JavaScript Ecosystem and Node Package Management

Shobowale Tola Joshua

Master's Thesis 2024

Tomas Bata University in Zlín
Faculty of Applied Informatics

Tomas Bata University in Zlín
Faculty of Applied Informatics
Department of Informatics and Artificial Intelligence

Academic year: 2023/2024

ASSIGNMENT OF DIPLOMA THESIS
(project, art work, art performance)

Name and surname:	**Tola Joshua Shobowale**
Personal number:	**A21788**
Study programme:	**N0613A140023 Information Technologies**
Specialization:	**Software Engineering**
Type of Study:	**Full-time**
Work topic:	**JavaScript ekosystém a Node Package Management**
Work topic in English:	**JavaScript Ecosystem and Node Package Management**

Theses guidelines

1. Understand the programming language JavaScript, its structure, frameworks, and production usage.
2. Take an insight into previous related works.
3. Analyze the management of JavaScript codes and data using Node Package Manager.
4. Build a movie-app application using numerous APIs e.g., Netflix, ReactJS, NPM, GitHub, Mongo DB for the database.
5. Evaluate the obtained results from the application, and suggest the best practices for using JavaScript and any of its frameworks and dependencies in software development.

Form processing of diploma thesis:	**printed/electronic**
Language of elaboration:	**English**

Recommended resources:

1. GANDHI, Raju. JavaScript next: your complete guide to the new features Introduced in JavaScript, starting from ES6 to ES9. [United States]: Apress, [2019]. ISBN 978-1-4842-5393-9.
2. CROCKFORD, Douglas. JavaScript: the good parts. Sebastopol: O´Reilly, 2008. ISBN 978-0-596-51774-8.
3. Software Engineering – Ian Sommerville: Tenth Edition [online], 2015. [cit. 2022-01-13]. Available from: https://iansommerville.com/software-engineering-book/.
4. DOOLEY, John F. Software Development, Design and Coding: With Patterns, Debugging, Unit Testing, and Refactoring. 2nd ed. 2017. Imprint: Apress, 2017. ISBN 9781484231524.
5. ECO, Umberto, Caterina MONGIAT FARINA a Geoff FARINA. How to write a thesis. Cambridge, Massachusetts: MIT Press, [2015]. ISBN 978-0262527132.

Supervisors of diploma thesis:	**Ing. Bc. Pavel Vařacha, Ph.D.** Department of Informatics and Artificial Intelligence
Date of assignment of diploma thesis:	**November 5, 2023**
Submission deadline of diploma thesis:	**May 13, 2024**

Tomas Bata University in Zlín
Faculty of Applied Informatics

L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. m.p.
Dean

prof. Mgr. Roman Jašek, Ph.D., DBA m.p.
Head of Department

In Zlín January 5, 2024

ABSTRAKT

Czech abstract

Tato práce se zabývá ekosystémem JavaScriptu a důsledky správy balíčků Node, přičemž se zaměřuje na rozhodovací procesy při výběru rámců JavaScriptu a na omezení, která jsou vlastní aplikacím založeným na JavaScriptu. Hlavním cílem bylo zodpovědět dvě klíčové obchodní otázky:

1. Jaké faktory ovlivňují výběr frameworku/knihovny JavaScriptu softwarovými inženýry při jejich obchodním rozhodování o vytvoření webové aplikace založené na JavaScriptu?

2. Jakým omezením dnes čelí aplikace vytvořené pomocí programovacího jazyka JavaScript?

Pro získání informací o těchto otázkách byl proveden průzkum v rámci komunity uživatelů jazyka JavaScript, který přinesl výrazné preference frameworku React. Studie ukázala, že rozsáhlá podpora komunity, bohatý ekosystém nástrojů a škálovatelnost jsou významnými faktory, které podporují přijetí frameworku React mezi vývojáři.

Jako praktická aplikace výsledků výzkumu byla vyvinuta aplikace pro streamování filmů s názvem "Wura". Aplikace byla vytvořena pomocí frameworku Next.js React, přičemž interakce na straně serveru byly řešeny pomocí PrismaDB a MongoDB. Funkce ověřování, včetně přihlašování a odhlašování, byly implementovány pomocí Google a GitHub OAuth. Výzkum poukazuje na faktory, kterými se softwaroví inženýři řídí při výběru frameworků, a zdůrazňuje význam podpory komunity, flexibility a integračních schopností. Zabývá se také běžnými omezeními, jako jsou bezpečnostní rizika v open-source balíčcích a výkonnostní omezení, a poskytuje poznatky o osvědčených postupech pro vytváření robustních aplikací JavaS-cript. Tato práce přispívá k hlubšímu pochopení ekosystému JavaScriptu a nabízí cenné vodítko pro obchodní rozhodování při vývoji softwaru.

Klíčová slova: Balíčky s otevřeným zdrojovým kódem, NPM, Google Oauth, Github Oauth, MongoDB, React Js

ABSTRACT

English abstract

This thesis explores the JavaScript ecosystem and the implications of Node package management, focusing on the decision-making processes behind selecting JavaScript frameworks and the limitations inherent to JavaScript-based applications. The primary objective was to address two key business questions:
1. What factors influence software engineers' choice of JavaScript framework/library when making business decisions for building a JavaScript-based web application?
2. What limitations do applications built with the JavaScript programming language face today?

To gain insights into these questions, a survey was conducted within the JavaScript community, yielding a strong preference for the React framework. The study revealed that React's extensive community support, rich ecosystem of tools, and scalability are significant factors that drive its adoption among developers.

As a practical application of the research findings, a movie streaming application called "Wura" was developed. The app was built using a Next.js React framework, with server-side interactions handled via Prisma DB and MongoDB. Authentication features, including sign-in and sign-out, were implemented using Google and GitHub OAuth.
The research highlights the factors that guide software engineers' framework choices, emphasizing the importance of community support, flexibility, and integration capabilities. It also addresses common limitations, such as security risks in open-source packages and performance constraints, providing insights into best practices for building robust JavaScript applications. This thesis contributes to a deeper understanding of the JavaScript ecosystem, offering valuable guidance for business decision-making in software development.

Keywords: Open Source Packages, NPM, Google Oauth, Github Oauth, MongoDB, React

ACKNOWLEDGEMENTS

I am highly grateful to Ing. Pavel Vařacha, Ph.D. for his invaluable contribution and immerse support to the completion of my master’s Thesis. His Expertise have been very instrumental to the outcome of my research and I am truly honoured for the opportunity to have worked under his supervision.

I hereby declare that the print version of my Master's thesis and the electronic version of my thesis deposited in the IS/STAG system are identical.

CONTENTS

INTRODUCTION

This chapter provides the Background, Research Objectives and the Thesis Structure

BACKGROUND

JavaScript has become extremely dynamic, running client-side single-page applications, powering server-side applications with technologies like Node.js, contributing to desktop applications through Electron, and even being employed in single-board computers like Raspberry Pi. It plays a role in training machine-learning models in browsers using frameworks like TensorFlow.js. With its capability to run on every desktop, laptop, and phone with an embedded browser, JavaScript is arguably the most widely deployed language globally.

In 2015, the introduction of ECMAScript 2015 (ES6) marked a significant milestone. ES6 brought a host of new features and syntactic changes, aiming to modernize JavaScript for the web. The goal was to enhance the developer experience, incorporating constructs familiar to developers from other languages. The establishment of a yearly release cadence for future editions signaled a commitment to ongoing evolution and maturation of the language. However, JavaScript's journey spans 24 years, evolving from a scripting language for web page animation to a language used across various domains. Initially criticized for being quirky and error-prone, its reputation prompted the attention of the TC39, the committee responsible for JavaScript's evolution.

The analysis of extensive codebases, such as those found on GitHub, provides valuable insights into how programmers utilize or misapply programming languages, the prevalent programming patterns in different development ecosystems, and opportunities for language or runtime enhancements. Researchers have been focusing on software repositories since the mid-2000s, exploring aspects like code history in version control systems, build artifacts, library dependencies, developer activity, and popularity indicators of files and projects.

JavaScript, the universal language of the web, plays a crucial role in shaping the digital landscape. It's a versatile scripting language primarily executed on the client-side of web browsers. JavaScript enables developers to create dynamic and interactive web pages, offering a wide range of functionalities such as DOM manipulation, event handling, asynchronous requests, and data validation. These capabilities empower developers to build captivating and responsive web applications, enhancing the user experience.
The Internet and World Wide Web are essential aspects of daily life, hosting billions of websites and connecting countless devices. JavaScript plays a pivotal role in enabling dynamic and interactive web experiences, ranging from basic text to multimedia content.
JavaScript's ecosystem has expanded significantly, with its application in server-side development through Node.js, native desktop applications using GitHub's Electron, and mobile apps via technologies like Facebook's React Native or Apache Cordova. The prevalence of NPM as the primary module repository underscores its importance, surpassing other repositories like Maven Central, Rubygems, or Packagist by a considerable margin.

Research Objectives

Since the inception of the JavaScript language, the landscape of website content management has undergone a transformative shift, revolutionizing the way modern websites are utilized. Effectively managing website content is crucial for security and user-friendliness. However, achieving an efficient content management system often entails extensive and time-consuming development, leading to code rewriting and reuse in subsequent projects. As developers invest countless hours in building from scratch, the need for expedited and streamlined development methods becomes apparent.
Web frameworks have emerged as indispensable tools for crafting powerful web applications, not just for content management but for overall system effectiveness. The JavaScript programming language boasts an extensive array of frameworks such as Angular, ReactJs, VueJs, Svelte to name a few. Each offering unique approaches to development challenges. These frameworks empower software engineers to create sophisticated web applications swiftly, freeing them from the arduous task of constructing individual system cores from the ground up.

This paper aims to dissect the extensive functionality of the JavaScript language and its popular frameworks, exploring their profound impact on modern web and desktop application development. The analysis will delve into the distinctions among these frameworks, showcasing diverse development approaches. Given the abundance of frameworks—numbering over a hundred—this paper will focus on a representative sample, providing insights into their advantages for specific application types based on the requirements of each application.

The aim of these thesis Research is to answer these following Questions;

- What Factors affects the choice of JavaScript framework/library according to Software Engineers in Making Business Decision when Building a JS Web application?
- What Limitations does Application built with the JavaScript programming language face today?

Thesis Structure

In the initial chapter, the paper introduces essential terminology needed for understanding the procedures and technologies pivotal in the development of JavaScript-based web applications. The chapter also outlines the specified criteria chosen for assessing the quality of the selected frameworks.

Chapter 2 delves into an in-depth exploration of prior works related to languages similar to JavaScript and their respective architectures.

Chapter 3, the emphasis now lies in elucidating the criteria utilized in this thesis to assess the JavaScript Stack System, encompassing JavaScript Codes, along with its frameworks, libraries, compilers, and bundlers. We will delve into research regarding the preferred JavaScript framework among developers for Web Application development and elucidate the reasons behind its popularity. The insights gleaned from this chapter will serve as the foundation for our argument in Chapter 4, where we advocate for the utilization of the aforementioned framework in constructing the Movie App.

Chapter 4 is dedicated to the design of the Movie web application. It explains the software and other resources utilized, providing insights into the chosen development process for building the application.

As we progress to Chapter 5, 6 , the paper reviews the project results, reflecting on errors made during the development process and suggesting alternative approaches. The findings are then applied to address the questions posed in the research Objectives, offering a seasoned perspective on contemporary software development. Additionally, the chapter furnishes recommendations for aspiring developers embarking on their future projects.

I. THEORY

1 THE EMERGENCE OF JAVASCRIPT

Initially called LiveScript, JavaScript underwent a name change by Netscape, possibly influenced by the buzz surrounding Java. JavaScript debuted in Netscape 2.0 in 1995 under the name LiveScript. The language's versatile core has been integrated into Netscape, Internet Explorer, and various other web browsers [1].
JavaScript's history traces back to the early days of the Internet. Even with the creation of the first browser, Worldwide Web, in 1990 by Tim Berners-Lee, the Internet remained a relatively unfamiliar concept to the mainstream audience. In 1993, the Internet witnessed significant growth with the launch of the commercial Mosaic browser. Developed by Marc Andreessen and Eric Bina during their tenure at the National Center for Supercomputing Applications, the Mosaic browser made its debut in January 1993 for the UNIX system, followed by releases for the Macintosh and Windows systems in September of the same year. Distinguished by a graphical interface enabling inline image display with document text, it also introduced the concept of the Document Object Model (DOM) structure within the browser [2].

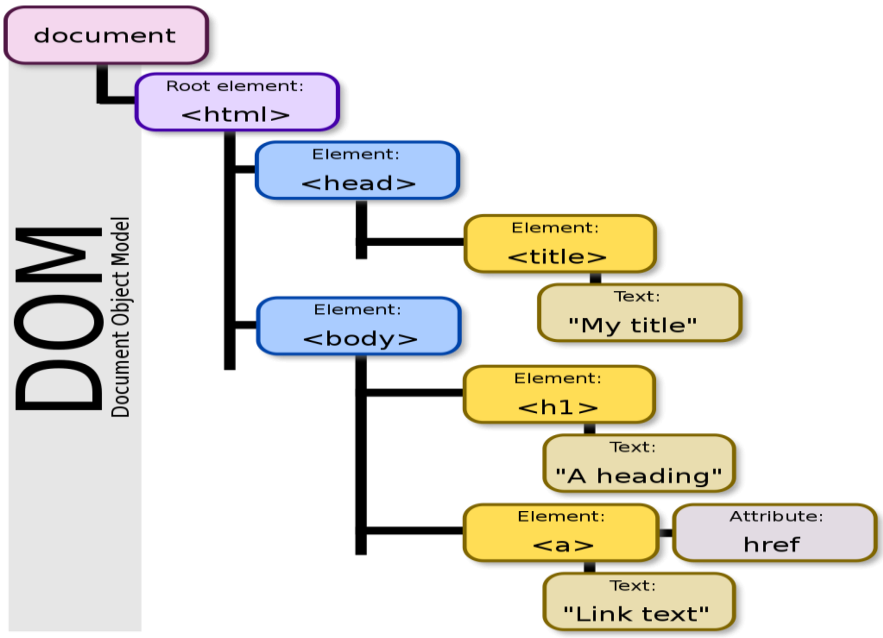


Figure 1. DOM Architecture

1.1 Standardization of JavaScript Programming language (ECMASCRIPT)

During the inception of JavaScript, Netscape led its development while Microsoft introduced their own version known as JScript. This led to a push for standardization. Netscape, being cautious of Microsoft's influence, enlisted Brendan Eich in 1996 to create a specification. Eich revamped the initial JavaScript engine, Mocha, into SpiderMonkey and established the JavaScript 1.2 specification.
From the outset of the JavaScript project in 1995, it was evident that an open scripting standard for the web was necessary. Microsoft's involvement underscored this need. Netscape and Sun Microsystems aimed to collaborate with Microsoft without succumbing to domination by Microsoft's technologies like VBScript. While organizations like W3C and IETF were considered for standardization, they weren't the right fit for JavaScript. Netscape eventually

connected with (Ecma) through a personal contact, which proved to be a suitable partnership [1].

1.1.1 EcmaScript 1 (ES 1) – 1997

The inaugural gathering of the Technical Committee 39 (TC39) took place in September 1996, with a total of thirty attendees. David Stryker, representing Netscape, put forward a proposal during the meeting. He suggested that the primary objective should be to develop a specification that closely aligns with existing implementations at that time. Stryker emphasized that any additions or extensions to the language should be postponed for future consideration. This approach aimed to accurately capture the current state of the language without introducing significant deviations. At the outset, Thomas Reardon of Microsoft advised the committee against redundant efforts by suggesting that the standardization of an HTML object model should be delegated to the W3C. This recommendation stemmed from the observation that while the core features of both Netscape and Microsoft were similar, their respective HTML APIs differed. These initial limitations significantly influenced TC39's focus, directing efforts toward the development of platform-independent features, a principle that continues to guide the group today. The intention was to establish a formal language definition while allowing room for competitive innovation outside of it. This approach has proven effective in practice, fostering healthy competition, particularly in areas such as interpreter performance.
The early challenges surrounding the creation of the initial specification were characterized by a narrative of political maneuvering and intrigue. Initially, Netscape, Borland, and Microsoft each presented their own specifications, necessitating the amalgamation of these disparate documents into a single cohesive standard. The central issue revolved around determining which specification would serve as the foundation for further refinement. Given Ecma's familiarity with Microsoft Word, they opted to commence work based on Microsoft's version of the initial specification, rather than Netscape's.
The timeline for the specification work was established, aiming for an initial draft by January 1997 and a final version by April 1997. Features common to all three proposals were readily accepted, while discrepancies required reconciliation. Unique features were earmarked for consideration under Proposed Extensions at a later stage. Additionally, a key priority was to minimize changes that would necessitate alterations to existing applications, establishing a guiding principle for subsequent standard editions. One unique aspect of the JavaScript standard was its use of a pseudocode-style definition language. This language helped people understand how JavaScript functions without needing to learn a specific programming language. It acted as a bridge between written language and programming, making it easier to grasp the concepts.

To meet the April deadline for finalizing the standard, the TC39 group met regularly. They created test cases for any unclear situations and tested them with different software systems to see how JavaScript behaved. If there were differences in behavior among these systems, the group had to agree on a consistent behavior and specify it. Some of these decisions still influence how programmers write JavaScript code today.
Although the group missed the April deadline by a month, they finished their work on May 2, 1997. The resulting document became known as ECMA Standard 262, or ECMA-262 for short. After some minor edits, it was submitted for the ISO fast-track process. The first version of the standard, called ES1, was published on September 10, 1997 [3].

1.1.2 ECMAScript 2 (ES2) - 1998

ES2 represented a minor update aimed at refining existing features and providing clarity to the language's specification. This version did not introduce any significant new features but instead focused on enhancing the functionality and coherence of JavaScript.

1.1.3 ECMASCRIPT 3 (ES3) – 1999

ECMAScript 3 (ES3) was launched in late 1999 – early 2000, marking a significant milestone in the evolution of JavaScript. This version brought forth several crucial features that greatly enhanced the language's capabilities [4].
Firstly, ES3 introduced the try/catch/finally statement. This provided developers with a powerful mechanism for handling errors and exceptions within their code, improving the robustness and reliability of JavaScript applications.
ES3 incorporated support for Regular Expressions. Regular Expressions are powerful tools for pattern matching and text manipulation, allowing developers to perform complex string operations with ease.
Another notable addition in ES3 was the ability to define functions using both function declaration and function expression syntax. This flexibility provided developers with more options for structuring their code and expressing logic, contributing to improved code organization and readability.
In ES3, objects are created using the constructor function pattern. This involves defining a function that serves as a constructor for the object, and then using the new keyword to instantiate new instances of the object. This method remains prevalent in modern JavaScript development practices. ES3 retains significance within the JavaScript ecosystem. It continues to be utilized extensively in legacy codebases and older web applications. Moreover, modern JavaScript engines maintain support for ES3 syntax, ensuring compatibility with both past and present versions of the language [5].

Limitations

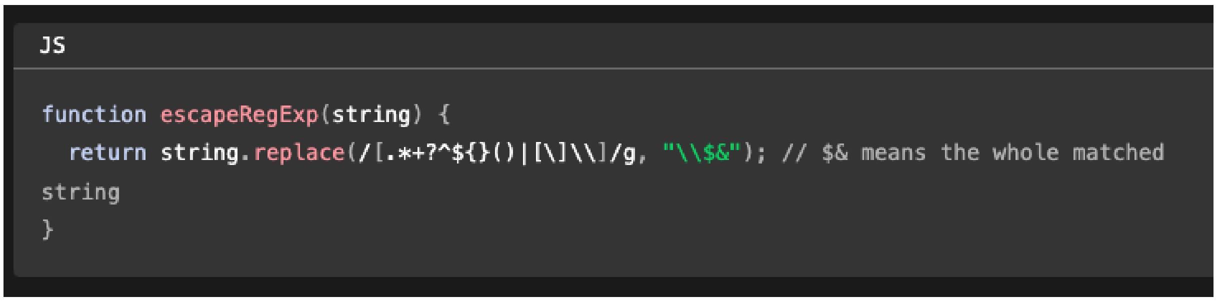
- It lacked support for contemporary programming concepts like classes and modules. Additionally, it contains certain idiosyncrasies such as the with statement and implicit global variables, which can complicate the writing of maintainable code.

Notable features

- New Object Methods
  - ➔ Oject.Create
  - ➔ Object.Keys
  - ➔ Object.defineProperty

- Regular Expressions(Regex)

It introduced support for regular expressions, empowering developers with advanced capabilities for matching text patterns.



```
JS

function escapeRegExp(string) {
  return string.replace(/[.*+?^${}()|[\]\\]/g, "\\$&"); // $& means the whole matched
string
}
```

“*This function, `escapeRegExp`, takes a string as input and returns a new string with special characters escaped using backslashes. The regular expression `[.*+?^${}()|[\]\\]` matches any of the characters `.`, `*`, `+`, `?`, `^`, `$`, `{`, `}`, `(`, `)`, `[`, `]`, `\`, and `|`. Within the replacement string `"\\$&"`, the `$&` signifies the entire matched substring. This ensures that any special characters within the input string are properly escaped, preventing them from being interpreted as metacharacters in a regular expression*. [6]”

- Try/Catch

ES3 brought in structured error handling with the introduction of the try, catch, and finally statements.

EcmaScript 3 played a crucial role in advancing the JavaScript language, introducing key features that continue to be foundational elements of modern JavaScript development.

1.1.4 ECMAScript 4 (ES4) – Discontinued

Initially planned for release, ES4 never saw the light of day due to disagreements and difficulties encountered during its development. Instead, the focus shifted towards crafting smaller, more manageable incremental updates.

1.1.5 ECMAScript 5 (ES5) - 2009

ES5 represented a significant advancement for JavaScript, introducing several pivotal features:

- **Strict Mode**: ES5 introduced "strict mode," implementing stricter rules to identify common coding errors and elevate code quality.

- **JSON Support**: ES5 elevated JSON (JavaScript Object Notation) to a primary status, streamlining data interchange processes.

- **Higher-Order Functions**: ES5 bolstered support for higher-order functions, simplifying the creation of expressive and succinct code

- **Array Methods**: ES5 enriched the language with potent array methods such as forEach, map, filter, and reduce, enhancing array manipulation capabilities.

- **Function Bind**: The introduction of the bind method allowed functions to be precisely bound to specific contexts, granting developers meticulous control over the behavior of the **this** keyword.

Summary of the ES5 Features according to (w3schools.com) [7].

- "use strict"

- String[number] access
- Multiline strings
- String.trim()
- Array.isArray()
- Array forEach()
- Array map()
- Array filter()
- Array reduce()
- Array reduceRight()
- Array every()
- Array some()
- Array indexOf()
- Array lastIndexOf()
- JSON.parse()
- JSON.stringify()
- Date.now()
- Date toISOString()
- Date toJSON()
- Property getters and setters
- Reserved words as property names
- Object methods
- Object defineProperty()
- Function bind()
- Trailing commas

1.1.6 ECMAScript 6 (ES6) - 2015

ES6, also known as **ES2015**, revolutionized JavaScript by introducing a wealth of features that modernized the language:

- **Classes**: ES6 introduced class syntax, providing a more structured way to create and inherit object prototypes.

- **Promises**: Promises were introduced in ECMAScript 2015, offering a cleaner syntax for handling asynchronous operations and improving readability in asynchronous programming.

- **Arrow Functions**: Arrow functions offer a concise syntax for writing anonymous function expressions, particularly useful for simplifying code and enhancing readability, especially in scenarios with straightforward functions.

- **Modules**: ES6 introduced modules, enabling developers to organize code into reusable and maintainable components. Modules facilitate the structuring of large codebases and encourage best practices such as separation of concerns and code reusability.

- **Template Literals**: Template literals brought string interpolation to JavaScript, streamlining dynamic string creation.

- **Destructuring Assignment**: ES6 introduced destructuring assignment, allowing values to be extracted from arrays and objects with ease.

1.1.6.1 Features of ES6 and Subsequent Version Updates to the language
JavaScript has continued to evolve beyond ES6, with ES7, ES8, and subsequent versions introducing features such as:

- Object and Array Methods: Improved data manipulation with new methods like Object.entries(), Object.Values(), and array methods like includes().

- Class Improvements: More robust class-based code organization with features like class fields and private methods.

- Optional Chaining: Safe access to nested object properties using optional chaining.

- Nullish Coalescing: Simplified default value assignment for potentially undefined variables with nullish coalescing.

- Proxies: Fine-grained control over object behavior enabled by proxies.

- BigInt: Representation of arbitrary-precision integers facilitated by BigInt.

- Async/Await: Revolutionized asynchronous programming in JavaScript, async and await keywords simplify asynchronous code by resembling synchronous code. Built on top of promises, async/await provides an elegant approach to working with asynchronous operations, introduced as part of ECMAScript 2017 (ES8) [4].

At its essence, a webpage is constructed with three fundamental components: Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript. Regardless of the intricacy and sophistication of the applied technologies and architectural design, everything ultimately converges into HTML, CSS, and JavaScript blocks post the compilation phase. HTML serves as the foundational blueprint, providing structure and content to the webpage, while CSS is employed to craft the visual appearance of its components. Crucially, JavaScript functions as a versatile tool for manipulating all the elements within the webpage.

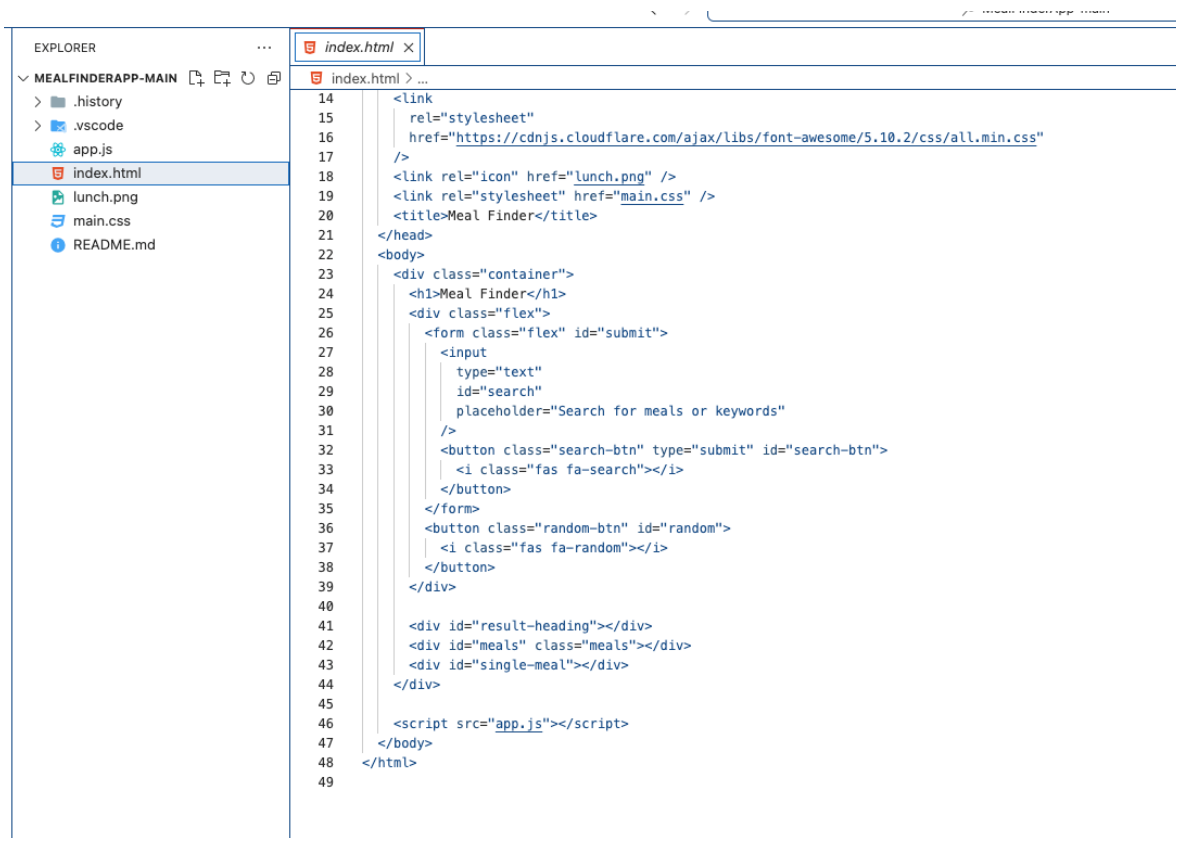
JavaScript stands as a pioneer in front-end development, evolving into a robust and flexible tool accessible to developers of various backgrounds. As of the present, the landscape allows for the creation of diverse applications exclusively using JavaScript. This singular language empowers developers to craft a server with a database and select the front-end for various platforms, including web, mobile, and desktop applications. Remarkably, JavaScript even facilitates the development of machine learning applications.
JavaScript is commonly described as a high-level, multi-paradigm, non-blocking, and asynchronous programming language. These terms, along with others like garbage-collected, interpreted, single-threaded, and concurrent, encapsulate the essential characteristics of JavaScript. Nevertheless, these descriptors can be overly abstract, making it challenging for average readers or individuals new to programming to grasp the full scope of the language.

"JavaScript is very powerful and can be used to create almost any kind of browser-based app, it can be time-consuming and repetitive to code every app from scratch. That's where libraries and frameworks come in—they encode some common patterns and best practices for creating apps. By creating a platform to build apps on top of, JavaScript libraries and frameworks save developers a lot of time" [2].

Given its considerable impact on technology, it is evident that JavaScript has played a pivotal role in driving innovation within the field. Despite its modest origins a quarter-century ago as a prototype scripting language, JavaScript has evolved into a powerhouse that fuels numerous emerging technology domains on the Internet. Beyond its initial role, it has become a versatile tool, extending its influence to various programming aspects outside its original purpose.

1.2 JavaScript a Scripting Language (High Level Programming language)

JavaScript, as a scripting language, proves advantageous for web developers due to its less complex and smaller scripts compared to other desktop languages like Java or C++. The use of a scripting language results in a faster development process.
JavaScript operates as an interpreted language, eliminating the need for pre-compilation of the source code before transmitting it to the browser. With an interpreter, the raw JavaScript code can be directly executed. Moreover, JavaScript is dynamically typed, distinguishing it from languages such as C and C++. In this context, variables declared using the 'var' keyword can accommodate various data types, including integers, strings, Booleans, and more intricate data types like objects and arrays [3].
All web browsers are designed to comprehend HTML and CSS and execute these languages to display content on the computer screen. Additionally, browsers come equipped with a built-in JavaScript interpreter, allowing the execution of JavaScript code.
To incorporate JavaScript into a webpage, it is necessary to inform the browser using the `<script>` tag. The browser recognizes the end of JavaScript code when it encounters the closing `</script>` tag, reverting to its normal behavior thereafter.



```html
      <link
        rel="stylesheet"
        href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.10.2/css/all.min.css"
      />
      <link rel="icon" href="lunch.png" />
      <link rel="stylesheet" href="main.css" />
      <title>Meal Finder</title>
    </head>
    <body>
      <div class="container">
        <h1>Meal Finder</h1>
        <div class="flex">
          <form class="flex" id="submit">
            <input
              type="text"
              id="search"
              placeholder="Search for meals or keywords"
            />
            <button class="search-btn" type="submit" id="search-btn">
              <i class="fas fa-search"></i>
            </button>
          </form>
          <button class="random-btn" id="random">
            <i class="fas fa-random"></i>
          </button>
        </div>

        <div id="result-heading"></div>
        <div id="meals" class="meals"></div>
        <div id="single-meal"></div>
      </div>

      <script src="app.js"></script>
    </body>
  </html>
```

Figure 2. JS Script Tag in HTML Workflow Architecture

Programming languages at a high level, like JavaScript, are considered less intricate because they leverage abstraction, incorporating features such as a garbage collector or dynamic typing to simplify the programming process for developers. In contrast, machine code languages utilize binary expressions directly executable by the computer's central processing unit (CPU). When constructing an application in a high-level language, programmers are relieved of the responsibility to manually handle memory or processor details, eliminating concerns about concepts like pointers [3].

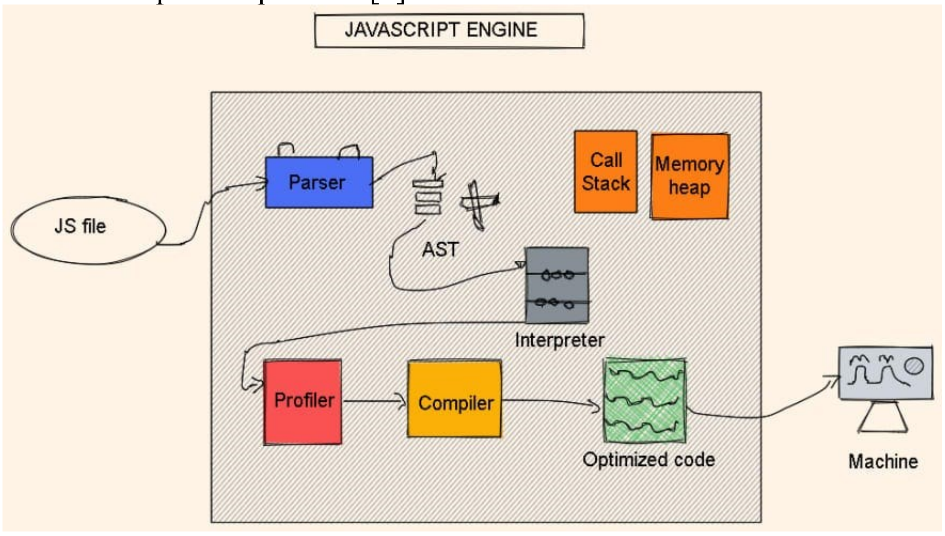


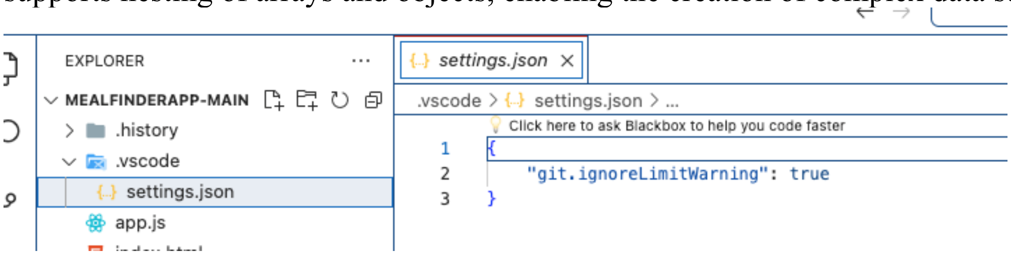
Figure 3. Google V8 JS Engine Architecture

In the JavaScript engine, code is first parsed into an Abstract Syntax Tree (AST), a structured representation. This step identifies syntax errors and serves as the basis for generating machine code. The next step involves compilation, where the AST is transformed into machine code. Modern engines use just-in-time (JIT) compilation, executing the code in the Call Stack. Clever optimization strategies involve creating an unoptimized version initially for quick execution, with further optimizations applied as needed. Google's V8, optimize and recompile code in the background during execution. This continuous process enhances speed, with parsing, compilation, and optimization occurring in specialized threads inaccessible from user code [6].

1.3 Distinct Features of the Language

1.3.1 JSON (Javascript Object Notation)

JSON, known as JavaScript Object Notation, provides a straightforward, text-based method for storing and transmitting structured data. With its simple syntax, users can store various data types, including numbers, strings, arrays, and objects, using plain text strings. It also supports nesting of arrays and objects, enabling the creation of complex data structures.

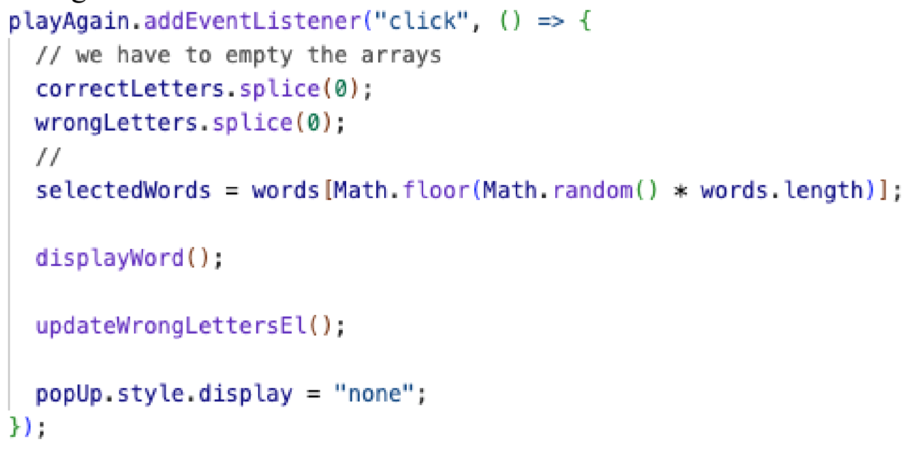

EXPLORER
MEALFINDERAPP-MAIN
.history
.vscode
settings.json
app.js
settings.json
.vscode > settings.json > ...
Click here to ask Blackbox to help you code faster

```json
{
    "git.ignoreLimitWarning": true
}
```

Figure 4.Json Format

1.3.2 JavaScript Events

JavaScript strongly emphasizes event-driven programming. When users interact with elements like buttons or links, the browser generates events that JavaScript listens for and responds to with predefined actions. This capability enables extensive manipulation of a web page's DOM, resulting in more interactive and dynamic web applications.
Event-driven programming operates on the premise that program flow is dictated by events like user actions, sensor outputs, or messages from other programs (Enyinnaya, 2019). Rather than following a linear sequence, event-driven programs react to specific occurrences, making them suitable for environments where external factors influence software behavior.

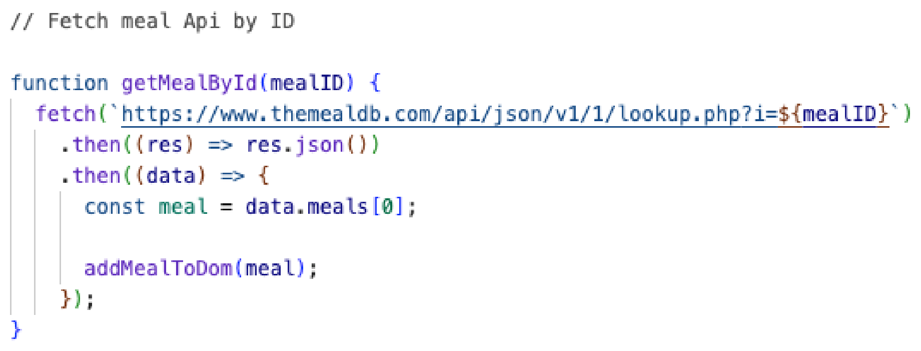


```
playAgain.addEventListener("click", () => {
  // we have to empty the arrays
  correctLetters.splice(0);
  wrongLetters.splice(0);
  //
  selectedWords = words[Math.floor(Math.random() * words.length)];

  displayWord();

  updateWrongLettersEl();

  popUp.style.display = "none";
});
```

Figure 5.Event coding in VanillaJS.

1.3.3 "non-blocking" operations (Asynchronous Programming)

Handling asynchronous operations used to be challenging, often resulting in "callback hell" - a situation where multiple nested callbacks lead to cumbersome code (Ecma International, 2015). This issue was addressed with the introduction of Promises in ECMAScript 6 in 2015. Promises represent values that may not be immediately available but will resolve in the future, offering a structured approach to dealing with asynchronous operations. Developers can chain operations using the .then() method for successful resolutions and handle errors with the .catch() method (Ayondip, 2023).



```
// Fetch meal Api by ID

function getMealById(mealID) {
  fetch(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${mealID}`)
    .then((res) => res.json())
    .then((data) => {
      const meal = data.meals[0];

      addMealToDom(meal);
    });
}
```

Figure 6.JavaScript Promise Code sample

ECMAScript 8 introduced the async and await syntax, building upon Promise principles. Async functions always return a Promise, with the await keyword pausing execution until resolution. This synchronous-like coding style improves code readability (Ecma International, 2017).

```
<script>
async function getFile() {
  let myPromise = new Promise(function(resolve) {
    let req = new XMLHttpRequest();
    req.open('GET', "mycar.html");
    req.onload = function() {
      if (req.status == 200) {
        resolve(req.response);
      } else {
        resolve("File not Found");
      }
    };
    req.send();
  });
  document.getElementById("demo").innerHTML = await myPromise;
}

getFile();
```

Figure 7. Async Await Typing

1.3.4 Javascript prototype Inheritance

In JavaScript, objects utilize prototype-based inheritance, inheriting directly from other objects, rather than through traditional classes. Each object has a linked prototype, from which it inherits properties and methods. JavaScript searches an object's prototype chain to access properties or methods, traversing linked prototypes until it finds the desired property or reaches a null prototype.

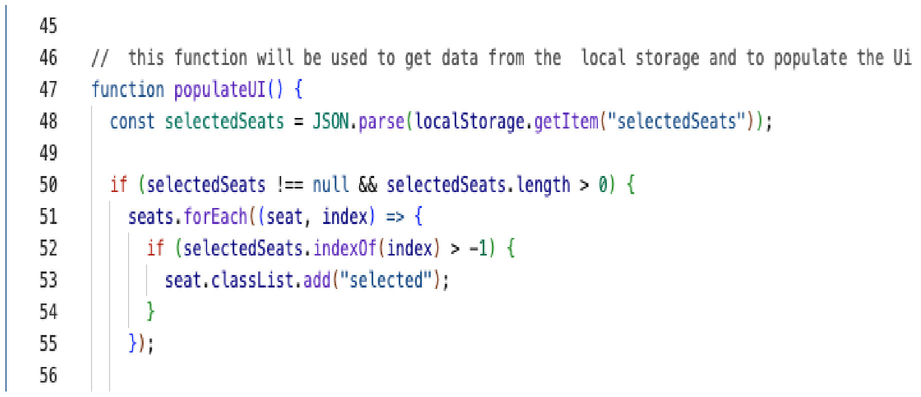
```
<script>
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

Person.prototype.name = function() {
  return this.firstName + " " + this.lastName
};

const myFather = new Person("John", "Doe", 50, "blue");
document.getElementById("demo").innerHTML =
"My father is " + myFather.name();
</script>

</body>
</html>
```

Figure 8.Prototype inheritance

1.3.5 Functional Programming

Functional programming systems, characterized by less coupling and fewer side effects due to the absence of shared state between components, utilize higher-order functions, which accept functions as inputs. This stands in contrast to object-oriented programming (OOP), where only data or objects can be passed through.

Functional programming, falling under the declarative paradigm, has gained traction alongside JavaScript's widespread adoption, particularly with the rise of UI libraries like React and Angular. It revolves around mathematical functions, known as pure functions, which produce consistent output based solely on input.



```
// this function will be used to get data from the  local storage and to populate the Ui
function populateUI() {
  const selectedSeats = JSON.parse(localStorage.getItem("selectedSeats"));

  if (selectedSeats !== null && selectedSeats.length > 0) {
    seats.forEach((seat, index) => {
      if (selectedSeats.indexOf(index) > -1) {
        seat.classList.add("selected");
      }
    });

```

Figure 9. A JavaScript function to get Data from local storage

1.4 Registry Management System

In today's software landscape, where open source, proprietary, and third-party components are integral to development, repository management plays a crucial role. Organizations rely on repository management systems to efficiently source, store, share, and deploy these components. The sheer volume and pace at which these components are utilized in software development create what can be termed as a 'software supply chain.' In this context, a repository manager serves as the official warehouse for these components.

Furthermore, repository managers offer invaluable insight into component quality, enabling development teams to make informed decisions upfront. By doing so, teams can mitigate the risk of accumulating technical debt and reduce the need for unplanned or unscheduled work downstream.
It's worth noting that a significant portion—80-90%—of a typical application comprises various component formats and types. These include libraries, frameworks, modules, packages, assemblies, and other parts. As development paradigms shift towards microservices and containers, the usage of components becomes even more pronounced.

1.4.1 Registry as a Component

A component is essentially a piece of software or resource that your application relies on. These resources could be anything from a library or framework that helps your application function to an image file that adds visual elements. Components are utilized at different stages of your application's lifecycle, such as when the application is running, during testing, or as part of the process of building and deploying the application. They can take various forms, ranging from small code snippets to entire applications or even larger entities like an entire operating system when used in certain environments like container-based systems such as Docker. In essence, components are the building blocks that make up your software application.
Components typically consist of a diverse array of files, ranging from Java bytecode in class files to C object files, as well as binary files like images, PDFs, and audio files, among others. These files are packaged into archives using various formats such as Java JAR, WAR, EAR, plain ZIP, tar.gz, as well as other package formats like NuGet packages, RubyGems, npm packages, Docker images, and more.
Moreover, components can themselves be composed of multiple nested components. For instance, a Java web application packaged as a WAR component may contain numerous JAR components and JavaScript libraries. While these nested components are standalone entities in their own right, they are included as part of the larger WAR component. Essentially,

components can be thought of as self-contained units that encapsulate a collection of related files and resources necessary for a specific functionality within a software application.

A diverse range of components is developed by both the open-source community and proprietary vendors, forming a vast and rapidly expanding ecosystem. As an illustration, consider the Central Repository of Maven/Java components, which boasts over 120,000 unique components and more than 1 million total component versions. This indicates the extensive scale of the ecosystem and its continuous growth over time. In essence, it highlights the abundance of resources available to developers, catering to a wide array of needs and preferences within the software development landscape [8].

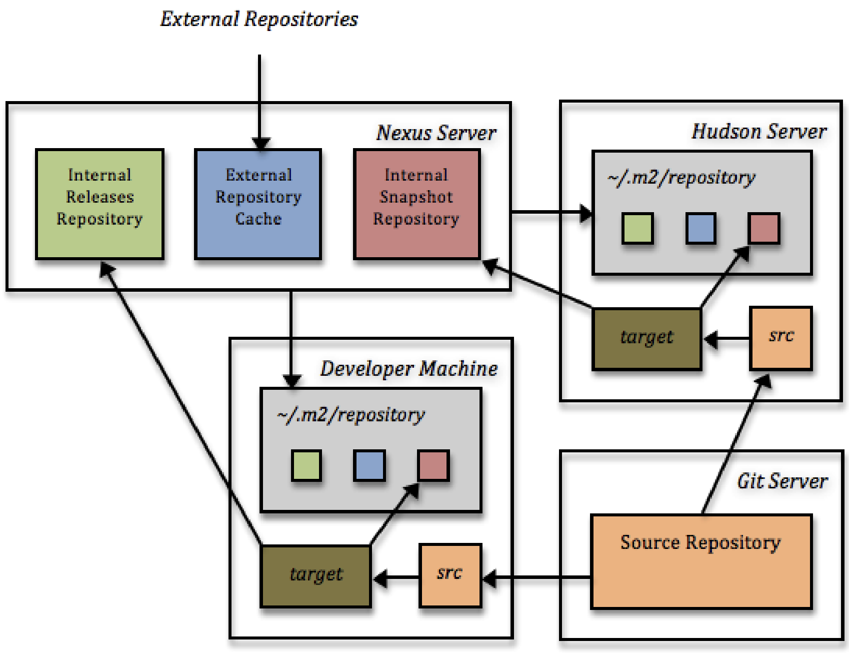


Figure 10. Maven Registry Architecture

1.4.2 Component Architecture in Public repositories

In order to make components easily accessible to developers, the open-source community organizes collections of these components into what are known as "public repositories." These repositories are typically hosted on the Internet and are freely accessible. Different platforms may use terms like "registry" to refer to the same concept. Some well-known repositories include The Central Repository, NuGet Gallery, RubyGems.org, npmjs.org, and Docker Hub. Components stored in these repositories can be accessed by various tools, such as package managers, build tools, IDEs (Integrated Development Environments), provisioning tools, and custom integrations using scripting languages.
Public repositories offer a more efficient solution compared to simple directory structures or download websites. Instead of manually searching for components and their dependencies and then storing them in their own infrastructure, users can simply declare the components they need, and tools will take care of the rest, handling tasks such as locating the components and managing their dependencies automatically.

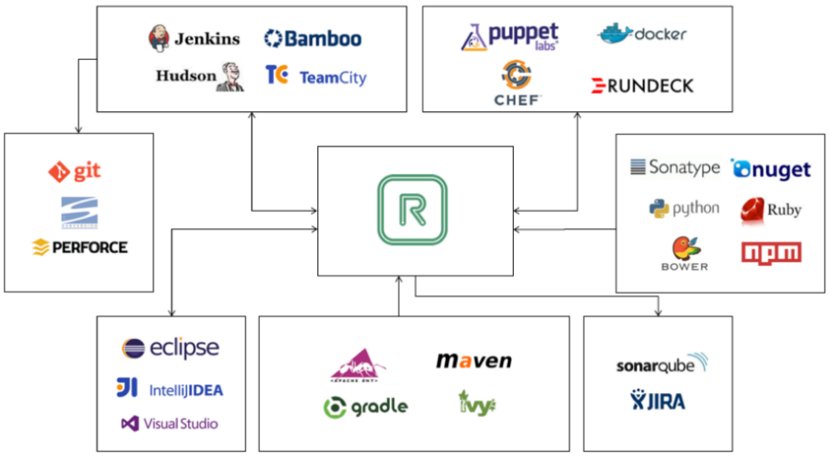


Figure 11. Repository managers in a DevOps Setup

complexity has escalated as components are now ubiquitous across various development stacks, leading to a blend of technologies within most applications. For instance, a server-side application might employ Java technologies to implement a REST API, accessing components through Maven. However, the web application utilizing these APIs to create a user interface may opt for a purely JavaScript-based approach, sourcing its components via npm [29].

1.5 Advance Reasons to consider Repository system when building a software Application

We will take a quick look into various importance of the repository system and how efficient it aids software development using the JavaScript programming language [9]

- Component intelligence

Certain professional editions of Repository Managers offer health checks, which offer immediate insight into potential security, licensing, and quality risks associated with components. This empowers development teams to swiftly and proactively address any issues that arise.

- Authentication

Given that the Repository Manager houses project-related binaries, it's prudent to apply the same permissions as those enforced for the projects themselves, including source code access permissions, to safeguard the resulting binaries. In certain scenarios, access to the binaries may be granted independently of access to the source code, and this can be managed at the repository level.

- Efficient Use in Distributed system

When teams accessing repositories are geographically dispersed, it becomes crucial to ensure access to all components, whether internal or third-party. Given that a Repository Manager essentially functions as a caching proxy, it's essential to have one deployed in each physical location where there's a significant developer presence. Without this setup, developers may encounter slow and unreliable build times, as they fetch components from the internet or across wide area networks (WANs).

- Efficient High availability

Relying on a Repository Manager to store all development dependencies makes it a vital component of your infrastructure. Any downtime experienced by the repository disrupts development activities, leading to potentially severe consequences. In a Continuous Integration/Continuous Deployment (CI/CD) environment, the unavailability of a Repository Manager prevents builds from executing and deployments to production, posing significant risks to the business or organization.

- Building High Level Projects

When constructing an application with Maven or similar build tools, it retrieves components from a designated location, configured through Maven settings, and compiles them. This process can yield an application or another binary or component. Jenkins facilitates this automation by enabling the integration of a Maven build step for both freestyle and multi-configuration projects.

Our Movie app in chapter 4 will based solely on **Client-Server System architecture** (CSA). The CSA will be solely explained in chapter 3.

Before delving deeper into the expansive JavaScript ecosystem and exploring the myriad factors influencing the selection of an appropriate JavaScript tech stack for web application development, our primary focus in this project, let's explore some of the relevant literature that precedes this thesis topic.

2 RELATED LITERATURE REVIEW

Numerous studies and initiatives have delved into the realm of Big Code and code repositories, aiming to enhance development practices. Researchers and practitioners have explored ways to leverage vast code repositories to improve the efficiency and effectiveness of development tools. This work involves addressing the challenges posed by the sheer scale and complexity of Big Code, seeking innovative solutions to extract meaningful insights, facilitate code reuse, and enhance overall development processes. These efforts underscore a collective pursuit to harness the potential of extensive code repositories for advancing software development methodologies.
A subset of information sourced from the GitHub and StackOverflow APIs is included in publicly accessible datasets within Google BigQuery. The GitHub segment comprises repository metadata, encompassing details such as programming languages and licenses, alongside the actual contents of the repositories. These BigQuery datasets are open to any user with a Google account, allowing them to execute SQL queries and run arbitrary JavaScript code on the data. An illustrative instance of data analysis on these datasets can be found in [18], where the authors parsed a billion files from 400,000 repositories across 14 programming languages, focusing on the comparison of tabs and spaces for indentation. For JavaScript, the results indicate that 18% of files use tabs, while the remaining majority use spaces. Additional analyses of a similar nature are discussed in [10].
Decan et al. (2016) try to identify the differences in software package ecosystems (CRAN, PyPI, NPM), though package dependency graphs. In light of their outcomes, it can be asserted that NPM stands out as an ecosystem that fosters an ethos of extreme reusability and micropackaging culture. This is achieved through adherence to the single-responsibility principle at the package level. In (Kikas et al., 2017) The authors demonstrated that the JavaScript ecosystem is indeed experiencing the fastest growth and exhibits significant interconnectivity among its packages. Finally, Bogart et al. (2016) Through interviews, it was observed that developers, to ensure the stability of their packages and due to a lack of constant awareness regarding the status of their dependencies, tend to minimize their reliance on them. Instead, they opt for the adoption of packages that are considered "best-practice." [11].
UnuglifyJS, an implementation of a JavaScript deminifier, is introduced to extract properties and features from code. This client serves as a valuable starting point for researchers and developers interested in utilizing the Nice2Predict framework. The open-source implementation is accessible on GitHub at: **https://github.com/eth-srl/UnuglifyJS**.
The responsibilities of this client encompass three key aspects: i) defining known and unknown properties, ii) specifying features, and iii) acquiring training data. It commences by articulating known and unknown properties. Known properties encompass constants, object properties, methods, and global variables—elements of a program that cannot be (soundly) renamed, such as DOM APIs. Conversely, unknown properties involve all local variables.

2.1 Meteor

Meteor is a JavaScript-based platform designed for developing web applications entirely in JavaScript. It offers a distinctive approach to web application development, enabling developers to write JavaScript functions that execute on the server, the client, or both. Geared towards single-page web applications, Meteor integrates core features such as collaboration and data synchronization. A standout feature of Meteor is its capability to provide database access from anywhere. This means developers can execute the same database queries on both the client and the server.

2.2 Boa Language Architecture

Boa emerges as a specialized language and infrastructure tailored for simplifying the extraction of insights from software repositories. Distinguished by its domain specificity, Boa's infrastructure harnesses the power of distributed computing techniques to adeptly execute queries across a vast landscape of software projects, ensuring remarkable efficiency. Positioned at the intersection of language design and infrastructure, Boa serves a pivotal role in facilitating the testing of hypotheses related to mining software repositories (MSR).

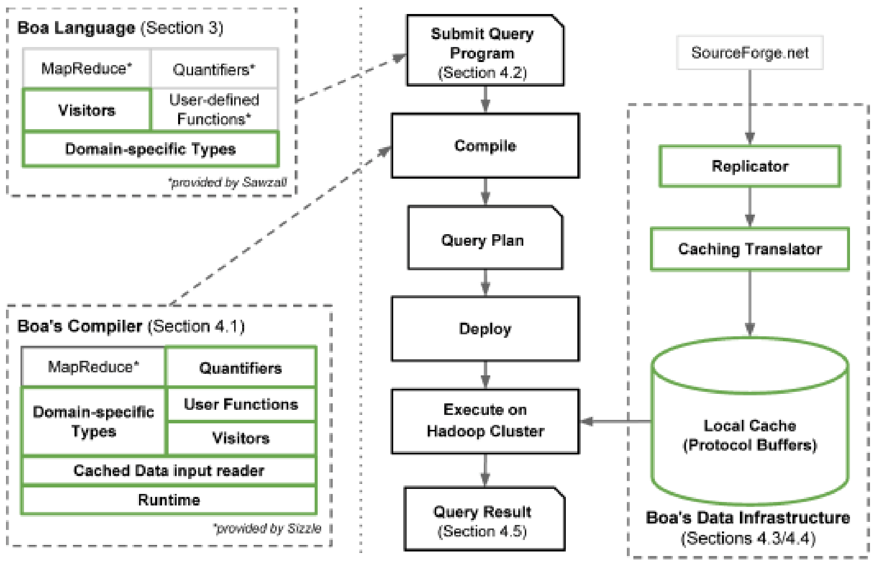
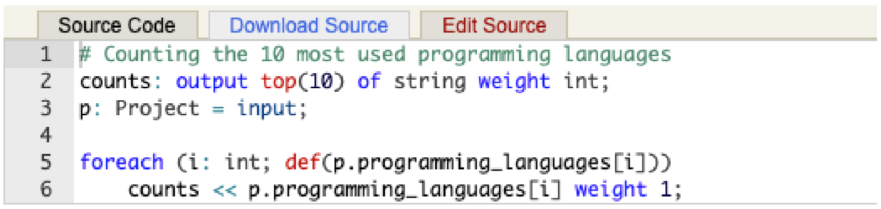


Figure 12.Boa build Infrastructure

The impetus behind the creation of Boa arose as the authors endeavored to address a pervasive challenge on an extensive scale, delving into intricate details with complete historical information. This comprehensive approach reflects Boa's ambition to provide a versatile solution for mining software repositories, offering a fine-grained exploration of data and a nuanced understanding of software project evolution [12].

Code Snippet of Boa Architecture:



```
# Counting the 10 most used programming languages
counts: output top(10) of string weight int;
p: Project = input;

foreach (i: int; def(p.programming_languages[i]))
    counts << p.programming_languages[i] weight 1;
```

Output of the result:

```
counts = JavaScript, 1473096.0
counts = Ruby, 889738.0
counts = Shell, 700831.0
counts = Python, 620213.0
counts = Java, 554864.0
counts = PHP, 489082.0
counts = C, 419044.0
counts = CSS, 354715.0
counts = C++, 333877.0
counts = Perl, 274174.0
```

result can be accessed here: https://boa.cs.iastate.edu/boa/?q=boa/job/107801

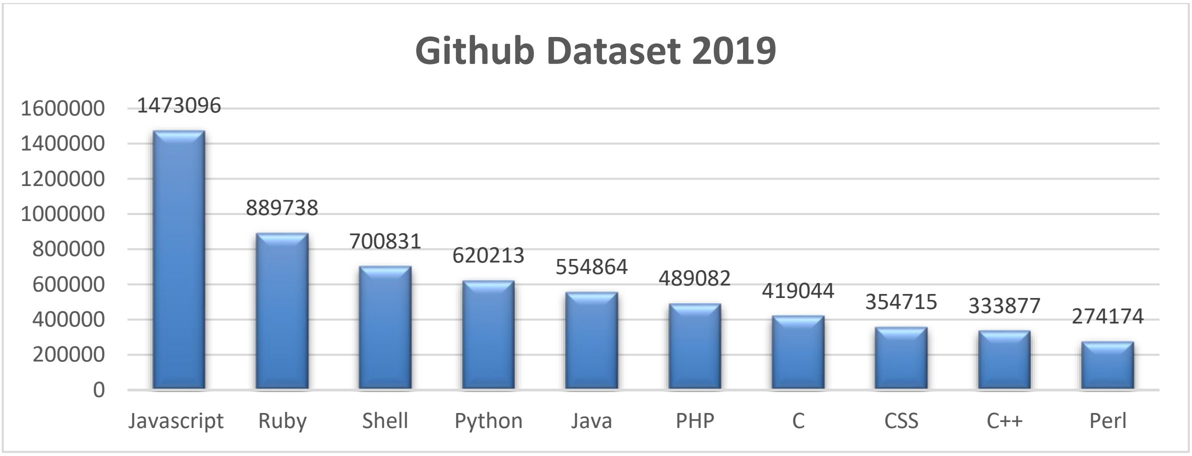


Figure 13. project counts obtained from Boa's September 2019 GitHub dataset

In the ever-evolving realm of software development and infrastructure, the choice of programming languages plays a pivotal role in shaping the technological landscape. As of 2019, data sourced from GitHub through the boa architecture reveals a clear leader among the top 10 most used languages — JavaScript. With a staggering count of 1,473,096 repositories, JavaScript not only dominates the charts but also stands as a testament to its unparalleled influence in the developer community.

2.3 GitHub Activity Data

GitHub Activity Data is a project that provides a public dataset on Google Big Query containing a record of public GitHub events. It's a collaborative effort between GitHub and Google, allowing developers and data enthusiasts to analyze and gain insights into the vast amount of data generated on GitHub.

A straightforward query offers a glimpse into the dataset, providing insights. For instance, a query counting the number of repositories committed so far currently yields a result of 265,419,190.

Code snippet for total repo commit 2023:

```
SELECT
 COUNT(*) AS total_commits FROM `bigquery-public-data.github_repos.commits`
LIMIT 1000
```

2.4 Model Core J2EE patterns

The Java Sun team introduced a 5-tier architecture [13] to embody the Core J2EE Patterns Architecture [14], as depicted in Table 8. Java further facilitates the implementation of the Model-View-Controller architecture through the utilization of the Observer Interface and Observable classes, collectively realizing the observer pattern. The Observable class serves as a representation of an observable object, essentially the "data" within the model-view paradigm. It can be extended to signify an object that the application intends to be observed. An observable object has the capacity to enlist one or more observers. In this context, an observer is any object that adheres to the Observer.

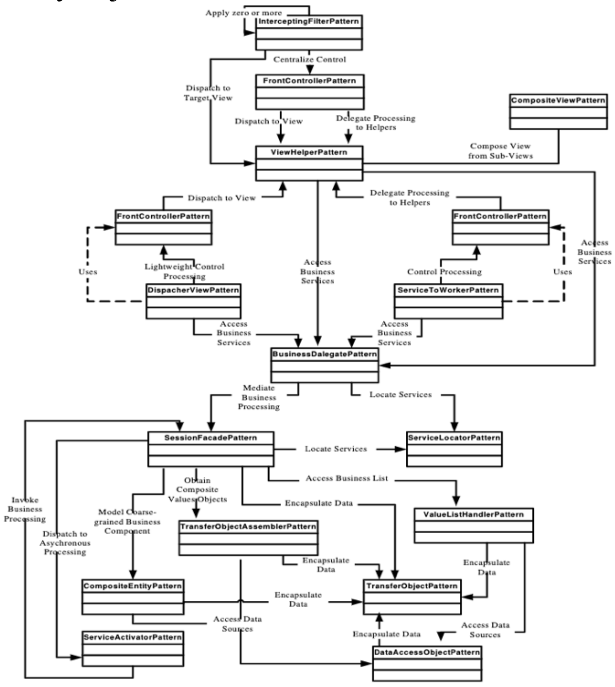


Figure 14. core J2EE Patterns structure

The proposed Java team 5-tier architecture to model the J2EE is explained below;

- Browser

This segment is frequently not indicative of the overall architecture, but it is plausible that it includes application components commonly referred to as "First-Level Tests." First-level tests primarily involve verifying the contents of input forms, ensuring the accurate input of all mandatory fields, for example. However, it is imperative that these tests are categorized within the presentation layer. This designation is crucial because end-users may opt to disable JavaScript functionality in their browsers. Additionally, this layer serves the purpose of rendering dynamic pages, such as those in DHTML format.

- Presentation

This tier is responsible for managing navigation logic and frequently employs JSP/Servlets technologies.

- Logical Subject

Implemented through Java Beans or EJB, this layer encompasses all the processes within an application.

- Middleware

This segment of the architecture addresses interactions with other patterns at the same level or composite patterns across different levels.

- Persistence

This segment of the architecture addresses interactions with other patterns at the same level or composite patterns across different levels.

Expanding upon the principles of the Model Core J2EE patterns, we will take a look at the Model-View-Controller (MVC) pattern also.

2.5 Model-View-Controller (MVC)

The MVC pattern, a widely adopted structure for web applications with substantial processing demands. This choice enhances the coding and maintenance aspects, leveraging MVC as a 3-tier architecture to delineate the core elements of web application architecture. In this context, MVC serves to describe the fundamental components of web applications, with its 3-tier structure commonly utilized by designers to manage multiple views of identical data. At the design level, the MVC pattern emphasizes a clear segregation of three types of objects.
Expanding upon the principles of the Model-View-Controller (MVC) pattern [15]. This is how the 3 core models affect the development of web applications;

- Model

A model component serves as a repository for both data and logic. For example, a Controller object might fetch customer information from a database. Data interchange occurs among controller components or various business logic elements. It handles data manipulation and either returns it to the database or employs it for rendering the same information.
Moreover, it reacts to requests from views and receives directives from the controller, enabling it to self-update. It stands as the lowest level of the pattern, tasked with overseeing data maintenance.

- View

The View is responsible for depicting how data appears within the application. Views are generated using data obtained from the model. When the user requests information from the model, the output presentation is presented to them. In addition to visualizing data through charts, diagrams, and tables, the view can also showcase data from various sources. All user interface elements, including text boxes, drop-down menus, etc., are visible in any customer view.

- Controller

Controllers are the elements within an application responsible for managing user interactions. The controller interprets user input, leading to changes in both the model and view based on the received information. Through interaction with a controller's associated view, users have the ability to alter the view's presentation (such as scrolling through a document) and update the state of the connected model (such as saving a document).

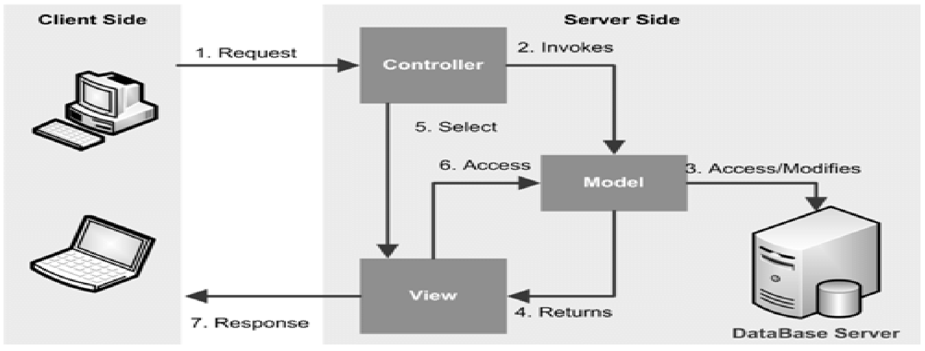


Figure 15. Illustration of the System Applying the MVC Architecture Pattern

3 PART A - CLIENT-SERVER ARCHITECTURE (JS ECOSYSTEM)

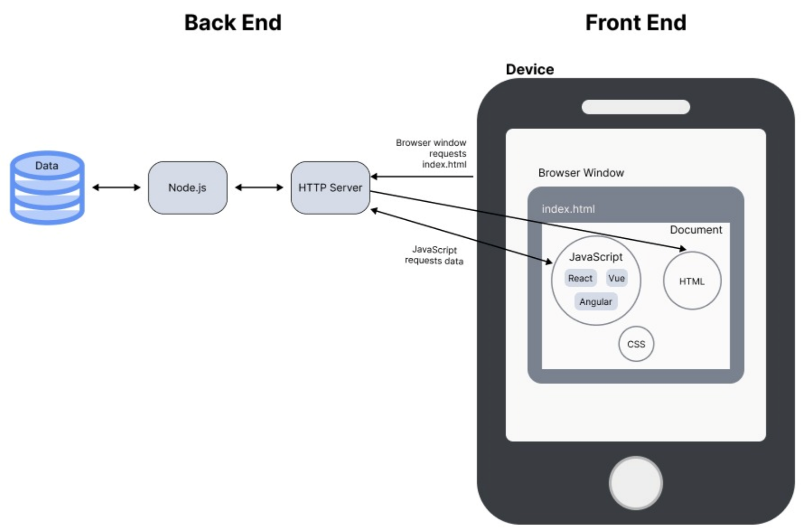


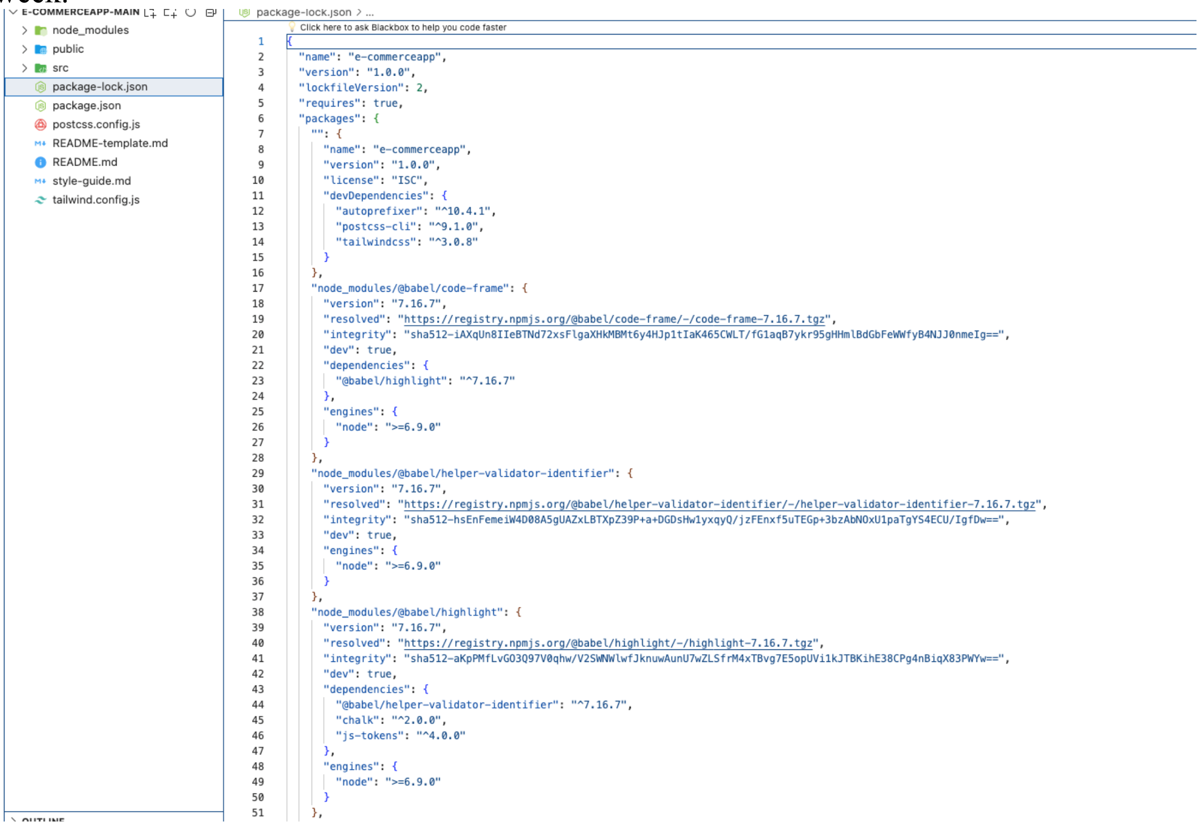
Figure 16.Client-Server Illustration for a JS Web Application

Client-server architecture is a model used in building web applications where client devices, such as web browsers, interact with a server to access and manipulate data. In this architecture, the client is responsible for displaying the user interface and handling user interactions, while the server stores and manages data, processes requests from clients, and sends back responses. JavaScript is commonly used on both the client and server sides to enhance interactivity and functionality as seen in the figure, there are 3 popular JavaScript frameworks used in building a scalable web application. This architecture allows for scalable, distributed systems where clients and servers communicate over a network, enabling dynamic and responsive web applications. We will explain further the various build tools used frequently by professionals in building an industry standard application.

In this chapter, our goal is to explore popular build tools for creating scalable JavaScript applications. Following that, we will conduct a survey to gather opinions from software developers on their choice of primary frameworks used for building web applications and why? what factors do they consider? efficiency? speed? performance? maintenance? Or is it solely due to Client Business Decision? Analyzing the survey data will guide our decision in selecting the appropriate framework for developing our own web application in Chapter 4.

3.1 NODE PACKAGE MANAGER (NPM)

User-contributed Open Source Software (OSS) ecosystems have gained prominence in the software engineering realm, attracting attention from both practitioners and researchers. Notably, ecosystems comprising 'collections of third-party software,' such as the node package manager (npm) in the JavaScript package ecosystem [10], play a pivotal role in the development of extensive server-side NodeJS and client-side JavaScript applications. As of 2016, a study reported that the npm ecosystem for the NodeJS platform accommodates over 230 thousand packages [10], witnessing 'hundreds of millions of package installations every week.'



```
E-COMMERCEAPP-MAIN
> node_modules
> public
> src
  package-lock.json
  package.json
  postcss.config.js
  README-template.md
  README.md
  style-guide.md
  tailwind.config.js
```

```
{
  "name": "e-commerceapp",
  "version": "1.0.0",
  "lockfileVersion": 2,
  "requires": true,
  "packages": {
    "": {
      "name": "e-commerceapp",
      "version": "1.0.0",
      "license": "ISC",
      "devDependencies": {
        "autoprefixer": "^10.4.1",
        "postcss-cli": "^9.1.0",
        "tailwindcss": "^3.0.8"
      }
    },
    "node_modules/@babel/code-frame": {
      "version": "7.16.7",
      "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.16.7.tgz",
      "integrity": "sha512-iAXqUn8IIeBTNd72xsFlgaXHkMBMt6y4HJp1tIaK465CWLT/fG1aqB7ykr95gHHmlBdGbFeWWfyB4NJJ0nmeIg==",
      "dev": true,
      "dependencies": {
        "@babel/highlight": "^7.16.7"
      },
      "engines": {
        "node": ">=6.9.0"
      }
    },
    "node_modules/@babel/helper-validator-identifier": {
      "version": "7.16.7",
      "resolved": "https://registry.npmjs.org/@babel/helper-validator-identifier/-/helper-validator-identifier-7.16.7.tgz",
      "integrity": "sha512-hsEnFemeiW4D08A5gUAZxLBTXpZ39P+a+DGDsHw1yxqyQ/jzFEnxf5uTEGp+3bzAbNOxU1paTgYS4ECU/IgfDw==",
      "dev": true,
      "engines": {
        "node": ">=6.9.0"
      }
    },
    "node_modules/@babel/highlight": {
      "version": "7.16.7",
      "resolved": "https://registry.npmjs.org/@babel/highlight/-/highlight-7.16.7.tgz",
      "integrity": "sha512-aKpPMfLvGO3Q97V0qhw/V2SWNWlwfJknuwAunU7wZLSfrM4xTBvg7E5opUVi1kJTBKihE38CPg4nBiqX83PWYw==",
      "dev": true,
      "dependencies": {
        "@babel/helper-validator-identifier": "^7.16.7",
        "chalk": "^2.0.0",
        "js-tokens": "^4.0.0"
      },
      "engines": {
        "node": ">=6.9.0"
      }
    },
```

Figure 17.a package.json file

According to Jansen et al. , ecosystems emerge when 'large computation tasks are split up and shared by a collection of small, nearly independent, specialized units depending on each other.' Building upon this concept, we posit that third-party software ecosystems like npm foster the philosophy of specialized software within these self-organizing ecosystems.
A micro-package is the outcome when a package adopts a 'minimalist' approach in size and focuses on a single task [11]. For example, the negative-zero1 package addresses the straightforward task of determining whether an input number has a negative-zero value. Micro-packages operate as individual units, establishing 'transitive' dependencies between dependent packages (i.e., dependency chains) across the ecosystem.

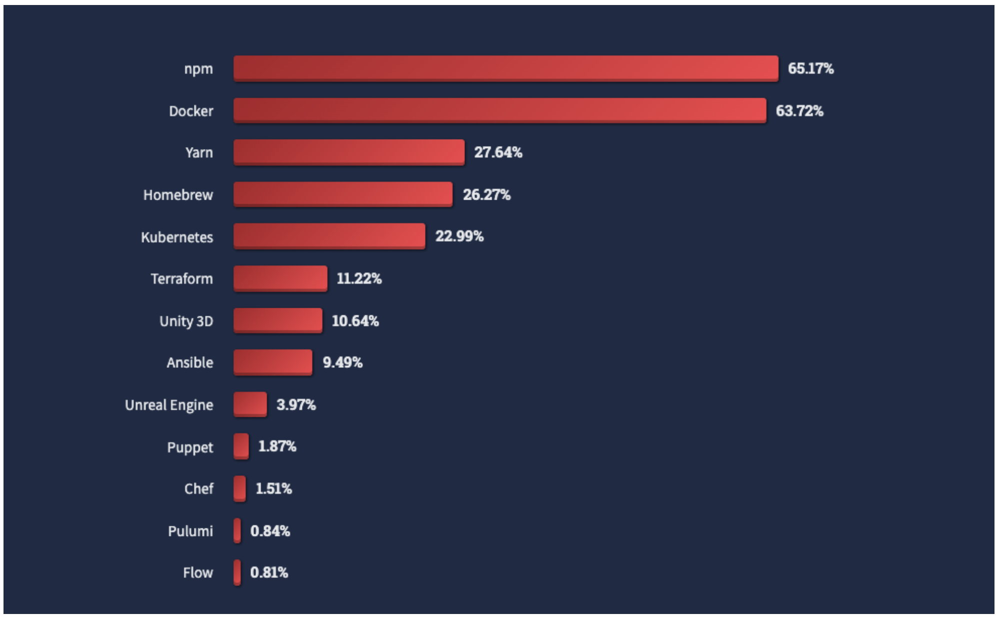
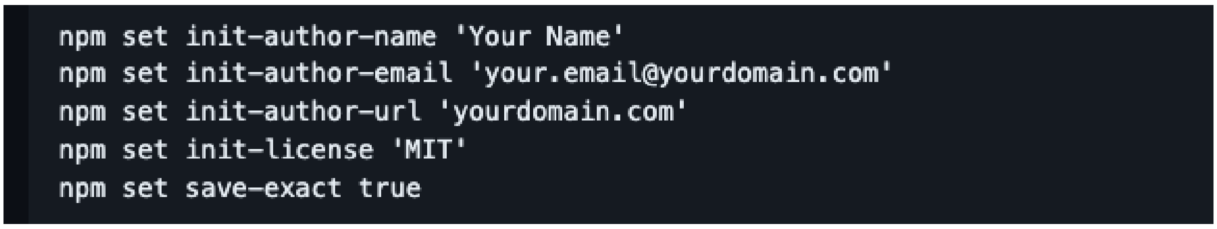


Figure 18. Stack overflow chart 2022 (NPM)

NPM (Node Package Manager) has evolved significantly since its initial release on January 13th, 2010. Initially included as part of the Node.js framework, it has become the leading package manager for JavaScript, commanding over 65.17% of the userbase as of 2022 (Stack overflow User Survey 2022). This dominance is largely attributed to its integration with Node.js.

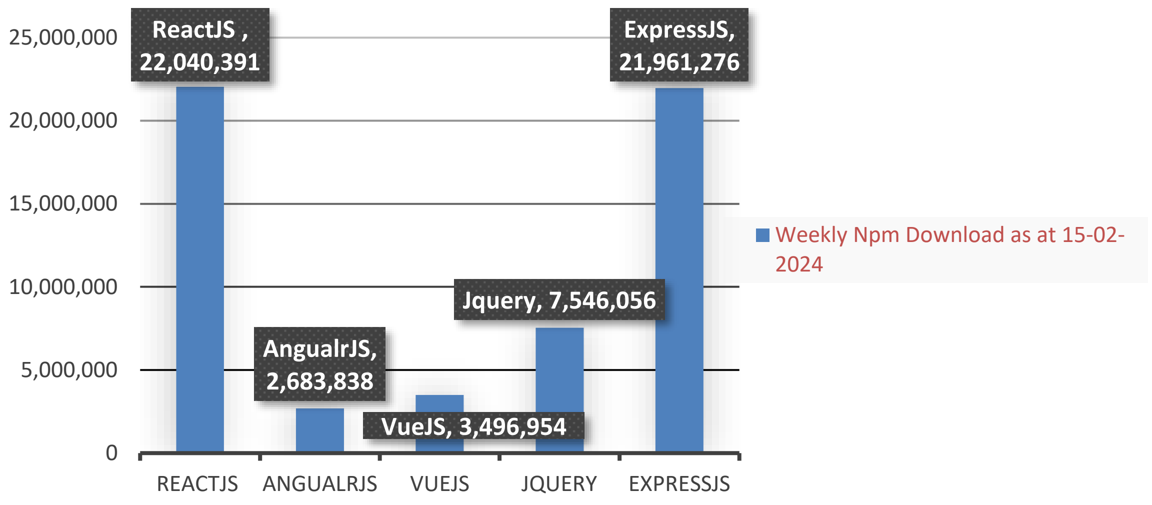
Presently, NPM is overseen by NPM Inc., a company established in 2014 dedicated to maintaining the NPM software and package repository. They also offer paid services and support tailored for enterprise usage (NPM Inc. 2019).



```
npm set init-author-name 'Your Name'
npm set init-author-email 'your.email@yourdomain.com'
npm set init-author-url 'yourdomain.com'
npm set init-license 'MIT'
npm set save-exact true
```

Configuring npm and creating a package.json [16]

The development of NPM remains highly active, with the latest stable version, 10.4.0, released on January 24th, 2024 (NPM GitHub 2024).
As of 2022, the NPM registry stands as the largest online package registry. It hosts packages submitted by developers and boasts more a significant importance to its nearest competitor, the Docker (Linux.com 2017).



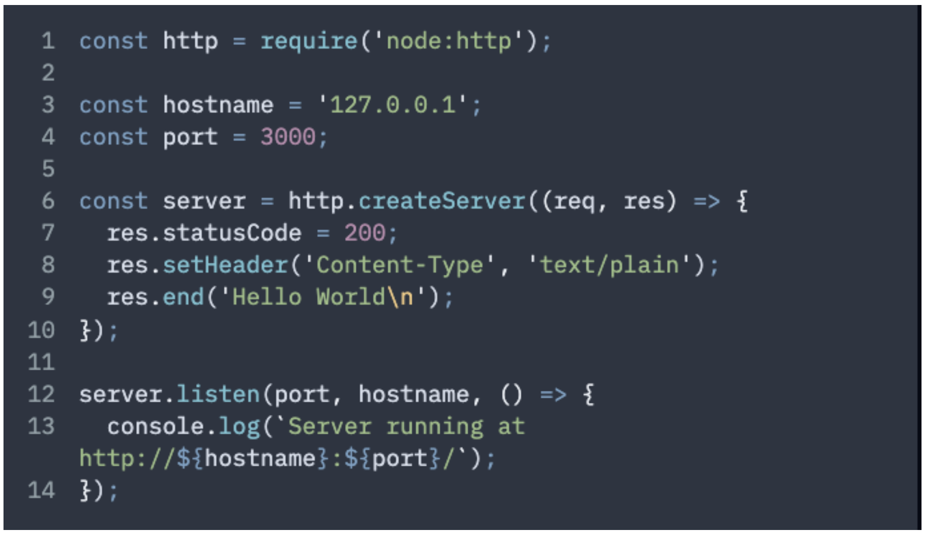
Source -> https://www.npmjs.com/package/express

3.2 Understanding the JavaScript Client – Server-Side System

Developing a comprehensive JavaScript software product using a single technology is nearly impractical, leading to the rise of frameworks and libraries. Typically, this entails utilizing tech stacks, which are combinations of JavaScript programming language, it's frameworks, libraries, database system, templates, and other tools**.**

3.2.1 NodeJS

Node.js® stands as an open-source JavaScript runtime environment constructed upon Chrome's V8 engine. It operates in an event-driven manner with non-blocking I/O, rendering it lightweight, efficient, and exceptionally rapid for crafting web applications.
Widely adopted across various industries such as IT and Finance sector, Node.js has emerged as the preferred choice. Its inherent simplicity is difficult to overlook. Nevertheless, like every technology, Node.js comes with its own set of pros and cons. Therefore, let us delve into the advantages and disadvantages of Node.js, empowering you to align your requirements and make a well-informed decision.



```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at
http://${hostname}:${port}/`);
});
```

Figure 19. An Example of Node.js Application

"*To run this snippet, save it as a server.js file and run node server.js in your terminal. This code first includes the Node.js http module.*

Node.js has a fantastic standard library, including first-class support for networking. The createServer() method of http creates a new HTTP server and returns it. The server is set to listen on the specified port and host name. When the server is ready, the callback function is called, in this case informing us that the server is running. Whenever a new request is received, the request event is called, providing two objects: a request (an http.IncomingMessage object) and a response (an http.ServerResponse object)" [17].

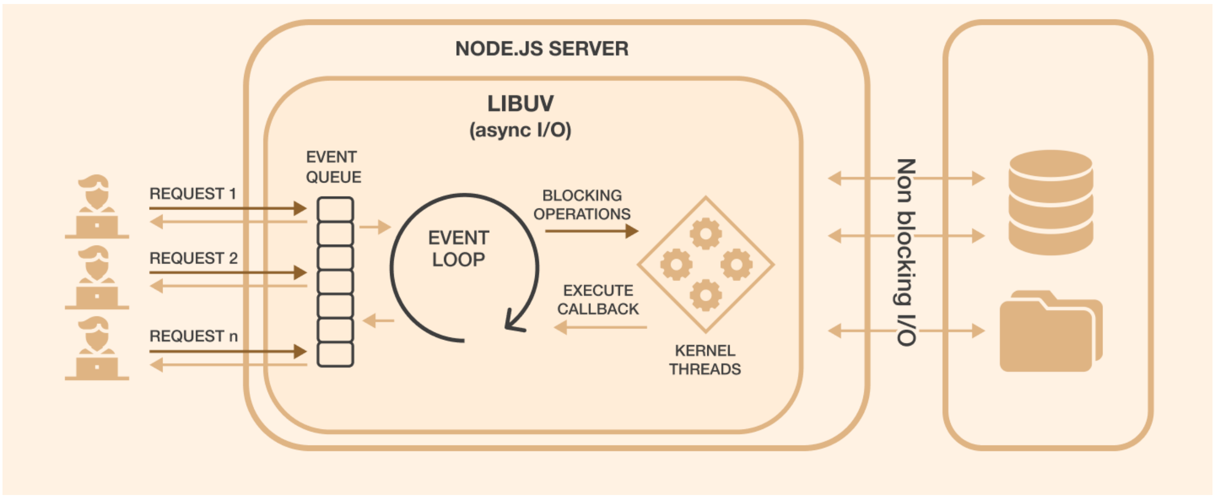


Figure 20. NodeJS Server Architecture

Node.js utilizes non-blocking I/O and asynchronous request handling to manage operations efficiently. This means that instead of waiting for a function to complete before moving on, Node.js processes requests concurrently, allowing it to handle multiple tasks simultaneously. This is made possible through the use of call-backs and promises, which manage asynchronous operations by executing functions when certain tasks are completed. Node.js application remains responsive and can handle incoming requests effectively.

3.2.1.1 Logical Reasons to use NodeJS

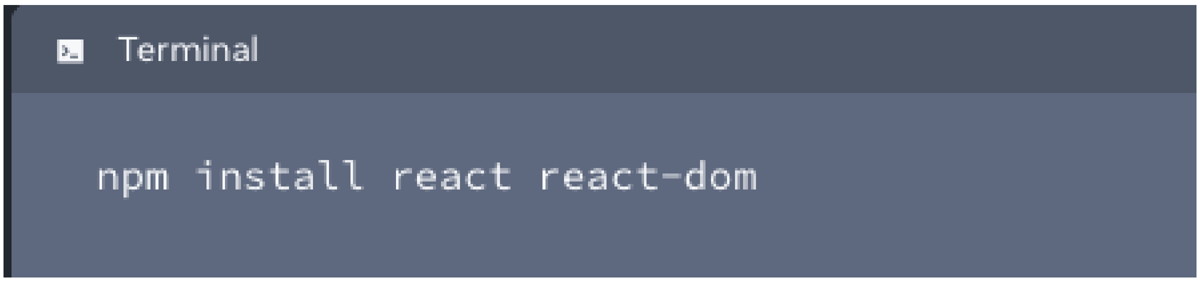
- Robust Technology
- Fast Processing for web tasks – asynchronous Thread (Non-Blocking I/O Model)
- Scalable technologies for microservice applications
- Seamless JSON support

3.2.1.2 Issues with Building an Application with NodeJS

- Call-back Hell issue
- The asynchronous programming model of Node.js poses challenges in code maintenance.
- Performance is compromised when tasked with heavy computing operations.
- Node.js opens up a lot of code changes due to Unstable application programming interface (API)

3.2.2 REACT

ReactJS, commonly referred to as React, is a front-end JavaScript library pioneered by Facebook, specifically by (Jordan Walke) [18]. It is designed for constructing highly responsive user interfaces and finds extensive application in the web applications.



```
npm install react react-dom
```

Figure 21. installing React on an existing project using NPM

React operates on a declarative paradigm, employing components to structure user interfaces. It maintains a "virtual" representation of the UI in memory, which synchronizes with the "real" DOM. Integrating React into an already-existing web page is feasible, allowing developers the flexibility to incorporate as much or as little as desired. Putting JSX markup close to related rendering logic makes React components easy to create, maintain, and delete [19].
Writing user interface with React requires a bit of a shift in how you think about web applications.

// JSX to express UI components.

```
Let dropdown =
<dropdown>
        A dropdown list
        <Menu>
                <MenuItem> Do This </MenuItem>
                <MenuItem> Do This </MenuItem>
                <MenuItem> Do This </MenuItem>
</Menu>
</Dropdown>
 render(dropdown);
```

ReactDom
React DOM is the essential engine behind the efficiency of React-based user interfaces in handling the multitude of screen changes demanded by modern web applications. It achieves this feat by leveraging a Virtual DOM, which optimizes the process of updating the user interface.

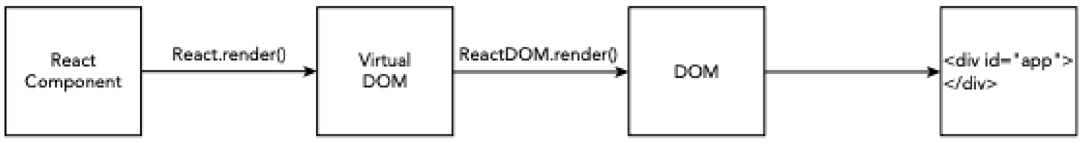


Figure 22. React DOM Structure

Between 2022 and 2024, the React library has seen significant popularity and usage, with over 22 million downloads recorded via npm. This trend underscores React's continued relevance and widespread adoption within the development community for building web applications [20]. The illustration below proves our point.

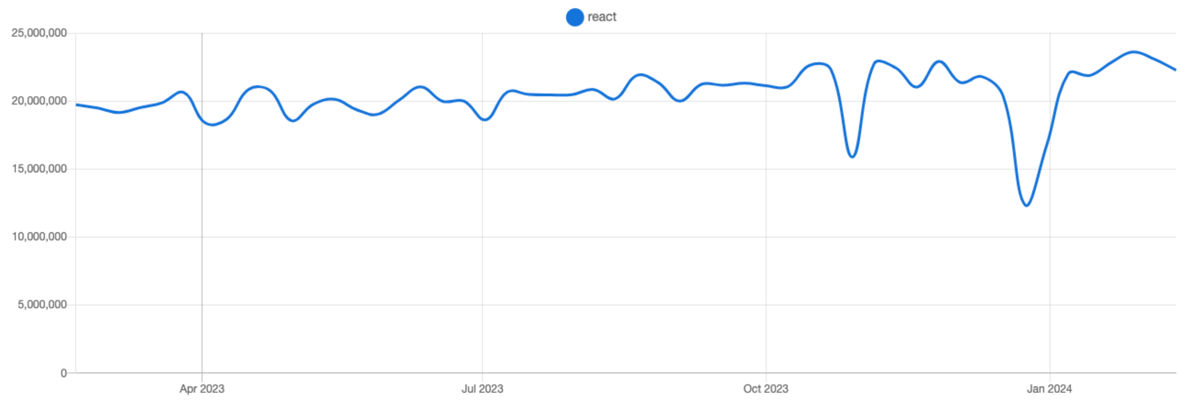


Figure 23. React Library download (NPM trend stat 2024)

3.2.2.1 Logical Reasons to use React framework

- React Components – Build once and use in multiple projects.

React facilitates rapid project development through reusable components, which are isolated pieces of code utilized within an app or across various projects.

- One directional data flow

In React, data flows unidirectionally from parent to child components. The parent component's data, known as state, dictates the screen content. The state portion passed to a child is called a prop, which is read-only.

- React VirtualDOM - Reduces reloads in applications

Introduced for React-based apps, the virtual DOM enhances speed and interactivity.

- React Hook

React Hooks simplify complex logic which Improves code readability, testability, and logic reusability, while also reducing app bundle size.

- A large community of React Devs – awesome community support

3.2.2.2 Issues with Building an Application with ReactJS

- Excessive use of additional technologies

While React itself is lightweight and user-friendly, incorporating numerous additional technologies is often necessary before starting work with it. A general guideline is to refrain from adding new technologies on top of React unless absolutely necessary.

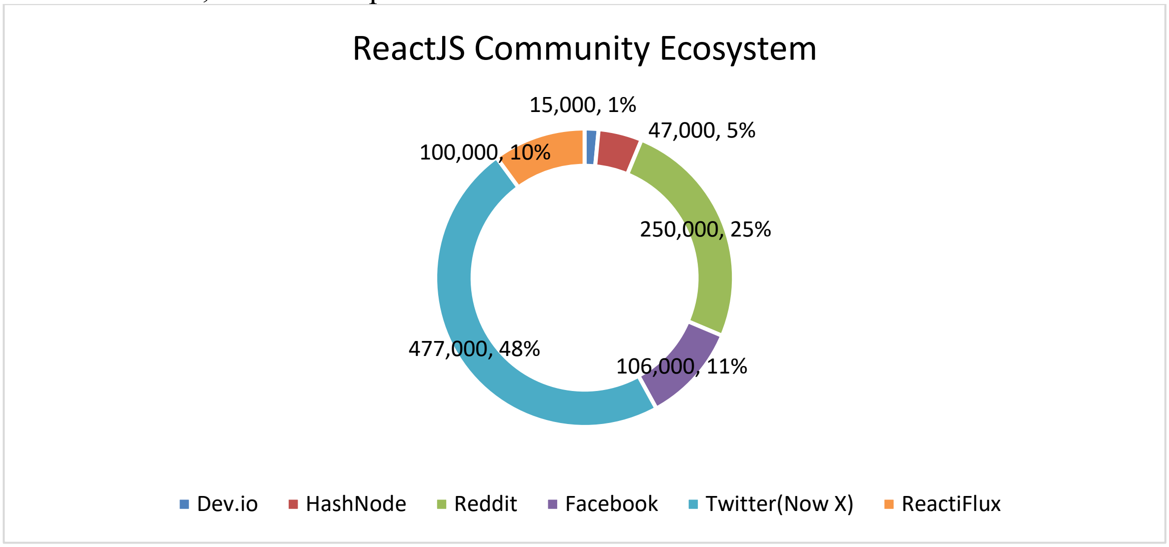
- Excessive liberty

React lacks a definitive roadmap for web app development, offering considerable freedom but with associated drawbacks.

- Excessively wordy code

Due to the abundance of technologies and approaches, combined with the absence of strict guidelines, the code can be challenging for newcomers to grasp. As a result, onboarding new developers for large React projects may require additional time.
The most fascinating thing about React is its huge Ecosystem of developer communities whom are always willing to help other developers answers questions and concerns encountered during the course of using the framework to build an application.
According to the survey;

Hashnode: 47,000+ followers
Reddit: 250,000+ React developers
X(formerly twitter): 477,000+ followers
Facebook: 106,000+ followers
Dev.to: 15,000+ posts published
Reactiflux: 100,000+ developers



React is part of the **MERN stack**, which encompasses four key technologies:
M - MongoDB: A NoSQL (non-relational) database
E - Express: A back-end middleware
R - React: A front-end framework
N - Node.js: A runtime environment

3.2.3 Angular Framework

Angular, backed by Google, is an open-source JavaScript-based platform for front-end user interface development. Its origins can be traced back to 2009, when Google engineers Misko Hevery and Adam Abrons developed the framework, initially known as AngularJS, and officially released it in 2010 [21]. Angular is an open-source JavaScript framework written in TypeScript. Its main focus is on building single-page applications (SPAs).
In Angular applications, the architecture is based on foundational concepts. **Angular components** serve as the fundamental building blocks [22]. These components define **views,** which represent sets of screen elements that Angular can dynamically modify based on program logic and data. **Components** utilize **services** for background functionalities like data fetching, decoupling view-related tasks. Services, injected as dependencies into components, enhance code modularity, reusability, and efficiency. Currently 93k stars on github.

Installation of Angular on local machine
```
npm install -g @angular/cli
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
ng new my-app
cd my-app
ng serve –open
```

Angular Architecture

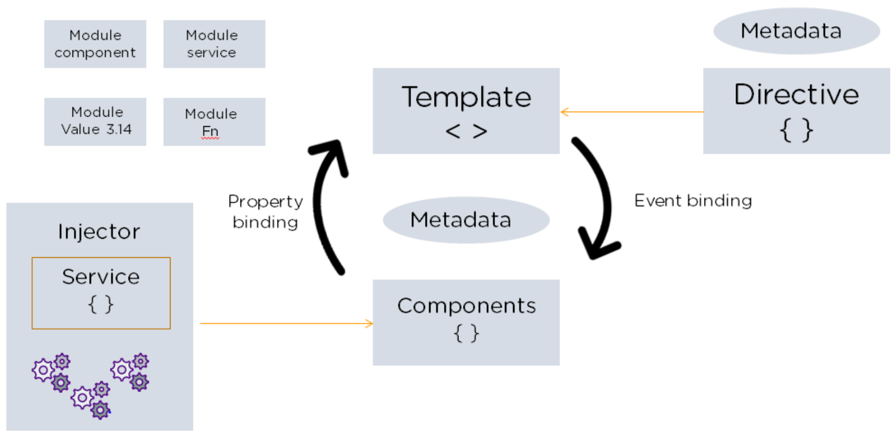


Figure 24.Angular Building Architecture

3.2.3.1 The Building Blocks of Angular

- Modules

In Angular, an app features a root module called AppModule. This module serves as the bootstrap mechanism for launching the application.

- Angular Components

Each component defines a class containing application logic and data, typically representing a portion of the user interface (UI).

- Services

When there's data or logic not tied to the **view** but needs to be shared across components, a service class is created, always marked with the **@Injectable** decorator.

- Injection Dependency

Dependency Injection (DI) facilitates providing components with necessary services and data. In Angular, DI is integrated into the framework, eliminating the need to fetch data from the server, validate user input, or perform console logging. Services can be injected into components seamlessly, as the injector manages a container of service instances. If an instance is absent, the injector creates and adds it to the container for future use.

- Data Binding

It synchronizes and coordinates components and templates, establishing a communication channel between template parts and component parts.

Angular Code sample for Data binding

```
<p>Topic: {{angular.Architecture}} </p>
<p>Place: {{Data.Flair}}</p>
```

- Angular Directives

A directive, marked by the @Directive decorator, guides Angular in transforming the DOM as templates render dynamically. Angular components consist of both templates and directives. There are two types of directives: Structural, which adjust layout by modifying DOM elements, and Attribute, which alter the behavior of existing elements.

- Ng – Model Directives
- Ng-Bind Directive

- MetaData

Metadata, often referred to as data about data, in Angular is represented by decorators. It serves as a guiding principle for Angular in processing a class, configuring it according to its intended behavior. Decorators, such as @Component, are attached to TypeScript code, like AppComponent, indicating to Angular that it's a component.

A metadata code sample [42].

```
export class AppComponent {
  constructor(@Environment('test' private appTitle:string) { }
}
```
"@Environment metadata is applied to the property appTitle with the value 'test'."

Template
The template is essential for formatting and enhancing the application prior to display. It merges Angular Markup with HTML, allowing for manipulation of elements pre-display. It also facilitates program logic and links application data with the DOM using binding markup. Two types of data binding are employed:

a. Event Binding: Responds to user input by updating application data.
b. Property Binding: Reflects computed values based on application data into the HTML.
Installing Angular CLI with NPM to start a new Project

3.2.3.2 Logical Reasons to Adopt Angular

- Browser Compatibility
- Highly Comprehensive Framework

Angular offers built-in solutions for server communication, application routing, and more, making it a comprehensive framework.

- Testing

Tests are integral to Angular, designed for comprehensive testability. It's highly recommended to test every part of your application.

- Angular CLI facilitates seamless updates.

The Angular command-line interface (CLI) is popular among engineers for several reasons. It's easy to set up, newbie-friendly, includes testing tools out of the box, and offers simple commands.

3.2.3.3 Issues with Building an Application with Angular

- Migrating from AngularJS to Angular takes time.

Lazy loading allows rendering parts of the AngularJS app within the Angular application. Migrating from AngularJS to Angular is a significant task, unlike updating between Angular versions. It can be challenging, especially with legacy systems.

- Can be difficult to learn

The learning curve is steep due to the array of topics like modules, dependency injection, components, services, and more. Additionally, mastering RxJS for asynchronous programming is essential but daunting. TypeScript, while beneficial for code maintainability, adds another layer of complexity to the learning process.

- Complex Angular Component Structure

Managing components in Angular can be overly complex. Each component may require up to five files, involving dependency injection and lifecycle interface declarations. Additionally, Angular-specific third-party libraries and syntax add further complexity.
ANGULAR is a key component of the **MEAN stack,** encompassing four essential technologies for software product development:
M - MongoDB: A NoSQL (non-relational) database
E - Express: A back-end middleware
A - Angular: A front-end framework
N - Node.js: A runtime environment

3.2.4 VueJS

VueJS is a dynamic JavaScript framework tailored for crafting engaging web interfaces. It prioritizes front-end development, offering seamless integration with various projects and libraries. Installation is straightforward, making it accessible even to beginners who can swiftly embark on creating their own user interfaces. Vue has over 3.5 million downloads in December 2023 and over 180,000 stars on Github.

3.2.4.1 Vue installations

- Using NPM

```
npm  install vue
```

- Using CDN(content delivery network)

The latest version of VueJS can be accessed via the link: https://unpkg.com/vue. Additionally, VueJS is available on jsDelivr (https://cdn.jsdelivr.net/npm/vue/dist/vue.js) and cdnjs (https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.0/vue.js).

- CLI command Line Installation

```
npm install --global vue-cli
```

- HTML Script Tag

```
<html>
  <head>
    <script type = "text/javascript" src = "vue.min.js"></script>
  </head>
  <body></body>
</html>
```

3.2.4.2 Vuex

Vuex, described on its official site, serves as a state management pattern and library for Vue.js applications. It establishes a central store accessible by any component, promoting better state management and security. Developers can create rules and methods to control state mutations, ensuring data changes adhere to a specified pattern to prevent conflicts and unwanted behavior. As Vue.js applications expand, passing data between components via props can become cumbersome and time-consuming. Vuex alleviates this complexity and facilitates easier maintenance.

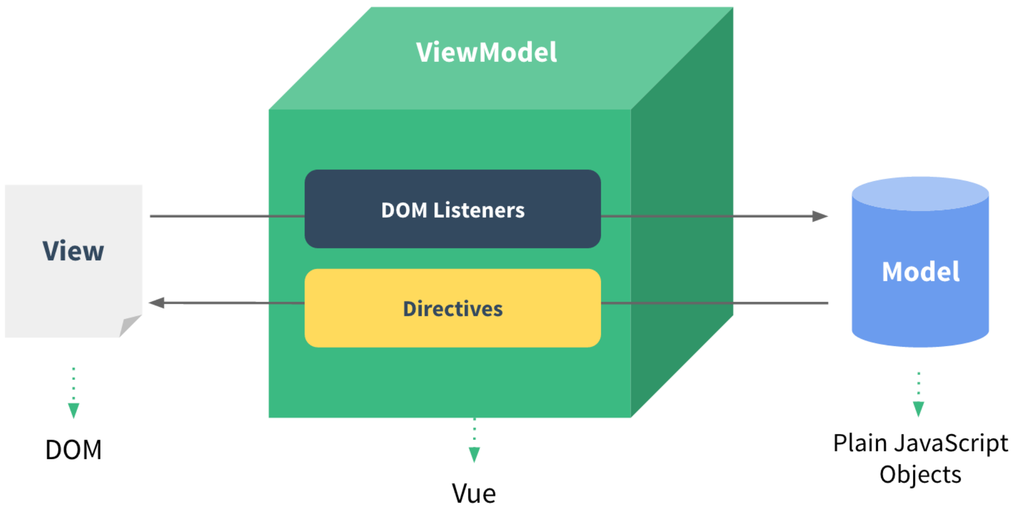


Figure 25. Vue View Structure (vuejs.org 2024) [23]

ViewModel

The Vue.js ViewModel is the core instance of a Vue application that connects the data and logic defined in the Vue component with the DOM. It manages the data and state of the application, and acts as a bridge between the data model and the view, enabling reactive and declarative rendering of the user interface. In essence, the ViewModel represents the application's state and behavior, allowing for efficient data binding and manipulation [23].

```
var vm = new Vue({ /* options */ }) #-> code snippet from vuejs.org
```

View

Vue.js employs DOM-based templating, associating each Vue instance with a DOM element. Upon creation, a Vue instance traverses its root element's child nodes, establishing data bindings. Once compiled, the View becomes reactive to data alterations.

```
vm.$el  -> snippet from vuejs.org 2024
```

Model

VueJS excels in reactivity, simplifying data binding between HTML and JavaScript. Its seamless handling of two-way reactive data binding ensures that changes in data reflect in the DOM effortlessly. This feature makes VueJS ideal for SPAs and any application necessitating real-time updates.

```
vm.$data //  -> snippet from vuejs.org
```

3.2.4.3 Logical Reasons to Adopt VueJS

- **Easy to Learn**

Experienced developers transitioning from other JavaScript frameworks find Vue easy to pick up due to its simplicity and clarity. Vue's straightforward syntax combines HTML and JavaScript seamlessly in its components, making the structure intuitive. This simplicity facilitates the development of large-scale templates while maintaining ease of error tracking.

- **Virtual DOM rendering enhances performance.**

Vue.js leverages a virtual DOM to efficiently track and update changes in data and UI, minimizing DOM operations and enhancing performance. Additionally, its reactivity system automatically detects dependencies between data and UI, updating the UI dynamically upon data changes.

- **Vue Components**

Components and views are small, interactive app parts seamlessly integrated into existing infrastructure without compromising the system.

- **Browser Dev Tools**

The Vue team has developed excellent browser devtools extensions for their framework. These tools enable programmers to inspect Vuex state, components, and views, modify data, and analyze events in depth.

3.2.4.4 Issues with Building an Application with VueJs

- Limited Plugins, Extensions and Libraries
- Not Enough support from Vuejs community of developers
- Too flexible leading to code errors and irregularities

VueJS is part of the **MEVN stack**, which encompasses four key technologies:
M - MongoDB: A NoSQL (non-relational) database
E - Express: A back-end middleware
V- Vue: A front-end framework
N - Node.js: A runtime environment

As we can see below, ReactJS seems to be the favorite amongst JavaScript developers. Our qualitative assessment in the part B would give us a better understanding why the React Framework is a popular choice amongst JavaScript and its community of developers.

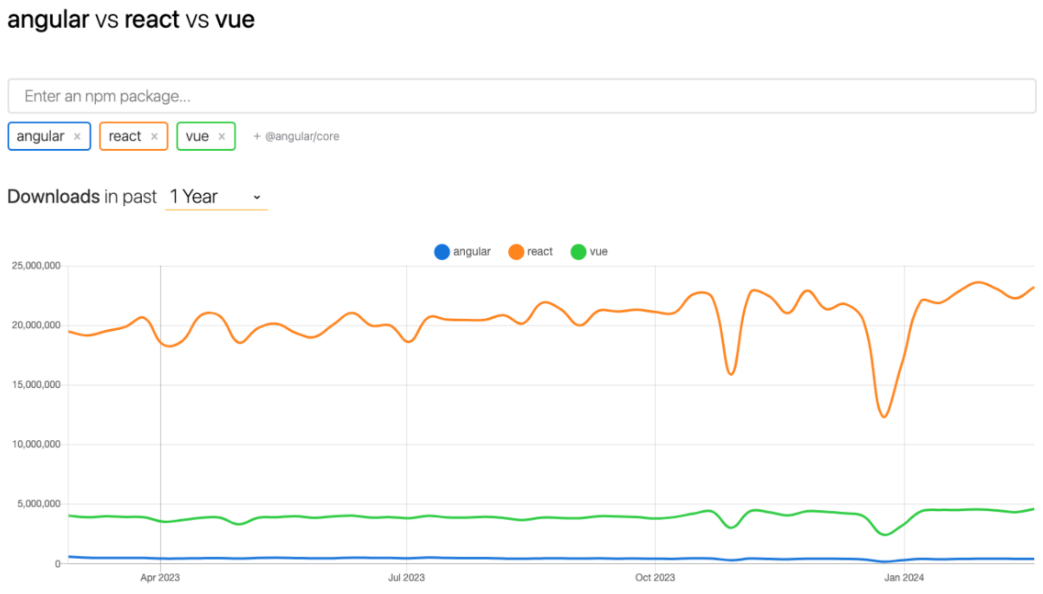


Figure 26. popularity between the 3 major frameworks [npmtrends.com Jan 2024]

II. ANALYSIS

3.3 PART B - RESEARCH APPROACH (Qualitative Model)

As part of this research, a survey (consisting of 14 Questions) has been distributed to JavaScript users through various channels such as FreeCodeCamp forums, CodeAcademy Forums, workplace employees, Discord JavaScript Forums, and WhatsApp groups. The aim is to collect valuable insights on JavaScript developers' attitudes and thoughts regarding the JavaScript Tech Stack.
The survey was created using Google Forms for its user-friendly interface and automatic graphing of submitted results, facilitating efficient data analysis for this thesis.

I'm presently engaged in qualitative research on the JavaScript ecosystem within the European and African tech communities. The insights being gained and analyzed from this research are driving my decision towards adopting the **MERN STACK** for building the web application discussed in Chapter 4.
All Research documents used will be included in the final version of this Thesis.

3.3.1 RESEARCH QUESTION

➔ **Developer Bio (KYD- know your developer)**

- Kindly state your name (Information to be kept anonymous due to data confidentiality)
- State your nationality
- Select your employment status
- What is your job role?

➔ **Main Research Questions**

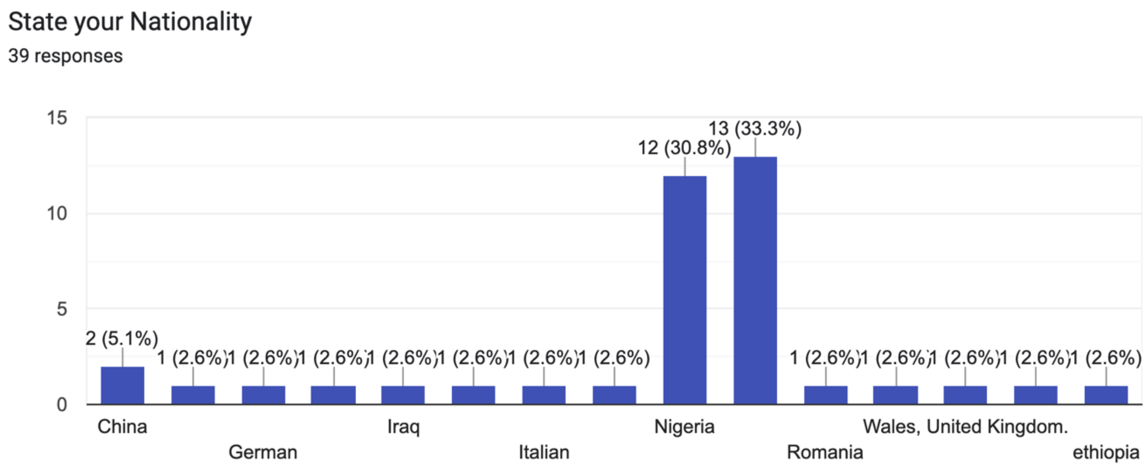
- What is your JavaScript programming experience level?
- Are you currently working in software development?
- What is your job Role?
- What industry do you work in and where is your company located?
- Why did you decide to adopt JavaScript as a programming language for development?
- Do you currently use a JavaScript Framework for development?
- Which of the Listed Frameworks do you use mostly professionally?
  ➔ ReactJS
  ➔ Angular
  ➔ VueJS
- How often do you switch between JavaScript Frameworks when developing a software and why?
- Which of the package Manager do you use in development?
- How Important has Package Managers been to your development lifecycle?
- What security concerns have you encountered using NPM and what do you think can be improved?
- How would you rank the learning curve of your preferred Framework?
- How Important has the developer community of your framework of choice been to your development workflow?
- What are the drawbacks you encounter in production when using your preferred Framework?

- What do you personally think could be improved in the ever-growing JavaScript ecosystem?
- Do you have any Additional Thoughts that you think is not been covered in this survey questions?

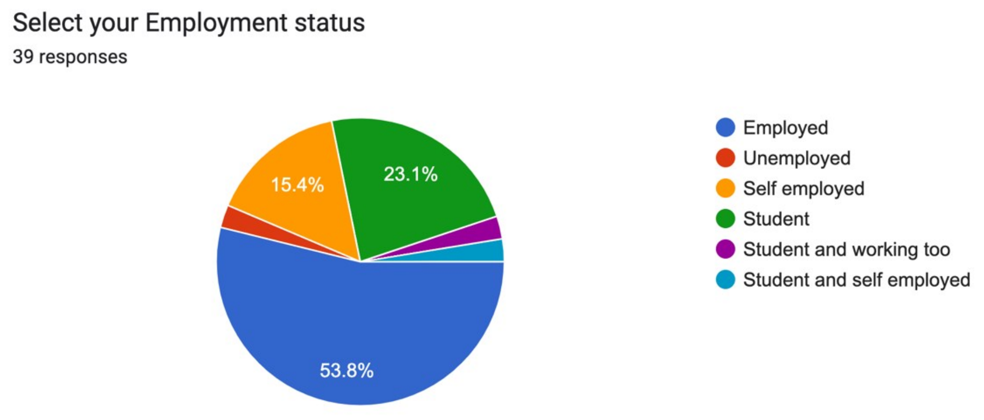
3.3.2 Qualitative Result of the Survey

The survey didn't randomly select participants, and it only included people who were willing to take it, which naturally limits how representative the results are of all developers. So, we can't use these results to say much about the broader developer community in the grand scheme of things. These results are solely of the basis of this Thesis research.

3.3.2.1 Know Your Developer (KYD)

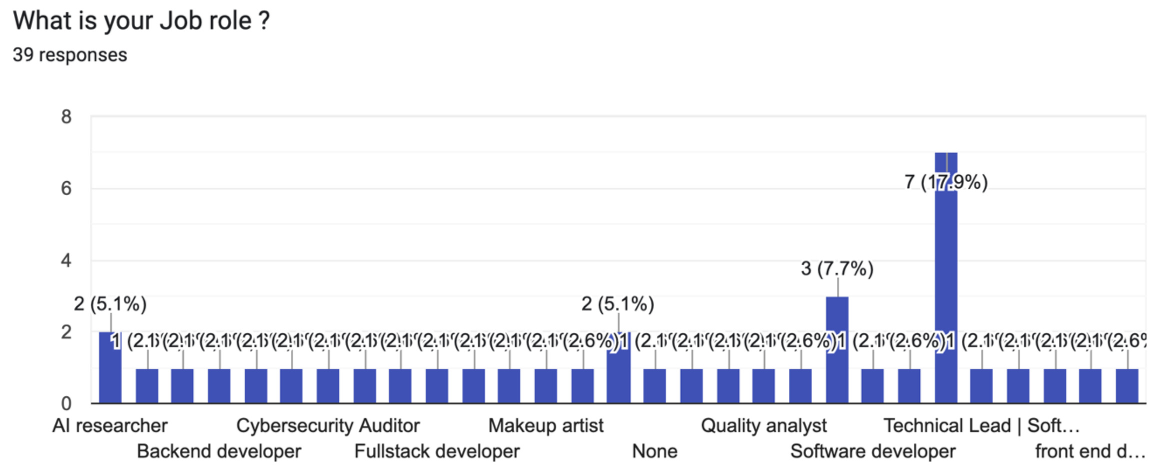


Survey Results:
26 developers stated Nationality as Nigeria/Nigerian
2 Chinese surveyed developers
1 czech/Czechia developer
1 German developer
1 Ghanaian developer
1 Iraqi developer
1 Irish developer
1 Italian developer
1 Romanian developer
1 Welsh developer
1 Trinidad and Tobaggo developer
1 Zimbabwean developer
1 Ethiopian developer



Survery Results:

21 (53.8%) of the respondents were gainfully employed. 15.4% (6 developers) were self-employed, 11 respondents (28.3%) were students/looking for work/part-time work/student/coding for fun e.t.c, 1 respondent (2.6%) was unemployed.



Survey Results:

The survey results revealed that 2 respondents work as AI researchers, 2 respondents are Backend Developers, 1 respondent is a Freelancer, 1 is a Human Resource/Business Data Analyst, 1 is a CEO, 1 works as a Developer for McDonald's enterprise, 1 is a Cybersecurity Auditor, 1 is a Full Stack Developer, 1 works as a Graphic/Web Designer, 1 is an IT Administrator, 1 is a Makeup Artist currently transitioning to tech, 3 did not specify a job role (N/A), 1 works in operations, 1 is a Product Consultant, 1 works in QA (Quality Assurance), 1 is a Social Media Marketer, 6 are Software Engineers/Software Developers/Technical Leads, 4 are Front-End Developers/Front-End Engineers, and 7 are students.

3.3.2.2 Q1- Q14 Experience with Javasript, FrameWorks and Node Package Manager(npm)

The purpose of these questions was to assess the significance of the JavaScript ecosystem, including its frameworks, to the surveyed developers, as well as to evaluate their perceived ease of use.

The percentages indicate the distribution of respondents in each category.

Q1

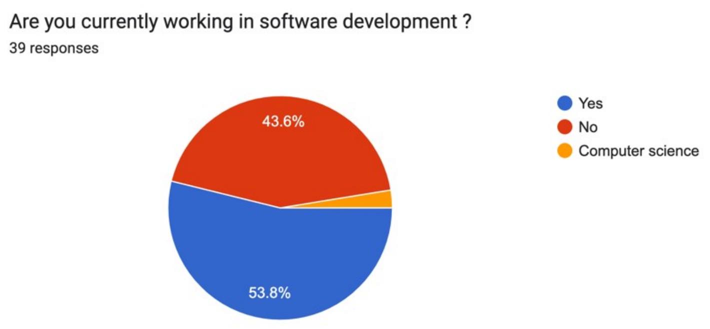


Figure 27. Q1 survey result

Out of the total respondents:

21 (53.8%) are somewhat working in or have recently worked in software development.
17 (43.6%) are currently not working in software development but in other fields.

Q2

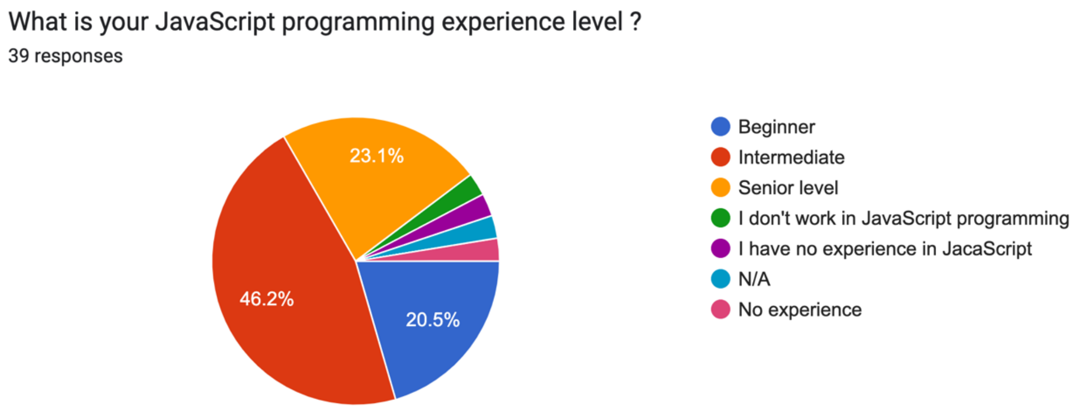


Figure 28. Q2 survey result

Among the respondents, 9 (23.1%) listed senior-level experience, while 19 (46.2%) have intermediate experience in the JavaScript language, and 8 (20.5%) have a beginner-level knowledge of the language.

Q3 – ("Why did you decide to adopt JavaScript as a programming language for development")?

The following responses was submitted from 22 respondents.

- For upgrade
- It was the most suitable for my job role
- Because of its asynchronous nature.
- It is easy to understand
- JavaScript's flexibility and robust ecosystem make it an ideal choice for developing modern web applications that meet our project requirements.
- I adopted JavaScript because it is the language of the web. It works on all major web browsers, making it the front-end development language. JavaScript is also used on the server side (Node.js), for the development of mobile applications (React Native, Native Script) and desktop applications (Electron). It's versatility and wide adoption make it an invaluable technology for developers.
- I enjoy using JavaScript because it has a diverse ecosystem of libraries, frameworks, and tools that facilitate development across multiple domains. Whether I'm creating a simple website, a complex web application, or a mobile application, there is likely a JavaScript library or framework that I can use to develop these additional processes and work.
- With the advent of frameworks like React Native and Electron, I can use JavaScript to create cross-platform applications that run on a variety of devices and platforms. This allows me to use JavaScript technology to create a variety of applications without having to learn another language.
- Easy to use
- The adoption of JS started off as a need to add interactivity to websites.
- Because of its versatility
- Easy to learn and was a widely used language
- Because fasting my work
- I was learning to become a frontend developer and it was the next logical step in my path
- Convenience of deployment, availability of talent pool
- because of its functionality
- It was kind of what I had available to me at the time. The only Udemy course I had access to back then was NodeJS so JavaScript was the way forward.
- because it is basically useful for frontend programming and user-friendly
- it is dynamic with static pages. Also, JavaScript is very useful in websites development implementation 3Js, which develops websites in 3D.Our startup company also uses CSS, html and JavaScript on its website. i am sending you the address https://association-for-engineering-in-science.webflow.io
- For its versatility, as it's a widely supported language that can be used for both front-end and back-end development. Its ecosystem and community support also contribute to its popularity.
- Cause my interest was mainly on the frontend part of the website
- I wanted to have a skill

Q4

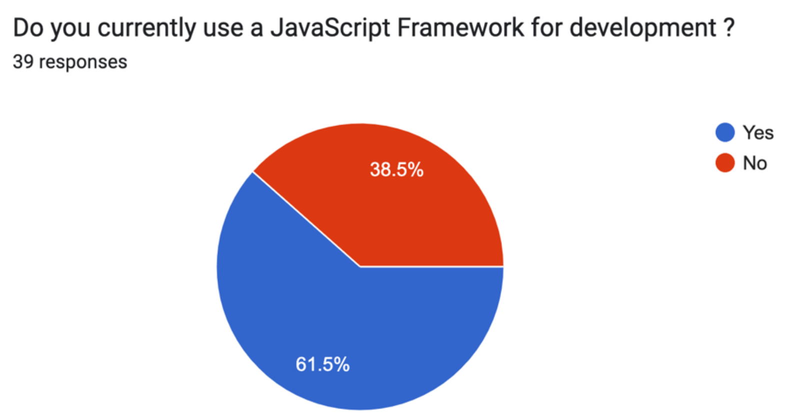


Figure 29. Q4 survey Result

Out of the respondents, 24 (61.5%) currently utilize JavaScript in some capacity for development, whether professionally or for research purposes, while 15 (38.5%) are currently employing other programming languages instead of JavaScript.

Q5

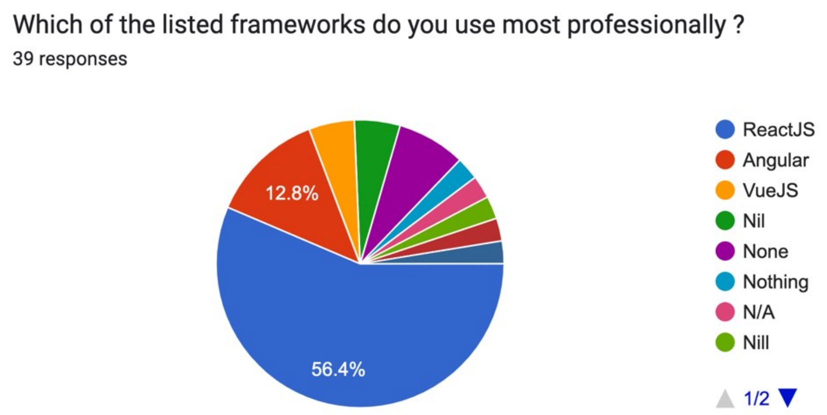


Figure 30. Q5 Survey Result

Among the respondents, 22 (56.4%) utilized ReactJS the most, 5 (12.8%) used Angular professionally, 2 (5.1%) used VueJS, while the remainder utilized ExpressJS or jQuery.

Q6 – ("Why do you use the above selected frame work ")?

The following responses was submitted from 18 respondents

- that is what the company I work uses in building their applications
- Cos it offers two-way features like data binding and dependency injections for building single page applications.
- ReactJS is user friendly, allowing developers like myself reuse codes at will.
- It has a virtual DOM and integration framework that helps improve performance by reducing the number of DOM manipulations. React also provides tools like

React, memo and Pure Component to optimize actions and avoid unnecessary repetition.
- I make use of React because it follows a unidirectional data flow, where data flows in only one direction: from the parent component to the child component. This makes data easier to manage and reduces the chance of bugs due to inconsistent state.
- I prefer ReactJS because of its virtual DOM and integration framework which can help improve performance by reducing the number of DOM manipulations. React also provides tools like React, memo and Pure Component to optimize actions and avoid unnecessary repetition.
- Because Vue.js has an easy learning curve, making it accessible to both beginners and experienced developers. Simplicity allows developers to speed up content and start building applications efficiently.
- It's the one I'm best familiar with.
- Vue felt natural to pick up and was quite easy to pick up.
- Building user interfaces on React is quite interesting and innovative
- It's very good for frontend web development
- Easier to navigate
- It is the current industry standard and has more opportunities
- It's the framework with very available jobs
- We use React because it's a combination of JavaScript, HTML and CSS. This allows us to create an integrated website
- For its component-based architecture, virtual DOM for efficient updates, and a strong community. It simplifies building interactive user interfaces and enables the creation of reusable UI components.
- Reusable component, great state management, declarative design
- it is a powerful library for building web and native user interfaces. Whether you're creating a simple web page or a complex application, ReactJS allows you to construct user interfaces by combining individual pieces called components.

Q7 – **(“**How often do you switch between JavaScript Frameworks when developing a software and **Why”)?**

The following responses was submitted from 13 respondents

- quite often due to the vast range of projects we work on
- Not really often, I stick to ReactJS
- Not so often, ReactJS serves me better.
- not so often, I prefer VueJS
- I've only used one framework till date; Vue.
- I don't develop much, but when I do ReactJS does it for me Same reasons as the above
- Can't use reactJS for everything I'm doing
- Not often. Requirements are defined before project starts
- We don't switch because Express is sufficient for all our needs so far
- I seldom do so since I had experience with a vast majority of them so I can easily plan which to use and why to use.

- I do not switch between JavaScript frameworks
- I hardly switch because ReactJS is enough for now
- Sometimes, the requirements of a project may evolve in such a way that the current framework becomes less suitable or efficient. In such cases, we might switch to another framework that better aligns with the new requirements.

Q8 – **(“**What package Manager do you use in development”)?

10 respondents listed npm, 3 listed Yarn, 1 listed Jenkins, and 1 listed webpack.

Q9 – (“How Important has Package Managers been to your development lifecycle”)?

The following responses was submitted from 11 respondents

- As my projects increased in size and complexity, manually managing dependencies became increasingly difficult for me. Package managers are a great solution for managing dependencies, making it easy to manage large code, refactor code, and introduce new features without unnecessary hassle or expense.
- Package managers maintain control to ensure that I use the same set of dependencies across different environments. This improves the consistency and reproducibility of the development process, reducing the possibility of bugs or bad behavior due to version conflicts.
- The package manager integrates with development automation tools like npm scripts, webpack, or Gradle, allowing developers to perform tasks like installing dependencies, bundling, minifying, and optimizing the coded. This automation helps improve efficiency and consistency, especially for large projects.
- Package managers improve development performance by providing a central repository of reusable components, libraries, and tools. Developers can quickly find and install the packages they need, saving time and effort compared to downloading and managing dependencies.
- They provide essential software and easily install those packages for use
- They simplify dependency management, making it easier to install, update, and share libraries or packages. Package managers streamline the development process, enhance collaboration, and ensure consistency across different environments. They contribute significantly to the efficiency and reliability of software development.
- Made development easier
- It's impossible to do anything without them.
- Very helpful
- It has helped me reuse codes at my own free will, easy to use, easy for sharing of codes, and easy to manage.
- makes development faster.

Q10 – (“What security concerns have you encountered in using NPM and what do you think can be improved”)?

The following responses was submitted from 12 respondents

- Vulnerability Malicious package Lack of package verification
- NPM requires developers to authenticate when publishing packages or accessing private domain names. However, credential leakage, such as breach of sensitive information (such as an authentication token or API key) to an administrative authority, may result in not being authorized to access your npm account or your private packages.
- I've had an experience where an attacker published a package with the same name as a package used by my company. If an organization's internal package, json file does not specify the package, NPM may download the wrong package from the public registry instead of the internal registry, leading to security violations. A way I think the security of NPM can be improved is by carefully reviewing a reliable and reputable package before including it as a dependency.
- The NPM open-source registry allows anyone to publish packages, which can sometimes lead to faulty or broken packages. These packets may contain malicious code, such as backdoors, malware, or code designed to steal sensitive information.
- When i used older versions of packages that have known security vulnerabilities due to compatibility issues or rejected updates. Constantly updating dependencies made my project vulnerable to exploits because attackers started targeting these vulnerabilities to compromise my application
- I've not faced any till now. They already allow me audit packages that I plan to use.
- I haven't encountered any issues so far
- dependency vulnerability
- Outdated packages and packages with known security vulnerabilities. For improvement, auto updating of know vulnerabilities option should be provided
- Security concerns in npm include the risk of installing malicious packages and vulnerabilities in dependencies. Improvements can be made through continuous monitoring, tools like npm audit, code reviews, access controls, and maintaining transparency in package information.
- Poor audit of npm packages
- Dependency Vulnerabilities,Package Hijacking and for Malicious Package.

Q11 – (How Important has the developer community of your framework of choice been to your development workflow)?

The following responses was submitted from 14 respondents

- without the community i would not be a professional developer today
- They've been of great importance and support

- The developer community provides me with valuable feedback and peer reviews to help developers like myself improve their code quality, design decisions, and solutions. Peer review promotes accountability, transparency, and continuous improvement, resulting in better software products and practices.
- The developer community has provided a platform for sharing knowledge, experiences, best practices, tips, and tricks. Forums, online communities, meetups, conferences, and workshops allow developers to learn from each other, collaborate to solve problems and stay up to date with the latest trends and new technologies
- The developer community provides support and training to me in all experience.
- The developer community champions important values such as diversity and inclusion, accessibility, appropriate software development, and environmental sustainability. By raising awareness and encouraging positive change, developers can use their voices to benefit me, people, and the tech industry as a whole.
- Very important! It's quite easy to find solutions to problem on the web.
- Well, for the community bit, it's a small group of friends and former colleagues who just assist when certain issues are encountered
- Feedback and critique
- Very important, it's important to have access to resources when you need them
- Very important
- The developer communities have been essential components of the software development ecosystem, fostering collaboration, innovation, and continuous improvement. Active engagement with the community can enhance the development workflow, accelerate learning, and contribute to the success of projects built using a particular framework or technology.
- Fresh ideas
- They've been of great importance and support.

Q12 – ("What are the drawbacks you encounter in production when using your preferred Framework") ?

The following responses was submitted from 5 respondents

- limited flexibility and bloated code
- Limited pool of experienced developer
- Incompatibility of packages with Node version when updates to the deployment server have to be made
- Common issues include performance bottlenecks, framework updates affecting compatibility, and potential security vulnerabilities. Additionally, a steep learning curve or lack of certain features may pose challenges. It's crucial for developers to stay informed, address these issues promptly, and adapt their strategies to ensure smooth production experiences.
- Minification size of production Build

Q13 – ("What do you personally think could be improved in the ever-growing JavaScript ecosystem")?

The following responses was submitted from 10 respondents

- The language needs to be strongly typed like python
- JavaScript's standard library lacks many functions and data structures found in other languages. Enhanced standard libraries with more comprehensive and consistent APIs for operations such as string processing, data processing, and data structures reduce reliance on third-party libraries and improve production.
- Dynamic typing in JavaScript can cause errors that are only discovered at runtime. Adding options like typing, like those found in TypeScript or Flow, can help you catch more errors during development and provide better support for code navigation and refactoring
- Since Javascript's ecosystem still relies heavily on CommonJS modules, which have different syntax and behavior. Improved support for ES modules across all environments and libraries will help unify the module ecosystem and improve interoperability.
- JavaScript engines have seen significant improvements over the years, but there is still room for improvement, particularly in areas such as garbage collection, memory management, and just-in-time (JIT) compilation compatibility. . Additional JavaScript performance optimization can improve the responsiveness and performance of your website.
- Standardization: Enhancing standardization across libraries and frameworks to promote consistency and reduce fragmentation.
- Libraries with less steep learning curves would be a welcome development
- Performance Improvement
- Package Security: Strengthening package security to minimize the risk of installing malicious or compromised dependencies.
- Documentation: Enhancing and maintaining comprehensive, up-to-date documentation for libraries and frameworks to facilitate easier adoption and troubleshooting.

Q14 – (Do you have any Additional Thoughts that you think is not been covered in this survey questions)? If yes, kindly state them (optional)

The respondents largely ignored this last question and in some cases their responses were inconsequential to our research statement, hence Q14 not going to be taking into consideration.

4 WURA APP

In this chapter, we're diving into WURA APP, an online platform for movie lovers. It's a real-life example of how we combine what we learn in the previous chapters about JavaScript with actually building something useful.

WURA started with our understanding of JavaScript, the programming language behind it all. We studied its basics like data types and how it handles events. With this knowledge, we decided to create a platform that solves a real problem.

In the last chapter, we looked at what other experts have said about programming languages like JavaScript. This helped us see the bigger picture and understand where WURA fits in.

Then, we got practical. We compared three popular JavaScript frameworks to figure out which one would work best for WURA. We even surveyed professionals who use JavaScript daily to make sure we were on the right track.

Now, it's time to get our hands dirty with WURA. We'll explore how it's built, what features it has, and how users interact with it. This will show how frameworks not only make coding easier but also improve the user experience.

As we wrap up this chapter, we want to think about why WURA matters. It's not just about building cool stuff; it's about how frameworks shape the internet and empower developers like us.

In the next chapter, we'll bring everything together. We'll share what we've learned from building WURA and give some advice on using frameworks effectively. Our goal is to inspire others and keep pushing the boundaries of web development.

4.1.1 Functional Requirements: (What the app must do)

- Feature a SignUp Screen for a user to create account
- A login Screen for a user to login after creating the account
- A Sign up feature with Google Auth
- A Sign Up feature with Github Auth
- An Error Popup to indicate Incorrect Credentials
- Registered User information is automatically saved in MongoDB using PrismaClient
- A profile Screen that shows the registered User
- The HomeScreen of the application
- A play Button that Plays the movie selected by the User
- An Info Button that displays the selected Movie details
- A modal screen Linked to the (Info button)
- A SignOut Feature on the HomePage
- A UserInfo section displaying the name of the registered User
- A trending Movie section
- It is responsive on mobile Screen

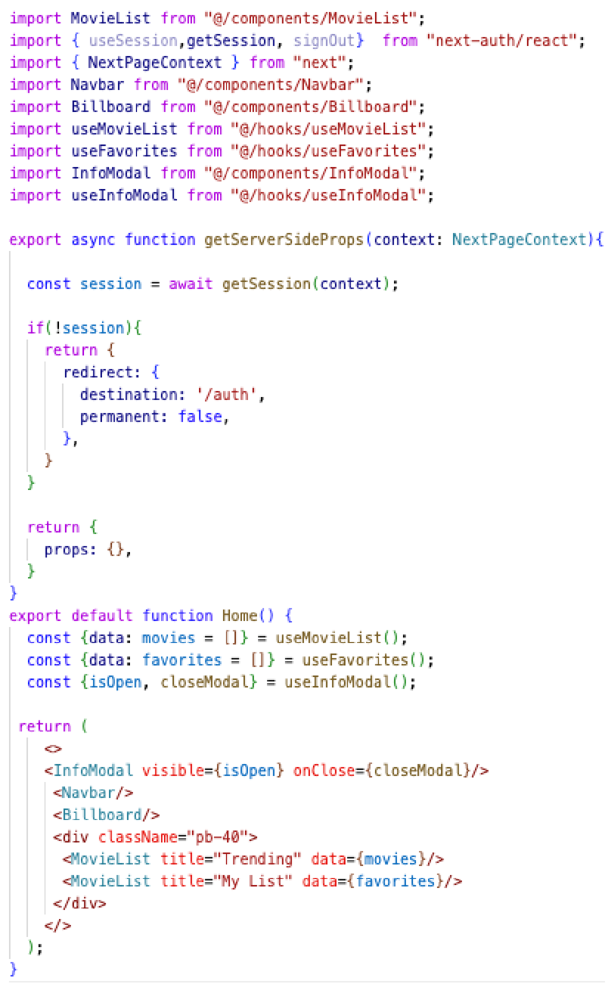
Tools used in development
React, TypeScript, React-Icons, NextJS, Next-auth, MongoDB, Mongoose, PrismaDB, Vercel, Npm, Tailwind CSS, NodeJS.

4.2 Implementation Phase

4.2.1.1.1 App Logo

This is the application logo.

4.2.2 Application Main Build Structure



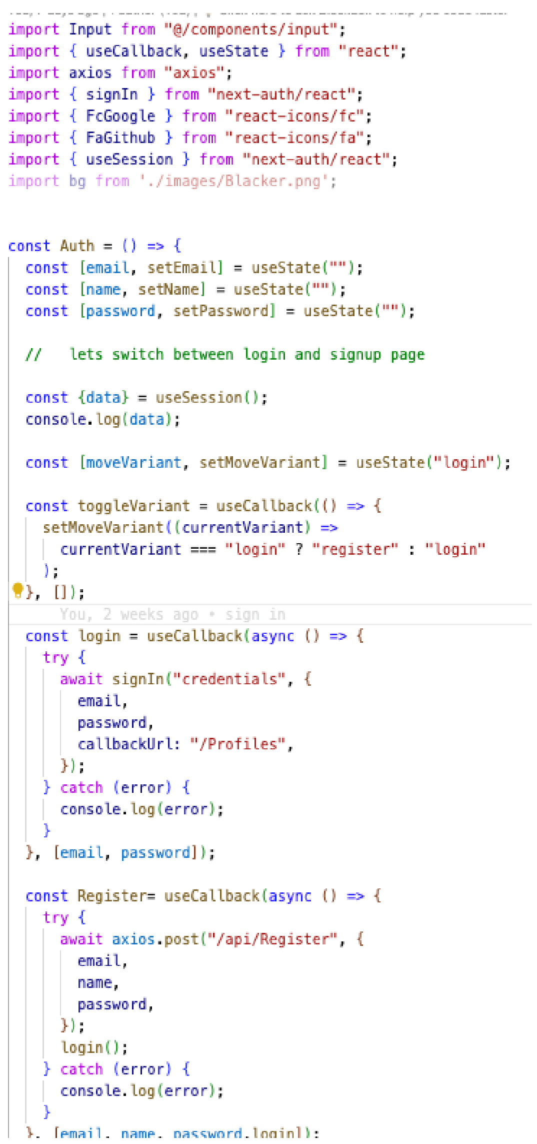
```
import MovieList from "@/components/MovieList";
import { useSession,getSession, signOut}  from "next-auth/react";
import { NextPageContext } from "next";
import Navbar from "@/components/Navbar";
import Billboard from "@/components/Billboard";
import useMovieList from "@/hooks/useMovieList";
import useFavorites from "@/hooks/useFavorites";
import InfoModal from "@/components/InfoModal";
import useInfoModal from "@/hooks/useInfoModal";

export async function getServerSideProps(context: NextPageContext){

  const session = await getSession(context);

  if(!session){
    return {
      redirect: {
        destination: '/auth',
        permanent: false,
      },
    }
  }

  return {
    props: {},
  }
}
export default function Home() {
  const {data: movies = []} = useMovieList();
  const {data: favorites = []} = useFavorites();
  const {isOpen, closeModal} = useInfoModal();

 return (
    <>
    <InfoModal visible={isOpen} onClose={closeModal}/>
     <Navbar/>
     <Billboard/>
     <div className="pb-40">
      <MovieList title="Trending" data={movies}/>
      <MovieList title="My List" data={favorites}/>
     </div>
    </>
  );
}
```

4.2.2.1 SignUp section

Client Side:

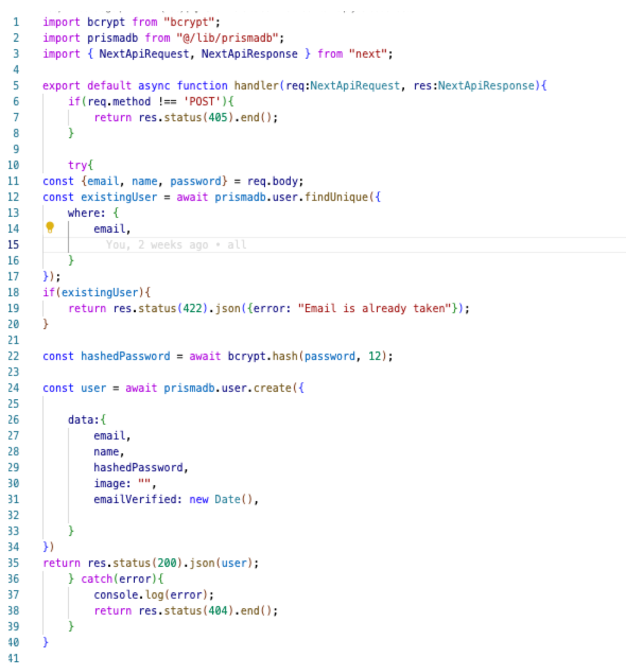
This page is for handling authentication in the application. It includes functionality for both signing in and registering new users. The user interface allows switching between the login and registration forms. It supports signing in with Google and GitHub accounts as well. The page layout is responsive, adjusting to different screen sizes. Overall, it provides a straight-forward and visually appealing way for users to authenticate and access the application's features.

Code Sample: Auth.tsx

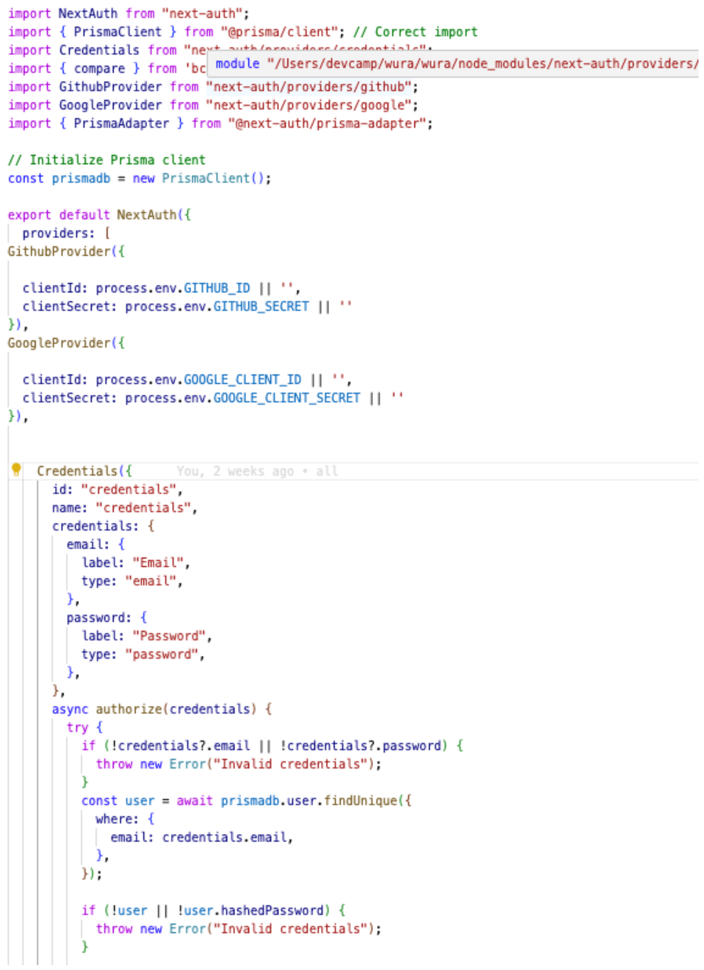


```
import Input from "@/components/input";
import { useCallback, useState } from "react";
import axios from "axios";
import { signIn } from "next-auth/react";
import { FcGoogle } from "react-icons/fc";
import { FaGithub } from "react-icons/fa";
import { useSession } from "next-auth/react";
import bg from './images/Blacker.png';


const Auth = () => {
  const [email, setEmail] = useState("");
  const [name, setName] = useState("");
  const [password, setPassword] = useState("");

  //   lets switch between login and signup page

  const {data} = useSession();
  console.log(data);

  const [moveVariant, setMoveVariant] = useState("login");

  const toggleVariant = useCallback(() => {
    setMoveVariant((currentVariant) =>
      currentVariant === "login" ? "register" : "login"
    );
  }, []);
      You, 2 weeks ago • sign in
  const login = useCallback(async () => {
    try {
      await signIn("credentials", {
        email,
        password,
        callbackUrl: "/Profiles",
      });
    } catch (error) {
      console.log(error);
    }
  }, [email, password]);

  const Register= useCallback(async () => {
    try {
      await axios.post("/api/Register", {
        email,
        name,
        password,
      });
      login();
    } catch (error) {
      console.log(error);
    }
  }, [email, name, password,login]);
```

ServerSide: Register.ts



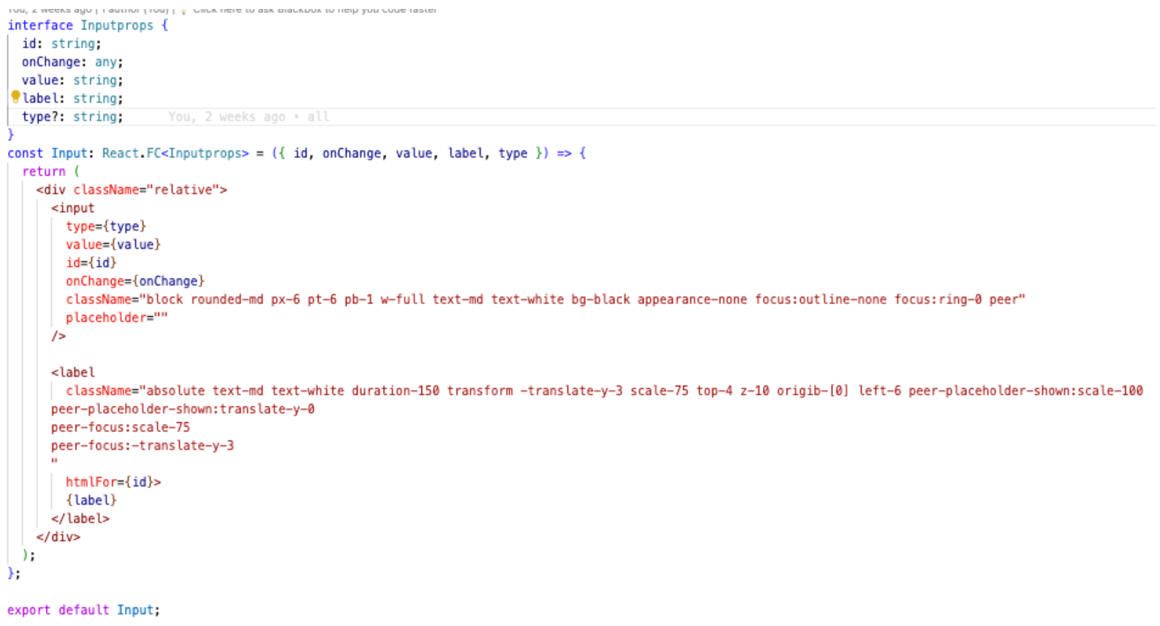
```
import bcrypt from "bcrypt";
import prismadb from "@/lib/prismadb";
import { NextApiRequest, NextApiResponse } from "next";

export default async function handler(req:NextApiRequest, res:NextApiResponse){
    if(req.method !== 'POST'){
        return res.status(405).end();
    }

    try{
const {email, name, password} = req.body;
const existingUser = await prismadb.user.findUnique({
    where: {
        email,
          You, 2 weeks ago • all
    }
});
if(existingUser){
    return res.status(422).json({error: "Email is already taken"});
}

const hashedPassword = await bcrypt.hash(password, 12);

const user = await prismadb.user.create({

    data:{
        email,
        name,
        hashedPassword,
        image: "",
        emailVerified: new Date(),

    }
})
return res.status(200).json(user);
    } catch(error){
        console.log(error);
        return res.status(404).end();
    }
}
```



```
import NextAuth from "next-auth";
import { PrismaClient } from "@prisma/client"; // Correct import
import Credentials from "ne
import { compare } from 'bc module "/Users/devcamp/wura/wura/node_modules/next-auth/providers/
import GithubProvider from "next-auth/providers/github";
import GoogleProvider from "next-auth/providers/google";
import { PrismaAdapter } from "@next-auth/prisma-adapter";

// Initialize Prisma client
const prismadb = new PrismaClient();

export default NextAuth({
  providers: [
GithubProvider({

  clientId: process.env.GITHUB_ID || '',
  clientSecret: process.env.GITHUB_SECRET || ''
}),
GoogleProvider({

  clientId: process.env.GOOGLE_CLIENT_ID || '',
  clientSecret: process.env.GOOGLE_CLIENT_SECRET || ''
}),


    Credentials({      You, 2 weeks ago • all
      id: "credentials",
      name: "credentials",
      credentials: {
        email: {
          label: "Email",
          type: "email",
        },
        password: {
          label: "Password",
          type: "password",
        },
      },
      async authorize(credentials) {
        try {
          if (!credentials?.email || !credentials?.password) {
            throw new Error("Invalid credentials");
          }
          const user = await prismadb.user.findUnique({
            where: {
              email: credentials.email,
            },
          });

          if (!user || !user.hashedPassword) {
            throw new Error("Invalid credentials");
          }
```

Figure 31.Credentials authenticator.

Code for the Input feature:input.tsx (Input button component)



```
interface Inputprops {
  id: string;
  onChange: any;
  value: string;
  label: string;
  type?: string;       You, 2 weeks ago • all
}
const Input: React.FC<Inputprops> = ({ id, onChange, value, label, type }) => {
  return (
    <div className="relative">
      <input
        type={type}
        value={value}
        id={id}
        onChange={onChange}
        className="block rounded-md px-6 pt-6 pb-1 w-full text-md text-white bg-black appearance-none focus:outline-none focus:ring-0 peer"
        placeholder=""
      />

      <label
        className="absolute text-md text-white duration-150 transform -translate-y-3 scale-75 top-4 z-10 origib-[0] left-6 peer-placeholder-shown:scale-100
      peer-placeholder-shown:translate-y-0
      peer-focus:scale-75
      peer-focus:-translate-y-3
      "
        htmlFor={id}>
        {label}
      </label>
    </div>
  );
};

export default Input;
```

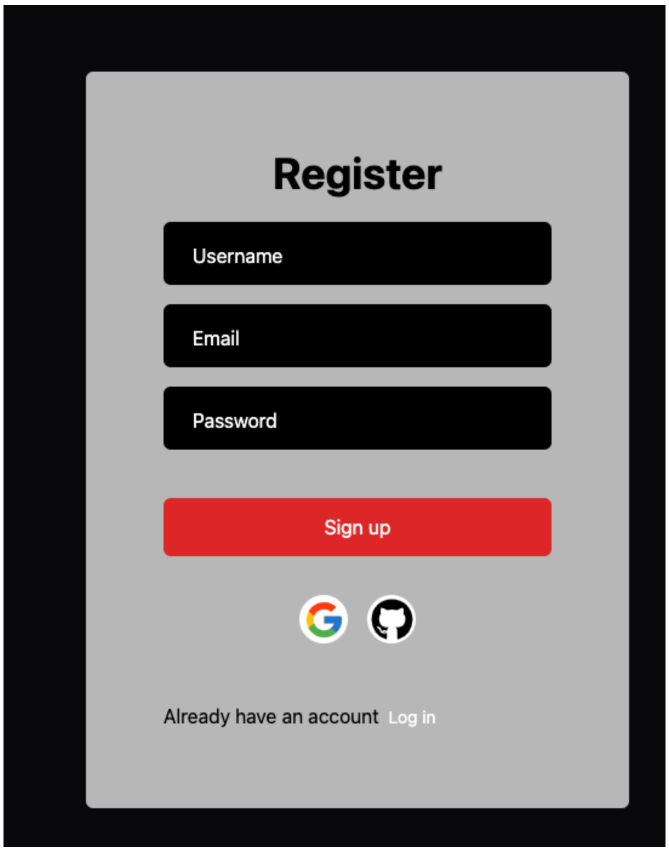


Figure 32. User on the SignUp Session

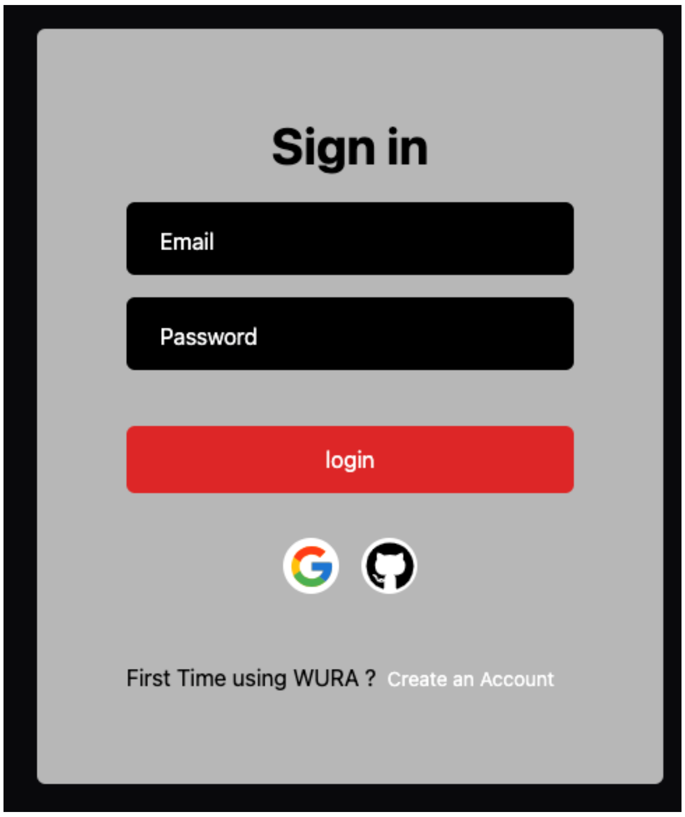
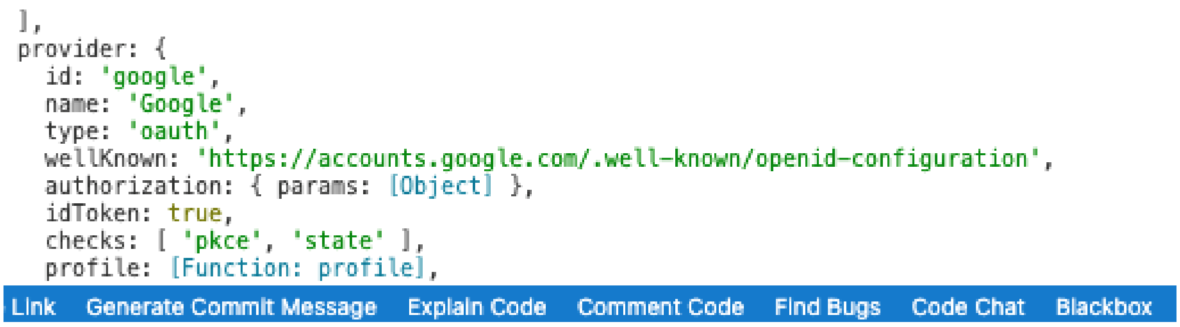


Figure 33. User on the Login Screen



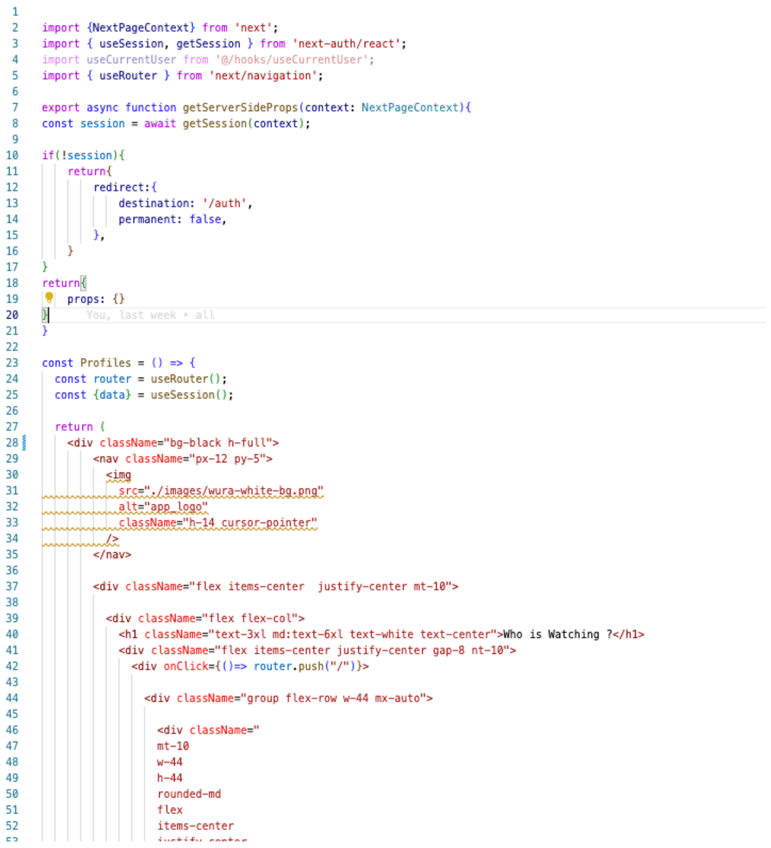
```
],
provider: {
  id: 'google',
  name: 'Google',
  type: 'oauth',
  wellKnown: 'https://accounts.google.com/.well-known/openid-configuration',
  authorization: { params: [Object] },
  idToken: true,
  checks: [ 'pkce', 'state' ],
  profile: [Function: profile],
```

Figure 34. Next-Auth Provider Response after successful SignIn.

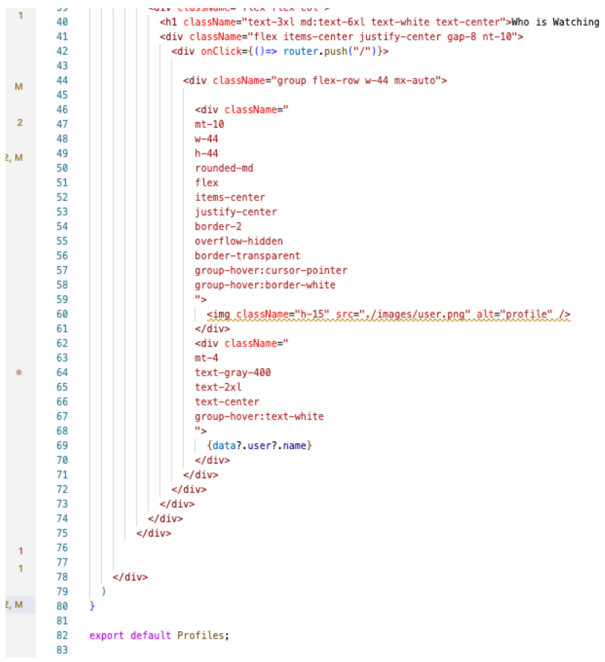
4.2.2.2 Registered User Profile Screen

This code handles user authentication and redirects users to the login page if they are not authenticated. Once authenticated, it displays a user profile page with the option to go back to the home page. It also fetches the current user's data and displays their name.

Client Side:



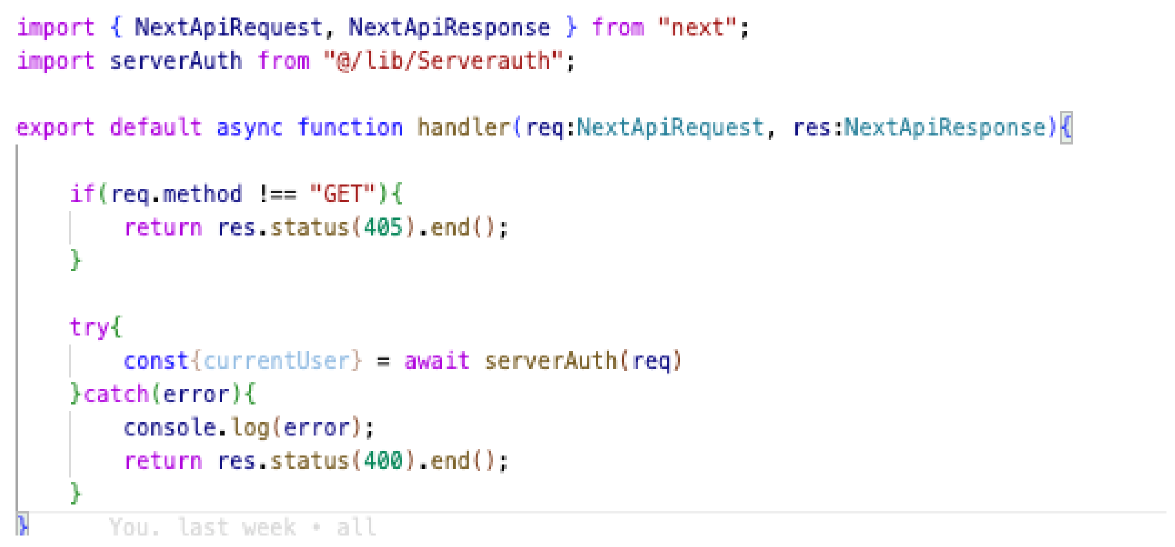
```
import {NextPageContext} from 'next';
import { useSession, getSession } from 'next-auth/react';
import useCurrentUser from '@/hooks/useCurrentUser';
import { useRouter } from 'next/navigation';

export async function getServerSideProps(context: NextPageContext){
const session = await getSession(context);

if(!session){
    return{
        redirect:{
            destination: '/auth',
            permanent: false,
        },
    }
}
return{
    props: {}
}
}

const Profiles = () => {
  const router = useRouter();
  const {data} = useSession();

  return (
    <div className="bg-black h-full">
        <nav className="px-12 py-5">
          <img
            src="./images/wura-white-bg.png"
            alt="app_logo"
            className="h-14 cursor-pointer"
          />
        </nav>

        <div className="flex items-center  justify-center mt-10">

          <div className="flex flex-col">
            <h1 className="text-3xl md:text-6xl text-white text-center">Who is Watching ?</h1>
            <div className="flex items-center justify-center gap-8 nt-10">
              <div onClick={()=> router.push("/")}>

                <div className="group flex-row w-44 mx-auto">

                  <div className="
                  mt-10
                  w-44
                  h-44
                  rounded-md
                  flex
                  items-center
```



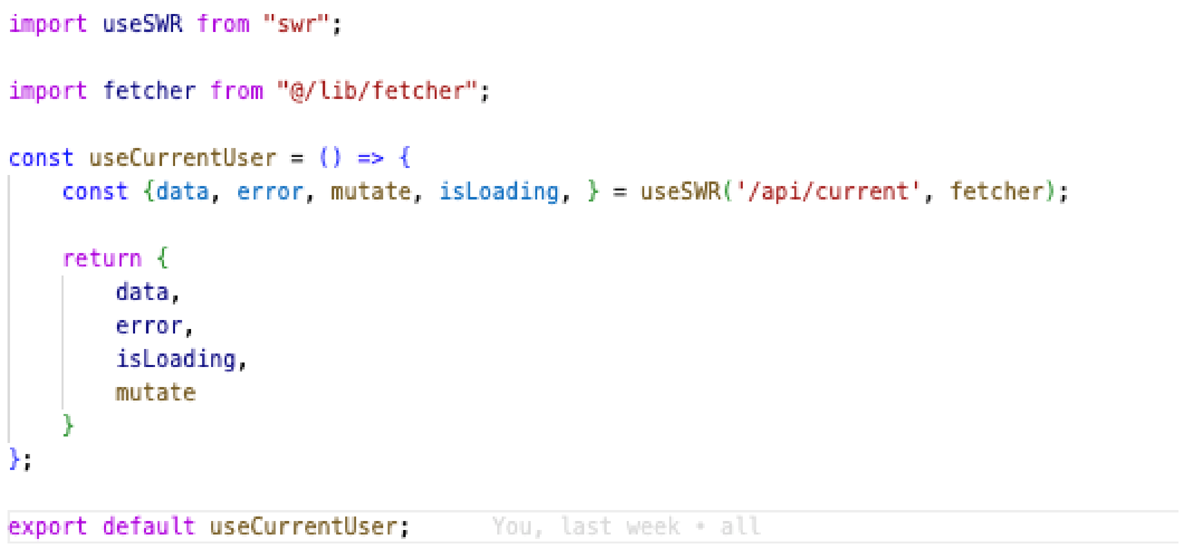
```
            <h1 className="text-3xl md:text-6xl text-white text-center">Who is Watching
            <div className="flex items-center justify-center gap-8 nt-10">
              <div onClick={()=> router.push("/")}>

                <div className="group flex-row w-44 mx-auto">

                  <div className="
                  mt-10
                  w-44
                  h-44
                  rounded-md
                  flex
                  items-center
                  justify-center
                  border-2
                  overflow-hidden
                  border-transparent
                  group-hover:cursor-pointer
                  group-hover:border-white
                  ">
                    <img className="h-15" src="./images/user.png" alt="profile" />
                  </div>
                  <div className="
                  mt-4
                  text-gray-400
                  text-2xl
                  text-center
                  group-hover:text-white
                  ">
                    {data?.user?.name}
                  </div>
                </div>
              </div>
            </div>
          </div>


        </div>
    )
}

export default Profiles;
```

Server Side: **current.ts**

The code defines an API route handler using Next.js. It only accepts GET requests; if any other method is used, it returns a 405 status code. It then attempts to authenticate the user using a server-side authentication function. If successful, it fetches the current user's data; otherwise, it logs an error and returns a 400 status code.



```
import { NextApiRequest, NextApiResponse } from "next";
import serverAuth from "@/lib/Serverauth";

export default async function handler(req:NextApiRequest, res:NextApiResponse){

    if(req.method !== "GET"){
        return res.status(405).end();
    }

    try{
        const{currentUser} = await serverAuth(req)
    }catch(error){
        console.log(error);
        return res.status(400).end();
    }
}       You, last week • all
```



```
import useSWR from "swr";

import fetcher from "@/lib/fetcher";

const useCurrentUser = () => {
    const {data, error, mutate, isLoading, } = useSWR('/api/current', fetcher);

    return {
        data,
        error,
        isLoading,
        mutate
    }
};

export default useCurrentUser;       You, last week • all
```

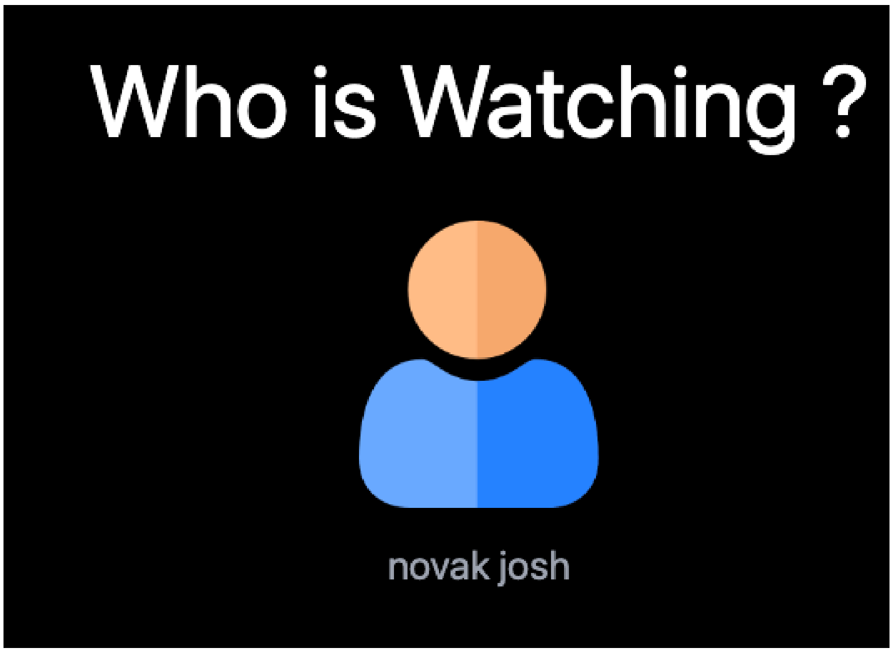
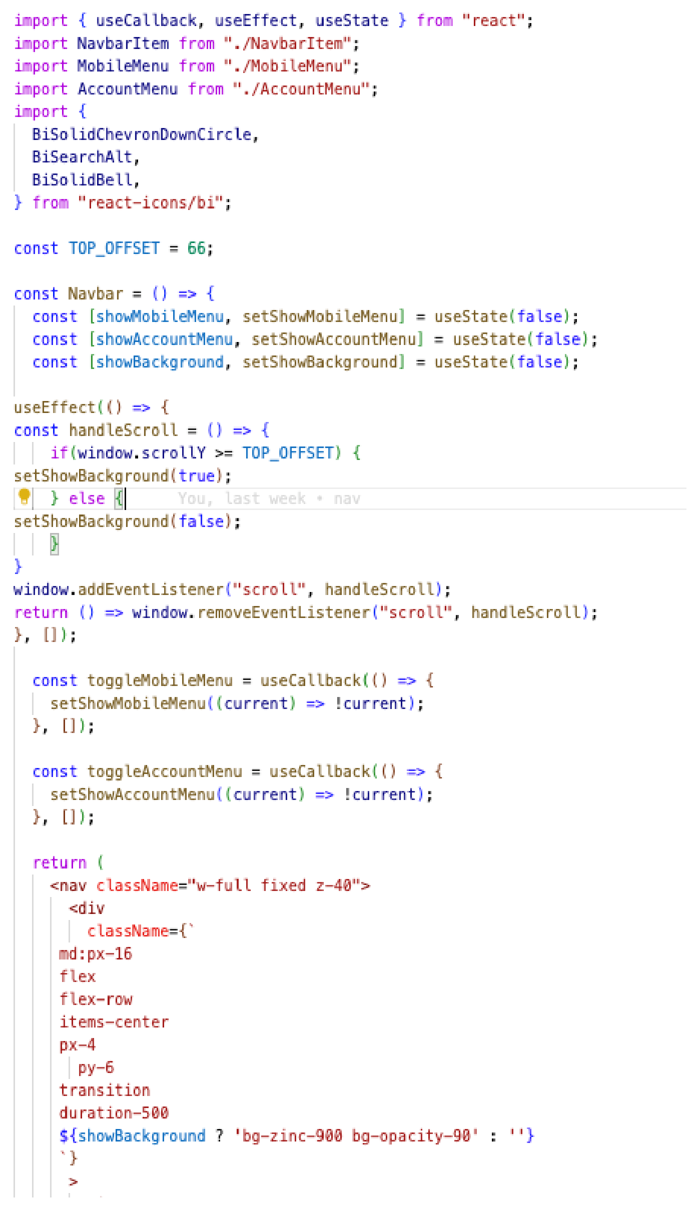


Figure 33. A logged In user Profile Screen

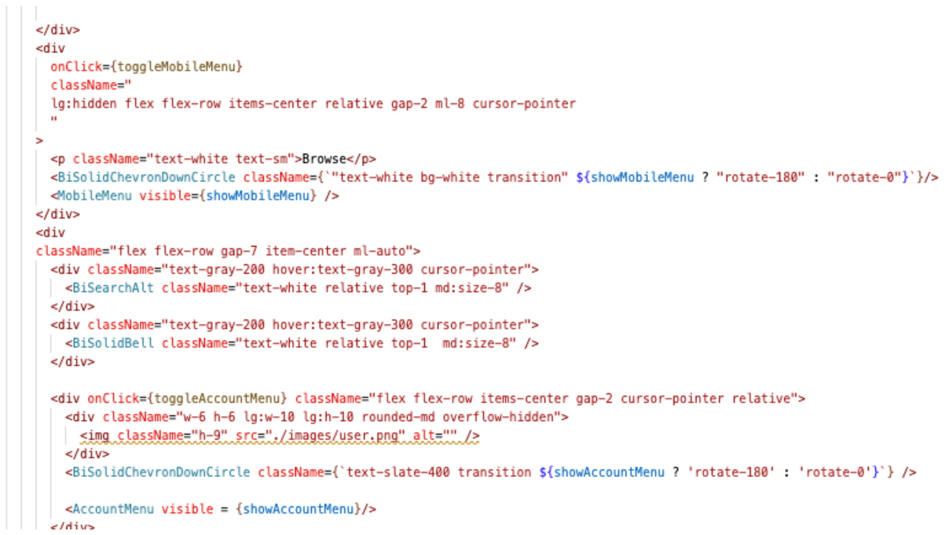
4.2.2.3 Navigation Bar Section

Client side : navbar.tsx (Navbar Component)

This code creates a **navigation bar component** for a web application. It includes features like a mobile menu, account menu, and icons for search and notifications. The background color changes when scrolling down the page. It also displays different navigation items based on screen size.



```
import { useCallback, useEffect, useState } from "react";
import NavbarItem from "./NavbarItem";
import MobileMenu from "./MobileMenu";
import AccountMenu from "./AccountMenu";
import {
  BiSolidChevronDownCircle,
  BiSearchAlt,
  BiSolidBell,
} from "react-icons/bi";

const TOP_OFFSET = 66;

const Navbar = () => {
  const [showMobileMenu, setShowMobileMenu] = useState(false);
  const [showAccountMenu, setShowAccountMenu] = useState(false);
  const [showBackground, setShowBackground] = useState(false);

useEffect(() => {
const handleScroll = () => {
    if(window.scrollY >= TOP_OFFSET) {
setShowBackground(true);
    } else {      You, last week • nav
setShowBackground(false);
    }
}
window.addEventListener("scroll", handleScroll);
return () => window.removeEventListener("scroll", handleScroll);
}, []);

  const toggleMobileMenu = useCallback(() => {
    setShowMobileMenu((current) => !current);
  }, []);

  const toggleAccountMenu = useCallback(() => {
    setShowAccountMenu((current) => !current);
  }, []);

  return (
    <nav className="w-full fixed z-40">
      <div
        className={`
     md:px-16
     flex
     flex-row
     items-center
     px-4
       py-6
     transition
     duration-500
     ${showBackground ? 'bg-zinc-900 bg-opacity-90' : ''}
     `}
       >
```



```
</div>
<div
  onClick={toggleMobileMenu}
  className="
  lg:hidden flex flex-row items-center relative gap-2 ml-8 cursor-pointer
  "
>
  <p className="text-white text-sm">Browse</p>
  <BiSolidChevronDownCircle className={`"text-white bg-white transition" ${showMobileMenu ? "rotate-180" : "rotate-0"}`}/>
  <MobileMenu visible={showMobileMenu} />
</div>
<div
className="flex flex-row gap-7 item-center ml-auto">
  <div className="text-gray-200 hover:text-gray-300 cursor-pointer">
    <BiSearchAlt className="text-white relative top-1 md:size-8" />
  </div>
  <div className="text-gray-200 hover:text-gray-300 cursor-pointer">
    <BiSolidBell className="text-white relative top-1  md:size-8" />
  </div>

  <div onClick={toggleAccountMenu} className="flex flex-row items-center gap-2 cursor-pointer relative">
    <div className="w-6 h-6 lg:w-10 lg:h-10 rounded-md overflow-hidden">
      <img className="h-9" src="./images/user.png" alt="" />
    </div>
    <BiSolidChevronDownCircle className={`text-slate-400 transition ${showAccountMenu ? 'rotate-180' : 'rotate-0'}`} />

    <AccountMenu visible = {showAccountMenu}/>
  </div>
```

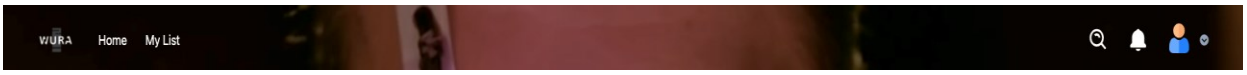
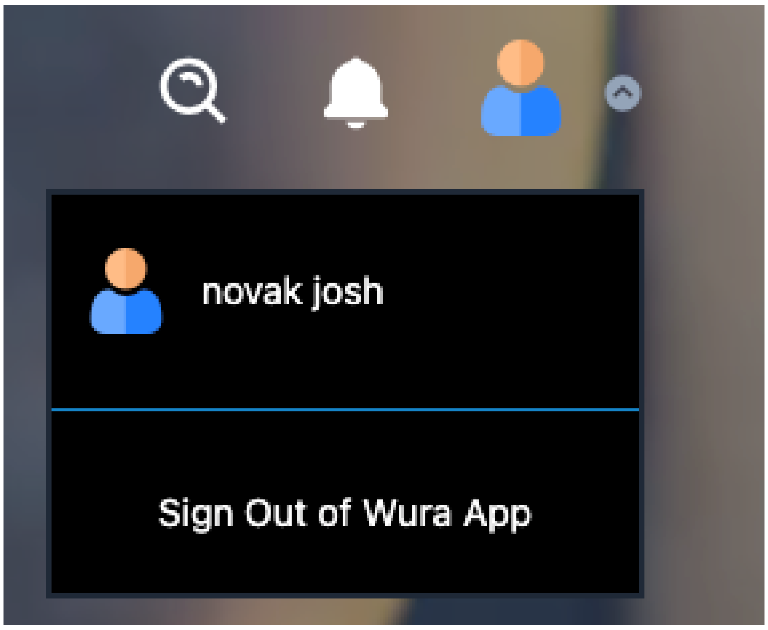


Figure 35.Navbar Item

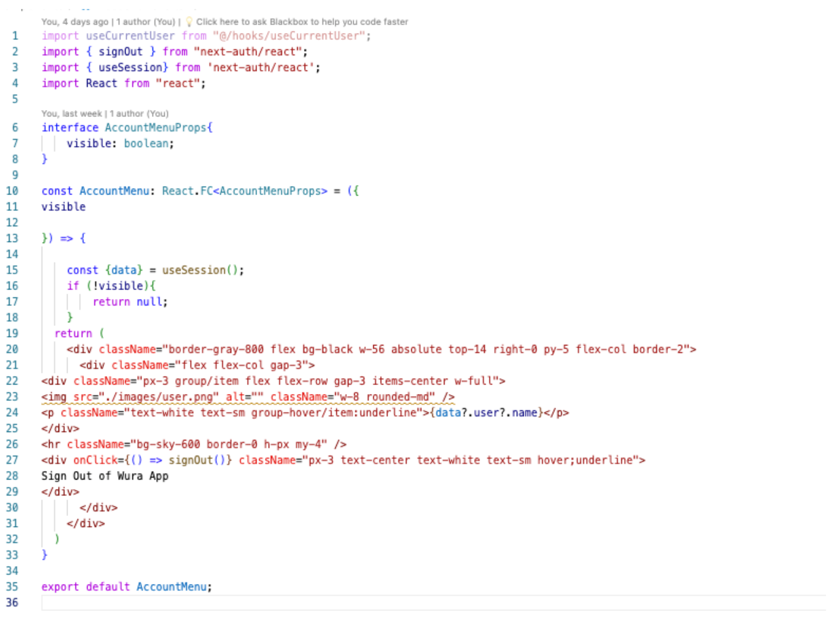
4.2.2.4 Account Menu Screen and SignOut Feature

A signedIn User on the application Main Screen



This code defines a component for the account menu in the application. It displays the user's name and provides an option to sign out. The menu is only visible when the **`visible`** prop is true.

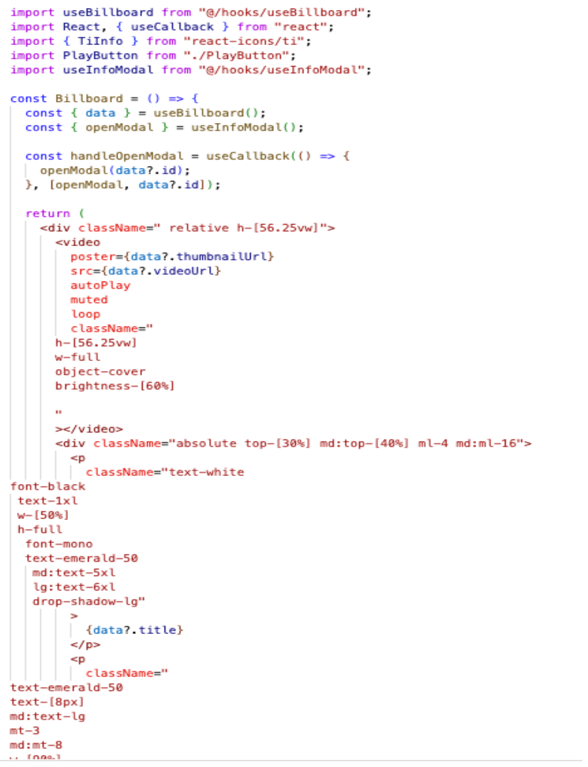
Client Side: AccountMenu.Tsx (AccountMenu Component)



```
You, 4 days ago | 1 author (You) | 💡 Click here to ask Blackbox to help you code faster
import useCurrentUser from "@/hooks/useCurrentUser";
import { signOut } from "next-auth/react";
import { useSession} from 'next-auth/react';
import React from "react";

You, last week | 1 author (You)
interface AccountMenuProps{
    visible: boolean;
}

const AccountMenu: React.FC<AccountMenuProps> = ({
visible

}) => {

    const {data} = useSession();
    if (!visible){
        return null;
    }
  return (
    <div className="border-gray-800 flex bg-black w-56 absolute top-14 right-0 py-5 flex-col border-2">
      <div className="flex flex-col gap-3">
<div className="px-3 group/item flex flex-row gap-3 items-center w-full">
<img src="./images/user.png" alt="" className="w-8 rounded-md" />
<p className="text-white text-sm group-hover/item:underline">{data?.user?.name}</p>
</div>
<hr className="bg-sky-600 border-0 h-px my-4" />
<div onClick={() => signOut()} className="px-3 text-center text-white text-sm hover;underline">
Sign Out of Wura App
</div>
      </div>
    </div>
  )
}

export default AccountMenu;

```

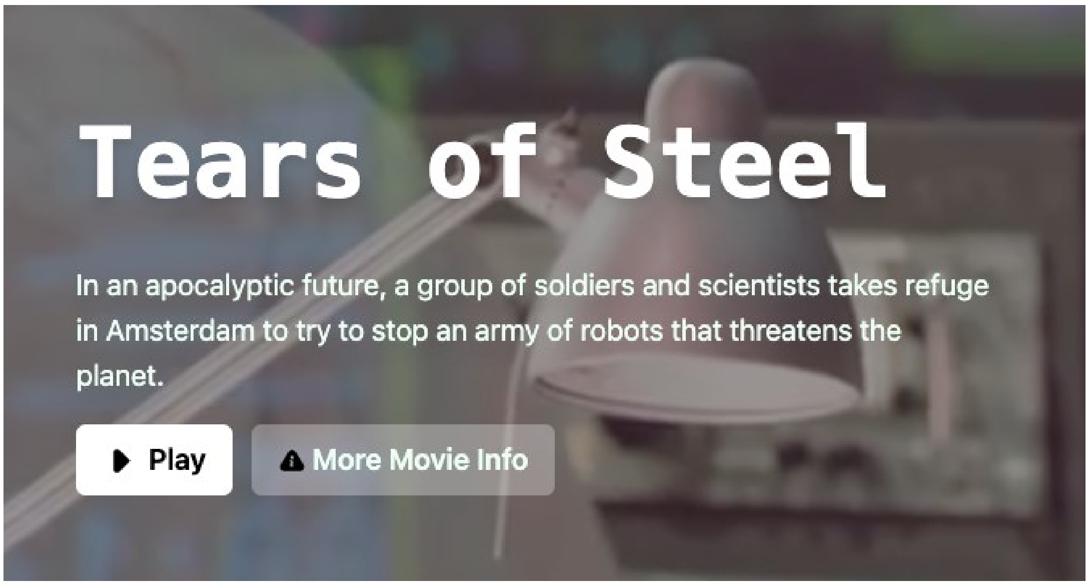
4.2.2.5 Billboard Section (Movies response from the API/DB on the HomeScreen)

Client Side: Billboard.tsx



```
import useBillboard from "@/hooks/useBillboard";
import React, { useCallback } from "react";
import { TiInfo } from "react-icons/ti";
import PlayButton from "./PlayButton";
import useInfoModal from "@/hooks/useInfoModal";

const Billboard = () => {
  const { data } = useBillboard();
  const { openModal } = useInfoModal();

  const handleOpenModal = useCallback(() => {
    openModal(data?.id);
  }, [openModal, data?.id]);

  return (
    <div className=" relative h-[56.25vw]">
      <video
        poster={data?.thumbnailUrl}
        src={data?.videoUrl}
        autoPlay
        muted
        loop
        className="
      h-[56.25vw]
      w-full
      object-cover
      brightness-[60%]

      "
      ></video>
      <div className="absolute top-[30%] md:top-[40%] ml-4 md:ml-16">
        <p
          className="text-white
font-black
 text-1xl
 w-[50%]
 h-full
  font-mono
  text-emerald-50
   md:text-5xl
   lg:text-6xl
   drop-shadow-lg"
        >
          {data?.title}
        </p>
        <p
          className="
text-emerald-50
text-[8px]
md:text-lg
mt-3
md:mt-8
```

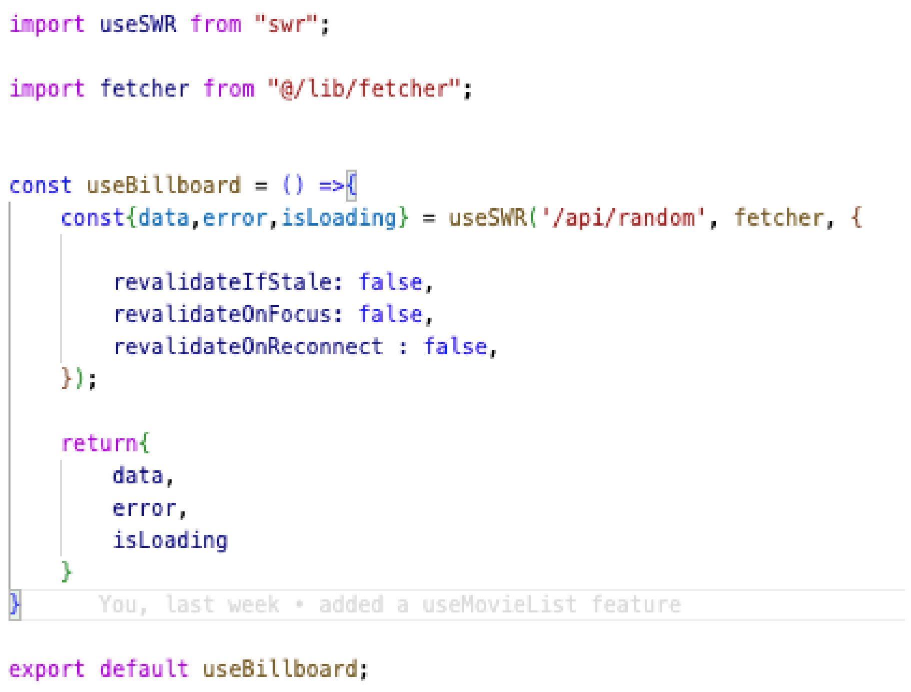
This code defines a component for a billboard section in the application. It displays a video background with dynamic content such as title, description, and buttons for playing the video and accessing more information. The video is fetched dynamically using the `useBillboard` hook, and the information modal is triggered using the `useInfoModal` hook.


Tears of Steel

In an apocalyptic future, a group of soldiers and scientists takes refuge in Amsterdam to try to stop an army of robots that threatens the planet.

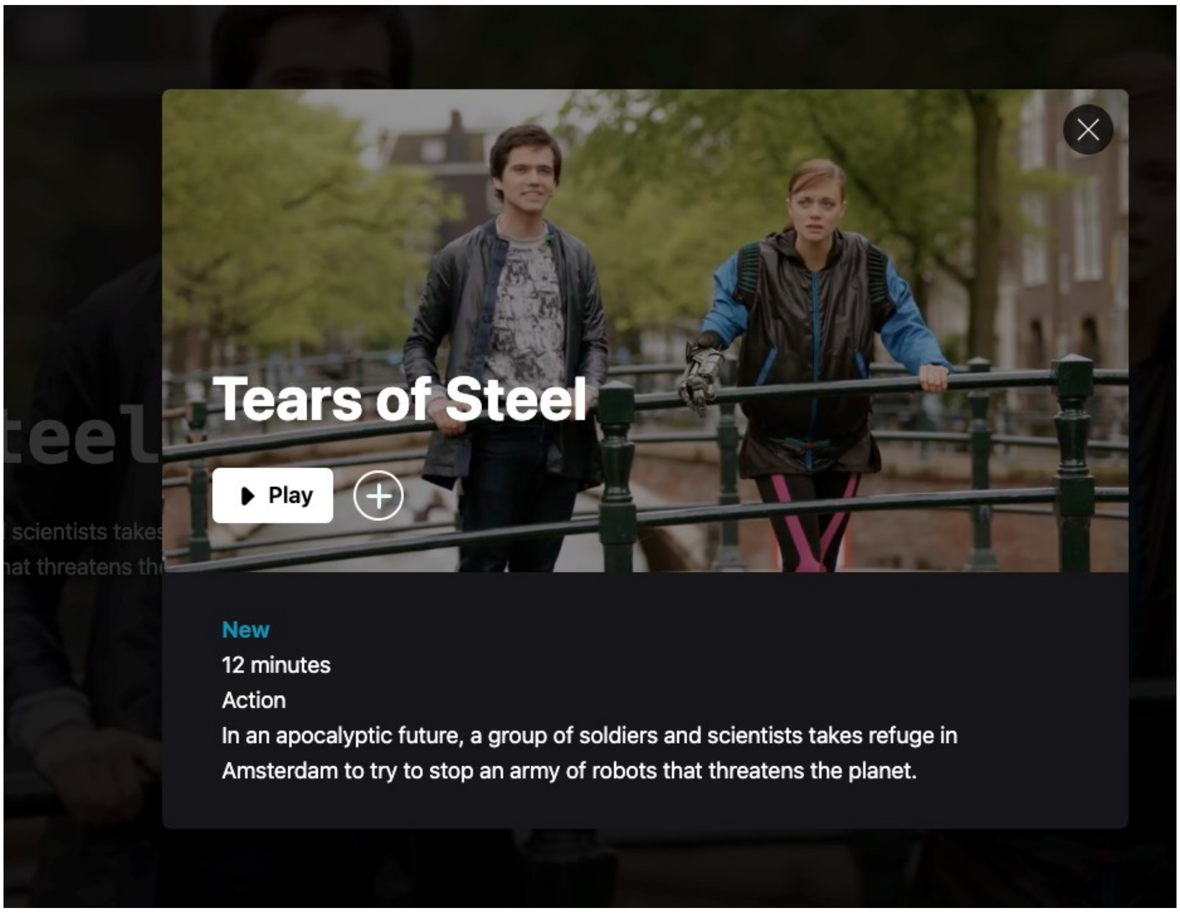
Figure 36. rendering a BillBoard on the Homescreen

Billboard hook


```
import useSWR from "swr";

import fetcher from "@/lib/fetcher";


const useBillboard = () =>{
    const{data,error,isLoading} = useSWR('/api/random', fetcher, {

        revalidateIfStale: false,
        revalidateOnFocus: false,
        revalidateOnReconnect : false,
    });

    return{
        data,
        error,
        isLoading
    }
}       You, last week • added a useMovieList feature

export default useBillboard;
```

This code defines a custom hook called `useBillboard` using SWR (Stale-While-Revalidate), a React hook for data fetching. It fetches data from the '/api/random' endpoint using the `fetcher` function defined elsewhere. The hook returns the fetched data, any errors encountered during fetching, and a boolean indicating whether the data is still loading. It also configures SWR to not automatically revalidate the data when it becomes stale, when the window regains focus, or when the connection is reestablished.

4.2.2.6 InfoModal Component (VideoPlayer) – InfoModal.tsx



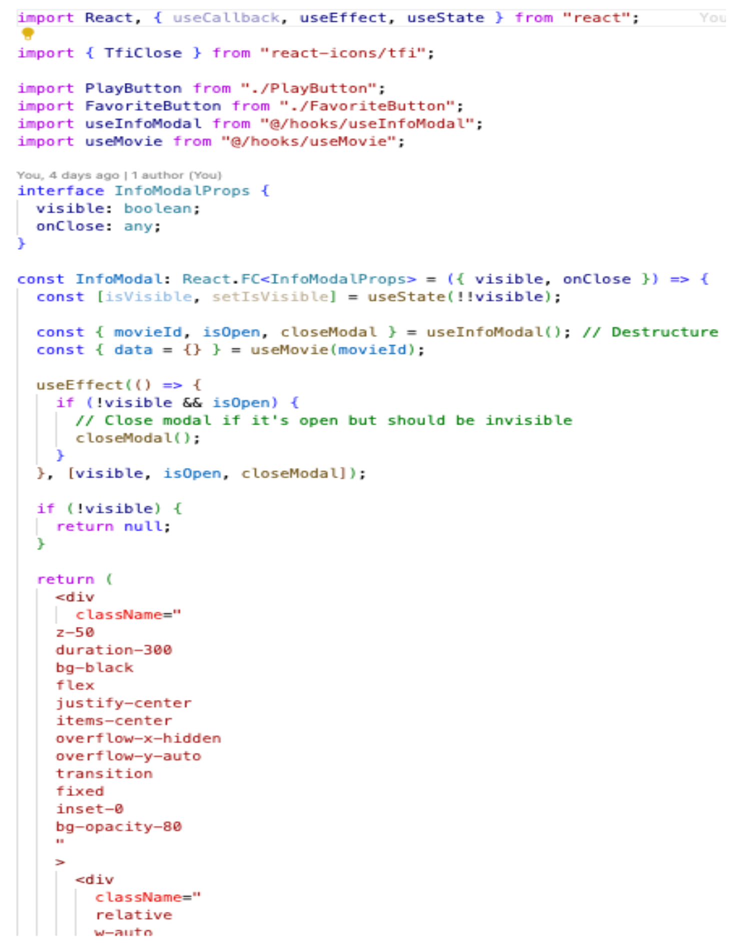
New
12 minutes
Action
In an apocalyptic future, a group of soldiers and scientists takes refuge in Amsterdam to try to stop an army of robots that threatens the planet.

Client Side :

This code defines a React component called `InfoModal` that represents a modal window displaying detailed information about a movie. It takes two props: `visible`, a boolean indicating whether the modal should be visible, and `onClose`, a function to close the modal.

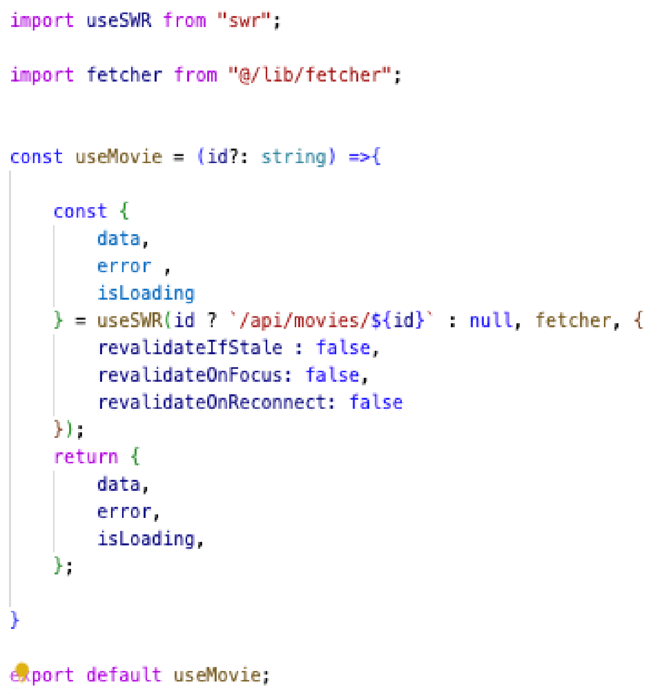
Inside the component, it manages the visibility of the modal using local state (`isVisible`). It uses custom hooks `useInfoModal` and `useMovie` to fetch and manage movie data and modal state.

The modal contains a video player with the movie's poster and plays the movie's trailer. It also displays the movie's title, duration, genre, and description. Users can play the movie, add it to favorites, and close the modal. The modal is styled with a black background and appears in the center of the screen.



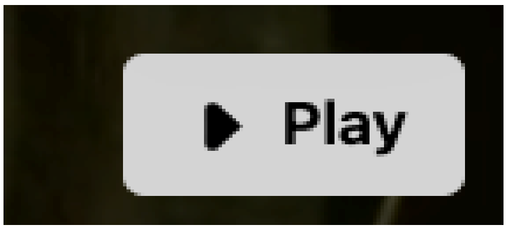
```
import React, { useCallback, useEffect, useState } from "react";        You

import { TfiClose } from "react-icons/tfi";

import PlayButton from "./PlayButton";
import FavoriteButton from "./FavoriteButton";
import useInfoModal from "@/hooks/useInfoModal";
import useMovie from "@/hooks/useMovie";

You, 4 days ago | 1 author (You)
interface InfoModalProps {
  visible: boolean;
  onClose: any;
}

const InfoModal: React.FC<InfoModalProps> = ({ visible, onClose }) => {
  const [isVisible, setIsVisible] = useState(!!visible);

  const { movieId, isOpen, closeModal } = useInfoModal(); // Destructure
  const { data = {} } = useMovie(movieId);

  useEffect(() => {
    if (!visible && isOpen) {
      // Close modal if it's open but should be invisible
      closeModal();
    }
  }, [visible, isOpen, closeModal]);

  if (!visible) {
    return null;
  }

  return (
    <div
      className="
    z-50
    duration-300
    bg-black
    flex
    justify-center
    items-center
    overflow-x-hidden
    overflow-y-auto
    transition
    fixed
    inset-0
    bg-opacity-80
    "
    >
      <div
        className="
        relative
        w-auto
```

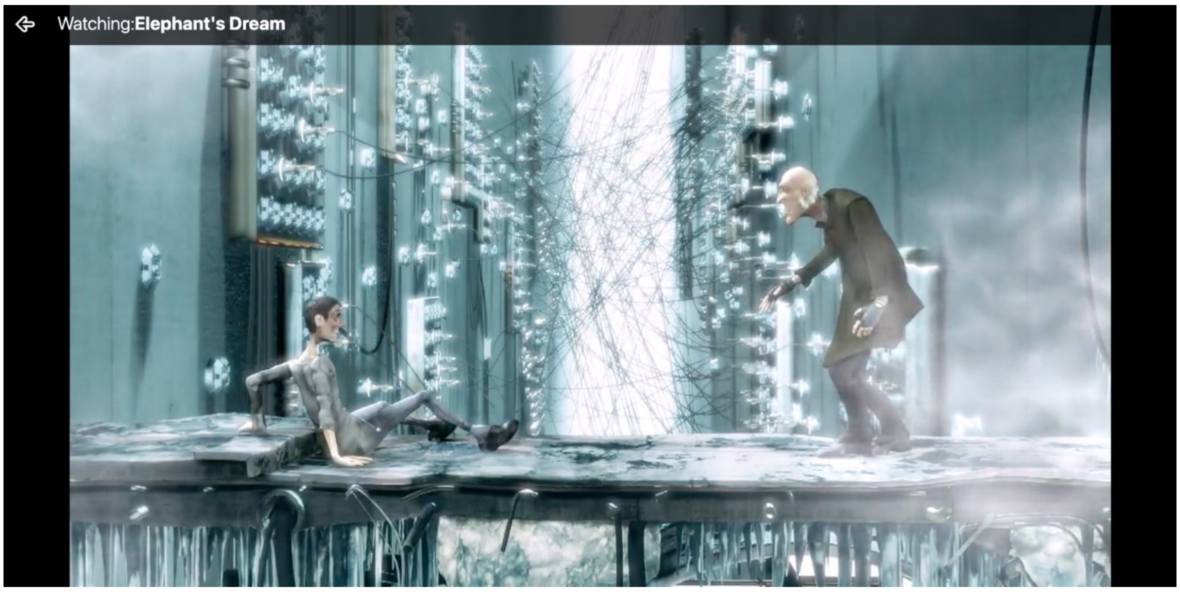
UseMovie Hooks to manage the movie data when rendering the information in the modal.



```
import useSWR from "swr";

import fetcher from "@/lib/fetcher";


const useMovie = (id?: string) =>{

    const {
        data,
        error ,
        isLoading
    } = useSWR(id ? `/api/movies/${id}` : null, fetcher, {
        revalidateIfStale : false,
        revalidateOnFocus: false,
        revalidateOnReconnect: false
    });
    return {
        data,
        error,
        isLoading,
    };

}

export default useMovie;
```

4.2.2.7 PlayButton Feature – (PlayButton Component)

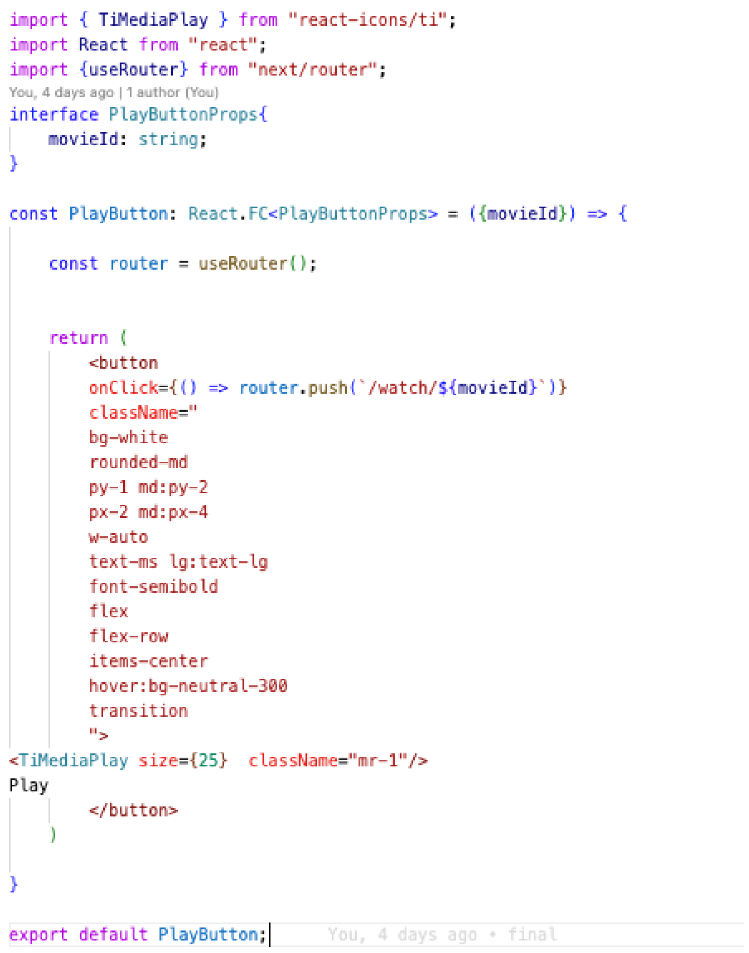




This code defines a React component called `**PlayButton`,** which represents a button used to play a movie. It takes a single prop `movieId`, which is a string representing the ID of the movie to be played.

Inside the component, it utilizes the `**useRouter`** hook from Next.js to access the router object. When the button is clicked, it triggers a navigation to the '/watch/[movieId]' route, where `[movieId]` is replaced by the actual ID of the movie. This allows users to navigate to the page dedicated to watching the specific movie.

The button is styled with a white background, rounded corners, and text indicating "Play". It also includes a play icon from the TiMediaPlay component. When hovered over, the background color changes to a neutral shade to provide visual feedback.



```
import { TiMediaPlay } from "react-icons/ti";
import React from "react";
import {useRouter} from "next/router";
You, 4 days ago | 1 author (You)
interface PlayButtonProps{
    movieId: string;
}

const PlayButton: React.FC<PlayButtonProps> = ({movieId}) => {

    const router = useRouter();


    return (
        <button
        onClick={() => router.push(`/watch/${movieId}`)}
        className="
        bg-white
        rounded-md
        py-1 md:py-2
        px-2 md:px-4
        w-auto
        text-ms lg:text-lg
        font-semibold
        flex
        flex-row
        items-center
        hover:bg-neutral-300
        transition
        ">
<TiMediaPlay size={25}  className="mr-1"/>
Play
        </button>
    )

}

export default PlayButton;      You, 4 days ago • final
```

4.2.3 Database Schema Model (Prisma)

```
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
}
model User{
  id String @id @default(auto()) @map("_id") @db.ObjectId
  name String
  image String?
  email String? @unique
  emailVerified DateTime?
  hashedPassword String?
  createAt DateTime @default(now())
  updateAt DateTime @updatedAt
  favoriteIds String[] @db.ObjectId
  sessions Session[]
  accounts Account[]
}
```

```
model Account{
   id String @id @default(auto()) @map("_id") @db.ObjectId
   userId String @db.ObjectId
   type String
   provider String
   providerAccountId String
   refresh_token String? @db.String
   access_token String?@db.String
   expires_at Int?
   token_type String?
   scope String?
   id_token String? @db.String
   session_state String?
   user User @relation(fields: [userId], references: [id], onDelete: Cascade)
@@unique([provider, providerAccountId])

}
model Session{
  id String @id @default(auto()) @map("_id") @db.ObjectId
  sessionToken String @unique
  userId String @db.ObjectId
  expires DateTime
user User @relation(fields: [userId], references: [id], onDelete: Cascade)
}
model VerificationToken{
  id String @id @default(auto()) @map("_id") @db.ObjectId
  Identifier String
  token String @unique
  expires DateTime

  @@unique([Identifier, token])
}
model Movie{
  id String @id @default(auto()) @map("_id") @db.ObjectId
  title String
  description String
  videoUrl String
  thumbnailUrl String
  genre String
  duration String
}
```

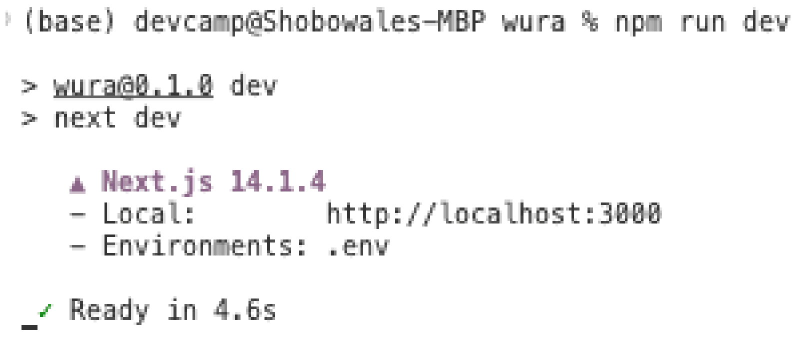
4.2.4 DataBase Url, Google and Github Auth Token (Confidential Information)

```
DATABASE_URL= "mongodb+srv://Tolani:Informativewuraolami123@cluster0.58aqsfh.mongodb.net/test"

NEXTAUTH_JWT_SECRET="NEXT-JWT-SECRET"
NEXTAUTH_SECRET="NEXT-SECRET"
# Google and Github Oauth sign on providers for the app
GITHUB_ID = e8d4b74a095687a71cfd
```

GITHUB_SECRET = 4d7776754d5328fe5f54274856382be22b4064e3

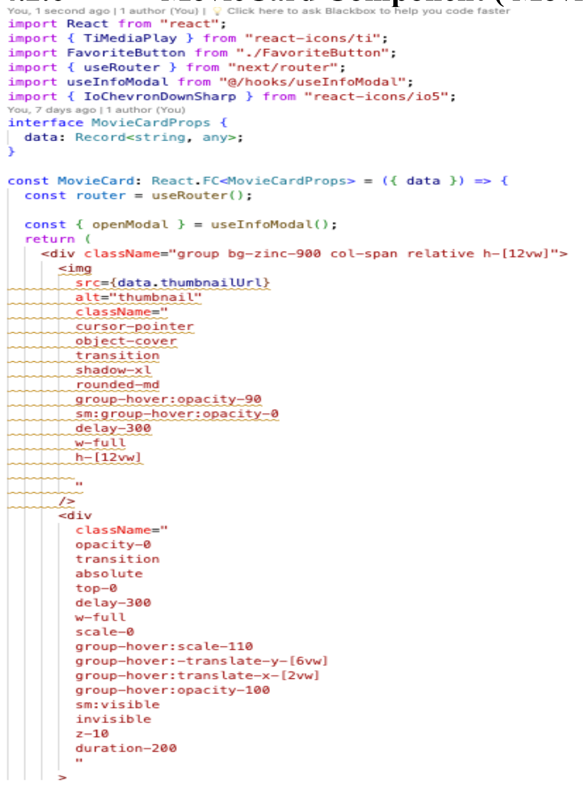
GOOGLE_CLIENT_ID = 1098359435879-llg04rqr2p4uks9at1iudkte4tglqft2.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET = 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

4.2.5 Running the Application using Npm (.env file)

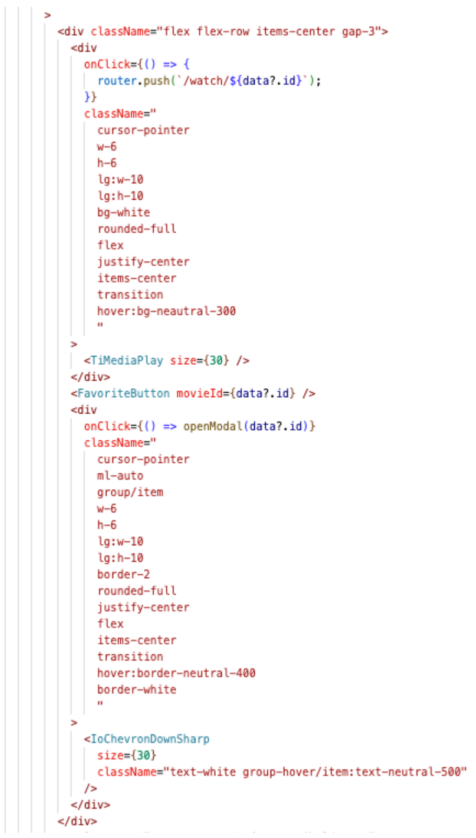


```
(base) devcamp@Shobowales-MBP wura % npm run dev

> wura@0.1.0 dev
> next dev

   ▲ Next.js 14.1.4
   - Local:        http://localhost:3000
   - Environments: .env

 ✓ Ready in 4.6s
```

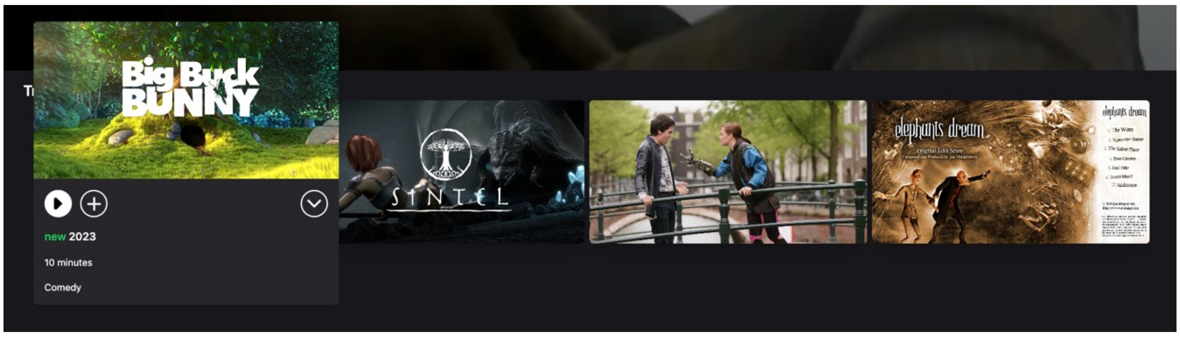
4.2.6 MovieCard Component (MovieCard.tsx)



```
You, 1 second ago | 1 author (You) | 💡 Click here to ask Blackbox to help you code faster
import React from "react";
import { TiMediaPlay } from "react-icons/ti";
import FavoriteButton from "./FavoriteButton";
import { useRouter } from "next/router";
import useInfoModal from "@/hooks/useInfoModal";
import { IoChevronDownSharp } from "react-icons/io5";
You, 7 days ago | 1 author (You)
interface MovieCardProps {
  data: Record<string, any>;
}

const MovieCard: React.FC<MovieCardProps> = ({ data }) => {
  const router = useRouter();

  const { openModal } = useInfoModal();
  return (
    <div className="group bg-zinc-900 col-span relative h-[12vw]">
      <img
        src={data.thumbnailUrl}
        alt="thumbnail"
        className="
        cursor-pointer
        object-cover
        transition
        shadow-xl
        rounded-md
        group-hover:opacity-90
        sm:group-hover:opacity-0
        delay-300
        w-full
        h-[12vw]

        "
      />
      <div
        className="
        opacity-0
        transition
        absolute
        top-0
        delay-300
        w-full
        scale-0
        group-hover:scale-110
        group-hover:-translate-y-[6vw]
        group-hover:translate-x-[2vw]
        group-hover:opacity-100
        sm:visible
        invisible
        z-10
        duration-200
        "
      >
```



```
>
  <div className="flex flex-row items-center gap-3">
    <div
      onClick={() => {
        router.push(`/watch/${data?.id}`);
      }}
      className="
        cursor-pointer
        w-6
        h-6
        lg:w-10
        lg:h-10
        bg-white
        rounded-full
        flex
        justify-center
        items-center
        transition
        hover:bg-neautral-300
        "
    >
      <TiMediaPlay size={30} />
    </div>
    <FavoriteButton movieId={data?.id} />
    <div
      onClick={() => openModal(data?.id)}
      className="
        cursor-pointer
        ml-auto
        group/item
        w-6
        h-6
        lg:w-10
        lg:h-10
        border-2
        rounded-full
        justify-center
        flex
        items-center
        transition
        hover:border-neutral-400
        border-white
        "
    >
      <IoChevronDownSharp
        size={30}
        className="text-white group-hover/item:text-neutral-500"
      />
    </div>
  </div>
```



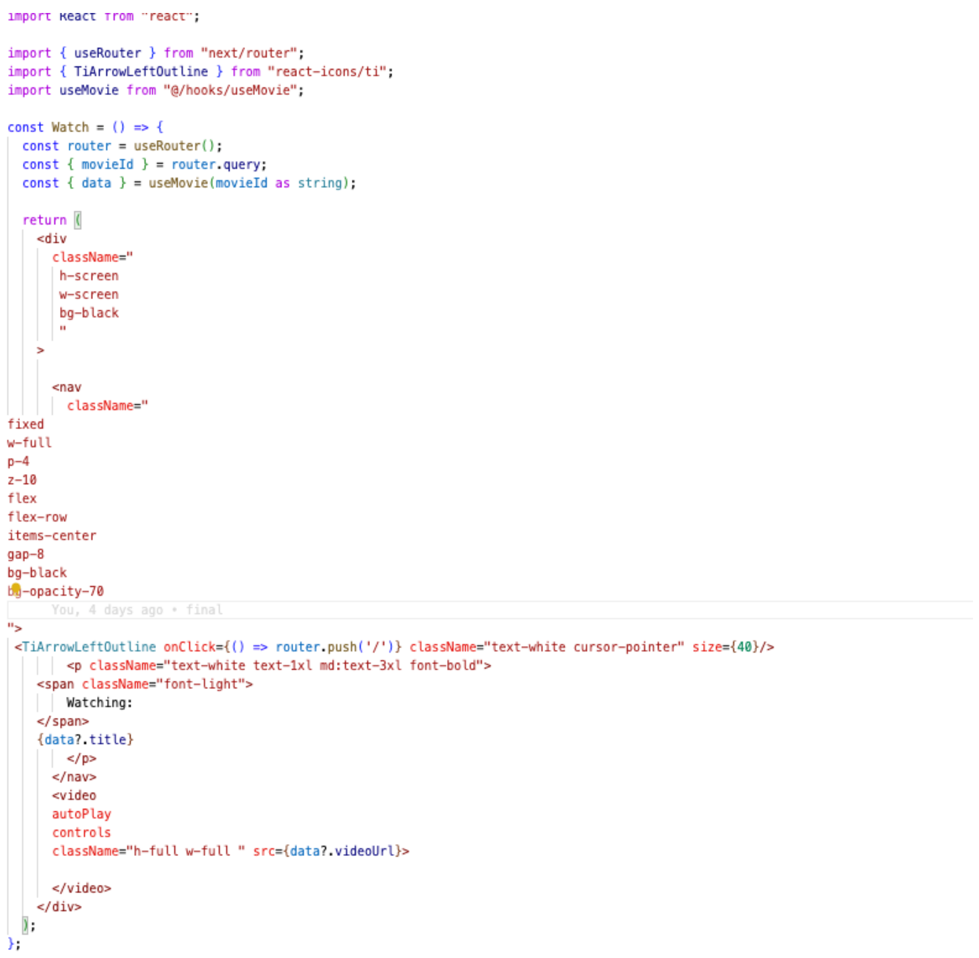
This code defines a React component called `MovieCard`, representing a card displaying information about a movie. It takes a prop `data`, which contains details of the movie.

Inside the component, it renders an image of the movie's thumbnail, and when hovered over, it displays additional information such as a play button, a favorite button, and an icon to open a modal with more details about the movie.

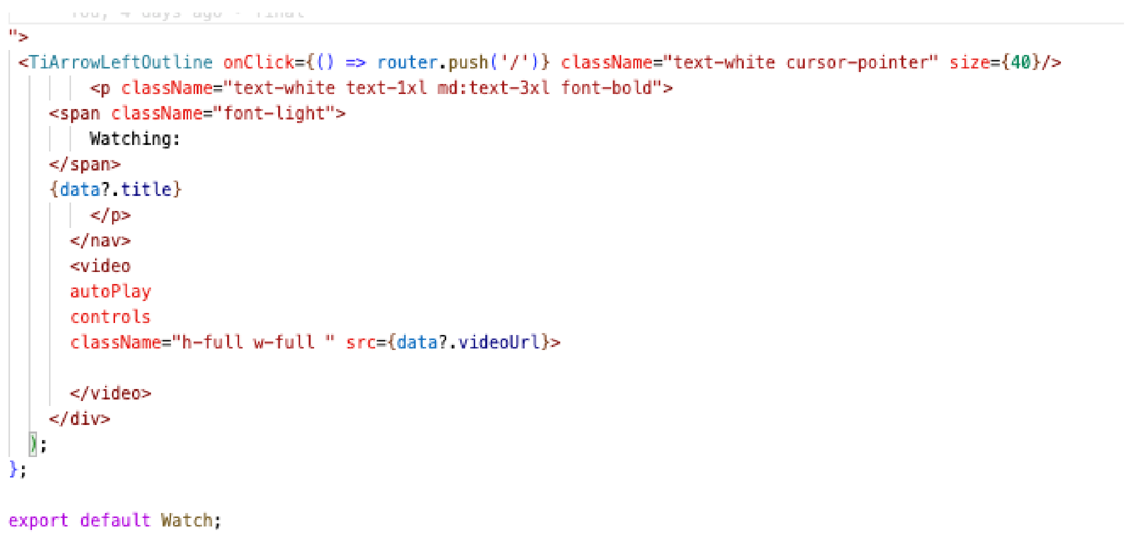
The play button allows users to navigate to the watch page of the movie, the favorite button lets users add the movie to their favorites, and the icon triggers the modal displaying more information about the movie.

Overall, it provides a visually appealing way to showcase movie details and interact with them.

4.2.7 Watch Current Played Movie ([movieid-tsx])



```
import React from "react";

import { useRouter } from "next/router";
import { TiArrowLeftOutline } from "react-icons/ti";
import useMovie from "@/hooks/useMovie";

const Watch = () => {
  const router = useRouter();
  const { movieId } = router.query;
  const { data } = useMovie(movieId as string);

  return (
    <div
      className="
       h-screen
       w-screen
       bg-black
       "
    >

      <nav
        className="
fixed
w-full
p-4
z-10
flex
flex-row
items-center
gap-8
bg-black
bg-opacity-70
      You, 4 days ago • final
">
 <TiArrowLeftOutline onClick={() => router.push('/')} className="text-white cursor-pointer" size={40}/>
        <p className="text-white text-1xl md:text-3xl font-bold">
    <span className="font-light">
        Watching:
    </span>
    {data?.title}
        </p>
      </nav>
      <video
      autoPlay
      controls
      className="h-full w-full " src={data?.videoUrl}>

      </video>
    </div>
  );
};
```



```
">
 <TiArrowLeftOutline onClick={() => router.push('/')} className="text-white cursor-pointer" size={40}/>
        <p className="text-white text-1xl md:text-3xl font-bold">
    <span className="font-light">
        Watching:
    </span>
    {data?.title}
        </p>
      </nav>
      <video
      autoPlay
      controls
      className="h-full w-full " src={data?.videoUrl}>

      </video>
    </div>
  );
};
```

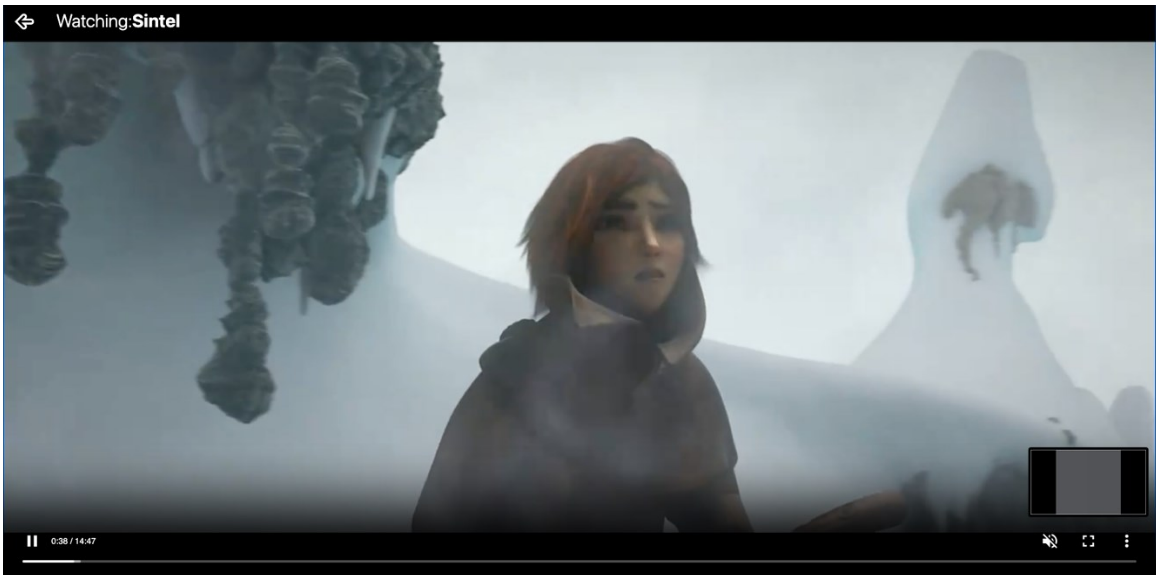
```
export default Watch;
```

The code defines a React component called `Watch`, representing a page for watching movies. It imports necessary modules such as React, `useRouter` from Next.js for routing, and `useMovie` for fetching movie data.

Inside the component, it fetches the movie ID from the router query parameters and uses it to fetch movie data. It then renders a fullscreen video player with controls, playing the movie corresponding to the fetched data.

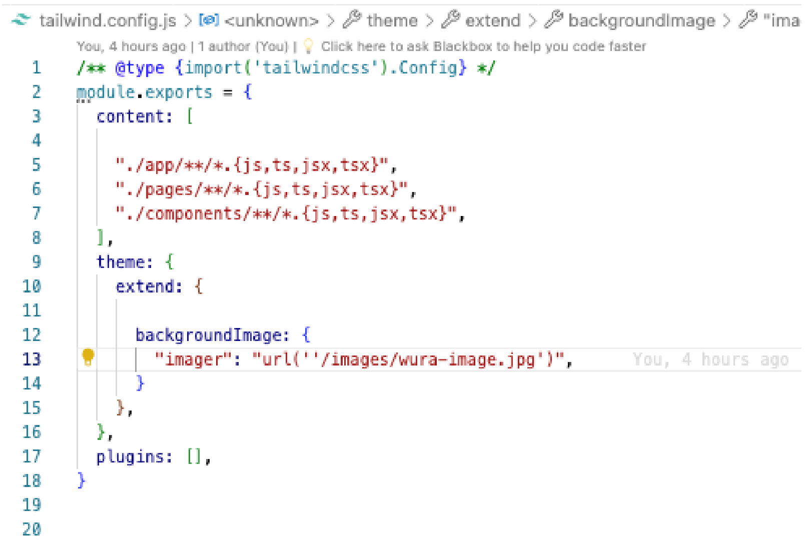
Additionally, it includes a navigation bar at the top of the screen with a back arrow icon that navigates back to the homepage when clicked. The navigation bar also displays the title of the movie currently being watched.

Overall, the component provides a simple and functional interface for watching movies with basic playback controls.



4.2.8 Tailwind WorkFlow

```
@tailwind base;
@tailwind components;
@tailwind utilities;

body{
  @apply bg-zinc-900 h-full overflow-x-hidden;
}

#__next{
  @apply h-full;
}

html{
  @apply h-full;
}
```



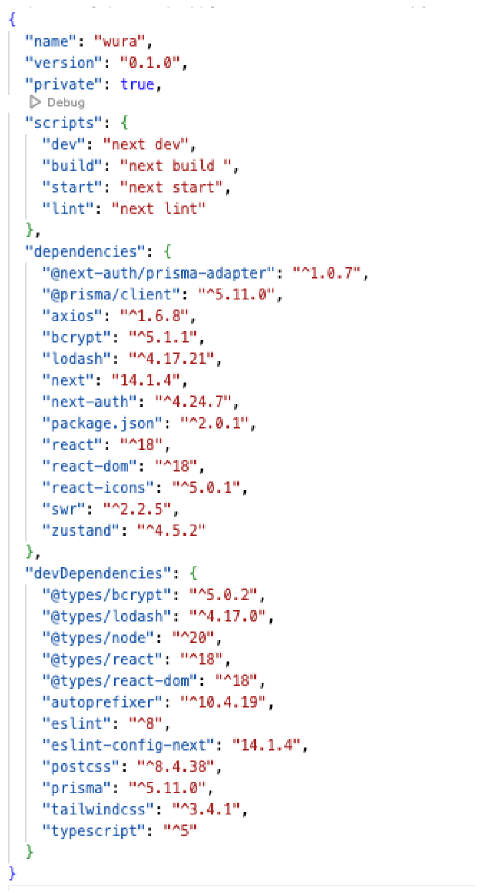
tailwind.config.js > <unknown> > theme > extend > backgroundImage > "ima

You, 4 hours ago | 1 author (You) | Click here to ask Blackbox to help you code faster

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [

    "./app/**/*.{js,ts,jsx,tsx}",
    "./pages/**/*.{js,ts,jsx,tsx}",
    "./components/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {

      backgroundImage: {
        "imager": "url(''/images/wura-image.jpg')",      You, 4 hours ago
      }
    },
  },
  plugins: [],
}


```

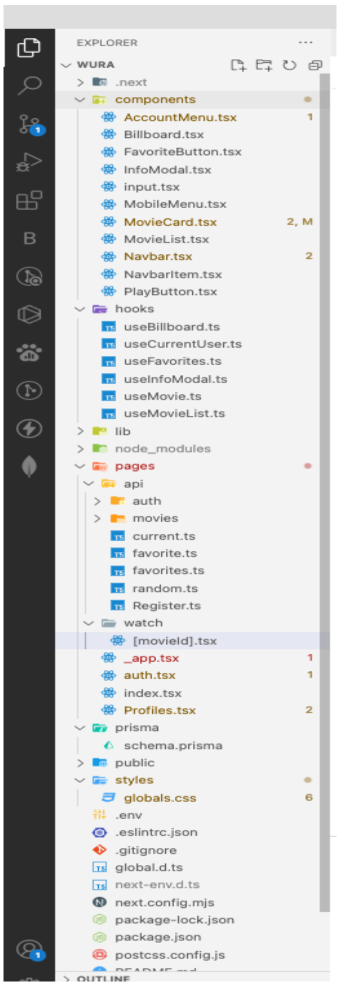
Figure 37. Tailwind.config file

4.2.9 Application package.Json file



```
{
  "name": "wura",
  "version": "0.1.0",
  "private": true,
  ▷ Debug
  "scripts": {
    "dev": "next dev",
    "build": "next build ",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "@next-auth/prisma-adapter": "^1.0.7",
    "@prisma/client": "^5.11.0",
    "axios": "^1.6.8",
    "bcrypt": "^5.1.1",
    "lodash": "^4.17.21",
    "next": "14.1.4",
    "next-auth": "^4.24.7",
    "package.json": "^2.0.1",
    "react": "^18",
    "react-dom": "^18",
    "react-icons": "^5.0.1",
    "swr": "^2.2.5",
    "zustand": "^4.5.2"
  },
  "devDependencies": {
    "@types/bcrypt": "^5.0.2",
    "@types/lodash": "^4.17.0",
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "autoprefixer": "^10.4.19",
    "eslint": "^8",
    "eslint-config-next": "14.1.4",
    "postcss": "^8.4.38",
    "prisma": "^5.11.0",
    "tailwindcss": "^3.4.1",
    "typescript": "^5"
  }
}
```

4.3 My Development Workflow



```
EXPLORER                              ...
WURA
  > .next
  v components
      AccountMenu.tsx                   1
      Billboard.tsx
      FavoriteButton.tsx
      InfoModal.tsx
      input.tsx
      MobileMenu.tsx
      MovieCard.tsx                  2, M
      MovieList.tsx
      Navbar.tsx                        2
      NavbarItem.tsx
      PlayButton.tsx
  v hooks
      useBillboard.ts
      useCurrentUser.ts
      useFavorites.ts
      useInfoModal.ts
      useMovie.ts
      useMovieList.ts
  > lib
  > node_modules
  v pages
    v api
      > auth
      > movies
        current.ts
        favorite.ts
        favorites.ts
        random.ts
        Register.ts
    v watch
        [movieId].tsx
      _app.tsx                          1
      auth.tsx                          1
      index.tsx
      Profiles.tsx                      2
  v prisma
      schema.prisma
  > public
  v styles
      globals.css                       6
    .env
    .eslintrc.json
    .gitignore
    global.d.ts
    next-env.d.ts
    next.config.mjs
    package-lock.json
    package.json
    postcss.config.js
> OUTLINE
```

5 DISCUSSION AND FINDINGS

5.1 DISCUSSION

5.1.1 Guideline 1: **Understand the programming language JavaScript, its structure, frameworks and production usage**

In the first chapter, we explored the history of LiveScript and its transformation into ECMAScript, which eventually became JavaScript. We outlined key features that have shaped the language, such as JSON, event handling, and prototyping. These elements have had a significant impact on how JavaScript is used in modern software development.

5.1.2 Guideline 2: **Take an insight into previous related works**

In the second chapter, we focused on software development frameworks, particularly the Model-View-Controller (MVC) architectural pattern. This chapter emphasized the importance of MVC in building web applications and managing data flow. Since JavaScript relies heavily on client-server communication, this architecture is especially relevant. The second chapter provided foundational insights into structural patterns for creating software applications. We also cited some works related to repository mining by looking at the BOA architecture.

5.1.3 Guidelines 3: **Analyze the Management of JavaScript Codes and data using Node Package Manager.**

The third chapter, divided into two parts. **In part A**, I examined the three most popular JavaScript frameworks, highlighting their importance and unique features. We discussed how these frameworks contribute to the broader JavaScript ecosystem, referencing authoritative sources like Stack Overflow, NPM Trends, GitHub, and various developer communities.

In Part B, To address the key questions of this research, we conducted a qualitative survey to understand why software engineers choose certain JavaScript frameworks or libraries when developing web applications. Additionally, we looked into the limitations that JavaScript-based applications face in today's context. The survey provided valuable insights into the factors influencing business decisions and the constraints developers encounter.

The comprehensive analysis in part B in Chapter 3 sheds more light on the factors driving framework selection and the common limitations encountered in JavaScript development. These findings play a crucial role in answering the research objectives and are discussed in detail to provide a complete understanding of the modern JavaScript landscape..

The findings suggest that the choice of framework depended on the type of project. Developers tend to prefer frameworks that are reliable and can be maintained over a long period, as many projects have extended lifespans. More established frameworks like Angular and React are generally favored over newer ones like Vue, which may not yet have the same level of maturity and support.

The reputation of a framework often shapes developers' choices, influenced by factors like the company backing it. Frameworks supported by large companies such as Google or Facebook tend to be seen as more reliable, partly because these companies have the resources to ensure continuous development and longevity. Additionally, frameworks that have been around longer often feel more established, contributing to their perceived stability.

However, developers might choose different frameworks depending on specific project requirements. While many tend to favor ReactJS as their default option, others might have a personal preference for a lesser-known framework, like VueJS as one respondent mentioned that it has been the only framework he has worked with which I find quite fascinating, based on their past experiences. Ultimately, the choice of framework is largely driven by the unique needs of each project.

Selecting a framework with a vibrant community was also a crucial factor as mentioned by majority of the respondents as it contributes to the framework's vitality through user-generated contributions. A larger user base generally means more resources for solving problems, as community members are more active on platforms like Discord, Google, Stack Overflow, making it easier for developers to find answers to issues.

5.1.4 Guideline 4: Build a Movie-app application using numerous APIs e.g., Netflix, ReactJs, NPM, GitHub, MongoDB for the database.

In Chapter 4, I embarked on the development of the Wura App, The project was a React/Next.js application, utilizing a variety of development tools and technologies. MongoDB was used for the database, while Git/GitHub served as our version control system. Node.js and NPM played a crucial role, especially given our reliance on numerous open-source React dependencies. NPM was particularly useful for bundling our application, which helped to optimize runtime memory usage.

For server-side operations, we employed Prisma, a powerful tool for interacting with databases. User authentication was implemented with the help of Google and GitHub OAuth services, providing a secure and convenient login experience for our users. The chapter detailed the entire build process, outlining each step from setting up the development environment to deploying the final product.

5.2 FINDINGS

One challenge we encountered was related to package security. Due to the open-source nature of many NPM packages which were old, not properly managed , there were potential vulnerabilities that required careful attention. These Vulnerability issues was addressed by updating NPM and Nodejs to a more stable version, constantly running “npm audit fix” to auto fix a lot of vulnerabilities, ensuring the safety and stability of our application.

The creation of the application revealed several common areas for improvement in NPM and JavaScript, as also noted by respondents in the survey. The primary concerns centered around the quantity and duplication of downloaded packages which was further seen as limitations, highlighting the need for better management of package dependencies. Additionally, security emerged as a significant concern, indicating a desire for enhanced measures to protect against vulnerabilities, hackers e.t.c.

Lastly, more emphasizes should be centered around the importance of standardizing open source dependencies in JavaScript to ensure consistency and efficiency across entire JavaScript ecosystem.

6 CONCLUSION

Overall, I found the thesis process to be relatively smooth, encountering only minor obstacles along the way. Despite having to balance my time between working as a Data Analyst and dedicating time to the thesis, I successfully completed the project's most crucial aspects. Working concurrently proved beneficial, as my professional experience with NPM and JavaScript programming informed my decision to explore this topic further. However, sourcing literature on the subject was challenging, leading me to rely heavily on online resources. While references to package managers were scarce in traditional literature, the supportive developer community aided in troubleshooting during the application's development. If I were to revisit this topic, I would prioritize expanding the survey's sample size. Acquiring respondents posed difficulties initially, highlighting an area for improvement in future research endeavors.

Overall, the research and writing skills acquired during this thesis will undoubtedly serve me well in my future career. These skills may prove particularly valuable should I pursue further my studies.

The data I collected revealed the significant importance of ReactJS and NPM in the realm of software development. A vast majority of respondents reported using a package manager and having experience building applications with React. Among the popular JavaScript frameworks, ReactJS emerged as the clear frontrunner in terms of usage and awareness. This trend is understandable given the abundance of job opportunities for React developers, as indicated by statistics from job platforms like Indeed and LinkedIn. Additionally, ReactJS benefits from the support of META (formerly Facebook), contributing to its widespread popularity within the developer community.

6.1 Overall Performance Analysis of the Wura Application

6.1.1 User Authentication and Account Management

The app provided essential user authentication with sign-up and login screens. New users can create accounts by providing basic information like username, email, and password. Returning users can log in securely and easily.

Besides traditional sign-up, the app supports social authentication through Google Auth and GitHub Auth, allowing users to sign up with existing accounts. This streamlines the sign-up process and improves user acquisition by reducing the need for new credentials.

6.1.2 User Interface and Experience

After logging in, users can access their profile, which displays personal information like name and account details, offering a personalized touch. The HomeScreen is the main hub, featuring a play button to start movies, an info button for detailed movie information, and a modal screen for additional content. These interactive features make it easy to explore and watch movies.

The HomeScreen also included a SignOut button for secure logout, crucial for account safety, especially in shared spaces. The UserInfo section displayed the registered user's name, reinforcing a sense of personalization. A trending movie section introduces popular content, encouraging users to discover new movies and keeping the experience fresh.

6.1.3 Responsiveness and Cross-Platform Compatibility

The app is responsive, ensuring smooth functionality on mobile devices. This is crucial in today's mobile-focused world, allowing users to access the app on various platforms without compromising the user experience.

6.2 Guidelines for Future Research

The software industry is always changing, with new frameworks emerging within the JavaScript ecosystem. This presents an opportunity to explore how these frameworks—like React, Angular, or Vue—interact with each other. However, this thesis does not cover the performance of each framework across different use cases as our application was built with only ReactJs due to our Survey Result from JavaScript Developers. Examining these dynamics could offer more informed valuable insights for developers and businesses seeking to leverage the latest advancements in software engineering, especially when using JavaScript as the primary development language.

7 BIBLIOGRAPHY

[1] [Online]. Available: https://www.wirfs-brock.com/allen/jshopl.pdf. [Accessed 2 2 2024].

[2] "History-Computer.com," Mosaic Browser, 14 10 2020. [Online]. Available: https://historycomputer.com/Internet/Conquering/Mosaic.html. [Accessed 22 12 2023].

[3] 1997. [Online]. Available: https://archives.ecma-international.org/1997/GA/97-063-excerpt.pdf. [Accessed 8 2 2024].

[4] "evolution-javascript-journey-from-es1-latest-version," [Online]. Available: https://www.linkedin.com/pulse/evolution-javascript-journey-from-es1-latest-version-part-lebbos-za9fe/. [Accessed 3 2 2024].

[5] "the-ecmascript-journey/medium," [Online]. Available: https://medium.com/@vitorbritto/the-ecmascript-journey-5332c42396c0. [Accessed 15 2 2024].

[6] "mozilla," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions. [Accessed 1 2 2024].

[7] "w3schools," [Online]. Available: - https://www.w3schools.com/js/js_es5.asp. [Accessed 4 2 2024].

[8] [Online]. Available: https://cdn2.hubspot.net/hubfs/1958393/Partner_Portal_-_Collateral/Concepts_and_Benefits_of_Repo_Management.pdf?t=1482418124868. [Accessed 12 2 2024].

[9] [Online]. Available: https://dzone.com/refcardz/binary-repository-management#section-5. [Accessed 21 1 2024].

[10] P. S. a. S. R. E. Wittern, "A look at the dynamics of the javascript package ecosystem," in *In Proceedings of the 13th International Conference on Mining Software Re-positories, MSR '16,*, New York, ACM, 2016, p. 351–361.

[11] T. Mens, " An ecosystemic and socio-technical view on software maintenance and evolu-tion.," *IEEE International Conference on Software Maintenance and Evolution (In-vited Paper), ICSME'16, ,* 2016.

[12] [Online]. Available: https://boa.cs.iastate.edu/papers/tosem15.pdf. [Accessed 11 1 2024].

[13] "Architecture multi-tiers.," [Online]. Available: http://java.developpez.com/archi_multi-tiers.pdf. [Accessed 15 1 2024].

[14] "Core J2EE Patterns," . [Online] available at [accessed 26-1-2024], [Online]. Available: http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html. [Accessed 26 1 2024].

[15] [Online]. Available: https://www.interviewbit.com/blog/mvc-architecture/. [Accessed 29 1 2024].

[16] "Github," [Online]. Available: https://github.com/creativedeveloper-net/npm-package-example. [Accessed 18 2 2024].

[17] "nodejs," [Online]. Available: https://nodejs.org/api/http.html#http_class_http_incomingmessage. [Accessed 18 2 2024].

[18] altexsoft, "altexsoft," [Online]. Available: https://www.altexsoft.com/blog/react-pros-and-cons/. [Accessed 15 2 2024].

[19] "react dev," [Online]. Available: https://react.dev/. [Accessed 18 02 2024].

[20] "npm trends," [Online]. Available: https://npmtrends.com/react. [Accessed 19 4 2024].

[21] "altexsoft 2," [Online]. Available: https://www.altexsoft.com/blog/the-good-and-the-bad-of-angular-development/ . [Accessed 19 3 2024].

[22] [Online]. Available: https://angular.io/guide/architecture. [Accessed 26 4 2024].

[23] [Online]. Available: https://www.tutorialspoint.com/vuejs/vuejs_environment_setup.html. [Accessed 28 2 2024].

[24] T. B. Lee, "The world wide web: A very short personal history," 5 12 1998. [Online]. Available: http://www.w3.org/People/Berners-Lee/ShortHistory.html.

[25] "envatotuts+," 8 8 2022. [Online]. Available: - https://code.tutsplus.com/what-is-javascript--cms-26177t#toc-0cyt-what-is-javascript. [Accessed 5 12 2023].

[26] [Online]. Available: https://medium.com/jspoint/how-javascript-works-in-browser-and-node-ab7d0d09ac2f. [Accessed 15 1 2024].

[27] "World Wide Web: Proposal for a HyperText Project," [Online]. Available: http://www.w3.org/Proposal.html. [Accessed 16 12 2023].

[28] [Online]. Available: https://ihoneymaan.medium.com/javascript-engine-and-how-it-works-e1fa2f7a657c. [Accessed 9 1 2024].

[29] P. Anderson, "All That Glisters Is Not Gold' -- Web 2.0 And The Librarian," *Journal of Librarianship and Information Science,* vol. 39, no. 4, p. 195–198, 2007.

[30] O'Reilly, "Definition of Web 2.0," 20 12 2006. [Online]. Available: http://radar.oreilly.com/archives/2006/12/web-20-compactdefinition-tryi.html. [Accessed 4 12 2023].

[31] N. Ossi, "Semantic Web: Definition," 31 3 2003. [Online]. Available: http://www.w3c.tut.fi/talks/2003/0331umediaon/slide6-0.html. [Accessed 25 2 2024].

[32] M. A. N. a. H. K. F. Sareh Aghaei, "Evolution of the World Wide Web: From Web 1.0 to Web 4.0",," *Computer Engineering Department, University of Isfahan, Isfahan, Iran, International Journal of Web & Semantic Technology (IJWesT),* vol. 3, no. 1, pp. 1-10, 2012.

[33] [Online]. Available: https://www.lxahub.com/stories/whats-the-difference-between-web-1.0-web-2.0-and-web-3.0. [Accessed 25 1 2024].

[34] [Online]. Available: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-web-3-0/. [Accessed 20 1 2024].

[35] A. F. a. S. B. S. Jansen, "A sense of community: A research agenda for software ecosystems," *Presented at the 31st International Conference on Software Engineering - Companion Volume, 2009. ICSE-Companion 2009, IEEE,* no. 31, p. 187–190, 2009.

[36] "Npm package manager for javascript," [Online]. Available: https://www.npmjs.com/. [Accessed 11 2 2024].

[37] [Online]. Available: https://sharvishi9118.medium.com/how-to-compose-micro-frontends-at-build-time-c5e484a40e10. [Accessed 17 2 2024].

[38] Chris.Minnick.Wiley. [Online]. Available: https://dl.ebooksworld.ir/books/Beginning.ReactJS.Foundations.Chris.Minnick.Wiley.9781119685548.EBooksWorld.ir.pdf. [Accessed 5 3 2024].

[39] "npmjs," [Online]. Available: https://www.npmjs.com/package/react. [Accessed 15 2 2024].

[40] "simplilearn," [Online]. Available: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular#why_angular. [Accessed 21 3 2024].

[41] [Online]. Available: = https://data-flair.training/blogs/angular-architecture-components/[accessed 22-2-2024]. [Accessed 25 3 2024].

[42] [Online]. Available: https://www.tutorialspoint.com/angular2/angular2_metadata.html. [Accessed 24 2 2024].

ABBREVIATIONS

DOM	Document Object Model
ECMASCRIPT	European Computer Manufacturers Association Script
MVC	Model View Controller
HTML	Hypertext Markup Language
NPM	Node Package Manager
API	Application Programming Interface
OSS	Open Source Software
UI	User Interface
DB	Database
JSON	JavaScript Object Notation
OOP	Object-Oriented Programming
CSS	Cascading Style Sheet
CI/CD	Continuous integration/Continuous Deployment

LIST OF FIGURES

APPENDIX