Route Optimization System Architecture

Summary of the Technology Stack

Frontend Application

- **TypeScript** - Strongly typed programming language that builds on JavaScript
- **ReactJS** - Frontend library for building user interfaces
- **Tailwind CSS** - Utility-first CSS framework for rapid UI development
- **Shadcn UI** - Component library built on Tailwind CSS (replacing JoyUI from initial plan)
- **Tanstack Router** - Type-safe routing for React applications
- **Tanstack Query** - Data synchronization library for fetching, caching, and updating data
- **Google Maps API** - Provides maps visualization and route rendering

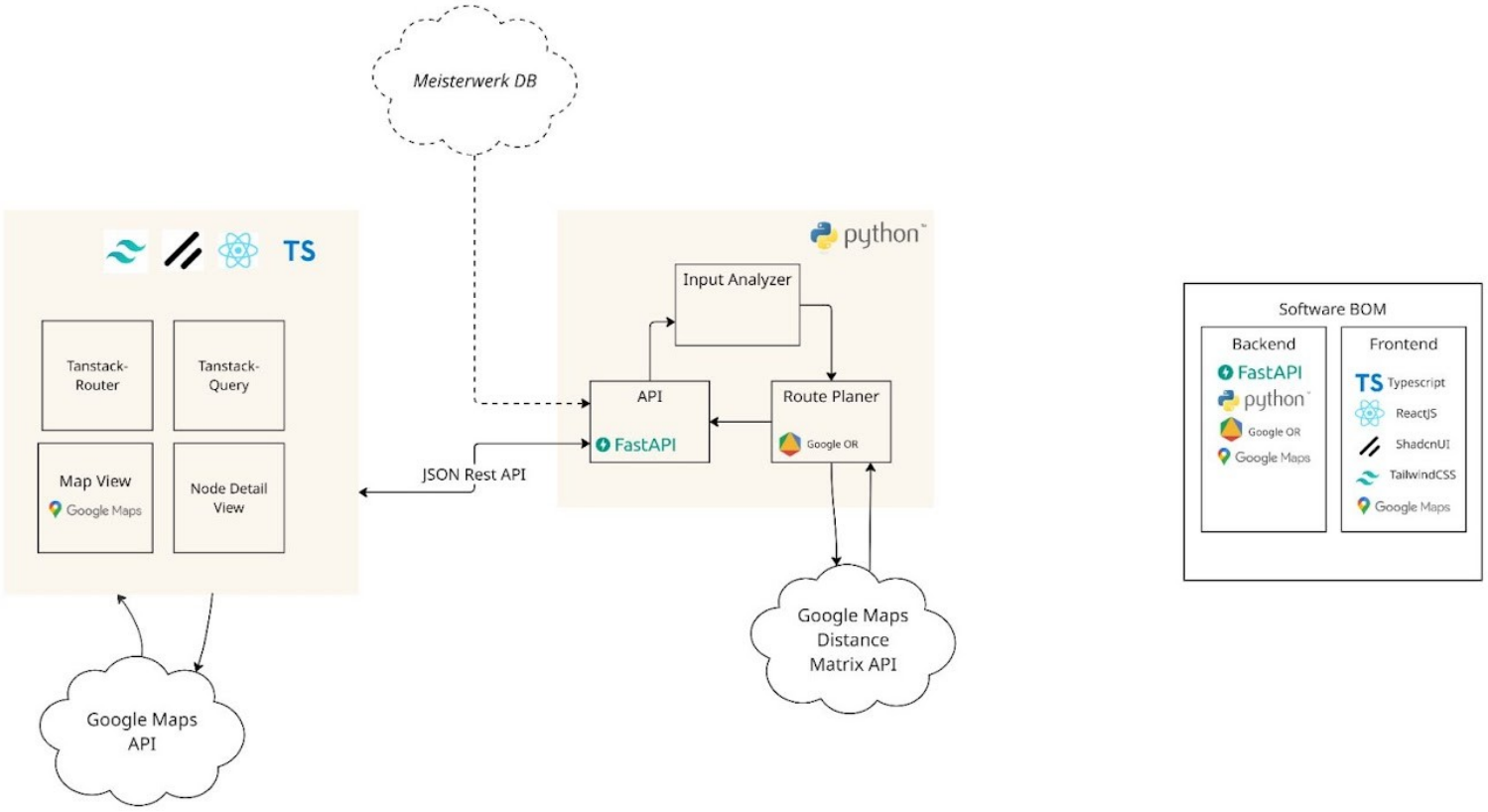
Backend Application

- **Python** - Programming language for the backend logic
- **FastAPI** - High-performance web framework for building APIs
- **Google OR-Tools** - Optimization library for solving route planning problems
- **Google Maps Distance Matrix API** - Provides accurate travel distances and times

Overview Diagram of Code Components (Static View)

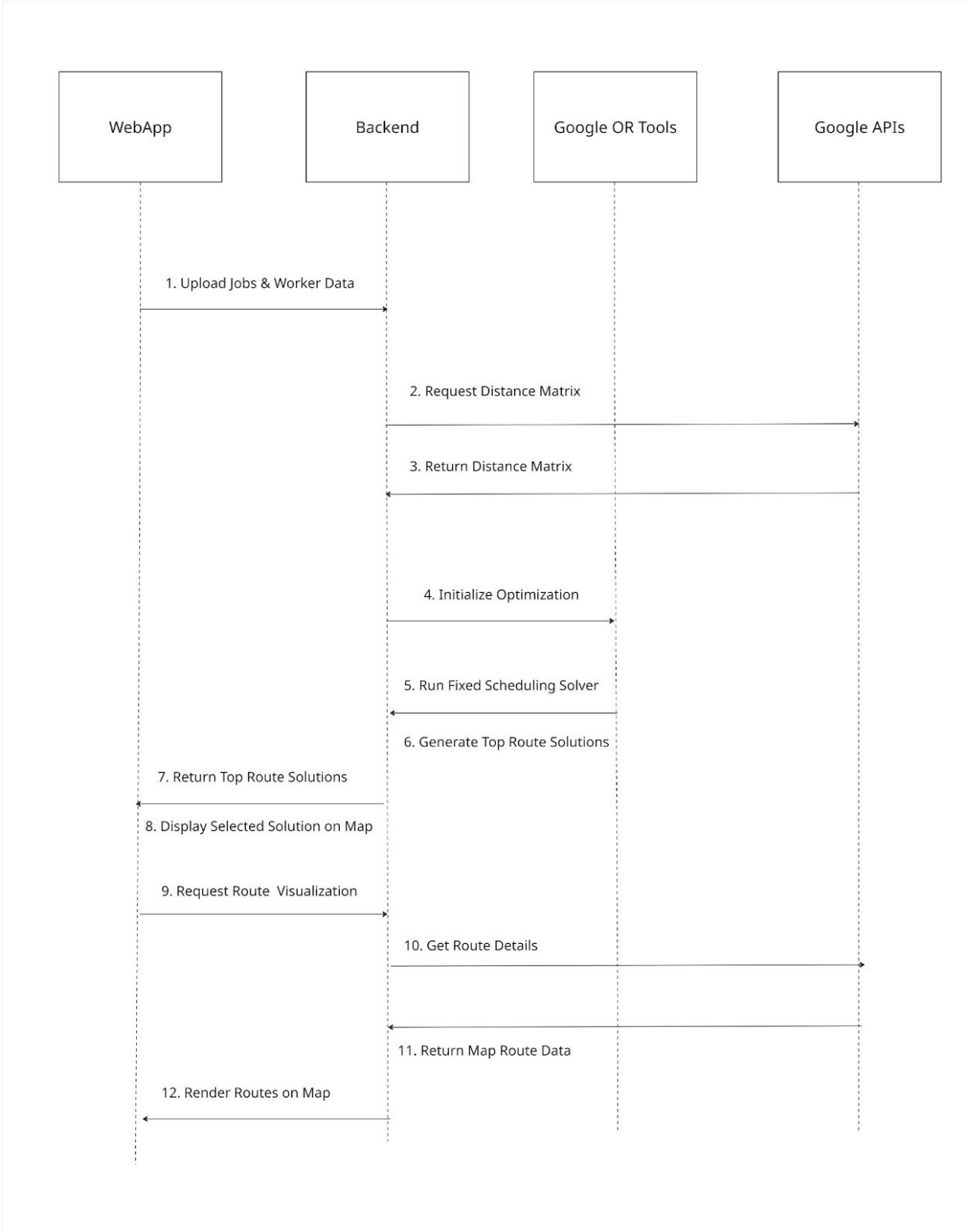
Frontend
The frontend of the application is built using Shadcn UI for accessible, consistent components and styled with Tailwind CSS. The layout includes key components such as a Map View (for visualizing routes), Schedule View (daily plans), Worker Details (individual schedules), and a Solution Comparison tool. Navigation is handled with Tanstack Router for type-safe routing. Data fetching and caching are managed with React Query, with local state handling for UI preferences and input validation via an Input Analyzer. The Google Maps integration enables route visualization, map controls, and geocoding for address-to-coordinate conversion.

Backend
The backend is built with FastAPI to provide high-performance RESTful endpoints. Core business logic is organized into service classes, data models for validation, and shared utilities. The optimization engine uses Google OR-Tools with separate solvers for fixed and flexible scheduling, supported by a constraints handler and a route calculator. External services include a Google Maps API client for distance matrix calculations and modules for importing/exporting JSON data.

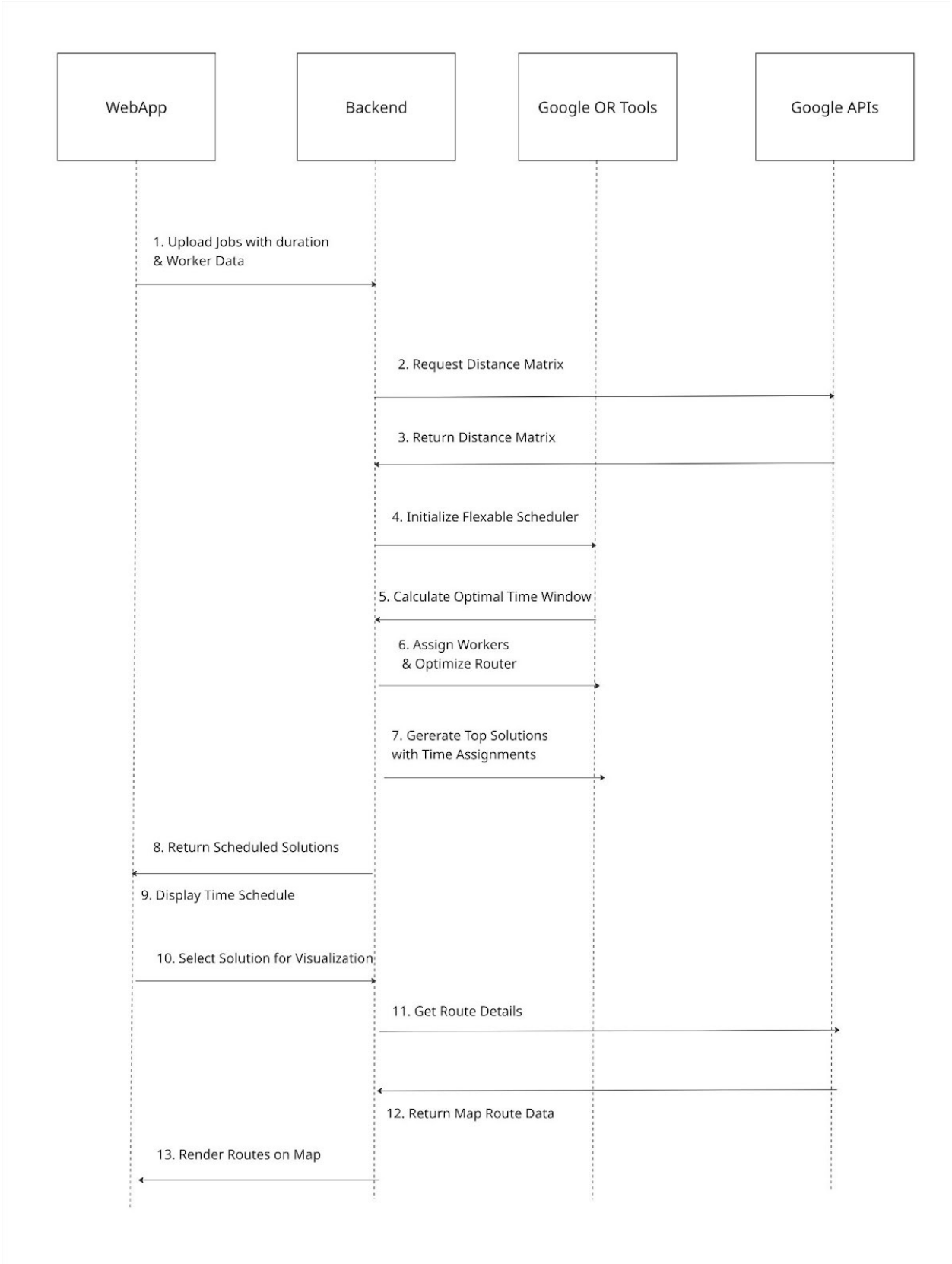

Meisterwerk DB
TS
Tanstack-Router
Tanstack-Query
Map View
Google Maps
Node Detail View
JSON Rest API
python
Input Analyzer
API
FastAPI
Route Planer
Google OR
Google Maps API
Google Maps Distance Matrix API
Software BOM
Backend
FastAPI
python
Google OR
Google Maps
Frontend
TS Typescript
ReactJS
ShadcnUI
TailwindCSS
Google Maps

Runtime Scenarios

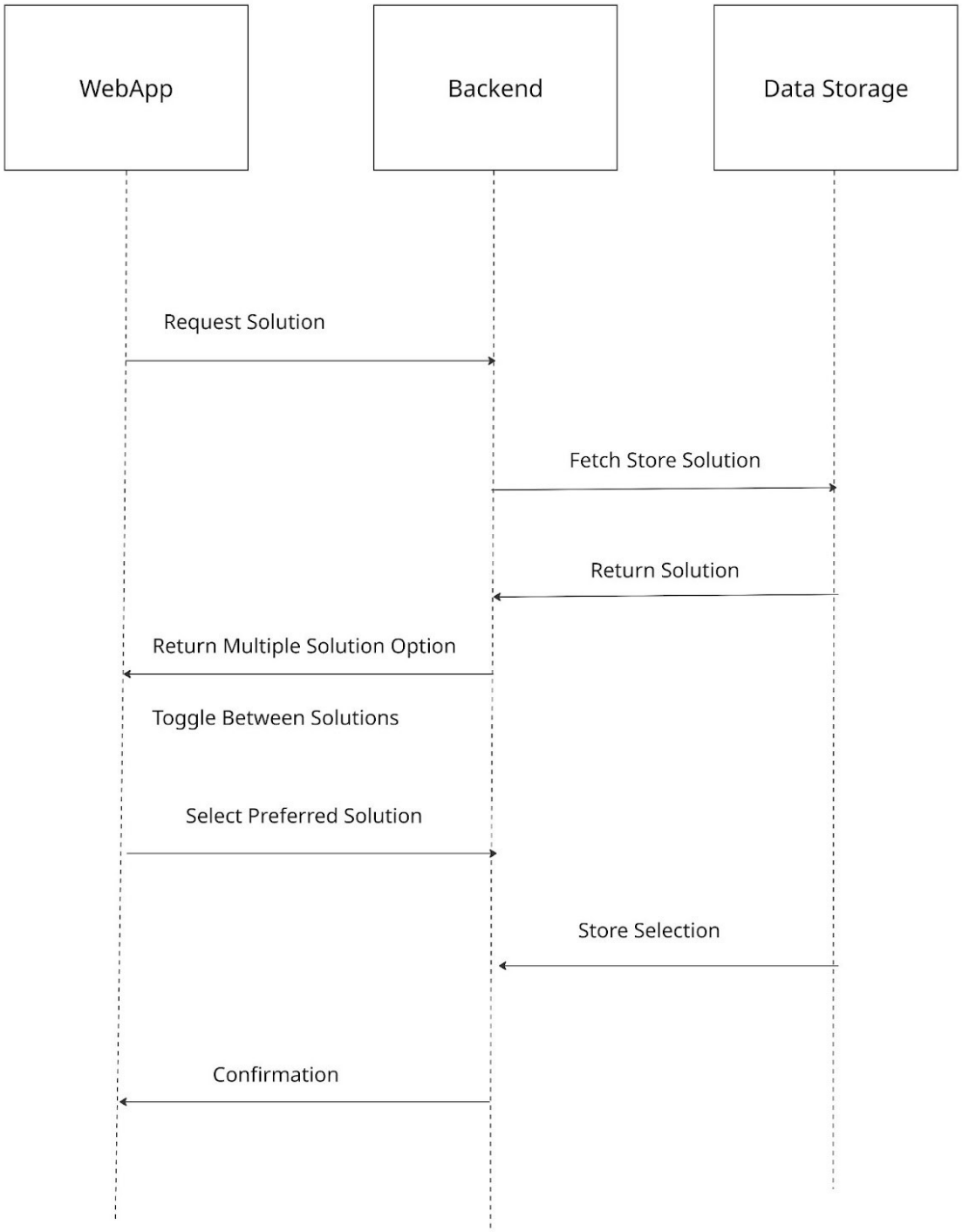
Scenario 1: Job Optimization with Fixed Scheduling



Scenario 2: Flexible Scheduling Optimization



Solution Comparison and Selection



Scenario Explanations

Scenario 1: Job Optimization with Fixed Scheduling

This scenario handles the case where appointments have fixed times. The system takes job and worker data, obtains a distance matrix from Google Maps, and uses Google OR-Tools to generate optimal worker assignments and routes. The resulting solutions are ranked by criteria like minimized travel time and maximized profit, with the top options sent back to the frontend for visualization and selection.

Scenario 2: Flexible Scheduling Optimization

In this scenario, jobs come with durations rather than fixed times. The system must both assign workers to jobs and determine the optimal timing for each job. Google OR-Tools handles this more complex optimization problem, considering worker qualifications, job requirements, and travel times to create a schedule that minimizes travel time while maximizing profit.

Solution Comparison and Selection

After generating multiple optimization solutions, this scenario allows users to compare them side by side. The frontend provides controls to toggle between different solutions, visualizing their differences in terms of routes, schedules, and KPIs (like total travel time, cost, and profit). Users can select their preferred solution, which is then stored for implementation.