ParTI

An ecosystem (database, server and user interface) to get information on bars and restaurants of Canton Ticino that are application's partners.

Bachelor Thesis

Sebastian Käslin

July 2024

Thesis supervisors:

Prof. Dr. Jacques Pasquier-Rocha
Software Engineering Group

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Software Engineering Group
Department of Informatics
University of Fribourg (Switzerland)

Software Engineering Group

Table of Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1. Motivation and Goals

This bachelor thesis presents the process of developing a software that allows users to get information on bars and restaurants of Canton Ticino. The thesis will treat in detail the main idea, its modelization, the final result and the technologies used for the implementation. The goal of this work is to create an application's ecosystem that provides to the user the expected functionalities and that it is developed in a manner that will facilitate future improvements.

1.2. Organization

Introduction

The introduction chapter treats the motivation and goals, the structure of the work by explaining the content of each chapter and in the end an overview of the formatting conventions.

Chapter 2: The project ParTI

This chapter tells the state of the art of applications that offer the same aimed functionalities. It first of all discusses and shows some intersting Graphical User Interface (GUI) aspects of one of these applications and finally it describes the application protagonist of this work, ParTI, from the point of view of the use cases and the database structure that supports the ecosystem.

Chapter 3: User point of view

This chapter contains the discussion on the final result of the application by showing screenshots of the GUI and then by describing in more detail two distinct scenarios that apply some of the defined use cases.

Chapter 4: Programmer point of view

It describes the technology stack used for the implementation by dividing the ecosystem into three subparts: database, backend and frontend. The technology are explained with the use of screenshots and code snippets.

Conclusion

It discusses the goals achieved and sets some directions to improve the application.

Appendix

It contains a list of common abbreviations, information about the github repository and the references.

1.3. Notations and Conventions

- Formatting conventions:
  - Abbreviations and acronyms as follows Entity-Relationship Diagram (ERD) for the first usage and ERD for any further usage;
  - The notation used for the ERD is the Modified Chen Notation (MC-Notation) [5];
  - `http://localhost:3000/businesses` is used for web addresses;
  - Code is formatted as follows:

```
1 public double division(int _x, int _y) {
2   double result;
3   result = _x / _y;
4   return result;
5 }
```

- Figure s, Table s and Listings s are numbered inside a chapter. For example, a reference to Figure j of Chapter i will be noted *Figure i.j.*

2
The project ParTI

2.1. Introduction

This chapter starts by covering applications that are similar to the one protagonist of this work, similar in the sense that they offer to the user a comparable set of functionalities. In particular it will be shown in detail one of these applications, TheFork, by presenting the most interesting part of its GUI. In the second part of the chapter instead we are going to define the ParTI application's goals by discussing the clients' use cases. Consequently once stated the client's use cases, it will be discussed and drawn an according Entity-Relationship Diagram (ERD) for the ecosystem's database.

2.2. State of the art

On the market there are already multiple applications that offer to users a similar typology of services as the one that are aimed for the realization of the ParTI ecosystem. Mainly they allow users to navigate a map to look for restaurants in user's proximity, or they directly enable the users to scroll a list of restaurants in an area, and the way the list is sorted is customizable. In both ways then the user can select and acquire information on a specific restaurant. The information can be objective, such as the menu, opening hours, address, and so on; or subjective, e.g., the experiences of other users shared in reviews.
In this section it will be analyzed one of the most popular application in this context, the well-known TheFork.

2.2.1. TheFork

TheFork [3] is a french platform created in 2007 by Bertrand Jelensperger, Partrick Dalsace and Denis Fayolle, originally named LaFourchette. In 2014 was bought by the american website TripAdvisor, and after the success and the expansion of the application across all Europe, in 2020 the name has been changed to TheFork to create a uniform and recognizable brand.

TheFork offers the possibility to the user to select an area near him to start the search of restaurants. The application has a partnership with over 55000 restaurants and TheFork shows to the users the subset of partners located in the selected area. As case study we can take Fribourg as area of interest, and by doing so we obtain a set of 24 restaurants that we can inspect, as shown in Figure 2.1.

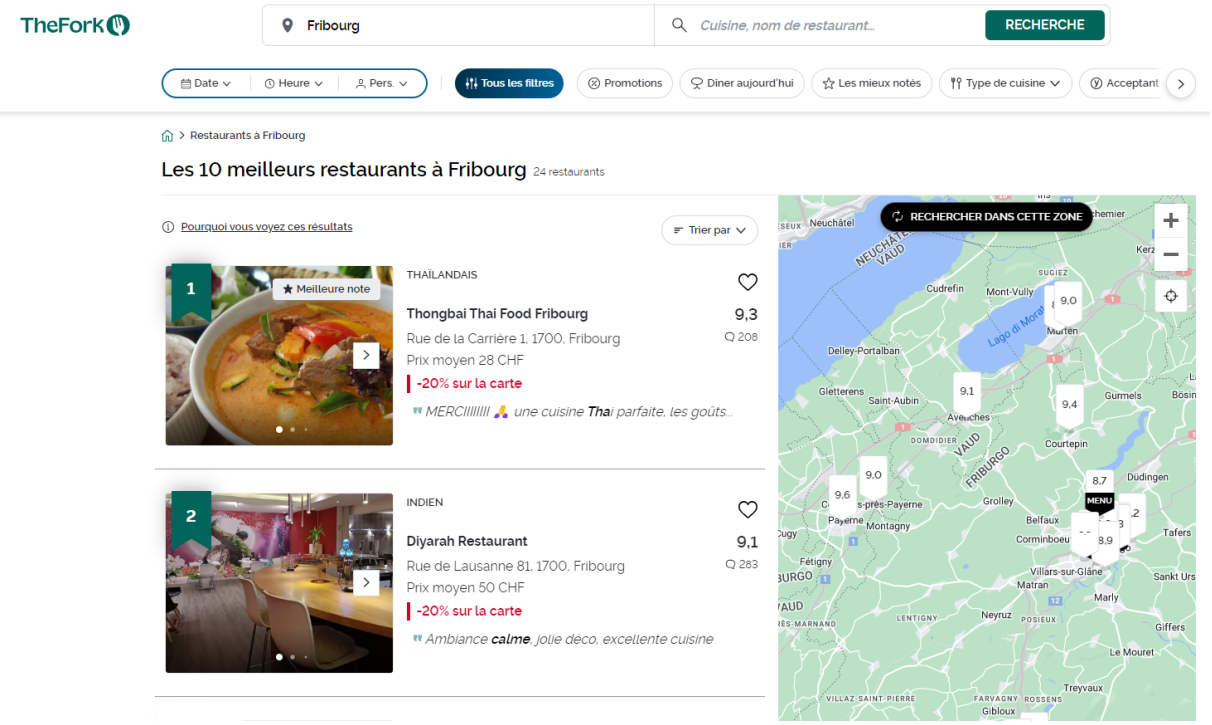

Figure 2.1.: TheFork screenshot - Fribourg partners

The user has then two options to obtain more information on a restaurant: he can scroll and click on a restaurant with the left GUI component; or he can use the map on the right and click on a marker to visualize a tooltip of the clicked restaurant, Figure 2.2.

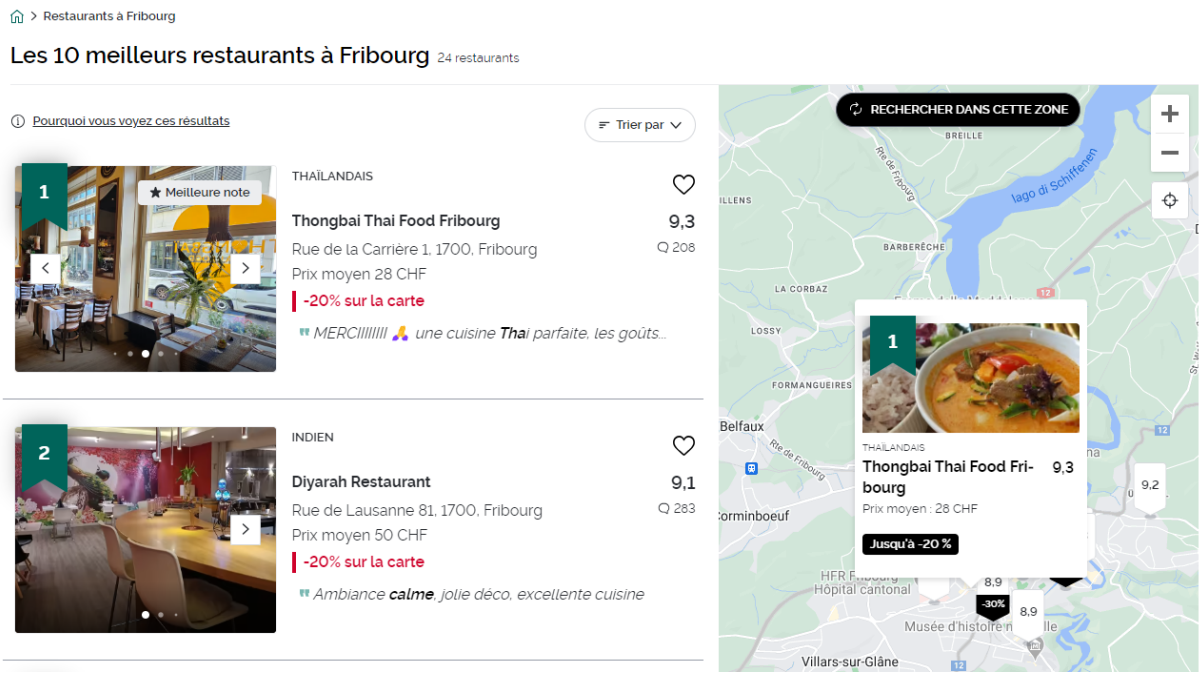

Figure 2.2.: TheFork screenshot - tooltip on the map

Once clicked on a restaurant it is possible to see its menu and many other additional information, as shown in Figure 2.3.

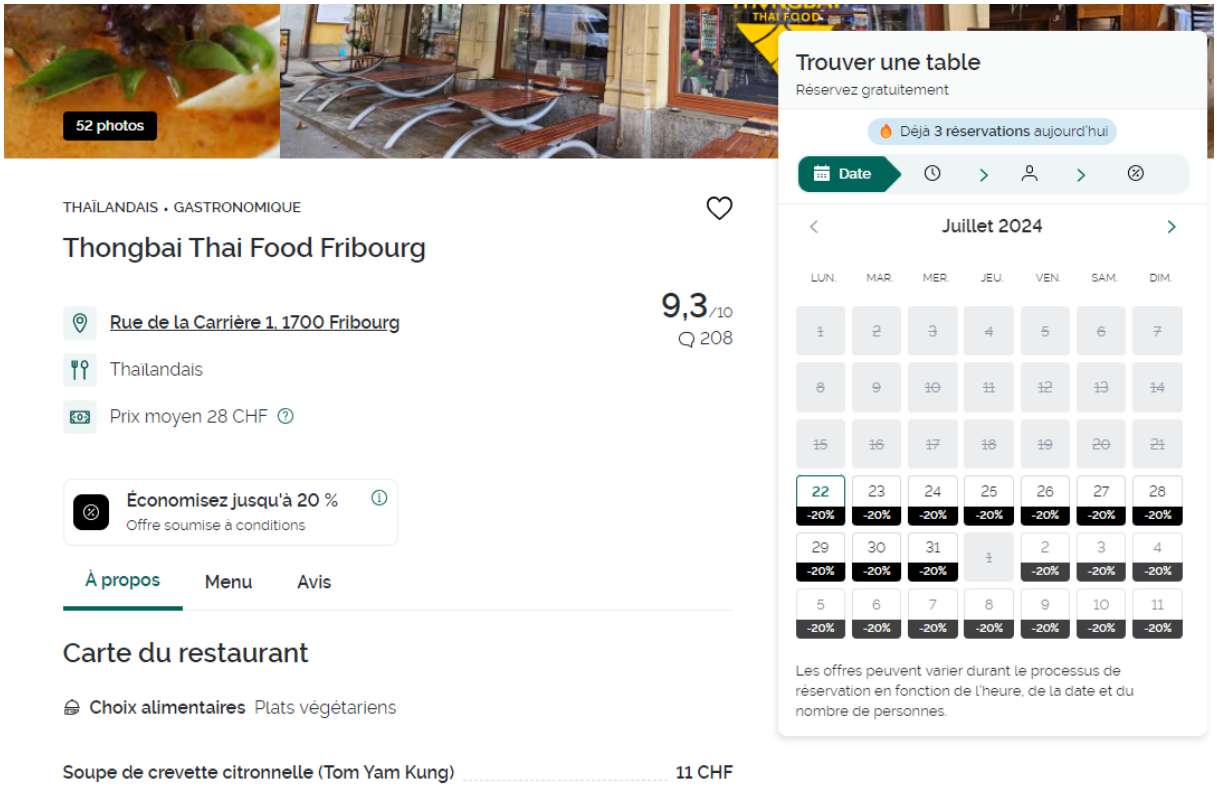

Figure 2.3.: TheFork screenshot - information on a restaurant

By scrolling down on the restaurant page the user can reach the reviews section, where he can consult the reviews written by other users, Figure 2.4.

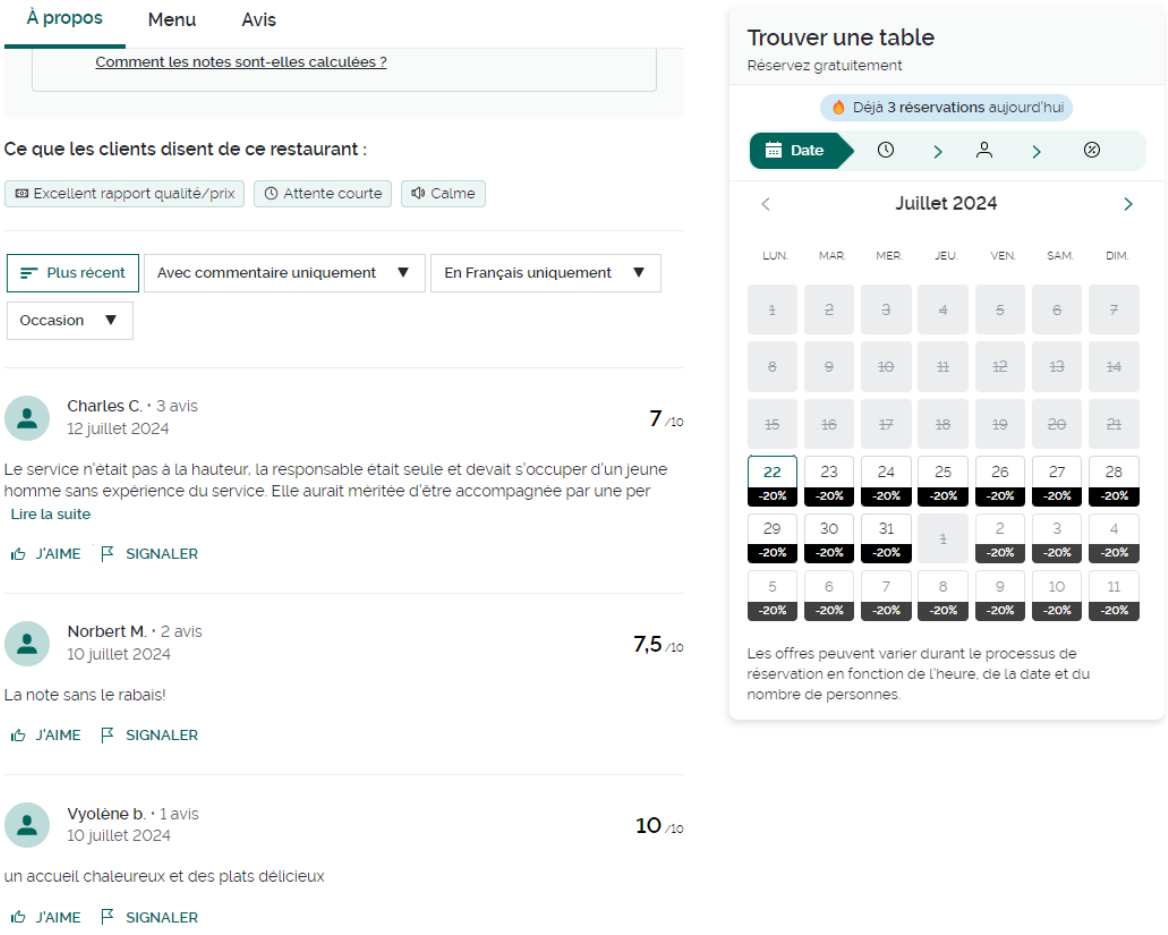

Figure 2.4.: TheFork screenshot - reviews' section

These discussed main characteristics will be present in the ParTI application, i.e., being able to navigate a map, obtaining more information on a restaurant and see the reviews, as well as other functionalities.

2.3. ParTI Application

2.3.1. Introduction

ParTI application has the aim of allowing users to inform themselves on the bars and restaurants in the region of Canton Ticino. In this section we will see in more detail what are the use cases that the application must provide to the users, and consequently a way to structure the database to support the ecosystem.

2.3.2. Use cases

The application idea comes with three distinct actors: clients, businesses and administrators. Each one of these actors has its own specific use cases, however the application was developed by starting to focus on the clients' use cases, and this work will only cover the latter. Therefore from now one we assume that businesses, word that I use to refer to both restaurants and bars, cannot actively participate into the application, even though the idea is to make it possible in the future. Instead there is a single administrator, myself, that is able to communicate with the businesses and manage their profiles on their behalf.

The ensemble of clients' use cases are shown in Figure 2.5, in which we can identify five subgroups distinguished by colors:

1. Use cases related to the client account (yellow)
2. General use cases on a restaurant (blue)
3. Use cases related to reviews on businesses (green)
4. Use cases related to comments on reviews (beige)
5. Use cases related to businesses' events and offers (red)

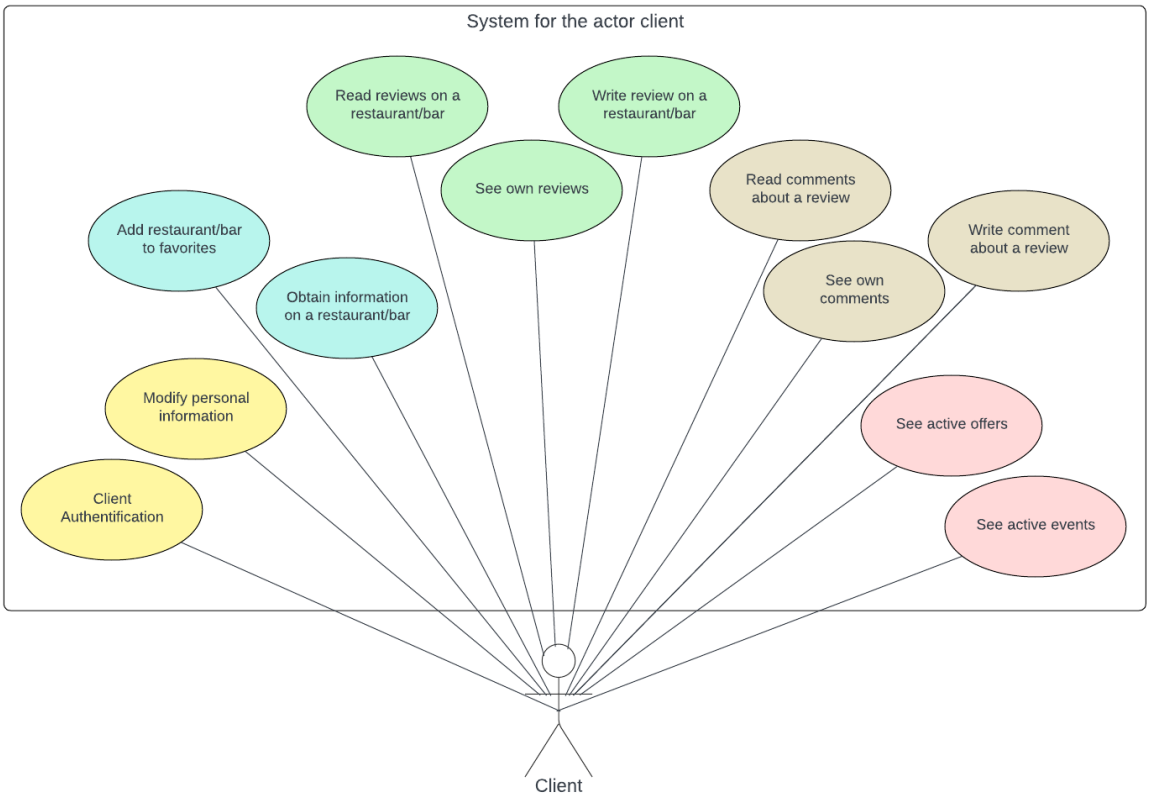

Figure 2.5.: Clients' use cases

The first subgroup implies that there must be a way to authenticate clients in the application, and that clients must be able to access and modify their personal information: first name, last name and email.
The second subgroup is a more general one and it defines that a client must be able to

visualize businesses that are application's partners, see their information as well as add them to their favorites to have a shortcut access in the future.
The third and fourth subgroup are built analogously, and represent the actions that clients can perform on respectively reviews and comments. For both reviews and comments the client must be able to read them, write them and see his own.
The fifth and last subgroup is related to the additional information that a client can obtain about a business. There are two types of additional information: offers and events. An offer represents an opportunity for the client to save money, and the business must define a validity time interval for the offer. Instead an event is something that is organized by the business and it doesn't necessarily translate in saving money for the client, but instead it consists in offering to the client a special experience. For example for a bar it could be an evening with theme music or a darts tournament, while for a restaurant it could be a dinner with multi-ethnic cuisine or accompanied by live music.

The definition of these use cases specify the different goals that the final ecosystem wants to reach. For a single use case there are multiple sub-goals, for example by taking the use case regarding obtaining information on a business, we can develop different ways to make it feasible, e.g., list of businesses and map, as seen implemented by TheFork.

2.3.3. Database

The first step in the development of the ecosystem is the definition of a database. The database represents the foundations of the application, and it needs to be defined in a way that makes the future implementation of the use cases the most straightforward possible. A good way to define the database is to create an ERD [4], even if then in the actual implementation a non-relational database may be used.

To understand what are the necessary entities we can analyse the clients use cases in Figure 2.5, from it we identify the main entities:

- Client
- Business: restaurant or bar
- Review
- Comment
- Offer
- Event

To find the relationships between the entities we need to think at the interactions that are implicit in the use cases. Starting from the Business entity, we have four interactions:

1. A Business can organize none or multiple Events
2. A Business can make available none or multiple Offers
3. A Business can have none or multiple Reviews
4. A Business can be the favorite of none or multiple Clients

From this defined interactions it is possible to draw an ERD with the main focus on the Business Entity, that is the entity involved in the largest number of relationships. To make it easier to follow I have drawn intermediate diagrams, that represent a sort of different steps to create the final ERD of the ecosystem, and the first one is shown in Figure 2.6.

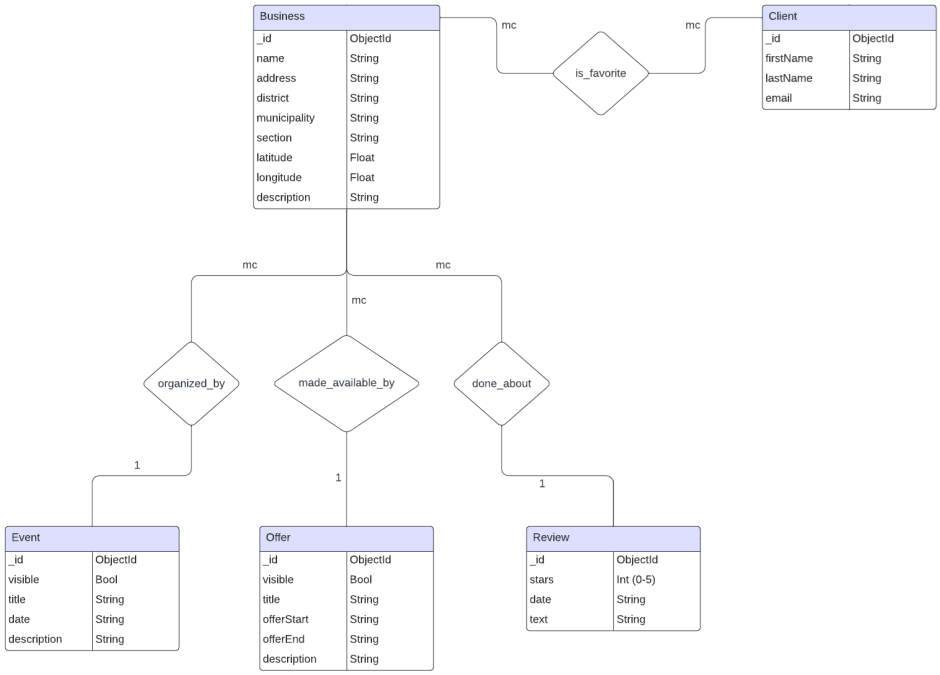

Figure 2.6.: Intermediate ERD - Business Entity

Now we can focus our attention on the Review Entity, and from the use cases are extracted two necessary relationships involving a Review:

1. A Review can have none or multiple Comments
2. A Review is written by a Client

These two new Review's relationships can be added to the intermediate ER diagram in Figure 2.6 to create a new updated version, shown in Figure 2.7.

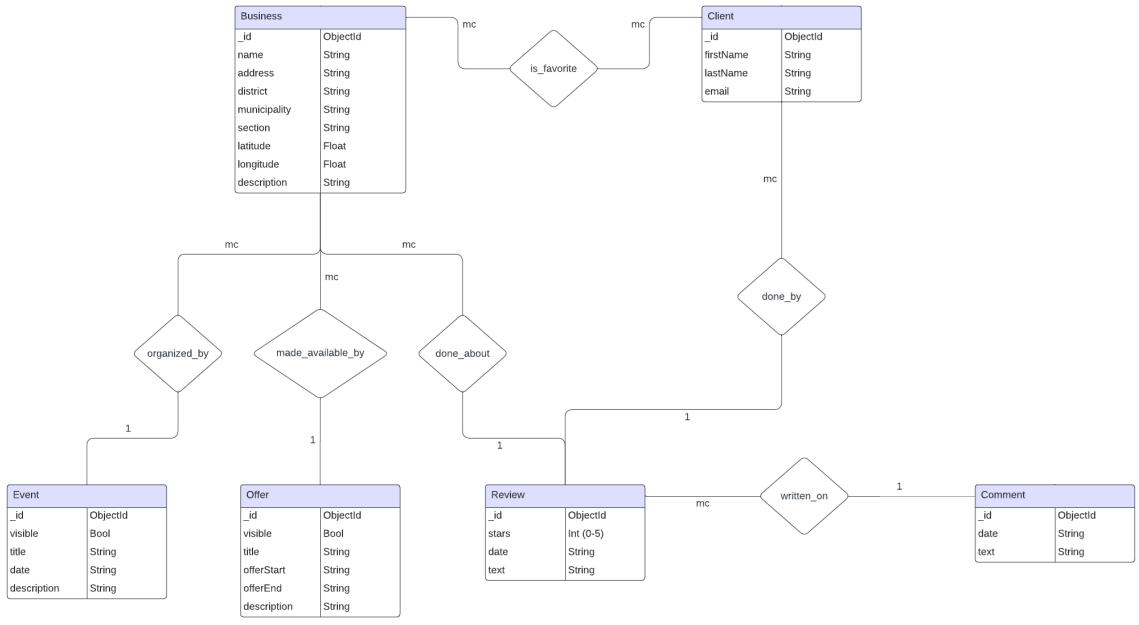

Figure 2.7.: Updated Version Intermediate ERD - Review Entity

From the client's use cases lastly we need to add to the ERD the relationship standing between a client and a comment, i.e., a comment is written by a client. By only considering the client's use cases this would be achievable simply by adding a relationship between the entities Client and Comment, stating that a client can write none or multiple comments. However this solution is not appropriate if for example we would like that in the future, when treating Businesses' use cases, a Business has the ability of writing a comment on a review that it has received. For this reason I opted for a less straightforward solution but that leave more room for maneuver for future implementations, that consists in adding a new entity, User, that is a common parent entity for businesses and clients. In this way the relationship stating who is the author of a comment is between the entities Comment and User, and a User can be both a Client or a Business. The final ERD of the ecosystem is shown in Figure 2.8.

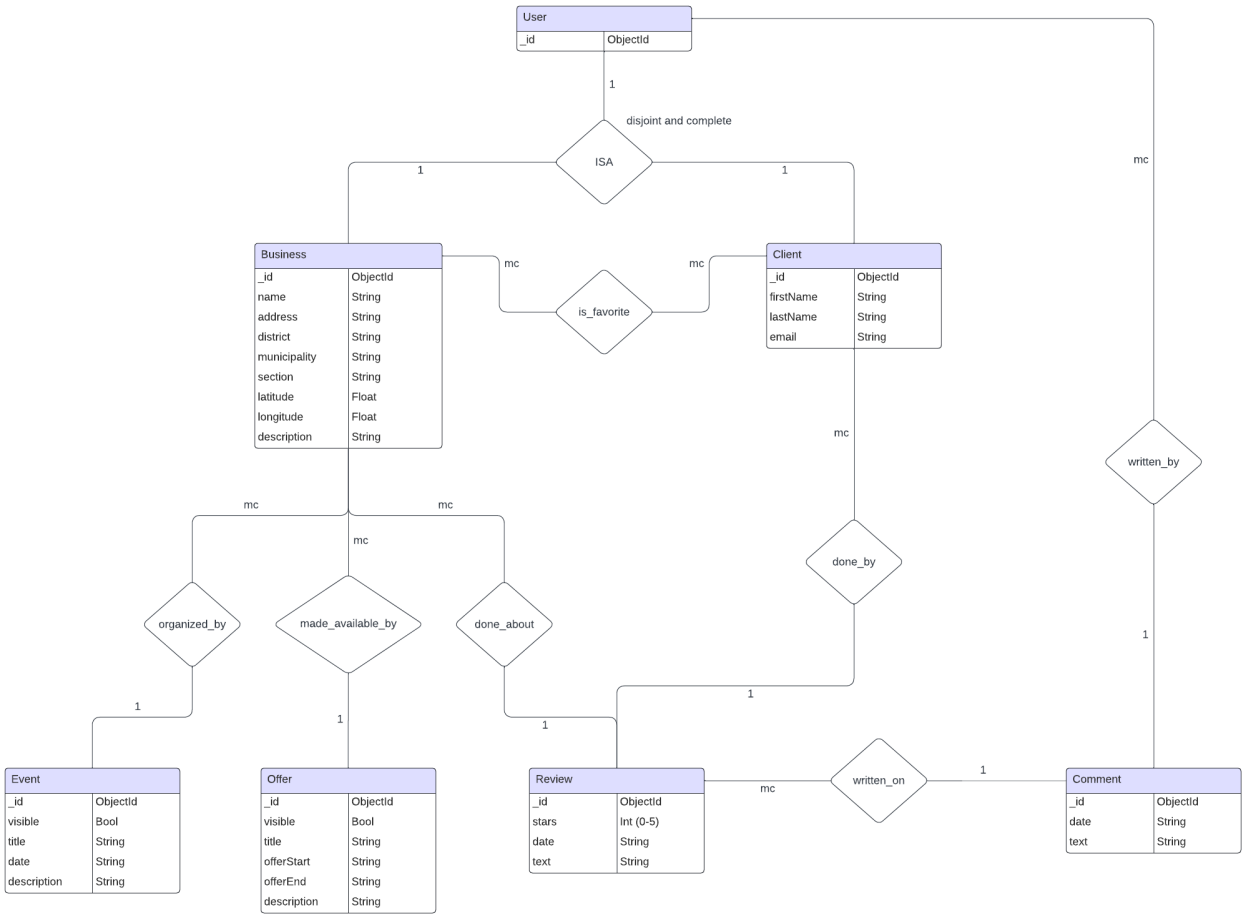

Figure 2.8.: ERD ecosystem's database

3
User point of view

3.1. Introduction

On this chapter is discussed the final result of the ParTI application from the point of view of a User. In particular firstly is shown the GUI in its different components, and then two different scenarios are presented in which are demonstrated how some use cases are covered by the application.

3.2. Global overview

The ParTI application main page is divided into two sub-components, following analogously the visualisation concept of TheFork. On the left sub-component there is a list of available businesses, while on the right there is the map with markers for each one of the available businesses, as shown in Figure 3.1.

3.2.1. Map component

The user can navigate the map component using the selectors on top of it or by manually zooming. By clicking the district selector it is possible to choose one of the eight districts of Canton Ticino, and by clicking one of them, the map automatically zooms in on the district and it shows its borders, as shown in Figure 3.2.

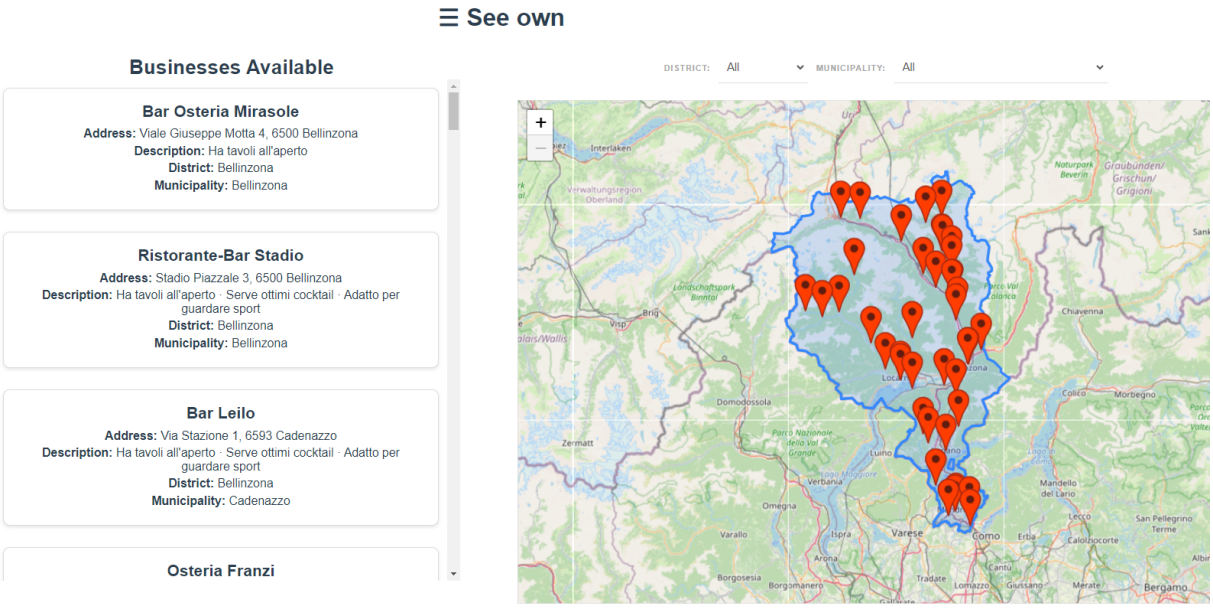

Figure 3.1.: ParTI main page

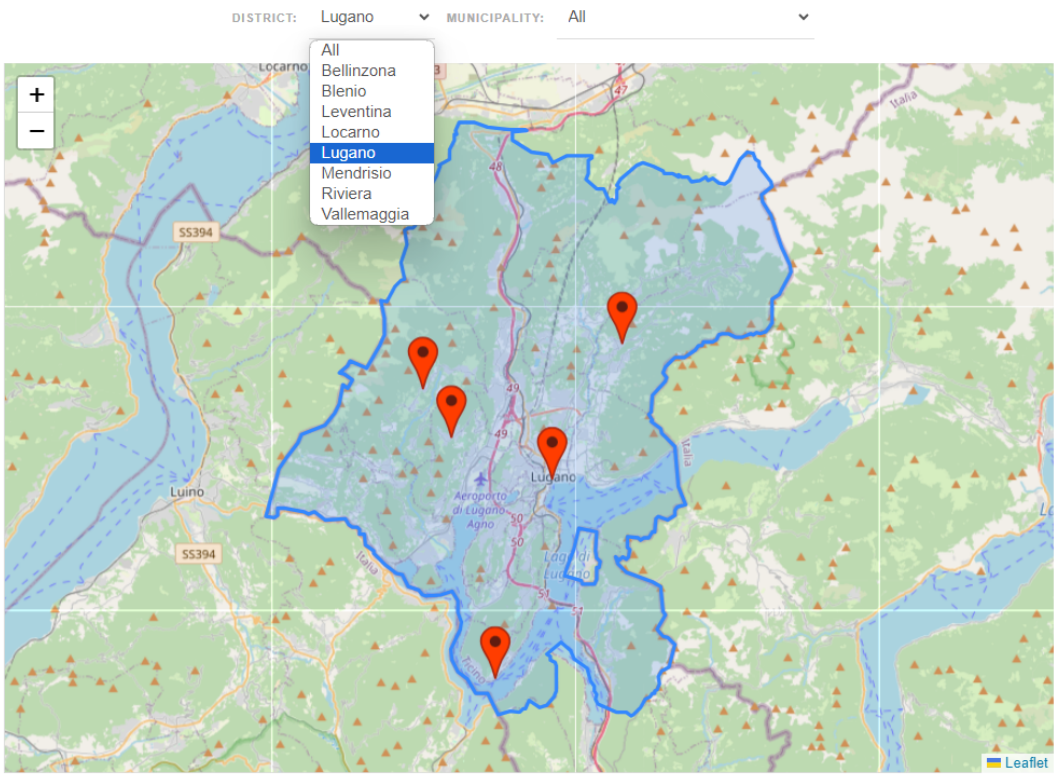

Figure 3.2.: ParTI Map - select a district

In function of the district selection the options listed in the municipality selector are dynamically modified to contain only the municipalities located in the selected district. By clicking on a marker on the map it switches color to green to distinguish it from the

others, as shown in Figure 3.3

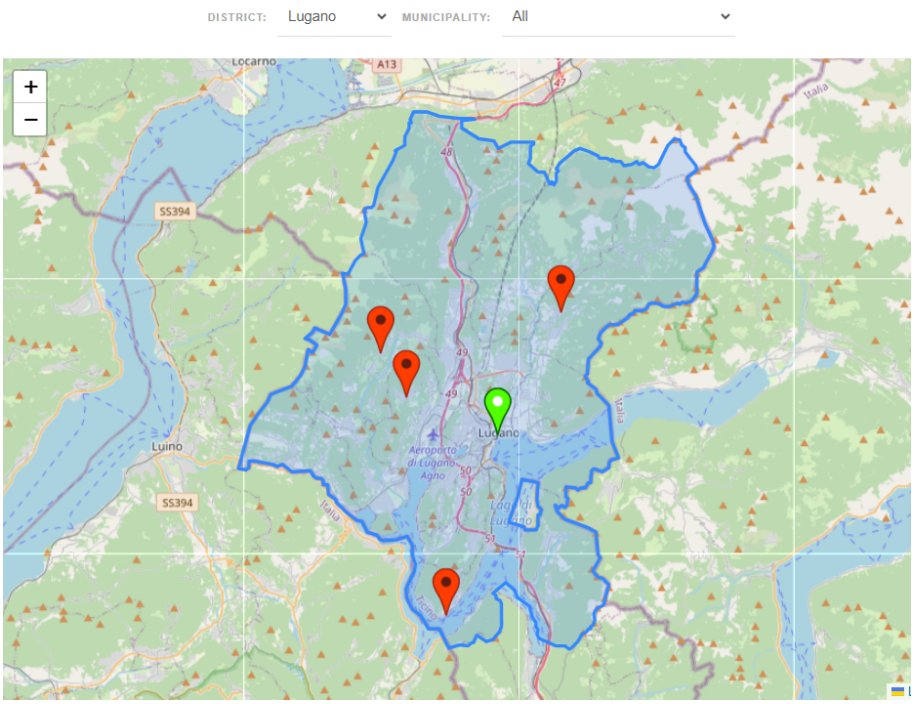

Figure 3.3.: ParTI Map - select a business

Instead of showing a tooltip once a marker is clicked as seen in the Fork implementation, there is a connection between the two GUI sub-components, so that when a marker is clicked on the map the corresponding business card in the list of businesses switches color to green and it is put at first place in the list, as shown in Figure 3.4. Vice versa when clicking on a business on the list its corresponding marker on the map becomes green.

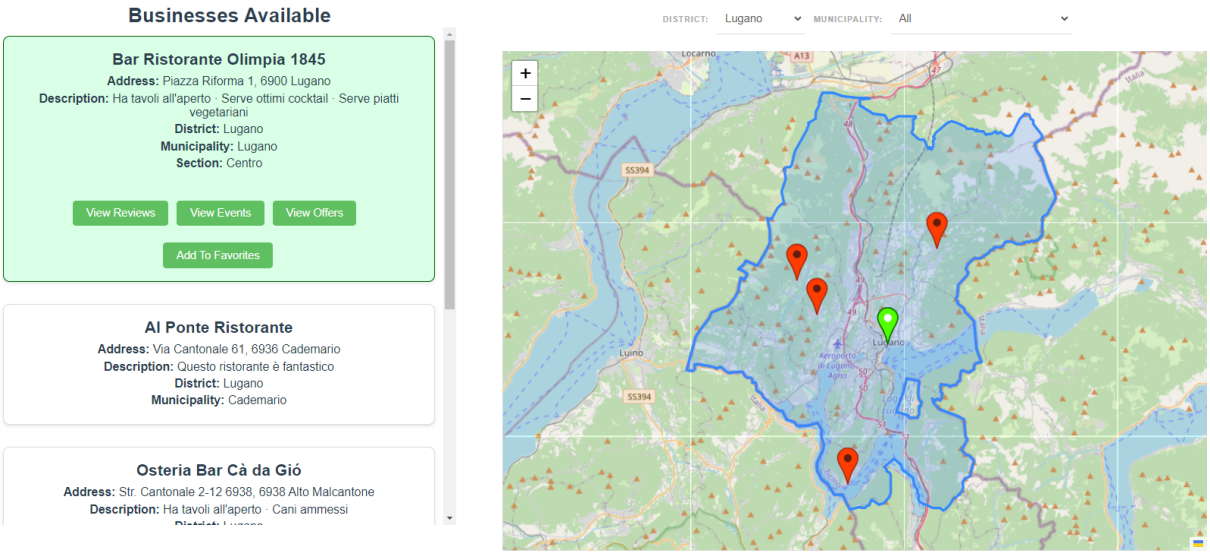

Figure 3.4.: ParTI Map - effect on list component

3.2.2. List component

As already mentioned the component containing the list of businesses and the map component are interconnected. When a business on the list component is selected, as shown in Figure 3.4, then it is possible to obtain more information on the selected business. By clicking on the *view reviews* button a pop-up appears that shows all the reviews on the business, and on each review it is possible to click on *view comments* to see all the comments on the review, Figure 3.5.

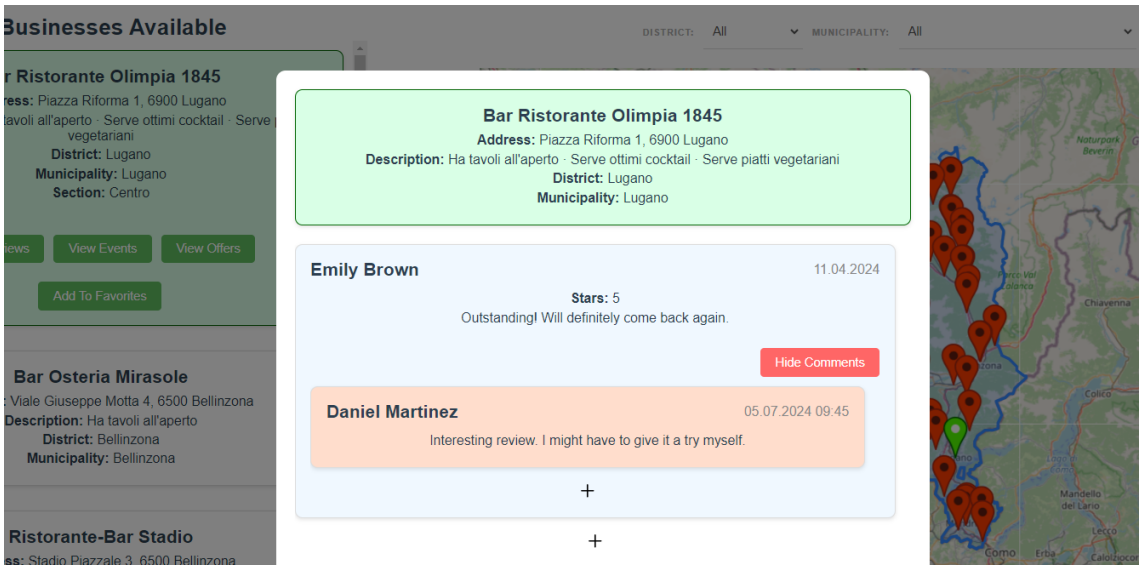

Figure 3.5.: ParTI List - view reviews and comments

By clicking the plus symbol at the bottom of the reviews' list it is possible to write a new review on the business, Figure 3.6.

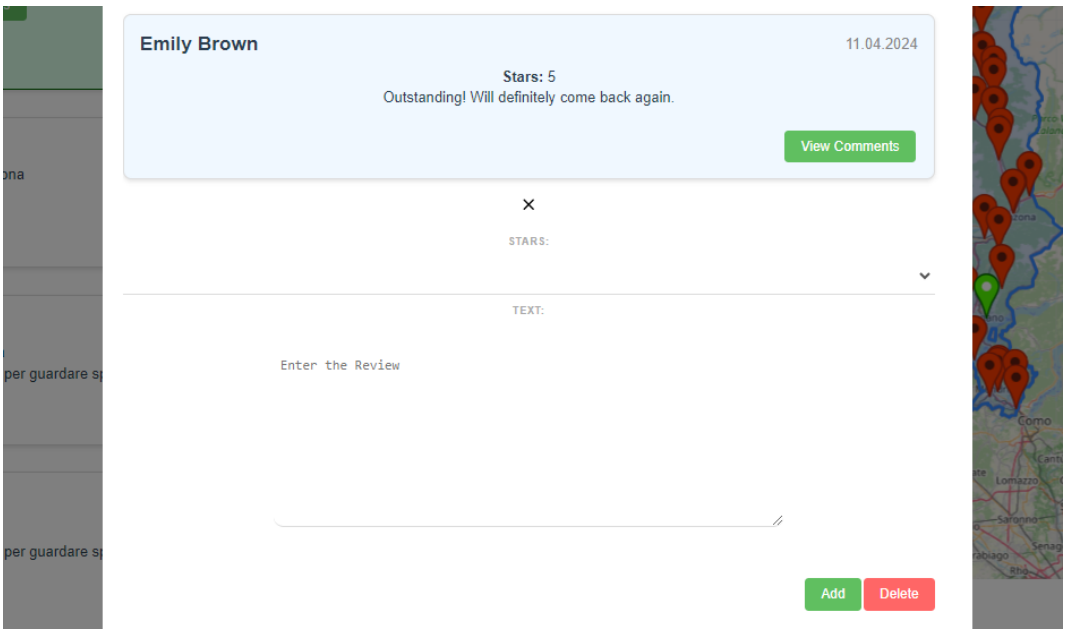

Figure 3.6.: ParTI List - write a review

Analogously by clicking on the plus symbol at the bottom of the comments' list it is possible to write a new comment on a review, Figure 3.7.

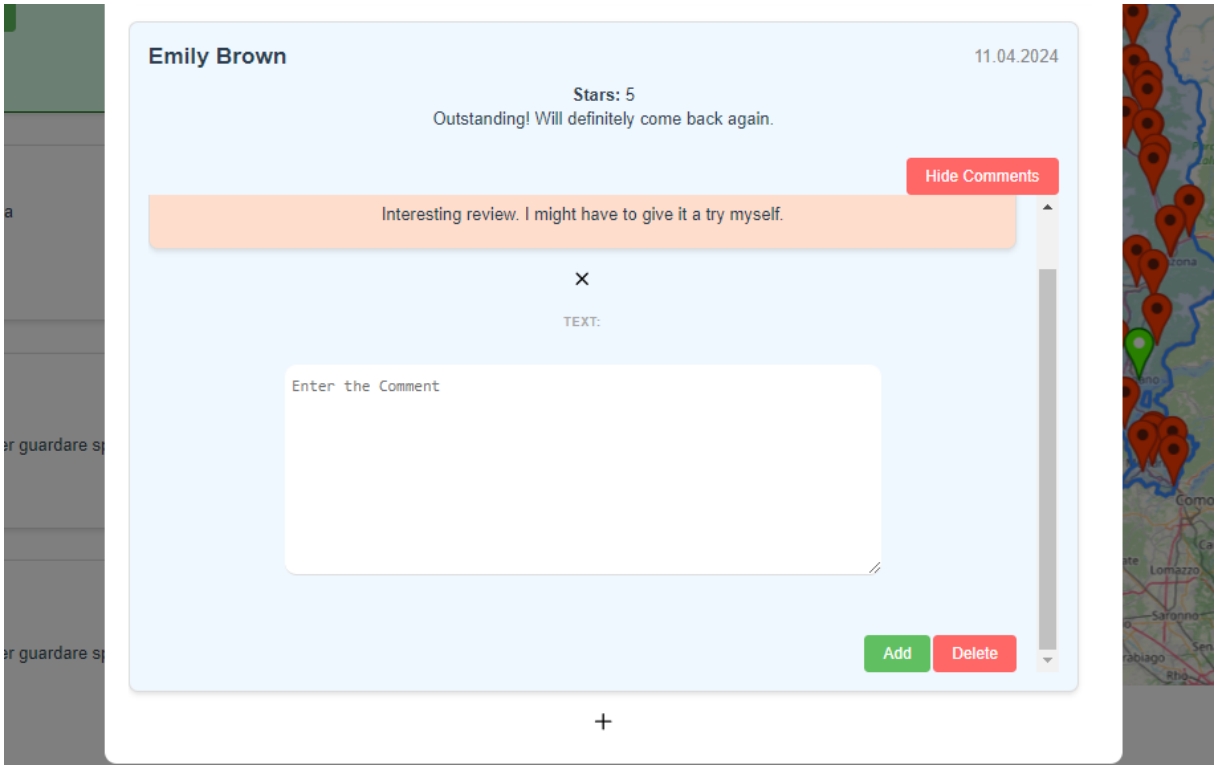

Figure 3.7.: ParTI List - write a comment

With the same logic by clicking on *view events* and *view offers* it is possible to visualize on the pop-up respectively the events and the offers. Finally there is the last button that makes sure that the user can dynamically keep a list of its favorite businesses, as shown in Figure 3.4. In case the client has already inserted into the favorites the selected business there would be a button that allow the user to remove the business from its favorites, as visible in Figure 3.8.

3.2.3. See own

The user can maintain a list of its favorite businesses, to visualize this list and visualize other personal information there is the see own menu. This menu gives access to the client's personal information, favorites, reviews and comments, as shown in Figure 3.8.

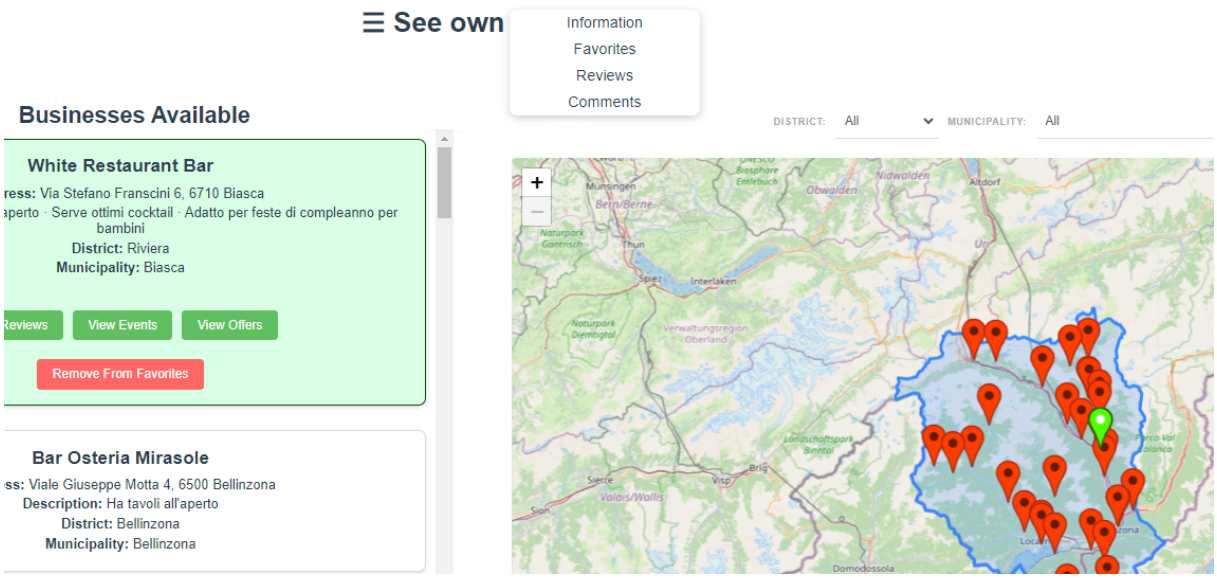

Figure 3.8.: ParTI Menu see own

By clicking on one of the four options a corresponding pop-up will appear on the screen, for example by clicking on *information* the user can visualize its personal information: first name, last name and email, Figure 3.9.

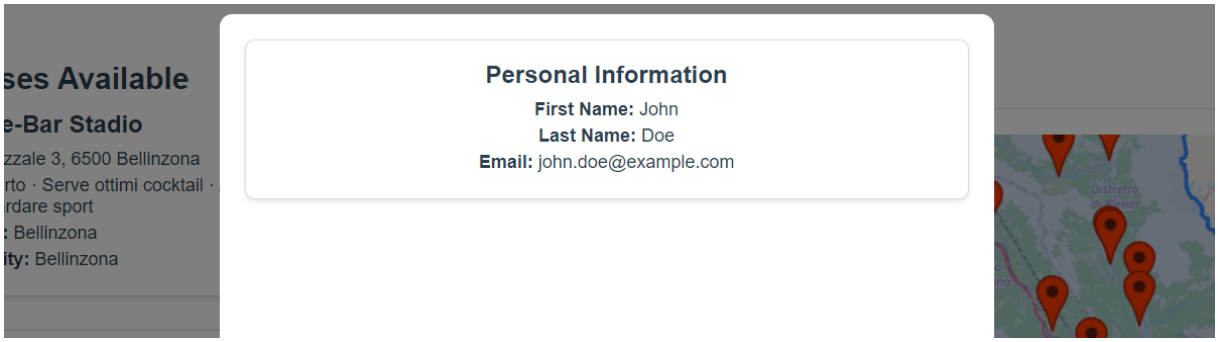

Figure 3.9.: ParTI user personal information

3.3. Scenarios

3.3.1. Write review on a business

In this scenario we assume that the client John Doe has eaten in the restaurant Reginella in Mendrisio. He is really satisfied about it and he would like to write a positive review and add the restaurant to its favorites. To do so therefore he first searches for the business on the map by selecting as municipality Mendrisio, as shown in Figure 3.10.

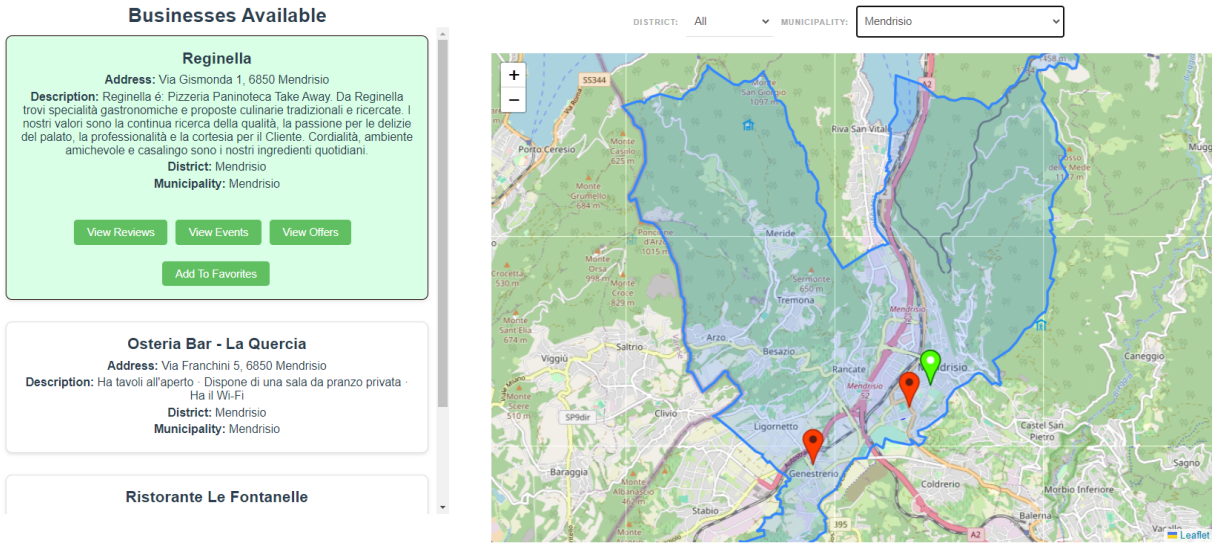

Figure 3.10.: Scenario 1.1 - find the business

Once Reginella is found, he can add it to its favorites and open the review pop-up to write a new review, Figure 3.11. The addition of the review is finalized by clicking *add*, and the result of this action is shown in Figure 3.12.

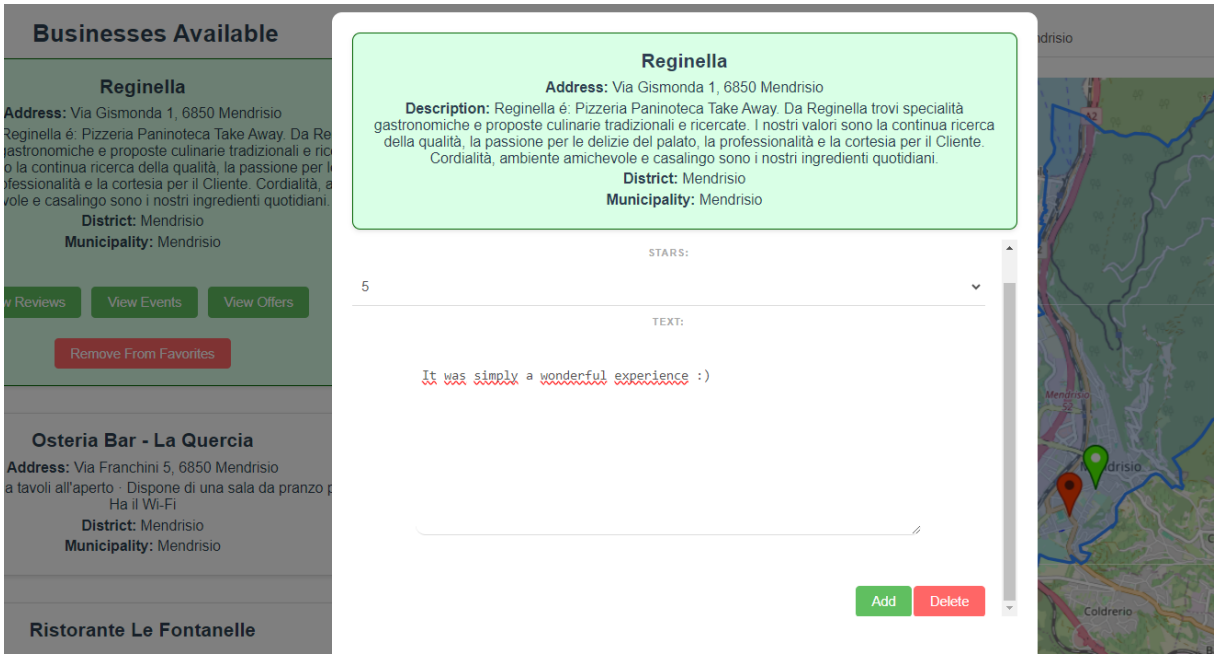

Figure 3.11.: Scenario 1.2 - add to favorites and write review

Now we can assume that after some time John Doe is again in Mendrisio but it doesn't remember anymore the name of that restaurant that had made such a good impression on him last time he was there. However he remembers that he had added it to its favorites in the ParTI application, therefore it opens up its favorites and he easily finds it, Figure 3.13.

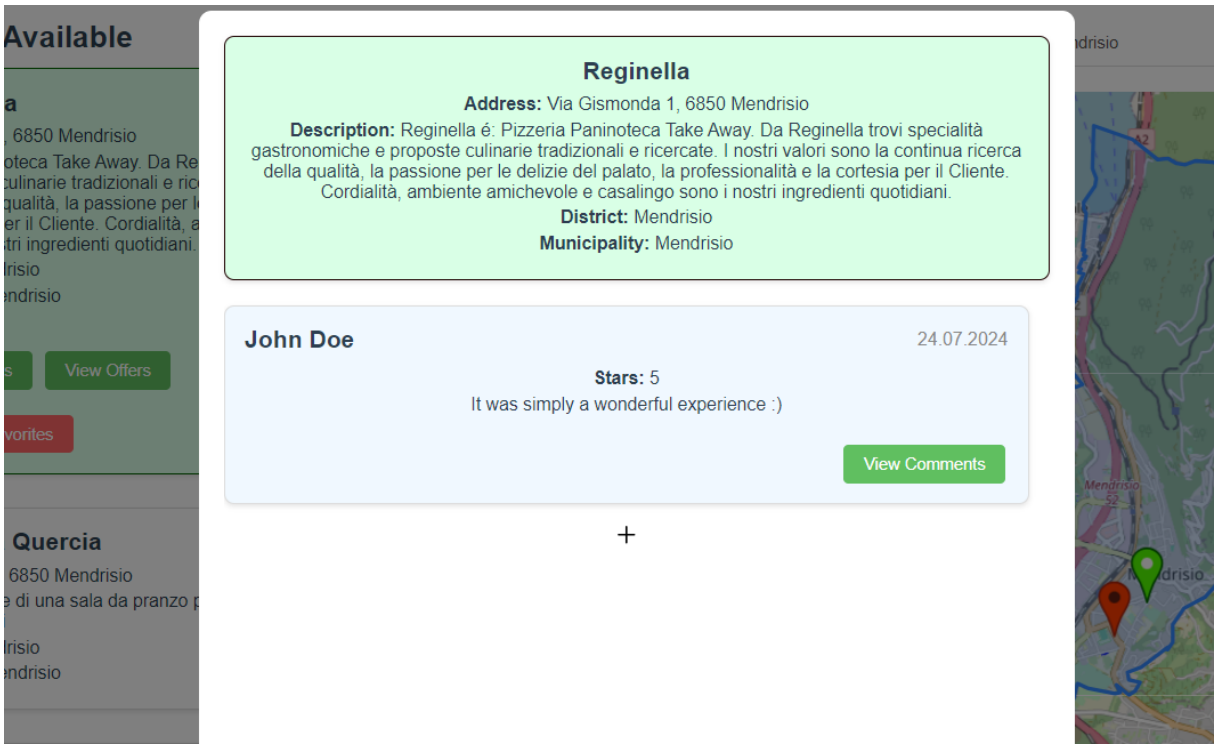

Figure 3.12.: Scenario 1.3 - new review

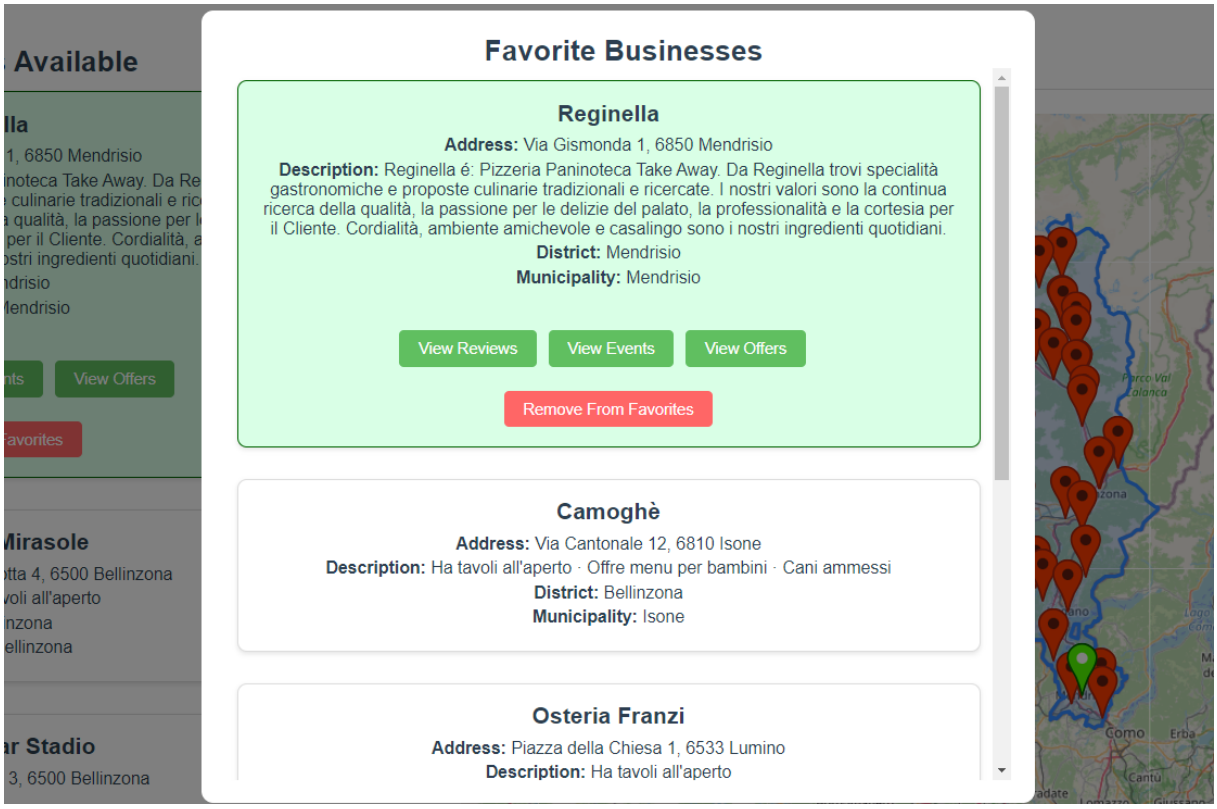

Figure 3.13.: Scenario 1.4 - see favorites

3.3.2. See comments under an own review

In this second scenario we assume that the client David Taylor wants to inspect the reviews that he has written to see if someone has written a comment about it. To do so firstly in the see own menu he selects the reviews, as shown in Figure 3.14.

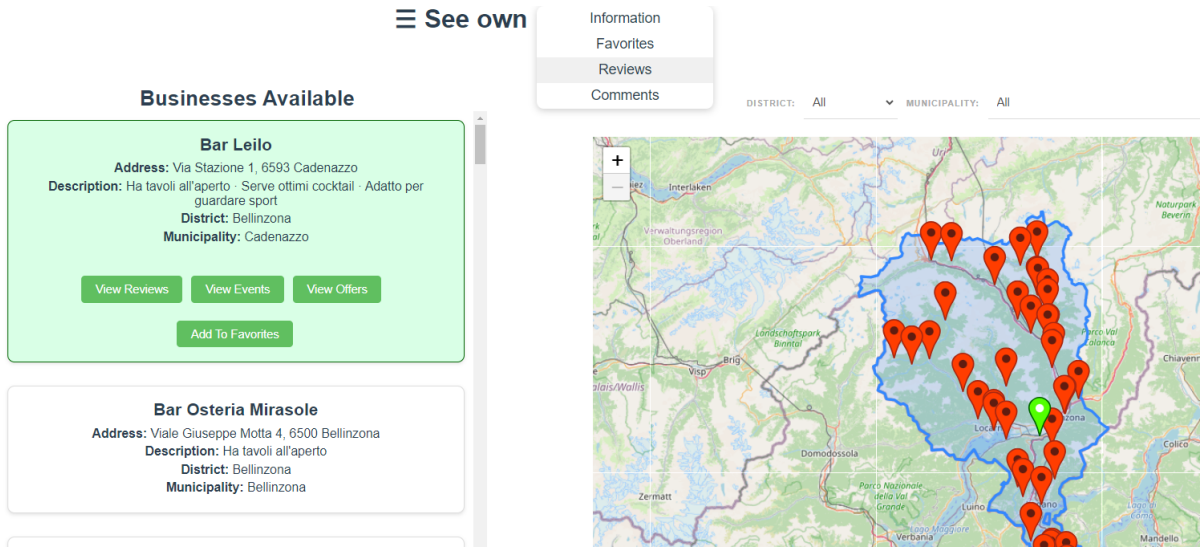

Figure 3.14.: Scenario 2.1 - see own reviews

Once opened the pop-up to visualize its reviews he can view the comments on each one of them, and ultimately he decides to answer to the comment done by Olivia Garcia, Figure 3.15.

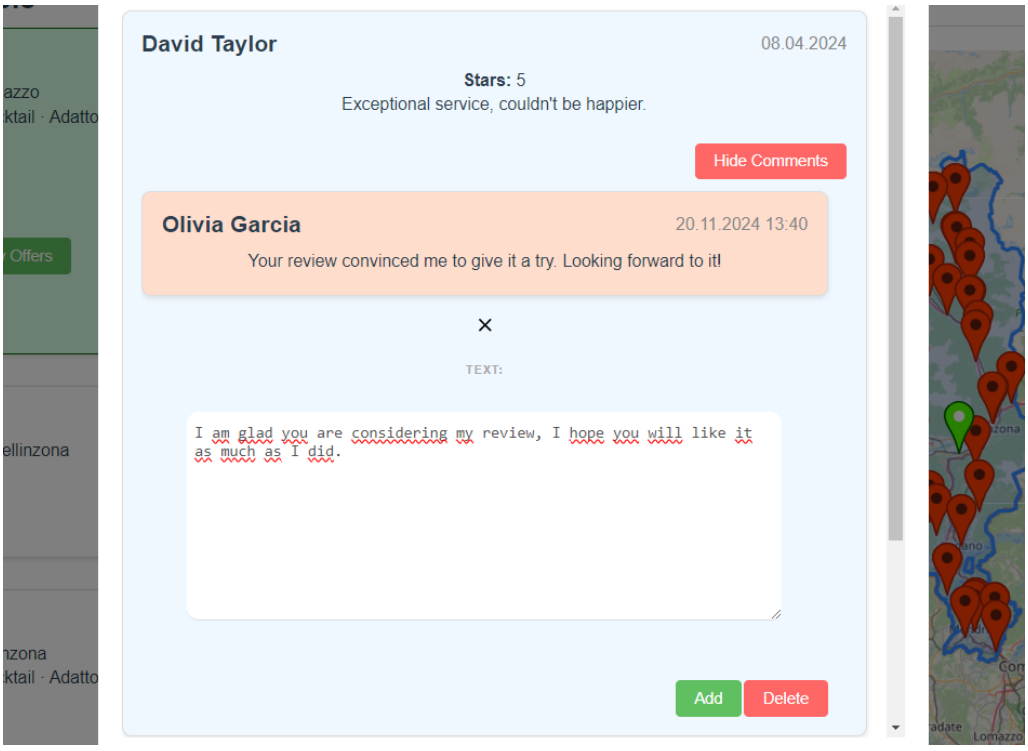

Figure 3.15.: Scenario 2.2 - see comments and answer to them

Once written the comment it is now possible to finalize the operation by clicking *add*, and then we can see the new comment under the review, Figure 3.16.

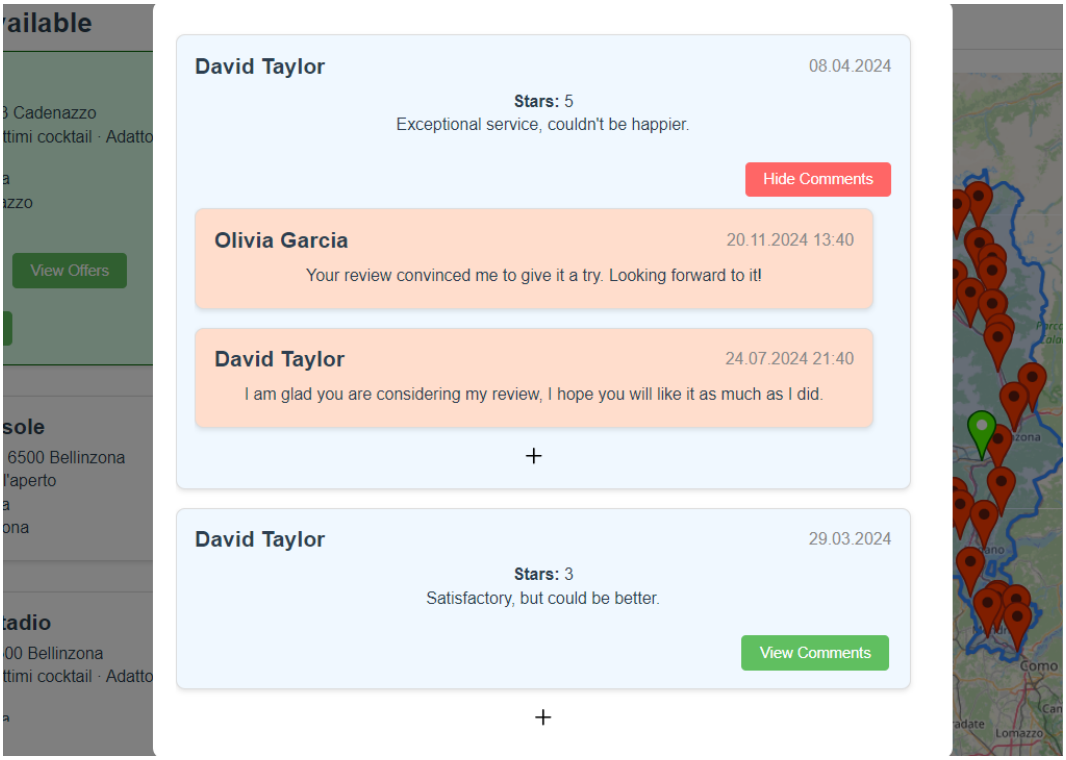

Figure 3.16.: Scenario 2.3 - visualize the newly written comment

4
Programmer point of view

4.1. Introduction

On this chapter we will discuss the ParTI application from the point of view of a programmer, and therefore we are going to discuss the different technologies used. With technology stack [6] we refer to the ensemble of programming languages, frameworks, libraries and technologies that combined create the ecosystem of the application. In this chapter we will treat the tech stack, starting from the database, then the backend and lastly the frontend. The ParTI application uses a full-stack JavaScript denominated MEVN, that stands for MongoDB, Express.js, Vue.js and Node.js, each singular of these technologies will be briefly discussed.

4.2. Database

Previously we have drawn an ERD for the application's database, Figure 2.8. As already mentioned, typically an ERD is drawn as a model for implementing a relational database, however for this application I have chosen to work with MongoDB. MongoDB [7] is a popular open-source NoSQL database, it is therefore non-relational, meaning that it is

not structured with tables and rows, but instead MongoDB is based on Collections and Documents. A collection contains multiple documents and it can be interpreted as an Entity Type, while a document can be seen as an actual Entity of a specific Entity Type. A document represents a record on the database, and it is stored as a Binary JSON (BSON), that is a special type of JavaScript Object Notation (JSON) format.

The implementation of the database in MongoDB is pretty much straightforward, for each Entity that we have we create a Collection, and inside each collection we populate the database by adding Documents. With MongoDB it comes a GUI called MongoDB Compass that facilitates the creation and population of the database, for example in Figure 4.1 we can see on the left the database called bachelor_work, that contains seven different collections and by opening the businesses collection there are multiple documents in JSON format.

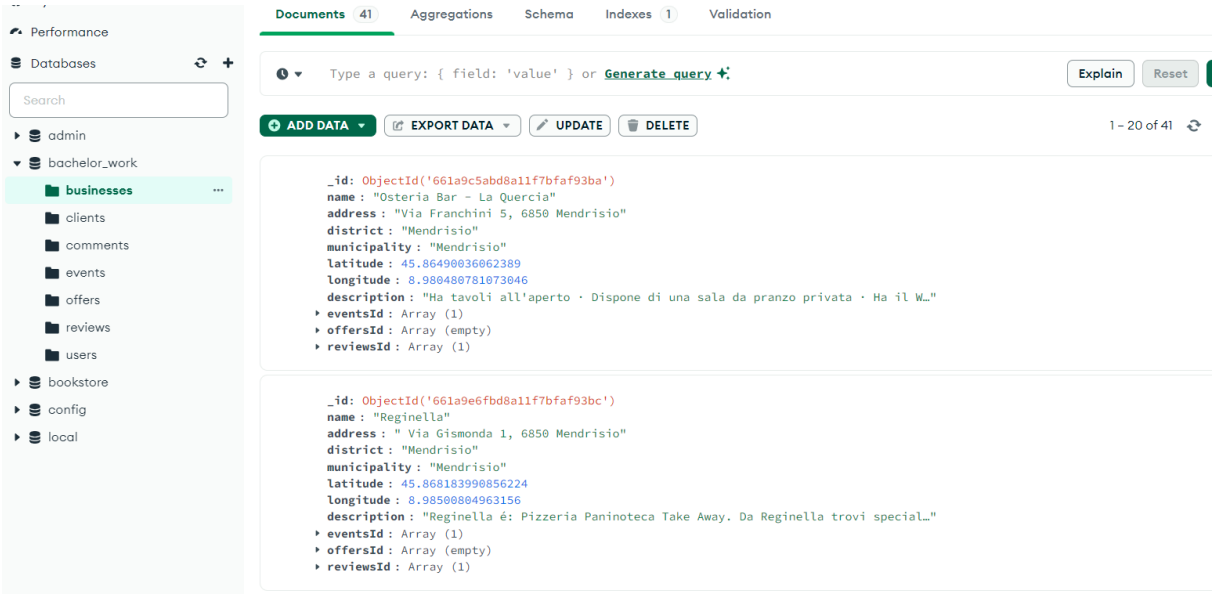

Figure 4.1.: MongoDB Compass screenshot

When inserting a new document in a collection it is not necessary to define explicitly the _id field, this field is added automatically by MongoDB, and the ObjectID generated is a 24-character hexadecimal string that uniquely identifies the document on the database. To implement the relationships between different Collections, I have added some additional fields that act as foreign keys, as shown in Figure 4.1 in a business document we have three fields that contains arrays of ids: eventsId, offersId and reviewsId.

To start working at the application with some entries in the database I have manually created a test database with 40 businesses, 10 clients, 20 reviews, 20 comments, 10 offers and 10 events. Having a test database we want now to start doing something with the data stored in it. To communicate with the database programmatically and not using the MongoDB Compass GUI or the mongo shell, we have to choose a MongoDB driver [8], and the choice depends on how we want to implement the backend of the application.

4.3. Backend

As already mentioned the ParTI application was conceived using the full-stack JavaScript MEVN, therefore the technologies used for the backend are Express.js and Node.js. Node.js [9] is a free, open-source JavaScript runtime environment that allows to create servers and web apps, while Express.js [10] is a web application framework for Node.js specialized in making quick and easy the creation of an Application Programming Interface (API). The API is the interface that will establish a way of communicate between the frontend and the backend of the application, concretely it will allow the user to request information about bars and restaurant of Canton Ticino. The API implemented will be of type Representational State Transfer (REST), that means that it uses the standard Hypertext Transfer Protocol (HTTP) methods (GET, POST, PUT, DELETE, PATCH, ...); the communication is stateless and the resources are returned in this case in JSON format. The role of the backend is therefore to establish a connection with the database and handle requests from the frontend.

4.3.1. Database connection

Using the MongoDB driver we have to first of all create a connection to the database, to do so we simply need the Uniform Resource Locator (URL) that locates the application's database, as shown in Listing 4.1.

```
1  // MongoDB driver for node.js
2  const { MongoClient } = require("mongodb");
3
4  // Variable that stores the database connection
5  let dbConnection;
6
7  // Select the functions to export in a module in node.js
8  module.exports = {
9    // It has as argument a callback function.
10   connectToDb: (cb) => {
11     // The argument is a connection string, that is a special URL for a MongoDB
           database.
12     MongoClient.connect("mongodb://localhost:27017/bachelor_work")
13       .then((client) => {
14         dbConnection = client.db();
15         return cb();
16       })
17       .catch((err) => {
18         console.log(err);
19         return cb(err);
20       });
21   },
22   // Get the connection to the database.
23   getDb: () => dbConnection,
24 };
```

Listing 4.1: MongoDB database connection

This module now allows us to establish a connection to our database and this is the first thing that we do when the backend Node.js application is started, Listing 4.2.

```
// Require the package express
const express = require("express");
// Import functions defined to create and get connection to the database
const { connectToDb, getDb } = require("./db");
// Init app and middleware
const app = express();
// Call the imported function and define the callback function
connectToDb((err) => {
  if (!err) {
    // listen for requests on a port number, localhost:3000
    app.listen(3000, () => {
      console.log("app listening on port 3000");
    });
    db = getDb();
  }
});
```

Listing 4.2: Establish database connection

4.3.2. Backend REST API

The backend server REST API [11] defines what are the requests that a client's application can make and the responses it can expect. We have already defined the client's use cases, Figure 2.5, the REST API must cover all of them. For doing so I have defined the different routes shown in Table 4.1.

Method	Route
GET	.../businesses
GET	.../business/{businessId}
GET	.../businessesOnLocation?d={district}&m={municipality}&s={section}
PATCH	.../addFavorite/{businessId}
PATCH	.../removeFavorite/{businessId}
GET	.../businessReviews/{businessId}
GET	.../clientReviews/{clientId}
POST	.../addReview
GET	.../reviewComments/{reviewId}
GET	.../clientComments/{clientId}
POST	.../addComment
GET	.../businessOffers/{businessId}
GET	.../businessEvents/{businessId}

Table 4.1.: REST API backend application

The framework express.js facilitates the implementation of our REST API. To use the express framework we first instantiate the app as shown in Listing 4.2, then we can call special methods on the app instance dedicated to handle different types of HTTP requests, Listing 4.3.

```
// handle a GET request of a single business with a given id
app.get("/business/:id", (req, res) => {
  getDocById(db, "businesses", req, res);
});
// handle a POST request of inserting a business in the businesses collection
app.post("/addBusiness", (req, res) => {
  addUser(db, "businesses", req, res);
});
```

Listing 4.3: Express.js REST API implementation

In Listing 4.3 we see that to handle a specific type of request we give to the according express.js method two arguments, the route of the request and an anonymous function that handles the request. Taking as example the GET request of a single business with a given id the function that handles the request is the one shown in Listing 4.4.

```
function getDocById(db, collection, req, res) {
  // isValid checks whether the given id is a 24 character hex string, 12 byte
      Uint8array, or an integer
  if (!ObjectId.isValid(req.params.id)) {
    res.status(500).json({ error: "Not a valid doc id" });
  } else {
    // Instantiate an ObjectId of mongodb with the id in the req
    let idObject = new ObjectId(req.params.id);
    db.collection(collection)
      // Find the document with the given id
      .findOne({ _id: idObject })
      .then((doc) => {
        if (!doc) {
          res.status(404).json({
            error: "No doc in " + collection + " with the given id was found!",
          });
        } else {
          res.status(200).json(doc);
        }
      })
  }
```

Listing 4.4: Function to handle the request of a business

4.3.3. Postman

Postman is an API platform for building and using APIs, it comes with a user-friendly interface. I have used this tool to test my REST API implementation, indeed it is possible to define and save different requests with their paths and arguments. Then we can execute the request and retrieve and visualize the response from our backend server. The GUI of Postman with an example of request and visualized response is shown in Figure 4.2.

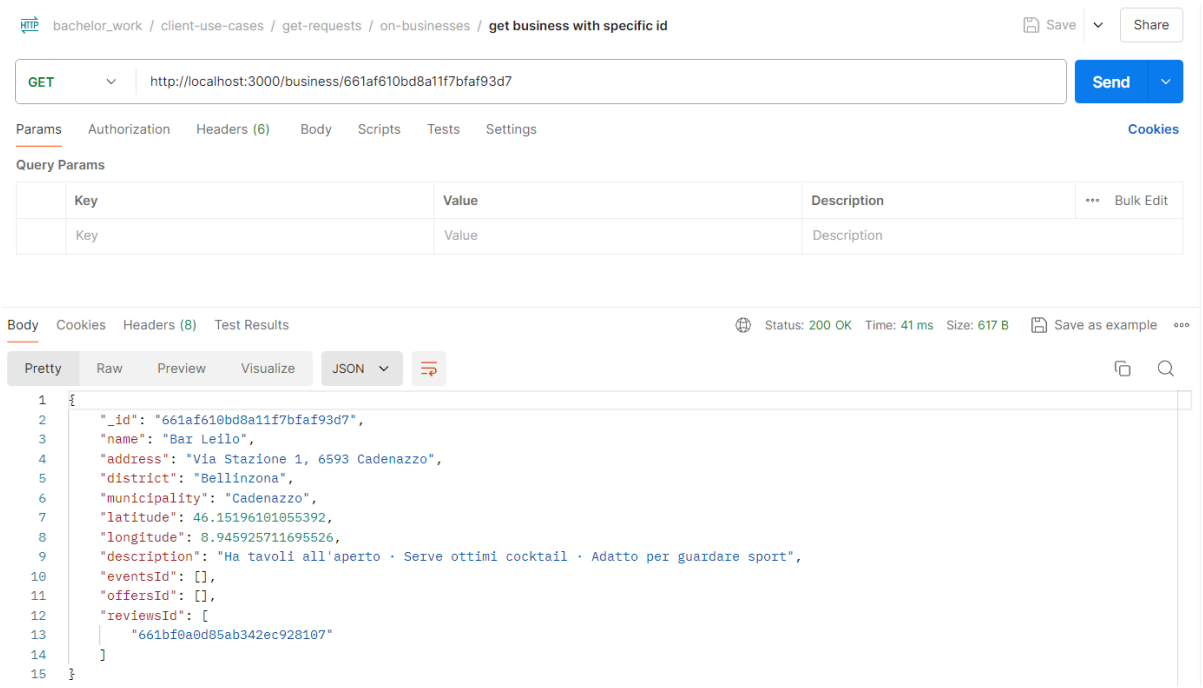

Figure 4.2.: Postman screenshot

4.4. Frontend

The frontend of the ecosystem represents the GUI of the application. The objective of the work is the realization of an ecosystem that provides to the user the set of functionalities defined by the client's use cases, Figure 2.5. The first component that I started developing for the frontend was the map. The map indeed is an important part of the GUI, it must show to the users where are located the businesses and the user must be able to navigate the map. The first approach was to obtain the basic geographical information of Canton Ticino, i.e., its border and its subdivision in districts and municipalities, and use it to draw a map.

4.4.1. GeoJSON data

Geographic JSON (GeoJSON) [12] is a widely used format in web mapping applications and it encodes a variety of geographic data structures using the JSON format. The structure of a GeoJSON file is configured in the following way:

- there is a top-level structure that is a collection of features (type FeatureCollection);
- a single feature (type Feature) is composed of two objects, geometry and properties;
- the geometry object defines the shape of the feature, the types of shapes are Point, LineString, Polygon, MultiPolygon, and so on;
- the properties object contains additional information about the feature, e.g., the feature's name.

As an example for the districts of Canton Ticino [13] we have a GeoJSON file with a FeatureCollection that has an array of 8 features, and each feature has its own properties and geometry objects, as shown in Listing 4.5.

```
{
  "type": "FeatureCollection",
  name: "Ticino",
  features: [
    {
      "type": "Feature",
      properties: {
        name: "Blenio",
        ...
      },
      geometry: {
        "type": "MultiPolygon",
        coordinates: [
          [
            [
              [9.040158148352059, 46.493947296456597, 3389.864999999999782],
              ...,
              [8.95397576570887, 46.356003928842917, 286.404]
            ]
          ]
        ]
      }
    },
    {
      "type": "Feature",
      properties: {
        name: "Mendrisio",
        ...
      },
      geometry: {
        "type": "MultiPolygon",
        coordinates: [
          [
            [
              [8.939342910131256, 45.86379349305961, 440.334],
              ...,
              [8.939342910131256, 45.86379349305961, 440.334]
            ]
          ]
        ]
      }
    },
    ...
  ]
}
```

Listing 4.5: Districts' GeoJSON

4.4.2. D3.js

The first technology that I have decided to use to implement a map visualization was D3.js. D3.js [15] is a JavaScript library that is specialized in producing dynamic and interactive data visualizations in web browsers. It binds data to Document Object Model (DOM) elements and it makes it possible to use the GeoJSON data to plot a map as a Scalable Vector Graphics (SVG). To do so using D3.js three steps are necessary:

1. Select a element in the DOM
2. Create a function that projects latitude and longitude coordinates from the GeoJSON data into pixels coordinates in the element.
3. Bind the GeoJSON data to create the map, it is also possible to add event listeners to each feature.

Following the previous steps we obtain the code in Listing 4.6.

```
1  // Creation of the SVG element in the div with divId
2  const svg = d3
3    .select("#" + mergedOptions.divId)
4    .append("svg")
5    .attr("width", mergedOptions.width)
6    .attr("height", mergedOptions.height);
7  // define the transform function to project latLng coordinates into pixels
8  const pathFct = d3.geoPath(
9    d3
10     .geoMercator()
11     .fitSize([mergedOptions.width, mergedOptions.height], filteredGeoJsonMap)
12 );
13 // All the paths that represent the border of each feature
14 svg
15   .selectAll("path")
16   .data(filteredGeoJsonMap.features)
17   .join("path")
18   .attr("d", pathFct)
19   .style("fill", () => {
20     return mergedOptions.fillColorFct(counterColor++);
21   })
22   .on("mouseenter", (event, d) => {
23     const pathElement = d3.select(event.currentTarget);
24     pathElement.classed("selected", true);
25   });
```

Listing 4.6: Map in D3.js

In the final code I have defined different event listeners, such as *mouseenter*, *mousemove*, *mouseout*, *click*, etc., to make the map interactive. I have also added the canton names, special tooltips and drawn the two Ticino's lakes, the screenshot of the map is present in Figure 4.3. D3.js is a really powerful tool, and my initial idea was to use a combination of this type of maps and other ones with leaflet.js when the user actually zooms in on a municipality or in a sufficiently small surface. However in the end as the work advanced and the complexity increased I have noticed that this way of proceeding it wasn't optimal, since it would make difficult to add new GUI components and connect it to the map. Until now the frontend was written entirely in pure JavaScript, with no frameworks. However, as the implementation progressed, this approach became unsustainable. To simplify modularization and code reusability, I decided to use a JavaScript framework.

As already anticipated when discussing the tech stack the JavaScript framework for the frontend's application is Vue.js, and due to this change the map implementation switched to using a special library built for handling maps in Vue.js, vue-leaflet.

Figure 4.3.: D3.js map screenshot

4.4.3. Vue.js

Vue.js [16] is an open-source JavaScript framework specialized for building web user interfaces and Single Page Applications (SPAs). SPAs [17] are specific ways of conceiving websites, in which the browser does an intial request to the server, the server responds to it with a bare-bone html document and a JavaScript bundle, and then when the user navigate the website he doesn't request new pages to the server, instead the bundle handles everything and dynamically changes the bare-bone html. This makes the website's transitions faster and the result is a website resembling a native app.

The main advantage of using Vue.js it is the possibility to write Single-File Components, that are files with a special .vue extension and are divided in three parts: <template>, <script> and <style>. That means that the file encapsulates all together respectively, the HTML, the JavaScript and the CSS of a component. This allows the programmer to create

modular and reusable components, making the code more readable and maintainable. Vue.js offers also many additional libraries that can expand its potential, one of them is vue-leaflet [18], a library that integrates the Leaflet map library in Vue.js, simplifying the implementation of interactive maps. Vue-leaflet provides to the programmer a set of vue components that wrap the functionalities of Leaflet. To implement the map in our application we can combine four distinct vue-leaflet components:

- <LMap>
  this component represents the root component;
- <LTileLayer>
  loads tiles from a map server and display them, the used map server for the application is OpenStreetMap [19];
- <LGeoJson>
  allows to display GeoJSON features on the map.
- <LMarker>
  allows to display a single marker on the map.

By combining these four components into a single .vue file we create a map component, the <template> part is shown in Listing 4.7.

```
<template>
     <!-- This is the vue-leaflet component for the map -->
     <LMap
       ref="myMap"
       :zoom="zoom"
       :min-zoom="minZoom"
       :center="center"
       :max-bounds="maxBounds"
       :max-bounds-viscosity="maxBoundsViscosity"
       style="height: 100%; width: 100%"
       @ready="setMaxBoundAndCenter"
     >
       <LTileLayer:url="url" />
       <LGeoJson
         v-if="geoJSONFeature"
         :goejson="geoJSONFeature"
         :options="{ style:{ color: '#ff7800', weight: 5, opacity: 0.65 } }"
       />
       <LMarker
         v-for="business in businesses"
         :key="business['_id']"
         :lat-lng="[business.latitude, business.longitude]"
         :icon="
           getIcon(
             selectedBusiness && selectedBusiness['_id'] === business['_id']
           )
         "
       />
     </LMap>
</template>
```

Listing 4.7: Vue map component

A Vue.js app can be visualized as a tree of components, in which a parent and a child can communicate. The child component can receive props, e.g., in Listing 4.7 the *LGeoJson*

component has two props: *geojson* and *options*. While at the same time the child component can emit arguments to the parent component, that can catch it with the keywork @, e.g. the component *LMap* emits ready to the parent component, this communicates to it that the map has been created and the parent component can define a function that handles this event, in this case it is the function *setMaxBoundAndCenter*. It is then possible to set up the child component to be reactive, meaning that as its props changes the component state changes, the vue-leaflet components all are reactive. The resulting map component is shown in Figure 4.4.

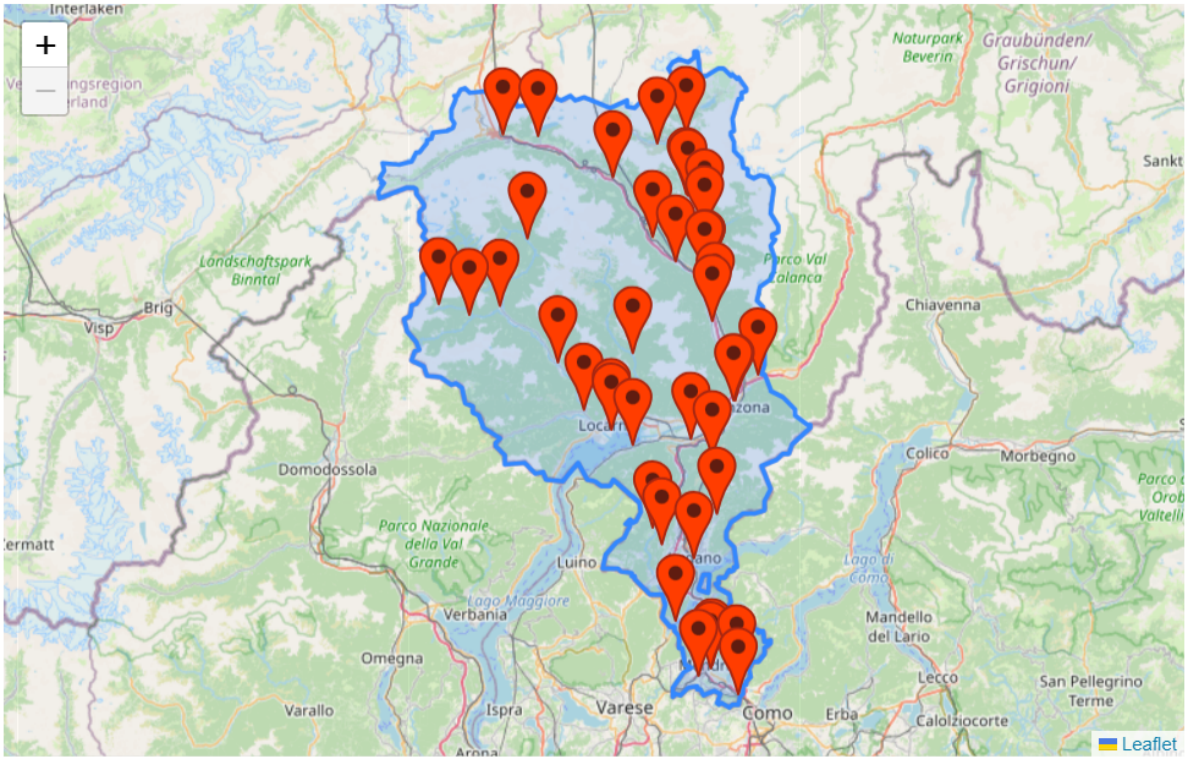

Figure 4.4.: Vue-leaflet map component

5 Conclusion

5.1. Summary

The aim of the work was to create an ecosystem that offers to users the possibility to inform themselves on businesses of Canton Ticino that are application's partners. The main goals were successfully reached, making it possible for the users to navigate a map, read and write reviews and comments, as well as keeping a list of favorite businesses. The part of client's use cases that isn't completely implemented is everything related to the authentication methods and the protection of certain routes of the API, indeed new users considering the current state of the application can be added only by the app administrator. The other goal was to make the code as modularized as possible to facilitate reusability and maintainability, and with the help of Node.js and Vue.js JavaScript frameworks that was mostly achieved.

5.2. The future

As stated above, there are some aspects of the application that can be improved and worked on in the future. Indeed the first thing to do will be to implement a way to authenticate users, and once done that also a way to distinguish between the three actors of the application: clients, businesses and administrators. The idea is to modify what the application offers in function of the actor that is logged in, therefore if the user is a business it should be restricted to write comments only on reviews about itself and be able to modify and customize the information that wants to show to the clients. An administrator, that can be viewed as a sort of app moderator, should have the power of deleting possible inappropriate reviews and comments. Therefore the main scope in the future will be to define new use cases and implement them in the most appropriate way, by considering also security measures to be sure that only authorized people can act as administrators.

A

Common Acronyms

API	Application Programming Interface
BSON	Binary JSON
CSS	Cascading Style Sheet
DBMS	Database Management System
DOM	Document Object Model
ERD	Entity-Relationship Diagram
GeoJSON	Geographic JSON
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
REST	Representational State Transfer
SQL	Structured Query Language
SPAs	Single Page Applications
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator

B License of the Documentation

C

Repository GitHub of the project

The source code and additional resources for this project are stored in a private GitHub repository. If you wish to access the repository, please send a request to sebastian.kaeslin@unifr.ch or contact me via my GitHub profile at `https://github.com/sebik2001/`.

References

[1] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (30 July 2024).

[2] TheFork, Wikipedia. https://fr.wikipedia.org/wiki/TheFork (27 July 2024).

[3] About TheFork. https://about.thefork.com/ (27 July 2024).

[4] ERD. https://www.lucidchart.com/pages/er-diagrams (20 July 2024).

[5] Modified Chen Notation. https://michael-fuchs-sql.netlify.app/2021/03/03/entity-relationship-diagram-erd/#modified-chen-notation-mc-notation (20 July 2024).

[6] Technology Stack. https://www.mongodb.com/resources/basics/technology-stack (23 July 2024).

[7] MongoDB doc. https://www.mongodb.com/docs/ (23 July 2024).

[8] MongoDB drivers. https://www.mongodb.com/docs/drivers/ (23 July 2024).

[9] Node.js doc. https://nodejs.org/docs/latest/api/documentation.html (25 July 2024).

[10] Express.js. https://expressjs.com/en/5x/api.html (25 July 2024).

[11] REST API. https://www.ibm.com/topics/rest-apis (25 July 2024).

[12] GeoJSON, Wikipedia. https://en.wikipedia.org/wiki/GeoJSON (25 July 2024).

[13] Swisstop, GeoJSON data. https://www.swisstopo.admin.ch/de/landschaftsmodell-swissboundaries3d#Weiterf%C3%BChrende-Informationen (26 July 2024).

[14] Swisstop, data information. https://backend.swisstopo.admin.ch/fileservice/sdweb-docs-prod-swisstopoch-files/files/2024/01/15/433421c4-a282-421b-b7ad-b18e6bbb8f1b.pdf (26 July 2024).

[15] D3.js. https://d3js.org/ (26 July 2024).

[16] Vue.js. https://vuejs.org/ (27 July 2024).

[17] SPAs, Wikipedia. https://en.wikipedia.org/wiki/Single-page_application (27 July 2024).

[18] Vue-leaflet. https://vue2-leaflet.netlify.app/components/ (27 July 2024).

[19] OpenStreetMap. https://www.openstreetmap.org/about (28 July 2024).