Trinity WayFinders

WayFinder Application

Functional Architecture Document

Revision History

Version	Created Date	Author	Comments
1.0	11/02/19	Nicholas Bonello	

Authors List

Sl. No.	Name	Role
1.	Nicholas Bonello	Author
2.	Saad Tariq Malik	Author

Table of Contents

1. Objective

This document contains the details of the functional architecture as well as detailed descriptions of the component interfaces of the Trinity Wayfinder application.

2. Requirements Overview

The following sections will contain the functional architecture and detailed descriptions of all components; explaining their functions and any communications with other internal components.

3. High-Level Architecture

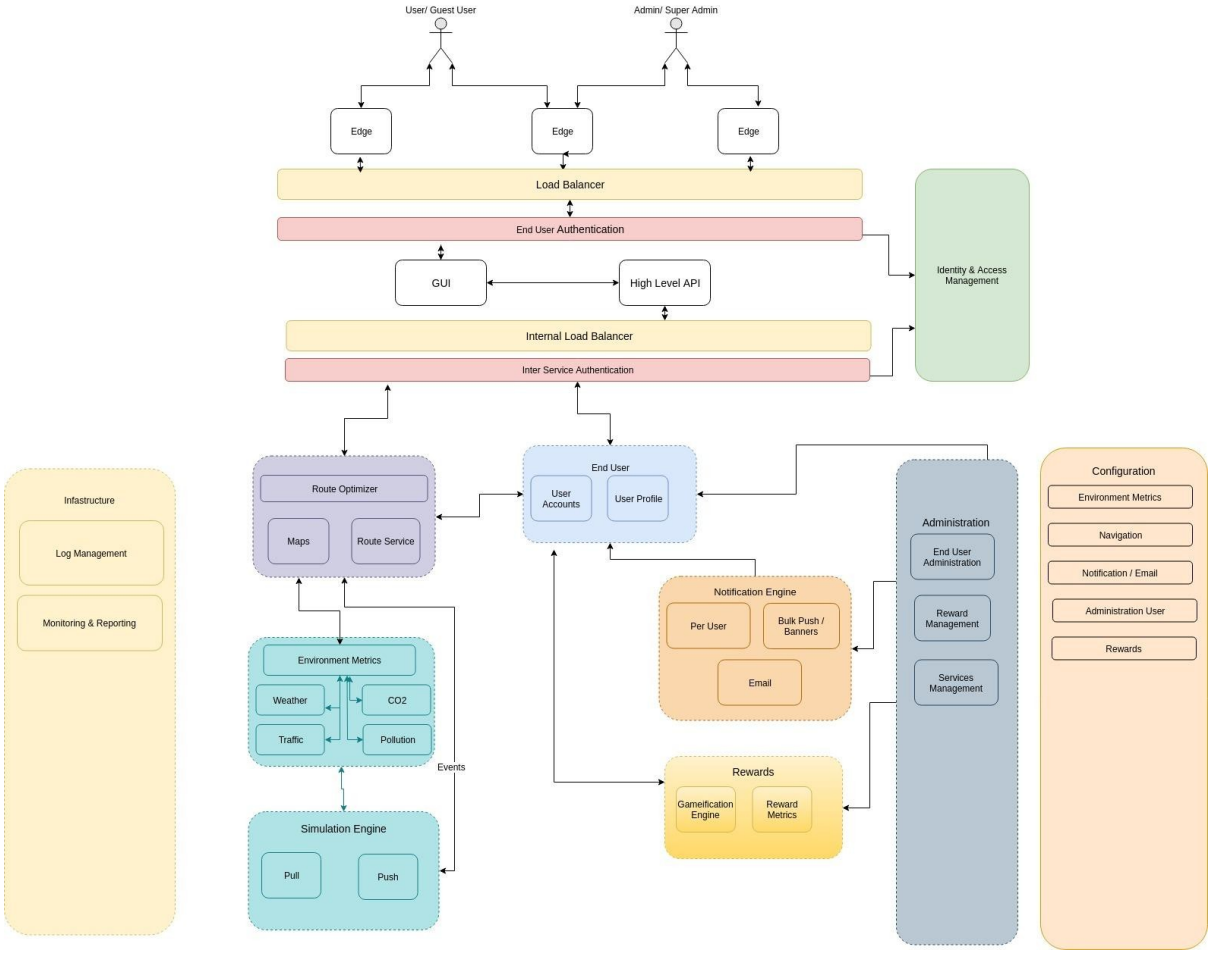

Figure 1 – High-Level Architecture

4. High-Level Component Descriptions & Interactions

4.1. GUI

The graphical user interface allows end-users to interact with the TrinityWayFinders application. All data will go through the High-Level API which will, in turn, call all the relevant services and methods securely. The GUI will only communicate with the Edge Nodes and High-Level API through REST APIs.

4.2. High Level API

The High-Level API acts as the intermediary between the user-visible GUI and all the lower level microservices. Edge nodes would skip the GUI and directly contact this API through an authentication layer, which will, in turn, contact all the necessary APIs through their load balanced URLs.

API would be in communication with all services via REST APIs.

4.3. Environmental Metrics Service

General Overview

This service will be responsible for handling all environmental data that could affect the applications routing decisions. Each factor (eg. weather, traffic, pollution levels) will have their own sub-service responsible for requesting, handling and transforming the external data to an acceptable format.

Configuration

Each factor's module will need to have its own configuration settings depending on the external APIs requirements, including but not limited to API keys, URLs, tokens, how often the cache should be cleared etc.

In the event of an external API no longer returning data, the application is expected to seamlessly switch over to either another API or simulated data. The configuration management service will handle this transition through the key-value store. Through the key-value store, the Environmental metrics service will always use the secondary API until the primary API re-sends a heartbeat.

Communicates with:

1. **Navigation Service**

When calculating the most optimal route, the route optimizer will need to consider the environmental factors before making a decision. Communication will be done via REST APIs that would temporarily cache the data depending on the time limits enforced by the configuration

management service.

2. Simulation Service

When no environmental data is obtainable through real APIs, the application will switch over to using simulated data through the simulation service. These will communicate over REST requests.

4.4. Simulation Engine

See Figure 2 below for detailed Simulation Engine Architecture

General Overview

When live data is no longer available, the simulation engine is responsible for mocking any data including but not limited to any environmental features and route optimization changes such as closed roads, traffic etc. In scenarios where a real API exists but temporarily fails and no backup API is available, the simulation engine should seamlessly provide mocked data instead.

Configuration

Each environmental and route optimization metric that may affect the final routing will have its own set of APIs and configurations stored in the configuration management service. Should all of the APIs fail, then the configuration management service will update the Key-Value store values for the service in question to the simulation engine at which point simulated data would start being used.

The simulated data that is returned for each metric can also be altered through this service.

Communicates with:

1. Environmental Metrics Service

When no API data is available, simulated environmental data is to be used instead. Through REST APIs the simulated environmental metrics are sent to the service. Calls could be cached for a short duration of time to reduce the load.

2. Navigation Service

Similarly, the navigation service will require simulated data when traffic or road block events are no longer being received by the real API. In this case the navigation service should request mocked data from the simulation engine via REST APIs. Calls could be cached for a short duration of time to reduce load.

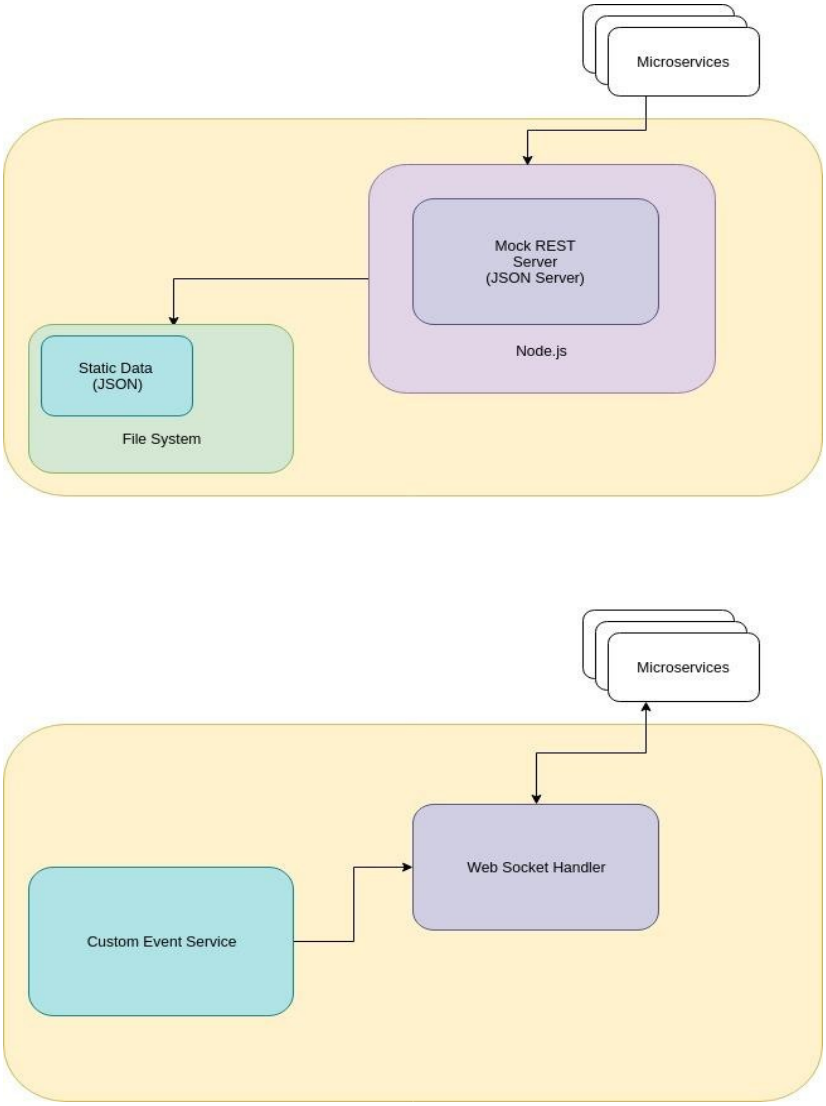

Figure 2 - Simulation Service Architecture

4.5. End-User Management Service

General Overview

This service is responsible for managing the end-user's profile and data. It would consist of the following two sub-services:

- User Access
  - Responsible for providing the end-user access to his profile and data
- User Profile
  - Responsible for management of an end-user's profile (e.g adding/removing information, setting preferences and other options).

Configuration

N/A

Communicates With:

1. Rewards
The End-User Management service would need to communicate with the Rewards service to redeem awards etc.

2. Infrastructure
The service would be in communication with the Infrastructure service for logging etc.

3. Notification
The service would be in communication with the notification service in to receive notifications from the network.

4.6. Navigation Service

General Overview
This service would be responsible for recommending routes to the user in real-time. This service would consist of 3 sub-services:

- Route Service
  - This service is responsible for suggesting possible routes and the possible modes of transport between the departure point and the destination.
- Maps
  - Given a route, the Maps service will give you a section of a geographical map that will contain the route.
- Route Optimizer
  - This service is responsible for analyzing the different routes suggested by the Route Service, selecting an appropriate route, segmenting it based on user preferences and eventually devising a final route that corresponds to the user's goals and preferences.

Configuration
N/A

Communicates with:

1. End-user Management Service
In order to optimize a route for the user's preferences, the service needs to communicate with End-user Management Service to obtain user's profile and preferences.

2. Infrastructure
The service would be in communication with the Infrastructure for logging purposes

3. Environmental Metrics Service

The service would be in communication with the Environment Service to obtain environmental information which would be utilized in devising a route for the end-user

4. Configuration Management Service

The end-user service would be in communication with the configuration management service for easy configuration of any of its parameters.

4.7. Administration Service

General Overview

This service provides the backend for the Admin user. It will consist of the following 3 sub-services which represent the admin role:

1. Services Management
    - Responsible for configuring/modifying the parameters of the rest of the services in the system.
2. End-user Management
    - Querying specific information about a user
    - Block a user from the network
    - Etc.
3. Rewards management
    - Introducing new rewards in the system
    - Deactivating rewards
    - Awarding rewards to users

Communicates with:

1. Configuration Management Service

The service will communicate with all the other services via the configuration management service.

4.8. Rewards Management Service

General Overview

The Rewards Management Service handles all the gamification aspects and user rewards, including badges and levels.

Configuration

Administrators have the ability to add new rewards, modify and delete existing rewards through the configuration management service.

Communicates with:

1. End-User Service

When users receive new badges or level up their profiles should be updated to reflect this change. Messages should be sent via a Message-Queue system invoking an update profile method when received.

2. Notification Engine

When users receive new badges or level up they should be notified through the notification engine. Messages should be sent via a Message-Queue system that handles the message whenever possible.

4.9. Notification Service

General Overview

1. This service would be responsible for dispatching notifications to the users.
2. It would handle both ‘push’ notifications (initiated by the user) and ‘pull’ notifications (not initiated by a user).
3. The notifications would be in the form of emails and in-app pop-up messages.
4. The service would allow for showing certain notification to a selective subset of users and sending notifications in bulk

Configuration

The service should allow for configurations:

1. The frequency of notifications
2. When a notifications is triggered
3. The users who should receive the notification (one user, a subset of users, all users etc)
4. Record of notifications (checking whether a particular notification was delivered to a user)

Communicates with:

1. Environmental Metrics

The service would be in communication with the Env Metrics service to publish environment information about the routes.

2. Rewards

The service would need to be in communication with the Rewards service to publish information about new promotions and rewards received etc.

3. End User

The End-User service would need to be in communication with the Notifications service in order to notify the user of any notifications, e.g notifications about account inactivity.

4. Infrastructure Service

The notifications service will need to be in communication with the Infrastructure service to record information about pushed notifications etc.

4.10. Configuration Management Service

General Overview
The Configuration Management Service is responsible for all admin-configurable options allowed on each system. This service allows the admin to make basic changes without needing developer assistance.

Communicates with:

- Environmental Metrics Service
  - Contains all the external environmental API specific configuration settings
- Navigation Service
  - Contains all the external navigation API specific configuration settings
- Rewards Management Service
  - Add reward
  - Modify reward
  - Delete reward
- Notification Engine
  - The frequency of notifications
  - When a notification is triggered
  - The users who should receive the notification (one user, a subset of users, all users etc)
  - Record of notifications (checking whether a particular notification was delivered to a user)
- Simulation Engine
  - Add new mocked data
  - Edit mocked data
  - Delete mocked data
  - Define rules when simulated data should be used

4.11. Infrastructure

General Overview
The infrastructure module is responsible for Monitoring & Reporting the health status all of the servers, load-balancers, modules, and services. It is also this modules responsibility to ensure that each service is up and running as expected. This would be done by ensuring that each service has a REST API with at minimum a *'/health'* route, returning nothing but a 200 OK response to ensure that the service is up. Load balancers will then handle the load according to how many services are up. Each service is expected to have standardized logging that will then be aggregated by this module, storing all logs into a single common and indexable directory.

This service will communicate with every other service and load balancer in the system via REST API calls.

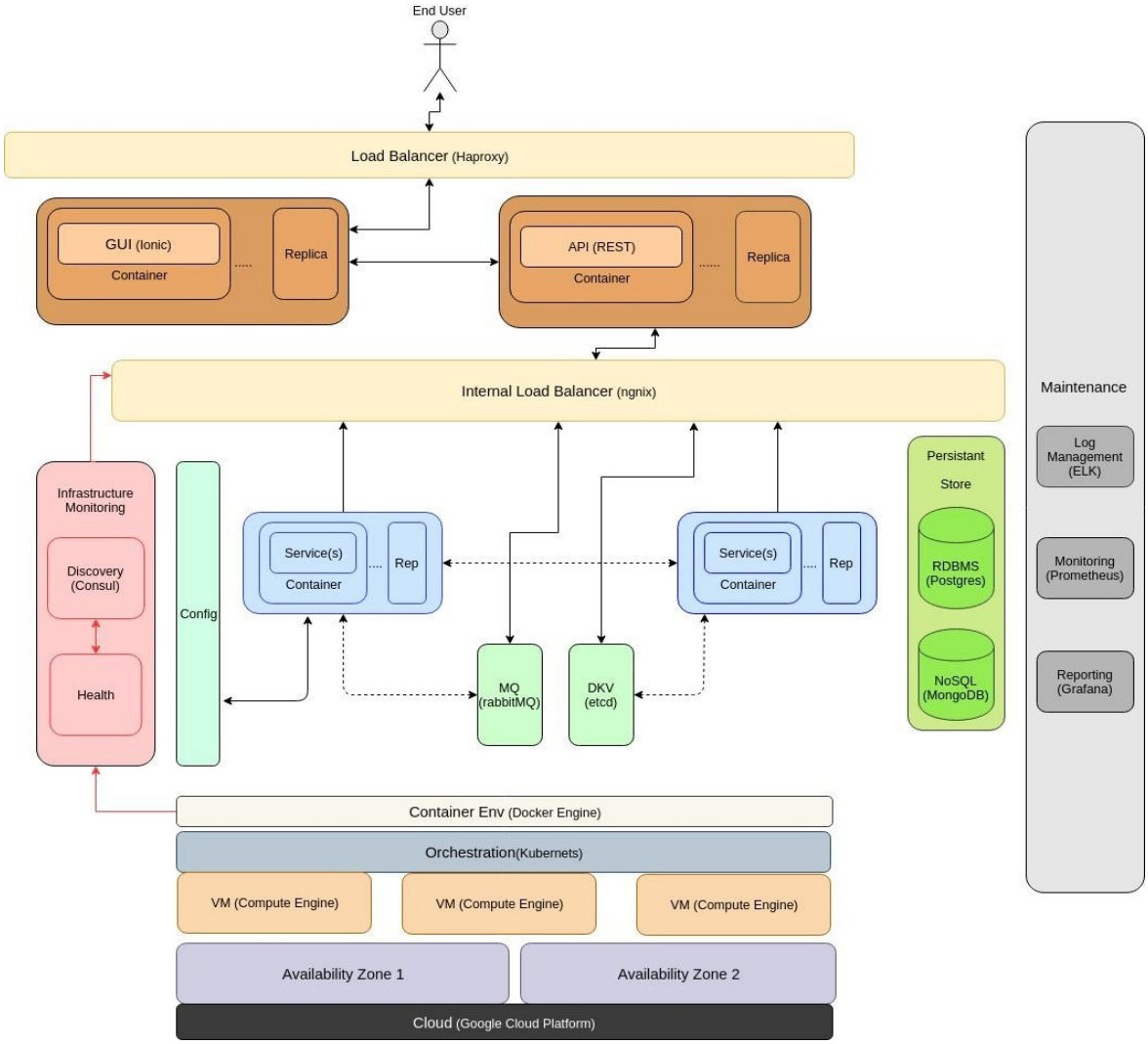

Figure 3 - Infrastructure Details

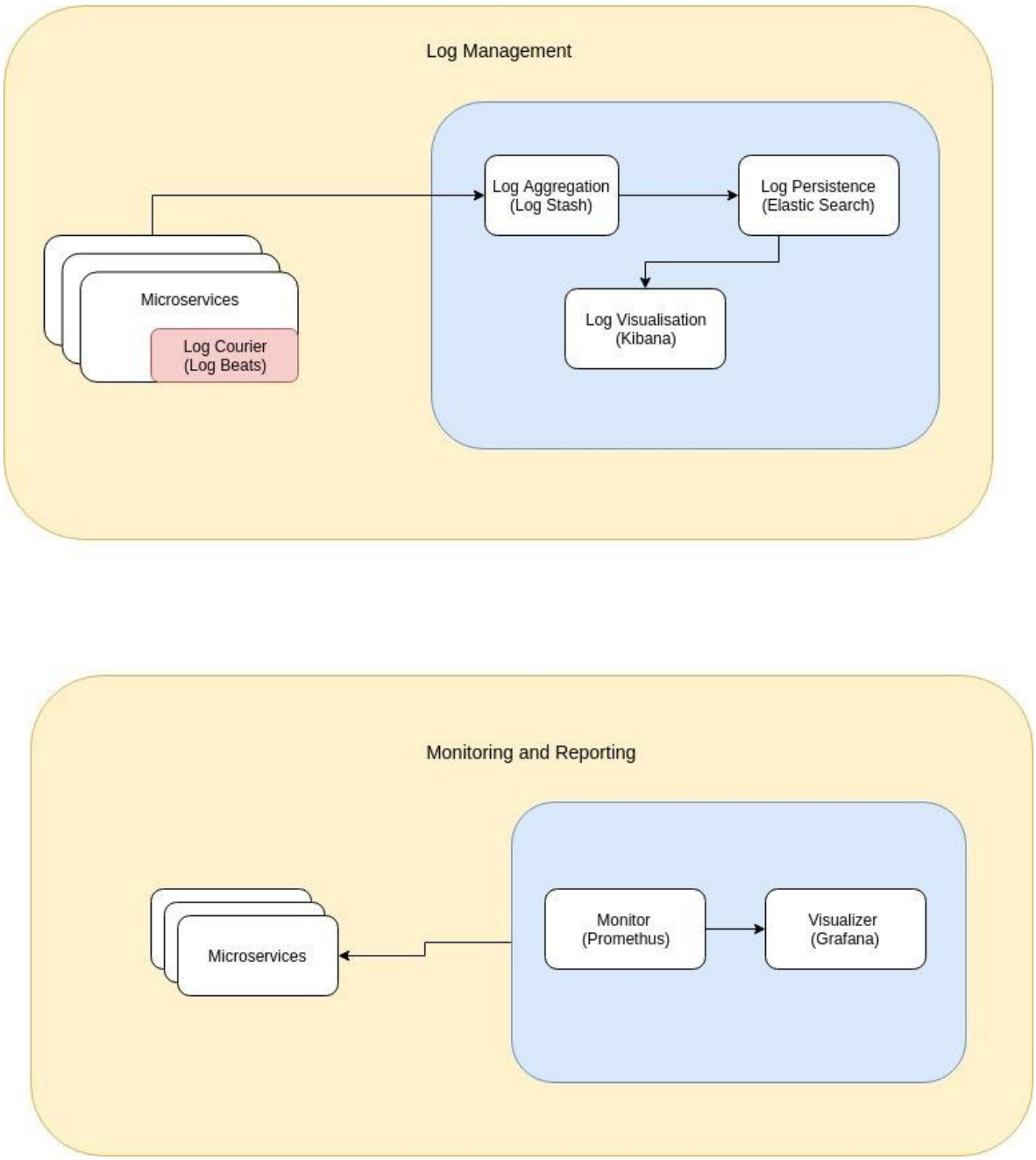

Figure 4 -Log Management, Monitoring and Reporting Infrastructure Architecture Diagram

4.12. IAC

General Overview

The Identity and Access Management (IAM) service facilitates the management of electronic or digital identities. This module will include organizational policies for managing digital identities as well as the technologies needed to support identity management.

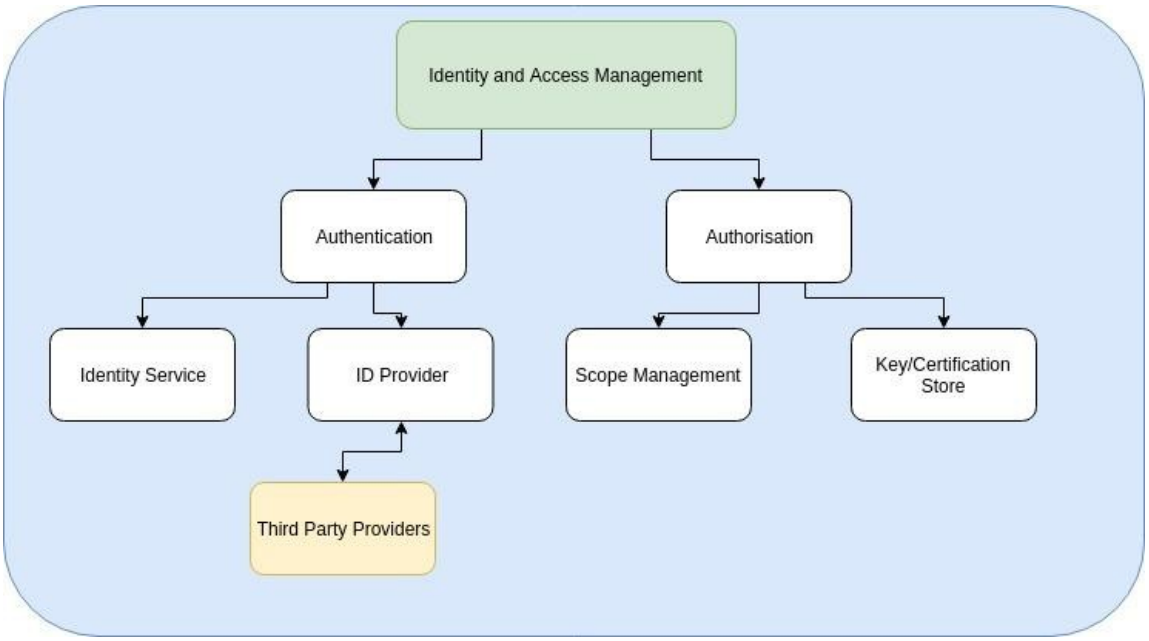

Figure 5 - IAC Architecture

5. Decentralized Services

5.1. Edge Architecture

The below diagram showcases the architecture of the edge node of the Trinity Wayfinder application. The edge nodes facilitate the decentralized working of the Trinity Wayfinder application along with the business logic on the constrained devices.

The persistent connections interface on the edge node ensures connection with the cloud while the Edge core logic handles the orchestration of connections with the client connections serving content in a information-centric manner. The edge nodes long term persistent data in the CDN and the dynamic data in the cache enabling the clients (resource constrained) to use data even during the absence of an internet connection.

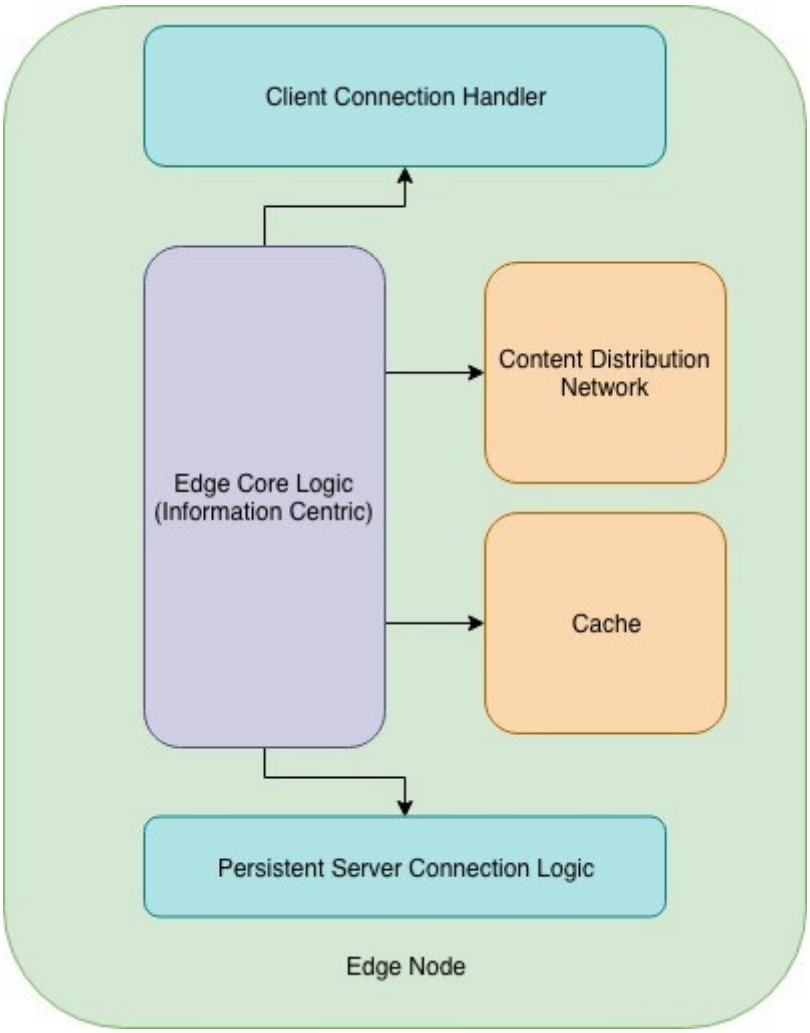

Figure 6 - Edge Architecture

5.2. Constrained Device Architecture

The constrained devices are also designed with a decentralized architecture, enabling them to function even when there are intermitted connection issues with the internet. The abstract connection logic in the constrained devices including Tablets and Mobile phone provides a simple interface to connect to the closest edge when available else to the cloud in a uniform manner.

The business logic takes care of the GUI and the API service requests whereas the ‘Connection Interruption Handler’ handles the logic for enabling the application to be fault tolerant and enable functioning of the application even when there are intermittent connection issues to the edge node or cloud. The intermitted connection handler performance active caching, detects failed connection and provides the device with the cached data as and when required.

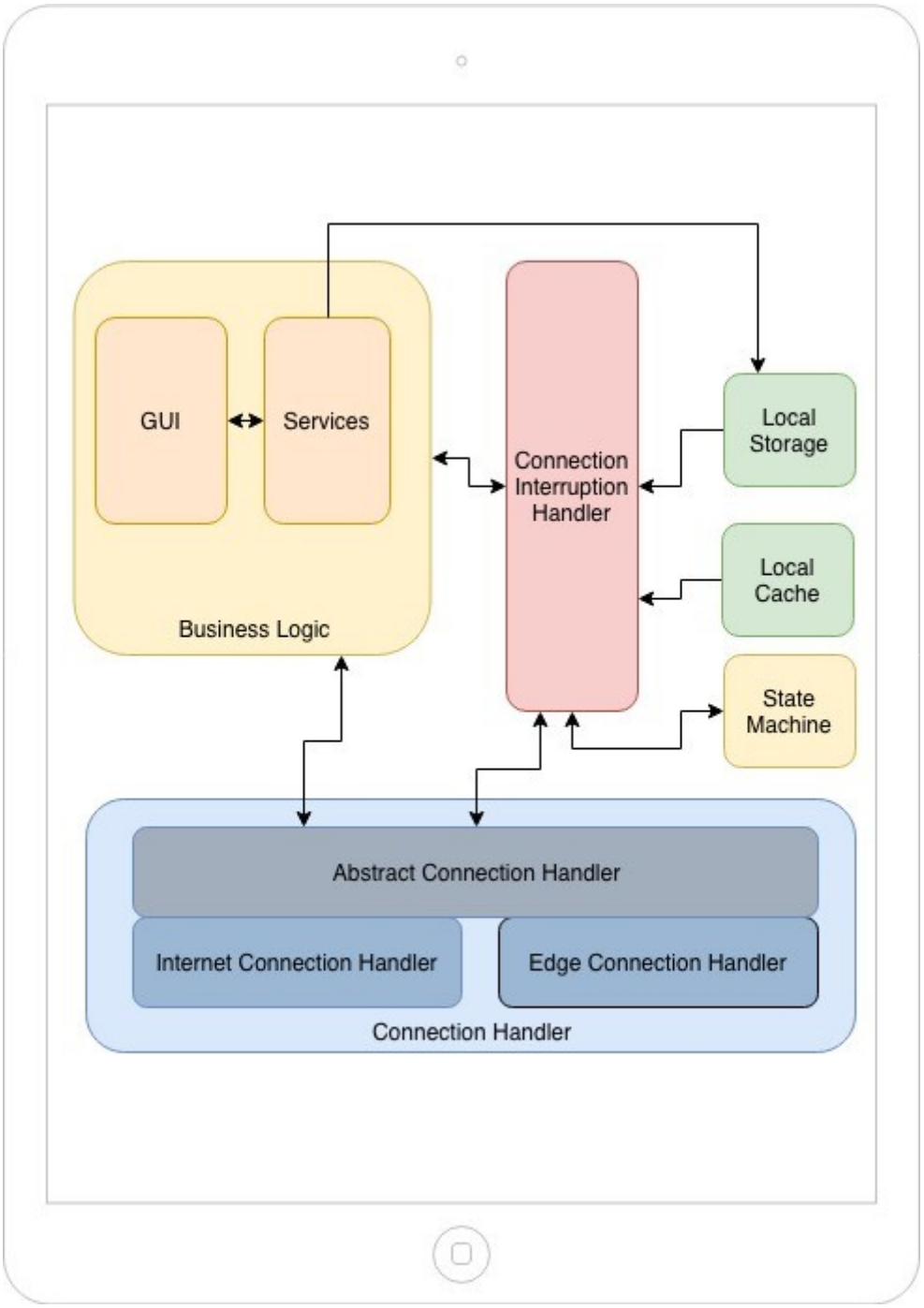

Figure 7- Constrained Devices Architecture

6. Component Communications Summary

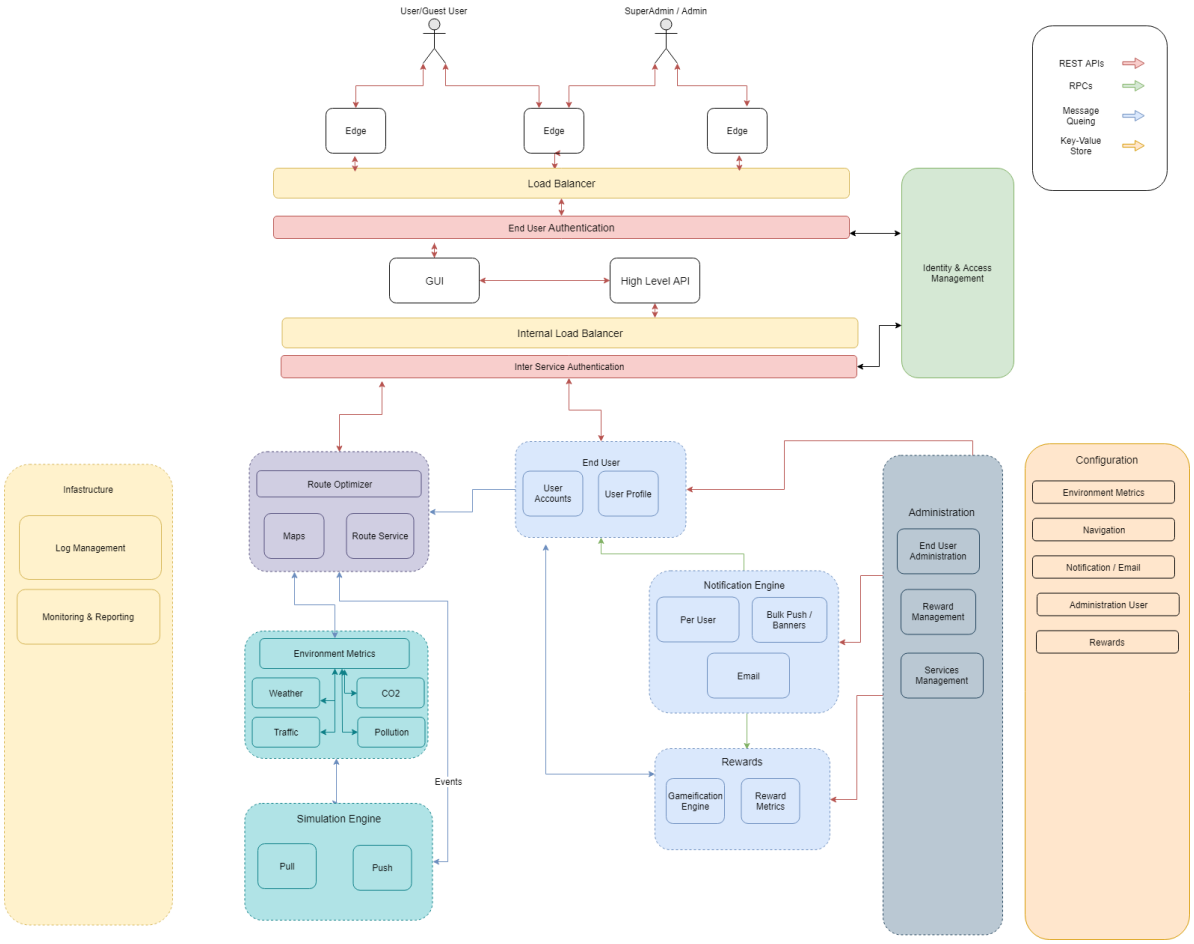

Figure 8 - Component Interfaces

Service Name	Communicates with	Communication Protocol
High-level API	● Env. Metrics Service ● End-User Service ● Navigation Service ● Rewards Service ● Administration Service ● Configuration Service	REST APIs
Environmental Metrics Service	● Simulation Service	REST APIs
	● Configuration Management Service	Distributed Key-Value store
End user Management Service	● Notification	Message Queue
	● Infrastructure ● Rewards service	REST APIs

Navigation Service	● Env. Metrics Service ● Simulation Service	Message Queue
	● Configuration Management Service	Distributed Key-Value store
Rewards Service	● End-User Service ● Notification Service	Message Queue
	● Configuration Management Service	Distributed Key-Value store
Notification Service	● Rewards Service ● End User Service ● Administration Service ● Logging Service	RPC
Administration Service	● End User Service ● Notification Engine ● Rewards Service	REST
Configuration Management Service	● Environmental Metrics Service ● Navigation Service ● Rewards Management Service ● Notification Engine ● Simulation Engine	REST

7. Assumptions, Constraints and Dependencies

Assumptions:

- No component communications with external APIs are described.
- All component to component REST calls goes through a load balancer.