HIGH-LEVEL DESIGN REPORT

GUIDE

(Guided User Itinerary & Destination Explorer)

QUAD-CORE Team

Ebrar Sude Doğan
Erdem Baran
Kayrahan Toprak Tosun
Tuna Kodal

TOBB University of Economics and Technology
Department of Computer Engineering
BIL 495 / YAP 495 Graduation Project

Version: 1.0
Date: January 2026

Contents

1 Introduction

This document presents the High-Level Design (HLD) of the GUIDE system. The purpose of this document is to describe the overall architectural structure of the system, identify its major subsystems, and outline their responsibilities and interactions at a conceptual level.
GUIDE (Guided User Itinerary & Destination Explorer) is designed as a software system that generates structured travel itineraries based on user preferences and predefined data sources. The HLD focuses on architectural decisions, subsystem boundaries, and design rationale, while implementation-level details are intentionally excluded.

1.1 Purpose of the system

The purpose of the GUIDE system is to provide a scalable, modular, and maintainable software platform that supports itinerary generation and route-oriented travel planning. The system is designed to deliver its core services reliably while allowing future functional and non-functional extensions without requiring major architectural changes.
This High-Level Design (HLD) document describes the overall architectural structure of the system, identifies its main subsystems, and explains their responsibilities and interactions. The document focuses on high-level design decisions and subsystem boundaries, and intentionally excludes low-level implementation and technology-specific details.

1.2 Design goals

The design goals define the architectural principles that guide the high-level design of the GUIDE system. These goals are derived from expected usage scenarios, system constraints, and long-term maintainability considerations. They serve as criteria for evaluating architectural decisions and subsystem decomposition.

The primary design goals are as follows:

- Modular and loosely coupled subsystems
- Maintainability and extensibility
- Security and controlled access
- Performance-awareness for typical usage (targeting responsive route generation)
- Reliability via isolation of heavy components and controlled dependencies

1.3 Definitions, acronyms, and abbreviations

The following acronyms and abbreviations are used throughout this document.

- **API:** Application Programming Interface
- **CDN:** Content Delivery Network
- **GDPR:** General Data Protection Regulation

- **HLD:** High-Level Design
- **KVKK:** Kişisel Verilerin Korunması Kanunu
- **OSM:** OpenStreetMap
- **OSRM:** Open Source Routing Machine
- **POI:** Point of Interest
- **REST:** Representational State Transfer
- **TTS:** Text-to-Speech
- **UI:** User Interface
- **UX:** User Experience

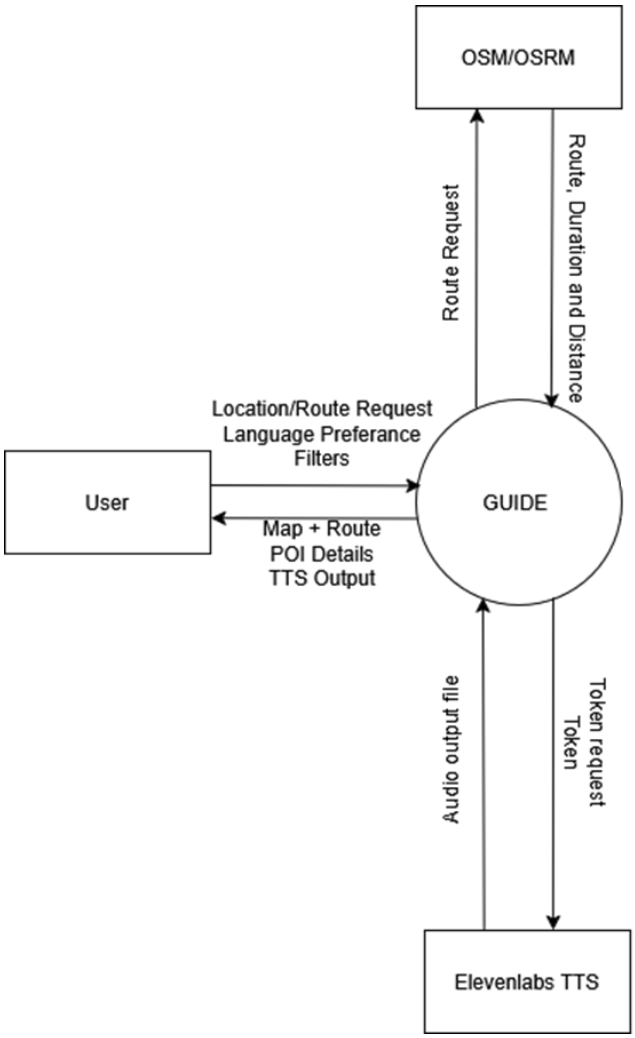

Figure 1: GUIDE system context diagram.

1.4 Overview

The diagram in Figure 1 illustrates the system boundary of GUIDE and the primary external entities that interact with it. This context diagram focuses exclusively on runtime interactions and external dependencies that directly affect system behavior, providing a clear and simplified view of the operational environment. The GUIDE system is shown as a single logical unit, emphasizing its role as the central coordinator between users and external services.
End users interact with GUIDE by providing input parameters and receiving an itinerary and routing outputs. Routing and map-related computations are delegated to external OSM [4] and OSRM [5] services, which supply geographic data and route calculations required by the system. In addition, text-to-speech functionality is supported through an external TTS provider, ElevenLabs [7], which enables the generation of audio content used by GUIDE.

2 Current software architecture

At the current stage, the system exists primarily at the design and documentation level. Existing components are conceptual and serve as architectural references. Therefore, this section is left empty.

3 Proposed software architecture

3.1 Overview

The proposed architecture follows a hybrid Layered and Service-Oriented approach. Responsibilities are separated across presentation, application logic, data management, and integration layers to support scalability, maintainability, and clear separation of concerns.
The layered structure provides a systematic organization of system responsibilities, allowing each layer to evolve independently with minimal impact on others. Service-oriented boundaries are applied within and across layers to encapsulate core functionalities such as itinerary generation, routing orchestration, and content delivery.
This architectural approach enables the system to accommodate future growth, integrate additional services, and adapt to changing requirements while preserving architectural clarity and stability.

Architecture summary. The diagram in Figure.2 illustrates the major layers and core services of GUIDE. It highlights how user-facing components connect to the API gateway layer and how backend services collaborate with data storage and routing components.

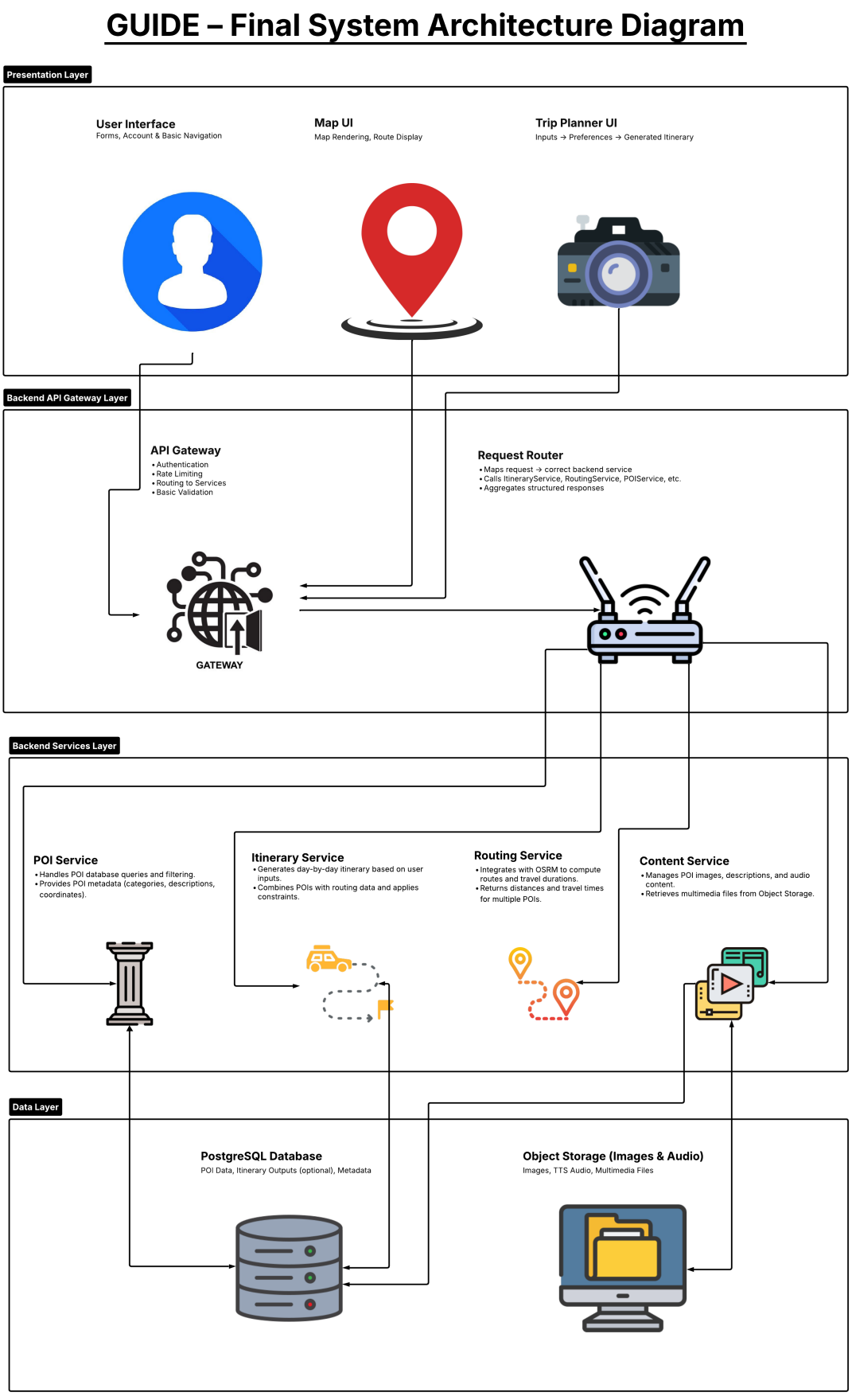

Figure 2: High-level architecture of GUIDE (layers and core services).

3.2 Subsystem decomposition

The system is composed of the following subsystems:

- **Presentation Layer:** Web-based UI components such as map visualization and itinerary display.
- **Backend API Gateway Layer:** Request validation, routing, and basic protection mechanisms such as request control and rate limiting.
- **Backend Services Layer:** Core services such as POI service, itinerary generation, routing orchestration, and content delivery.
- **Data Layer:** Persistent storage for POIs and optional outputs, plus media storage for images and audio assets.

Subsystem summary table. The following table summarizes the main subsystems, their responsibilities, and their key interfaces.

Subsystem	Primary responsibilities	Key interfaces
Presentation (Web UI)	Collects user inputs (city, duration, preferences), renders interactive maps and itinerary views, displays POI details, and triggers audio playback.	HTTPS REST calls to backend API, map tiles (OSM/Leaflet) [4], media retrieval endpoints.
Backend API Gateway	Authenticates/validates requests (where applicable), applies basic rate limiting, routes requests to internal services, and returns structured responses to the UI.	Public REST API endpoints, internal service routing, logging/monitoring hooks.
Backend Services (Core)	Implements business logic: POI filtering/selection, itinerary construction, routing orchestration, and content aggregation for responses.	Internal service-to-service calls, OSRM HTTP API [5], data access layer.
Data Layer	Stores POI metadata and optional itinerary outputs; serves read-only static assets (images, pre-generated TTS audio, etc.). Supports migration path from JSON/SQLite to PostgreSQL/PostGIS.	Database interface, object/media storage access.

3.3 Hardware/software mapping

The deployment structure of GUIDE is shown in Figure 3. Client devices access the system via HTTPS. The backend runs as a stateless API service, interacts with an OSRM routing engine over an internal network interface, and accesses a database and media storage for POI metadata and static multimedia assets.

Deployment summary. The diagram in Figure.3 shows how GUIDE components are mapped onto physical nodes. It emphasizes the separation between runtime components and external APIs that are used for offline dataset preparation.

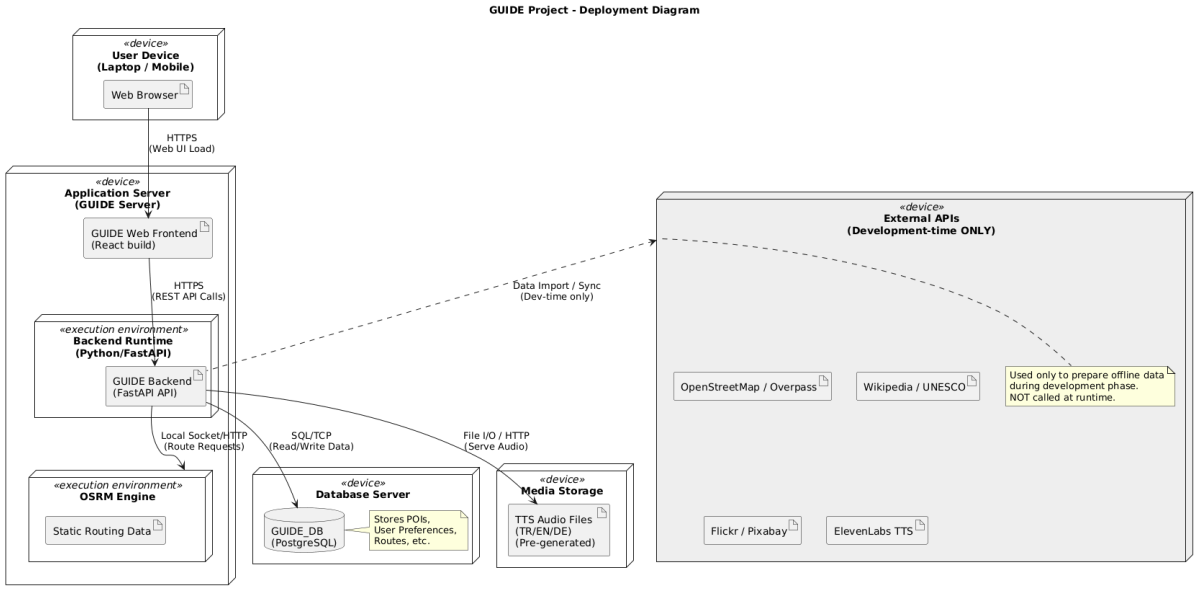

Figure 3: GUIDE deployment diagram.

Scalability and growth strategy. As user traffic increases, the GUIDE deployment can evolve from a single-node setup into a horizontally scalable multi-node topology. The backend is designed as a *stateless* API layer, enabling multiple identical application instances to run behind a load balancer. Static assets (images and pre-generated TTS audio) can be served from shared media storage or a CDN-like layer to reduce backend load. The routing tier (OSRM) is deployed as a separate service and can be scaled by running multiple OSRM instances behind an internal load balancer, isolating routing computation from the API tier. Importantly, external content APIs are used only during dataset preparation, so runtime scalability does not introduce third-party coupling; capacity planning remains focused on the application, routing, and storage layers that are fully under project control.

3.4 Persistent data management

Persistent data is managed via a centralized data store and local repositories. Data access is abstracted through service layers to ensure consistency and maintainability. Static multimedia (images and TTS audio) is stored separately and served as read-only assets during runtime.

Baseline vs growth path. For the baseline system, POI data can be maintained as a local, read-only repository, such as a JSON-based dataset, to simplify deployment and avoid operational complexity. As the dataset grows or concurrency increases, the architecture supports migrating to a lightweight embedded database and finally to a full database system without changing the overall subsystem boundaries.

3.5 Access control and security

Security is enforced through controlled access to protected resources, secure communication channels (HTTPS/TLS), and isolation of sensitive components. The API gateway layer may also apply basic validation, rate limiting, and request throttling to protect backend services from abuse or sudden traffic increases.

3.6 Global software control

System control is distributed across services. Logging, configuration, and monitoring are treated as global concerns to avoid single points of failure and to support operational visibility.

3.7 Boundary conditions

The system accounts for varying user loads, partial subsystem failures, network latency, and external service unavailability. A key architectural boundary is the *runtime vs development-time* separation: third-party content APIs are used only during offline dataset preparation, while runtime operation relies on self-hosted components. This reduces dependency risks and keeps runtime behavior deterministic and controllable under different operational conditions.

4 Subsystem services

This section summarizes the major services provided by each subsystem at a high level. Service responsibilities are intentionally defined through clear boundaries to support maintainability, testability, and scalability.

4.1 Presentation services (Web UI)

The Presentation Subsystem provides user-facing services that enable interaction with GUIDE:

- **Preference input and validation (client-side):** Collects trip parameters such as city, duration, categories, and optional filters.
- **Map visualization:** Renders POIs and routes on an interactive map (e.g., Leaflet with OSM tiles) and supports zoom/pan/selection.
- **Itinerary visualization:** Displays ordered POIs, day-by-day splits, and route summaries returned by the backend.
- **Media playback and POI details:** Shows descriptions and triggers the playback of pre-generated audio guides when available.
- **System explanation and flow guidance ("How It Works" view):** Presents a high-level, step-by-step overview of the system workflow, explaining how user inputs are transformed into an itinerary and routing outputs to improve transparency and user understanding.
- **Itinerary replanning and adjustment:** Enables users to revise previously generated itineraries by modifying preferences or constraints and triggers partial or full regeneration requests to the backend.

4.2 Application services

The Backend Services Layer provides the following core application services:

- **POI Filtering and Selection Service:** Selects candidate points of interest (POIs) by filtering on city and category. Following this filtering stage, an AI-based decision support mechanism is employed to evaluate, prioritize, and rank feasible POIs by considering user preferences, enabling the generation of realistic and time-aware itineraries.
- **Itinerary Generation Service:** Constructs an itinerary plan from selected POIs, including ordering POIs and possible daily grouping.
- **Routing Orchestration Service:** Interacts with the routing engine to compute paths, distances, and travel times required by itinerary construction.
- **Content Delivery Service:** Aggregates POI metadata and media references such as images/audios and returns a response that fits the format of the UI.

4.3 Data services (POI & media storage)

The Data Layer provides data services as read-heavy operations:

- **POI Repository Service:** Read-only access to POI metadata and categories.
- **Media Asset Service:** Serves static multimedia assets efficiently without requiring external calls at runtime.

4.4 Security services (API protection and access control)

Security is enforced through a combination of transport security and API protection mechanisms:

- **Secure communication:** HTTPS/TLS between client and backend.
- **Request validation:** Input validation and schema checks at the API boundary.

4.5 Integration services (routing and offline preparation)

Integration services define how GUIDE interacts with internal and external systems:

- **OSRM integration service (runtime):** Communicates with self-hosted OSRM [5] through HTTP APIs and normalizes route responses for internal use.
- **Offline data preparation pipeline:** Collects POI descriptions, images, and TTS audio from third-party providers [6, 7], and produces the static dataset used during runtime.

Service boundary note. A key design principle is that third-party content providers are excluded from runtime dependencies. Runtime service quality and availability are therefore primarily determined by self-hosted components (backend services, local data store, and OSRM), which improve reliability and support deterministic performance under load.

5 Glossary

- **API (Application Programming Interface):** A set of protocols and tools for building software applications, enabling communication between different software components.
- **CDN (Content Delivery Network):** A distributed system of servers used to deliver static content such as images, audio, or other media to users efficiently. In the context of GUIDE, a CDN-like mechanism refers to serving static assets from shared or replicated storage to reduce backend load and improve response times.
- **Context-Aware System:** A system that uses contextual information (location, time, preferences) to provide relevant services and recommendations.
- **GDPR (General Data Protection Regulation):** European Union regulation (EU 2016/679) governing data protection and privacy for individuals within the EU.
- **KVKK (Kişisel Verilerin Korunması Kanunu):** Turkish Personal Data Protection Law (Law No. 6698) regulating the processing and protection of personal data.
- **OSM (OpenStreetMap):** A collaborative project that creates a free, editable map of the world, used for geographical data and map visualization.
- **OSRM (Open Source Routing Machine):** A high-performance routing engine designed to find the shortest path in road networks, used for route calculation and navigation.
- **POI (Point of Interest):** A specific location that may be of interest to travelers, such as historical sites, museums, restaurants, or natural landmarks.
- **RESTful API:** An application programming interface that follows the REST architectural principles for web services communication.
- **Route Optimization:** The process of determining the most efficient route between multiple points, considering various constraints like distance, time, and preferences.
- **TTS (Text-to-Speech):** Technology that converts written text into spoken audio output.
- **UI (User Interface):** The visual elements and interactive components through which users interact with the system.
- **UX (User Experience):** The overall experience and satisfaction a user has when interacting with a product or system.

References

[1] IEEE Computer Society, *IEEE Std 1016-2009: Systems and Software Engineering—Software Design Descriptions*, 2009.

[2] ISO/IEC, *ISO/IEC 25010:2011 Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models*, 2011.

[3] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148:2018 Systems and software engineering—Life cycle processes—Requirements engineering*, 2018.

[4] OpenStreetMap Foundation, "API v0.6 Documentation," [Online]. Available: `https://wiki.openstreetmap.org/wiki/API_v0.6`. [Accessed: Jan. 2026].

[5] Project OSRM, "Open Source Routing Machine HTTP API Documentation," Version 5.24, [Online]. Available: `https://project-osrm.org/docs/v5.24.0/api/`. [Accessed: Jan. 2026].

[6] Wikimedia Foundation, "MediaWiki Action API Documentation," [Online]. Available: `https://www.mediawiki.org/wiki/API:Main_page`. [Accessed: Jan. 2026].

[7] ElevenLabs, "ElevenLabs Text to Speech API Reference," [Online]. Available: `https://elevenlabs.io/docs/api-reference`. [Accessed: Jan. 2026].