and redirect to `/register` when clicked on "register".
   - For the free shipping header, once the coin flip has been flipped, the value should be stored using the browser's cookies. The banner should be displayed accordingly to the value of the browser's cookie on page refresh.

5.2.3 Addon backlog items

The addon backlog items are designed to modify and extend the base functionality.

1. **Login:** Implement login functionality. This will allow users to purchase items and view their purchase history. This task is considered complete when the following steps have been completed:

   - Similar to the register page, the login needs to use the existing implementations and libraries to handle form validation.
   - If the login is successful, the user's tokens and username should be stored such that the username can be displayed in the header and the tokens can be used to communicate with the API endpoints that require authentication.

2. **Cart functionality:** Implement functionality to add items to cart and the ability to make a purchase. This task is considered complete when the following steps have been completed:

   - Add functionality to add items to the cart. If the same item is added more than once, the item quantity should increase.
   - Add a cart page where the user can remove items from their cart or checkout to purchase their items.
   - All items in the cart must be stored in the cookies to achieve persistence. This means if the user reopens the webshop, their items are still present in the cart. Cookies should expire after 7 days.
   - A cart should be cleared once a purchase has been made. A purchase should be made by communicating with the API.

3. **Purchase history:** Implement functionality for users to view their purchase history and view what items were purchased for each order. This task is considered complete when the following steps have been completed:

   - Add the two different pages, one for viewing all orders and one for viewing the items purchased in a specific order.
   - Retrieve the purchase history from the API and display it accordingly.

4. **Product reviews & comments:** Modify existing page in the webshop and extend it with functionality to view and create reviews on products. The task is considered complete when the following steps have been completed:

   - Retrieve product reviews and comments from the API and display them accordingly.
   - Add functionality for users to write their own reviews. This should have form validation, before communicating with the API.

5.3 Experiment 1 - Angular

5.3.1 Framework background

Angular is one of the most used frameworks in the frontend community. It is known to be one of the best frameworks for large frontend applications, and provides an extensive set of tools and libraries to assist developers. It is known to have an active community, which results in many resources and support for developers. It is, however, also known to have a steep learning curve [33].

Angular is backed by Google, who back in 2010 created AngularJS, which is Angular's predecessor. 6 years later, Angular was released, which was an enormous change from how AngularJS previously worked. It is fully based on TypeScript and comes with performance gains, better architecture, and many more improvements [34].

Based on a Stack Overflow survey from 2022, only 10% of people learn coding by using Angular. On a professional level that number increases to 22% of developers using Angular [10].

Angular uses the real DOM and updates this incrementally when changes to components are being made, also referred to as incremental DOM. Generally, you would assume that the incremental DOM approach is faster than having a virtual DOM [6] since memory is only allocated where needed. However, in most cases, a virtual DOM is faster, since the incremental DOM makes use of a solution to calculate the differences made in the DOM tree, which ultimately impacts the overall performance [35].

However, the incremental DOM shines especially on smaller devices with lower memory capacity such as mobile devices. This is due to the effects of tree shaking, and that running the diffing algorithm can be memory intensive.

5.3.2 Framework metrics assessment

Angular	
GitHub stars	88.000
Weekly downloads	3.300.000
Npm packages	13.400
Issues	25.200
Pull requests	23.900
Stack Overflow questions	295.000

Table 3: Framework Metrics Assessment of Angular [36].

[6]Visualised in figure 25.

5.3.3 Experiment results

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Setup	3m	1m	4m	1/10	N/A
Front-page	1h 31m	45m	2h 16m	8/10	2.5/10
All products page	1h 47m	46m	2h 33m	5/10	1/10
Product page	1h 16m	15m	1h 31m	4/10	5/10
Register	1h 48m	35m	2h 23m	4/10	6/10
Header	24m	10m	34m	2/10	1/10
Total	6h 49m	2h 32m	9h 21m		

Table 4: Deniz Isik - Inexperienced developer, Angular base.

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Setup	5m	2m	7m	1/10	N/A
Front-page	28m	38m	1h 6m	5/10	1/10
All products page	15m	30m	45m	3/10	1/10
Product page	30m	18m	48m	3/10	1.5/10
Register	1h	32m	1h 32m	5/10	4.5/10
Header	30m	30m	1h	3/10	1.5/10
Total	2h 48m	2h 30m	5h 18m		

Table 5: Mikkel Bech - Experienced developer, Angular base.

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Login	1h 37m	46m	2h 23m	7/10	5/10
Cart Functionality	56m	42m	1h 38m	5/10	2/10
Review & Comments	45m	57m	1h 42m	5/10	4/10
Purchase History	48m	1h 17m	2h 05m	3/10	2/10
Total	4h 06m	3h 42m	7h 48m		

Table 6: Michael- Inexperienced developer, Angular Addon.

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Login	8m	37m	45m	7/10	3/10
Cart Functionality	5m	58m	1h 3m	2/10	1/10
Review & Comments	11m	1h 25m	1h 36m	5/10	2/10
Purchase History	5m	55m	1h	2/10	1/10
Total	29m	3h 55m	4h 24m		

Table 7: Deniz Yildirim - Experienced developer, Angular Addon.

Developer	Average learning curve	Average extensions
Deniz Isik	4.00/10	3.10/10
Mikkel Bech	3.33/10	1.90/10
Michael Tran	5.00/10	3.25/10
Deniz Yildirim	4.00/10	1.75/10

Table 8: Angular average results

5.3.4 Experiment mini-analysis

5.3.4.1 Learning curve & development speed

The tables above show that the inexperienced developers spent a significant time reading documentation, while the experienced developers did not spend nearly as much time. This shows that Angular initially requires more time for inexperienced developers to understand the framework, but once they get familiar with the framework the documentation time is significantly reduced.

Both experienced developers spent between 4 to 6 hours completing their tasks, meanwhile it took the inexperienced developers between 7 and 9 hours. This is nearly double the time needed for inexperienced developers, which again emphasizes that choosing Angular requires initially more time to understand and master.

5.3.4.2 Extensions

The tables show that it has been easy to work with extensions and many of them come preinstalled within the framework but are not built in. The majority of the extensions used are backed and developed by the Angular team [7] giving the

[7]Extensions such as @angular/router [37], @angular/forms [38] etc.

stakeholders confidence in the maintainability of the system.

5.3.5 Subjective takes

5.3.5.1 Base

In the Angular development process, the experienced developers appreciate the comprehensive documentation, seamless initial setup, and reusable service files [8]. On the other hand, the inexperienced developers encountered challenges with the steep learning curve, the required coding and structuring conventions, and the need to manage multiple files while adding new features. Overall, Angular presents a diverse range of experiences for developers with varying levels of expertise.

5.3.5.2 Addon - Modifiability

Although the inexperienced developers found Angular's documentation difficult to understand, extending the previous code was manageable due to the well-organized code structure provided by the experienced developers. The experienced developers initially faced challenges with the framework, as multiple files and elements needed attention. They also noted that the framework's focus on separation of concerns, while beneficial, could feel overwhelming. However, they ultimately enjoyed working with Angular and appreciated its opinionated approach to front-end development, which offered a clear and concise way to accomplish tasks. As a result, Angular emerged as a potential choice for future projects, highlighting the adaptability and easily modifiable nature of the framework.

5.4 Experiment 2 - Vue

5.4.1 Framework background

Vue was officially released in February 2014 by Evan You [39]. It has gained immense popularity with more than 200.000 stars on GitHub [40] and created a strong ecosystem that contains training material and official certifications. [41]

For companies, Vue provides an ecosystem of Vue Partners, that can help companies get support with Vue-related projects. The Vue Partners consist of consultancy firms with expertise in Vue, that has been verified and partnered with Vue to offer first-class support, such as consultancy, training, development, and

[8] A service file is a collection of functionality responsible for a single area, such as a user service. The service files can be used across components and pages.

support [42].

The rendering system of Vue is based on a virtual DOM which is stored in memory, and updated once changes to the application are made ??25Visualised in figure 25 updated Virtual DOM is always compared to the previous one, which allows the actual DOM to only update what has been changed from the previous version. This results in high performance as changes to the website are only made to components that require updating [43].

5.4.2 Framework metrics assessment

Vue	
GitHub stars [9]	240.000
Weekly downloads	3.700.000
Npm packages	67.000
Issues	602
Pull requests	365
Stack Overflow questions	103.000

Table 9: Framework Metrics Assessment of Vue [44].

5.4.3 Experiment results

Backlog item	**Documentation**	**Coding time**	**Total time**	**Learning curve**	**Extensions**
Setup	10m	25m	35m	3/10	N/A
Front-page	35m	23m	58m	5/10	2/10
All Products Page	10m	14m	24m	2/10	1/10
Product Page	42m	1h 1m	1h 43m	5/10	4/10
Register	32m	26m	58m	3/10	5/10
Header	30m	47m	1h 17m	4/10	3/10
Total	2h 39m	3h 16m	5h 55m		

Table 10: Michael - Inexperienced developer, Vue Base.

[4]Combined GitHub stars of Vue 2 and Vue 3

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Setup	7m	18m	25m	1/10	N/A
Front-page	18m	52m	1h 10m	3/10	3/10
All products page	12m	10m	22m	5/10	1/10
Product page	4m	16m	20m	1/10	2/10
Register	22m	58m	1h 20m	5/10	5.5/10
Header	0m	15m	15m	2/10	1/10
Total	1h 3m	2h 49m	3h 52m		

Table 11: Deniz Yildirim - Experienced developer, Vue base.

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Login	1h 31m	22m	1h 53m	3/10	7/10
Cart Functionality	15m	42m	57m	1/10	1/10
Review & Comments	30m	1h	1h 30m	4/10	3/10
Purchase History	12m	1h 33m	1h 45m	1/10	6/10
Total	2h 30m	3h 39m	6h 9m		

Table 12: Deniz Isik - Inexperienced developer, Vue addon

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Login	19m	46m	1h 5m	2/10	2/10
Cart Functionality	15m	42m	57m	2/10	1/10
Review & Comments	12m	30m	42m	2/10	2/10
Purchase History	6m	40m	46m	2/10	1/10
Total	52m	2h 38m	3h 30m		

Table 13: Mikkel Bech - Experienced developer, Vue addon

Developer	Average learning curve	Average extensions
Deniz Isik	2.25/10	4.25/10
Mikkel Bech	2.00/10	1.66/10
Michael Tran	3.67/10	3.00/10
Deniz Yildirim	2.83/10	2.50/10

Table 14: Vue average results

5.4.4 Experiment mini-analysis

5.4.4.1 Learning curve & development speed

Based on the results for Vue, the overall learning curve for all developers is placed on the lower end of the spectrum. This suggests that developing in Vue has a relatively low learning curve overall. The maximum learning curve also never exceeds 5, which further indicates the ease of learning.
It is also notable, that the coding time for all developers exceeds the amount of time spent on documentation. This implies that Vue likely has good documentation since the developers are spending less time reading documentation and more time coding. It also implies that Vue is generally easier to work with, offering easy-to-use features.

5.4.4.2 Extensions

Out of all the developers, the maximum average for the extensions was rated 4.25, which indicates that extensions in Vue have been fairly easy to use. Vue comes with a pre-installed routing extension, which is made and managed by Vue. This ensures longevity and great compatibility with the framework. Vue's large community allows developers to have a wide variety of extensions to choose from. This was particularly clear when selecting an extension for the form validation, where they had a large collection of specific extensions tailored to the Vue framework.

5.4.5 Subjective takes

5.4.5.1 Base

Both base developers noticed that building the foundation for Vue was natural and logical. For the inexperienced developer, the main challenge was the dynamic routing and documentation, which could also be related to limited knowledge in regards to routing. However, for the experienced developer, Vue provided a positive experience, requiring minimal documentation thanks to its intuitive setup. Even when documentation was needed, the built-in route handling and extensive feature set made it easy and straightforward.

5.4.5.2 Addon - Modifiability

According to both addon developers, Vue has been found to be a user-friendly framework with a self-explanatory file structure that allows for a rapid understanding of the existing code base, making it easy to identify areas for modification or extension of functionality. The inexperienced developer appreciates the organization of the project, with all components contained within a single file. However, they may encounter implications in understanding the approach for structuring methods and data within `.vue` files.

5.5 Experiment 3 - Svelte

5.5.1 Framework background

Svelte is a new and modern approach to building user interfaces, released in November 2016. With more than 65.0000 stars on GitHub, it has gained a lot of popularity in the last few years and is a loved framework amongst developers. A 2022 Stack Overflow survey, shows that it is the second most loved framework, and in 2021 it was the most loved out of any frameworks [10].

Svelte was recently backed by Vercel [10], and today, you can deploy your Svelte application to Vercel with zero configuration, enabling you to take advantage of features such as web analytics, serverless functions, and more [46].

It differs from popular frameworks such as React and Vue by not using a virtual DOM. Instead, it uses an approach referred to as "surgical updates" because it modifies the DOM where necessary whenever the state of the app changes [47]. This is done without extensive comparisons or updating unnecessary elements. The majority of the heavy lifting happens at the compile step, where Svelte generates highly efficient JavaScript code that can update the DOM directly. This is opposed to Virtual DOM diffing where the main work happens on the user's browser. This results in a small bundle size [11] for the users.

[10]Vercel is a cloud platform for deploying web applications and serverless functions[45].

[11]Bundle size refers to the size of the compressed JavaScript, CSS, and other assets that are downloaded by a browser when loading a web page. [48]

5.5.2 Framework metrics assessment

Svelte	
GitHub stars	67.000
Weekly downloads	520.000
Npm packages	1.260
Issues	755
Pull requests	53
Stack Overflow questions	4.800

Table 15: Framework Metrics Assessment of Svelte [49].

5.5.3 Experiment results

Backlog item	**Documentation**	**Coding time**	**Total time**	**Learning curve**	**Extensions**
Setup	6m	2m	8m	1/10	N/A
Front-page	1h 22m	22m	1h 44m	7/10	N/A
All products page	59m	20m	1h 19m	5/10	N/A
Product page	55m	31m	1h 26m	6/10	N/A
Register	1h 05m	16m	1h 21m	3/10	8/10
Header	0m	20m	20m	1/10	1/10
Total	4h 27m	1h 41m	6h 8m		

Table 16: Deniz Isik - Inexperienced developer, Svelte base.

Backlog item	**Documentation**	**Coding time**	**Total time**	**Learning curve**	**Extensions**
Setup	5m	5m	10m	1/10	N/A
Front-page	8m	15m	23m	3/10	N/A
All products page	25m	35m	1h	3/10	N/A
Product page	20m	35m	55m	3/10	N/A
Register	25m	45m	1h 10m	4/10	3/10
Header	15m	30m	45m	2/10	1/10
Total	1h 38m	2h 45m	4h 23m		

Table 17: Mikkel Bech - Experienced developer, Svelte base.

Backlog item	**Documentation**	**Coding time**	**Total time**	**Learning curve**	**Extensions**
Login	37m	21m	58m	6/10	4/10
Cart Functionality	18m	32m	50m	6/10	1/10
Review & Comments	14m	33m	47m	3/10	3/10
Purchase History	35m	1h 9m	1h 44m	4/10	1/10
Total	1h 44m	2h 35m	4h 19m		

Table 18: Michael- Inexperienced developer, Svelte Addon.

Backlog item	**Documentation**	**Coding time**	**Total time**	**Learning curve**	**Extensions**
Login	25m	55m	1h 20m	9/10	2/10
Cart Functionality	10m	1h	1h 10m	3/10	1/10
Review & Comments	0m	34m	34m	1/10	1/10
Purchase History	0m	37m	37m	1/10	1/10
Total	35m	3h 6m	3h 41m		

Table 19: Deniz Yildirim - Experienced developer, Svelte Addon.

Developer	**Average learning curve**	**Average extensions**
Deniz Isik	3.83/10	4.50/10
Mikkel Bech	2.66/10	2.00/10
Michael Tran	4.75/10	2.25/10
Deniz Yildirim	3.50/10	1.25/10

Table 20: Svelte average results

5.5.4 Experiment mini-analysis

5.5.4.1 Learning curve & development speed

As seen in the presented tables, the initial learning phase for Svelte can be challenging for developers, but this difficulty rapidly diminishes with increasing experience and proficiency. This observation suggests that once developers obtain a grounded understanding of Svelte's fundamental file structure and key components, the framework may become more intuitive and straightforward to work with.

Furthermore, the tables indicate that the overall development time for each developer was roughly 4 hours, with the exception of one developer who required 6 hours due to spending extra time reviewing documentation to fully comprehend the framework. This outcome reinforces the earlier assertion that a strong grasp of the Svelte framework is necessary to effectively utilize it and achieve greater efficiency.

5.5.4.2 Extensions

As observed in the experiment, the usage of extensions in the base scenario was minimal, with only two backlog items requiring extension implementation. This outcome can be attributed to Svelte's inherent routing and other built-in functionality that sufficiently met the requirements of the backlog items. This is a noteworthy benefit for developers considering Svelte, as it assures ongoing support and maintenance for crucial functionality for as long as the framework persists. Svelte's growing size allows for a fair number of extensions that are available to its developers. This allows the developers to save time when developing.

In the addon experiment, extensions were utilized for all the backlog items, though with an overall low rating. This finding suggests that Svelte is easily extendable with extensions, a valuable characteristic for developers seeking to customize and expand the framework's capabilities.

5.5.5 Subjective takes

5.5.5.1 Base

Svelte is a favored framework because everything is contained in a single page, making it easier to structure projects. Unlike other frameworks, Svelte does not require a routing path for pages but instead utilizes folders and files as a path name for a route. Each folder contains a single .svelte page where all HTML, CSS, and TypeScript code is located. However, the smaller community may not be suitable for inexperienced developers as there may be limited help available.

The development experience with Svelte was excellent, with a seamless process due to everything being contained within the same file, making it easy to get started. The file structure is intuitive, and there are no unnecessary functions to make it work.

5.5.5.2 Addon - Modifiability

In terms of modifiability, both developers had challenges with Svelte. One of the developers found it difficult to create addons within the framework and encountered unexpected issues and behaviors that were hard to decipher. Additionally, the uniform naming convention for pages caused confusion and sometimes resulted in code being added to the wrong files. Another developer expressed dissatisfaction with Svelte, citing poor documentation and frequent errors that could break the application.

5.6 Experiment 4 - Solid

5.6.1 Framework background

Solid is one of the newer frameworks in the frontend community and is very liked amongst its developers [12]. Solid is very similar to React but instead uses a compiler that generates very optimized code during build time. Despite Solid being relatively new, it has gained a lot of popularity among developers because of the simplicity and performance benefits that the framework provides.

Solid was developed and made open source by Ryan Carniato in April 2018. [50] Today it has more than 25.000 GitHub stars [51] and is considered one of the fastest web frameworks almost performing as fast as plain vanilla JavaScript [13] on browser benchmarks [53].

Solid uses a "fine-grained reactivity system" meaning that components only re-render when relevant data changes. This is done by tracking when dependencies change, by using reactive primitives [14] which give developers a way to clearly state and specify which data their code relies on and reacts to when it changes. Solid can then track these changes on re-render and only update the affected part of the user interface [54].

[12]Comparing the GitHub stars to the weekly download ratio in table 21

[13]Vanilla JavaScript is when an application in JavaScript is made without using any third party libraries. [52]

[14]The building blocks of Solid

5.6.2 framework metrics assessment

Solid	
GitHub stars	27.500
Weekly downloads	84.000
Npm packages	350
Issues	22
Pull requests	8
Stack Overflow questions	183

Table 21: Framework Metrics Assessment of Solid [55].

5.6.3 Experiment results

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Setup	38m	8m	46m	3/10	N/A
Front-page	47m	34m	1h 21m	5/10	2/10
All Products Page	12m	25m	37m	3/10	N/A
Product page	23m	20m	43m	3/10	2/10
Register	43m	38m	1h 21m	8/10	6/10
Header	23m	12m	35m	4/10	1/10
Total	3h 6m	2h 17m	5h 23m		

Table 22: Michael Tran - Inexperienced developer, Solid base.

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Setup	5m	8m	13m	2/10	N/A
Front page	22m	33m	55m	4/10	2/10
All products page	2m	18m	20m	2/10	N/A
Product page	2m	16m	18m	1/10	1/10
Register	5m	21m	26m	2/10	2/10
Header	0m	30m	30m	1/10	1/10
Total	36m	2h 6m	2h 42m		

Table 23: Deniz Yildirim - Experienced developer, Solid base.

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Login	1h 13m	29m	1h 42m	4/10	4/10
Cart Functionality	0	1h 16m	1h 16m	1/10	1/10
Review & Comments	40m	23m	1h 3m	8/10	7.5/10
Purchase History	0m	55m	55m	1/10	1/10
Total	1h 53m	3h 3m	4h 56m		

Table 24: Deniz Isik - Inexperienced developer, Solid addon.

Backlog item	Documentation	Coding time	Total time	Learning curve	Extensions
Login	13m	45m	58m	2/10	2/10
Cart Functionality	9m	41m	50m	3/10	1/10
Review & Comments	12m	40m	52m	5/10	3/10
Purchase History	15m	46m	1h 1m	1/10	1/10
Total	49m	2h 52m	3h 41m		

Table 25: Mikkel Bech - Experienced developer, Solid addon.

Developer	Average learning curve	Average extensions
Deniz Isik	3.50/10	3.37/10
Mikkel Bech	2.75/10	1.75/10
Michael Tran	4.33/10	2.75/10
Deniz Yildirim	2.00/10	1.50/10

Table 26: Solid average results

5.6.4 Experiment mini-analysis

5.6.4.1 Learning curve & development speed

Based on the results of this experiment, it is clear that both experienced developers spent very little time on the documentation. Furthermore, their learning curve was rated low, indicating that both experienced developers had an easy time coding in Solid. This is likely due to Solid's familiarity with React, as both experienced developers have a lot of experience with React.

The inexperienced base developer spent around 33% more time than the inexperienced addon developer. This could indicate that the documentation for setting up the project was less complete, compared to addon where the existing codebase is used. Both developers' average learning curve is around 5, which means they both generally did not struggle as much.

For the inexperienced base developer, the learning curve was rated high for the register page, and for the inexperienced addon developer, the review and comments page was rated high. Both of these are due to the use of an extension that handles form validation, indicating that the use of extensions in Solid has a steeper learning curve if not done before.

5.6.4.2 Extensions

Both experienced developers had an easy time with the extensions used, scoring an average below 2. In Solid, the process of routing is managed by a pre-installed extension that is developed and maintained by the Solid organization [15]. The extension conforms to the design patterns of Solid, thereby ensuring a reliable and consistent routing mechanism. This is a critical aspect to take into account, as it establishes a robust foundation for continuous support throughout the lifespan of the Solid framework. Furthermore, the routing is based on React's extension of route handling, which most React developers are familiar with.

Regarding form validation, the inexperienced developers performed well but encountered challenges with specific tasks. The limited size of the Solid community constrained the range of available options for form validation, resulting in the selection of extensions with limited prior community usage. It is important to consider that depending on the nature of your project, there may arise a necessity to create customized solutions to solve your problem, instead of relying on established and widely-tested extensions from the community. Ultimately leading to more development time.

5.6.5 Subjective takes

5.6.5.1 Base

Working with Solid had both positive and negative experiences. On one hand, the language felt natural to developers coming from a React background. The syntax and functions/hooks were similar, and the built-in routing feature was a bonus. Solid's performance gains make it an attractive choice. However, the limited community size was a concern when searching for extensions for forms. The

[15] Extensions such as @solid/router [56]

available options had few downloads, indicating a potential lack of community support. The documentation was also a significant challenge, particularly in implementing form validation, as the available code examples were not compatible with TypeScript. Overall, Solid could be an excellent choice for developers, but it needs improvements in documentation and community support to make it more accessible and user-friendly.

5.6.5.2 Addon - Modifiability

The Solid framework is similar to React, as noted by one developer who has prior experience with the latter. The project structure is simple, allowing for easy navigation and straightforward page creation. Additionally, the documentation is lacking in some areas, with limited explanations and insufficient coverage of certain topics.

Despite these shortcomings, the Solid framework is generally easy to work with and modify. Furthermore, the file structure inherited from React results in low coupling, enabling straightforward code modification and addition.

5.7 Lines of Code Results

Lines of code is a metric used to compare the amount of code produced by each developer while conducting the experiments. The lines of code metric is excluding non-framework elements, such as CSS styling and HTML files.

The actual lines of code are calculated with the use of GitHub's version control, where additions and deletions are visible for each commit from the developers.

Framework	**Inexperienced developer**	**Experienced developer**
Angular	653 lines	792 lines
Vue	804 lines	483 lines
Svelte	630 lines	681 lines
Solid	467 lines	369 lines

Table 27: Lines of code

The table above shows the results for each framework, separated for the inexperienced and experienced developers. Further analysis and details can be read in the experiment analysis in section 6.1.

6 Experiment conclusion

6.1 Experiments analysis

This experiment analysis aims to compare the various frameworks previously covered in this report. Frontend frameworks are generally designed to simplify and make the development process easier, which may explain why we observed very similar learning curves and documentation times across the different frameworks.

We acknowledge that our analysis is limited by the fact that only four developers were involved. As such, our findings should be interpreted with discretion, as further outlined in the limitations section. Nonetheless, this analysis seeks to identify key differences between the frameworks that are critical to consider when selecting the most suitable framework.

Looking at the community of each framework, Solid suffers from a lack of community engagement, with only 183 Stack Overflow questions available. This may suggest that either the documentation is exceptionally well-crafted, as evidenced by our experiments indicating low documentation readings (Figure 6), or that there are insufficient numbers of users to ask or answer questions. A contributing factor could also be its similarity to React. Most developers who try out Solid are likely coming from a React background, resulting in the majority of the framework being relatively straightforward.

An interesting factor to consider when evaluating these frameworks is the GitHub Stars ratio to Weekly Downloads. Notably, Solid has GitHub Stars that correspond to 32.7% of weekly downloads, whereas Svelte has GitHub Stars that correspond to 12.88% of weekly downloads, and Vue has GitHub Stars corresponding to 6.4% of weekly downloads. This observation suggests that Solid has a high level of popularity and esteem, but has yet to achieve widespread adoption and usage. Solid's popularity may be due to its similarity to React, the most used JavaScript frontend framework [10], combined with its superior performance, as demonstrated in Krausests' benchmark tests. This suggests that developers may be drawn to Solid as a viable alternative, particularly when performance is a priority.

Furthermore, it is worth noting that Angular has only 88,000 GitHub Stars, which corresponds to a mere 2.6% of weekly downloads. This observation may indicate that many organizations use Angular in their daily operations, but the framework itself is not used often for personal or smaller projects. This is likely due to Angular's complexity and opinionated approach to how the framework should be used, which corresponds well with the development time throughout the experiments, as Angular was rated the most difficult to learn and took the most time to code with.

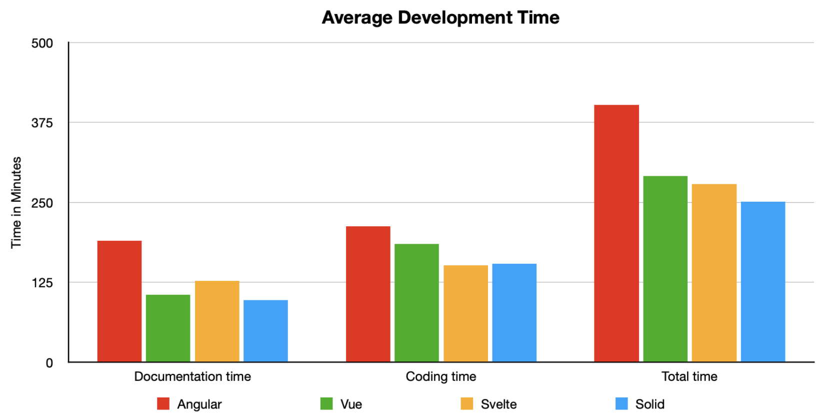


Figure 6: Average development time across frameworks

Based on the framework metrics assessment, Vue showed to be the most popular and well-liked framework. This observation aligns with the subjective takes from this project received from the developers, who praised Vue's file structure as being particularly excellent. In contrast, other frameworks like Angular had a strict file structure that may work well for larger applications, but can pose challenges during the initial setup, as evidenced by Figure 6 where Angular had the longest development time across all developers.

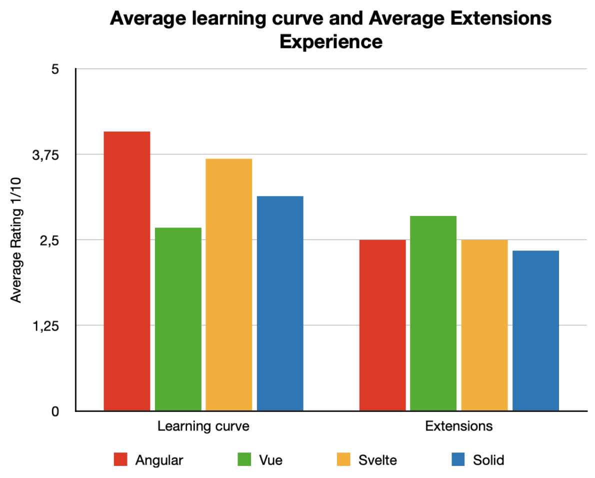


Figure 7: Average learning curve and extension experience

The average learning curve and average extension usage can be seen in Figure 7. It is clear that angular had the highest and steepest learning curve, while Vue had the lowest. This can again correlate to the strict structure and very opinionated framework that Angular has.

An intriguing observation from the experiment is the comparison of the average learning curve for Vue and Svelte. Both frameworks have nearly identical file structures, foreseeing that they would have a similar learning curve. Instead, the graph reveals that Svelte was more challenging to learn than Vue. This finding, coupled with the average development time (Figure 6), may suggest that Svelte's documentation and community are respectively harder to comprehend and get answers from during development. This results in Svelte having a steeper learning curve than Vue despite its similarities in structure and functionality.

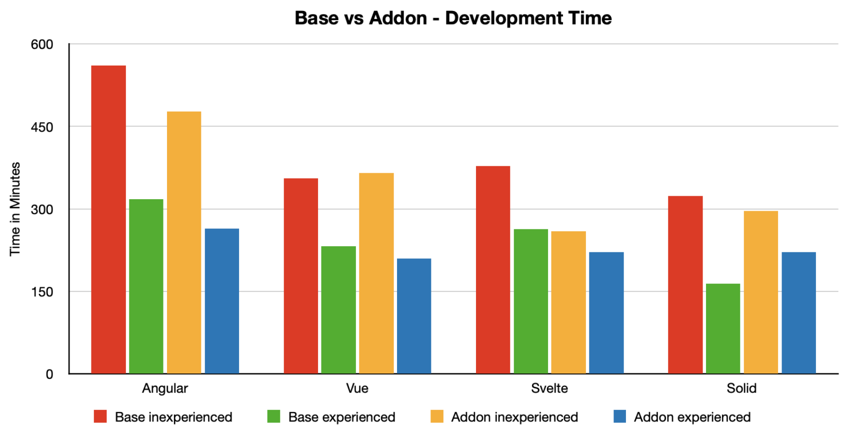


Figure 8: Development time for Base and Addon

Looking at Figure 8 and considering development time as the sole metric for learning a framework, it is evident that both experienced and inexperienced developers find it easier to learn Angular and Svelte when working on addon compared to the base. On the other hand, when it comes to Vue and Solid, there were instances where it proved to be easier to build the base rather than modify existing code.

Furthermore, the data from Figure 8 shows that inexperienced developers spent the most time on Angular based on the coding time, compared to all other frameworks, regardless of whether they are working on addons or building the base. However, in the case of Svelte, it could be argued that the experience of the developer is less influential in determining the ease of completing a task.

Based on Figure 7 and Figure 8, it is evident that Angular and Svelte, the two frameworks where it is hardest to build the base, also have the steepest learning curve. Therefore when learning a harder framework, it may be more beneficial to learn by modifying and extending existing code, rather than attempting to set up everything from scratch. Especially if time plays a role in choosing the framework.

In the case of Vue and Solid, which have the lowest learning curve, it becomes apparent that it is easier to build the base, rather than extend on existing code. As a result, the opposite approach would be more suitable for these frameworks.

Another important software quality examined is the learning curve. The next graphs show the average learning curve for experienced and inexperienced developers across the different frameworks.

It is worth mentioning that the setup backlog item has not been considered in the calculations of the learning curve. [16]

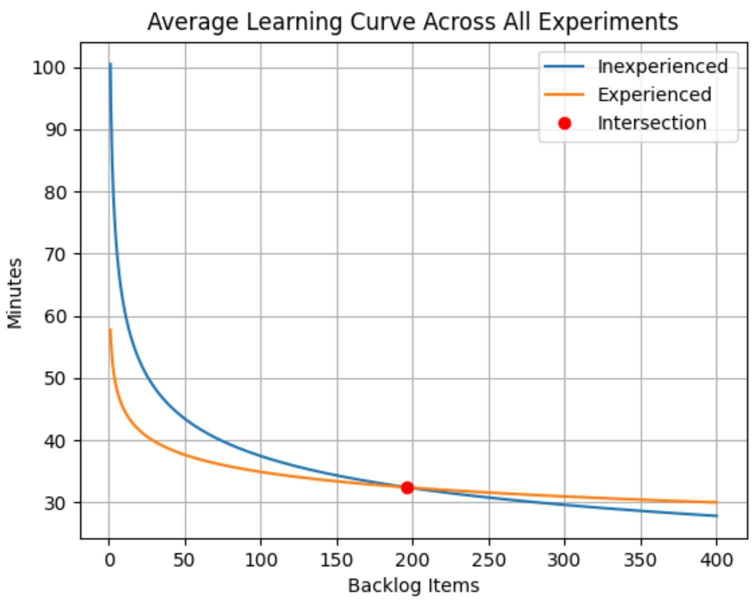


Figure 9: The learning curve for experienced and inexperienced developers across all frameworks

The graph illustrates that inexperienced developers took almost twice the time of experienced developers for their initial task. However, inexperienced developers demonstrated a faster learning curve, indicating that they progressed faster over time. It is possible that the lack of prior experience in other frameworks among the inexperienced developers has contributed to their faster learning of the project's framework. As they do not have any pre-existing habits or practices to hinder their learning process.

Additionally, the graph indicates that inexperienced developers will reach the same level of proficiency as experienced developers after completing 195 backlog items.

[16] The setup backlog item proved to not have the same weight required as the other backlog items, resulting in our data being distorted if it would have been included.

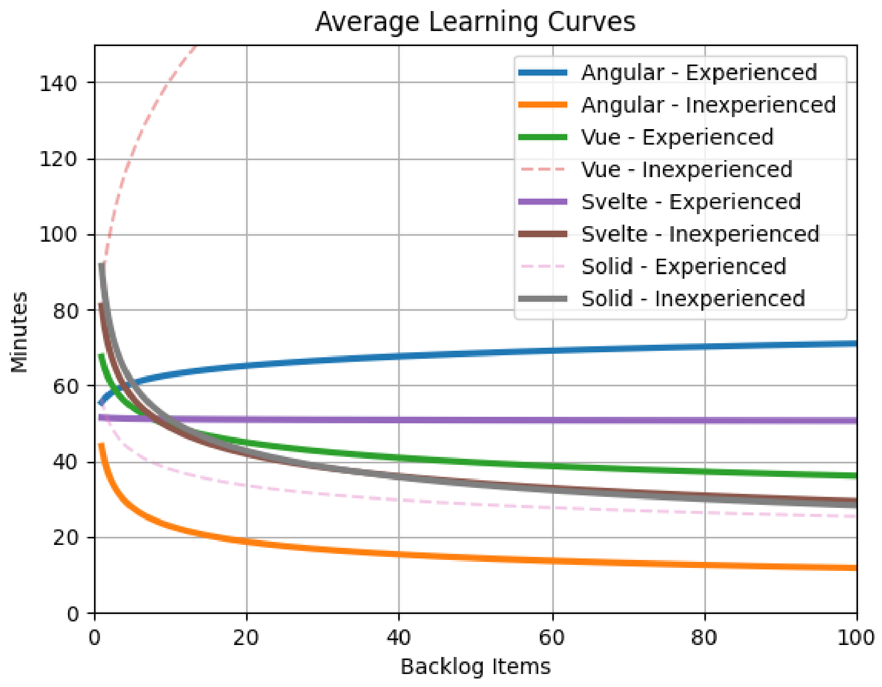


Figure 10: The learning curve for each framework

The graph above shows the learning curve for each framework with the inexperienced and experienced developer separated. The dashed and transparent learning curves, should not be considered due to its faulty and misleading data. The reason for this is explained in detail in limitations (section 7.4).

As previously explained, this graph confirms that the experienced developers almost have a flat learning curve while the inexperienced has a faster learning curve.

Moreover, it is worth noting that despite the similarities in structure and approach between Vue and Svelte, the experienced developers showed a significant difference in their learning curves. The experienced developers in Svelte nearly had a flat learning curve, while those in Vue had a much faster one. This finding aligns well with our previous observation that Svelte was subjectively perceived as more challenging to learn than Vue (section 6.1).

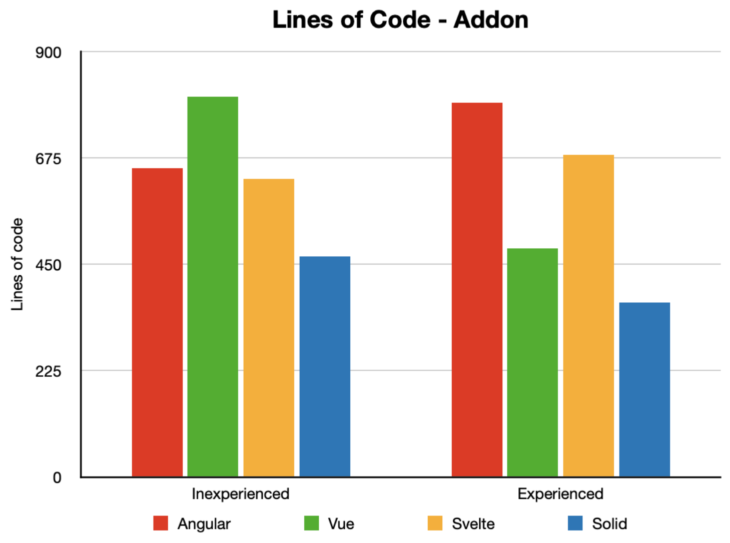


Figure 11: Lines of code for inexperienced and experienced developers

Although it can be difficult to compare lines of code as a metric because of the varying nature of each framework, it's worth examining the lines of code written by the developers. Looking at Angular, the experienced developer wrote more lines of code than the inexperienced, which could be attributed to the experienced developer's ability to make use of Angular's more opinionated and strict principles which requires more files and lines of code as explained in 5.3.1.

For Vue and Svelte, it appears that the complexity of the Vue ecosystem makes it harder for an inexperienced developer to modify Vue compared to Svelte. On the other hand, experienced developers, who have the ability to quickly pick up new frameworks, may find it easier to do addons in Vue compared to inexperienced developers. In this case, the complexity of the documentation becomes a less significant factor.

In general, Svelte appears to be an intuitive framework for both inexperienced and experienced developers, as the number of lines of code written is nearly identical.

Solid stands out as the most straightforward framework for doing addons, although as previously mentioned, this is not necessarily the subjective feeling shared amongst the developers.

6.2 Experiments discussion

6.2.1 Identical Product Across Experiments

An important consideration of the design of these experiments is the consistent replication of the same product across all four experiments. This may introduce a degree of bias into the results due to the learning curve associated with the initial implementation.

In the first two sprints, it is reasonable to anticipate a more challenging and time-consuming process, given that the developers are encountering the task for the first time. By comparison, the third and fourth sprints would potentially benefit from the experience and familiarity, which could accelerate the implementation process.

This must be taken into consideration when interpreting the results, as the apparent efficiency improvements in the latter experiments may not entirely reflect the performance of the different frameworks, but rather the developers' growing expertise with the assigned tasks.

6.2.2 Non-Framework Related Challenges

An additional consideration that potentially influenced the outcome of the experiments involves challenges encountered by developers that are not directly related to the specific framework in use. These could include issues arising from the application of HTML, CSS, or TypeScript.

While it is challenging to precisely quantify this factor, it is important to acknowledge its potential impact on the metrics that were measured. The project tried to accommodate this factor by building precise and well-made non-framework-related elements, for the developers to use. When interpreting the results, it is therefore crucial to consider that certain difficulties may stem from these non-framework-related elements rather than the specific properties of the framework itself.

6.2.3 Lines of code as a metric

Something we should have considered is a predefined coding style, which ideally should have been applied to all the written code. Not doing this increased the margin of error when looking at the lines of code metric, since some developers used linters [17] and autoformatters [18], which splits lines of code into multiple lines in order to make the code more readable.

One might think that fewer lines of code mean that a framework is easier to modify. However, it is important to recognize that this assumption can be misleading and oversimplified. Due to the limited scope of this project, we will not discuss this topic further. Therefore, we advise readers to evaluate this metric with caution and consider it alongside other relevant factors.

6.3 Experiments conclusion

Based on all of the experiments conducted for all the frameworks and the subsequent analysis of the results, we can conclude that all of the frameworks have their advantages and disadvantages.

6.3.1 Angular

6.3.1.1 Positives

Angular, backed by Google, is one of the more mature frameworks in the industry. Angular comes pre-installed with a palette of extensions, all of which are built and maintained by the Angular team. This guarantees that the extensions align with Angular's design principles and uphold the highest industry standards. It further ensures that issues or updates that may arise within the extensions are resolved rapidly.

The most prominent feature of Angular is its opinionated approach to development within the framework. It offers a singular, defined method to accomplish specific tasks, ensuring consistent and organized coding practices. This feature makes Angular particularly suitable for large projects requiring such consistency and systematic organization, and therefore, improves the modifiability of the framework, as seen in the experiments.

6.3.1.2 Negatives

Angular's complexity resulted in it being the framework where developers, especially the inexperienced, spent the most time. It has a relatively steep learning

[17]A code style linter is a type of linter that analyzes source code to ensure it adheres to a specific set of coding conventions or style guidelines.

[18]An autoformatter is an automated process that formats the code to adhere to style guidelines

curve, requiring significant investment in understanding and referring to the documentation.

Furthermore, Angular's opinionated approach may be intimidating and potentially overwhelming for some developers. It can restrict the freedom of developers in making structural decisions, which could be perceived as a limitation.

6.3.2 Vue

6.3.2.1 Positives

Vue is distinguished by its remarkably fast learning curve, making it an accessible choice for developers to quickly start coding. A significant contributor to its positive experience is its file structure: Vue's design, which locates all elements within a single file, facilitates ease of use and clarity. This makes Vue quite modifiable and extendable.

The framework benefits from a robust community, ensuring a wealth of extensions that cater to a wide range of specific developer needs, often enhancing usability. Furthermore, Vue's built-in routing system is intuitive and straightforward. Performance-wise, Vue ranked as the second most efficient framework after Solid, positioning it as an excellent choice for those seeking a performance-oriented solution.

6.3.2.2 Negatives

Despite the positives, there are areas where Vue could improve. The framework's documentation is currently not satisfactory, creating challenges for those who are just starting with Vue. The existing documentation lacks sufficient examples, which may make problem-solving more difficult for developers. However, based on the conducted experiments, these limitations are relatively minor. Overall, Vue demonstrates itself as a robust and reliable choice for a framework, with its advantages largely overshadowing its few drawbacks.

6.3.3 Svelte

6.3.3.1 Positives

Svelte offers a file structure that is similar to Vue, an attribute that was well-received by the developers. Additionally, Svelte includes an intuitive and user-friendly built-in routing system. These features, similarly to Vue, enhance the modifiability and extendability of the framework, making it a flexible choice for various projects. Furthermore, Svelte's compiler allows for small bundle sizes as previously stated.

6.3.3.2 Negatives

Svelte's relatively small user community results in a limited variety of available extensions. This deficiency can pose challenges for addressing diverse use cases, and the available extensions may be outdated or come with their own set of issues. Furthermore, Svelte's documentation has been criticized as being difficult to comprehend, which could present challenges, especially for inexperienced developers.

6.3.4 Solid

6.3.4.1 Positives

Solid demonstrates exceptional performance, outpacing all other tested frameworks. Its syntactical alignment with React positions it as an optimal choice for developers with a background in React. Furthermore, for React developers seeking a framework that offers similar features and syntax but with significantly enhanced performance, Solid emerges as an excellent choice.

The framework has a relatively fast learning curve and ranks among the fastest frameworks for coding, making it accessible to both inexperienced and experienced developers.

6.3.4.2 Negatives

Solid's foundational relationship with React also means that it inherits React's challenges. These include the absence of an opinionated methodology for accomplishing certain tasks and concerns related to the scalability and maintenance of large projects. As one of the more recent frameworks, Solid's community is still relatively small. This can present challenges for developers who frequently rely on extensions, as Solid currently offers limited alternatives. In some instances, developers may encounter extensions that are no longer supported or maintained. Furthermore, Solid's documentation proved to be incomplete and lacking in many areas.

7 Limitations

7.1 Prior experience

One limitation to consider is that all developers involved in the project have prior experience with React. While React itself is not included in the experiment suite, it is worth noting that the Solid framework shares similarities with React in terms of its structure and state management approach. In theory, the knowledge from React could therefore impact the Solid experiment to some degree, both in relation to results and perception of the framework.

Additionally, the group consists of members with varying levels of experience working with TypeScript. In particular, the inexperienced developers noted that some tasks took particularly long to solve due to TypeScript-related errors, and not only because of learning the framework itself.

7.2 Measuring time

During the last two experiments, we noticed a significant factor that could play a role in the time spent per framework and the associated tasks. The time was lower for some developers, during the last two experiments even though the frameworks felt hard to work with. Although we implemented the functionality in a different framework, our prior experience building the same things would potentially influence the time making it easier to implement the functionality as we complete more experiments. This could in theory have been avoided by developing different tasks and functionalities for each experiment. However, this approach would have introduced new limitations to the experiments, since it is almost impossible to create four different experiments with equal difficulty, that would be comparable and measurable.

7.3 Evaluating documentation

When evaluating the documentation of a framework while doing addons, it is important to consider that time should not be the only determining factor for assessing the quality of a framework's documentation. In the case of Angular addons, a developer may have spent lower time on documentation because it was more convenient to refer to existing code and replicate its structure. However, this approach may not provide the same level of comprehensive understanding that would be gained from thoroughly studying the documentation.

7.4 Small dataset

An important limitation to consider is the limited amount of data that can be collected by just four developers within the timeframe of this project. This limitation could affect the accuracy and comprehensiveness of the final analysis and dataset. To obtain a more reliable and comprehensive understanding of each framework, it would be necessary to involve a larger number of developers and a larger number of backlog items for each experiment. This would help to normalize the data and provide a broader perspective on each framework. Especially, the small amount of backlog items in each experiment really makes it hard to evaluate progress and learning over time.

8 Future work

The experiment paves the way for promising future work, which in the end could result in a final product. The envisioned product would be a web interface featuring an algorithm that is capable of considering various factors examined throughout the study. These factors include development time, experience, learning curve, subjective assessments such as framework structure, and more. By inputting project expectations and requirements, users would receive recommendations for the most suitable framework, ensuring the most appropriate frontend framework for their upcoming project.

Conducting additional tests could result in a product that has the potential to assist hobby developers, senior developers, and tech leads in selecting the optimal frontend framework for their upcoming projects.

In the future, the product has the potential to also include a wider range of frontend frameworks. Additionally, it could consider the preferred programming language of the developer, moving beyond the limitation of only TypeScript. This is particularly important as different frameworks may be more relevant depending on whether the user is a TypeScript developer or a Python developer.

Furthermore, it is relevant to test and incorporate more metrics that hold relevance for the users, such as security, cross-platform development, and the ease of test integrations. By addressing these factors, the product can offer a more holistic and valuable solution for developers in their decision-making process.

9 Conclusion

Selecting the right framework is critical for the success of any software development project, and this report has explored the factors that developers should consider when making this decision. Our research question aimed to identify the most suitable framework for a given situation based on project scope and timeline. To answer this question, we made a comparative study using architectural prototyping to compare and evaluate existing frontend frameworks based on various essential metrics, including performance, learning curve, development speed, community, and extensions.

This comparative study resulted in an analysis that aimed to provide developers with valuable insights into the trade-offs between different frameworks and help them make informed decisions that align with their project's specific needs and constraints.

Following our analysis, it is clear that each framework comes with its own set of advantages and disadvantages. Consequently, it is crucial for developers and teams to dedicate time to identify the most suitable framework, considering their project's scope and timeline.

In short, Solid stands as an excellent choice for those prioritizing performance and a syntax similar to React. Angular is well-suited for large-scale projects with many developers due to its opinionated structure. Vue proves to be a remarkable choice for developers seeking an enjoyable and fast learning experience, with a big community and longevity. Lastly, Svelte is the perfect option for developers preferring a syntax similar to Vue, but with the advantage of a smaller bundle size.

10 References

References

[1] L. Bass, P. Clements, and R. Kazman, "Why is software architecture important?" In *Software architecture in practice*. Addison-Wesley, pp. 37–37. [Online]. Available: https://edisciplinas.usp.br/pluginfile.php/5922722/mod_resource/content/1/2013%20-%20Book%20-%20Bass%20%

20Kazman-Software%20Architecture%20in%20Practice%20%281%29.pdf.

[2] ibm. “What is an api?” (), [Online]. Available: https://www.ibm.com/topics/api. (accessed: 12.05.2023).

[3] techtarget. “What is an endpoint?” (), [Online]. Available: https://www.techtarget.com/searchapparchitecture/definition/API-endpoint. (accessed: 12.05.2023).

[4] mozilla. “What is a dom?” (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. (accessed: 12.05.2023).

[5] reactjs. “What is a virtual dom?” (), [Online]. Available: https://legacy.reactjs.org/docs/faq-internals.html. (accessed: 12.05.2023).

[6] reactjs. “What is a diffing?” (), [Online]. Available: https://www.geeksforgeeks.org/what-is-diffing-algorithm/. (accessed: 12.05.2023).

[7] mozilla. “What is a tree shaking?” (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Tree_shaking. (accessed: 12.05.2023).

[8] Scrum.org. “What is scrum?” (), [Online]. Available: https://www.scrum.org/resources/what-scrum-module. (accessed: 08.05.2023).

[9] agility.im. “What is a backlog item?” (), [Online]. Available: https://agility.im/frequent-agile-question/what-is-a-backlog-item/. (accessed: 12.05.2023).

[10] Stackoverflow. “Web frameworks and technologies.” (), [Online]. Available: https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-webframe-love-dread. (accessed: 04.05.2023).

[11] TypeScript. “Typescript is javascript with syntax for types.” (), [Online]. Available: https://www.typescriptlang.org/. (accessed: 08.05.2023).

[12] guru99. “What is a benchmark testing in software?” (), [Online]. Available: https://www.guru99.com/benchmark-testing.html. (accessed: 12.05.2023).

[13] M. Gadhiya. “20 best javascript frameworks for 2023.” (), [Online]. Available: https://www.lambdatest.com/blog/best-javascript-frameworks/. (accessed: 08.05.2023).

[14] Coursera. “Most popular programming languages in 2023.” (), [Online]. Available: https://www.coursera.org/articles/popular-programming-languages. (accessed: 08.05.2023).

[15] A. Sugandhi. “What are frontend frameworks?” (), [Online]. Available: https://www.knowledgehut.com/blog/web-development/front-end-development-frameworks. (accessed: 12.05.2023).

[16] Appmaster. “How to select an effective software development framework?” (), [Online]. Available: https://appmaster.io/blog/effective-software-development-framework. (accessed: 08.05.2023).

[17] K. Gharibyan. “How to select an effective software development framework?” (), [Online]. Available: https://www.spiritofsoft.com/how-to-select-a-web-framework/. (accessed: 08.05.2023).

[18] S. SCHERBAK. “Thow product requirements define your project’s success.” (), [Online]. Available: https://freshcodeit.com/freshcode-post/product-requirements. (accessed: 08.05.2023).

[19] E. Mota. “The ultimate guide to translating business requirements into technical requirements.” (), [Online]. Available: https://mamatalkstech.com/the-ultimate-guide-to-translating-business-requirements-into-technical-requirements/. (accessed: 08.05.2023).

[20] L. Bass, P. Clements, and R. Kazman, “Analysis at different stages of the life cycle,” in *Software architecture in practice*. Addison-Wesley, pp. 287–287. [Online]. Available: https://edisciplinas.usp.br/pluginfile.php/5922722/mod_resource/content/1/2013%20-%20Book%20-%20Bass%20%20Kazman-Software%20Architecture%20in%20Practice%20%281%29.pdf.

[21] H. B. C. Jakob Eyvind Bardram and K. M. Hansen. “Architectural prototyping: An approach for grounding architectural design and learning.” (), [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e0f0c2bc37034ed7e124b604045e7c069bdb3e43. (accessed: 28.04.2023).

[22] C. Dictionary. “Meaning of architecture in english.” (), [Online]. Available: https://dictionary.cambridge.org/dictionary/english/architecture. (accessed: 28.04.2023).

[23] I. D. Foundation. “Prototyping.” (), [Online]. Available: https://www.interaction-design.org/literature/topics/prototyping. (accessed: 28.04.2023).

[24] L. Bass, P. Clements, and R. Kazman, "Experiments, simulations, and prototypes," in *Software architecture in practice*. Addison-Wesley, pp. 286–286. [Online]. Available: https://edisciplinas.usp.br/pluginfile.php/5922722/mod_resource/content/1/2013%20-%20Book%20-%20Bass%20%20Kazman-Software%20Architecture%20in%20Practice%20%281%29.pdf.

[25] L. Bass, P. Clements, and R. Kazman, "Understanding quality attributes," in *Software architecture in practice*. Addison-Wesley, pp. 79–79. [Online]. Available: https://edisciplinas.usp.br/pluginfile.php/5922722/mod_resource/content/1/2013%20-%20Book%20-%20Bass%20%20Kazman-Software%20Architecture%20in%20Practice%20%281%29.pdf.

[26] Valamis. "Learning curve." (), [Online]. Available: https://www.valamis.com/hub/learning-curve. (accessed: 08.05.2023).

[27] B. Bruegge and A. H. Dutoit, "Identifying design goals," in *Object-oriented software engineering: Using UML, patterns, and Java*. Prentice Hall, 2010, pp. 249–252.

[28] B. Bruegge and A. H. Dutoit, "Architectural styles," in *Object-oriented software engineering: Using UML, patterns, and Java*. Prentice Hall, 2010, pp. 238–238.

[29] S. Krause. "Benchmarking js-frontend frameworks." (), [Online]. Available: https://www.stefankrause.net/wp/?p=218. (accessed: 08.05.2023).

[30] krausest. "Js-framework-benchmark." (), [Online]. Available: https://github.com/krausest/js-framework-benchmark. (accessed: 08.05.2023).

[31] Wikipedia. "Geometric mean." (), [Online]. Available: https://en.wikipedia.org/wiki/Geometric_mean. (accessed: 08.05.2023).

[32] Google. "Lighthouse." (), [Online]. Available: https://developer.chrome.com/docs/lighthouse/overview/. (accessed: 08.05.2023).

[33] R. Salnik. "Angular vs react. which js framework is better in 2022?" (), [Online]. Available: https://brocoders.com/blog/angular-vs-react/. (accessed: 05.05.2023).

[34] M. Joshi. "Angular vs angularjs." (), [Online]. Available: https://www.browserstack.com/guide/angular-vs-angularjs. (accessed: 05.05.2023).

[35] V. Savkin. "Understanding angular ivy: Incremental dom and virtual dom." (), [Online]. Available: https://blog.nrwl.io/understanding-angular-ivy-incremental-dom-and-virtual-dom-243be844bf36. (accessed: 03.05.2023).

[36] Angular. “Angular.” (), [Online]. Available: https://www.npmjs.com/package/@angular/core. (accessed: 11.05.2023).

[37] Angular. “Angular routing.” (), [Online]. Available: https://angular.io/guide/routing-overview. (accessed: 12.05.2023).

[38] Angular. “Introduction to forms in angular.” (), [Online]. Available: https://angular.io/guide/forms-overview. (accessed: 12.05.2023).

[39] E. You. “First week of launching vue.js.” (), [Online]. Available: https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/. (accessed: 05.05.2023).

[40] E. You. “Vue.” (), [Online]. Available: https://github.com/vuejs/vue. (accessed: 05.05.2023).

[41] Vue. “Vue page.” (), [Online]. Available: https://vuejs.org/. (accessed: 05.05.2023).

[42] V. JS. “Vue partners.” (), [Online]. Available: https://vuejs.org/partners/. (accessed: 05.05.2023).

[43] V. JS. “Rendering mechanism.” (), [Online]. Available: https://vuejs.org/guide/extras/rendering-mechanism.html. (accessed: 05.05.2023).

[44] E. You. “Vue.” (), [Online]. Available: https://www.npmjs.com/package/vue. (accessed: 11.05.2023).

[45] Vercel. “Develop. preview. ship.” (), [Online]. Available: https://vercel.com/. (accessed: 12.05.2023).

[46] Vercel. “Sveltekit on vercel.” (), [Online]. Available: https://vercel.com/docs/frameworks/sveltekit. (accessed: 04.05.2023).

[47] Svelte. “Svelte cybernetically enhanced web apps.” (), [Online]. Available: https://svelte.dev/. (accessed: 04.05.2023).

[48] S. Kostic. “Bundle size javascript and webpack.” (), [Online]. Available: https://blog.devgenius.io/bundle-size-javascript-and-webpack-8861ddd54620. (accessed: 12.05.2023).

[49] R. Harris. “Svelte.” (), [Online]. Available: https://www.npmjs.com/package/svelte. (accessed: 11.05.2023).

[50] R. Carniato. “Solidjs: The tesla of javascript ui frameworks?” (), [Online]. Available: https://ryansolid.medium.com/solidjs-the-tesla-of-javascript-ui-frameworks-6a1d379bc05e. (accessed: 05.05.2023).

[51] R. Carniato. “Solidjs.” (), [Online]. Available: https://github.com/solidjs/solid. (accessed: 05.05.2023).

[52] Javatpoint. “What is vanilla javascript?” (), [Online]. Available: https://www.javatpoint.com/what-is-vanilla-javascript. (accessed: 12.05.2023).

[53] Solid. “Performance focused on both client and server.” (), [Online]. Available: https://www.solidjs.com/. (accessed: 05.05.2023).

[54] SolidJS. “Solid docs.” (), [Online]. Available: https://www.solidjs.com/docs/latest. (accessed: 05.05.2023).

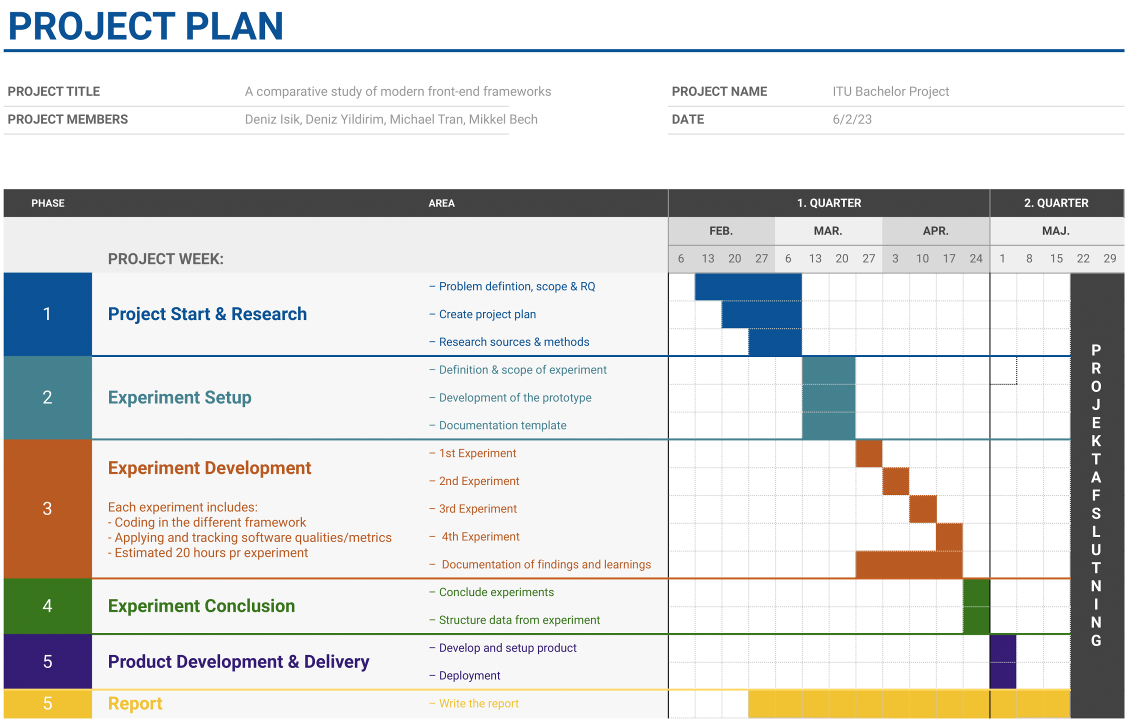
[55] R. Carniato. “Solid-js.” (), [Online]. Available: https://www.npmjs.com/package/solid-js. (accessed: 11.05.2023).

[56] Solid. “Solid router.” (), [Online]. Available: https://github.com/solidjs/solid-router. (accessed: 12.05.2023).

11 Appendix

11.1 Project Plan

The following image is the initial plan that was designed at the beginning of the project. We emphasized the importance of dedicating the necessary time to make certain that the experiments were arranged to guarantee enough time was available for all experiments to be conducted thoroughly. This plan was given the green light by our supervisor at the start of the project.



PROJECT PLAN

PROJECT TITLE	A comparative study of modern front-end frameworks	PROJECT NAME	ITU Bachelor Project
PROJECT MEMBERS	Deniz Isik, Deniz Yildirim, Michael Tran, Mikkel Bech	DATE	6/2/23

PHASE	AREA		1. QUARTER												2. QUARTER				
			FEB.				MAR.				APR.				MAJ.				
	PROJECT WEEK:		6	13	20	27	6	13	20	27	3	10	17	24	1	8	15	22	29
1	Project Start & Research	– Problem defintion, scope & RQ		■	■	■	■												
		– Create project plan			■	■	■												
		– Research sources & methods				■	■												
2	Experiment Setup	– Definition & scope of experiment						■	■										
		– Development of the prototype						■	■										
		– Documentation template						■	■										
3	Experiment Development Each experiment includes: - Coding in the different framework - Applying and tracking software qualities/metrics - Estimated 20 hours pr experiment	– 1st Experiment								■									
		– 2nd Experiment									■								
		– 3rd Experiment										■							
		– 4th Experiment											■						
		– Documentation of findings and learnings								■	■	■	■						
4	Experiment Conclusion	– Conclude experiments												■					
		– Structure data from experiment												■					
5	Product Development & Delivery	– Develop and setup product													■				
		– Deployment													■				
5	Report	– Write the report				■	■	■	■	■	■	■	■	■	■	■	■		

PROJEKTAFSLUTNING

Figure 12: Project Timeline

11.2 The process of the experiment

Below are the questions used to evaluate and conduct the experiments.

"Before" questions

- How much experience do you have with general coding using JavaScript frontend frameworks? On a scale from 1-10.
- How much experience do you have with the framework already? On a scale from 1-10.
- How big is the community of this framework? Count GitHub stars, issues, pull requests, stack overflow tags/questions, etc.
- How many extensions are available for this framework?

"During" questions

- How long did it take to complete each backlog item? How much documentation and coding time did you spend on the backlog item?
- How did you feel while developing the backlog item? On a scale from 1-10.
- Did you need an extension to fulfill your backlog item? How hard was it to use that extension from 1-10?
- Did you require external help from the community to solve your backlog item? and how hard was it to find that help? On a scale from 1-10.

"After" questions

These questions are *only* for the addon experiment.

- How many lines of code did you write to complete the addon-experiment?
- How did it feel modifying the base experiment and extending the code base?

11.3 Experiment 1

Experiment 1 Questions					
Before Experiment					
Software Quality	**Questions**	**Michael - Vue**	**Deniz Y - Vue**	**Deniz I - Angular**	**Mikkel Bech - Angular**
Learning curve	How much experience do you have with the framework?[19]	1/10	1/10	1/10	2/10
Learning curve	Are you working on base or addon?	base	base	base	base

Table 28: Overview of questions answered before the experiment

11.4 Experiment 2

Experiment 2 Questions					
Before Experiment					
Software Quality	**Questions**	**Michael - Angular**	**Deniz Y - Angular**	**Deniz I - Vue**	**Mikkel Bech - Vue**
Learning curve	How much experience do you have with the framework?[20]	1/10	1/10	1/10	8/10
Learning curve	Are you working on base or addon?	addon	addon	addon	addon

Table 29: Overview of questions answered before the experiment

[19] 1 = Never worked with the framework before // 10 = Working with the framework professionally

[20] 1 = Never worked with the framework before // 10 = Working with the framework professionally

11.5 Experiment 3

Experiment 3 Questions					
Before Experiment					
Software Quality	**Questions**	**Michael - Solid**	**Deniz Y - Solid**	**Deniz I - Svelte**	**Mikkel Bech - Svelte**
Learning curve	How much experience do you have with the framework?[21]	1/10	1/10	1/10	2/10
Learning curve	Are you working on base or addon?	base	base	base	base

Table 30: Overview of questions answered before the experiment

11.6 Experiment 4

Experiment 4 Questions					
Before Experiment					
Software Quality	**Questions**	**Michael - Svelte**	**Deniz Y - Svelte**	**Deniz I - Solid**	**Mikkel Bech - Solid**
Learning curve	How much experience do you have with the framework? [22]	1/10	1/10	1/10	1/10
Learning curve	Are you working on base or addon?	addon	addon	addon	addon

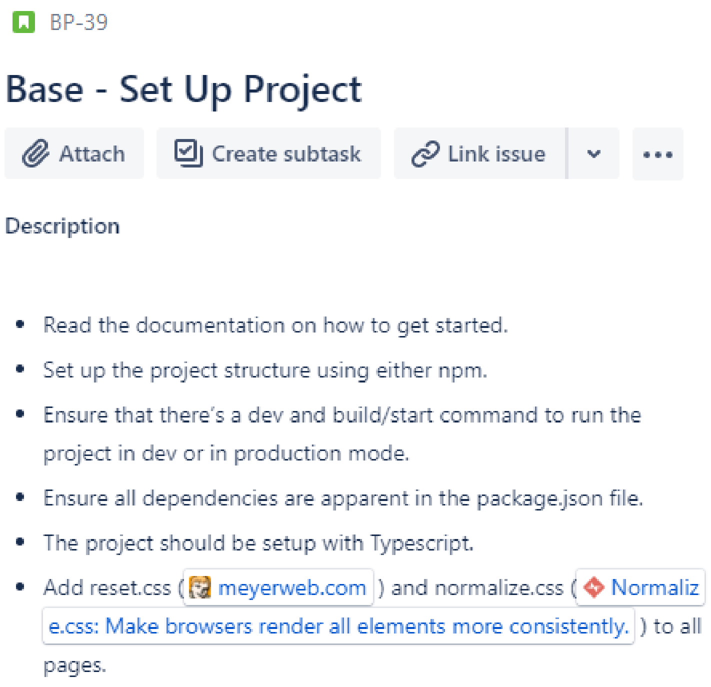
Table 31: Overview of questions answered before the experiment

[21] 1 = Never worked with the framework before // 10 = Working with the framework professionally

[22] 1 = Never worked with the framework before // 10 = Working with the framework professionally

11.7 Backlog Items - Base

11.7.1 Setup Project



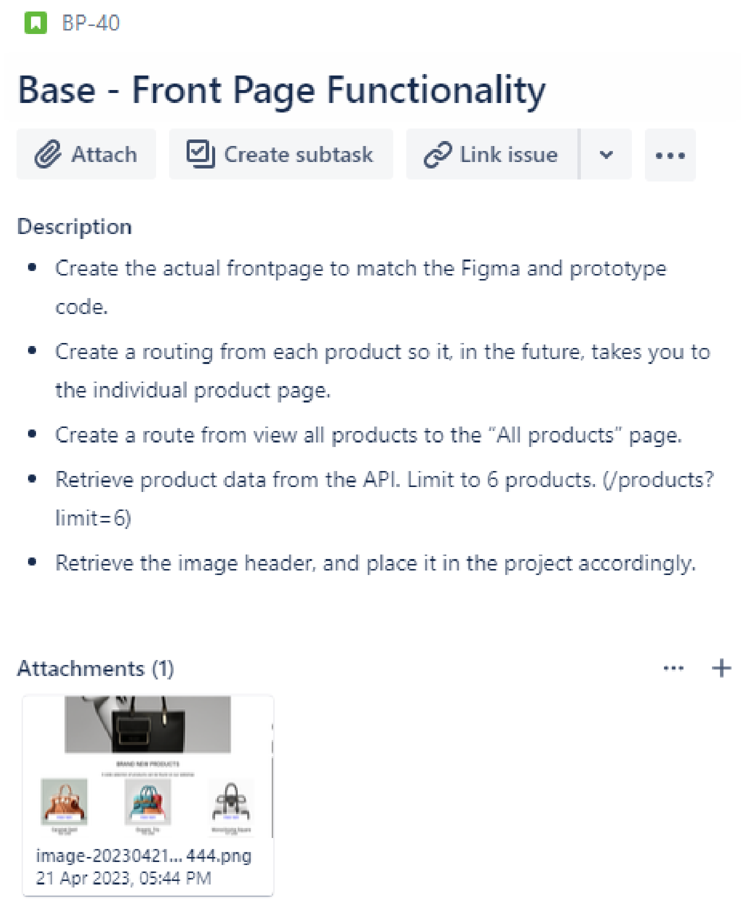
Base - Set Up Project

Description

- Read the documentation on how to get started.
- Set up the project structure using either npm.
- Ensure that there's a dev and build/start command to run the project in dev or in production mode.
- Ensure all dependencies are apparent in the package.json file.
- The project should be setup with Typescript.
- Add reset.css (meyerweb.com) and normalize.css (Normalize.css: Make browsers render all elements more consistently.) to all pages.

Figure 13: Backlog item for the Setting up the project

11.7.2 Front Page



BP-40

Base - Front Page Functionality

Description

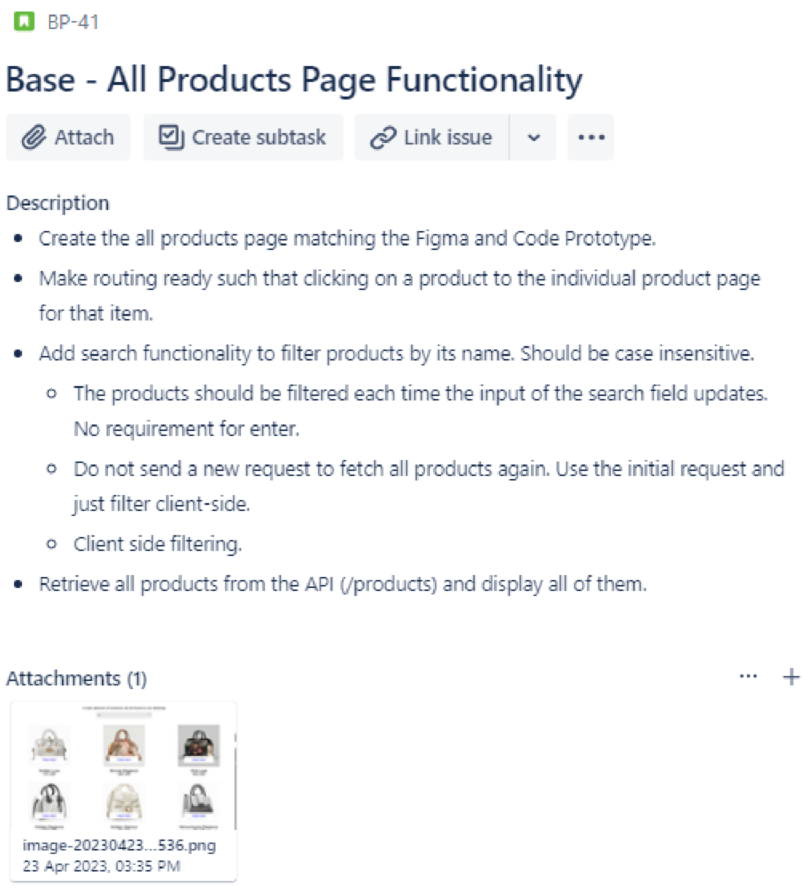
- Create the actual frontpage to match the Figma and prototype code.
- Create a routing from each product so it, in the future, takes you to the individual product page.
- Create a route from view all products to the "All products" page.
- Retrieve product data from the API. Limit to 6 products. (/products?limit=6)
- Retrieve the image header, and place it in the project accordingly.

Attachments (1)

image-20230421... 444.png
21 Apr 2023, 05:44 PM

Figure 14: Backlog item for the Front Page

11.7.3 All Products Page


BP-41

Base - All Products Page Functionality

Description

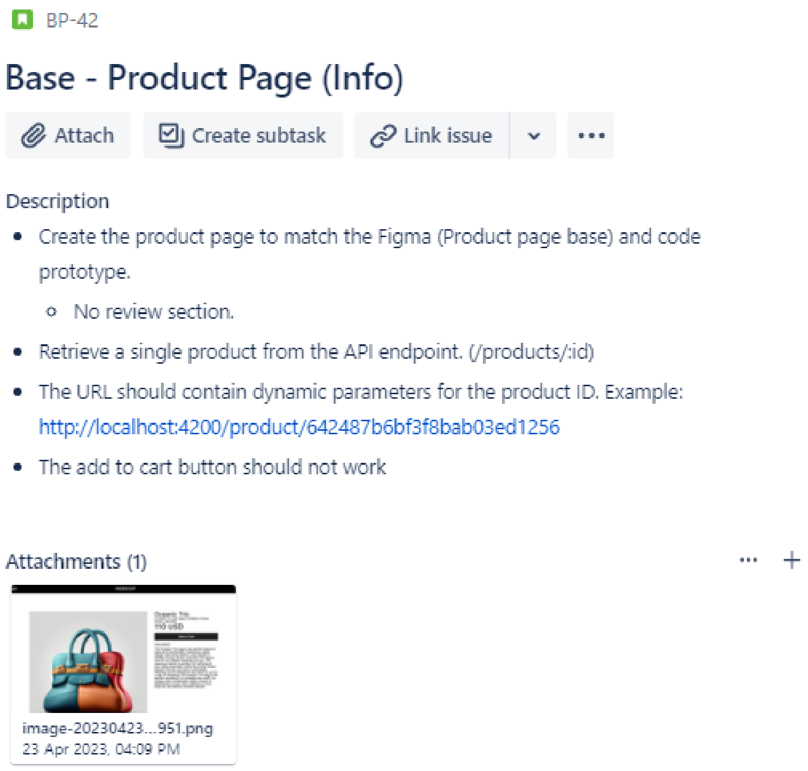
- Create the all products page matching the Figma and Code Prototype.
- Make routing ready such that clicking on a product to the individual product page for that item.
- Add search functionality to filter products by its name. Should be case insensitive.
  - The products should be filtered each time the input of the search field updates. No requirement for enter.
  - Do not send a new request to fetch all products again. Use the initial request and just filter client-side.
  - Client side filtering.
- Retrieve all products from the API (/products) and display all of them.

Attachments (1)

image-20230423...536.png
23 Apr 2023, 03:35 PM

Figure 15: Backlog item for the All Products Page

11.7.4 Product Page


BP-42

Base - Product Page (Info)

Description

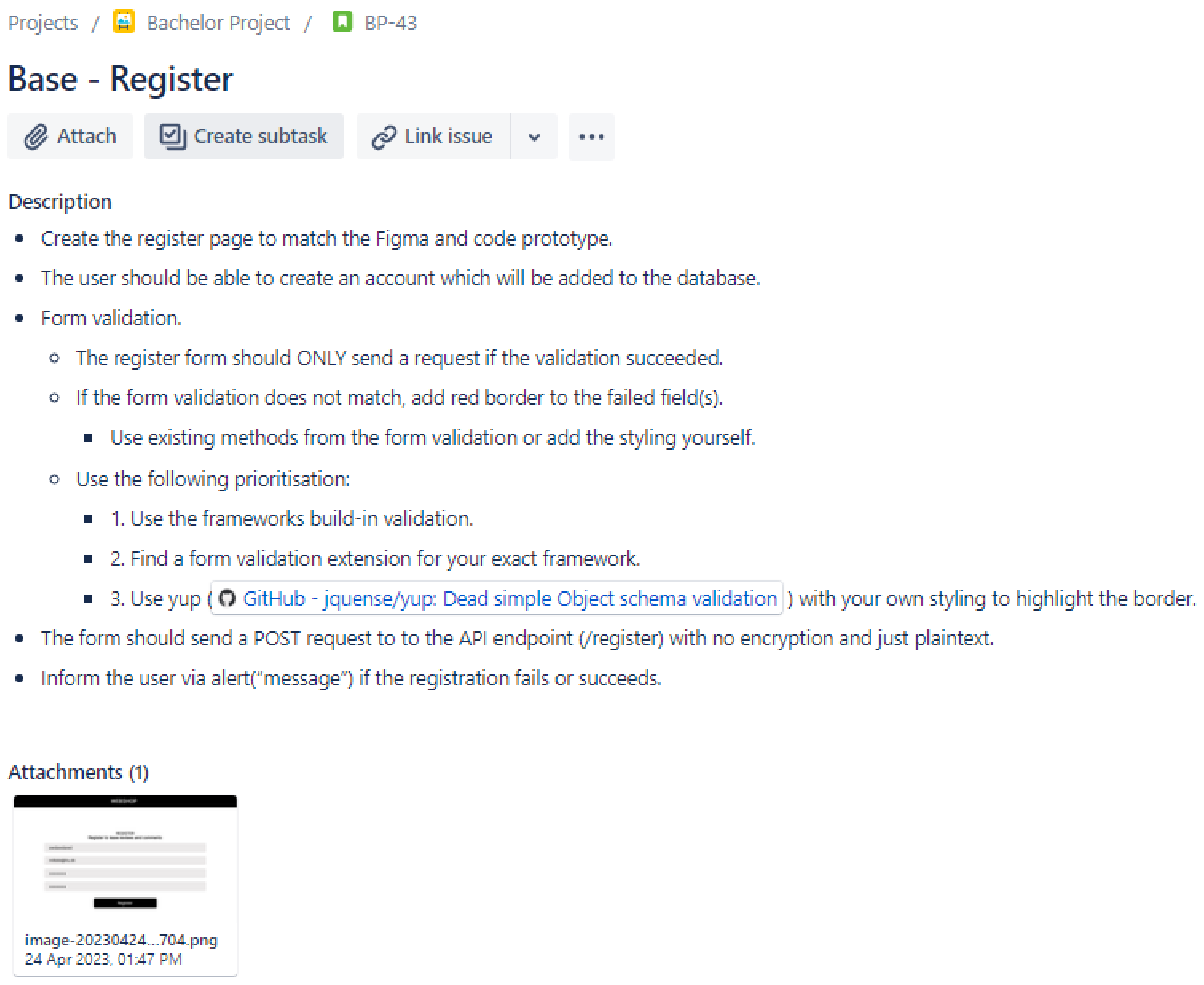
- Create the product page to match the Figma (Product page base) and code prototype.
    - No review section.
- Retrieve a single product from the API endpoint. (/products/:id)
- The URL should contain dynamic parameters for the product ID. Example: http://localhost:4200/product/642487b6bf3f8bab03ed1256
- The add to cart button should not work

Attachments (1)

image-20230423...951.png
23 Apr 2023, 04:09 PM

Figure 16: Backlog item for the Product Page

11.7.5 Register Page


Projects / Bachelor Project / BP-43

Base - Register

Description

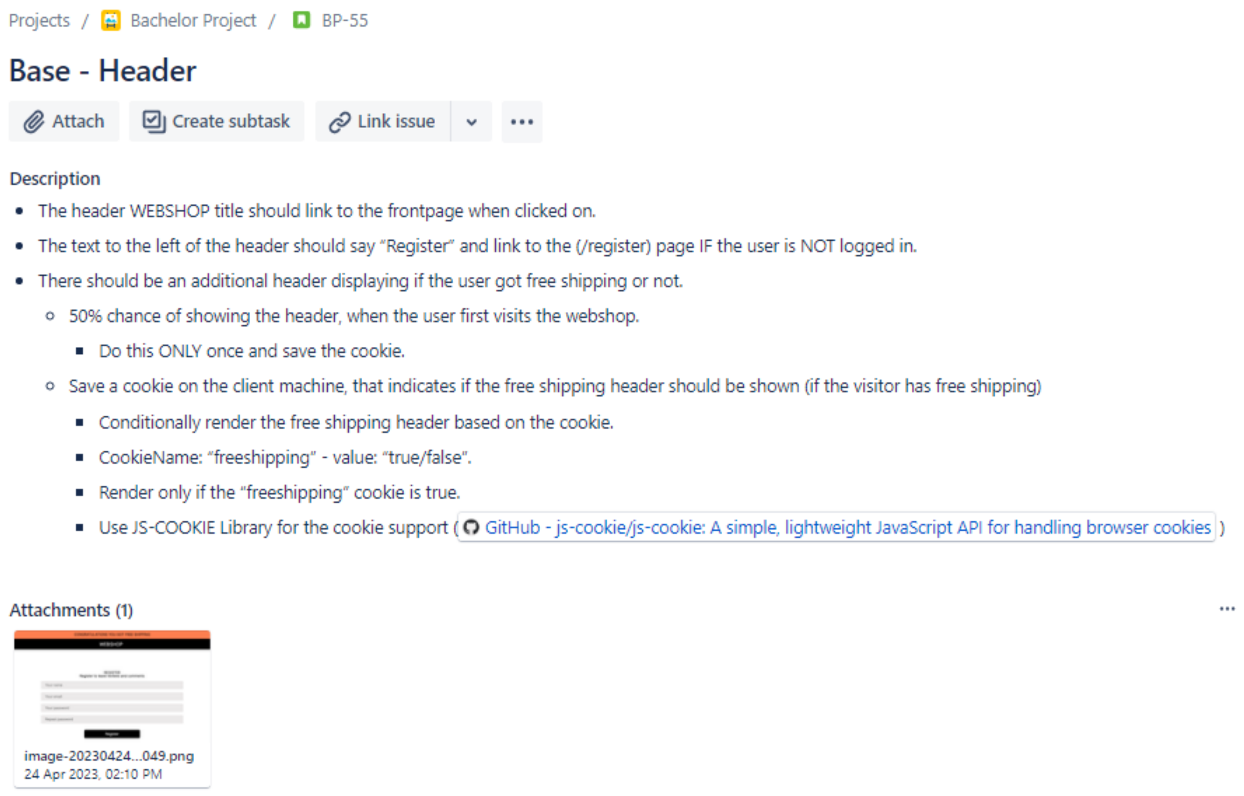
- Create the register page to match the Figma and code prototype.
- The user should be able to create an account which will be added to the database.
- Form validation.
  - The register form should ONLY send a request if the validation succeeded.
  - If the form validation does not match, add red border to the failed field(s).
    - Use existing methods from the form validation or add the styling yourself.
  - Use the following prioritisation:
    - 1. Use the frameworks build-in validation.
    - 2. Find a form validation extension for your exact framework.
    - 3. Use yup (GitHub - jquense/yup: Dead simple Object schema validation) with your own styling to highlight the border.
- The form should send a POST request to to the API endpoint (/register) with no encryption and just plaintext.
- Inform the user via alert("message") if the registration fails or succeeds.

Attachments (1)

image-20230424...704.png
24 Apr 2023, 01:47 PM

Figure 17: Backlog item for the Register Page

11.7.6 Header



Base - Header

Description

- The header WEBSHOP title should link to the frontpage when clicked on.
- The text to the left of the header should say "Register" and link to the (/register) page IF the user is NOT logged in.
- There should be an additional header displaying if the user got free shipping or not.
  - 50% chance of showing the header, when the user first visits the webshop.
    - Do this ONLY once and save the cookie.
  - Save a cookie on the client machine, that indicates if the free shipping header should be shown (if the visitor has free shipping)
    - Conditionally render the free shipping header based on the cookie.
    - CookieName: "freeshipping" - value: "true/false".
    - Render only if the "freeshipping" cookie is true.
    - Use JS-COOKIE Library for the cookie support (GitHub - js-cookie/js-cookie: A simple, lightweight JavaScript API for handling browser cookies)

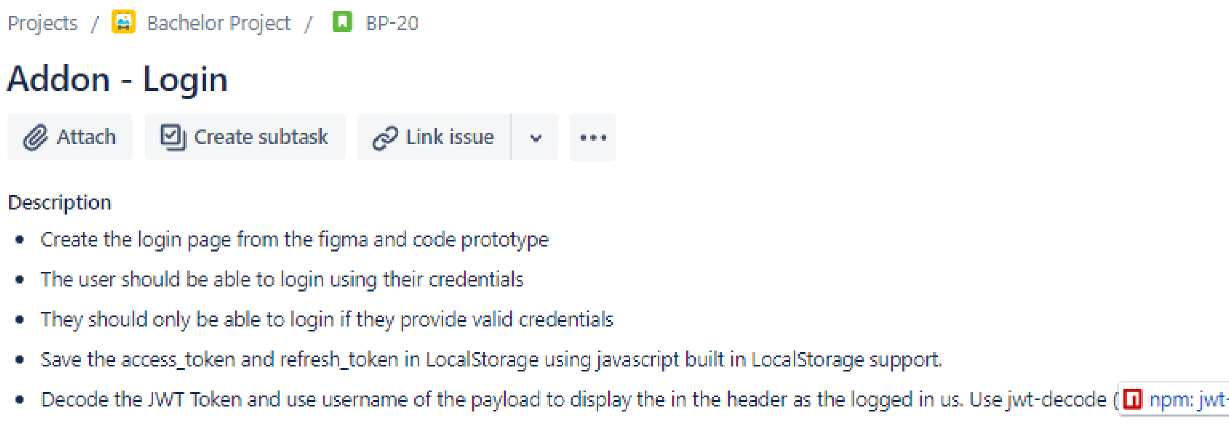
Attachments (1)

image-20230424...049.png
24 Apr 2023, 02:10 PM

Figure 18: Backlog item for the Header

11.8 Backlog Items - Addon

11.8.1 Login Page



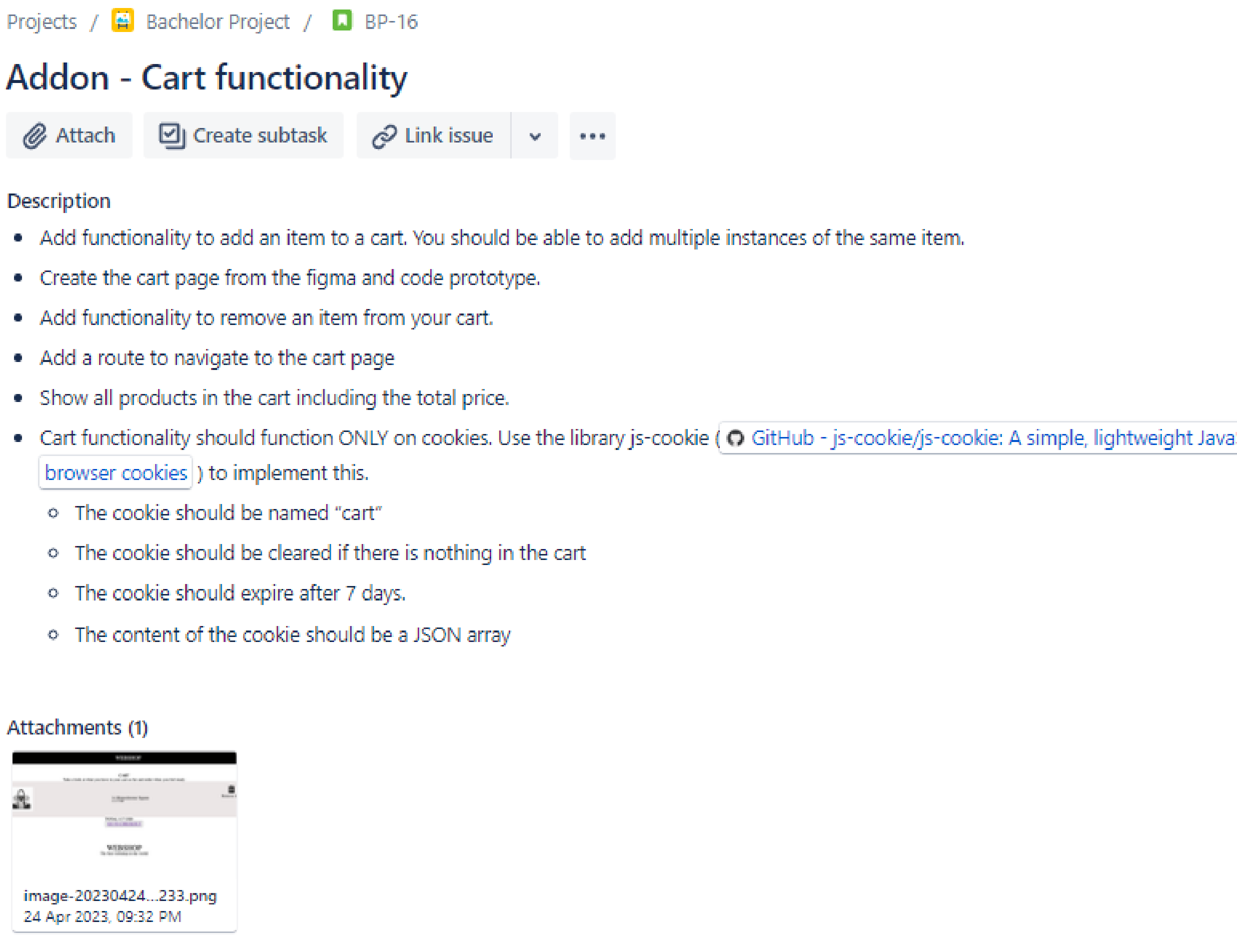
Addon - Login

Description

- Create the login page from the figma and code prototype
- The user should be able to login using their credentials
- They should only be able to login if they provide valid credentials
- Save the access_token and refresh_token in LocalStorage using javascript built in LocalStorage support.
- Decode the JWT Token and use username of the payload to display the in the header as the logged in us. Use jwt-decode (npm: jwt-

Figure 19: Backlog item for the Login Page

11.8.2 Cart Page



Projects / Bachelor Project / BP-16

Addon - Cart functionality

Description

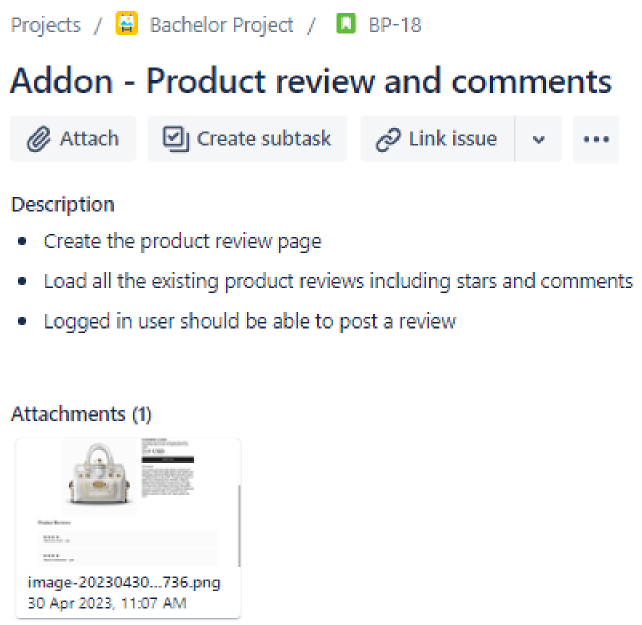
- Add functionality to add an item to a cart. You should be able to add multiple instances of the same item.
- Create the cart page from the figma and code prototype.
- Add functionality to remove an item from your cart.
- Add a route to navigate to the cart page
- Show all products in the cart including the total price.
- Cart functionality should function ONLY on cookies. Use the library js-cookie (GitHub - js-cookie/js-cookie: A simple, lightweight Java browser cookies) to implement this.
  - The cookie should be named "cart"
  - The cookie should be cleared if there is nothing in the cart
  - The cookie should expire after 7 days.
  - The content of the cookie should be a JSON array

Attachments (1)

image-20230424...233.png
24 Apr 2023, 09:32 PM

Figure 20: Backlog item for the Cart Page

11.8.3 Product Review and Comments Page


Projects / Bachelor Project / BP-18

Addon - Product review and comments

Description

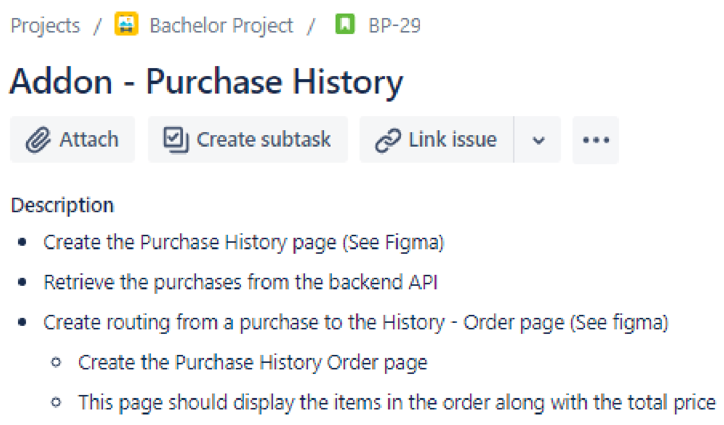
- Create the product review page
- Load all the existing product reviews including stars and comments
- Logged in user should be able to post a review

Attachments (1)

image-20230430...736.png
30 Apr 2023, 11:07 AM

Figure 21: Backlog item for the Product Review and Comments Page

11.8.4 Purchase History Page


Projects / Bachelor Project / BP-29

Addon - Purchase History

Description

- Create the Purchase History page (See Figma)
- Retrieve the purchases from the backend API
- Create routing from a purchase to the History - Order page (See figma)
  - Create the Purchase History Order page
  - This page should display the items in the order along with the total price

Figure 22: Backlog item for the Purchase History Page

11.9 Incremental Dom Visualization

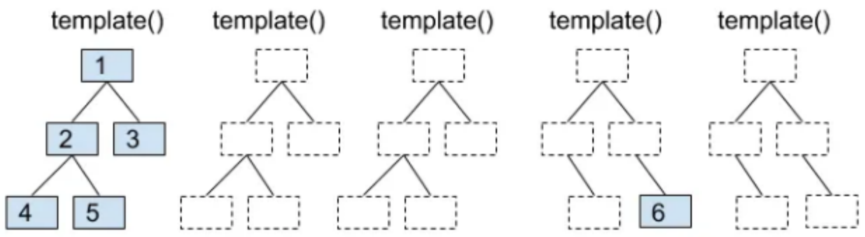


Figure 23: Visualization of incremental DOM

The visualization seen on figure 25 show, that the incremental DOM does not need and memory to rerender the view when no changes are made to the DOM [35]. Only once the ”6” node is addeed on the fourth rerender, the change is allocated to the tree.

Had this been the virtual DOM, the tree would have been recreated everytime a rerender happens, as seen below on figure 24. Even though nothing is changed on the first 2 rerenders, memory is allocated and the whole tree is created.

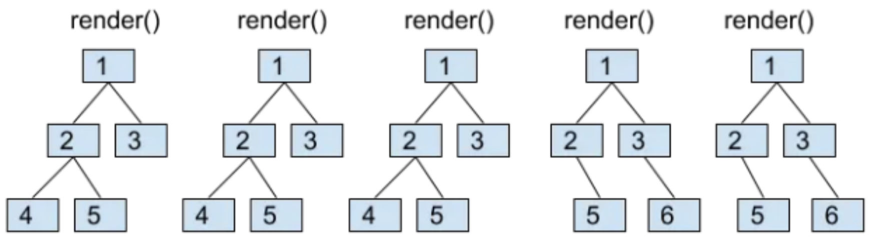


Figure 24: Visualization of what happens on rerender for virtual DOM [35]

11.10 Virtual Dom Visualization

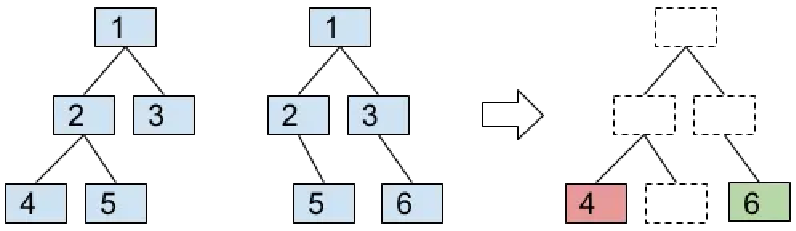


Figure 25: Visualization of Virtual Dom Tree

The visualization shows how React compares the virtual DOM trees created from each component, and matches that into the new virtual DOM tree which gets rendered. Changes to the new virtual DOM tree are then only applied when the old and new virtual DOM trees differ [35].