

High-Level Design Report

1. Introduction

1.1 Purpose of the system

The "Roadrunner" is an AI-powered travel assistant designed to enhance the itinerary planning process. Its primary purpose is to centralize and personalize the travel planning lifecycle, which is currently fragmented across multiple platforms. By utilising Large Language Models (LLMs), Natural Language Processing (NLP), and Data Mining, the system transforms unstructured user "stories" into optimized, executable travel plans. It acts as a smart intermediary, aggregating data from third-party sources (Wikipedia, Google Places, OpenStreetMap) to provide a unified, stress-free planning experience.

1.2 Design goals

Personalization and Intelligence: The architecture must support advanced NLP processing to deeply understand user intent and context.

Performance and Responsiveness: Initial plan generation must complete within 30 seconds, and UI interactions must provide feedback within 500ms.

Scalability: The system is designed to support at least 100 concurrent users initially, with a microservices-inspired structure to allow independent scaling of computationally intensive components.

Reliability and Robustness: Target uptime is 99.5%. The system must handle third-party API failures gracefully.

Security & Privacy: Adherence to GDPR and KVKK is fundamental. All data in transit must be encrypted (HTTPS/TLS), and sensitive user data must be protected.

Modularity: The system is decomposed into distinct subsystems (Chatbot, Data Extractor, Route Engine) to facilitate parallel development and maintenance.

1.3 Definitions, acronyms, and abbreviations

POI (Point of Interest): A specific location such as a museum, restaurant, or park (also referred to as a Waypoint).

User Story: Natural language input describing the user's travel desires.

Vertex: A geographical search radius or area for planning.

1.4 Overview

The Roadrunner is a web-based application utilizing a client-server architecture. The Client Side offers a synchronized dual-interface experience comprising a Chatbot and an Interactive Map. The Server Side orchestrates the logic for input parsing, route optimization, and data management. Crucially, the system aggregates data from external APIs but persists this information in its own database to ensure consistency, reduce API costs, and improve latency. Furthermore, the Chatbot utilizes backend-driven tool calling mechanisms to directly manipulate the map state (adding or removing POIs) upon user confirmation, bridging the gap between conversational intent and visual planning.

2. Current software architecture (if any)

Currently, no single unified system exists for the proposed solution. The "current system" refers to the manual, fragmented process of modern travel planning. Users are forced to navigate a distributed ecosystem of unconnected websites and applications to research destinations, compare transport and accommodation options, and organize daily activities.

This traditional approach has several flaws, listed below:

- **Information Overload:** Users must sift through massive amounts of irrelevant information.
- **Fragmentation:** Data is distributed across blogs, booking sites, and map applications without a central hub.
- **Low Utility Content:** Most web content is filled with ads and sponsored material that are not useful to the users.
- **Generic Filters:** Existing tools offer generic filters (price, rating) but fail to understand nuanced preferences like "quiet cafes" or "historical battlefields."

3. Proposed software architecture

3.1 Overview

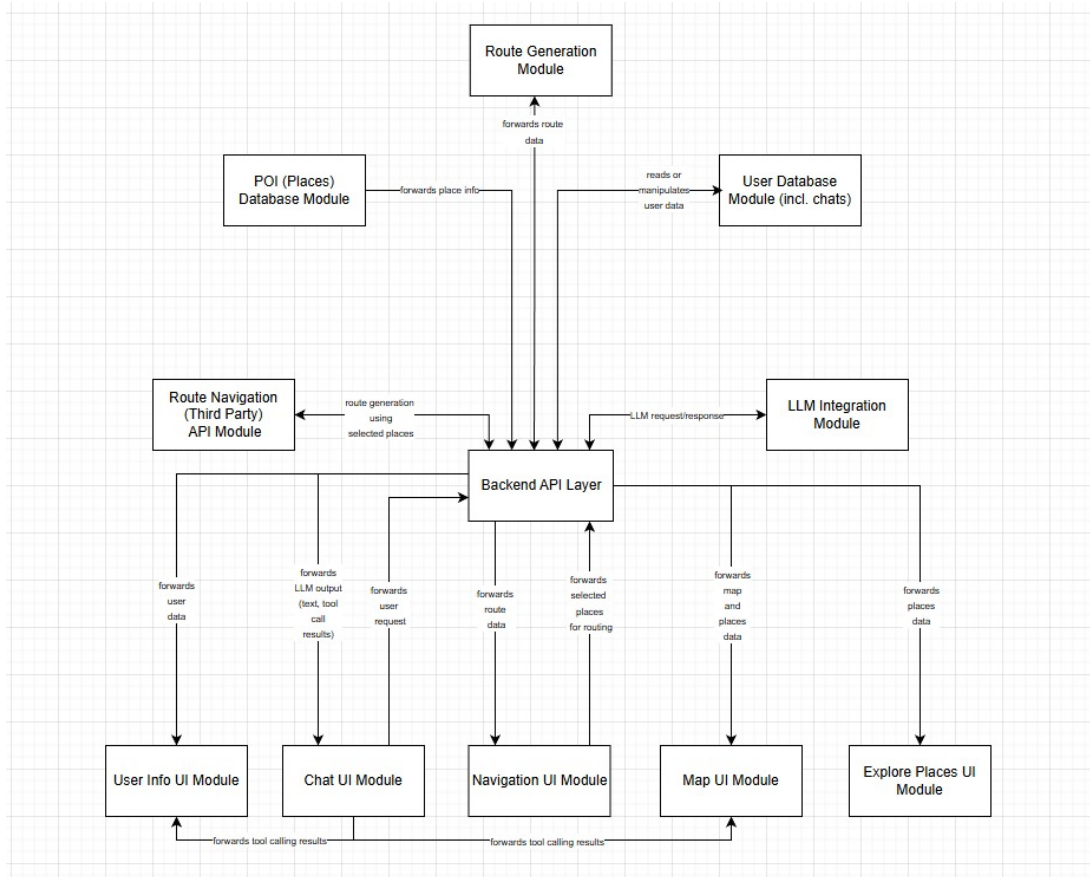
The proposed architecture follows a Microservices-based (or Modular Monolith initially) approach, containerized using Docker. The core logic is separated into distinct processing pipelines: Intent Recognition, Data Retrieval, Scoring/Ranking, and Route Optimization.

3.2 Subsystem decomposition

The system is decomposed into the following major subsystems:

- **Backend API Layer (Orchestration):** This module serves as the centralized API Gateway and controller for the system. It implements the Facade pattern, abstracting the complexity of underlying services from the client. Its primary role is to validate incoming requests, route them to the appropriate specialized modules (such as logic processing or data retrieval), and aggregate asynchronous responses, particularly those from the LLM and Route Generator, into a unified synchronous response format for the UI.

- **Route Generation Module (Optimization Engine):** This computational engine is dedicated to solving the routing logic. It encapsulates algorithms for pathfinding and itinerary optimization, processing constraints such as travel time, geographical sequence, and user preferences to produce viable travel plans.
- **LLM Integration Module (AI Agent):** This module manages the lifecycle of interactions with the Large Language Model. It is responsible for prompt construction, context window management, and parsing unstructured natural language into structured system commands (tool calls), thereby bridging the gap between user intent and executable software logic.
- **POI (Places) Database Module:** Acts as the single source of truth for geographical data, storing normalized attributes of Points of Interest (location, metadata, categories) to ensure consistency and reduce reliance on external APIs during runtime.
- **Route Navigation API Module (External Interface):** A specialized adapter module designed to communicate with third-party navigation providers. It translates internal route data into turn-by-turn navigation instructions required for the final itinerary execution.
- **Chat UI Module:** Provides the conversational interface, handling the rendering of LLM responses and capturing natural language input.
- **Map & Navigation UI Modules:** These modules handle the visualization of spatial data, rendering interactive maps, route overlays, and directions based on backend data.
- **Explore & User Info Modules:** Dedicated components for content discovery (browsing places) and user account management (profile settings, history).



3.3 Hardware/software mapping

- **Client Node:** Any device with a modern web browser (Chromium, Firefox, Safari). Handles UI rendering.
- **Application Server:** Cloud instances (e.g., AWS EC2, GCP Compute Engine) hosting the backend containers.
 - **Resource Needs:** Moderate CPU/RAM for API handling; potentially higher for local routing algorithms.
- **Data Server / Database:**
 - **Relational DB (PostgreSQL):** The central repository storing the massive dataset of aggregated POIs, normalized blogger data, user profiles, and saved itineraries. It acts as the primary data source for all runtime operations.
 - **Cache (Redis):** Stores frequently accessed database query results and user session states to ensure sub-500ms UI response times (Performance optimization).
- **External AI Services:**
 - Google Gemini / OpenAI API for LLM processing.

3.4 Persistent data management

User Data: Stored in a relational database. Includes encrypted credentials, profile tags, and past trip logs.

Itinerary Data: Generated routes are serialized (JSON) and stored to allow users to save, share, and revisit plans.

Data Retrieval Strategy: To mitigate "Economic Constraints" (API costs) and improve "Performance," the system retrieves POI data exclusively from its internal database. External APIs are used strictly for data ingestion and updates, ensuring that runtime operations do not rely on live third-party API calls.

3.5 Access control and security

Authentication: Users must register/login to save plans. Passwords are hashed using Argon2 or scrypt.

Authorization: Role-based access control. Users can only access/modify their own itineraries.

Communication Security: All data in transit is encrypted via HTTPS/TLS.

API Security: API keys for third-party services are managed via a secret management system (AWS Secrets Manager or environment variables), never hardcoded.

Privacy: Compliance with GDPR/KVKK ensures users can request data deletion.

3.6 Global software control

The system operates primarily on an Event-Driven and Pipeline control model for plan generation:

1. Trigger: User submits a story.
2. Pipeline Step 1 (NLP): Extract Intent -> Return structured constraints.
3. Pipeline Step 2 (Data): Query DB -> Aggregate POIs.
4. Pipeline Step 3 (Logic): Score POIs -> Filter top N.
5. Pipeline Step 4 (Opt): Run Routing Algorithm on top N POIs.
6. Response: Return JSON to frontend.

3.7 Boundary conditions

Initialization: On startup, the system establishes connections to the internal PostgreSQL database and validates configuration for the NLP Engine.

Termination: Graceful shutdown ensures that any in-progress data ingestion tasks are paused safely and user sessions are preserved.

Failure:

- **External Data Source Unavailability:** Because the system retrieves POI data exclusively from the internal database during runtime, failures or outages in third-party APIs (e.g., Google Places) do not disrupt the planning process. The system continues to function using the existing, persisted dataset.
- **No Solution:** If user constraints are technically impossible to satisfy ("See all of London in 1 hour"), the system returns a polite explanation and suggests relaxing specific constraints (time, budget, or location).

4. Subsystem services

● Backend Service (API Layer)

- Serves as the central entry point and orchestrator. It manages the communication lifecycle between the frontend and specialized backend modules. It is responsible for request validation, session management, and aggregating partial data from multiple internal sources into a cohesive response.
- **Responsibilities:**
 - Validating incoming request headers and body structures.
 - Routing authenticated requests to LLM, Routing, or Database modules.
 - Aggregating asynchronous results from the AI agent and the routing engine.
- **Input:** Raw user requests (text or interaction events), authentication tokens.
- **Output:** Aggregated structural responses (UI-consumable format), error status codes.

● Route Generation/Optimization Service

- Encapsulates the core logic for travel planning. This service accepts a set of desired locations and constraints to produce an optimized itinerary sequence.
- **Responsibilities:**
 - Calculating distances between multiple geographical points.
 - Solving the routing problem based on time windows and geographical proximity.
 - Applying logical constraints.
- **Input:** Selected places, user preferences.
- **Output:** Waypoints (itinerary), total estimated duration, and distance metrics.

● Chat UI Service

- Manages the state of the conversation and renders appropriate UI components based on the message type (text vs. interactive card).
- **Responsibilities:**
 - Capturing user input and managing loading states.
 - Rendering structured responses embedded in the chat stream.
- **Input:** User text, System Responses.
- **Output:** Interaction events forwarded to the backend.

- **Map UI Service**
 - Handles the rendering of map layers, markers, and route paths. It synchronizes the visual map state with the current itinerary data.
 - **Responsibilities:**
 - Plotting POI markers on the interactive map.
 - Drawing route polylines representing the travel path.
 - Handling user interactions (clicks, zooms) on map elements.
 - **Input:** Points, Lines.
 - **Output:** Visual map updates, user selection events.
- **Places Data Management Service:**
 - Manages the storage and retrieval of static geographical data. It abstracts the underlying database technology, providing a semantic interface for querying places.
 - **Responsibilities:**
 - Indexing places by location and category.
 - Normalizing data ingested from various sources.
 - Serving detailed metadata.
 - **Input:** Search criteria or Place Identifiers.
 - **Output:** Place Objects containing descriptive metadata and coordinates.
- **External Navigation Interface Service**
 - Acts as an adapter for third-party mapping providers. It isolates the system from external API changes and translates internal route data into navigation-ready formats.
 - **Responsibilities:**
 - Fetching turn-by-turn directions for the finalized itinerary.
 - Retrieving real-time traffic estimations if available.
 - **Input:** Geographical coordinates.
 - **Output:** Navigation polyline data, step-by-step instruction sets.
- **User Identity & Context Service:**
 - Manages persistent user state, including authentication, profile customization, and historical interaction logs.
 - **Responsibilities:**
 - Verifying credentials and managing session tokens.
 - Storing and retrieving user-specific preference weights.
 - Archiving chat history for context-aware AI processing.
 - **Input:** User credentials, profile updates, chat log entries.
 - **Output:** User profile snapshots, session validation status, historical context data.
- **LLM Integration & Intent Service:**
 - Acts as the cognitive engine of the system. It processes natural language to extract structured intent and determines necessary actions (Tool Calling) without executing the logic itself.
 - **Responsibilities:**

- Constructing context-aware prompts using chat history.
- Parsing natural language into system-understandable intent (e.g., "Add Place," "Create Route").
- Extracting entities (dates, location names) from user text.
- **Input:** Conversational history, current user message, system instructions.
- **Output:** Structured intent definitions, tool execution requests, text responses.

5. Glossary

- **Containerization:** Packaging software with its dependencies.
- **GDPR/KVKK:** General Data Protection Regulation / Kişisel Verilerin Korunması Kanunu.
- **LLM (Large Language Model):** AI models used for processing natural language user stories.
- **Microservices:** Architectural style structuring an application as a collection of services.
- **POI (Point of Interest):** A specific location such as a museum, restaurant, or park (also referred to as a Waypoint).
- **RAG (Retrieval-Augmented Generation):** Technique to enhance LLM responses with external data.
- **User Story:** Natural language input describing the user's travel desires.
- **Vertex:** A geographical search radius or area for planning.
- **XAI (Explainable AI):** A subsystem that provides text-based justifications for specific AI recommendations.

6. References

1. Google AI for Developers, 2025.
2. Standard Encyclopedia of Philosophy, "Computer and Information Ethics."