AXXES

Carpool Application

Mikolaj, Fabrice, Arne, Jef, Eva, Maxim

Contents

Introduction

1 General Project Plan

1.1 ERD

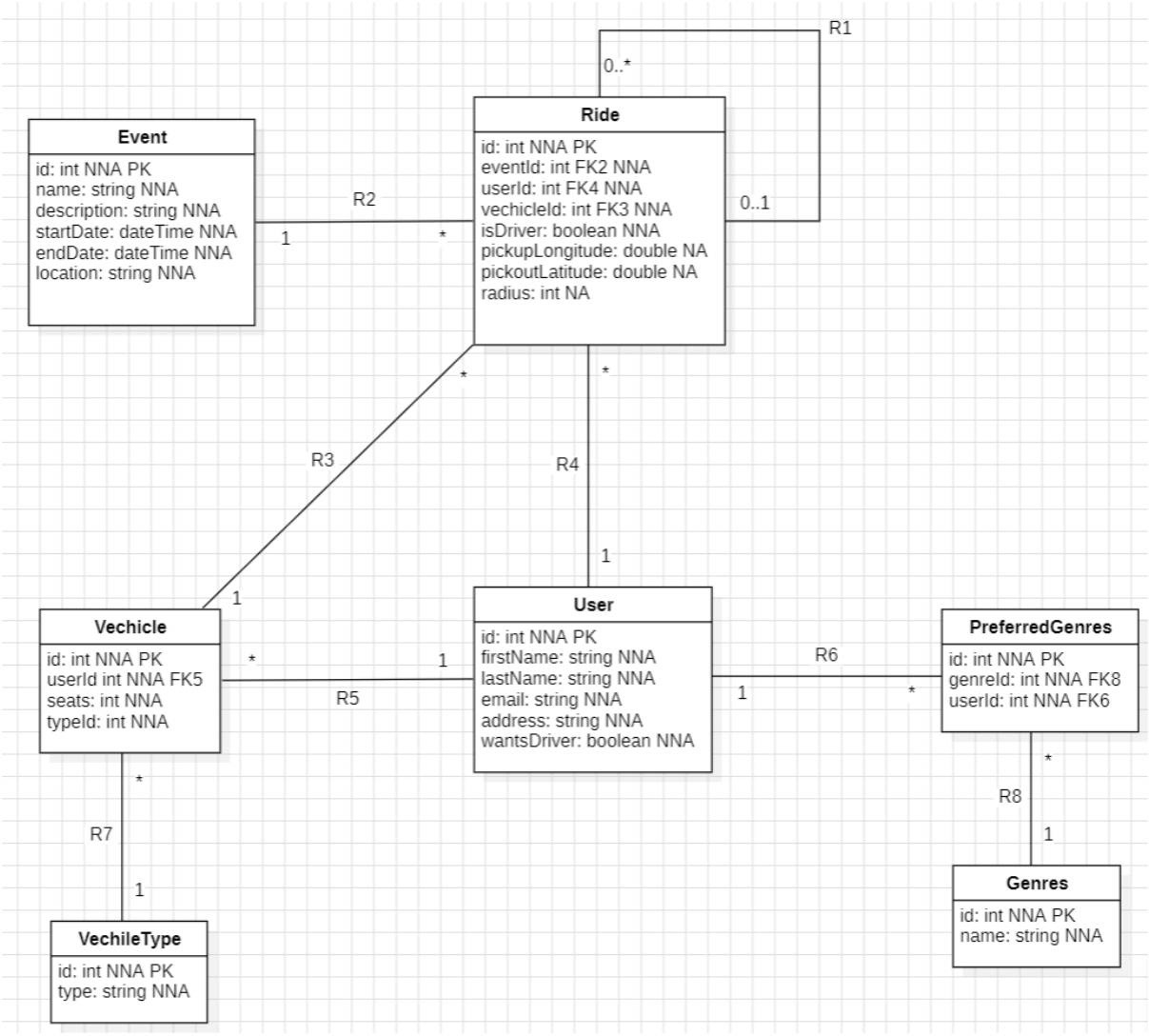

Figure 1: ERD

1.2 Use case diagram

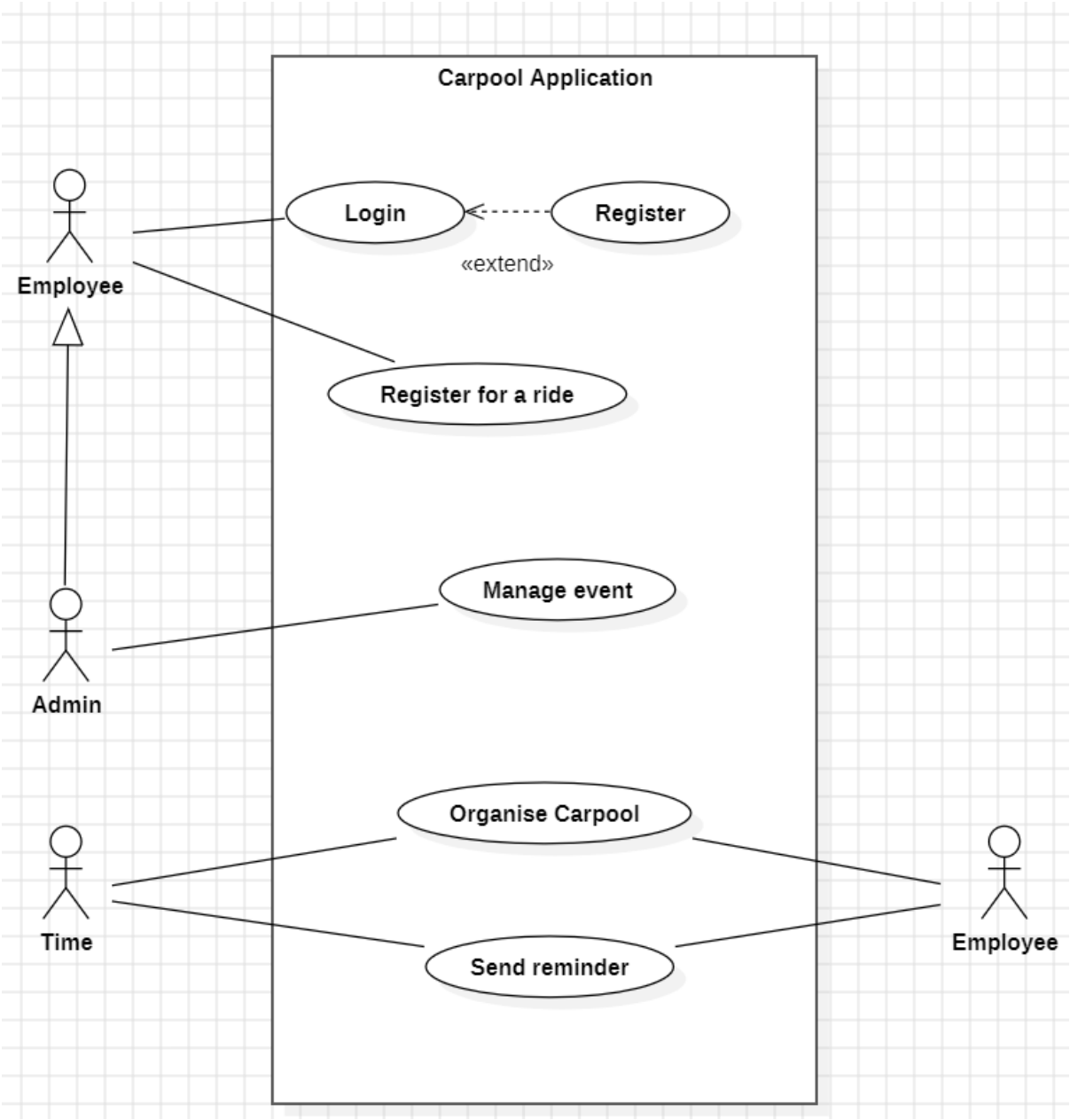

Figure 2: Use case diagram

1.3 Branching strategy

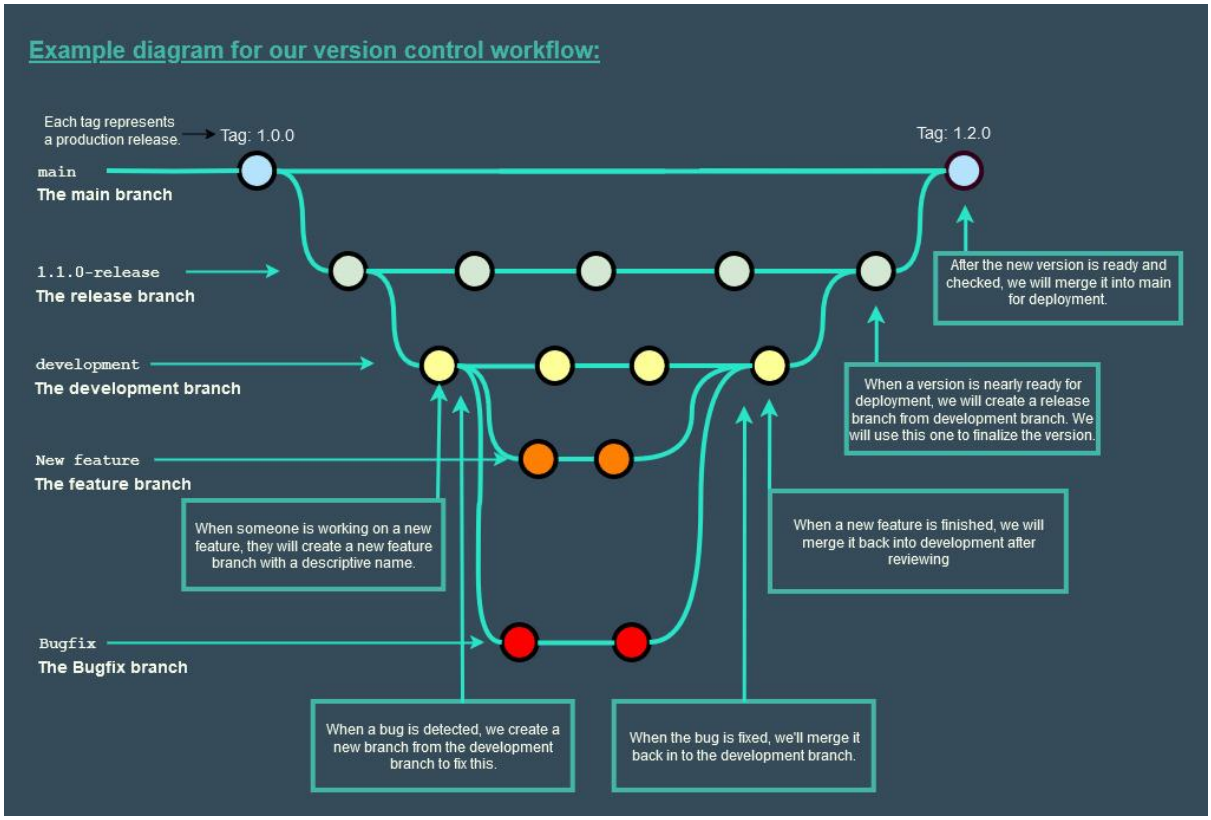

Figure 3: Branching Strategy

1.3.1 Branches:

Main branches:

Main (master): This branch contains the fully working and approved version of the application. Only completed and tested work is merged here.

Development branch: The main working branch where all features and fixes are integrated. This is where the team collaborates.

Feature branches:

A new **"feature branch"** will be created for each new functionality from the develop branch. We will use clear and descriptive names, such as:

- *Feature/user-authentication*
- *Feature/event-management*
- *Feature/carpool-matching*

Bugfix branches:

For bugs discovered in the **development or main** branch we'll use a **"bugfix branch"**, such as:

- *Bugfix/ui-alignment*
- *Bugfix/api-endpoint-error*

Release branches:

When a version is nearly ready for deployment, we will create a **“release branch”** from **“development branch”**, for example:

- *Release/v1.0*

We will use this branch to finalize the version, fix minor issues and update the documentation BEFORE merging it into **“Main branch”.**

1.3.2 Workflow

New features:

- Start from **development** branch
- Work in a feature branch
- Once completed, create a pull request to **develop.**
- Ask team members to review your code before merging.
- When reviewed, only the team lead will merge after receiving confirmation from the developer AND the reviewer.

Bug fixes:

- Create a bugfix branch from **development** branch.
- Merge the fix back into the relevant branches. (first into development branch, then doublecheck, then into main)

Releases:

- We will use the release branch to polish and prepare a stable version.
- Merging into main, will be done with a tag with a version number.

1.3.3 Tools:

We will use a version control platfrom, GitLab for managing branches, pull requests and code review. This tool make collaboration more organized and transparent.

Key things to remember:

- Never work directly in the main branch.
- Make sure your commit messages are clear and describe what you’ve done.
- Always ask for a code review.

1.3.4 An example of how we will use branching

Step 1: We will start with the **main** and **develop** branches

1. **Main branch:** This branch contains the stable, finished version of our app. We only merge into main when a new version is fully ready.
2. **Develop branch:** This is where our team collaborates and integrates new features.

Step 2: We create separate branches for new features

Each new functionality gets its own branch. For example:

1. If you're working on "Creating User Profiles," you create a branch: *feature/profile-creation*.
    - Write and test your code in this branch.
2. If another team member works on "Managing Events," they create: feature/event-management.

Step 3: Merge your code into the **development** branch

When you finish your feature:

1. Create a Pull Request (PR) in GitLab. This is a request to merge your feature into development branch.
2. Ask a teammate to review your code to ensure it works and follows the project rules.
3. Once approved, your branch is merged into **development** branch by the team lead.

Step 4: Prepare a new version

When we’re ready to release a version:

1. Create a release branch from **development branch**, e.g., *release/v1.0*.
2. Fix any minor bugs and update documentation in this branch.
3. Let the team lead after reviewing merge the **release branch** into **main** and tag it with a version number.

Step 5: Fix bugs

1. If there’s a bug in the develop branch, create a bugfix branch, e.g., *bugfix/api-error*.
2. Fix the bug, then let your fix be reviewed and after review the team lead will merge the branch back into **development branch**.

2 Front-end

For the front-end we were given four different options to choose from: Angular, React, Vue, Svelte. The other perquisite was the use of Typescript. We researched, compared and chose which one to use under a couple of different factors. The importance of each factor was determined using the assignment guidelines and additional information provided by Axxes.

Security is by far the most important factor, as we are working with real employee address data. For this reason, we have structured our research into two parts: General and Security.

In the following pages, you will find our research, followed by the security-focused weighted ranking method

Finally we draw our conclusion using a general weighted ranking method

2.1 Angular

Angular is a TypeScript-based free and open-source single-page web application framework. It is developed by Google and by a community of individuals and corporations. (contributors., Angular (web framework), 2024)

Below, you will find a summary of the research we did on the Angular framework.

2.1.1 General

Throughout the semester, Angular is by far the framework we used most, making it the best option experience-wise. However, to ensure we made the right decision and avoided being overly biased towards Angular, we did extensive research.

Positive points	Negative points
Very structured	Hard to master
Well documented	High amount of boilerplate code
Support of two-way data binding	Large bundle size
Developed using typescript	

(Agiliway, 2024)

2.1.2 Security

XSS Protection:

Angular has **built-in mechanisms** in its templating system that automatically sanitize user inputs. For advanced use cases like sanitizing user-generated URLs, it offers **modules like DomSanitizer** to handle security-sensitive scenarios. (Angular, Security, n.d.)

CSRF Protection:

Not directly included, but Angular's HttpClient makes it straightforward to integrate **CSRF tokens** into API requests. This means backend security measures like token generation can easily be connected to frontend requests. (Angular, Security, n.d.)

Security Headers:

Angular Universal (for server-side rendering) integrates well with middleware to manage **CSP**, **HSTS**, and other security headers, ensuring a secure content delivery pipeline. (Angular, Security, n.d.)

Dependency Management:

Angular's CLI includes **built-in dependency monitoring**. It checks for vulnerabilities during builds, reducing risks from outdated or insecure packages.

- **Security Features**:
  - **XSS Protection**: Angular's templating system escapes HTML by default, preventing XSS attacks.
  - **Dependency Injection (DI)**: Built-in DI framework reduces risks of injection attacks by ensuring safe code execution. (Angular, Understanding dependency injection, n.d.)
  - **Sanitization**: Angular's DomSanitizer explicitly sanitizes unsafe HTML, URLs, and scripts. (Angular, DomSanitizer, n.d.)
- **How to Implement**:
  - Use Angular's security documentation for best practices.
  - Always use HttpClient for API calls instead of raw HTTP libraries.
  - Apply **Content Security Policies (CSP)** on the server to prevent unauthorized script execution.

2.2 React

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library that aims to make building user interfaces based on components more "seamless". It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. (contributors., React (software), 2024)

Below, you will find a summary of the research we did on the React framework.

2.2.1 General

In class we did use React, but not as extensively as Angula. Unlike Angular, we didn't have to build a project for this course. We covered the basics of both regular React and React Native

Positive points	Negative points
Easy to learn and use	High paced development
JSX combines HTML and JavaScript in one file	Poor documentation
SEO friendly	JSX might create huge files

2.2.2 Security

XSS Protection:
React offers protection against Cross-Site Scripting (XSS) attacks by **escaping embedded expressions** within JSX. This ensures that any dynamic content (e.g., user input rendered in a component) is treated as plain text and cannot execute malicious code.

However, when rendering raw HTML using dangerously SetInnerHTML, React **bypasses its default escaping mechanism**. This can introduce vulnerabilities if the HTML content is not sanitized. Developers must manually sanitize such content using libraries like DOMPurify to ensure it is safe before rendering.

CSRF Protection:
React does not provide built-in support for handling Cross-Site Request Forgery (CSRF). Developers are responsible for implementing CSRF protections, often by integrating with backend solutions that generate CSRF tokens. Tools like axios can be used to include CSRF tokens in API requests.

Security Headers:
React does not manage security headers (e.g., Content Security Policy, HSTS) natively. These must be configured server-side using middleware such as Helmet in Node.js or appropriate server configurations (e.g., Nginx or Apache). These headers help mitigate risks such as clickjacking, code injection, and other attacks.

Dependency Management:
React does not offer built-in dependency management tools. Developers must use tools like npm audit to identify vulnerabilities in dependencies. Regularly updating dependencies and monitoring for security patches is essential for maintaining application security.

Community and Ecosystem:
React has a vast ecosystem of third-party tools and libraries for enhancing security. While the framework itself is minimal, it integrates well with solutions like Helmet for server-side security and DOMPurify for sanitizing content. Its large community ensures that vulnerabilities are quickly addressed and documented.

Security Features:

- **XSS Protection**: React automatically escapes content in JSX to prevent XSS.
- **Content Security Policies**: Not enforced natively but can be implemented via server configuration.
- **State Management Tools**: Tools like Redux enhance control over application state, indirectly improving security.

How to Implement:

- Use libraries like DOMPurify for sanitizing user input and preventing XSS.
- Implement **strict CSP headers** in your backend for additional security.
- Use tools like Helmet to manage security headers in React.

(React, n.d.)

(Axios, n.d.)

(HelmetJS, n.d.)

2.3 Svelte

Svelte is a free and open-source component-based front-end software framework, and language created by Rich Harris and maintained by the Svelte core team members. (contributors, Svelte, 2024)

Below, you will find a summary of the research we did on the Svelte framework.

2.3.1 General

We haven't used Svelte in class, so we can't compare it directly with frameworks we know. However, from what we've seen, it has similarities to React.

Positive points	Negative points
Less boilerplate code	Very small community
No more virtual DOM → faster	Lack of IDE support
Minimal state management	Lack of documentation
Significantly faster than competitors	Few toolkits

(Russo, n.d.)

2.3.2 Security

XSS Protection

Svelte lacks automatic protection against XSS. Developers are expected to **manually sanitize user inputs** using libraries or backend validation.

CSRF Protection

There's no built-in mechanism for CSRF, meaning developers must rely on backend solutions and pass **CSRF tokens** through HTTP headers in API requests.

Security Headers

Like React, Svelte does not manage headers directly. Security features like **CSP** or **HSTS** must be configured server-side through middleware or server settings.

Dependency Management

Svelte's lightweight nature reduces the number of dependencies, which minimizes the attack surface. However, updates and audits are entirely manual.

Security Features:

- **XSS Protection**: Does not escape content by default; developers must sanitize manually. (Svelte, Guide to Preventing XSS in Svelte, n.d.)
- **Lightweight Framework**: Fewer dependencies reduce the attack surface.

How to Implement:

- Sanitize all user inputs using libraries like DOMPurify or similar.
- Rely on strong backend validation for all data and API responses.
- Use server-side headers like CSP to prevent unauthorized code execution.

(Svelte, Official Documentation, n.d.)

2.4 Vue

Vue.js is an open-source model–view–viewmodel front end JavaScript framework for building user interfaces and single-page applications. (contributors, 2024)

Below, you will find a summary of the research we did on the Vue framework.

2.4.1 General

Like Svelte we did not cover this during the lessons, but from what we've seen, it shares similarities with Svelte, particularly in how it handles reactivity and components.

Positive points	Negative points
Lightweight & fast	Not mainly TypeScript
Simple learning curve	Code structure risks
Good community support	Small community
Good documentation	

(Bojanowska, 2024)

2.4.2 Security

XSS Protection:

Vue **partially auto-sanitizes** templates, offering some protection against XSS. For advanced scenarios, additional tools like vue-sanitize or DOMPurify are recommended for stronger protection of user inputs. (VueJS, Vue-sanitize, n.d.)

CSRF Protection:

Vue does not include CSRF protections but integrates well with libraries like axios to handle **CSRF tokens** in requests.

Security Headers:

Security headers like **CSP** or **HSTS** must be managed server-side. Vue SSR (server-side rendering) tools can help integrate middleware for managing headers.

Dependency Management:

Dependency updates and audits are largely manual, but Vue CLI offers basic support for managing and monitoring project dependencies.

Security Features:

- **XSS Protection**: Partially escapes content in templates; additional libraries recommended for complete protection.
- **Reactivity System**: Helps prevent malicious script injection via controlled state updates.

How to Implement:

- Use tools like vue-sanitize to clean user-generated content.
- Leverage Vue's v-bind for dynamic binding while avoiding raw HTML rendering.

(VueJS, Official Documentation, n.d.)

(VueJS, Vue Security Best Practices, n.d.)

2.5 Security comparison table

Before we dive into the weighted ranking method, let's summarize the security features of each frontend framework we've researched.

Security Principle	Angular	React	Svelte	Vue
Built-in Protection Against XSS	Comprehensive; strict template-based approach to prevent XSS.	Relies on manual sanitization and libraries.	Limited; depends on manual sanitization.	Partial; some sanitization but needs libraries for robust defense.
CSRF Protection	No built-in solution; requires backend CSRF tokens.	No built-in solution; relies on third-party libraries.	Backend handling or libraries required.	Relies on external tools like libraries or server-side CSRF tokens.
Security Headers Integration	Middleware and libraries integrate easily.	Requires backend or libraries to manage security headers.	Relies on developer/server-side configuration.	Middleware or backend tools required for managing headers.
Dependency Management	Strong; Angular CLI checks and updates dependencies to reduce risks.	Moderate; developers manage dependencies manually.	Moderate; fewer dependencies, but updates rely on developers.	Moderate; dependency updates and integrity checks are manual.
Preventing Injection Attacks	Built-in dependency injection reduces risks.	No inherent protection; relies on backend.	Minimal protection; backend validation is essential.	No specific features; relies on backend or libraries.
Community and Ecosystem	Strong focus on security with frequent updates and official best practices.	Large ecosystem with extensive third-party libraries.	Smaller ecosystem; limited security-specific tools.	Moderate ecosystem; security requires additional tools and practices.

2.6 Security – weighted ranking method

The Weighted Ranking Method (WRM) outlined in this document is constructed based on the OWASP Top 10 Web Application Security Risks. The OWASP (Open Web Application Security Project) is a globally recognized non-profit organization dedicated to improving software security. Their "Top 10" list identifies the most critical security risks facing web applications, serving as an essential resource for developers, security professionals, and organizations worldwide.

The OWASP Top 10 represents a prioritized list of the most impactful and commonly exploited vulnerabilities in web applications. These risks include well-known categories such as Injection Attacks (A03:2021), Broken Access Control (A01:2021), and Security Misconfiguration (A05:2021), among others. By addressing these risks, developers can ensure the foundational security of their applications and mitigate potential exploitation.

In this WRM, I have evaluated four popular frontend frameworks—Angular, React, Svelte, and Vue—against six key security criteria. These criteria directly map to relevant OWASP risks, ensuring that the selection of a framework aligns with the most critical security concerns for modern web applications. The weights assigned to each criterion reflect the relative importance of the corresponding OWASP risks, emphasizing the prevention of high-impact vulnerabilities like injection attacks and cross-site scripting (XSS).

This systematic approach allows for an informed comparison of frontend frameworks, grounded in globally accepted security standards. By basing the WRM on the OWASP Top 10, I aim to prioritize security in the decision-making process for selecting a framework for my web application development.

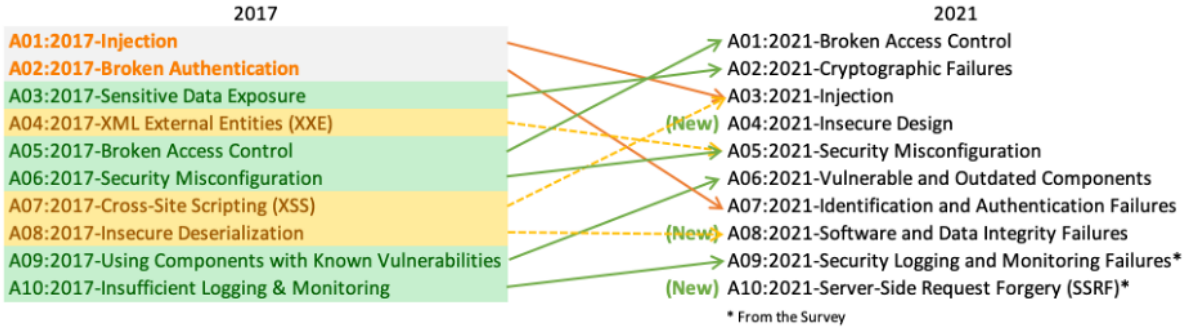

Figure 4: Web application security risks 2021 - Owasp. https://owasp.org/www-project-top-ten/

Criteria	Weight	Angular	React	Svelte	Vue
XSS Protection	4	4	3	2	3
CSRF Protection	3	4	3	3	3
Security Headers Integration	2	4	2	3	3
Dependency Management	3	5	3	3	3
Injection Attack Prevention	5	5	3	2	2
Community and Ecosystem	3	4	5	3	4
Total Weighted Score	**N/A**	88	64	51	58

XSS Protection

- **Relevance to OWASP**: Related to **A03:2021-Injection**. Cross-Site Scripting (XSS) is a subset of injection vulnerabilities where attackers inject malicious scripts into trusted websites. Frameworks with built-in XSS protections mitigate this risk by sanitizing inputs and escaping HTML.
- **Suitability**: Crucial for frontend frameworks as they directly handle user inputs and outputs.

CSRF Protection

- **Relevance to OWASP**: Tied to **A07:2021-Identification and Authentication Failures**. Cross-Site Request Forgery (CSRF) exploits users' authenticated sessions to perform malicious actions. Frontend frameworks that facilitate CSRF token integration strengthen this aspect.
- **Suitability**: Essential for applications requiring strong session management and user authentication.

Security Headers Integration

- **Relevance to OWASP**: Connected to **A05:2021-Security Misconfiguration**. Security headers such as Content Security Policy (CSP) and HTTP Strict Transport Security (HSTS) protect applications against common attacks like clickjacking and XSS.
- **Suitability**: Important for preventing configuration-related vulnerabilities in the frontend.

Dependency Management

- **Relevance to OWASP**: Supports **A06:2021-Vulnerable and Outdated Components**. Frontend frameworks often rely on third-party libraries, and managing their security is critical to avoid vulnerabilities introduced through dependencies.
- **Suitability**: Vital for maintaining a secure development lifecycle.

Injection Attack Prevention

- **Relevance to OWASP**: Directly linked to **A03:2021-Injection**. Frameworks with mechanisms to prevent injection attacks (e.g., SQL injection or command injection) reduce the likelihood of such vulnerabilities in the client-server communication.
- **Suitability**: High priority as injection attacks are among the most common and severe web application vulnerabilities.

Community and Ecosystem

- **Relevance to OWASP**: Indirectly supports multiple risks, including **A09:2021-Security Logging and Monitoring Failures**. A strong community ensures prompt identification and resolution of vulnerabilities, sharing of best practices, and timely updates.
- **Suitability**: Secondary but important, as it ensures ongoing support and security enhancements.

(OWASP., n.d.)

2.7 Conclusion – weighted ranking method

As previously mentioned, we used the weighted ranking method to determine which framework was the best fit for the use case.

We discussed in group what were the most important criteria for this application.

We decided with the whole group which framework gets what points on which criteria. Below the table we will discuss why a certain framework won on that criterion

Criteria	Weight	React	Angular	Svelte	Vue
Ecosystem and extensibility	4	4	5	2	3
Team experience and expertise	4	4	5	1	2
Security	5	3	5	4	3
Testing & debugging	4	4	4	3	3
Community and popularity	1	5	4	2	3
Official documentation	3	4	5	2	2
Performance	5	5	4	4	5
Responsive mobile development	4	5	4	3	3
Scalability	4	3	4	2	3
Total	170	137	152	92	106

Ecosystem and extensibility :

Angular is a more mature ecosystem for the reason it has a rich library and tooling ecosystem. It also integrates seamlessly with other technologies like TypeScript, RxJS, and Material Design, providing a comprehensive development stack. And of course, it is being backed by Google for the ongoing maintenance and support. This is the reason we gave Angular here a 5.

Team experience and expertise :

For the different frameworks we had the most experience with Angular. For the reason we have this year a course Angular, we also have a course React. But for the course Angular we have 6 credits and for React 3 credits, this is the reason we gave a 5 to Angular and a 4 to React. For the other 2 frameworks we have little knowledge.

Security :

Angular is also winner here. For the reasons the security here is very good. IF you take a look in the security document you can see that Angular has excellent build in protection against XSS, dependency management, prevention for injection attacks and so much more. With the other frameworks, the security things are not as easy to implement as with Angular. They are also often better executed on Angular.

Testing and debugging :

For testing and debugging, there are two winners. Angular, for example, has very good tools for unit testing, such as Jasmine and Karma. These tools are excellent for testing and debugging. On the other hand, testing and debugging in React is also very easy. It offers a lot of libraries that you can install for testing, for example, Jest and Enzyme.

Community and popularity :

The winner here is React, we gave it here a 5 for the simple reason React has the biggest community and popularity. If you look at the numbers, 11,908,579 websites use React and 327,765 websites uses Angular. (Vishal-Siddhpara, 2024)

Official documentation :

The official documentation of Angular is very huge, it has a good and large documentation. The official documentation of React is also huge, but not so huge like Angular. Thats why React gets a 4, the other frameworks lack official documentation.

Performance :

Here are 2 frameworks that get a 5, Vue is known for its speed and React is also known for this. Angular is not slow but slower than these 2 frameworks.

Responsive mobile development :

React is often used for developing smaller mobile applications. It's particularly well-suited for creating mobile apps or small-scale web apps. While the other frameworks can also be used for mobile development, Reacts performance and flexibility make it a popular choice for mobile development.

Scalability:

The tooling options on Angular make it a little easier to create a scalable app. It is also possible on React, but here it is a bit more complicated. That is why Angular gets here a 4 and React a 4.

3 Back-end

For the back-end, Axxes didn't provide any specific preferences, so we decided to compare the most commonly used languages and frameworks: .NET, Python, Java, and JavaScript.

As with the front-end, security is our top priority due to the use of employee address data.

In the following pages, you will find our research, followed by the security-focused weighted ranking method

Finally we draw our conclusion using a general weighted ranking method

3.1 .NET

The .NET platform is a free and open-source, managed computer software framework for Windows, Linux, and macOS operating systems. The project is mainly developed by Microsoft employees by way of the .NET Foundation and is released under an MIT License. (Wikipedia contributors, 2024)

Below, you will find a summary of the research we did on the .NET framework.

3.1.1 General

In class we mainly use .NET CORE as our backend framework, so we have the most experience with it. However, we still did proper research to determine if it would be the best fit for this project.

Positive points	Negative points
Object-Oriented Development	Cost of hosting
Cross-platform	Difficult Scalability
Simplified maintenance	
Large community	

(BairesDev, 2022)

3.1.2 Security

Built-in Security Features:

The System.Security.Cryptography namespace in .NET provides robust support for encryption, hashing, and key management.

(Microsoft, .NET cryptography model, n.d.)

Vulnerability Management:

NuGet includes built-in vulnerability notifications and helps developers manage updates to address security risks.

(Microsoft, Best practices for a secure software supply chain, n.d.)

Community and Ecosystem:

The .NET community supports secure development through the Security Development Lifecycle (SDL) and tools such as the Microsoft Threat Modeling Tool.

(Microsoft, Microsoft Security Development Lifecycle (SDL), n.d.)

Integration with Secure Frameworks:

ASP.NET Core includes default protections against CSRF, XSS, and clickjacking, and integrates seamlessly with identity providers like Azure Active Directory.

(Microsoft, ASP.NET Core security topics, n.d.)

Common Vulnerabilities:

Privilege escalation can occur if .NET applications run on poorly configured servers or with excessive permissions.

3.2 Python

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. (contributors., Python (programming language), 2024)

Below, you will find a summary of the research we did on the Python programming language as back-end.

3.2.1 General

During the lessons we did use Python as a programming language, but we never used it as a back-end language.

Positive points	Negative points
Really simple syntax	Performance Overhead
Large and active community	Slow
Easy integration with database	

(Oluseye, Python Backend Development: A Complete Guide for Beginners, 2024)

3.2.2 Security

Built-in Security Features:

Python provides built-in modules such as hashlib for cryptographic hashing and secrets for token generation, making it straightforward to build secure applications.

(Python, hashlib — Secure hashes and message digests, n.d.)

(Python, secrets — Generate secure random numbers for managing secrets, n.d.)

Vulnerability Management:

Python offers tools like pip-audit and Safety that scan dependencies for vulnerabilities and provide actionable insights for remediation.

(Pip, n.d.)

Community and Ecosystem:

Frameworks like Django include built-in security features to prevent XSS, CSRF, and SQL injection, encouraging secure development practices.

(Django, n.d.)

Integration with Secure Frameworks:

Django includes robust security mechanisms such as CSRF and SQL injection protection through its ORM.

Flask, a lightweight framework, can be extended with plugins like Flask-Security for enhanced security.

(Flask, n.d.)

(Django, n.d.)

Common Vulnerabilities:

Reliance on third-party packages from PyPI increases the risk of supply chain attacks. Attackers have previously uploaded malicious packages to PyPI, impersonating popular libraries or introducing

vulnerabilities into unmaintained projects. Developers should verify package maintainers, inspect code if possible, and use automated tools to detect malicious or outdated dependencies.

3.3 Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need to recompile. (contributors, Java (programming language), 2024)

Below, you will find a summary of the research we did on the Java programming language as back-end.

3.3.1 General

We have used Java in class, but not for back-end API development. While we are familiar with the basics of Java syntax, we do not have experience in creating a web API with Java.

Positive points	Negative points
Object-Oriented	Slow and poor performance
Simple to learn, create, maintain, understand	Poor GUI
Platform-Independent	Less efficient memory management

(Sharma, 2024)

3.3.2 Security

Built-in Security Features:

Java provides robust cryptographic support through the java.security package, which includes key management, signature verification, and secure random number generation.

(Oracle, n.d.)

Vulnerability Management:

Tools such as Maven and Gradle include plugins like OWASP Dependency-Check, which help detect vulnerabilities in dependencies.

(OWASP, OWASP Dependency-Check, n.d.)

Community and Ecosystem:

Spring Security is a robust framework that provides strong authentication, authorization, and CSRF protection features.

(Spring, n.d.)

Integration with Secure Frameworks:

Java EE offers a built-in security API for role-based access control and cryptographic operations.

(Jakarta, n.d.)

Common Vulnerabilities:

Java applications can be vulnerable to deserialization attacks when processing untrusted data, which requires careful handling by developers.

(OWASP, Deserialization of untrusted data, n.d.)

3.4 JavaScript

JavaScript is a high-level programming language that follows the ECMAScript standard. It was originally designed as a scripting language for websites but became widely adopted as a general-purpose programming language, and is currently the most popular programming language in use.

(Wikimedia, 2024)

Below, you will find a summary of the research we did on the JavaScript programming language as back-end.

3.4.1 General

Similarly to Java, we have used JavaScript, but not for back-end API development. We typically use it for front-end development.

Positive points	Negative points
Flexible	Security issues
Speed	Browser inconsistency
Rich ecosystem	Complexity in large projects
Active community	

(GeeksforGeeks, 2024)

3.4.2 Security

Built-in Security Features:

The crypto module in Node.js provides powerful capabilities for hashing, encryption, and securing data.

(NodeJS, n.d.)

Vulnerability Management:

npm audit allows developers to identify and remediate known vulnerabilities in their dependencies.

(Npm, n.d.)

Community and Ecosystem:

Middleware such as Helmet enhances HTTP security by setting headers like Content Security Policy and X-Content-Type-Options.

OWASP provides comprehensive guidelines in its Node.js Security Cheat Sheet.

(HelmetJS, n.d.)

(OWASP, NodeJS Security Cheat Sheet, n.d.)

Integration with Secure Frameworks:

Express.js supports middleware for security headers, input validation, and CSRF protection.

(Expressjs, n.d.)

Common Vulnerabilities:

Prototype pollution is a common vulnerability in Node.js, especially when merging untrusted objects.

3.5 Security comparison table

Before we go to the weighted ranking method, let's summarize our research security wise.

Criterion	Python	Java	JavaScript (Node.js)	.NET
Built-in Security Features	Modules like hashlib (cryptography) and secrets (secure token generation).	java.security package for cryptography, TLS, and secure random number generation.	crypto module for hashing and encryption.	System.Security.Cryptography namespace for encryption, hashing, and secure key storage.
Vulnerability Management	Tools like pip-audit and Safety scan for dependency vulnerabilities.	Maven and Gradle with OWASP Dependency-Check for automated vulnerability scanning.	npm audit identifies known vulnerabilities in dependencies.	NuGet provides notifications for vulnerable dependencies and updates.
Integration with Secure Frameworks	Django and Flask offer robust security features (e.g., CSRF, XSS prevention, SQL injection protection).	Spring Security offers comprehensive tools for authentication, CSRF protection, and more.	Express.js integrates middleware like Helmet for headers and CORS for secure requests.	ASP.NET Core includes built-in protections for CSRF, XSS, and clickjacking.
Community and Ecosystem	Extensive libraries and community support; OWASP Python Security project provides best practices.	Large ecosystem with enterprise-grade frameworks like Spring and Jakarta EE.	Massive npm ecosystem with middleware like Helmet and express-validator for added security.	Strong enterprise support with Microsoft-backed tools like the SDL and Threat Modeling Tool.
Common Vulnerabilities	Dependency risks from third-party PyPI packages; dynamic typing may introduce subtle errors.	Deserialization attacks; misconfigurations in older frameworks or insecure defaults.	Prototype pollution, supply chain attacks in npm packages.	Privilege escalation risks on misconfigured servers; insecure serialization vulnerabilities.

3.6 Security – weighted ranking method

The Weighted Ranking Method (WRM) in this document evaluates backend programming languages based on their alignment with the OWASP Top 10 Web Application Security Risks. OWASP (Open Web Application Security Project) is a globally recognized non-profit organization dedicated to improving software security. Its "Top 10" list identifies the most critical security risks affecting web applications, offering guidance to developers and organizations on mitigating these vulnerabilities.

For this WRM, backend programming languages—Python, Java, JavaScript (Node.js), and .NET—were evaluated against key criteria reflecting OWASP risks. Each criterion is weighted on a scale of 1 to 5 based on its importance in mitigating these risks, and each language is scored for its performance in addressing the criteria.

Criteria	Weight	Python	Java	JavaScript	.NET
Built-in Security Features	5	4	5	4	5
Vulnerability Management	4	4	5	4	4
Integration with Secure Frameworks	5	5	5	4	5
Community and Ecosystem	3	5	5	4	4
Common Vulnerabilities Mitigation	5	3	4	3	4
Total score	110	91	105	83	98

Built-in Security Features :

Relevance to OWASP: This criterion is directly related to A03:2021-Injection, A01:2021-Broken Access Control, and A02:2021-Cryptographic Failures. Backend languages with built-in libraries for hashing, encryption, and access management make it easier to mitigate these risks.

Suitability: This is a crucial criterion as the inherent security capabilities of a language significantly impact the overall application security.

Vulnerability Management :

Relevance to OWASP: This aligns with A06:2021-Vulnerable and Outdated Components. Tools like dependency scanners (e.g., Maven for Java or pip-audit for Python) help developers detect outdated or vulnerable components.

Suitability: This criterion is very important as supply chain attacks and outdated software are major sources of vulnerabilities.

Integration with Secure Frameworks :

Relevance to OWASP: This supports multiple risks, such as A03:2021-Injection, A07:2021-Identification and Authentication Failures, and A05:2021-Security Misconfiguration. Secure frameworks like Django, Spring Security, or ASP.NET Core provide built-in protections against SQL injection, XSS, and CSRF.

Suitability: This is essential as frameworks guide developers in implementing secure practices.

Community and Ecosystem :

Relevance to OWASP: Although not directly tied to a specific risk, the size and quality of a community play a vital role in addressing vulnerabilities quickly and sharing best practices. This indirectly supports A09:2021-Security Logging and Monitoring Failures and other risks.

Suitability: This criterion is useful but secondary compared to the more technical aspects of security.

Built-in Security Features :

Relevance to OWASP: This criterion is directly related to A03:2021-Injection, A01:2021-Broken Access Control, and A02:2021-Cryptographic Failures. Backend languages with built-in libraries for hashing, encryption, and access management make it easier to mitigate these risks.
Suitability: This is a crucial criterion as the inherent security capabilities of a language significantly impact the overall application security.

Vulnerability Management :

Relevance to OWASP: This aligns with A06:2021-Vulnerable and Outdated Components. Tools like dependency scanners (e.g., Maven for Java or pip-audit for Python) help developers detect outdated or vulnerable components.
Suitability: This criterion is very important as supply chain attacks and outdated software are major sources of vulnerabilities.

Integration with Secure Frameworks :

Relevance to OWASP: This supports multiple risks, such as A03:2021-Injection, A07:2021-Identification and Authentication Failures, and A05:2021-Security Misconfiguration. Secure frameworks like Django, Spring Security, or ASP.NET Core provide built-in protections against SQL injection, XSS, and CSRF.
Suitability: This is essential as frameworks guide developers in implementing secure practices.

Community and Ecosystem :

Relevance to OWASP: Although not directly tied to a specific risk, the size and quality of a community play a vital role in addressing vulnerabilities quickly and sharing best practices. This indirectly supports A09:2021-Security Logging and Monitoring Failures and other risks.
Suitability: This criterion is useful but secondary compared to the more technical aspects of security.

Common Vulnerabilities Mitigation :

Relevance to OWASP: This criterion directly addresses A08:2021-Software and Data Integrity Failures, A10:2021-Server-Side Request Forgery (SSRF), and other OWASP risks. It evaluates how well a language helps developers avoid common vulnerabilities like deserialization attacks and prototype pollution.
Suitability: This is a critical criterion as it directly measures how effectively a language mitigates real-world threats.

Relevance to OWASP: This criterion directly addresses A08:2021-Software and Data Integrity Failures, A10:2021-Server-Side Request Forgery (SSRF), and other OWASP risks. It evaluates how well a language helps developers avoid common vulnerabilities like deserialization attacks and prototype pollution.

Suitability: This is a critical criterion as it directly measures how effectively a language mitigates real-world threats.

(OWASP., n.d.)

3.7 Conclusion – weighted ranking method

For the back-end we will also use a weighted ranking method to decide which language/framework fits best.

Below you can find the raking table we used to determine our winner, followed up by a motivation for the framework that won each criterion.

Criteria	**Weight**	**.NET**	**Java**	**JavaScript**	**Python**
Ecosystem and extensibility	**4**	4	4	4	5
Team experience and expertise	**4**	5	4	2	4
Security	**5**	4.45	4.75	3.75	4.15
Testing & debugging	**5**	5	5	3	5
Community and popularity	**1**	5	4	3	5
Official documentation	**3**	5	5	3	5
Performance	**5**	4	3	4	4
Scalability	**4**	5	5	5	4
Integration Angular	**5**	5	5	3	4
Hosting cost	**3**	2	4	5	5
Total		**174.25**	**171.75**	**157.75**	**172.75**

Ecosystem and extensibility :

Python has the largest ecosystem; it is also easy to integrate. Actually, it was the same with the other languages, but Python is just a little bigger. This is why we gave Python a 5 and the rest a 4.

Team experience and expertise :

Our team experience is the best with .NET. Because we make everything in the backend with .NET. As a result, our knowledge at .NET is the greatest. We also have good knowledge of Java and Python. But since we make everything in the backend with .NET, it gets a 5 here.

Security :

In our research into the security of these languages; We noticed that Java was clearly the best. It has the best built-in security features along with .NET. But for Vulnerability Management, Java was better than all the rest. It scores on the following 3 criteria: Integration with Secure Frameworks, Community and Ecosystem and Common Vulnerabilities Mitigation. Also, as the highest, or with a shared first place. If you want to read more details about it, you can always read our security file

Testing and debugging :

.NET, Java, and Python are all excellent in testing and debugging, offering robust unit testing frameworks and effective error-handling. Each has strong community support and tools that make debugging efficient. Thats why we give those 3 a 5.

Community and popularity :

The Python and .NET communities for the backend are both the same size. These 2 programming languages are both most commonly used in the backend.

Official documentation :

The official documentation of Python, Java and .NET are both very large for the backend. There may be too much supply for .NET, so it can take a long time before you find something. This means we give them both the same amount.

Performance :

The performance for Java is a bit less than the rest. The other 3 perform well, but none stands out. For this reason, they all get a 4 except Java.

Scalability :

The scalability is very well provided for all frameworks. We did notice that it is a little more difficult for Python. For this reason, we gave Python one point less and the rest a 5.

Integration Angular :

The integration with Angular is an important point. Since we are going to use Angular in the front. The integration with .NET is very good, this is the reason we always use it. But the integration with Java also went smoothly. It was a little more difficult for Python and even more so for JavaScript. That's why Java and .NET both get a 5.

Hosting cost :

If we look at the hosting cost, we see that Python and JavaScript are the cheapest compared to the rest. Java is slightly more expensive than this. But .NET is much more expensive, for .NET you had to pay almost twice as much as for Java.

4 Database

Since Axxes didn't provide any specific preferences, we chose to use three popular and widely used databases to store our data: MySQL, PostgreSQL, and MongoDB.

In the following pages, you will find our research, followed by the security-focused weighted ranking method

Finally we draw our conclusion using a general weighted ranking method

4.1 MySQL

MySQL is an open-source relational database management system (RDBMS). A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. SQL is a language that programmers use to create, modify and extract data from the relational database, as well as control user access to the database.

(contributors, MySQL, 2024)

4.1.1 General

Despite MySQL being the most popular open-source DBMS, we also use it a lot in every class where we build an app that requires a database. This means that the difference between knowledge of other DBMS's is really big. However, we still did proper research to determine if it would be the best fit for this project.

Positive points	Negative points
Open source	Stability issues
Large community	Poor performance with high loads
High performance, scalability and flexibility	

(The Blue Claw, 2021)

4.1.2 Security

MySQL offers a robust set of security features designed to protect sensitive data, enforce secure access, and comply with industry standards. Below is a detailed breakdown of MySQL's security capabilities.

General Security Considerations

MySQL emphasizes the importance of secure configurations and operational practices to minimize vulnerabilities:

- Always run MySQL as a non-root user to limit the impact of potential security breaches.
- Limit the permissions of MySQL directories and files to prevent unauthorized access.
- Regularly update MySQL to the latest version to mitigate known vulnerabilities.

(MySQL, General Security Issues, n.d.)

Access Control and Authentication

MySQL provides flexible access control mechanisms to enforce secure authentication and manage user permissions:

- **Role-Based Access Control (RBAC)**: Assigns roles to users for easier permission management.
- **Granular Privileges**: Privileges can be defined at the global, database, table, and even column level.

- **Authentication Plugins**: Support for authentication mechanisms, including:
  - Native password authentication.
  - Pluggable authentication via LDAP or PAM.

(MySQL, Security Components and Plugins., n.d.)

(MySQL , Access Control and Account Management., n.d.)

Encrypted Connections

To secure data in transit, MySQL supports encrypted client-server communication using TLS/SSL:

- **TLS/SSL Support**: Encrypts communication between MySQL servers and clients to protect against interception.
- Certificates can be self-signed or issued by a trusted Certificate Authority (CA).
- MySQL supports OpenSSL for enhanced compatibility and security.

(MySQL, using Encrypted Connections., n.d.)

Data Encryption

MySQL ensures data confidentiality through advanced encryption mechanisms:

- **Enterprise Encryption**: Provides support for data encryption using AES and RSA algorithms for securing sensitive data.
- **Transparent Data Encryption (TDE)**: Encrypts tablespaces to ensure that data at rest is protected from unauthorized access.
- **Key Management**: Uses keyring plugins to securely store encryption keys, supporting integration with external key management services.

(MySQL, Enterprise Encryption, n.d.)

Data Masking

MySQL Enterprise Edition includes data masking capabilities to protect sensitive information:

- Masks or obfuscates sensitive data in query results, preventing unauthorized users from viewing actual values.
- Useful for ensuring compliance with data privacy regulations.

(MySQL, MySQL Enterprise Data Masking and De-Identification, n.d.)

Security Plugins

MySQL offers a range of plugins to enhance security:

- **Authentication Plugins**: Extend MySQL's authentication capabilities with LDAP and PAM integration.
- **Audit Plugins**: Log database events for auditing and compliance.
- **Firewall Plugin**: Monitors and blocks unauthorized SQL queries to protect against injection attacks.

(MySQL, Security Components and Plugins., n.d.)

SELinux Support

For Linux environments, MySQL integrates with Security-Enhanced Linux (SELinux):

- Provides mandatory access control to restrict database operations based on predefined policies.
- Ensures that even root access does not bypass security controls.

(MySQL, SELinux, n.d.)

FIPS Mode

MySQL supports Federal Information Processing Standard (FIPS) mode for compliance with stringent security standards:

- Uses FIPS-validated cryptographic libraries to meet requirements for government and regulated industries.
- Ensures cryptographic operations align with approved standards.

(MySQL , FIPS Support, n.d.)

Practical Implementation

For a carpool application handling sensitive user data, MySQL's TLS/SSL encrypted connections ensure data protection during communication between the database and application. Features like Transparent Data Encryption (TDE) safeguard stored data, while data masking ensures that personal information is protected in reporting and debugging scenarios. The use of role-based access control and SELinux integration further strengthens access restrictions and system security.

4.2 PostgreSQL

PostgreSQL also known as Postgres, is a free and open-source relational database management system (RDBMS) emphasizing extensibility and SQL compliance. (contributors, PostgreSQL, n.d.)

4.2.1 General

We haven't used PostgreSQL much in class, but it has a similar approach to MySQL. Both are open-source and use SQL for creating databases and running queries.

Positive points	Negative points
Object-oriented features	Performance issues with complex queries
Open-source	Challenging installation
User-defined types and functions	Smaller community
Point in time recovery to restore to a specific moment	

(IBM, 2024)

(Scott, 2024)

(PostgreSQL, The PostgreSQL security, n.d.)

4.2.2 Security

PostgreSQL provides a comprehensive set of security features to protect data integrity, ensure confidentiality, and manage access control effectively. These features make it a robust choice for secure database implementations.

Authentication Methods

PostgreSQL supports multiple authentication methods to verify user identities and secure client connections:

- **Password-Based Authentication**:
  - Supports scram-sha-256 (recommended for its enhanced security) and md5.
  - Ensures secure handling of passwords during authentication.
- **GSSAPI Authentication**:
  - Enables single sign-on capabilities using Kerberos-based secure authentication.
- **Certificate Authentication**:
  - Uses SSL certificates to authenticate clients securely.
- **LDAP Authentication**:
  - Integrates with Lightweight Directory Access Protocol (LDAP) for centralized user management.
- **PAM Authentication**:
  - Utilizes Pluggable Authentication Modules (PAM) for flexible and external authentication.

(PostgreSQL, Authentication methods., 2024)

Access Control

PostgreSQL uses roles and privileges to implement fine-grained access control:

- **Role-Based Access Control (RBAC)**:
  - Roles can represent individual users or groups, and specific privileges can be assigned to these roles.
- **Granular Privileges**:
  - Access can be defined at the database, schema, table, or column level to restrict unauthorized access.

(PostgreSQL, Authentication methods., 2024)

Row-Level Security (RLS)

PostgreSQL supports **Row-Level Security (RLS)** to enforce access restrictions at the row level in tables:

- **Policy-Based Access Control**:
  - Allows defining conditions to determine which rows are accessible or modifiable by specific roles or users.
- **Multi-Tenant Applications**:
  - RLS is particularly useful for isolating data in multi-tenant environments.

(PostgreSQL, Row Security policies, 2024)

Encryption

PostgreSQL ensures data confidentiality through robust encryption mechanisms:

- **In-Transit Encryption**:
  - Uses SSL/TLS to encrypt communication between the client and the server, preventing interception during data transfer.
- **Password Encryption**:
  - Stores passwords securely using cryptographic hashing methods like SCRAM-SHA-256.
- **Column-Level Encryption**:
  - The pgcrypto extension provides encryption capabilities for sensitive data at the column level.

(PostgreSQL, Encryption options, 2024)

(PostgreSQL, Secure TCP/IP Connections with SSL, 2024)

Auditing and Logging

PostgreSQL includes logging and auditing tools to monitor database activity:

- **Query Logging**:
  - Tracks SQL queries, user connections, and other database events for monitoring and troubleshooting.
- **Enhanced Auditing**:
  - Extensions like pgaudit provide advanced auditing capabilities, enabling detailed tracking for compliance requirements.

(PostgreSQL, Error reporting and logging, 2024)

Security Updates and Vulnerability Management

PostgreSQL prioritizes security by promptly addressing vulnerabilities and providing updates:

- **Regular Updates**:
  - Security releases ensure known vulnerabilities are patched in a timely manner.
- **CVE Tracking**:
  - PostgreSQL assigns CVEs (Common Vulnerabilities and Exposures) to identified security issues to maintain transparency.

(PostgreSQL, Security Informatio, n.d.)

Practical Implementation

For a carpool application managing sensitive user data, PostgreSQL's Row-Level Security ensures proper isolation of user-specific data. TLS/SSL encryption protects communication between the application and database, while column-level encryption secures sensitive fields such as payment details. Role-based access control allows separation of privileges, ensuring that only authorized users have access to critical data.

4.3 MongoDB

MongoDB is a source-available, cross-platform, document-oriented database program. Classified as a NoSQL database product, MongoDB utilizes JSON-like documents with optional schemas. (contributors, MongoDB, 2024)

4.3.1 General

In our classes, we never used MongoDB. Unlike the other two databases we researched, MongoDB doesn't use SQL queries; instead, it works with JSON to store data and execute queries. Switching to a NoSQL database comes with a learning curve.

Positive points	Negative points
Data is very structured	Consumes more memory
Big data ready	Learning curve
High performance	Limited in large queries
Horizontally scalable	

(AWS, n.d.)

(MongoDB, n.d.)

(Simplilearn, 2023)

4.3.2 Security

MongoDB provides robust security measures to protect sensitive data throughout its lifecycle, including features for encryption, access control, network security, and auditing. These features align well with the security requirements for modern applications.

Data Encryption

MongoDB ensures data security using encryption in every stage:

- **In-Transit Encryption**:
  Data communication between clients and MongoDB servers is secured using **TLS/SSL**, preventing interception by unauthorized parties. (MongoDB, MongoDB Developer data platform with strong security capabilities, n.d.)
- **At-Rest Encryption**:
  MongoDB supports encrypted storage to protect data on disk. Encryption keys are managed using KMIP-compatible key management systems, ensuring secure key storage and rotation.

  (MongoDB, MongoDB Data Encryption, n.d.)
- **Client-Side Field-Level Encryption (CSFLE)**:
  CSFLE encrypts data before it is sent to the server, ensuring that sensitive fields remain encrypted even during storage and processing. Additionally, **Queryable Encryption** allows encrypted data to be searched without decryption, improving functionality while maintaining security.

Authentication Mechanisms

MongoDB offers versatile authentication methods to verify user and client identities:

- **SCRAM (Default)**:
  Uses Secure Challenge Response Authentication Mechanism for secure password-based authentication.
- **LDAP Integration**:
  Centralized user authentication via Lightweight Directory Access Protocol (LDAP).

- **x.509 Certificates**: Mutual SSL authentication allows secure verification using certificates for client-server communication.
- **OpenID Connect (OIDC)**: Allows integration with external identity providers for single sign-on.

  (MongoDB, Authentication on Self-Managed deployments., n.d.)

Authorization and Access Control

MongoDB implements **Role-Based Access Control (RBAC)** to ensure that users only have permissions necessary for their roles:

- Granular permissions can be defined at the database, collection, or even operation level.
- Predefined roles (e.g., read, readWrite, dbAdmin) streamline access control, but custom roles can also be created.

(MongoDB, Role-Based access control in Self-Managed deployments, n.d.)

Network Security

MongoDB secures network access with isolation and IP filtering:

- **IP Access Lists**: Connections are restricted to specific IP addresses or ranges, minimizing exposure to unauthorized access.
- **Virtual Private Cloud (VPC) Peering**: MongoDB supports private endpoints and VPC peering, enabling secure communication between the database and application within isolated network environments.

(MongoDB, MongoDB.com, n.d.)

Auditing and Monitoring

MongoDB provides detailed auditing capabilities to monitor database activity for security and compliance:

- Logs include user authentication attempts, data access, and modifications to sensitive collections.
- These features help meet regulatory requirements such as **GDPR** and **HIPAA**.

(MongoDB, Auditing Self-Managed deployments, n.d.)

MongoDB Atlas Security (Cloud-Specific)

For applications deployed on **MongoDB Atlas**, additional security measures include:

- Automated **TLS/SSL encryption** for all data in transit.
- Managed key rotation and encryption at rest using KMIP providers.
- IP whitelisting and private networking for enhanced protection against unauthorized access.
- Automatic security patching ensures compliance with evolving standards.

(MongoDB, MongoDB Atlas Security, n.d.)

Practical Implementation and Use Cases

For a carpool application that handles sensitive user information such as personal details and ride preferences, MongoDB's encryption and access control features are highly applicable. The use of CSFLE ensures data confidentiality, while RBAC ensures users have limited access based on their roles (e.g., admins vs. end users). Furthermore, MongoDB's network security options, such as VPC peering and IP access lists, are ideal for isolating the database from public exposure while allowing secure connections from trusted systems.

4.4 Security comparison table

Before deciding the winner using weighted ranking methods, we will compare the security features of each database service.

Criterion	MongoDB	MySQL	PostgreSQL
Authentication Methods	SCRAM, LDAP, x.509 certificates, OpenID Connect (OIDC)	Native password authentication, LDAP, PAM	SCRAM-SHA-256, GSSAPI, certificate authentication, LDAP, PAM
Access Control	Role-Based Access Control (RBAC) with granular permissions	Role-Based Access Control (RBAC) with privileges at global, database, table, and column levels	Role-Based Access Control (RBAC) with granular privileges and Row-Level Security (RLS)
Encryption In Transit	TLS/SSL	TLS/SSL with OpenSSL support	TLS/SSL
Encryption At Rest	Encrypted storage with KMIP-compatible key management systems	Transparent Data Encryption (TDE)	Column-level encryption with pgcrypto
Data Masking	Field-Level Encryption (CSFLE)	Enterprise-only data masking capabilities	Not natively supported
Auditing and Logging	Detailed logs for authentication and data access (audit logs in Enterprise)	General Query Log, Audit Plugins	Query logging, extensions like pgaudit for advanced auditing
Network Security	IP access lists, Virtual Private Cloud (VPC) peering	IP whitelisting, SELinux support	Integration with SELinux
Compliance Features	Supports GDPR and HIPAA compliance (Atlas)	FIPS mode for cryptographic operations	Supports compliance with CVE tracking and security releases
Best for Multi-Tenancy	Strong due to field-level encryption and VPC peering	Limited multi-tenancy capabilities	Strong with Row-Level Security (RLS)
Recommended Use Case	Applications requiring advanced encryption for sensitive data, e.g., client-side encryption	Applications needing flexible plugins and enterprise-level encryption	Scenarios requiring robust row-level security and open-source flexibility

4.5 Security – weighted ranking method

The Weighted Ranking Method (WRM) in this document evaluates database management systems based on their alignment with the OWASP Top 10 Web Application Security Risks. The OWASP (Open Web Application Security Project) is a globally recognized organization that identifies and promotes practices to mitigate the most critical security risks in applications. Its "Top 10" list highlights vulnerabilities such as A01:2021-Broken Access Control, A02:2021-Cryptographic Failures, and A09:2021-Security Logging and Monitoring Failures.

Databases are critical components of web applications and are often the target of attacks. To ensure a secure foundation, this WRM assesses databases—MongoDB, MySQL, and PostgreSQL—against criteria such as authentication methods, encryption, and access control. Each criterion is weighted based on its importance in addressing specific OWASP risks, ensuring the evaluation prioritizes robust security practices.

By basing this WRM on the OWASP Top 10, the analysis provides a systematic comparison of databases, helping developers choose the most secure and appropriate option for their web application requirements.

Criterion	**Weight**	**MongoDB Score**	**MySQL Score**	**PostgreSQL Score**
Authentication Methods	5	5	4	5
Access Control	5	4	4	5
Encryption In Transit	4	5	5	5
Encryption At Rest	4	5	5	4
Data Masking	3	5	4	2
Auditing and Logging	4	4	4	5
Network Security	3	5	4	5
Compliance Features	2	4	5	5
Total Weighted Score	**N/A**	139	130	137

Authentication Methods

- Relevance to OWASP: Related to A07:2021-Identification and Authentication Failures. Secure and flexible authentication mechanisms ensure only authorized users can access the database.
- Suitability: Essential, as robust authentication methods reduce the likelihood of unauthorized access.

Access Control

- Relevance to OWASP: Supports A01:2021-Broken Access Control. Granular permissions and role-based access help ensure users only access what they are authorized to see or modify.
- Suitability: Critical for enforcing least privilege principles and securing sensitive data.

Encryption In Transit

- Relevance to OWASP: Tied to A02:2021-Cryptographic Failures. Encrypting data in transit protects against interception during client-server communication.
- Suitability: High priority, especially for applications requiring secure communication over networks.

Encryption At Rest

- Relevance to OWASP: Also related to A02:2021-Cryptographic Failures. Ensures data stored on disk is protected, reducing risks if storage is compromised.
- Suitability: Highly relevant for databases storing sensitive user information.

Data Masking

- Relevance to OWASP: Supports A05:2021-Security Misconfiguration. Masking sensitive data ensures it is not exposed to unauthorized users or during testing.
- Suitability: Important for compliance with data protection regulations but secondary to other encryption methods.

Auditing and Logging

- Relevance to OWASP: Related to A09:2021-Security Logging and Monitoring Failures. Logs provide insights into database activity, helping to detect and respond to breaches.
- Suitability: Essential for compliance and monitoring, especially in regulated environments.

Network Security

- Relevance to OWASP: Related to A05:2021-Security Misconfiguration. Secure network configurations (e.g., IP whitelisting, private endpoints) prevent unauthorized access.
- Suitability: Vital for isolating the database from unauthorized external connections.

Compliance Features

- Relevance to OWASP: While not explicitly mapped to a single OWASP risk, compliance features align with multiple requirements, especially A02:2021-Cryptographic Failures and A09:2021-Security Logging and Monitoring Failures.
- Suitability: Highly relevant for organizations adhering to regulations like GDPR, HIPAA, or FIPS.

4.6 Conclusion – weighted ranking method

[not made]

5 GPS API

To generate the optimal route for the driver, we need an API capable of providing the best routes from point to point.

We researched five different APIs used for route generation. Some are popular because they have a native app, some are less well-known because they don't have a native app.

The following pages provide detailed research on each route API

5.1 Research

5.1.1 Google Maps Directions API

Functionality:

Supports route calculation with waypoints, traffic information, travel times, and route optimization.

Documentation:

https://developers.google.com/maps/documentation/directions/overview

Available App:

Google Maps is available for both iOS and Android and offers extensive features for navigation, traffic information, and public transportation.

Integration:

The Directions API allows you to open routes, waypoints, and even specific locations in the Google Maps app via deep linking. This allows your driver to send routes and stops from your application to the Google Maps app.

App Download:

Available on the App Store and Google Play Store.

Note:

Requires an API key and has pricing models, but offers free limits for smaller projects.

Pros:

- Highly accurate traffic data
- Multiple waypoints support
- Wide platform support
- Deep integration options

Cons:

- Costs
- Dependency on Google

(Google, n.d.)

(Google, Directions API overview, n.d.)

(Google, Directions API usage and billing, n.d.)

(Google, Get started, n.d.)

5.1.2 Waze Transport SDK

Functionality:

Provides navigation capabilities with real-time traffic information and user-driven reporting (such as accidents and delays).

Documentation:

https://developers.google.com/waze/intro-transport

Available App:

Waze is also available on iOS and Android, with unique community-based traffic information such as traffic jam, hazard, and speed trap notifications.

Integration:

The Waze Transport SDK allows apps to open routes in the Waze app, allowing a driver to follow the planned route in Waze with a simple button, including any waypoints.

App Download:

Available on the App Store and Google Play Store.

Note:

Waze SDK has limitations when it comes to routes with multiple waypoints, so it may be useful in conjunction with another API.

Alternative for Waypoints:

We can consider using Waze for individual segments of the route, opening each stop one by one. Another option is to first optimize the calculated route with another API (such as Google Maps) and then send the individual segments of the route to Waze.

The following table lists the capabilities of the Waze Transport SDK and Waze Deep Links:

	Waze Transport SDK	Waze Deep Links
Start a drive from your app	✔	✔
Trigger a search in Waze from your app	✔	✔
Set the Waze map to a specific location from your app	✔	✔
Open Waze from your mobile website	✖	✔
Waze button that brings users back to your app	✔	✖
Data sent to your app from Waze: • ETA, sent to both driver and passenger apps. • Route points (latitude, longitude) • Next turn instruction (right, left, u-turn) • Distance to next turn	✔ This feature can be disabled.	✖

Figure 5: list of capabilities of the Waze Transport SDK and Waze Deep Links

Pros:

- Community-driven traffic information
- Real-time updates
- Easy integration for drivers

Cons:

- Limited waypoint support
- Limited customization
- Reliant on user data

(Google, About the Waze Transport SDK, n.d.)

5.1.3 Mapbox Directions API

Functionality:

Supports calculating routes with multiple stops, and has a customizable map style that is visually appealing.

Documentation:

https://docs.mapbox.com/api/navigation/directions/

Available App:

Mapbox does not provide a direct GPS app like Google Maps or Waze, but the Mapbox SDKs and APIs are often used by other apps for navigation.

Note:

In our case, Mapbox would be an option for building a custom navigation experience within our own app, rather than an app that users can already download.

(Mapbox, n.d.)

(Mapbox, Waypoints, n.d.)

(Mapbox, Mapbox API pricing, n.d.)

5.1.4 Here maps API

Functionality:

Supports calculating routes with multiple stops, and has a customizable map style that is visually appealing.

Documentation:

https://www.here.com/docs/category/routing-api-v8

Available App:

HERE WeGo, the navigation app from HERE Technologies, provides maps, turn-by-turn navigation and traffic information.

Integration:

While less directly usable than Google Maps or Waze, the HERE Maps API supports integration with the HERE WeGo app via URL schemes and deep linking, allowing a planned route with waypoints to be opened in the app.

App download:

Available on the App Store and Google Play Store.

(HERE, n.d.)

(HERE, Technologies Documentation, n.d.)

(HERE, Wego, n.d.)

5.1.5 OSRM open street map

Functionality:

OSRM is an open-source route calculation tool that uses OpenStreetMap data and supports multiple stops.

Documentation:

https://project-osrm.org/docs/v5.24.0/api/#

Available App:

OSRM (Open Source Routing Machine) and OpenStreetMap are open-source and can enable their own navigation system, but do not have an accompanying GPS app. However, you could use these to integrate a fully customized navigation experience within your application.

Note:

May require some technical knowledge to set up and maintain, but offers complete freedom.

(OSRM, n.d.)

5.2 In summary

Google Maps, Waze, and HERE WeGo offer both an API for integration and a companion GPS app for users.

Feature/Capability	Google Maps Directions API	Waze Transport API	Mapbox Directions API	HERE Maps API	OSRM
Start navigation from app	✅	✅	✅	✅	❌ Custom implemantation required
Multiple waypoints support	✅	❌ Limited	✅	✅	✅
Real-Time traffic updates	✅	✅	✅	✅	❌
Custom map styling	❌ Limited	❌	✅	✅	✅
Community driven traffic information	❌	✅	❌	❌	❌
Native GPS app available	✅	✅	❌	✅	❌
Price flexibility for large projects	❌ Limited (Paid with free tier)	N/A	✅	❌ Limited	✅ Free and open-source
Integration with other services	✅	❌	✅	✅	✅
Geofencing and asset tracking	❌	❌	✅	✅	❌
Open-source	❌	❌	❌	❌	✅

5.2.1 Google Maps Directions API

Ease of integration with other applications:

Google Maps is designed with extensive APIs that make integration with other applications straightforward, including routes, waypoints, and location-based services.

Documentation:

https://developers.google.com/maps/documentation/directions/start

Multiple waypoints support:

Google Maps Directions API supports multiple waypoints on a route, making it suitable for applications that require complex routing.

Documentation: https://developers.google.com/maps/documentation/directions/overview#Waypoints

Customizability for specific use cases:

Limited customization options for map styling, but allows flexible routing and can be customized to a degree through URL parameters.

Documentation:

https://developers.google.com/maps/documentation/directions/start

Availability of a native app:

Google Maps has a well-established native app on both iOS and Android that can be launched directly from the API.

Documentation:

https://developers.google.com/maps/documentation/urls/get-started

Reliability and uptime:

Google Maps is known for its high reliability and uptime, supported by Google's robust infrastructure.

Documentation: General platform support is discussed at

https://cloud.google.com/maps-platform

Ease of obtaining and managing API keys:

Requires a Google Cloud Console account for managing API keys, which provides a user-friendly interface for generating and securing keys.

Documentation: https://developers.google.com/maps/gmp-get-started

5.2.2 Waze transport SDK

Ease of integration with other applications:

Waze SDK offers basic integration capabilities, allowing apps to launch Waze with specific destinations, though it is more limited in flexibility compared to Google Maps.

Documentation:

https://developers.google.com/waze/intro-transport

Multiple waypoints support:

Waze SDK has limited support for multiple waypoints, primarily handling start and end locations with limited intermediate stops.

Customizability for specific use cases:

Customization options are minimal, as Waze primarily supports basic navigation features without extensive styling or routing modifications.

Documentation:

https://developers.google.com/waze/faq

Availability of a native app:

Waze has a popular native app that can be opened from other applications via the SDK or deep links.

Documentation:

https://developers.google.com/waze/intro-transport

Reliability and uptime:

Waze provides reliable navigation with community-driven data but may not be as consistent as Google Maps for non-community data.

Documentation:

General reliability is inferred, but specific uptime documentation is not available. Waze is supported by Google's infrastructure.

Ease of obtaining and managing API keys:

Waze SDK requires API keys from Google's developer console, making it relatively straightforward to obtain and manage.

Documentation:

https://developers.google.com/waze/intro-transport

5.2.3 Mapbox Directions API

Ease of integration with other applications:

Mapbox offers extensive SDKs and APIs that facilitate integration with applications, including custom maps and navigation features.

Documentation:

https://docs.mapbox.com/api/navigation/directions/

Multiple waypoints support:

Mapbox supports multiple waypoints and allows for complex routing options, making it suitable for custom navigation applications.

Documentation:

https://docs.mapbox.com/api/navigation/directions/#waypoints

Customizability for specific use cases:

High level of customization for map styling, route display, and interaction, providing flexibility for unique use cases.

Documentation:

https://docs.mapbox.com/api/maps/styles/

Availability of a native app:

Mapbox does not have its own GPS app but is often used by third-party applications that leverage its maps and navigation tools.

Documentation:

N/A – Mapbox is primarily used as an embedded service.

Reliability and uptime:

Mapbox provides strong reliability and uptime, designed to support commercial applications and backed by a dedicated infrastructure.

Documentation:

https://docs.mapbox.com/api/maps/#service-level-agreements

Ease of obtaining and managing API keys:

Mapbox offers a straightforward interface for obtaining and managing API keys through the Mapbox dashboard.

Documentation:

https://docs.mapbox.com/help/getting-started/access-tokens/

5.2.4 HERE Maps API

Ease of integration with other applications:

HERE provides robust APIs for integration, including detailed routing, location, and asset-tracking services.

Documentation:

https://developer.here.com/documentation

Multiple waypoints support:

HERE Maps API supports multiple waypoints, allowing for complex route planning.

Documentation:

https://developer.here.com/documentation/routing-api/

Customizability for specific use cases:

Allows for detailed customization options, including routing parameters, traffic analysis, and map styles, suitable for specific needs.

Documentation:

https://developer.here.com/documentation

Availability of a native app:

HERE WeGo is a native app by HERE Technologies that supports routing and navigation and can be integrated with HERE APIs.

Documentation:

https://wego.here.com/

Reliability and uptime:

HERE provides reliable services for commercial applications, with a focus on enterprise-grade reliability and SLAs.

Documentation: https://developer.here.com/documentation

Ease of obtaining and managing API keys:

HERE offers a developer portal where users can easily register, obtain, and manage API keys.

Documentation:

https://developer.here.com/getting-started

5.2.5 OSRM (Open Source Routing Machine)

Ease of integration with other applications:

OSRM requires more technical setup but can be integrated into custom applications for routing functionality using OpenStreetMap data.

Documentation:

http://project-osrm.org/

Multiple waypoints support:

OSRM supports multiple waypoints and allows for flexible routing solutions, though it may require additional setup.

Documentation:

http://project-osrm.org/docs/v5.24.0/api/#route-service

Customizability for specific use cases:

Highly customizable as an open-source solution, allowing users to modify the source code to fit specific needs.

Documentation:

https://github.com/Project-OSRM/osrm-backend

Availability of a native app:

OSRM does not have a native app; it is designed to be embedded into custom solutions.

Documentation:

N/A – OSRM is an open-source routing engine without a standalone app.

Reliability and uptime:

Reliability depends on the server infrastructure used, as OSRM is self-hosted. No centralized SLA.

Documentation:

N/A – Open-source; uptime is based on hosting setup.

Ease of obtaining and managing API keys:

OSRM does not require API keys, as it is open-source and self-hosted, making it flexible for internal use.

Documentation: https://github.com/Project-OSRM/osrm-backend

5.3 Comparison table

Criteria	Google Maps Directions API	Waze Transport SDK	Mapbox Directions API	HERE Maps API	OSRM (Open Source Routing Machine)
Ease of integration with other applications	Extensive integration capabilities, suitable for various apps.	Basic integration; launches Waze with destinations.	Highly customizable integration with SDKs.	Robust APIs for integration with detailed features.	Requires technical setup, designed for custom use cases.
Multiple waypoints support	Full support for multiple waypoints.	Limited; only start and end points supported.	Supports multiple waypoints and complex routing.	Full support for multiple waypoints.	Supports multiple waypoints, flexible but requires setup.
Customizability for specific use cases	Limited map styling, but flexible routing options.	Minimal customization options.	High customization for maps and routes.	Detailed customization for routing, traffic analysis.	Open-source, allows code modifications for custom use.
Availability of a native app	Google Maps app available for iOS and Android.	Waze app available with SDK support.	No native app; typically embedded in custom apps.	HERE WeGo app available, integrates with HERE API.	No native app; designed for custom embedding.
Reliability and uptime	High reliability, supported by Google infrastructure.	Reliable but community-driven data may vary. Supported by Google infrastructure.	Commercial-grade reliability with SLA.	Enterprise-grade reliability, suitable for commercial use.	Dependent on self-hosting setup; no centralized SLA.
Ease of obtaining and managing API keys	User-friendly interface through Google Cloud Console.	Managed via Google Developer Console.	Simple key management through Mapbox dashboard.	Easy key management through HERE developer portal.	No API keys required, as it is open-source and self-hosted.

5.4 Conclusion – weighted ranking method

Feature	Weight	Google Maps	Waze	Mapbox directions API	Here Maps API	OSRM
Ease of integrations with other applications						
Multiple waypoints support						
Customizability for specific use cases						
Availability of a native app						
Reliability and uptime						
Ease of obtaining and managing API keys						

6 AI

Integrating AI into the carpool application requires careful planning to ensure seamless interaction between the **backend** AI components and the **frontend** user experience. This document outlines the necessary steps to implement and integrate AI for clustering, recommendation, and route optimization, detailing how each part will interact with the overall system architecture, including the backend and frontend components.

6.1 Step-by-Step Plan for the AI Component

6.1.1 Define AI Components and Functionalities

The AI component will be divided into several modules, each responsible for specific functionalities:

- **User Clustering**: Using **K-means clustering** or **DBSCAN** to group users based on location and preferences.
- **Carpool Recommendations**: Developing a **recommendation engine** to suggest suitable carpool partners or events.
- **Pickup and Route Optimization**: Implementing a routing algorithm to determine the optimal pickup points for each carpool.

These modules will work independently but communicate effectively through APIs to ensure seamless integration.

6.1.2 Development of AI Models

- **Data Preprocessing**: Clean and prepare user and event data, including converting addresses to geographic coordinates.
- **Model Training**:
  - Develop **K-means clustering** for grouping users.
  - Use **content-based filtering** and **collaborative filtering** to develop a hybrid recommendation system.
- **Model Evaluation**: Test and evaluate models using metrics such as **silhouette score** for clustering and **precision/recall** for recommendations.

6.1.3 Backend Integration

The backend will serve as the intermediary between the AI models and the frontend application. The backend will manage **user data**, **requests**, and **AI predictions** to ensure smooth user interactions.

- **Technology Stack**: Use **FastAPI** or **Flask** to create RESTful APIs that will interact with the AI models.
- **APIs for AI Models**:
  - **User Grouping API**: An endpoint that takes user data as input and returns group assignments.
  - **Route Optimization API**: An endpoint to determine optimal pickup points and return the recommended route for the driver.
- **Data Flow**: The backend will receive user data from the frontend, process it using the AI models, and then return the results through the API endpoints.

6.1.4 Frontend Integration

The frontend will handle all user interactions and visualizations, allowing users to access the carpooling features.

- **UI Elements**:
  - **User Profiles**: Allow users to input preferences, availability, and locations.
  - **Carpool Recommendations**: Show **recommended carpool partners,** events, and pickup points based on AI suggestions.
  - **Map Integration**: Display routes and pickup points using map APIs like **Google Maps**

6.1.5 Workflow Overview

1. **User Registration and Input**: Users provide data, such as location, preferences, and carpooling role, through the frontend UI.
2. **Data Submission**: This data is sent to the backend through an API.
3. **AI Processing**:

a. The **clustering model** groups users based on their proximity.

b. The **recommendation engine** provides suggestions for carpool partners or events.

c. The **route optimization** algorithm determines the best route for pickups.

4. **API Response**: The backend returns the clustering, recommendations, and route details to the frontend.
5. **User Visualization**: The frontend displays the results, including carpool partners and optimized routes, through the user interface.

6.1.6 Technologies and Tools Required

- **Backend**:
  - **FastAPI/Flask**: To develop RESTful APIs.
  - **pandas, scikit-learn, geopy**: For data manipulation, clustering, and geographic calculations.
  - **PostgreSQL/MySQL**: To store user information and carpool data.
- **Frontend**:
  - **Angular**: For creating an interactive and responsive user interface.
  - **JavaScript**: To make API requests and handle user data.
  - **Map API**: To visualize routes and user locations.

6.1.7 Challenges and Considerations

- **Real-Time Updates**: Users may change their preferences or availability. Implement **websockets** or **polling** to provide real-time updates.
- **Scalability**: As the number of users grows, ensure that the clustering and recommendation models can scale effectively.
- **Security and Privacy**: Ensure all personal data is handled securely to comply with **GDPR** regulations.

6.1.8 Conclusion

Integrating AI into the carpool application involves the development of clustering and recommendation models, setting up an efficient backend for API management, and connecting the backend to a responsive frontend. By combining these components, the application will provide users with an intelligent and user-friendly carpool experience, optimizing partner suggestions and routes to make commuting more efficient and environmentally friendly.

(Adjiman, 2020)

6.2 Report on AI Components for Carpooling Algorithm

Carpooling has emerged as a sustainable transportation solution, reducing costs and environmental impact. Our project aims to design an intelligent carpooling application for Axxes, with a primary focus on developing an AI-driven algorithm to select the best and fastest paths for drivers. This document identifies the core AI components required to build the algorithm, emphasizing trip time optimization rather than distance.

6.2.1 Ai components

In this section, I will explore the AI components planned for the project, detailing how they function and the value they bring to ensure the success of our project.

Route Optimization

Objective: Calculate the fastest routes between waypoints (pickup locations and the destination).
Approach:

- Use graph algorithms like Dijkstra's or A* to find optimal paths (NetworkX, n.d.).
- Google Maps API for real-time traffic data integration (Google Maps API, n.d.).

Outcome: Optimized routes to minimize total travel time for all participants.

Travel Time Estimation

Objective: Predict travel times accurately using historical and real-time data.
Approach:

- Use real-time traffic data from APIs to adjust travel times (Google Maps API, n.d.).
- Train machine learning models (e.g., regression models or neural networks) to predict travel times based on features like time of day, road type, and traffic density

(Scikit-learn, n.d.).

Outcome: Improved decision-making by providing accurate travel time predictions.

Dynamic Pickup Sequencing

Objective: Determine the most time-efficient sequence for picking up colleagues.
Approach:

- Model the problem as a Vehicle Routing Problem (VRP) with time constraints.
- Use optimization techniques like Genetic Algorithms or Simulated Annealing for sequencing (Google OR-Tools, n.d.).

Outcome: Minimized total trip time while considering multiple pickup points and constraints.

Real-Time Updates

Objective: Adapt routes dynamically based on unforeseen circumstances such as traffic jams or delays.
Approach:

- Implement an event-driven system to detect changes in traffic conditions.
- Re-compute routes dynamically using real-time traffic data (Flask, n.d.; Requests, n.d.).

Outcome: Enhanced reliability and responsiveness of the carpooling system.

Scalability

Objective: Ensure the system operates efficiently with up to 300 users.
Approach:

- Optimize graph representation (e.g., adjacency lists) for efficient route calculations (OSMNX, n.d.).
- Use Python libraries like NumPy for matrix operations and tools like multiprocessing for parallel computation (NumPy, n.d.).

Outcome: A scalable system capable of handling high user volumes effectively.

Data Handling

Objective: Manage input and historical data for efficient processing.
Approach:

- Preprocess input data to standardize formats (e.g., geocoding addresses) (GeoPandas, n.d.; Fiona/Shapely, n.d.).
- Store travel data in a relational database for future use.

Outcome: Robust data management ensures accurate and efficient processing.

Simulation and Testing

Objective: Validate the algorithm in various scenarios to ensure reliability.
Approach:

- Simulate diverse carpooling scenarios using synthetic or historical datasets (Faker, n.d.; SimPy, n.d.).
- Test for edge cases like overlapping pickup points or extreme traffic conditions.

Outcome: A well-tested and reliable algorithm.

6.2.2 Potentioal Tools and Libraries

Route Optimization

- **NetworkX:** For graph-based calculations (e.g., Dijkstra's, A*).
- **OSMNX:** For downloading and modeling real-world street networks using OpenStreetMap data.
- **Google Maps API:** For real-time traffic and routing data.

Travel Time Estimation

- **Pandas:** For handling and preprocessing data.
- **Scikit-learn:** For training regression models (if using historical data to predict travel times).
- **NumPy:** For efficient numerical computations.

Dynamic Pickup Sequencing

- **OR-Tools (Google Optimization Tools):** For solving the Vehicle Routing Problem (VRP) and handling constraints like time optimization.
- **SimPy:** For simulating dynamic systems and testing routing behavior.

Real-Time Updates

- **Flask/Django:** For building a backend API to fetch and process real-time traffic updates.
- **Celery + Redis:** For scheduling and handling periodic updates to route information.
- **Requests:** For making API calls to traffic data providers.

Scalability

- **Multiprocessing (Python standard library):** For parallelizing computations across multiple cores.
- **PySpark:** For distributed computing if handling very large datasets.
- **Database tool:** For storing preprocessed and historical routing data.

Data Handling and Geocoding

- **GeoPandas:** For handling geospatial data like latitude/longitude.
- **Fiona/Shapely:** For spatial data processing and manipulation.
- **Google Geocoding API or Nominatim (OpenStreetMap):** For converting addresses into latitude/longitude coordinates.

Simulation and Testing

- **Matplotlib/Seaborn:** For visualizing routes and testing results.
- **pytest:** For automated testing of individual components.
- **Faker:** For generating synthetic datasets with plausible addresses and user data.

General Development Tools

- **Jupyter Notebook:** For prototyping and data exploration.
- **VS Code:** For efficient coding and debugging.

- **Git/GitHub:** For version control and collaboration.

6.2.3 Conclusion

By integrating these AI components, we can develop a sophisticated algorithm tailored to Axxes's needs. This system will prioritize minimizing travel time while accommodating user preferences and adapting to real-time changes. With robust data handling and scalable architecture, the carpooling application will provide a seamless and efficient solution for Axxes employees.

6.3 Clustering Techniques

Clustering is a fundamental unsupervised learning technique in machine learning that groups data points based on shared characteristics. This approach plays a crucial role in optimizing carpooling applications by forming user groups with similar interests or geographical proximity, facilitating efficient carpooling. This research explores various clustering algorithms, evaluates their suitability, and identifies the most effective technique for improving user group formation and route optimization.

Clustering algorithms aim to segment data into meaningful groups, identifying similarities among users within a cluster compared to those outside it. This method is essential for creating carpooling groups by clustering drivers and passengers based on location or shared travel routes. The following sections review different clustering techniques and evaluate their potential for application in carpooling systems.

6.3.1 K-Means Clustering

K-Means is a widely used clustering algorithm known for its simplicity and efficiency. It partitions data into K clusters by iteratively assigning data points to the nearest cluster centroid and updating the centroids until they stabilize.

(Santos, 2020)

Benefits:

- **Efficient and Scalable**: Handles large datasets effectively, making it ideal for extensive urban user bases.
- **Straightforward Implementation**: Works well with compact and well-separated clusters.

Challenges:

- **Predefined Number of Clusters**: Determining the value of K in advance can be challenging when the number of carpool groups is unknown.
- **Sensitivity to Outliers**: Outliers can distort the clustering outcome, particularly in sparsely populated areas.

Relevance to Project: K-Means is an appropriate method for grouping users by geographical coordinates. Enhancing the algorithm with dynamic K determination can improve adaptability to changing user data.

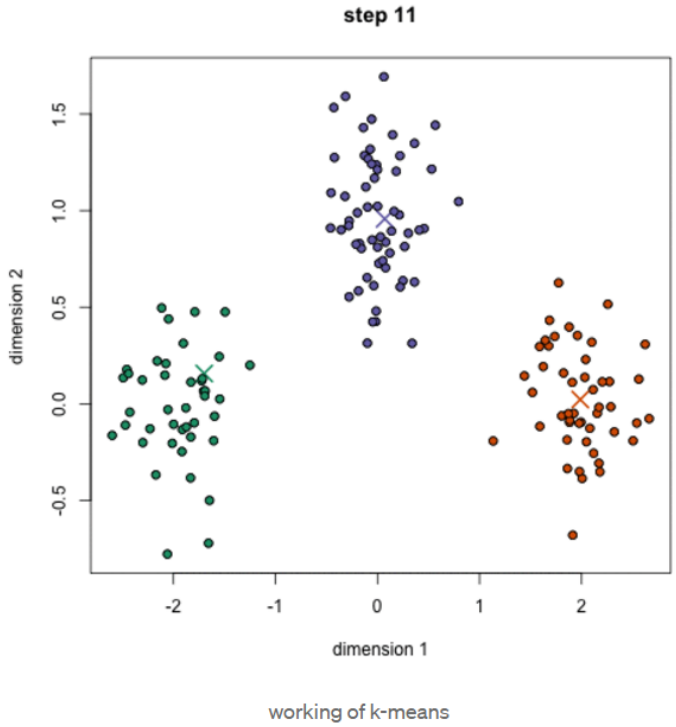

Figure 6: Example of using K-Means Clustering

(Chatterjee, 2019)

6.3.2 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN, or Density-Based Spatial Clustering of Applications with Noise, identifies dense regions as clusters and labels sparse areas as outliers.

Benefits:

- **Automatic Cluster Determination**: Clusters are formed based on data density, eliminating the need to specify their number beforehand.
- **Outlier Detection**: Effectively identifies and excludes outliers, beneficial for managing users in remote or atypical carpool locations.

Challenges:

- **Parameter Sensitivity**: Performance depends heavily on parameter choices like epsilon (radius) and minimum points.
- **Inconsistent Density Issues**: Struggles to handle clusters with varying densities, potentially leading to inaccurate groupings.

Relevance to Project: DBSCAN is well-suited for clustering users in sparsely populated or unevenly distributed areas, excluding outliers.

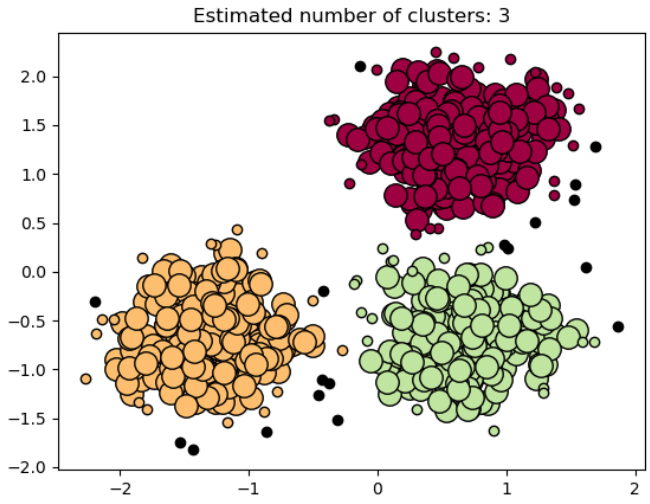

Figure 7: Example of clustering using DBSCAN :

(Scikit-learn, 2024)

6.3.3 Hierarchical Clustering

Hierarchical clustering constructs a nested tree structure of clusters, starting from individual data points and merging them upwards (agglomerative) or splitting clusters downwards (divisive).

Benefits:

- **Visual Relationships**: Offers a dendrogram to display cluster relationships.
- **No Predefined Cluster Requirement**: Unlike K-Means, the number of clusters doesn't need to be specified in advance.

Challenges:

- **Scalability Constraints**: Computationally intensive, making it less practical for large datasets.

Inflexibility: Merged or split clusters cannot be altered once formed.

- **Relevance to Project**: Hierarchical clustering is less suitable for a growing user base due to scalability issues but can be useful during initial exploratory analysis.

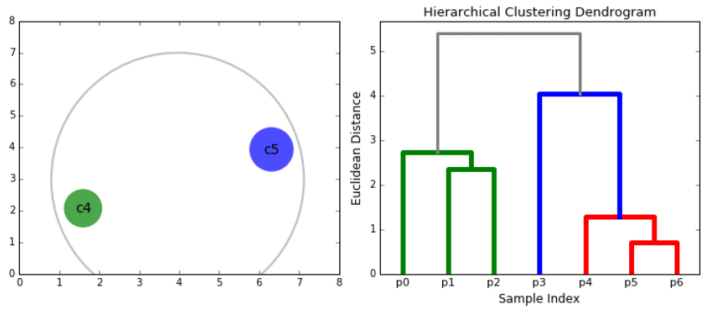

Figure 8: Example of Hierarchical Clustering

(Chatterjee, 2019)

6.3.4 Gaussian Mixture Models (GMM)

GMM uses a probabilistic approach to model data as a mixture of Gaussian distributions. It employs Expectation-Maximization (EM) for optimal parameter estimation.

Benefits:

- **Flexible Cluster Shapes**: Handles elliptical clusters, providing more flexibility than K-Means.
- **Probabilistic Assignments**: Assigns probabilities for data points belonging to different clusters, useful in cases of overlapping groups.

Challenges:

- **Complexity**: More computationally demanding compared to simpler methods like K-Means.
- **Cluster Number Requirement**: Similar to K-Means, GMM requires predefined cluster numbers.

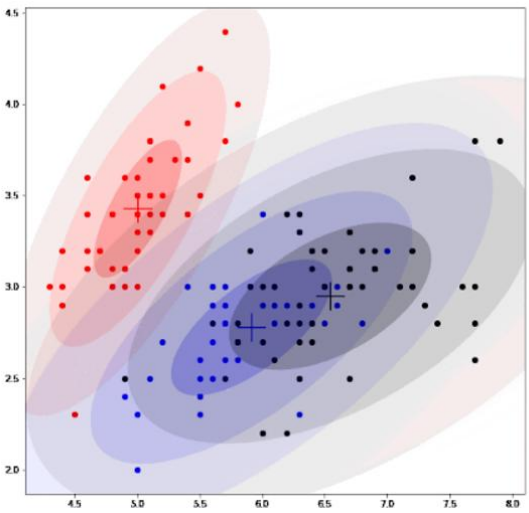

Figure 9: Example of Gaussian Mixture

(Carrasco, 2024)

- **Relevance to Project**: GMM is advantageous for identifying clusters with non-circular shapes, though its complexity may hinder real-time application.

6.3.5 Comparison table

Technique	Pros	Cons	Suitability
K-Means	Fast, simple , scalable	Needs predefined K, sensitive to outliers	**Suitable** for large datasets with adjustable K.
DBSCAN	Handles noise/outliers, no K required	Sensitive to parameter choices	**Suitable** for uneven user distributions and outlier handling.
Hierarchical	Visual relationships, no K	Poor scalability	**Less suitable** due to scalability limitations.
GMM	Flexible cluster shapes	Computationally complex	**Possibly suitable** for non-spherical clusters but complex.

For the carpooling application, K-Means is identified as the most effective clustering algorithm due to its simplicity, scalability, and ease of implementation. It is especially suited for grouping users based on geographical coordinates. DBSCAN may also be valuable for managing irregular user distributions or excluding outliers. Future work includes testing both algorithms in real-world scenarios and dynamically adapting parameters to user data variations. Implementing robust clustering methods will enhance group formation, reduce travel distances, and promote sustainable commuting practices.

6.3.6 Weighted Ranking Method

Criteria	**Weight**	**K-Means**	**DBSCAN**	**Hierarchical**	**GMM**
Scalability	5	5	4	2	3
Handling Outliers	4	4	5	1	3
Flexibility (Cluster Shapes)	3	3	3	3	5
Implementation Complexity	4	5	3	2	2
Suitability for Uneven Data	5	5	5	2	3
Total		**95**	**86**	**41**	**65**

The decision matrix highlights K-Means as the top choice for its scalability, simplicity, and adaptability to large datasets. DBSCAN ranks second, excelling in outlier handling and uneven data. GMM is flexible but computationally complex, and hierarchical clustering struggles with scalability and flexibility, making it less practical for the project.

6.4 About Google API

Google Maps API is a powerful tool for building location-based services. For the Axxes carpooling application, prioritizing travel time over distance is crucial. This report explores how Google Maps API can be used to achieve this goal.

6.4.1 Overview

Google Maps API offers several services to facilitate route planning, traffic analysis, and travel time estimation. Key APIs for this project include:

1. **Directions API**
    - Calculates routes, providing options for the shortest or fastest paths.
2. **Distance Matrix API**
    - Returns travel times and distances for multiple origins and destinations.
3. **Traffic Data Integration**
    - Real-time and historical traffic data adjust estimated travel times dynamically.

6.4.2 Prioritizing Travel Time Over Distance

Key Features of Google Directions API

- **travelMode Parameter:** Specifies the mode of transport (driving, walking, etc.). For carpooling, driving is the default.
- **avoid Parameter:** Allows avoiding specific road types (e.g., toll roads or highways).
- **traffic_model Parameter:** Used with departure times to adjust estimates based on traffic patterns:
    - best_guess: Combines real-time and historical traffic data.
    - pessimistic: Assumes heavier traffic conditions.
    - optimistic: Assumes lighter traffic conditions.

Approach to Prioritize Travel Time

1. Use the **Directions API** with optimizeWaypoints=true to calculate the fastest route, including stops.
2. Combine real-time traffic data using the **Distance Matrix API** to estimate travel times dynamically.
3. Incorporate departure times to account for varying traffic conditions.

(Google, Get started, n.d.)

6.4.3 Advantages of using Google Maps API

1. **Accuracy:** Combines real-time and historical traffic data for precise travel time estimation.
2. **Ease of Integration:** Simple Python libraries and REST API calls.
3. **Scalability:** Handles multiple origin-destination pairs simultaneously.

(Google, Get started, n.d.)

6.4.4 Challenges and Solutions

Challenge	Solution
High API Costs for Large Scale	Optimize API calls by batching requests and using cached data where possible. (Google Maps API, n.d.).
Traffic Variability	Use best_guess with frequent updates for accurate predictions. (Google Distance Matrix API, n.d.).
Rate Limits	Monitor usage and implement retries for rate-limited requests. (Google Maps API, n.d.).

6.4.5 Conclusion

Google Maps API provides robust tools for implementing a travel time-focused carpooling algorithm. By leveraging the Directions API and Distance Matrix API, combined with real-time traffic data, the Axxes carpooling application can efficiently prioritize minimizing travel time for drivers and passengers.

6.5 Research on Recommendation Engines

Recommendation engines are a crucial element in many modern applications, used to personalize and improve user experiences. In the context of the carpool application project, recommendation systems can be used to suggest potential carpool partners, events, and optimized routes based on user preferences and historical data. This research will explore different types of recommendation systems, their methodologies, and how they can be applied to enhance the user experience in the carpool application.

Recommendation engines can be broadly categorized into three types: **Collaborative Filtering**, **Content-Based Filtering**, and **Hybrid Approaches**. Each approach has its strengths and weaknesses, and their applicability will be discussed with a focus on the carpool project.

6.5.1 Colaborative filtering

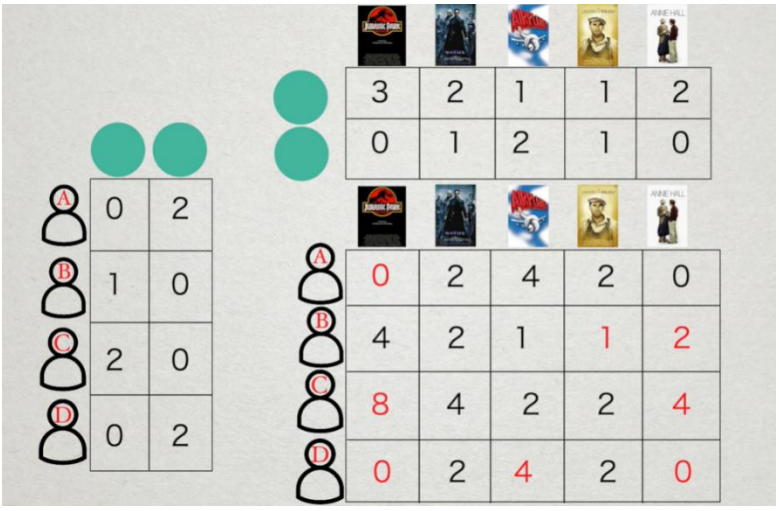

Figure 10: Illustration on Collaborative filtering

Collaborative Filtering (CF) is a widely used technique in recommendation systems that relies on user interactions and preferences to suggest items or matches. CF works by finding patterns among multiple users to predict what a particular user might like, based on similar users' behavior.

- **Types of Collaborative Filtering**:
  - **User-Based Collaborative Filtering**: Recommends items based on the preferences of similar users. In the carpool context, users with similar carpool partners or commuting patterns may receive suggestions to connect with each other.
  - **Item-Based Collaborative Filtering**: Recommends items (e.g., users) based on their similarity to items the user has interacted with in the past. For example, if a user has successfully carpooled with one person, they may be recommended similar people to join their carpool.
- **Advantages**:
  - **Personalization**: Offers personalized recommendations by learning from user behavior.
  - **Scalability**: Works well with a large user base, making it ideal for carpool apps as the user base grows.
- **Challenges**:
  - **Cold Start Problem**: Requires sufficient user interaction data, which may be challenging for new users with limited activity history.
  - **Data Sparsity**: If there are too many items or users but few interactions, the system may struggle to find meaningful recommendations.
- **Suitability for Project**: Collaborative filtering can be beneficial for suggesting **potential carpool partners** based on past carpool activities or user preferences. However, the **cold start problem** must be addressed by collecting enough initial data.

(Art of the problem, n.d.)

6.5.2 Content-based Filtering

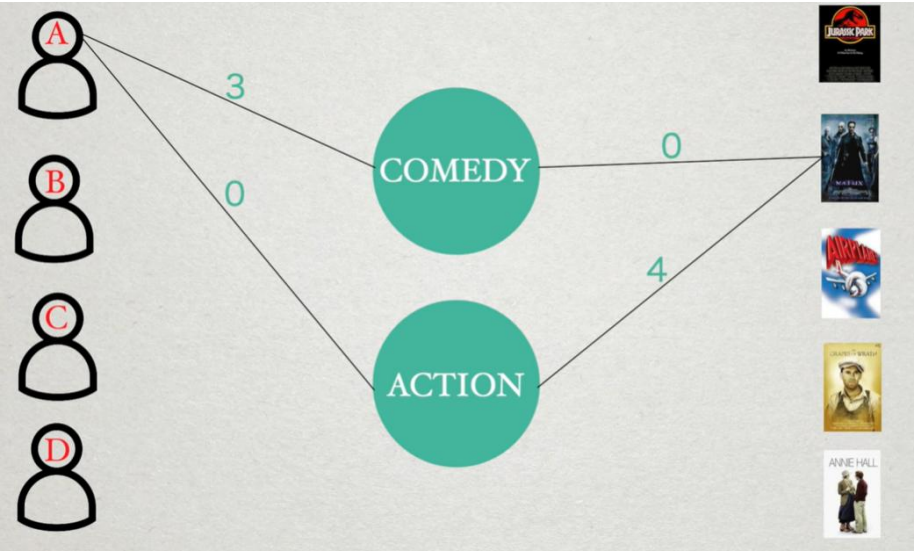

Figure 11 Ilustration on Content-based Filtering

Content-Based Filtering (CBF) recommends items based on the attributes of the items themselves and the user's preferences. In this approach, the system learns the user's likes and dislikes and recommends items with similar features.

- **Features Used for Content-Based Filtering**:
  - **User Preferences**: Attributes like preferred music, smoking/non-smoking, availability, etc., can be used to find suitable matches.
  - **Event Similarity**: For suggesting events, users who attended similar types of events may be recommended similar future gatherings.
- **Advantages**:
  - **Cold Start Resistant**: Unlike collaborative filtering, CBF can still recommend items to new users based on their stated preferences.
  - **Interpretability**: Easy to explain why a particular recommendation was made (e.g., recommending carpool partners with similar preferences).
- **Challenges**:
  - **Limited Novelty**: Recommends items similar to those already liked, which may limit the discovery of new or diverse options.
  - **Feature Engineering**: Requires detailed feature extraction to work effectively, which can be time-consuming.
- **Suitability for Project**: Content-based filtering is suitable for recommending carpool matches when **user preferences** are explicitly available. It can help new users find suitable matches quickly, based on their provided information.

(Art of the problem, n.d.)

6.5.3 Hybrid Approaches

Hybrid recommendation systems combine the strengths of collaborative filtering and content-based filtering to overcome their respective limitations. They leverage multiple sources of information to provide more accurate and diverse recommendations.

- **Examples of Hybrid Techniques**:
  - **Weighted Hybrid**: Combines CF and CBF, assigning weights to each based on their reliability.
  - **Switching Hybrid**: Switches between CF and CBF depending on the data available for each user (e.g., using CBF for new users and CF for experienced users).
- **Advantages**:
  - **Improved Accuracy**: Provides more precise and balanced recommendations by using multiple data sources.
  - **Handles Cold Start**: Alleviates the cold start problem of CF by integrating content-based methods.
- **Challenges**:
  - **Complexity**: Requires careful design to ensure that multiple recommendation methods work effectively together.
  - **Higher Computational Cost**: Processing multiple algorithms can increase computational requirements.

- **Suitability for Project**: A hybrid approach is ideal for the carpool project, as it can leverage the **strengths of both collaborative and content-based filtering**. For example, new users can be recommended based on preferences (CBF), while experienced users can benefit from collaborative recommendations based on their historical interactions.

(Art of the problem, n.d.)

6.5.4 Proposed Recommendation System for Carpool Application

For the carpool application project, the recommended approach is to use a **hybrid recommendation engine** that leverages both collaborative filtering and content-based filtering. The system would include the following components:

1. **Initial User Recommendations**: Use content-based filtering to suggest carpool partners based on explicitly provided user preferences and locations. This helps mitigate the cold start problem for new users.
2. **Ongoing User Suggestions**: Use collaborative filtering to provide personalized recommendations based on user history, including successful past carpools and interactions.
3. **Event Recommendations**: Suggest events that users may be interested in based on events they have attended in the past or user similarity using a mix of content-based and collaborative approaches.

6.5.5 Conclusion

Recommendation engines are essential for improving user engagement and experience in the carpool application project. For this school project, my team and I shall go for **content-based filtering** as the features come from human input, which allows us to recommend items based on explicit user preferences. This approach will help us provide personalized suggestions effectively, making the carpool application more engaging and user-friendly.

7 Sources

Adjiman, P. (2020, 09 26). *How Waze predicts carpools with Google Cloud's AI Platform*. Retrieved from cloud.google.com: https://cloud.google.com/blog/products/ai-machine-learning/how-waze-predicts-carpools-using-google-cloud-ai-platform

Agiliway. (2024, 02 19). *Agiliway*. Retrieved from Pros and cons of angular development: https://www.agiliway.com/pros-and-cons-of-angular-development/

Angular. (n.d.). *DomSanitizer*. Retrieved from v17.angular.io: https://v17.angular.io/api/platform-browser/DomSanitizer

Angular. (n.d.). *Security*. Retrieved from v17.angular.Io: https://v17.angular.io/guide/security

Angular. (n.d.). *Understanding dependency injection*. Retrieved from v17.angular.io: https://v17.angular.io/guide/dependency-injection

API, G. m. (n.d.). *Official Documentation*. Retrieved from developers.google.com: https://developers.google.com/maps/documentation

Art of the problem. (n.d.). *Youtube*. Retrieved from youtube.com: https://www.youtube.com/watch?v=n3RKsY2H-NE

AWS. (n.d.). *MongoDB vs MySQL - Difference Between Database Management Systems*. Retrieved from AWS: https://aws.amazon.com/compare/the-difference-between-mongodb-vs-mysql/

Axios. (n.d.). *Official Documentation*. Retrieved from axios-http: https://axios-http.com/docs/intro

BairesDev. (2022, 08 30). *The pros and Cons of .NET Development*. Retrieved from BairesDev: https://www.bairesdev.com/blog/pros-cons-net-development/

Bojanowska, E. (2024, 02 13). *Naturaily*. Retrieved from pros and cons of using vue js: https://naturaily.com/blog/pros-cons-vue-js

Carrasco, O. C. (2024, 09 30). *Builtin*. Retrieved from Gaussian mixture model: https://builtin.com/articles/gaussian-mixture-model

Celery. (n.d.). *Official documentation*. Retrieved from docs.selery.org: https://docs.celeryproject.org/en/stable/

Chatterjee, i. (2019, 10 13). *Study on clustering algorithms*. Retrieved from Medium: https://medium.com/analytics-vidhya/comparative-study-of-the-clustering-algorithms-54d1ed9ea732

contributors, W. (2024, 11 24). *Java (programming language)*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Java_(programming_language)

contributors, W. (2024, 12 02). *MongoDB*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/MongoDB

contributors, W. (2024, 11 15). *MySQL*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/MySQL

contributors, W. (2024, 11 26). *Svelte*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Svelte

contributors, W. (2024, 12 3). *Vue.js*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Vue.js

contributors, W. (n.d.). *PostgreSQL*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/PostgreSQL

contributors., W. (2024, 11 26). *Angular (web framework)*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Angular_(web_framework)

contributors., W. (2024, 12 2). *Python (programming language)*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Python_(programming_language)

contributors., W. (2024, 11 28). *React (software)*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/React_(software)

Django. (n.d.). *Security in Django¶*. Retrieved from docs.djangoproject.com: https://docs.djangoproject.com/en/5.1/topics/security/

Expressjs. (n.d.). *Production Best Practices: Security*. Retrieved from expressjs.com: https://expressjs.com/en/advanced/best-practice-security.html

Faker. (n.d.). *Official Documentation*. Retrieved from https://faker.readthedocs.io/en/master/

Fiona/Shapely. (n.d.). *Official Documentation*. Retrieved from shapely.readthedocs.io: https://shapely.readthedocs.io/en/stable/

Flask. (n.d.). *Flask-security*. Retrieved from github.com: https://github.com/pallets-eco/flask-security

GeeksforGeeks. (2024, 01 3). *Advantages and Disadvantages of JavaScript*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-javascript/

GeoPandas. (n.d.). *Official Documentation*. Retrieved from geopandas.org: https://developers.google.com/maps/documentation

Google. (n.d.). *About the Waze Transport SDK*. Retrieved from developers.google.com: https://developers.google.com/waze/intro-transport

Google. (n.d.). *Directions API overview*. Retrieved from developers.google.com: https://developers.google.com/maps/documentation/directions/overview

Google. (n.d.). *Directions API usage and billing*. Retrieved from developers.google.com: https://developers.google.com/maps/documentation/directions/usage-and-billing

Google. (n.d.). *Get started*. Retrieved from developers.google.com: https://developers.google.com/maps/documentation/urls/get-started

Google. (n.d.). *The Directions API demo*. Retrieved from developers.google.com: https://developers.google.com/maps/documentation/directions/start

Google, O.-T. (n.d.). *Official Documentation*. Retrieved from developers.google.com: https://developers.google.com/optimization

HelmetJS. (n.d.). *Official Documentation*. Retrieved from helmet.js: https://helmetjs.github.io/

HERE. (n.d.). *HERE Base Plan*. Retrieved from developer.here.com: https://developer.here.com/pricing

HERE. (n.d.). *Technologies Documentation*. Retrieved from developer.here.com: https://developer.here.com/documentation/routing-api/

HERE. (n.d.). *Wego*. Retrieved from wego.here.com: https://wego.here.com/

IBM. (2024, 11 25). *PostgreSQL vs MySQL*. Retrieved from IBM: https://www.ibm.com/think/topics/postgresql-vs-mysql

Jakarta. (n.d.). *Jakarta Security Project Website*. Retrieved from Jakarta.ee: https://jakarta.ee/specifications/security/

Mapbox. (n.d.). *Mapbox API pricing*. Retrieved from docs.mapbox.com: https://www.mapbox.com/pricing/

Mapbox. (n.d.). *Styles API*. Retrieved from docs.mapbox.com: https://docs.mapbox.com/api/maps/styles/

Mapbox. (n.d.). *Waypoints*. Retrieved from docs.mapbox.com: https://docs.mapbox.com/api/navigation/directions/#waypoints

Microsoft. (n.d.). *.NET cryptography model*. Retrieved from learn.microsoft.com: https://learn.microsoft.com/en-us/dotnet/standard/security/cryptography-model

Microsoft. (n.d.). *ASP.NET Core security topics*. Retrieved from learn.microsoft.com: https://learn.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-9.0

Microsoft. (n.d.). *Best practices for a secure software supply chain*. Retrieved from learn.microsoft.com: https://learn.microsoft.com/en-us/nuget/concepts/security-best-practices

Microsoft. (n.d.). *Microsoft Security Development Lifecycle (SDL)*. Retrieved from microsoft.com: https://www.microsoft.com/en-us/securityengineering/sdl?oneroute=true

MongoDB. (n.d.). *Auditing Self-Managed deployments*. Retrieved from MongoDB.com: https://www.mongodb.com/docs/v7.0/core/auditing/

MongoDB. (n.d.). *Authentication on Self-Managed deployments.* Retrieved from MongoDB.com: https://www.mongodb.com/docs/manual/core/authentication/#authentication-mechanisms

MongoDB. (n.d.). *MongoDB Atlas Security*. Retrieved from MongoDB.com: https://www.mongodb.com/collateral/mongo-db-atlas-security

MongoDB. (n.d.). *MongoDB Data Encryption*. Retrieved from Mongodb.com: https://www.mongodb.com/products/capabilities/security/encryption

MongoDB. (n.d.). *MongoDB Developer data platform with strong security capabilities*. Retrieved from mongodb.com: https://www.mongodb.com/products/capabilities/security#data-encryption

MongoDB. (n.d.). *MongoDB.com*. Retrieved from Set up a network peering connection: https://www.mongodb.com/docs/atlas/security-vpc-peering/

MongoDB. (n.d.). *MongoDB: the Developer Data platform*. Retrieved from MongoDB: https://www.mongodb.com/

MongoDB. (n.d.). *Role-Based access control in Self-Managed deployments*. Retrieved from MongoDB.com: https://www.mongodb.com/docs/v7.0/core/authorization/

Montano, D. (n.d.). *Why Choose .NET: Pros and Cons of .NET Development*. Retrieved from waverlysoftware.com: https://waverleysoftware.com/blog/why-choose-net-pros-and-cons/

MySQL . (n.d.). *Access Control and Account Management.* Retrieved from MySQL.dev: https://dev.mysql.com/doc/refman/8.4/en/access-control.html

MySQL . (n.d.). *FIPS Support*. Retrieved from MySQL.dev: https://dev.mysql.com/doc/refman/8.4/en/fips-mode.html

MySQL . (n.d.). *Security Components and Plugins*. Retrieved from MySQL.dev: https://dev.mysql.com/doc/refman/8.4/en/security-plugins.html

MySQL. (n.d.). *Enterprise Encryption*. Retrieved from MySQL.dev: https://dev.mysql.com/doc/refman/8.4/en/enterprise-encryption.html

MySQL. (n.d.). *General Security Issues*. Retrieved from dev.MySQL: https://dev.mysql.com/doc/refman/8.4/en/general-security-issues.html

MySQL. (n.d.). *MySQL Enterprise Data Masking and De-Identification*. Retrieved from MySQL.dev: https://dev.mysql.com/doc/refman/8.4/en/data-masking.html

MySQL. (n.d.). *Security Components and Plugins.* Retrieved from MySQL.dev: https://dev.mysql.com/doc/refman/8.4/en/security-plugins.html

MySQL. (n.d.). *SELinux*. Retrieved from MySQL.dev: https://dev.mysql.com/doc/refman/8.4/en/selinux.html

MySQL. (n.d.). *using Encrypted Connections.* Retrieved from MySQL.dev: https://dev.mysql.com/doc/refman/8.4/en/encrypted-connections.html

NodeJS. (n.d.). *Node.js v23.3.0 documentation - Crypto*. Retrieved from Nodejs.org: https://nodejs.org/api/crypto.html

Npm. (n.d.). *npm-audit*. Retrieved from docs.npmjs.com: https://docs.npmjs.com/cli/v7/commands/npm-audit

Oluseye, J. (2024, 8 18). *Python Backend Development: A Complete Guide for Beginners*. Retrieved from datacamp.com: https://www.datacamp.com/tutorial/python-backend-development

Oracle. (n.d.). *Java Cryptography Architecture (JCA) Reference Guide*. Retrieved from docs.oracle.com: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

OSRM. (n.d.). *Api docs*. Retrieved from project-osrm.org: http://project-osrm.org/docs/v5.24.0/api/

OWASP. (n.d.). *Deserialization of untrusted data*. Retrieved from owasp.org: https://owasp.org/www-community/vulnerabilities/Deserialization_of_untrusted_data

OWASP. (n.d.). *NodeJS Security Cheat Sheet*. Retrieved from cheatsheetseries.owasp.org: https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security_Cheat_Sheet.html#introduction

OWASP. (n.d.). *OWASP Dependency-Check*. Retrieved from owasp.org: https://owasp.org/www-project-dependency-check/

OWASP., F. (n.d.). *OWASP Top Ten*. Retrieved from OWASP: https://owasp.org/www-project-top-ten/

Pip. (n.d.). *pip-audit*. Retrieved from github.com: https://github.com/pypa/pip-audit

PostgreSQL. (n.d.). *Security Informatio*. Retrieved from PostgreSQL: https://www.postgresql.org/support/security/

PostgreSQL. (2024, 11 21). *Authentication methods.* Retrieved from PostgreSQL: https://www.postgresql.org/docs/current/auth-methods.html

PostgreSQL. (2024, 11 21). *Encryption options*. Retrieved from PostgreSQL: https://www.postgresql.org/docs/current/encryption-options.html

PostgreSQL. (2024, 11 21). *Error reporting and logging*. Retrieved from PostgreSQL: https://www.postgresql.org/docs/current/runtime-config-logging.html

PostgreSQL. (2024, 11 21). *Row Security policies*. Retrieved from PostgreSQL: https://www.postgresql.org/docs/current/ddl-rowsecurity.html

PostgreSQL. (2024, 11 21). *Secure TCP/IP Connections with SSL*. Retrieved from PostgreSQL: https://www.postgresql.org/docs/current/ssl-tcp.html

PostgreSQL. (n.d.). *The PostgreSQL security*. Retrieved from PostgreSQL: https://www.postgresql.org/support/security/

Python. (n.d.). *hashlib — Secure hashes and message digests*. Retrieved from docs.python.org: https://docs.python.org/3/library/hashlib.html

Python. (n.d.). *secrets — Generate secure random numbers for managing secrets*. Retrieved from docs.python.org: https://docs.python.org/3/library/secrets.html

React. (n.d.). *Official Documentation*. Retrieved from react.dev: https://react.dev/

Russo, D. (n.d.). *What is Svelte and Why You Should Consider it For Your Business?* Retrieved from bairesdev.com: https://www.bairesdev.com/blog/svelte-and-why-you-should-consider-it/

Santos, M. ,. (2020). A Comparison of Machine Learning Techniques in the Carpooling Problem. *Journal of Computer and Communications*.

Scikit-learn. (2024). *Plot dbscan*. Retrieved from Scikit-learn: https://scikit-learn.org/1.5/auto_examples/cluster/plot_dbscan.html

Scott, P. (2024, 11 25). *What is PostgreSQL?* Retrieved from Percona Database Performance Blog: https://www.percona.com/blog/what-is-postgresql-used-for/#:~:text=PostgreSQL%20offers%20advanced%20SQL%20features,and%20enforcement%20of%20business%20rules

Sharma, I. (2024, 11 7). *What are the Pros and Cons of Java?* Retrieved from Software and Technology Blog - TatvaSoft: https://www.tatvasoft.com/outsourcing/2023/10/pros-and-cons-of-java.html

Simplilearn. (2023, 02 17). *Why Use MongoDB: What It Is and What Are the Benefits.* Retrieved from Simplilearn: https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mongodb

Spring. (n.d.). *Spring Security6.4.1*. Retrieved from Spring.io: https://spring.io/projects/spring-security

Svelte. (n.d.). *Guide to Preventing XSS in Svelte*. Retrieved from svelte.dev: https://svelte.dev/blog/guards-against-xss

Svelte. (n.d.). *Official Documentation*. Retrieved from svelte.dev: https://svelte.dev/docs

The Blue Claw. (2021, 01 4). *MySQL Advantages and Disadvantages*. Retrieved from Blue Claw: https://blueclawdb.com/mysql/advantages-disadvantages-mysql/

Vishal-Siddhpara. (2024, 09 1). *React vs Angular: Which One is Best for Your Next Front-end Project?* Retrieved from Radixweb: https://radixweb.com/blog/react-vs-angular

VueJS. (n.d.). *Official Documentation*. Retrieved from vuejs.org: https://vuejs.org/

VueJS. (n.d.). *Vue Security Best Practices*. Retrieved from vuejs.org: https://vuejs.org/v2/guide/security.html

VueJS. (n.d.). *Vue-sanitize*. Retrieved from github.com: https://github.com/cure53/DOMPurify

Wikimedia, C. t. (2024, 11 29). *JavaScript*. Retrieved from Wikipedia: https://simple.wikipedia.org/wiki/JavaScript

Wikipedia contributors. (2024, 11 21). *.NET*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/.NET