Summit Finder

Sebastiano Cassol
sebastiano.cassol@studenti.unitn.it

github.com/cassolseba/summit-finder
summitfinder.docs.apiary.io

21 February 2024

1 Introduction

This project is to devise, design and develop a service-oriented application that applies technologies and techniques learned during the Service Design and Engineering lectures and laboratories. In particular, is required to develop a project that consists of multiple services. The idea is an application where the user can search for mountain peaks and alpine refuges to reach and save the result in a personal wishlist. Since the wishlist is private, the user is required to create an account and authenticate to access the application.

2 Design

Figure 1 illustrates the architecture of the application. It is composed by four different layers, each one with a specific purpose. The presentation layer provides access to the application via a user interface.

2.1 Process Centric Layer

The *process centric layer* consists of three services: the *Authentication service*, the *Management service* and the *Tour service*. This layer serves all requests coming directly from the user and act as a gateway to all other services in the application context, thus coordinating and orchestrating the work between the underlying services.

Authentication service The *authentication service* is primarily responsible for user authentication and user registration, but other user related functionalities are provided. It communicate directly with the *user service* through REST APIs, forwarding requests coming from the user.

Management service The *management service* is responsible for managing user wishlists and provides several operations such as adding, retrieving and deleting wishes and wishlists. It implements also authorization, enabling the users to retrieve only their wishlists and the admin to retrieve and delete all the wishlists. It communicate directly with the *wishlist service* through REST APIs, forwarding user requests.

Tour service The *tour service* is in charge to compose a unique JSON object, aggregating data arriving from different services. In particular it send requests to four services: *Forecast service*, *Nominatim adapter*, *Overpass adapter* and *Elevation adapter*. All these services abstract their usage through REST APIs.

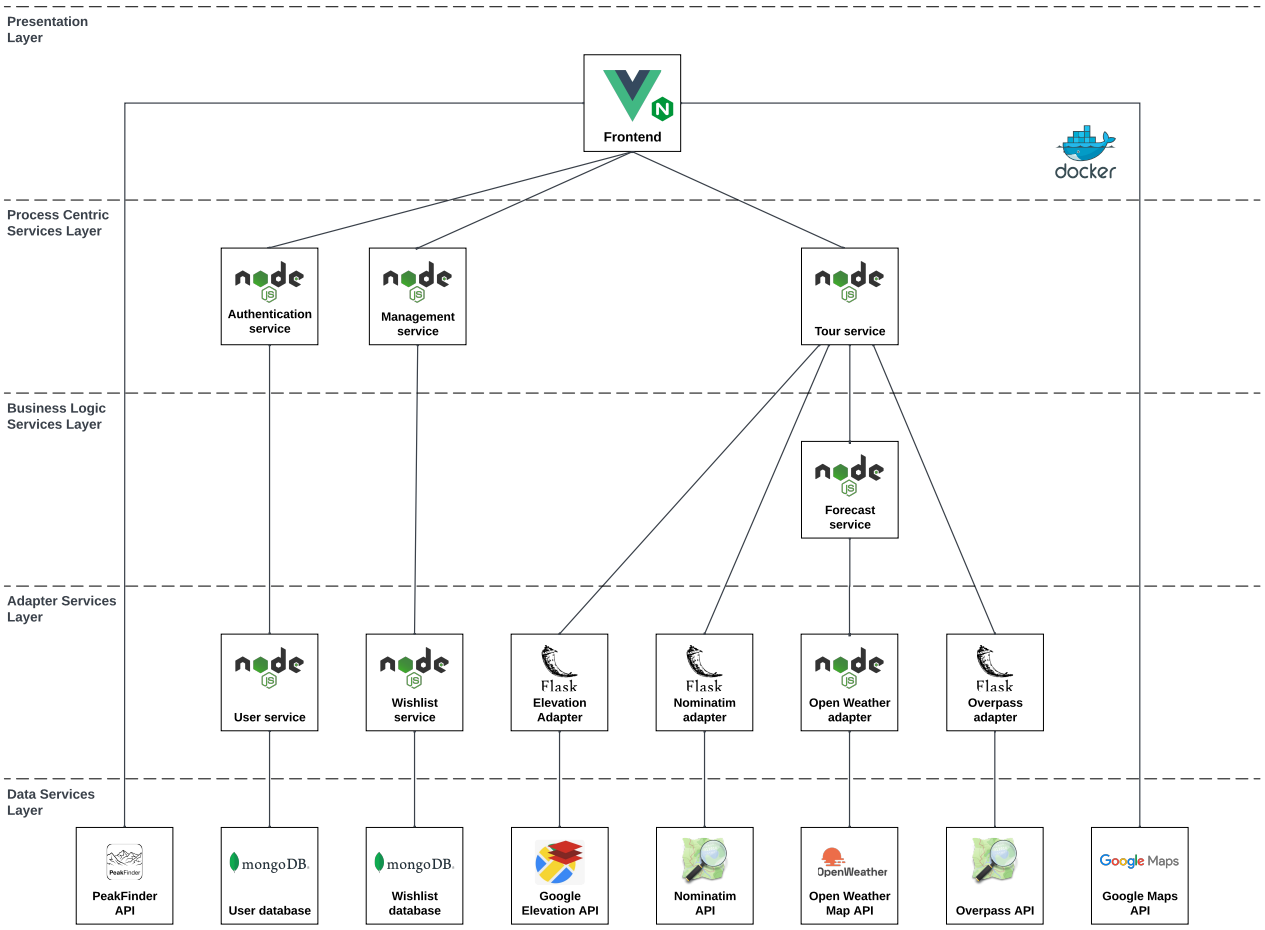

Figure 1: Architecture design

2.2 Business Logic Layer

The *business logic layer* is composed only by one service: the *Forecast service*. This layer server all requests coming from the process centric layer. It is responsible for retrieving data from the underlying layers and then perform logic manipulations, calculations and decisions.

Forecast service The *Forecast service* retrieves weather information from the *OpenWeather adapter* and, once data are received, performs logic calculations to compute several parameters such as average, maximum and minimum temperature, average humidity, average visibility, probability of rain/snow and so on.

2.3 Adapter Service Layer

The *adapter service layer* consists of six services: *User service*, *Wishlist service*, *Nominatim adapter*, *Overpass adapter*, *OpenWeather adapter* and *Elevation Adapter*. This layer extracts and fetches data from external data sources or services, such as databases and external APIs, and provides interfaces to abstracts its usage.

User service and Management service The *user service* and the *management service* have direct access to database and performs CRUD operations. Every service has access to its own MongoDB database.

Nominatim, Overpass, OpenWeather and Elevation adapters These adapters have access to data from external services that accept requests through REST APIs, except for Overpass APIs that works on Overpass QL[1].

[1]Overpass Query Language - wiki.openstreetmap.org

2.4 Data Layer

The *Data layer* is responsible for handling all data related requests, sitting on top of internal databases and external sources. In this project, several sources are involved, both internal and external. As internal databases, two MongoDB databases serve as persistent storage. As external sources, the following services are involved: OpenWeatherMap API (limited to 60 requests/hour), Google elevation API (limited to 1000 requests/month), Overpass API and Nominatim API (both based on OpenStreetMaps). Finally, other two external services are involved, Google Maps API and PeakFinder API, that are integrated directly within the frontend application.

3 Implementation

As mentioned before, in this project different techniques and technologies are involved. This section aims to briefly explain implementation choices and how the repository is structured.

3.1 Development

Most services are developed using *Node.js*[2] in combination with widely adopted packages, such as *express* to handle incoming requests and *axios* to perform requests to other services. Some services instead are developed using the python framework *Flask*[3]. User interface is developed using *Vue*[4] framework and *Vuetify*[5] library, and hosted on a *NGINX*[6] server. To provide consistency and isolation between processes, every service runs in its own docker container using *Docker Compose*, that provides a single-command setup. Docker compose relies on a single environment file[7] and dispatch environment variables to the various containers. Each container talks to each other through the docker default network. All services provides REST APIs.

3.2 Authentication and Authorization

Authentication and authorization are implemented using *JSON Web Token (JWT)*. Users and admins can sign in by sending a POST request[8] to the authentication service, that will reply with the token. Not all services require authentication: only user service and management service implement token validation. Once the token is decoded, an additional validation is performed on the payload to evaluate whether a user is an admin or not. Token refresh mechanisms are not implemented.

3.3 Repository structure

The project is hosted on GitHub at github.com/cassolseba/summit-finder. The repository contains all the source files of each service, as well as the documentation. Services are grouped in folders by layer.

3.4 Problems

To simplify the development of the project, some adapters and services such as user service and wishlist service can be accessed without providing an access token. Thus, sending a request directly to this services result in a security flaw. To address this problem, a CORS policy is applied to this services, permitting request only from the host and the specified docker container. A further improvement could be the extension of token validation and token forwarding to all the services.

[2] Node.js - nodejs.org
[3] Flask - flask.palletsprojects.com
[4] Vue - vuejs.org
[5] Vuetify - vuetifyjs.com
[6] NGINX - nginx.com
[7] An .env file is required to run the project.
[8] User is required to specify the username.