IT Inside (J1)

Project Plan of the Concept

Carpool Application

Prepared for Axxes

Prepared by
- Oleksii Pidnebesnyi
- Tom Bulen
- Yorben Wijnants
- Maciej Chuchra
- Jan-Peter Dhallé
- Mohammed Hamioui

Table of contents

1. Introduction

During our studies in applied computer science, we were tasked with designing and developing a carpooling application for Axxes. This project brought together our team's diverse skills in application development, artificial intelligence, and infrastructure to design and implement a functional and polished solution.

This document serves as a comprehensive plan for the application, detailing its background, business case, stakeholders, objectives, project timeline, and team composition. Additionally, to prepare for the implementation phase, we have evaluated and selected the most suitable technologies to ensure a successful and efficient development process.

2. Background

The product owner for this project has provided us with an assignment to complete. In this section, we will discuss information about the product owner and the assignment they have given us. Additionally, we will cover the current processes that the product owner wants to replace with our finished application.

2.1 Product owner

The product owner for this assignment is Axxes. Axxes is an IT consultancy company with office locations in Antwerp, Ghent, and Utrecht. It is a growing company that employs more than 400 employees.

They offer a wide assortment of services to their customers, including:

- Software engineering
- Data
- (Software) Architecture
- Quality assurance
- Infrastructure
- Coud
- DevOps
- Analysis
- ...

Axxes has strong values and goals. Their core values are **growth**, **quality**, and the **feel-good factor**. Their mission is to increase the growth of people and clients by connecting and strengthening technological expertise (Axxes, n.d.).

Axxes aims to improve employee relationships and ease transportation to events by providing a centralized carpooling solution. The app we are creating, will streamline the process, reduce stress, and align with Axxes' core values of growth, quality, and the feel-good factor. It will also

incorporate a backend, AI, frontend, and infrastructure elements to ensure scalability, efficiency, and a user-friendly experience.

2.2 Current processes

Axxes currently lacks an efficient system for employees to arrange shared rides, making the process of getting to events confusing and inefficient. Employees must manually contact others, determine addresses, and coordinate pickup points, leading to stress and excessive communication. This may discourage carpooling or attendance at events.

Our application addresses these issues by centralizing the carpool process. It uses AI to reduce communication needs and considers user preferences to maximize comfort. This makes carpooling easier, encouraging more employees to attend events and improving workplace relationships.

3. Business case and stakeholders

In this section, we outline the business case for the project and identify the key stakeholders involved. The focus is on understanding the impact, benefits, and responsibilities associated with the project.

3.1 Stakeholders

This subsection details the primary stakeholders whose engagement and participation are critical to the success of the project.

3.1.1 Employees of Axxes

Employees will benefit the most from this project. By improving commuting options and making carpooling easier, more employees will be able to get to events, which may result in higher participation in company activities.

3.1.2 Axxes event management

Event management will benefit from increased employee participation in events due to better transportation options. The project will also make event planning easier by ensuring smoother coordination of travel for attendees.

3.2 Added value

Next, we discuss the values that our carpool application will provide for Axxes.

3.2.1 Value to employees

- Improved commute efficiency
    - The project will make commuting easier and faster for the employees by helping them plan carpools and routes, so their journey to the event is more efficient.
- Participation in Company Events
    - The solution will make it easier for employees to participate in corporate events and potentially through better coordination of transportation and event communication. This could lead to higher attendance rates.

3.2.2 User Experience Improvements

- Better Carpooling Experience
    - A key aspect of the project is to enhance the user experience of carpooling among employees. With a user-friendly system, employees will find it easier to coordinate carpool arrangements, leading to increased usage.

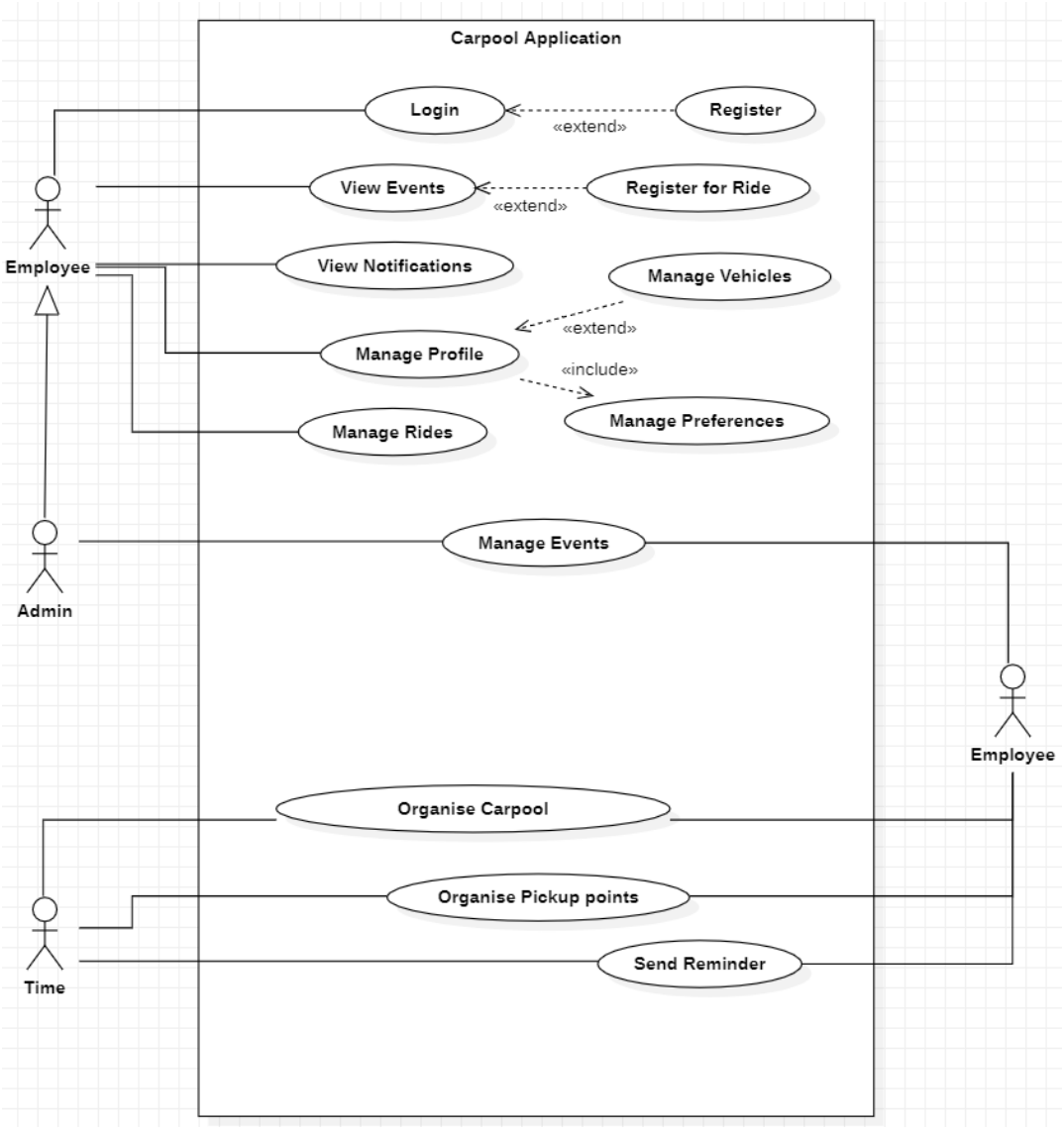
3.2.3 Stronger Employee Connections

- By encouraging more interaction and coordination through shared transportation, employees will form stronger connections. This can improve teamwork and collaboration across departments within the company.

4. Use Case Diagram

This use case diagram shows how different users (Employee, Admin, and Time) interact with the Carpool Application. Employees can log in, manage their own profiles, view events, and register for rides, while Admins manage events. The system also handles tasks like reminders and organizing carpools.

To read it, look at the actors (users) and their connections to the features (use cases). The system boundary (rectangle) shows what's included in our system, and each oval represents a feature the system provides. Relationships like «include» and «extend» show dependencies or optional features.



5. Project Breakdown Structure

- **1.1. Backend**
  - 1.1.1. Core Services
    - Data management for users, cars, events, and rides.
    - Managing event data in a database.
    - Communication with AI services for grouping and optimization.
  - 1.1.2. API Features
    - Authentication and authorization (OAuth 2.0, JWT).
    - Carpool creation and management.
    - Real-time notifications via WebSockets.
    - Event management and participation tracking.
  - 1.1.3. Database
    - MySQL for relational data management (users, rides, events).
    - Real-time updates and synchronization with AI services.
- **1.2. AI System**
  - 1.2.1. User Grouping
    - Match users to carpools based on:
      - Proximity to drivers and other passengers.
      - Preferences (e.g., music, smoking, seating).
      - Driver-defined criteria (pickup radius, vehicle capacity).
  - 1.2.2. Pickup Points Optimization
    - Automated pickup point identification for minimal detours.
    - Route sequencing for efficient passenger pickups.
  - 1.2.3. Real-Time Adaptations
    - Adjustments based on last-minute changes (e.g., new passengers or cancellations).
  - 1.2.4. Tools and Technologies
    - Scikit-learn for clustering and proximity-based grouping.
    - PyTorch for advanced model training (route optimization).
    - Geopy for distance calculations.
- **1.3. Frontend**
  - 1.3.1. User Interface
    - Events page: List and filter upcoming company events with carpool options.
    - Map integration: Display event locations and carpool routes.
    - Login/signup: Secure user authentication.
    - User (optionally driver) settings: Manage profile, vehicles, and notifications.
    - Event management:
      - Create, update, delete events
    - Carpool management:
      - Create, join, or manage active carpools.
      - Adjust pickup points dynamically.
    - Notifications: Real-time alerts for changes in carpools or events.
    - Live chat: Enable seamless communication between drivers and passengers.

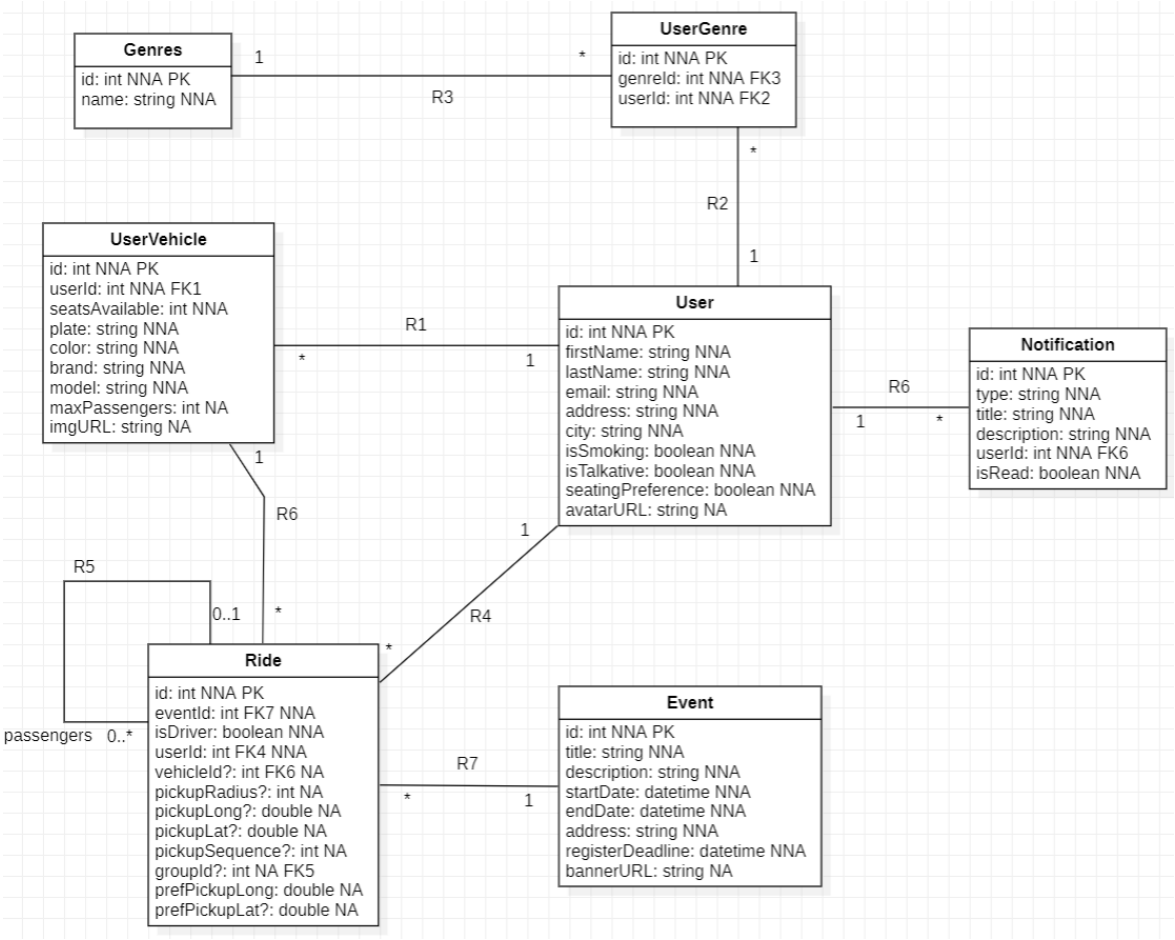
- 1.3.2. Real-Time Features
  - WebSocket integration for:
    - Live updates (chat, carpool status).
    - Notifications on schedule or route changes.
  - Multi-passenger route optimization with real-time mapping.
- 1.3.3. Performance Optimization
  - Lazy loading for essential screens.
  - Efficient error handling to prevent user disruptions.

- **1.4. Infrastructure**
  - 1.4.1. Scalability and High Availability
    - Kubernetes for container orchestration.
    - Terraform for infrastructure provisioning.
    - Auto-scaling with AWS Elastic Kubernetes Service (EKS).
  - 1.4.2. Continuous Integration/Deployment (CI/CD)
    - GitLab Actions for automated build, test, and deployment pipelines.
  - 1.4.3. Monitoring and Security
    - Prometheus and Grafana for system performance monitoring.
    - SonarQube for regular vulnerability scanning.
    - AWS Secrets Manager for secure credential storage.
  - 1.4.4. Data Management
    - Amazon RDS for relational database management.
    - Amazon S3 for static assets (e.g., route maps, logs).

6. ERD

This ERD shows how the carpool application's database is structured. It includes entities like User, Ride, Event, UserVehicle, and Genres, each with their attributes. For example, User has details like firstName, email, and preferences (isSmoking, isTalkative, seatingPreference), which are stored as booleans.

The Genres table handles music preferences, linked to User through the UserGenre table. Relationships connect the entities: a User can own many UserVehicles, and a Ride is linked to a User, Vehicle, and Event. Passengers are also connected to Ride. Numbers like 1, *, and 0..1 show how many records relate. For example, a User can have multiple vehicles, but each Vehicle belongs to one User.



7. Project Scope

We determined the scope for this project by using the MoSCoW model. This model groups project requirements into four different categories. These categories differ in priority.

The ‘Must have’ category includes the features that are required to complete the basic application. The ‘Should have’ category includes important features that aren’t necessary for a

working application but are important. Next up is the category ‘Could have’ which has less important features, and finally there is the category ‘Won’t have’ which includes features that are out of scope for this project.

Must have	**Should have**	**Could have**	**Won’t have**
UI: Profiles, events, carpool management, map, event management	Car management: Drivers manage car details.	Live chat: Real-time communication.	NLP reviews: Analyse user feedback.
API integration: Event data and user authentication	Pickup points: Location selection for passengers	Push notifications for trip updates and reminders.	Highly customizable branding for organizations.
Core functionality: Create/manage carpools	System functions: notifications	Rating system for drivers and passengers.	Complex AI for route prediction.
AI integration: Use AI to optimize pickup points and group users.	Full event management system	Automatic route optimization for drivers.	Fully offline functionality.
Set up Infrastructure	Share ride details via social media.	Multi-language support.	Integration with other apps (Google Maps/ Waze).
Basic Monitoring/Security	Insights into ride usage	Badges for frequent riders or eco-friendliness.	Payment integration.

8. Tech Stack Decisions

This section outlines the key technology choices made for the backend, AI components, frontend, and infrastructure of the project. It covers the rationale and considerations that went into selecting the appropriate tools, frameworks, and platforms to power the various aspects of the system.
The goal of this section is to provide a comprehensive overview of the technical foundation upon which the solution is built.

8.1 Backend

The backend is the engine of the Carpooling App, making sure everything runs smoothly, and all data is managed correctly. It connects the app to the database and the AI system while keeping everything in sync.

8.1.1 Key Features

Interaction with the Database:

- Manages database queries, CRUD operations.

Interaction with AI

- Facilitates communication with the AI system for route optimization and user grouping.

API endpoints

- Provides RESTful APIs for communication between the app's components.
- Ensures security of API endpoints.

WebSocket Live Chat

- Using WebSockets to implement live chat functionality.

Trigger notifications

- Trigger and send notifications.

8.1.2 Framework and Tools

We evaluated several backend frameworks based on key criteria including team experience, performance, scalability, real-time capabilities, security, database integration, and ecosystem support. Below is a summary of the criteria and how each framework measured up.

Before we start the comparison, we will summarize each option that gets compared.

First is Spring Boot, which is a very popular and robust framework for Java with a lot of features. Next is Express.js, combined with Node.js, for a JavaScript backend. Node.js is a JavaScript runtime, which enabled developers to use JavaScript in the backend, express.js is a minimal backend framework for writing REST APIs and building web applications.
Finally, we compared Django, a high-level web framework for Python that speeds up development and allows you to build your application with less code.

Criteria Summary:

- **Team Experience:**
  - **Spring Boot:** Multiple members have experience with java and Spring Boot. They have implemented similar APIs in it.
  - **Node.js:** Moderate experience.
  - **Django:** Limited experience with Python and Django, requiring a steeper learning curve.
- **Real-Time Capabilities:**

    - **Spring Boot:** Native WebSocket support, making it well-suited for live chat and real-time updates on carpool statuses.
    - **Node.js:** Excellent WebSocket support, ideal for real-time communication. Via external library called 'ws'.
    - **Django:** Limited WebSocket support, requiring third-party libraries (e.g., Django Channels) for real-time features. (GeeksForGeeks, n.d.) (DZone, 2019)
- **Security:**
    - **Spring Boot:** Comprehensive built-in security features including OAuth2, JWT, and robust authentication and authorization mechanisms. Auth0, the service we use for frontend authentication can also be used to secure Spring Boot API endpoints.
    - **Express.js:** Requires additional third-party libraries (e.g., Passport.js) for advanced security configurations. Auth0 authentication also works for express.js endpoints.
    - **Django:** Strong security out of the box, but not as flexible as Spring Boot for complex enterprise security needs. Protection against common attacks (SQL injection, cross-site scripting (XSS), and fake requests (CSRF)) out of the box. (GeeksForGeeks, n.d.) (DZone, 2019) Auth0 authentication also works for Django.

- **Database Integration:**
    - **Spring Boot:** Excellent integration with relational databases (PostgreSQL/MySQL) using JPA.
    - **Express.js:** Flexible database options, but less support for complex queries and database management compared to Spring Boot.
    - **Django:** Intuitive ORM integration but less customizable than Spring Boot's JPA. (GeeksForGeeks, n.d.) (DZone, 2019)
- **Ecosystem & Community:**
    - All the options have a highly active community and ecosystem.

8.1.3 Conclusion

After evaluating several backend frameworks, we selected Spring Boot for the carpooling app due to its robust, enterprise-grade features, robust security, and strong support for real-time capabilities like WebSockets. Compared to the other options, our team members have more experience with Java, which also influenced our final choice. The database interaction via Spring Boot JPA is customizable and convenient for this project, compared to alternatives offered by the other frameworks.

Thus, Spring Boot offers a solid combination of features for our app's current and potential future requirements, balancing real-time functionality, security, and scalability and combined with the team experience, we chose it as backend framework for this project.

8.2 Databases

This subsection covers the database technologies selected for the project, including the choice of relational or NoSQL databases and the specific DBMS chosen, based on factors that we deemed important for this project.

8.2.1 Key Features

Basic Data Storage

- Ability to reliably store and retrieve structured data such as user profiles, carpool groups, and trip details.

AI Integration

- Compatibility with our AI service for grouping employees into carpools.

Query Flexibility

- Support for simple filtering and retrieving data, such as fetching trips by date or user ID.

Low Maintenance

- A solution that is easy to manage, deploy, and scale for our small-scale app requirements.

Backups

- Ability to easily create and restore backups of existing data.

8.2.2 Databases and tools

When selecting a database for our carpooling application, it's essential to evaluate options based on how well they meet the specific needs of the project. These needs include ease of integration with our Java Spring backend and AI services, ease of deployment and hosting, team experience, suitability for small-scale applications, and potential for future scalability if required. Below is an evaluation of the most relevant database options.

Criteria Summary

- **Team experience**:

    - Our team has the most experience with relational databases, this excludes MongoDB.
    - Our team has limited experience in using the advanced features of PostgreSQL.
    - Multiple team members have used and deployed MySQL before during projects.
    - Some team members have used SQLite before.
- **Integration**:
    - PostgreSQL integrates seamlessly with Java Spring using tools like Hibernate or Spring Data JPA. It also works well with Python-based frameworks.
    - MySQL integrates effectively with Java Spring, supported by Spring Data JPA and other ORM tools. Python tools and frameworks also have good support for MySQL databases.
    - MongoDB's flexible schema is suitable for AI services, especially when working with semi-structured data. However, its NoSQL nature makes it less compatible with traditional relational queries, requiring additional effort for integration with Java Spring (e.g., using Spring Data MongoDB).
    - SQLite is extremely simple to integrate with Java and Python applications. It's supported out of the box in many development environments but lacks the robust tooling and frameworks available for more sophisticated databases. (DigitalOcean, 2022) (Vercel, 2023)
- **Ease of Hosting and Setup**:
    - PostgreSQL requires a dedicated database server but is well-documented and widely supported across platforms. All major hosting services support it as it is a very popular database (DB-engines, n.d.).
    - Like PostgreSQL, MySQL requires a server setup but is relatively simple to deploy locally and in production. Most hosting services support MySQL as it is even more popular than PostgreSQL (DB-engines, n.d.).
    - MongoDB is easy to set up locally and supports hosted solutions like MongoDB Atlas (MongoDB, n.d.), which provide built-in scalability and monitoring. Its lightweight nature is an advantage for development.
    - SQLite is embedded and requires no server, making it the easiest to set up and run. However, it doesn't scale well for multi-user or larger applications. (Vercel, 2023) (DigitalOcean, 2022)

8.2.3 Conclusion

After evaluating multiple database options, we have chosen MySQL as the database solution for the carpooling app. MySQL offers a solid balance of ease of use and performance, making it an ideal choice for this project. Its popularity ensures good documentation and a wide range of third-party tools. It will also be easy to host with most hosting providers.

While PostgreSQL offers more advanced querying capabilities and extensibility, its added complexity is unnecessary for our app's current scale and requirements.
MongoDB was considered for its scalability and flexibility but was not chosen because we are as a team more familiar with relational databases, that is also why we didn't compare other

NoSQL databases. SQLite was dismissed due to its limitations in handling concurrent users and scalability needs.

Overall, MySQL provides the best combination of reliability, performance, and simplicity for our use case, ensuring a smooth development process and future scalability.

8.3 AI

This section covers the AI-related technology features and decisions, including the machine learning models, frameworks, and libraries chosen to power the intelligent capabilities of the system.

8.3.1 Key Features

User Grouping

The carpooling system ensures efficient ride matching through a structured process.

Initial Registration

- Passengers register their need for transportation to a specific event.
- At this stage, passengers are not yet assigned to a carpool or driver.

Driver Assignment

- As the event approaches, the system evaluates all registered passengers and available drivers to form carpools based on:
    - Proximity between passengers and drivers.
    - Passenger preferences, such as music, smoking, and seating.
    - Driver-defined parameters, including pickup radius and vehicle capacity.

Pickup Points

To optimize routes and minimize travel times, the system automatically determines and manages pickup points:

Automatic Pickup Point Optimization

- The system identifies the most efficient pickup points and sequences passenger pickups to reduce detours for drivers.

User Interaction with Pickup Points

- **Drivers:**
    - View all assigned pickup points on their route.

    - Adjust the pickup sequence if necessary to suit preferences or accommodate changes.

- **Passengers:**
    - Receive notifications about their assigned pickup point.

8.3.2 Main Tasks and Tools

Programming Language

We evaluated several AI tools and frameworks based on key criteria, including team experience, performance, compatibility, flexibility, scalability, and community support. Below is a summary of the criteria and how each tool or framework measured up.

Before diving into the comparison, we summarize the options considered.

Python Ecosystem:

- Python is the most widely used language in AI and machine learning, with a rich ecosystem of libraries like PyTorch, TensorFlow, and Scikit-learn. It is flexible and efficient for developing AI solutions. (GeeksforGeeks, Best Python libraries for machine learning , 2023)

Alternatives:

- Java and C++ were considered for their speed and performance. However, they lack Python's extensive AI-focused libraries and ease of development.

AI Backend - REST API

We decided to use a separate backend for AI services to make the system easier to manage, more flexible, and efficient. AI tasks, like user grouping and route optimization, are highly specialized and benefit from being handled in their own environment. This separation allows us to update or improve the AI models without affecting the main system, ensuring that changes can be made seamlessly. (Documoto, n.d.)

AI Backend Tools

We chose **FastAPI** to handle AI tasks and expose REST APIs for integration with the main system.

Criteria Summary

FastAPI:

- High performance with asynchronous capabilities.
- Easy integration with Python-based AI libraries.
- Built-in OpenAPI support for documentation.

Alternatives: Flask and Django REST Framework

- Flask: Lightweight but lacks FastAPI's asynchronous features.
- Django REST Framework: Robust but overly complex for our AI-specific requirements.

Conclusion:

FastAPI was chosen for its performance, modern features, and compatibility with our AI tools. (GeeksforGeeks, Flask vs FastAPI: Which one to choose, 2023)

Geolocation and Route Optimization

Problem:
The carpooling app requires geolocation functionality to:

- Calculate distances between user locations and pickup points.
- Perform basic geocoding tasks for route optimization and pickup point determination.
- Ensure cost-efficiency while delivering reliable geolocation services.

Chosen Tool: Google Maps API
Why Google Maps API?

- Geocoding, route optimization, and real-world distance calculations. (Platform, n.d.)
- Handles multi-stop routes efficiently.
- Accurate and up-to-date global data.

Alternatives
Geopy

- Basic Functionality: Only calculates straight-line distances, no road or traffic data.
- No Route Optimization: Can’t handle multiple stops or advanced routing.
  Lacks real-world travel and traffic accuracy. (GeoPy, n.d.)

User Grouping and Clustering

Scikit-learn:

- Lightweight and easy to implement.
- Specialized clustering algorithms like K-Means and DBSCAN.
- Strong community support and extensive documentation.

Alternatives (TensorFlow and PyTorch):

- Overpowered for clustering tasks.
- Designed for deep learning, unnecessary for simpler clustering tasks. (GeeksforGeeks, Scikit-Learn vs TensorFlow: Which one should you choose?, 2023)

Pickup Points Optimization

Google Maps API

- Provides reliable distance matrix for route optimization.
- Includes routing features like shortest path calculations.
- Easy integration for location-based services

Alternatives
ORTools, NetworkX

Why Google Maps API?
1. Google Maps API is a proven solution for location-based optimizations.
2. It reduces development complexity by handling routing and distance calculations internally.
3. It has global data coverage and high reliability

8.3.3 Conclusion

After careful evaluation, we selected Python with libraries like Scikit-learn, PyTorch, and Geopy for the AI system. FastAPI will serve as the AI backend for REST API integration. These tools balance performance, flexibility, and simplicity, meeting the app's current and future AI requirements. This structured approach ensures scalability and seamless integration, enhancing the carpooling app's overall intelligence and user experience.

8.4 Frontend

The **frontend** of the carpooling app is designed to deliver an intuitive and user-friendly experience for managing carpools. The focus is on key elements such as interactive screens, real-time updates, performance optimization, and using the right tools to ensure a smooth and efficient user experience.

8.4.1 Key Features

User Interface (Screens):

- **Events:** Display a list of upcoming events, recommended events, and provide filters for users to find relevant options.
- **Map Integration:** Show event locations and carpool options for selected events.
- **Login/Signup:** Allow users to log in using email and password.
- **User Settings:** Enable users to manage profiles, vehicles (type, license plate, available seats, photo), and notification settings.
- **Carpool Management:** Allow users to post or request carpools, view active rides, and manage trip pickup points.
- **Event Management:** Allow admin users to add, edit and delete events.
- **Multi-Passenger Route Optimization:** Dynamically optimize driver routes when multiple passengers are added, minimizing detours and providing a real-time visual map for coordination.
- **Notifications:** Real-time alerts for carpool status, chat messages, and event updates.
- **Live Chat:** Facilitate communication between drivers and passengers.
- **My Carpools:** Provide an overview of active and past rides, including archived trips.

- **Real-time Updates:** We will use **WebSockets** to deliver live updates for chats, carpool statuses, and notifications, ensuring immediate feedback for users.
- **Performance Optimization:** To enhance performance, we will implement lazy loading for faster load times and prioritize essential components. Efficient error handling will ensure reliability and a seamless user experience.

8.4.2 Framework and Tools

Framework Selection Process

To determine the most suitable framework, we compared four popular options: **React/Next.js**, **Angular**, **Vue**, and **Svelte**. Each was evaluated based on team experience, performance, scalability, community support, ease of learning, and security.

- **React/Next.js:**
  - The most popular framework, with extensive community support and good developer tooling.
  - Team members have significant experience with React/Next.js.
  - Highly optimized for performance and scalability. (Dakowicz, 2024)
- **Angular:**
  - A popular choice with a rich feature set but less team experience compared to React/Next.js.
- **Vue:**

    - Offers a user-friendly syntax, but team members have limited experience with its ecosystem.
- **Svelte:**
    - Highly efficient and the "most loved" framework, but less mature and unfamiliar to the team. (Springer, n.d.)

After discussing with the team and comparing the options, React/Next.js emerged as the best choice due to its alignment with the team's expertise and project requirements for a responsive, mobile-friendly web application that performs well on low Wi-Fi and ensure good overall performance.

But the framework is not the only thing we need to complete the frontend. We can make use of libraries to make the frontend more performant and easier to maintain.

8.4.3 Key Frontend Tools

In this section, we will delve into the key frontend tools that will support our application development, including libraries, icon sets, design tools, services … These resources alongside our chose framework (Next.js), will enhance our ability to create a visually appealing and interactive user experience.

Authentication

For authentication we will use **auth0**. We chose this because some team members have used it in the past and it has all the features and integrations we need.
Another benefit is that it can also be used to secure the API endpoints of the backend.

Map View

To implement the map functionality in our application, we decided to use libraries built on Mapbox due to its extensive features and high customizability. Specifically, we will utilize both the **react-map-gl** and **mapbox-gl-js** libraries to achieve our goals.

State Management

We will use **Zustand** for efficient, lightweight, and flexible state management. Its simplicity and performance advantages make it superior to alternatives like Redux and Context.

UI Framework

Tailwind CSS and **shadcn/ui** will be used to create responsive, customizable interfaces. shadcn/ui provides component source code, enabling us to adapt components as needed.

Icons

Iconify will integrate a wide range of icons, supporting all popular icon sets for maximum flexibility.

Mock-ups

We will use **Figma** to design and prototype the user interface. Figma is free, easy to learn, and supports shadcn/ui components, and iconify icons (via plugins) ensuring consistency and a fast design process.

8.4.4 Conclusion

After evaluating various options, we chose **React/Next.js** as the frontend framework due to its strong community support, team familiarity, and scalability. Complemented by tools like Zustand, Tailwind CSS, shadcn/ui, Iconify, and Figma, the frontend will deliver a high-performance and user-friendly carpooling experience.

8.5 Infrastructure

To run our app, we need an infrastructure that supports its operation and ensures it performs well. This includes the servers, databases, and tools that power the app, handle user requests, and store data. A strong infrastructure is essential to keep the app fast, reliable, and scalable as more users use our app.

8.5.1 Key Features

- **Scalability**: Use Terraform and Kubernetes to provision a scalable infrastructure capable of managing varying loads, especially during peak event times.
- **Continuous Integration/Deployment (CI/CD)**: Implement **GitHub Actions** for seamless deployment, allowing regular updates and ensuring smooth app enhancements.
- **Monitoring & Logging**: Utilize **Prometheus** and **Grafana** for real-time monitoring of system performance, ensuring any issues are detected and resolved quickly.
- **Security**: Regularly scan for vulnerabilities, ensuring data and system security.
- **High Availability**: Ensure the app infrastructure is reliable, minimizing downtime and maintaining a high level of availability for users.

8.5.2 Infrastructure tools

In this section we will explore some tools available for running and managing our application. We will compare Docker and Kubernetes two of the most used technologies for containerization and orchestration. We will evaluate both tools based on key criteria such as ease of setup,

scalability, resource efficiency, fault tolerance, and security. After a comparison, we will decide on which tool best suits our infrastructure needs.

- **Ease of Setup**
  - Docker simplifies containerization, offering an easy-to-use CLI and tooling for building and managing containers.
  - Kubernetes adds complexity to the setup, requiring configuration of components like control planes and worker nodes. Tools like Minikube and Kind can simplify local setups for development.

- **Scalability**
  - Kubernetes is built for scalability, allowing applications to scale seamlessly across multiple nodes.
  - Features like horizontal pod autoscaling enable Kubernetes to dynamically adjust resource allocation based on demand, making Docker containers within Kubernetes highly scalable.
  - Docker is limited in scalability. It is designed for local or small-scale deployments and cannot dynamically manage resources or scale containers across multiple machines.
- **Resource Efficiency**
  - Docker containers are lightweight and resource-efficient, sharing the host kernel to minimize overhead.
  - Kubernetes enhances resource management by providing features like quotas, limits, and dynamic resource allocation for optimal performance.
- **Learning Curve**
  - Docker is beginner-friendly, with ample documentation and community support.
  - Kubernetes has a steeper learning curve, requiring familiarity with concepts like pods, services, deployments, and namespaces. However, its active community and rich ecosystem offer extensive learning resources.
- **Fault Tolerance**
  - Docker relies on manual intervention or scripting for fault recovery.
  - Kubernetes introduces advanced fault tolerance, with self-healing features like automatic pod restarts and replica sets to maintain desired states.
- **Networking/Load Balancing**
  - Docker provides basic networking capabilities for container communication and port exposure.
  - Kubernetes significantly enhances this with advanced networking policies, built-in load balancing, and traffic distribution across pods using services.
- **Security**
  - Docker offers features like secrets management and secure image handling but depends on user configuration for strong security.
  - Kubernetes strengthens security with features like Role-Based Access Control (RBAC), namespace isolation, and encrypted secrets, providing a robust security framework for containerized applications.

8.5.3 Conclusion

After comparing Docker and Kubernetes across various criteria, both tools have their strengths and weaknesses, making them suitable for diverse needs within an infrastructure. Docker is ideal for smaller-scale applications or environments where ease of setup and resource efficiency are prioritized. Its simplicity and lightweight nature make it a strong candidate for straightforward containerization tasks.
On the other hand, Kubernetes excels in scenarios that demand scalability, high availability, and fault tolerance. Its advanced features like horizontal pod autoscaling, self-healing, and robust networking and security frameworks make it the best choice for large-scale, production-ready environments.
For our needs Kubernetes is the better option since there are requirements for scalability and high availability.

8.5.4 Gitlab

CI/CD

One of the client's requirements was to implement Continuous Integration/Deployment (CI/CD) to streamline the software development process. We will achieve this using GitLab's built-in CI/CD features. GitLab offers the ability to create and manage pipelines, which automate tasks like building, testing, and deploying code. In our setup, a pipeline will automatically trigger whenever code is pushed to the repository. This ensures that new changes are tested and deployed efficiently, reducing manual intervention and the risk of errors.

Branches

GitLab also provides the ability to create and manage branches within the repository. Branching allows us to develop and test new features or fixes without affecting the main codebase (main branch). By creating a branch for each new task or feature, developers can work independently and safely, ensuring that unfinished or experimental changes do not disrupt the main application. Once the changes are complete and tested, the branch can be reviewed and merged back into the main branch using GitLab’s merge request feature.

To ensure good teamwork and to prevent confusion, branches (except the main branch) must follow a naming convention. Every branch must be made for a single feature and must include the name (or initials) of the person that is developing in the branch.

Benefits:

- Parallel Development
    - Multiple team members can work on different features simultaneously.
- Safe Testing
    - New code can be tested in isolation without risking bugs in the main branch.

Security

GitLab provides robust features to enhance security across the repository and pipelines, ensuring that both the code and its deployment process are protected.

Repository Security: Protected Branches
To enhance repository security, we can utilize protected branches in GitLab. A protected branch is a critical feature that helps safeguard important parts of the codebase, such as the main or production branch, from accidental or unauthorized changes.

What Protected Branches Do:

- Restrict Direct Pushes
    - only authorized users can push changes directly to a protected branch. This prevents unauthorized or untested updates from being added to key branches.
- Mandatory Code Review
    - Changes must be submitted via a merge request, which requires approval from designated reviewers before being merged into the protected branch. This ensures a review process is followed.

Pipeline Security

Securing pipelines is just as important as securing the repository. Pipelines handle sensitive tasks like building, testing, and deploying code, and they often use credentials or other secrets. Without proper security measures, they can become a weak point in the workflow.

Pipeline Secrets:

GitLab supports secure management of variables and secrets, such as API keys or database credentials, which can be stored securely and injected into the pipeline environment when needed. These secrets are encrypted and not exposed in logs.

Static and Dynamic Code Analysis:

GitLab pipelines can include security scanning jobs.

- Static Application Security Testing (SAST) to detect vulnerabilities in the code.
- Dependency Scanning to identify security issues in third-party libraries.
- Dynamic Application Security Testing (DAST) to test running applications for vulnerabilities.

8.5.5 Cloud environment

To host our app, we will use Amazon Web Services (AWS) this is a cloud platform that offers a wide range of services to meet our scalability, reliability, and security requirements.
The following aspects in AWS are services that fit our requirements.

Scalability and Elasticity

AWS provides Auto Scaling and Elastic Load Balancing (ELB) to ensure that the infrastructure scales dynamically with user demand.

Key components:

- Amazon EC2 Auto Scaling
    - Automatically adjusts the number of instances based on traffic and workload.
- Elastic Load Balancing (ELB)
    - Distributes incoming application traffic across multiple instances, ensuring optimal resource utilization and availability.

Managed Kubernetes with Amazon EKS

For container apps, we can utilize Amazon Elastic Kubernetes Service (EKS). EKS simplifies the deployment and management of Kubernetes clusters by offloading the operational overhead to AWS, providing a reliable, secure, and scalable platform for managing containers.

Benefits:

- Seamless integration with AWS
- Automatic scaling and self-healing of pods to maintain application reliability.

Storage and Databases

AWS offers wide range of storage and database solutions, ensuring data reliability and security:

- Amazon RDS (Relational Database Service)
    - For managing structured data efficiently with automated backups and scaling.
- Amazon S3 (Simple Storage Service)
    - To store and serve static assets like images, logs, or backups with high durability.

Monitoring and Logging

Aws also has some services we can use for monitoring like Amazon CloudWatch. Amazon CloudWatch offers real-time monitoring of metrics, logs, and alarms to ensure system health.

Security

Even when we use AWS it’s essential to take basic security measures to protect data, applications, and resources. While AWS provides several built-in security tools, it’s up to us to configure and use them effectively to safeguard our App. Below are services that AWS offers and that we can use to enhance the security of our application.

AWS Secrets Manager

AWS Secrets Manager is a fully managed service provided by Amazon Web Services that helps you securely store, manage, and retrieve sensitive information, such as database credentials, API keys, and other secrets. This service is particularly useful for managing secrets in cloud-

based applications, ensuring that sensitive data is securely handled, stored, and accessed in compliance with security best practices.

Security Groups

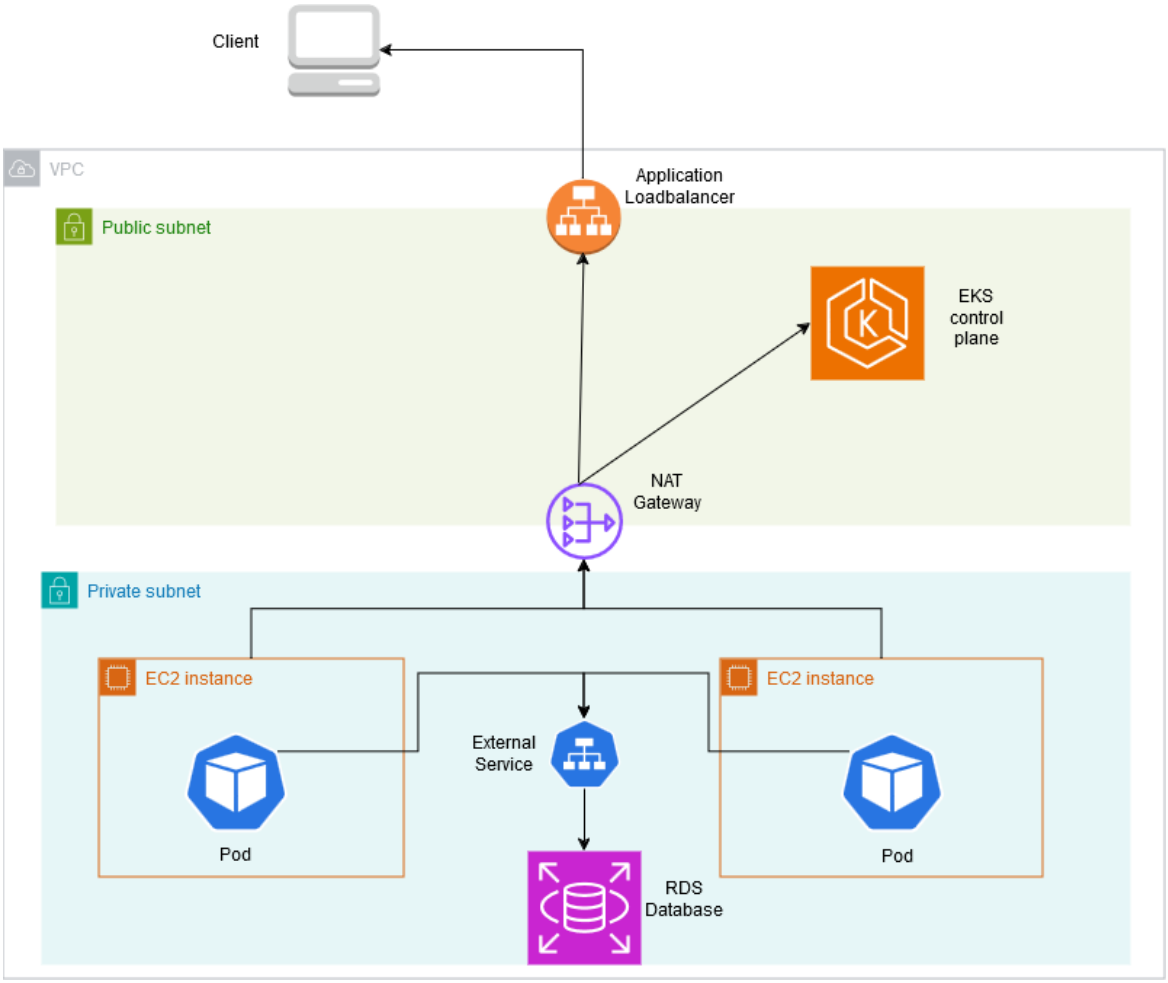
Security Groups are another big thing that we can use within AWS. Security Groups are virtual firewalls for controlling inbound and outbound traffic to resources like EC2 instances and load balancers. Each security group can be assigned to instances to define the allowed traffic rules, ensuring the infrastructure is protected from unauthorized access.

8.5.6 Cloud Diagram

This diagram is a first look at the infrastructure of our application, displaying its main components and how they interact. It includes a client facing Application Load Balancer (ALB) in the public subnet, which routes traffic to an EKS control plane for managing containerized workloads. A NAT Gateway provides secure internet access for resources in the private subnet, without exposing them directly to the internet.

The private subnet hosts critical components, such as EC2 instances running application pods, an External Service for internal communication, and a secure RDS database for persistent data storage. The flow of data ensures scalability, with Kubernetes managing workloads, and security measures like subnet isolation protecting sensitive resources.

This architecture supports high availability, scalability, and secure communication across components. Not everything is in this diagram yet. There are other services that we will use but are not on here. The goal of this diagram is to provide a global view of how the infrastructure is going to look.



8.5.7 cost analysis

Next is a cost analysis of different deployment options for running Kubernetes workloads in our infrastructure. We compared the expenses associated with Amazon EKS on EC2 instances, Amazon EKS Serverless (Fargate), and an alternative approach using an Auto Scaling Group with EC2 instances.

Each option offers distinct trade-offs in terms of cost, performance, and scalability. By evaluating factors such as resource utilization, pricing models, and workload characteristics, we aim to determine the most cost-efficient solution for our application while maintaining reliability and performance.

For this comparison we chose T3.medium instances, so the cost is the same everywhere. This way we can clearly see the cost difference between services.

Feature	EKS with EC2	EKS with Fargate	Auto Scaling Group
Compute Management	EC2 instances managed in Kubernetes clusters.	Fully serverless pod management.	EC2 instances directly managed in an ASG.

Control Plane Costs	$74/month per cluster.	$74/month per cluster.	Does not use one
Compute Costs	Instance costs based on type (e.g., t3.medium ~$30.37/month).	$74.82/month per pod (2vCPU, 4GB RAM).	Instance costs based on type (e.g., t3.medium ~$30.37/month).
Load Balancing	Application Load Balancer (~$20/month).	Application Load Balancer (~$20/month).	Application Load Balancer (~$20/month).
S3 storage	(~$0.023/GB/month for storage).	(~$0.023/GB/month for storage).	(~$0.023/GB/month for storage).
Database	RDS (e.g., t3.medium ~$126/month).	RDS (e.g., t3.medium ~$126/month).	RDS (e.g., t3.medium ~$126/month).

Based on the cost analysis, EKS with EC2 instances is the most cost-effective option for our Kubernetes-based infrastructure compared to EKS Serverless (Fargate) and an Auto Scaling Group (ASG). While Fargate simplifies management by eliminating the need for instance provisioning, it comes with higher compute costs and for our use case where cost efficiency is important, EKS with EC2 is the best option.

Auto scaling groups is an alternative that we could use. But the scaling isn't as good as the scaling EKS provides. ASG is also made for simple containerized apps.

8.5.8 Data security

Ensuring the security of data is important, both when it is stored (data at rest) and when it is transmitted over the network (data in transit). AWS provides several tools and best practices to ensure that sensitive data remains protected throughout its lifecycle.

We will develop this application with security in mind, this means that throughout the development, we will look and test for security vulnerabilities.

Data at Rest

Data at rest refers to data stored in databases, file systems, or storage services. To protect this data, we will use encryption.

Amazon RDS credentials

For relational databases like MySQL, AWS Secrets Manager can be used to securely store and manage database credentials. Secrets Manager ensures that sensitive credentials, such as usernames and passwords, are protected by securely storing them in an encrypted vault.

Amazon S3 Encryption

For object storage, S3 provides built-in encryption options. We can enable server-side encryption using SSE-S3 or SSE-KMS for more granular control over encryption keys. This protects data such as user-uploaded files or logs.

Data in Transit

Data in transit refers to data that is moving between systems or across networks. Encryption is needed to protect data as it travels over the internet or between components in the cloud.

HTTPS Encryption

All sensitive data transmitted over the web, such as user login information or payment details, must be encrypted using the HTTPS protocol. This way we ensure that all data sent between the user's browser and the app server is encrypted.

9. Timeline for Concept Phase

Phase 1 (Deadline: 2/12/2024)

- **View existing data**: Analyse the current data sources and structures to understand their format and usage.
- **Investigate OTA updates**: Research over-the-air (OTA) update mechanisms to understand feasibility and compatibility with the system.
- **Analyse hosting price & performance**: Compare various hosting providers to evaluate cost, scalability, and performance benchmarks.
- **Design server application**: Outline the architecture and workflow for the server-side application, including data handling, user authentication, and API requirements.
- **Design UI prototypes**: Develop wireframes and mock-ups for the user interface, focusing on usability and user experience.
- **Design data model**: Define the structure of the database and how it will interact with the application, ensuring it supports all necessary operations.
- **Determine functional requirements for server application**: Finalize the necessary features and capabilities that the server application must include, such as data storage, retrieval, and security protocols.
- **Submit programming tools and DBMS**: Decide on and submit the chosen programming languages, development tools, and the database management system to be used.

Phase 2 (Deadline: 16/12/2024)

- **Design data visualizations**: Develop and test data visualization tools to represent the data effectively, ensuring they align with user needs.
- **Prototype for dashboards for users**: Create a functional prototype for user dashboards that present data in a user-friendly, accessible manner.
- **Select Hosting site**: Make a final decision on the hosting provider based on the price, scalability, performance, and security criteria.
- **Network & communication choices**: Evaluate and decide on network protocols and communication methods between clients and the server.

- **Investigate security**: Analyse potential security risks and develop strategies for protecting data, both in transit and at rest.
- **Record DevOps configuration**: Document the chosen DevOps tools and processes, including version control, CI/CD pipelines, and automated testing.
- **Provision infrastructure**: Set up the necessary cloud infrastructure, including virtual machines, databases, and networking components.

Phase 3 (Deadline: 23/12/2024)

- **Prepare presentation concept phase**: Develop a concept and materials for presenting the work done during the concept phase, including visual aids and summaries of key findings.
- **Finish documentation**: Complete all necessary project documentation, including user guides, technical manuals, and the final project report.

10. Risks and measurements

When developing an app, there are always some risks. It is essential to identify them and consider appropriate measures to address them. By proactively managing these risks, we can ensure the app is secure, efficient, and user-friendly while maintaining trust and reliability.

Authentication and Authorization Issues:
Risks:

- Unauthorized access to user profiles and data
- Misconfigured permissions allowing users to escalate privileges (e.g., gaining admin rights)
- Data breaches if authentication mechanisms are poorly secured

Mitigations:

- Implement secure authentication
  - Use protocols like OAuth2 or OpenID Connect for authentication.
- Strong password policies
  - Require passwords to be at least 12 characters long and include a mix of character types.
- Rate limiting and account lockout
  - Limit failed login attempts to prevent brute force attacks.
- Role-Based Access Control (RBAC)
  - Ensure users only have access to features relevant to their role.

Insufficient Protection of Sensitive Data
Risks:

- Leakage of sensitive information such as addresses, preferences, or other personal data.
- Transmission of unencrypted data over the network.

Mitigations:

- Data encryption
    - Encrypt sensitive data both at rest (in the database) and in transit (use HTTPS with TLS).
- Tokenization
    - Replace sensitive data with tokens for processing where possible.
- Access control for data
    - Ensure that sensitive data can only be accessed by authorized users or services.

Security Vulnerabilities in the Backend API

Risks:

- Injection attacks (e.g., SQL injection) due to improper input validation.
- Exploitation of poorly documented or overly permissive endpoints.

Mitigations:

- Input validation
    - Sanitize and validate all inputs against strict criteria. Our frontend tooling (NEXT.js) will sanitize all input automatically.
- Parameterized queries
    - Use prepared statements or ORM tools to prevent injection attacks.
- API security
    - Implement rate limiting, strong authentication mechanisms, and endpoint access restrictions.
- API documentation
    - Clearly define and secure each endpoint using tools like Swagger/OpenAPI

Insufficient Transport Layer Security

Risks:

- Data intercepted during transmission over unsecured connections.
- Man-in-the-middle (MITM) attacks.

Mitigations:

- Enforce HTTPS
    - Redirect all HTTP traffic to HTTPS and ensure a valid TLS certificate.
- HSTS headers

    - Implement HTTP Strict Transport Security (HSTS) to prevent protocol downgrades.

Poor User Input Handling

Risks:

- Cross-Site Scripting (XSS) attacks via poorly sanitized input.
- Broken functionality or system crashes due to unexpected input formats.

Mitigations:

- Content sanitization
    - Use libraries or frameworks to sanitize inputs
- Validation
    - Validate inputs on both the client side and server side to ensure data integrity.

Insufficient Scalability and Infrastructure Security

Risks:

- Downtime or slow performance due to high traffic or resource constraints.
- Attacks on infrastructure, such as Distributed Denial of Service (DDoS).

Mitigations:

- Infrastructure as Code (IaC)
    - Use IaC tools (e.g., Terraform) to ensure consistent and secure deployment.
- Kubernetes for scalability
    - Deploy using a Kubernetes cluster for scalability and high availability.
- DDoS mitigation
    - Use cloud services with built-in DDoS protection (e.g., AWS Shield or Azure DDoS Protection)
- Monitoring and logging
    - Implement centralized logging and real-time monitoring with tools like Prometheus and Grafana.

Lack of AI Transparency and Bias

Risks:

- Users might perceive the AI-based carpool predictions as unfair or biased.
- Poor AI model decisions may reduce user trust.

Mitigations:

- Explainability

    - Provide users with insights into why specific carpool groups and pickup points were suggested.
- Bias mitigation
    - Use diverse datasets to train the AI and continuously evaluate the model for bias.
- User feedback
    - Allow users to submit feedback on group and route suggestions, improving the model over time.

Poor Deployment and Update Practices

Risks:

- Deployment of insecure or outdated code.
- Difficulty scaling or fixing bugs quickly.

Mitigations:

- GitOps
    - Automate deployment pipelines with Git-based workflows.
- Automated testing
    - Implement unit, integration, and security tests in CI/CD pipelines.
- Regular updates
    - Monitor for vulnerabilities and patch dependencies promptly.

Insufficient User Privacy Measures

Risks:

- Users feeling uncomfortable sharing personal data, leading to reduced adoption.
- Legal violations (e.g., GDPR) if data privacy isn't properly handled.

Mitigations:

- Privacy by design
    - Minimize data collection to what is strictly necessary.
- Anonymization
    - Store sensitive information in an anonymized or pseudonymized format where possible.
- User control
    - Provide users with options to manage or delete their data.

Lack of Security Awareness Among Users

Risks:

- Use of weak passwords.

Mitigations:

- Password policy
    - Enforce a strong password policy requiring a minimum length (e.g., 12 characters) and complexity (e.g., including uppercase, lowercase, numbers, and special characters).
- Multi-factor authentication (MFA)
    - Encourage or mandate the use of MFA for better account security.

11. Reporting

During the starting phase of this project, an employee of Axxes visited us to clarify the assignment and answer questions.

For this project we will report once to the project owner in the form of a presentation. This is planned on 22/10/2024. On this day we presented our initial idea and preparations to an employee of Axxes. This gave him a good idea of the application we envisioned and was an opportunity for us to receive feedback before we create a very detailed project plan.

After this first report we used the feedback from Axxes and our teacher to improve and further develop this project plan.

On 17/12/2024 we present our final application plan to an employee of Axxes. After this final report we will move forward with the implementation of the application.

12. Project Team

12.1 Team Members & Roles

- **Backend Developer:** Jan-Peter Dhallé, Tom Bulen, Oleksii Pidnebesnyi, Mohammed Hamioui
- **UI/UX and Frontend Developer:** Tom Bulen, Oleksii Pidnebesnyi, Mohammed Hamioui
- **AI/ML Specialist:** Oleksii Pidnebesnyi, Maciej Chuchra
- **DevOps and QA:** Jan-Peter Dhallé, Yorben Wijnants

12.2 Role Descriptions

This subsection defines the key roles and responsibilities for the project team, including the required skills for each position.

12.2.1 Backend Developer

Competences:

- **Server-Side Development:** Proficient in languages like Node.js, Python, or Java.
- **API Development:** Skilled in creating RESTful APIs for client-server communication.
- **Database Management:** Expertise in SQL and NoSQL databases.
- **Security Protocols:** Knowledge of data security best practices.

Contribution in project:

- The Backend Developer ensures efficient server operations and secure data handling, enabling scalability and reliability for user authentication, ride requests, and transaction management.

12.2.2 UI/UX and Frontend Developer

Competences:

- Proficiency in HTML, CSS, Tailwind CSS, shadcn/ui
- Familiarity with React, NextJs
- Understanding of Figma, PS
- Knowledge of UX/UI principles

Contribution in project:

1. Design and implement user interfaces for the applications.
2. Collaborate with backend developers to integrate APIs and other services.
3. Create responsive and accessible designs.

12.2.3 AI/ML Specialist

Competences:

- Proficiency in Python
- Familiarity with PyTorch, Scikit-learn
- Understanding of statistical methods and data analysis

Contribution in project:

1. Design and develop machine learning models and algorithms.
2. Collaborate with other teams to integrate AI capabilities into applications.
3. Validate and test models to ensure accuracy and reliability.

12.2.4 DevOps and QA

Competences:

- Proficiency in using **Terraform**
- Container Orchestration: Expertise in **Kubernetes**
- Skilled in GitHub Actions
- Knowledge of Prometheus and Grafana

Contribution:

1. Provision and manage infrastructure using Terraform and Kubernetes for scalability.
2. Implement CI/CD pipelines with GitHub Actions for seamless deployments.
3. Set up real-time monitoring with Prometheus and Grafana to track system performance.

13. Conclusion

The carpooling app for Axxes is designed specifically to help employees organize rides for company events. It reduces the hassle of arranging transportation, making it easier for employees to connect and attend events.

Before implementation, we dedicated significant effort to selecting the most suitable frameworks, tools, and methodologies. We defined a clear scope with well-outlined features and functionalities to implement, assessed potential risks, and devised mitigation strategies to address them.

With AI, efficient backend systems, and a simple design, the app will promote smoother event participation and stronger teamwork, aligning with Axxes' values and benefiting both the company and its employees.

14 Sources

Axxes. *Over Axxes*. Axxes: https://www.axxes.com/about

Dakowicz, J. (2024, November 14). *Pros and Cons of Next JS – 2025 Updated Version*. https://pagepro.co/blog/pros-and-cons-of-nextjs/

DB-Engines Ranking. https://db-engines.com/en/ranking

DigitalOcean. (2022, March 10). *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems*. https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

Set environment variables within your container's environment. dockerdocs: https://docs.docker.com/compose/how-tos/environment-variables/set-environment-variables/

Documoto. (n.d.). *REST API web services: Scalability, flexiblility, and security*. Retrieved from https://www.documoto.com/blog/rest-api-web-services-scalability-flexibility-and-security

DZone. (2019, February 18). *Web Development Comparison: Spring Boot vs. Express.js*. https://dzone.com/articles/web-development-comparison-springboot-vs-expressjs

GeeksforGeeks. (2022, October 15). *Camparison of FastAPI with Django and Flask*. https://www.geeksforgeeks.org/comparison-of-fastapi-with-django-and-flask/

GeeksforGeeks. (2023, November 15). *Best Python libraries for machine learning* . https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/

https://www.geeksforgeeks.org/flask-vs-fastapi/

GeeksforGeeks. (2023, September 15). *Scikit-Learn vs TensorFlow: Which one should you choose?* https://www.geeksforgeeks.org/scikit-learn-vs-tensorflow-which-one-should-you-choose/

GeeksForGeeks. https://www.geeksforgeeks.org/django-vs-spring-boot/

Welcome to GeoPy's documentation! https://geopy.readthedocs.io/en/stable/index.html

Kind. (2021, 8 21). *Non-Goals*. kind: https://kind.sigs.k8s.io/docs/contributing/1.0-roadmap/#non-goals

MakeUseOf. (2023, May 10). *7 benefits of using RESTful APIs*. https://www.makeuseof.com/benefits-of-restful-apis/

MongoDB Cloud Services. MongoDB : https://www.mongodb.com/products/platform/cloud

Numbers, C. (2024, January 9). *Camparing Python web frameworks: Django, Flask and FastAPI*. https://www.capitalnumbers.com/blog/django-vs-flask-vs-fastapi/

Svelte vs. Angular vs. React vs. Vue - Who wins? https://javascript-conference.com/blog/svelte-vs-angular-vs-react-vs-vue-who-wins/

Superface.ai. (2022, May 10). *Geocoding APIs compared: Pricing, free tiers & terms of use*. https://superface.ai/blog/geocoding-apis-comparison-1

Vercel. (2023, November 8). *Comparing MySQL, PostgreSQL, and MongoDB* . https://vercel.com/guides/mysql-vs-postgresql-vs-mongodb