Pizzicanto
201519 | 201504 | 201517 | 201536 | 201546

Pizza Roulette Architecture

1. Architecture analysis

Pizza Roulette will be a simple web-application mainly focusing on the GUI and the navigation of the randomly generated pizzeria, or chosen pizzeria.

For this purpose the most obvious architectural style needed is the **n-layered architecture**. The application's logical layers will be divided into separate physical layers, such as:

- The client layer - this layer consists of the web browser on which all clients will be able to access our application
- The web layer - representing the user interface which will be made adaptable to each client browser adequately
- The application layer - consisting of the business logic of the application and all case implementations
- The data layer - the layer from which all data shall be extracted, in our case it isn't a classical database, instead it is a simple.csv file which contains all the addresses and latitudes/longitudes of the pizzerias
- The external-web-service layer - which in our case is the external web-service for navigation and mapping OpenStreetMaps

Aside from the logical architecture of our application, it simultaneously encompasses a **distributed architecture**, as it uses data from different computational nodes to satisfy a single client. In our application, those nodes are our application software and the external-web-service software from which we use the navigation functionalities.

That same external-service also represents a **microservice**. Using separate services for achieving the same goal makes the job of our developers simpler, and our application richer.

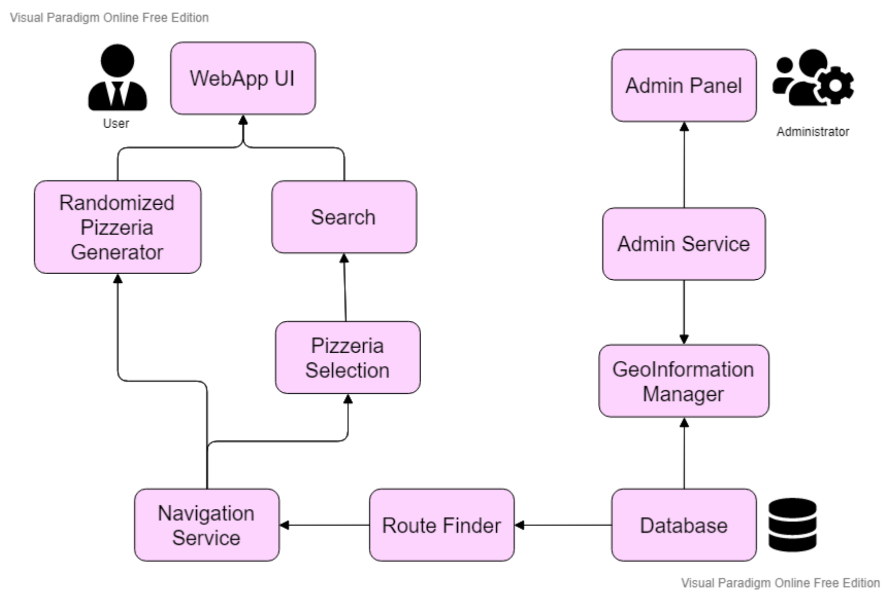
Having a microservice also makes our application **containerized**, meaning that our system and our external-web-service system run separately from each other, with isolated runtimes, but provide a client with a single application instead of a few separate ones. This also helps with the idea of a portable application that shall run seamlessly on every browser.

Lastly, having a big pool of data makes it difficult to run the application without problems, thus the need for a **pipe-and-filter architectural style**. It connects all the above-mentioned components and layers, where each component computes the data, also known as filtering, and passes it along to the next component via a pipe. In our case, the data is taken from the .csv file and passes along to the application layer, where the needed data is extracted and sent to the external-web-service, in this case OpenStreetMap. After the location is found, it is again sent back to the application layer, and transferred to the web layer to be displayed to the user on their interface.

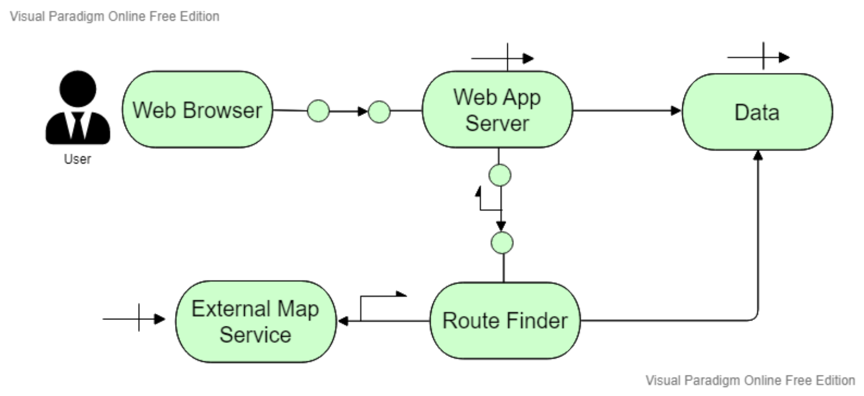
In conclusion, our web-application is with a ***hybrid architecture***, combining all the styles and designs mentioned above into one single application with the best possible performance which our developers are capable of executing.

2. Design views

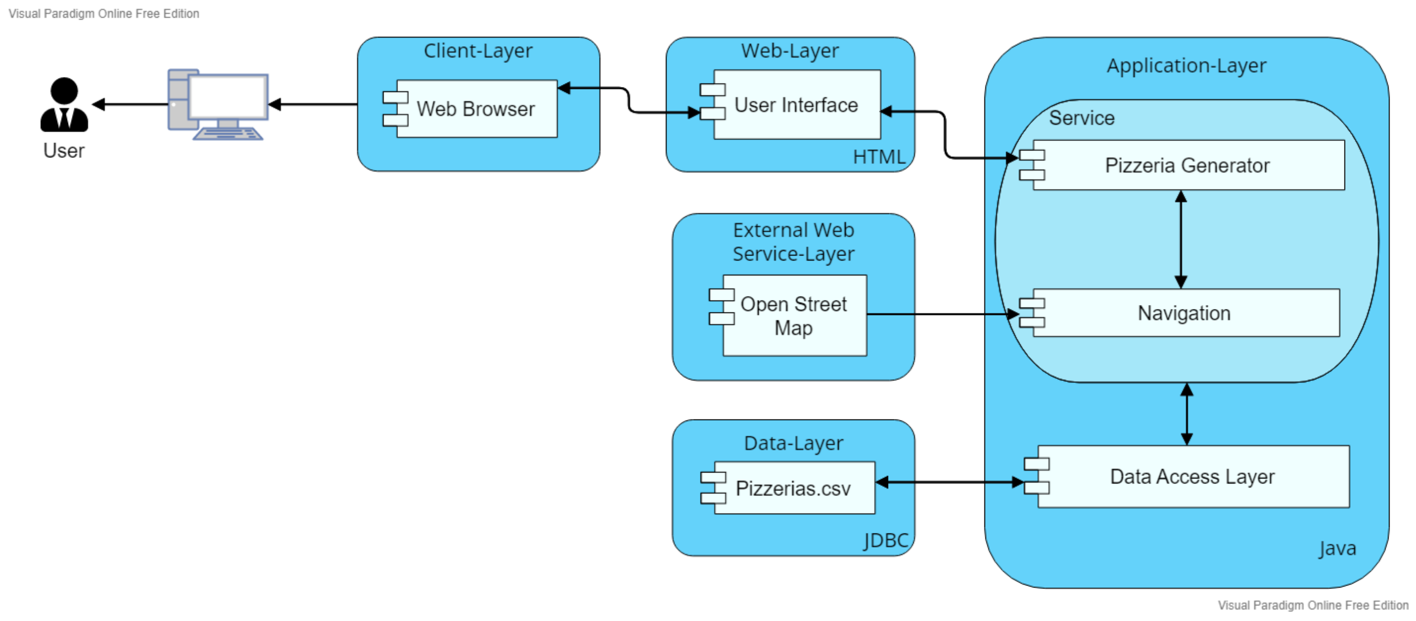
a. Conceptual view



b. Executional view


Visual Paradigm Online Free Edition
User
Web Browser
Web App Server
Data
External Map Service
Route Finder
Visual Paradigm Online Free Edition

c. Implementational view


Visual Paradigm Online Free Edition
User
Client-Layer
Web Browser
Web-Layer
User Interface
HTML
Application-Layer
Service
Pizzeria Generator
External Web Service-Layer
Open Street Map
Navigation
Data-Layer
Pizzerias.csv
JDBC
Data Access Layer
Java
Visual Paradigm Online Free Edition