Tampere University

COMP.SE.110-2024-2025 Software Design

Group project

Team Iphone:

1. Masood Ahmadi
2. Kashif Rasool
3. Salman Khan
4. Md Ashfak Haider Nehal
5. Selina Rautakoski

Route Planning App

[Design prototype (figma)](#)

Version History:

Version history	Dates	Authors	Descriptions
0.1	16.09.2024	Masood Ahmadi Salman Khan Selina Rautakoski Kashif Rasool Md Ashfak Haider	• Design prototype • Table contents
1.0	29.10.2024	Ashfak Haider	• Prepare design document

Table of Contents

1. Introduction

The Route Planning Application is designed to help users plan safe routes by integrating real-time data such as weather conditions, traffic, and road status. The primary interface allows users to view recent routes, save routes, download route data, and check weather forecasts specific to planned routes. A dashboard interface, accessible via a sidebar menu, provides centralized navigation to all app functionalities. The app is developed using React for the front end and Spring Boot for the backend.

1. High-Level Description of the Design

The Route Planning Application is a comprehensive tool for creating and managing travel plans based on real-time information, including traffic, road conditions, and weather forecasts. The application's core purpose is to assist users in planning safe routes by analyzing current road and environmental conditions. It offers a dashboard for centralized navigation, detailed route planning options, a history of past routes, and a dedicated weather panel for route-specific conditions.

- **Interactive Dashboard**: Displays current travel plans and navigation options.
- **Route Planning Interface**: Allows users to set start and end locations, view route options, and select transportation modes.
- **Weather Forecast Integration**: Provides real-time weather data relevant to the planned routes.
- **Route History and Saved Routes**: Enables users to view past routes, replan them, and save frequently used routes for quick access.

The application is developed using React for the frontend and Spring Boot for the backend, with a RESTful architecture to facilitate efficient data flow between the frontend and backend services. Additionally, APIs such as Google Maps (for routing and mapping) and a Weather API (for weather forecasts) are integrated for a richer user experience.

Frontend (React)
Built with React for a dynamic and responsive user interface.

- o Components: Dashboard, Sidebar, Route Planning, History, Saved Routes, Weather, Settings, Help.
- o Styling and UI/UX: Prototyped in Figma, ensuring consistency and user-friendly interactions.

- **Dashboard**: Central hub showing key functionalities like planning a new route, accessing history, viewing saved routes, and checking route-specific weather.
- **Sidebar**: A static navigation menu offering links to the primary sections, including:
  - o **Dashboard**: Quick access to main functionalities.
  - o **Routes History**: Displays a searchable list of past routes, with options to replan or delete routes.

- o **Saved Routes**: A repository of frequently used routes, which can be saved by users for easy access.
  - o **Weather Panel**: Provides weather information for specific routes, including forecasts and real-time data.
  - o **Download**: Enables users to download route information.
  - o **Log Out**: Located at the bottom of the sidebar to allow for easy session termination.
- **Top Navigation Bar**: Provides additional links to:
  - o **Add New Plan**: Starts a new route planning session.
  - o **History, Settings, and Help**: Quick access to these respective pages.

Backend (Spring Boot)

Developed with Spring Boot for robust handling of data and API integrations.

- o APIs: Google Maps API (for route mapping), Weather API (for live weather updates).
  - o Database: SQL for storing routes and user data.
- **Libraries & Dependencies**:
  - o React Router: For managing page navigation.
  - o Axios: For handling HTTP requests.
  - o Google Maps API: For route visualization and distance calculation.
  - o Weather API: For retrieving current and forecasted weather conditions.

- **Route Controller**: Manages route-related functionalities, including route retrieval and connection to the Google Maps API.
- **Weather Controller**: Interfaces with a Weather API to fetch weather data based on route coordinates.
- **User Controller**: Manages user-specific functionalities, including authentication, session control, and profile settings.

2. Architecture Overview

The application follows a RESTful architecture where the frontend and backend communicate through API requests and responses in JSON format. This separation enables the following:

- **Modularity**: The React frontend is fully separated from the backend logic, making both parts independently maintainable.
- **Scalability**: The backend can support more features or APIs without major changes to the frontend.
- **Extensibility**: New functionalities, such as additional transport modes or new weather data sources, can be integrated without impacting existing features.

Frontend (React) is developed as a Single Page Application (SPA), which enables a smooth and uninterrupted user experience by dynamically loading content without refreshing the page. The backend, implemented in Spring Boot, is organized based on MVC principles (Model-View-Controller), ensuring clear separation of concerns.

2.1. Visual Overview of Figma Prototype

2.1.1. Dashboard Screenshot

- **Sidebar**: On the left, a vertical menu provides access to Dashboard, History, Saved Routes, Download, and Weather.
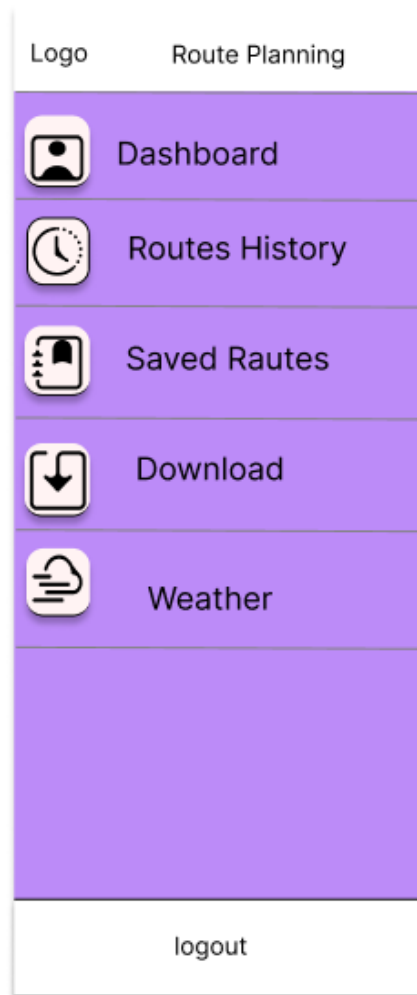
- **Top Navigation Bar**: Includes buttons for adding a new plan, navigating to History, Settings, and Help.
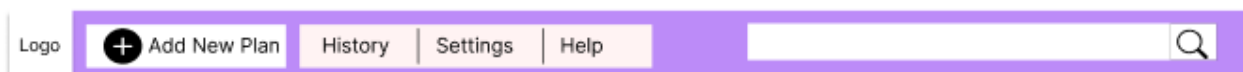
2.1.2. Navigation Flow (Sidebar & Top Bar)

The Sidebar offers consistent navigation across all pages, allowing users to quickly switch between features. Each section is clearly labeled for ease of

use. In the Top Bar, additional links are available to facilitate creating new plans and accessing settings.

Sidebar Elements:

- **Dashboard**: Main page with route planning options.
- **Routes History**: Quick access to recent routes with a search functionality.
- **Saved Routes**: Displays user-saved routes for easy re-access.
- **Download**: Allows users to download route data.
- **Weather**: Shows weather information for specific routes.

Top Bar Elements:

- **Add New Plan**: Opens the route planning interface.
- **History**: Takes users to past route information.
- **Settings and Help**: Provides application settings and support information.

Tampere University

3. Describing Components and Their Responsibilities

3.1. Frontend Components

The front end, developed in React, comprises several key components, each responsible for handling various aspects of the route planning application. The following sections describe each component's purpose and functionality.

3.1.1. Dashboard

The Dashboard is the central hub, allowing users to view route suggestions, access other main pages, and manage route planning tasks. This is the landing page users see upon logging in.

Responsibilities:

- Display current route options and quick links to add new plans.
- Provide access to recently planned routes and saved routes for easy navigation.
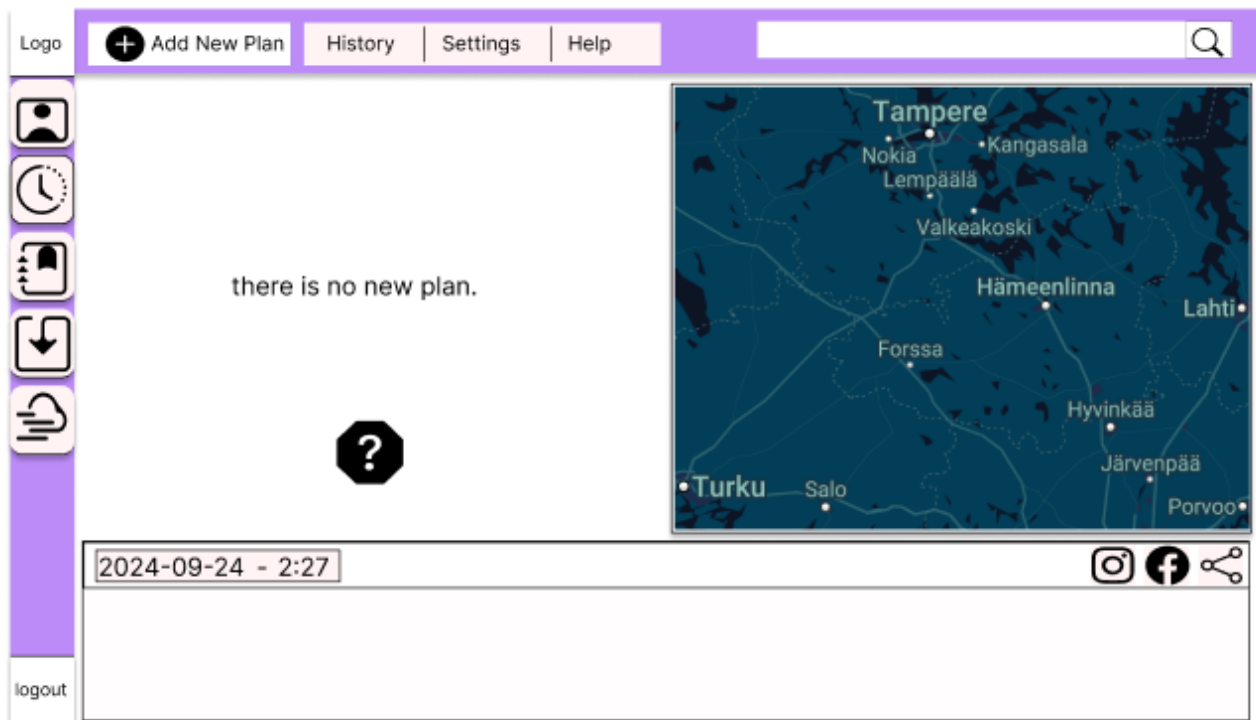
3.1.2. Sidebar

The Sidebar is the primary navigation element that provides consistent access to major sections of the application.

Responsibilities:

- List main navigation options: Dashboard, Routes History, Saved Routes, Download, Weather, and Log Out.
- Enable quick navigation between sections with a clean, intuitive layout.
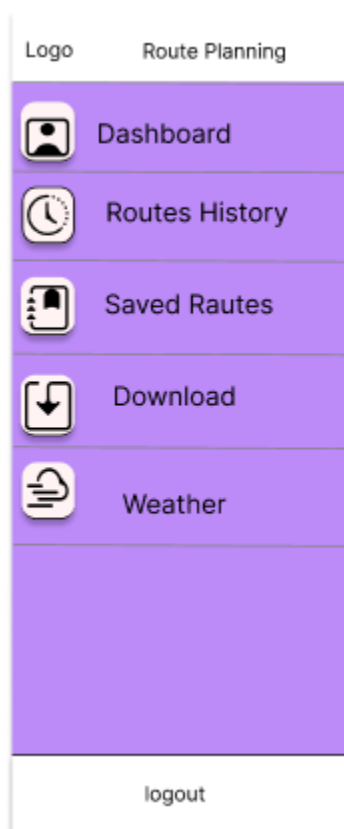
3.1.3.   Route Planning Page

The Route Planning Page allows users to specify start and destination points, set travel preferences, and generate a route. This page is essential for managing the route planning process.

Responsibilities:

- Accept user inputs for starting point, destination, and mode of transport.
- Fetch and display route options, time, and distance based on user inputs.
- Allow users to save or share their planned route.
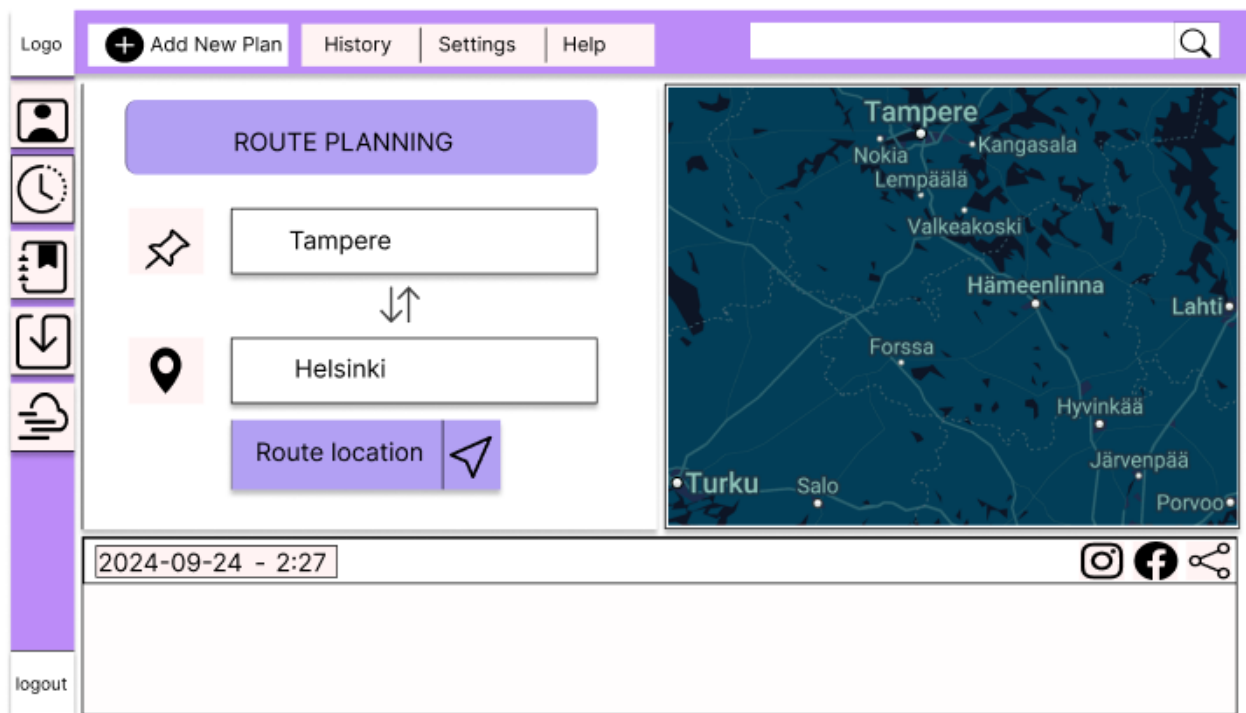
3.1.4. History Page

The History Page displays a list of recently traveled or planned routes, allowing users to view past route details and replan if needed.

Responsibilities:

- Display a list of past routes with search functionality for easy navigation.
- Provide a "Replan" option for each route, allowing users to adjust previous plans.
- Show transportation mode, distance, and duration details.
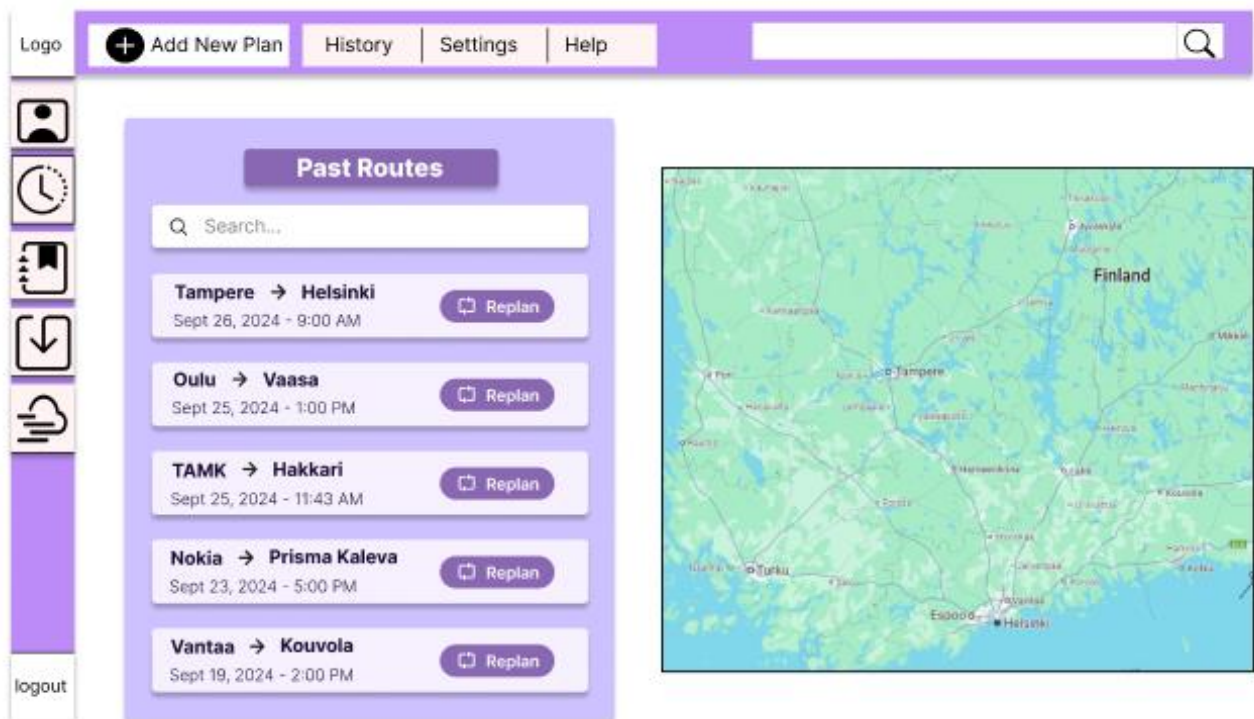
3.1.5. Saved Routes

The Saved Routes page provides a repository of frequently used or favorite routes, allowing users to quickly access and manage them.

Responsibilities:

- Display saved routes with details like the route history.
- Offer options to print saved routes for offline use.

3.1.6. Weather Panel

The Weather Panel provides route-specific weather information to help users make informed travel plans.

Responsibilities:

- Display real-time weather updates for planned routes, including conditions like temperature, humidity, and potential hazards.
- Update information based on changes in route or time, ensuring users are informed of relevant weather changes.
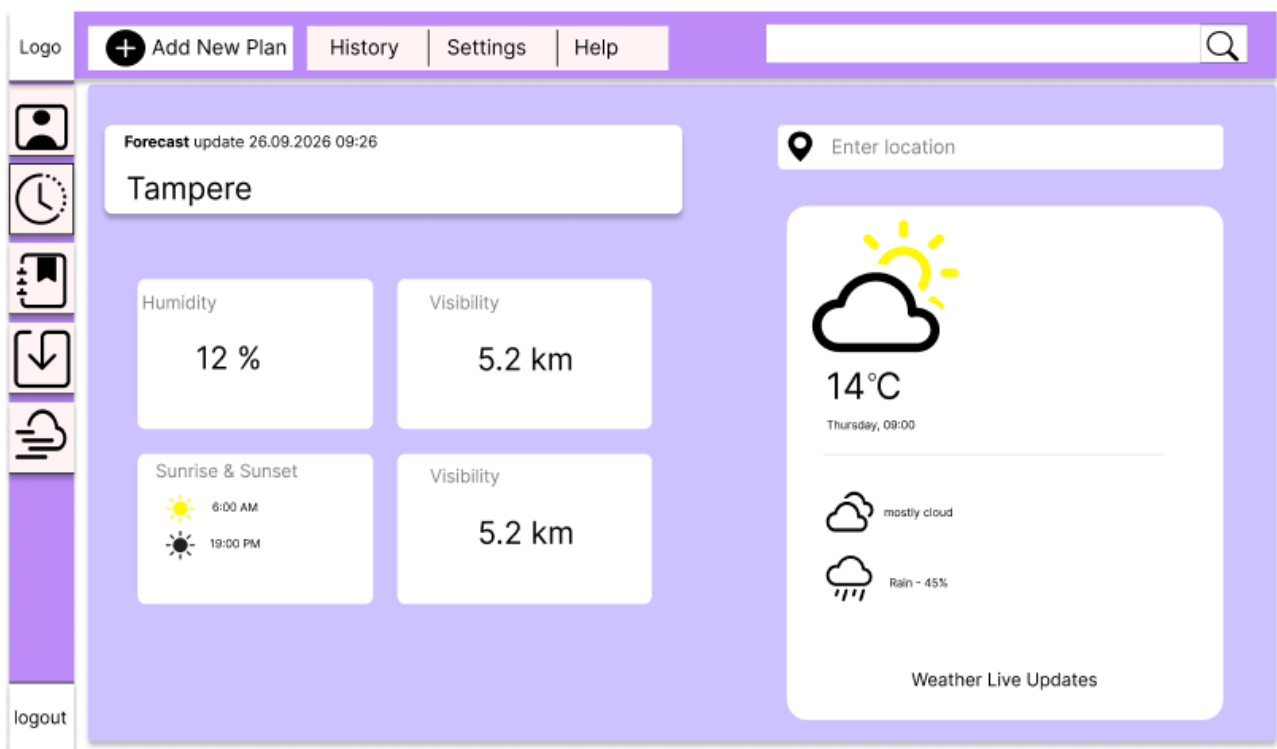
3.2. Backend component

We have adopted a Git-based workflow to efficiently manage collaboration and code sharing among team members. This workflow supports distributed task assignments and progress tracking, allowing each team member to focus on specific components of the project. By using Git issues and tickets, we maintain a clear overview of ongoing work, resolved issues, and contributions from each team member.

3.3. Git Repository Structure

Our Git repository is organized as follows:

- **Main Branch**: This branch holds the stable, production-ready code. Code is merged here only after thorough testing.
- **Development Branch**: This branch serves as the integration branch where team members submit pull requests for review.
- **Feature Branches**: Each feature or issue is developed on its own branch. Naming conventions like *feature/dashboard*, *bugfix/route-planning*, and *enhancement/weather-panel* are used to identify the purpose of each branch clearly.

3.3.1. Issue Tracking and Task Allocation

We use Git issues to create, assign, and track tasks within the team. Here's how this process works:

- **Issue   Creation**:
  Each team member can create issues based on new features, bugs, or enhancements needed in the project.
- **Assignment   of   Issues**:
  Team members assign themselves to issues by selecting a ticket based on their area of expertise or the part of the project they are responsible for. This self-assignment approach fosters accountability and lets each member choose tasks they're comfortable working on.
- **Resolving Issues:**
  Once a team member is assigned an issue, they create a dedicated branch for the task. After completing the task, they submit a pull request to the development branch, which is then reviewed by other members before merging.

3.3.2. Benefits of Git Issue Tracking

This approach provides several key benefits:

- **Clear Accountability**: Each member's contributions are transparent, and the system records who resolved which issue and when.
- **Efficient Collaboration**: By breaking down the project into smaller issues, team members can work concurrently on different parts, speeding up the development process.
- **Progress Monitoring**: The issue board offers a snapshot of current progress, highlighting completed tasks, ongoing work, and pending issues.

3.4. Issue Board

Below is the Git issue board distribution, demonstrating how the issues are assigned:

Issue Number	Development Area	Status	Member Name
18	Backend	Completed	Masood Ahmadi
1, 4, 6, 7, 8, 1	Backend	In Progress	
12, 14	Frontend	Completed	Md Ashfak Haider Nehal
		In Progress	
23, 26	Backend	Completed	Selina Rautakoski
37	Backend	In Progress	
20, 28, 29, 47, 48, 49, 50, 51,52, 53, 54, 58, 59	Backend	Completed	Kashif Rasool
40, 55, 56, 57	Backend	In Progress	
		Completed	Salman Khan
30, 31	Backend	In Progress	

4. Interfaces

4.1.   Interface Diagrams and Examples

4.1.1.   Route API Interface Example

4.1.2.   Weather API Interface Example

4.1.3.   User API Interface Example

5. Components' Internal Structure

5.1.   Route Planning Component (Frontend)

5.2.   Route Controller (Backend)

5.3.   Diagrams for Internal Relationships

5.4.   Component Diagram: Route Planning

5.5.   Flowchart: History Component

6. Design Patterns

6.1.   Model-View-Controller (MVC) Pattern,

This application's a backend implemented with java spring-boot's web layer on the MVC architecture, @controller and rest Controller, with model it represents the data transfer and entity classes. The view layer is managed by templating engines like Json in rest API.

7. Self-Evaluation

7.1. Anticipated Changes

8. References & AI Assistance

8.1. References

This section documents all third-party resources, libraries, and APIs that were utilized in the development of the Route Planning Application. These tools and resources were essential in enabling certain functionalities and optimizing the project workflow.

- **Google Maps API**
    - Used for route calculation and map display functionality.
    - **Documentation**: Google Maps API
- **React Library**
    - Utilized for the frontend, allowing for efficient and modular UI development.
    - **Documentation**: React
- **Spring Boot Framework**
    - Used to develop the backend, providing a solid foundation for REST API development.
    - **Documentation**: Spring Boot
- **GitLab**
    - Repository management, version control, and issue tracking.
    - **Platform**:

8.2. AI Assistance

ChatGPT helped us to design applications like weather monitoring dashboards and route planning.

Below are the shown examples from the ChatGPT:

ChatGPT ⌄

6. **User Interface:**
   - Simple, clean dashboard that shows a summary of road and weather conditions in specific areas or routes.
   - Ability to zoom into regions of interest and view detailed weather or traffic reports.

Technical Considerations:
- **DigiTraffic API** will provide the traffic-related information (real-time and historical traffic data, incidents, congestion reports).
- **FMI Open Data API** will provide weather conditions and forecasts.
- No third-party credentials needed, as both APIs are open and public.
- The project can be implemented as a **web application** using technologies like **React.js** for the frontend, **Node.js/Flask** for the backend, and **D3.js/Google Maps API** for visualization.

Potential Challenges:
- Handling large amounts of real-time data, especially from traffic sources.
- Integrating and synchronizing traffic and weather data in a meaningful way for the user.

Extensions (Optional):
- Add a mobile app version of the dashboard for users on the go.
- Integrate additional APIs such as the **Statistics Finland API** to provide broader insights into how traffic and weather affect certain areas over time.

This prototype idea is practical, leverages real-time data, and provides an application with clear value for users interested in safe travel.

‹ 3/3 ›

↓

Message ChatGPT

9. Conclusion