Internship Project

Université de Technologie de Compiègne

Project Title: Foodie Faces

Project Type: Web Application

Student: Andreea-Costinela Dumitru

Coordinator: Marie-Helene Abel

Academic Year 2017/2018

Contents

INTRODUCTION

Foodie Faces describes a simple tripAdvisor platform whose the main purpose is to help people in order to find a specific location, in their city, to spend great time. The application is based on community in as far as offers the chance to share the impressions and ratings between users. The most important feature gives the possibility to create an unofficial menu for every location, added by people, with real photos.

Google Maps API is integrated in application with the purpose to determine the coordinates – latitude and longitude – of every location address, so the users can find information about their favorite restaurant, check the address and the physic localization on Google Maps marked on pin-points.

The data set is crawled for three cities: Compiegne, Paris and Lyon. After selecting a city, the application shows a list with all the locations from the specific input and offers for every restaurant a lot of information about the address, phone, weekly schedule, type of cuisine etc.

After developing independently the application, the last phase describes the integration in MEMORAe platform.

TECHNOLOGIES, TOOLS AND WORKFLOW

MongoDB

“MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. It works on concept of collection and document.

MongoDB is a kind of NoSQL database. As a NoSQL database, MongoDB shuns the relational database’s table-based structure to adapt JSON-like documents that have dynamic schemas which it calls BSON. This makes data integration for certain types of applications faster and easier.”[1]

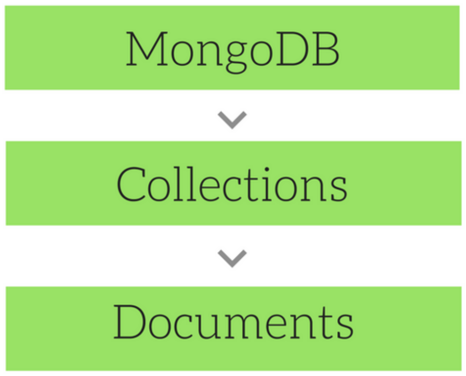


Figure 1 - MongoDB schema

Mongoose

“Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

[1] MongoDB Tutorial at https://www.studytonight.com/mongodb/introduction-to-mongodb

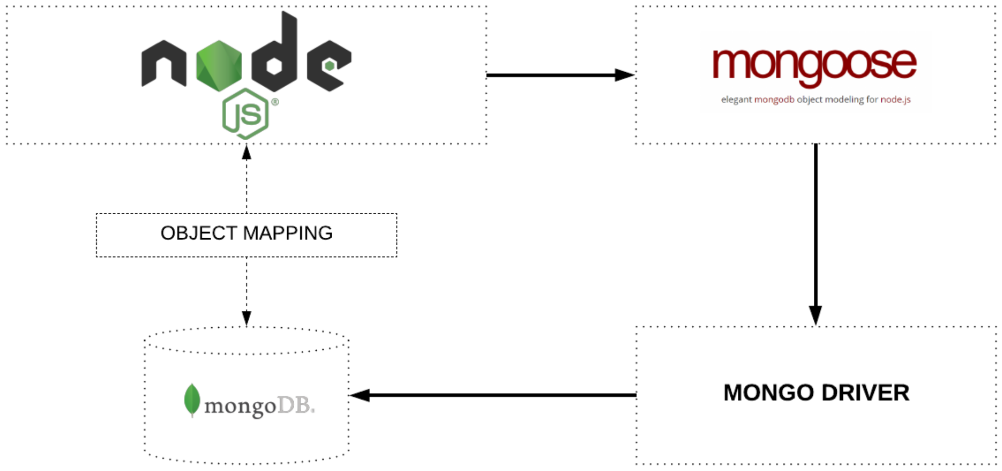


Figure 2 - Node.js and MongoDB communication

A Mongoose model is a wrapper on the Mongoose schema. A Mongoose schema defines the structure of the document, default values, validators, etc., whereas a Mongoose model provides an interface to the database for creating, querying, updating, deleting records, etc.

Terminologies:

- ‘Fields’ or attributes are similar to columns in a SQL table.
- While Mongo is schema-less, SQL defines a schema via the table definition. A Mongoose ‘schema’ is a document data structure (or shape of the document) that is enforced via the application layer.
- ‘Models’ are higher-order constructors that take a schema and create an instance of a document equivalent to records in a relational database.”[2]

Node.js

“Node.js is a JavaScript runtime built on Chrome’s V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. It’s package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

[2] Introduction to Mongoose for MongoDB at https://medium.freecodecamp.org/introduction-to-mongoose-for-mongodb-d2a7aa593c57

Node.js is a cross-platform runtime environment and has also library for running JavaScript applications outside the browser. Node is used for creating server-side and networking web applications. It is open source and free to use.

Consider a scenario where we request a backend database for the details of user1 and user2 and then print them on the screen/console. The response to this request takes time, but both of the user data requests can be carried out independently and at the same time.

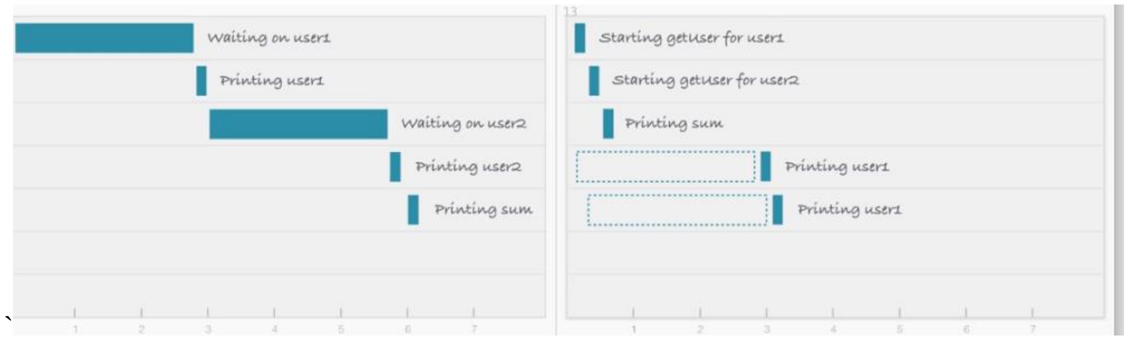


Figure 3 - A simple scenario for Node.js explanation

Blocking I/O

In the blocking method, user2's data request is not initiated until user1's data is printed to the screen.

If this was a web server, we would have to start a new thread for every new user. But JavaScript is single-threaded (not really, but it has a single-threaded event loop, which we'll discuss a bit later). So this would make JavaScript not very well suited for multi-threaded tasks.

That's where the non-blocking part comes in.

Non-blocking I/O

On the other hand using a non-blocking request, you can initiate a data request for user2 without waiting for the response the request for user1. You can initiate both requests in parallel.

This non-blocking I/O eliminates the need for multi-threading, since the server can handle multiple requests at the same time."[3]

[3] What exactly is Node.js? at https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5

Express

“Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and also mobile applications. Express.js and Node.js gave JavaScript newfound back-end functionality—allowing developers to build software with JavaScript on the server side for the first time. Together, they make it possible to build an entire site with JavaScript: You can develop server-side applications with Node.js and then publish those Node.js apps as websites with Express.”[4]

ReactJS

“React is a JavaScript library created by a collaboration of Facebook and Instagram. Its aim is to allow developers to create fast user interfaces easily. React makes no assumptions about the rest of the technology stack used, thus it’s easy to try it out on a small feature in an existing project.

It’s currently adopted in production by big companies like Facebook, Instagram, Yahoo!, Airbnb, and Sony. As I said in the introduction its popularity is growing a lot, so I expect more companies to employ it very soon. In my opinion, the community hasn’t been this excited about a JavaScript tool/library since the introduction of Node.js.

React isn’t a complete framework. It doesn’t offer all the components you’ll find in projects like Ember or AngularJS. In fact, many refer to React as just the V in MVC.”[5]

Material UI

“In a nutshell, Material-UI is an open-source project that features React components that implement Google’s Material Design. Material-UI is not only an implementation of the Material Design guidelines, but a general use UI library of components that are needed by many.”[6]

[4] Express/Node introduction at https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction
[5] Introduction to the React JavaScript Library at https://developer.telerik.com/featured/introduction-to-the-react-javascript-framework/
[6] Meet Material-UI—your new favorite user interface library at https://medium.freecodecamp.org/meet-your-material-ui-your-new-favorite-user-interface-library-6349a1c88a8c

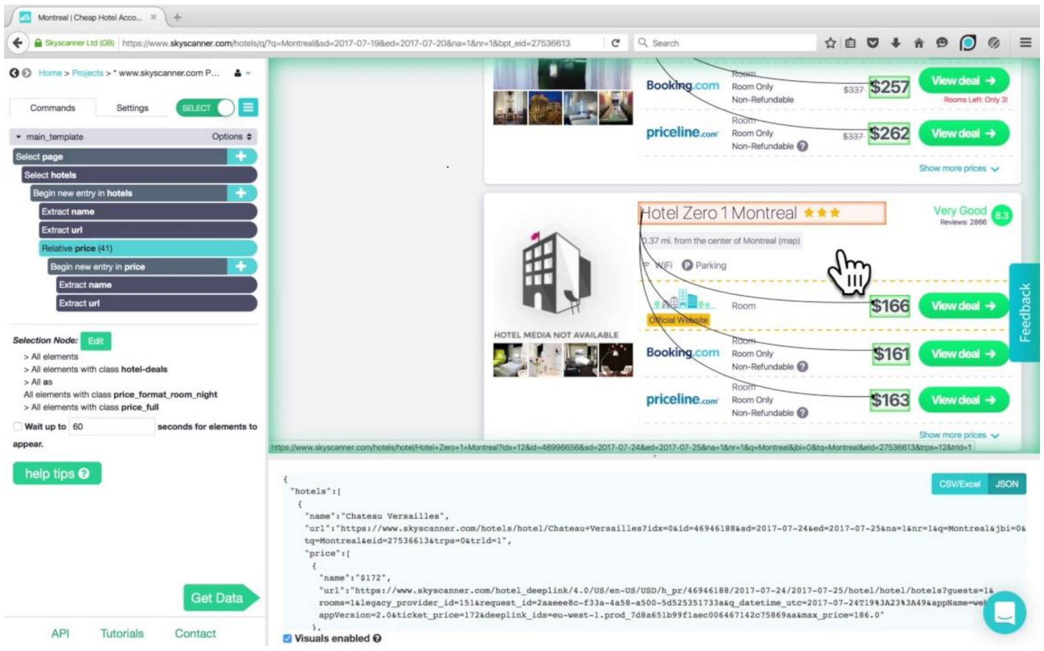
ParseHub

“Anyone should be able to pull data from the web and access it in the format they want. If a website does not have an API available, scraping is one of the only options to get the data you need. But figuring out how to scrape data in the complicated HTML is a pain.

ParseHub is a new web browser extension that you can use to turn any dynamic and poorly structured website into an API, without writing code. ParseHub is a scraping tool that is designed to work on websites with JavaScript and Ajax; it is similar to web scraping tools such as Import.io and Kimono Labs.

The ParseHub tool will identify relationships between elements, extract all of the data and provide it in a spreadsheet or easily accessible API for you. Both scrapers and data are cloud hosted.

For developers, ParseHub gives you full control over how you select, structure and modify elements, so you don’t have to hunt through your browser’s web inspector. You can use ParseHub to log into websites, automatically fill out forms, loop through search queries, click on interactive maps, handle infinite scrolling, drop downs and popups. Regex is built in as the handy tool to parse specific text from a webpage. Schedule to update and retrieve your data every minute if you wish, and instantly see the results of your scrape as you build your project.”[7]



- Select **page**
  - Select **hotels**
    - Begin new entry in **hotels**
      - Extract **name**
      - Extract **url**
      - Relative **price** (41)
        - Begin new entry in **price**
          - Extract **name**
          - Extract **url**

Selection Node:
> All elements
> All elements with class **hotel-deals**
> All as
All elements with class **price_format_room_night**
> All elements with class **price_full**

Wait up to 60 **seconds for elements to appear.**

https://www.skyscanner.com/hotels/hotel/Hotel+Zero+1+Montreal?idx=12&id=46996656&sd=2017-07-24&ed=2017-07-25&na=1&nr=1&q=Montreal&jbi=0&tq=Montreal&eid=27536613&trps=12&trld=1

```
{
 "hotels":[
  {
   "name":"Chateau Versailles",
   "url":"https://www.skyscanner.com/hotels/hotel/Chateau+Versailles?idx=0&id=46946188&sd=2017-07-24&ed=2017-07-25&na=1&nr=1&q=Montreal&jbi=0&
   tq=Montreal&eid=27536613&trps=0&trld=1",
   "price":[
    {
     "name":"$172",
     "url":"https://www.skyscanner.com/hotel_deeplink/4.0/US/en-US/USD/h_pr/46946188/2017-07-24/2017-07-25/hotel/hotel/hotels?guests=1&
     rooms=1&legacy_provider_id=151&request_id=2aaeee8c-f33a-4a58-a500-5d525351733a&q_datetime_utc=2017-07-24T19%3A23%3A49&appName=we
     appVersion=2.0&ticket_price=172&deeplink_ids=eu-west-1.prod_7d8a651b99f1aec006467142c75869aa&max_price=186.0"
    },
```

Visuals enabled

Figure 4 - How to use ParseHub

[7] Turn any interactive website into an API with ParseHub at http://scraping.pro/turn-any-interactive-website-into-api-with-parsehub/

MERN Stack

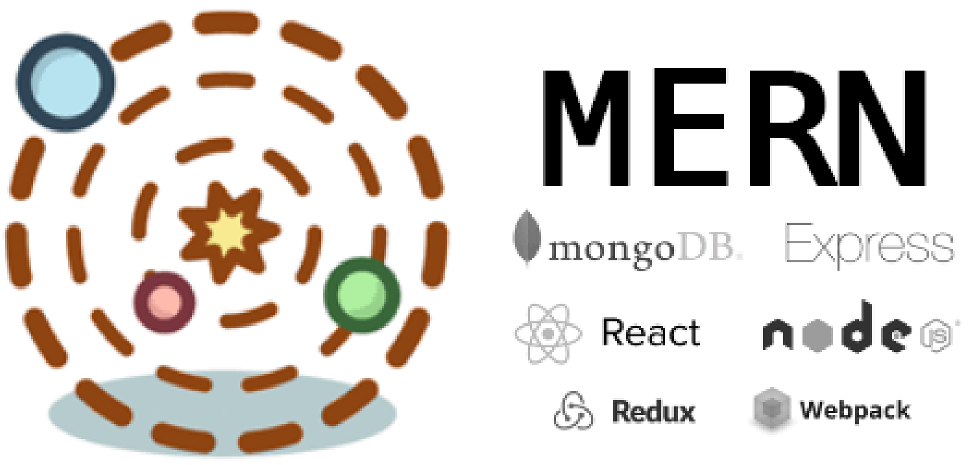


Figure 5 - MERN stack

“MERN stack is basically a collection of JavaScript-based web development technologies that includes **MongoDB**, **ExpressJS**, **ReactJS** and **NodeJS**.

These technologies work together to develop a web application. ReactJS being client makes AJAX calls to ExpressJS returning response in JSON format. ExpressJS that is running on NodeJS Server further communicate with MongoDB as persistent medium (a NoSQL database).

MERN represents a group of technologies which are known to synergize well together. The major benefit of the MERN stack is that it’s extremely quick to prototype with. Node.js allows you to use Javascript on the backend as well as the frontend which can save you from having to learn a separate language. In addition, the NoSQL nature of MongoDB allows you to quickly change and alter the data layer without having to worry about migrations, which is a very valuable attribute when you’re trying to build a product without clear specifications.”[8]

[8] Mean Stack Development[For Developers] at https://hackernoon.com/mean-stack-development-for-developers-4d88c40c4103

Cloudinary

“Cloudinary is a cloud-based service that provides an end-to-end image management solution, including upload, storage, administration, manipulation, optimization and delivery.

With Cloudinary you can easily upload images to the cloud and automatically perform smart image manipulations without installing any complex software. Cloudinary provides a secure and comprehensive API for easily uploading images from server-side code, directly from the browser or from a mobile application. You can either use Cloudinary's API directly or through one of Cloudinary's client libraries (SDKs), which wrap the upload API and simplify integration with web sites and mobile applications.

The uploaded images can then be automatically converted to all relevant formats suitable for web viewing, optimized for web browsers and mobile devices, normalized, manipulated in real time, and delivered through a fast CDN to users.

While uploading images you can also apply transformations (e.g. changing dimensions or format) and assign tags to them, and you can manage all uploaded images using Cloudinary's Media Library web interface and the Admin API.”[9]

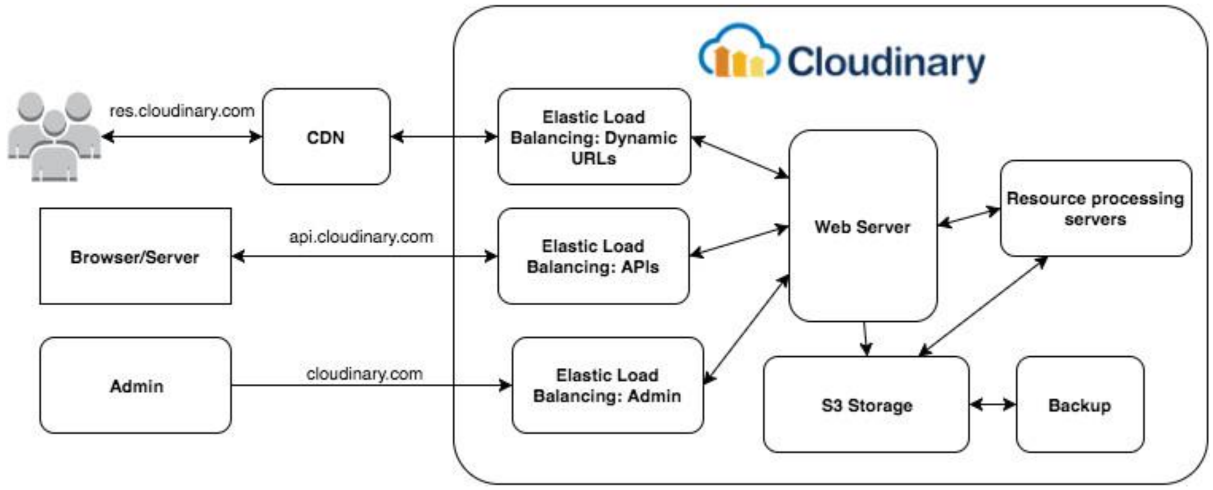

Cloudinary
res.cloudinary.com
CDN
Elastic Load Balancing: Dynamic URLs
Browser/Server
api.cloudinary.com
Elastic Load Balancing: APIs
Web Server
Resource processing servers
Admin
cloudinary.com
Elastic Load Balancing: Admin
S3 Storage
Backup

Figure 6 - Cloudinary schema

[9] Upload images at https://cloudinary.com/documentation/upload_images

REQUIREMENTS

In big lines, the following features are required for the application in order to reach the exposed goal:

- Find a location by city

The user can find information about a location, searching by his city. The available cities are: Paris, Lyon and Compiegne and there are crawled.

- Show locations on Google Maps

After choosing a city, the application show a map with pin-points for every location found in that city. Hovering a location, its marker highlits on map in order to help user to see the restaurant position.

- Filter the restaurants based on what user want to eat

This represents that the user can select a filter to choose what kind of place he wants to find (bar, restaurant, coffee shop etc.). More than that, he can have access at every special list of cheap, Italian/French/Spanish/etc., vegetarian, vegan restaurants, bars with special atmosphere etc.

- Review and rating system – made by a user

The application offers a special section for rating and reviews. An user can leave comments and scores for a place that they have seen.

- An unofficial menu based on community

After leaving a review, the application suggests to add a dish in order to help the community. Every user can add pictures, specify a name, a category and a price for a dish that he has eaten before at a specific location. After three users who suggest the same kind of food, the dish is added in a permanent menu with a price, represented by the average of the last three inputs.

- Recommendation system

The application should acknowledge every user actions and use their behavior to recommend specific data. In location details view, the user can receive a recommendation about the other restaurant that are similar (based on filters – for example: restaurants with pizza) or with the same specific dishes.

- Most recommended dishes

This feature implies that a user can have the possibility to recommend a kind of dish, that he liked it, from the permanent menu, in order to help the users to find the perfect location for them.

- Wish list

A wish list represents a list for every user which want to keep a track of restaurant that he wants to visit in the near future.

APPLICATION DEVELOPMENT

Database architecture

Using a non relational database, the standard tables became collections of documents while the links between them were implemented after the following rules:

„What is the cardinality of the relationship: is it “one-to-few”, “one-to-many”, or “one-to-squillions”?

Do you need to access the object on the “N” side separately, or only in the context of the parent object?

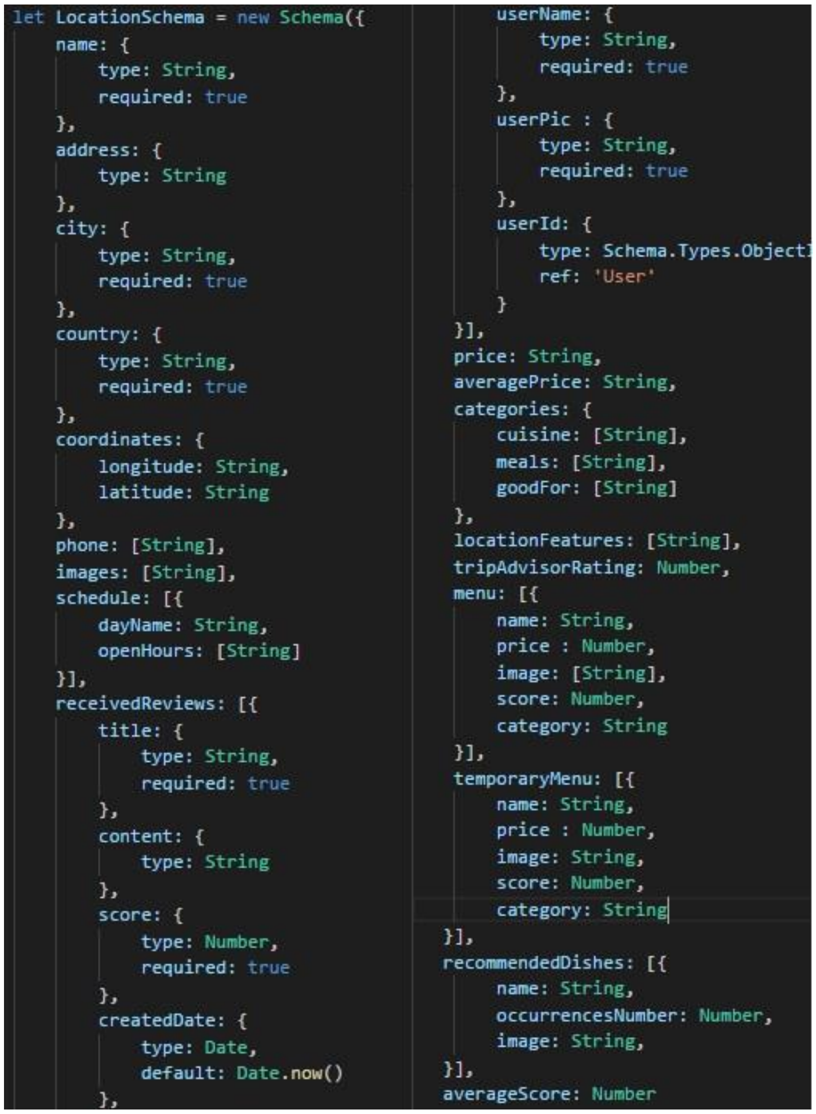
What is the ratio of updates to reads for a particular field?

The main choices for structuring the data are:

- For “one-to-few”, you can use an array of embedded documents.
- For “one-to-many”, or on occasions when the “N” side must stand alone, you should use an array of references. You can also use a “parent-reference” on the “N” side if it optimizes your data access pattern.
- For “one-to-squillions”, you should use a “parent-reference” in the document storing the “N” side.”[10]

[10] 6 Rules of Thumb for MongoDB Schema Design at https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1

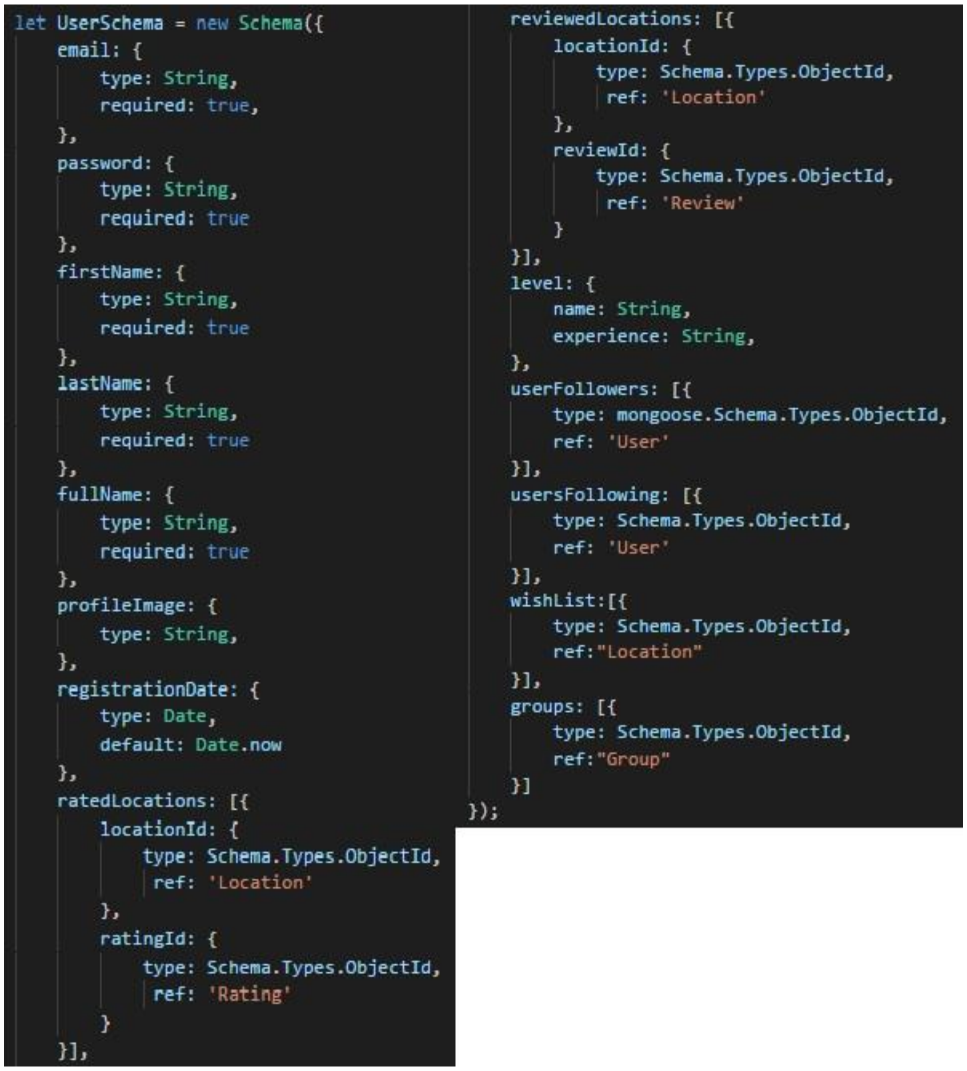
LocationSchema



```
let LocationSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    address: {
        type: String
    },
    city: {
        type: String,
        required: true
    },
    country: {
        type: String,
        required: true
    },
    coordinates: {
        longitude: String,
        latitude: String
    },
    phone: [String],
    images: [String],
    schedule: [{
        dayName: String,
        openHours: [String]
    }],
    receivedReviews: [{
        title: {
            type: String,
            required: true
        },
        content: {
            type: String
        },
        score: {
            type: Number,
            required: true
        },
        createdDate: {
            type: Date,
            default: Date.now()
        },
        userName: {
            type: String,
            required: true
        },
        userPic : {
            type: String,
            required: true
        },
        userId: {
            type: Schema.Types.ObjectI
            ref: 'User'
        }
    }],
    price: String,
    averagePrice: String,
    categories: {
        cuisine: [String],
        meals: [String],
        goodFor: [String]
    },
    locationFeatures: [String],
    tripAdvisorRating: Number,
    menu: [{
        name: String,
        price : Number,
        image: [String],
        score: Number,
        category: String
    }],
    temporaryMenu: [{
        name: String,
        price : Number,
        image: String,
        score: Number,
        category: String
    }],
    recommendedDishes: [{
        name: String,
        occurrencesNumber: Number,
        image: String,
    }],
    averageScore: Number
```

Figure 7 - Location Model

UserSchema



```
let UserSchema = new Schema({
    email: {
        type: String,
        required: true,
    },
    password: {
        type: String,
        required: true
    },
    firstName: {
        type: String,
        required: true
    },
    lastName: {
        type: String,
        required: true
    },
    fullName: {
        type: String,
        required: true
    },
    profileImage: {
        type: String,
    },
    registrationDate: {
        type: Date,
        default: Date.now
    },
    ratedLocations: [{
        locationId: {
            type: Schema.Types.ObjectId,
            ref: 'Location'
        },
        ratingId: {
            type: Schema.Types.ObjectId,
            ref: 'Rating'
        }
    }],
    reviewedLocations: [{
        locationId: {
            type: Schema.Types.ObjectId,
            ref: 'Location'
        },
        reviewId: {
            type: Schema.Types.ObjectId,
            ref: 'Review'
        }
    }],
    level: {
        name: String,
        experience: String,
    },
    userFollowers: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    }],
    usersFollowing: [{
        type: Schema.Types.ObjectId,
        ref: 'User'
    }],
    wishList:[{
        type: Schema.Types.ObjectId,
        ref:"Location"
    }],
    groups: [{
        type: Schema.Types.ObjectId,
        ref:"Group"
    }]
});
```

Figure 8 - User Model

Application architecture

Create React App is a quick way to get started with React development and it requires no build configuration. But it completely hides the build config which makes it difficult to extend. It also requires some additional work to integrate it with an existing Node.js/Express backend application.

In the development mode, 2 servers are running concurrently. The front end code will be served by the webpack dev server which helps with hot and live reloading. The server side Express code will be served by a node server.

In the production mode, only 1 server will run. All the client side code will be bundled into static files using webpack and it will be served by the Node.js/Express application.

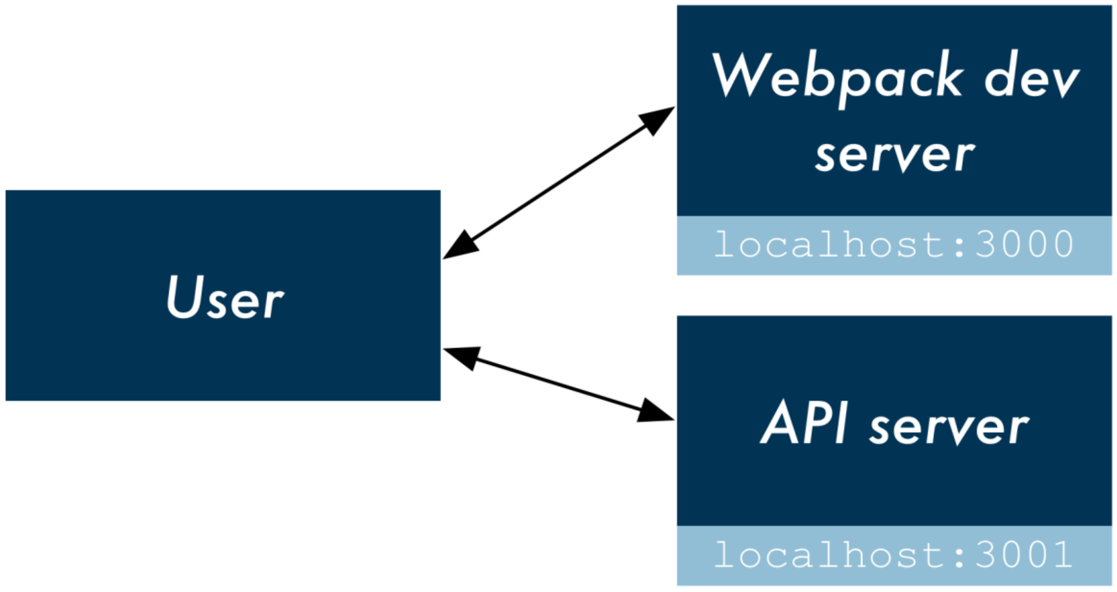


Figure 9 - Application Server Concurrency 1

In this flow, the user's browser makes a request to localhost:3000, loading the static assets from the Webpack dev server. The user's browser / React then makes requests as needed directly to the API server hosted on localhost:3001

This would produce an issue, however. The React app (hosted at localhost:3000) would be attempting to load a resource from a different origin (localhost:3001). This would be performing Cross-Origin Resource Sharing. The browser prevents these types of requests from scripts for security reasons.

create-react-app provides a mechanism for working with an API server in development. We can have the Webpack development server proxy requests intended for our API server, like this:

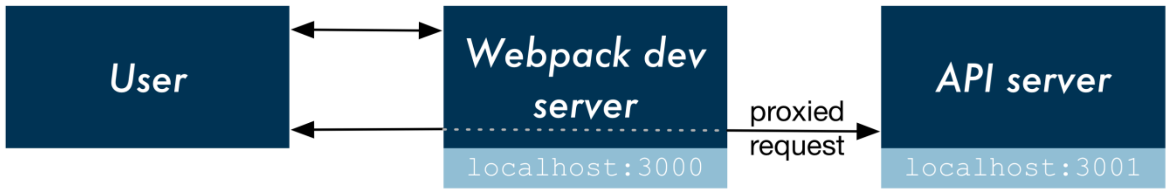


Figure 10 - Application Server Concurrency 2

Interface wireframes

- Restaurants list

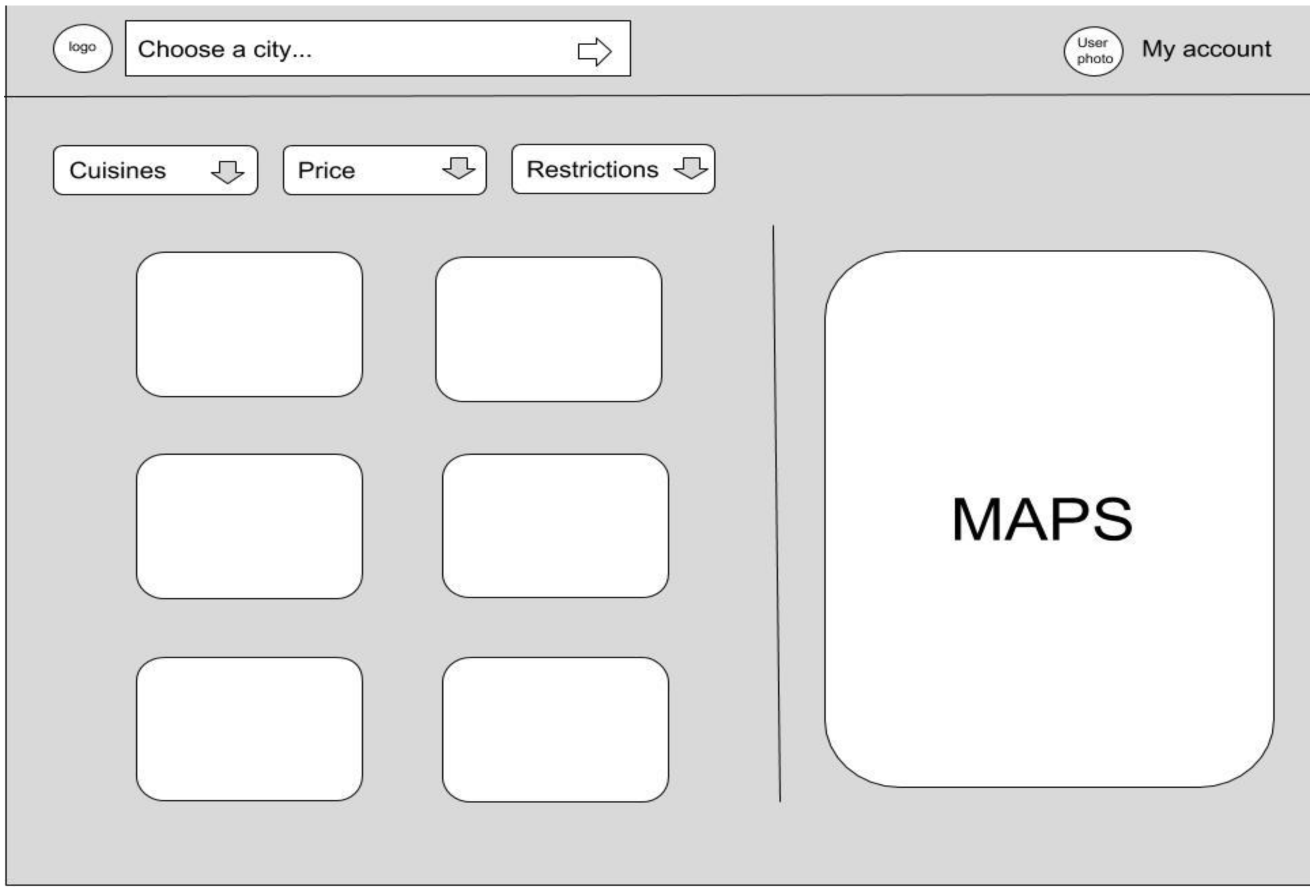


Figure 11 - Restaurants list wireframe

- Location details

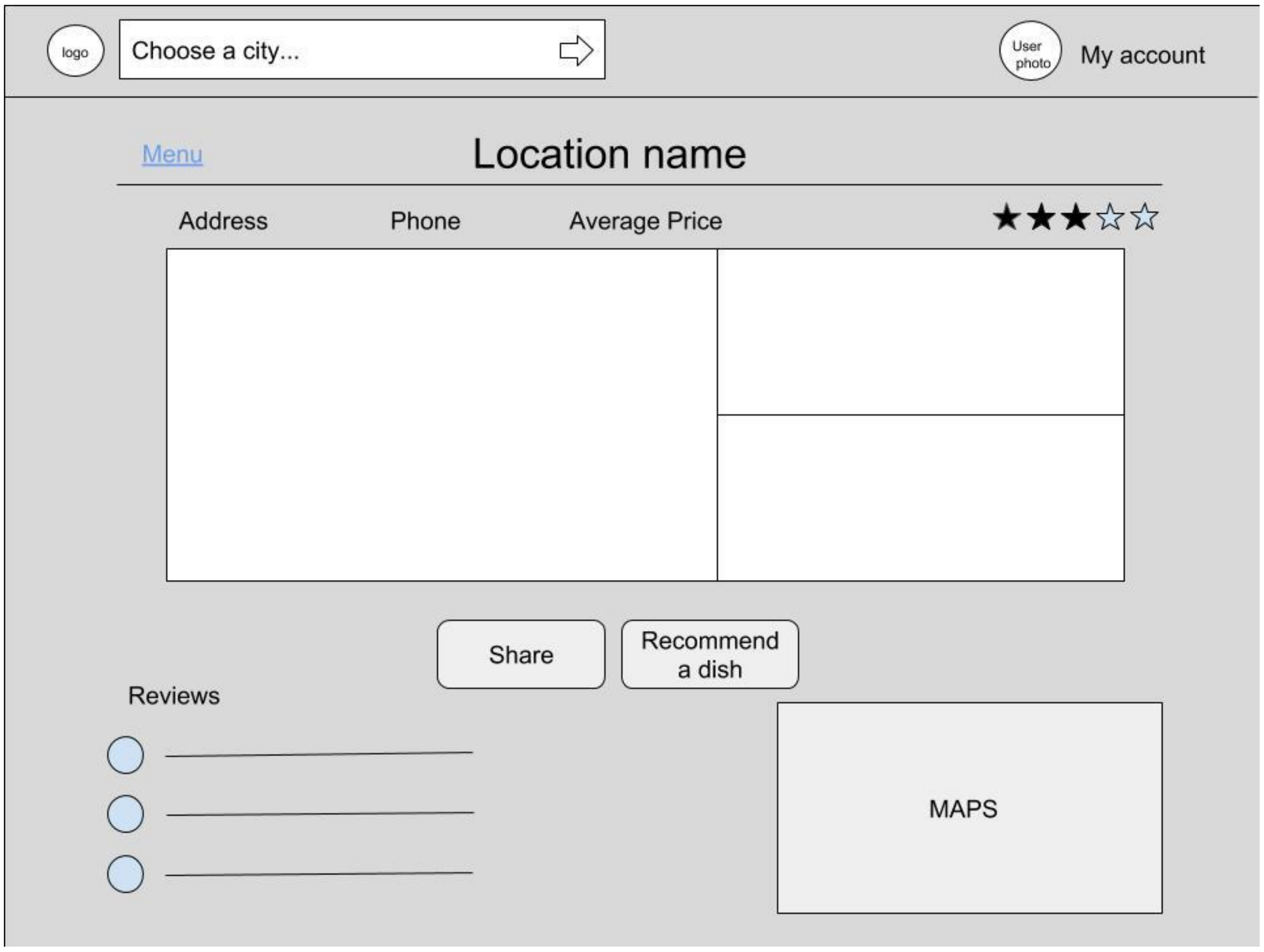


Figure 12 - Location Details wireframe

- Add a dish in menu

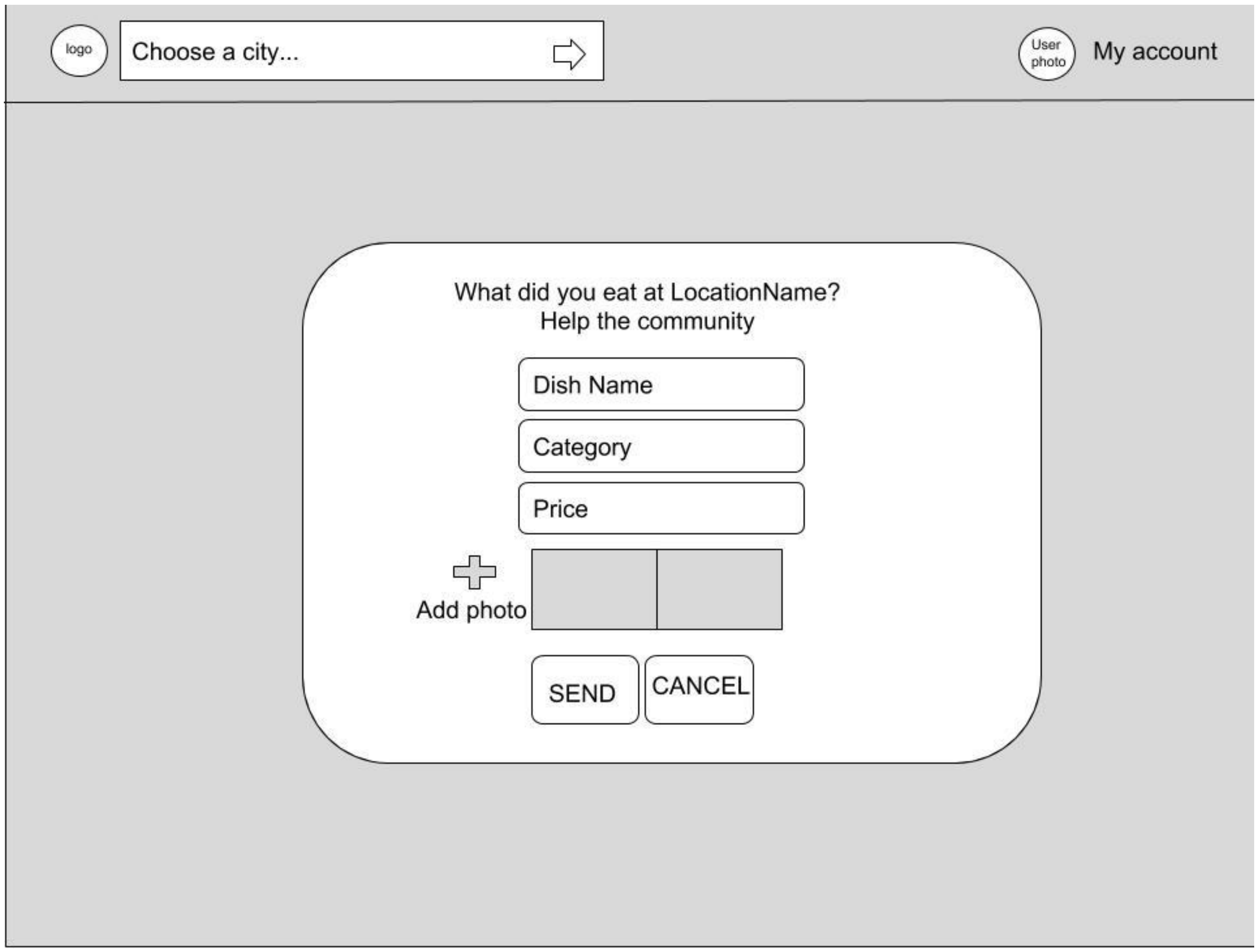


Figure 13 - Add a dish in menu wireframe

- Wish list

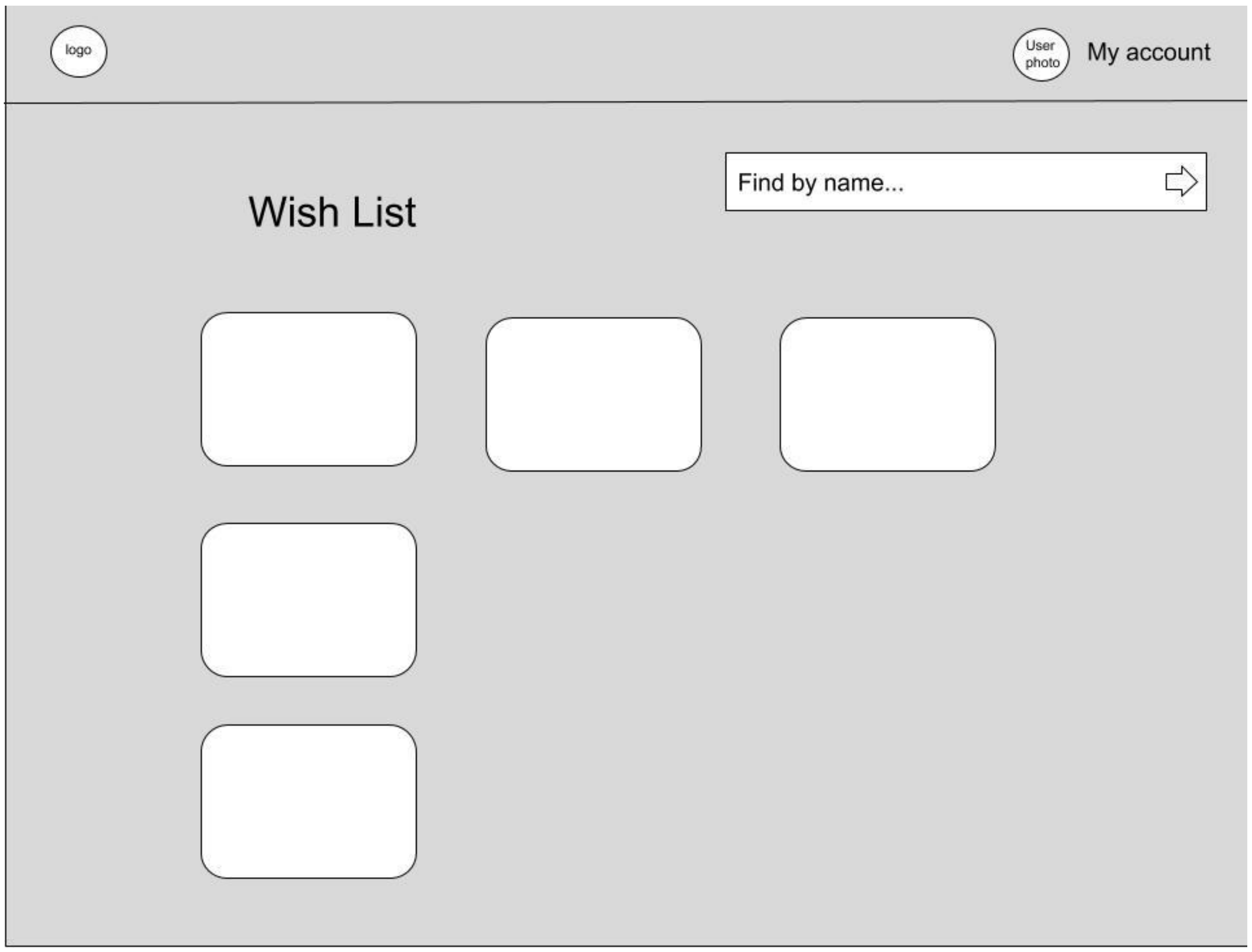


Figure 14 - Wish list wireframe

Use cases

- Login

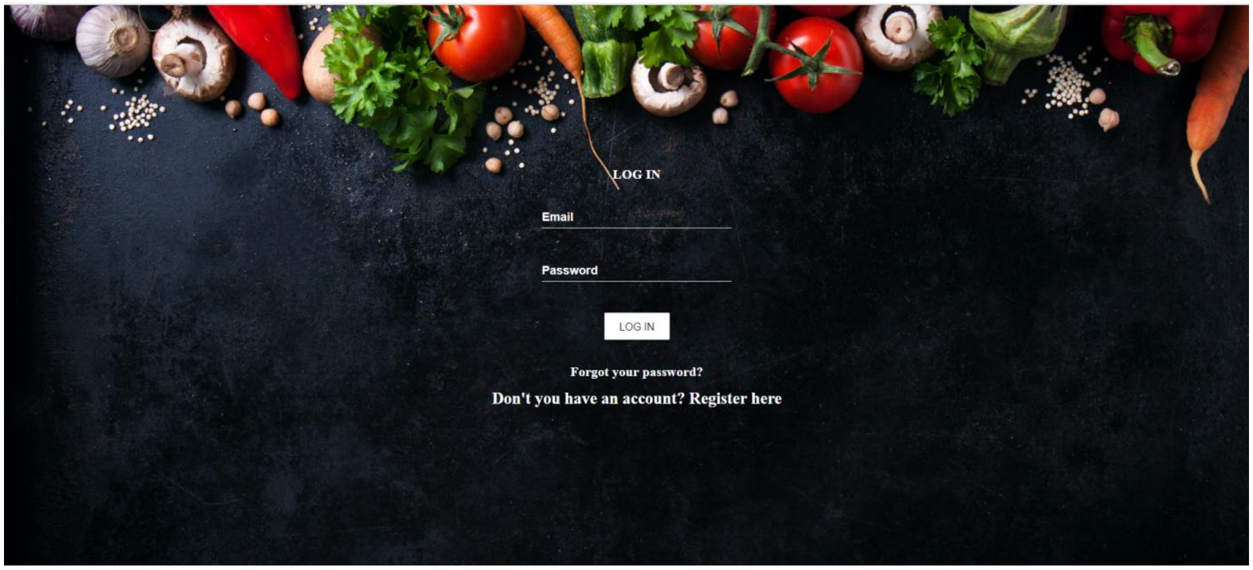


Figure 15 - Login

- Find locations by city – choose a city from the header search bar

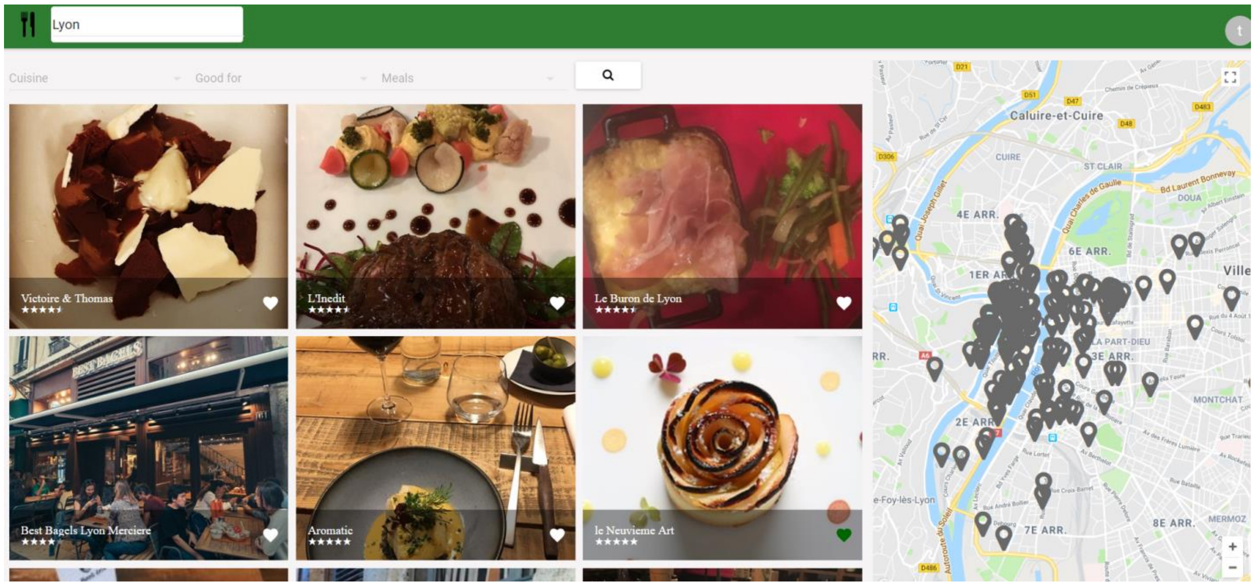


Figure 16 - Find locations by city

- Hover a location, in order to find its marker on map

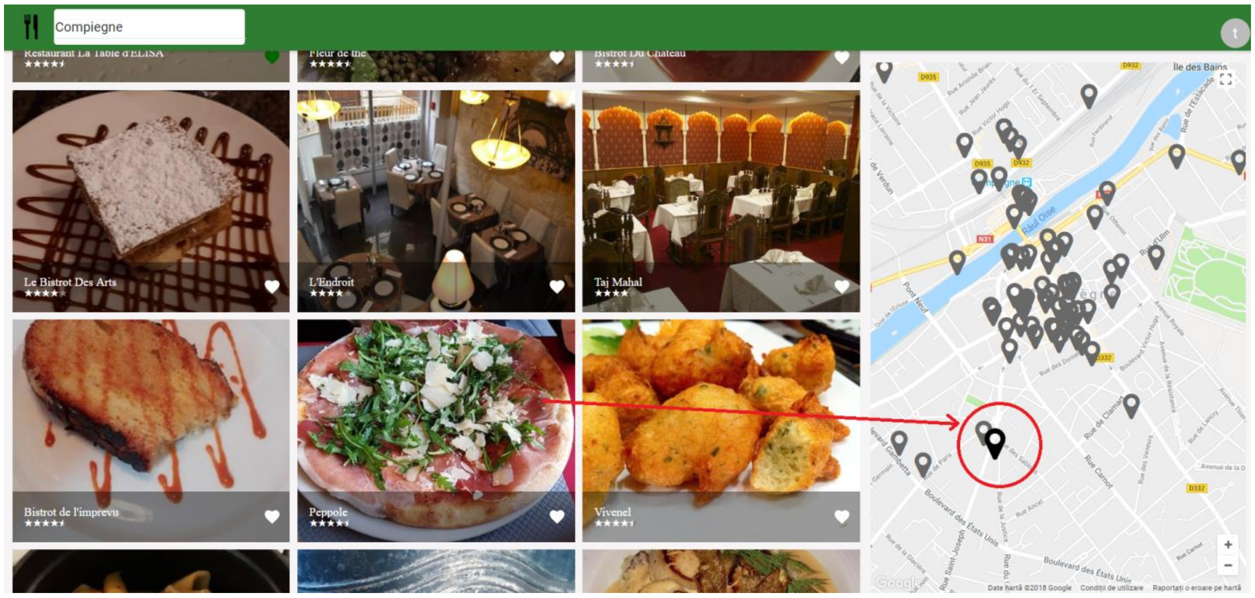
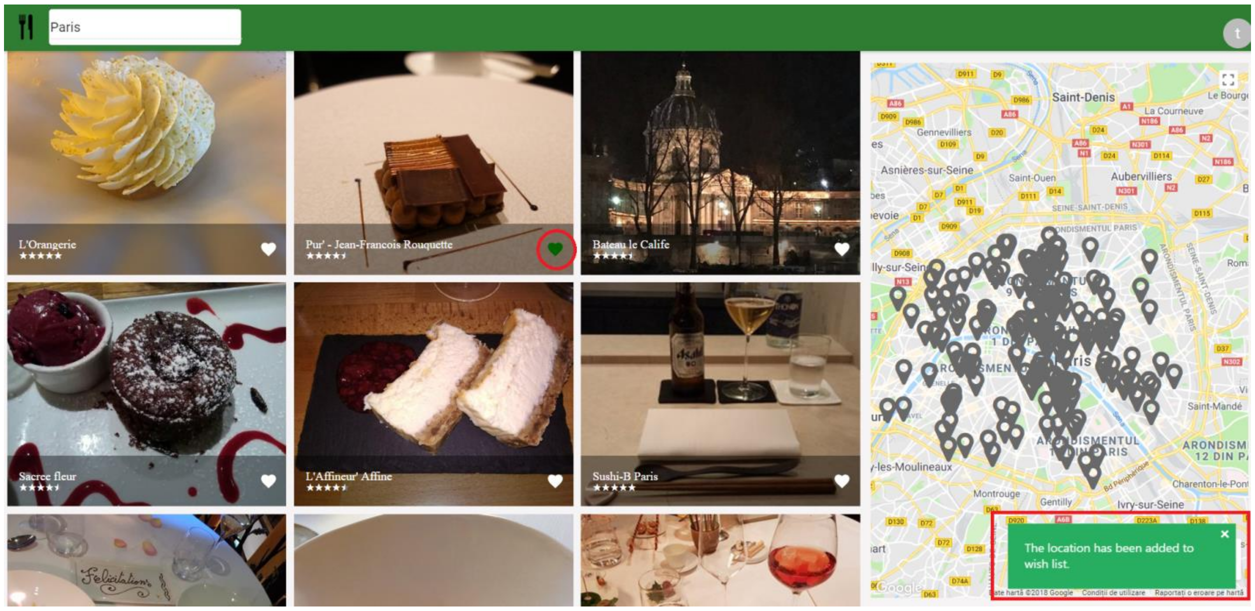


Figure 17 - Find marker on map

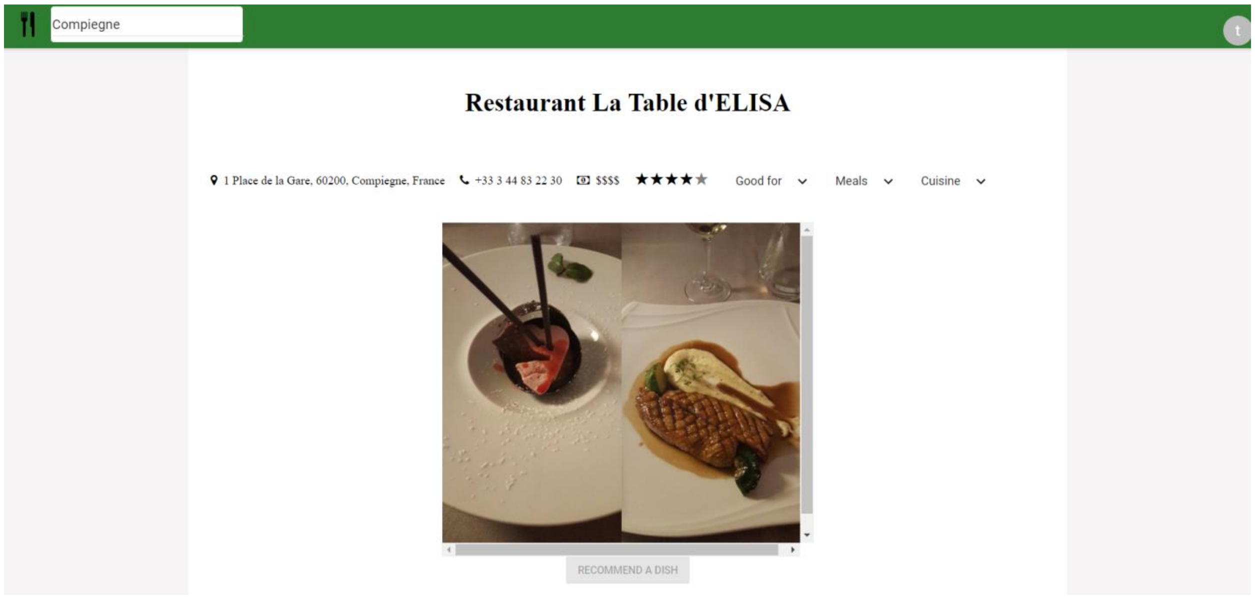
- Add to Wish list



The location has been added to wish list.

Figure 18 - Add to Wish list

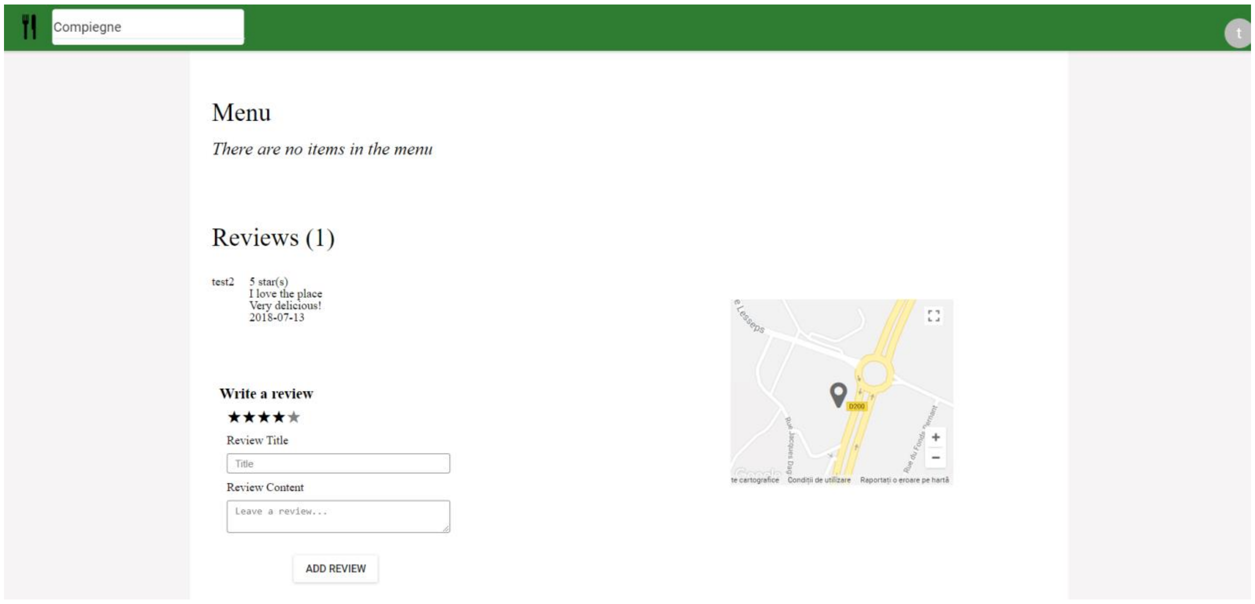
- Go to location details


Compiegne

Restaurant La Table d'ELISA

1 Place de la Gare, 60200, Compiegne, France · +33 3 44 83 22 30 · $$$$ · ★★★★★

Figure 19 - Location details 1


Compiegne

Menu

There are no items in the menu

Reviews (1)

test2 5 star(s)
I love the place
Very delicious!
2018-07-13

Write a review
★★★★★
Review Title
Title
Review Content
Leave a review...

Figure 20 - Location details 2

- Recommendation system

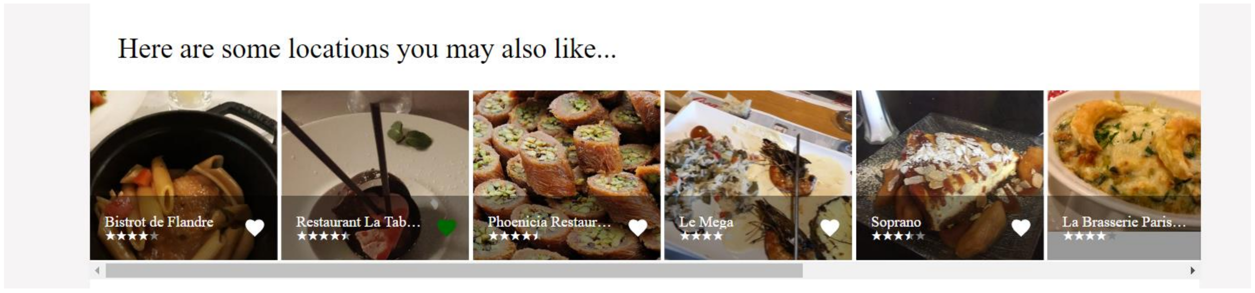


Figure 21 - Recommendation system

- Menu and most recommended dishes

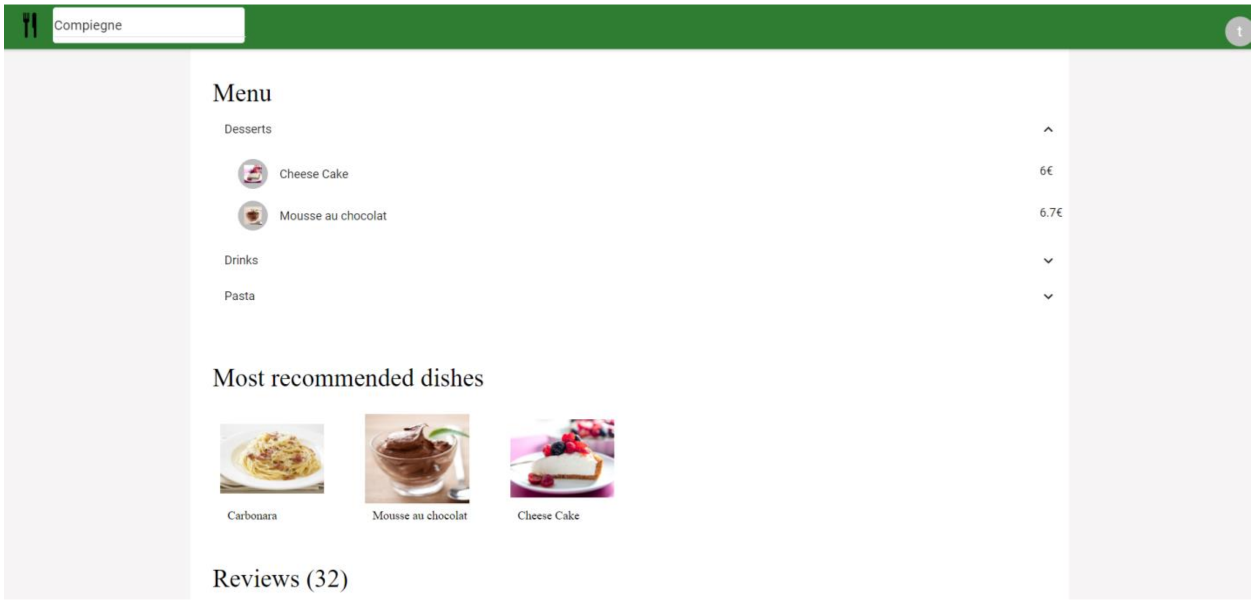


Figure 22 - Menu and most recommended dishes

- Add a dish in menu

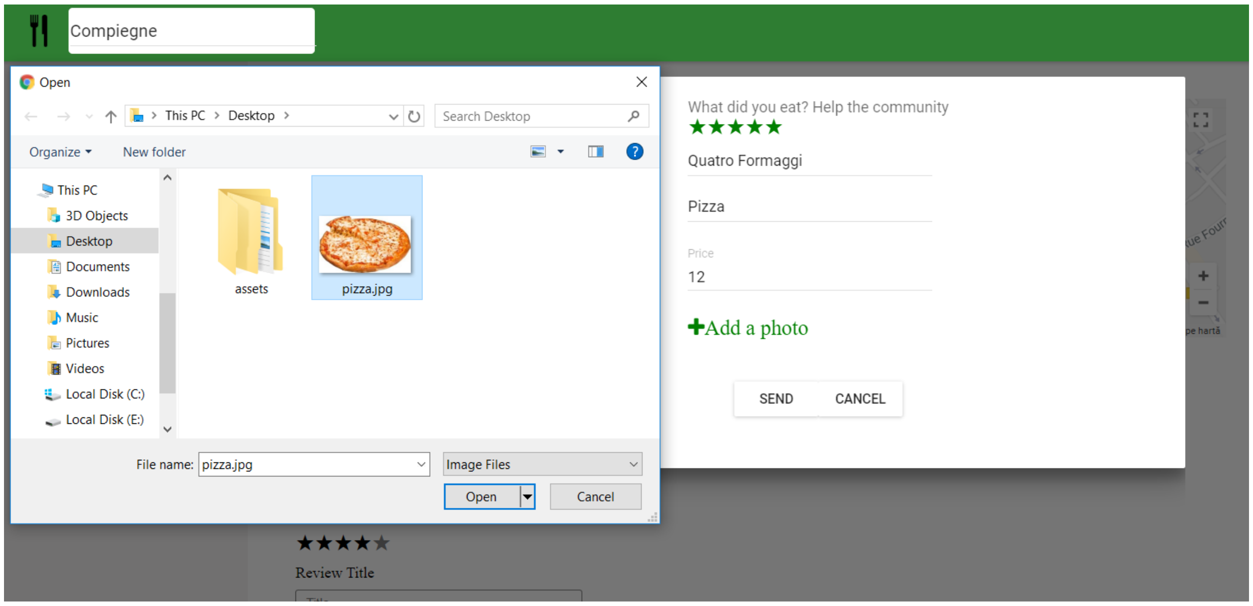


Figure 23 - Add a dish in menu 1

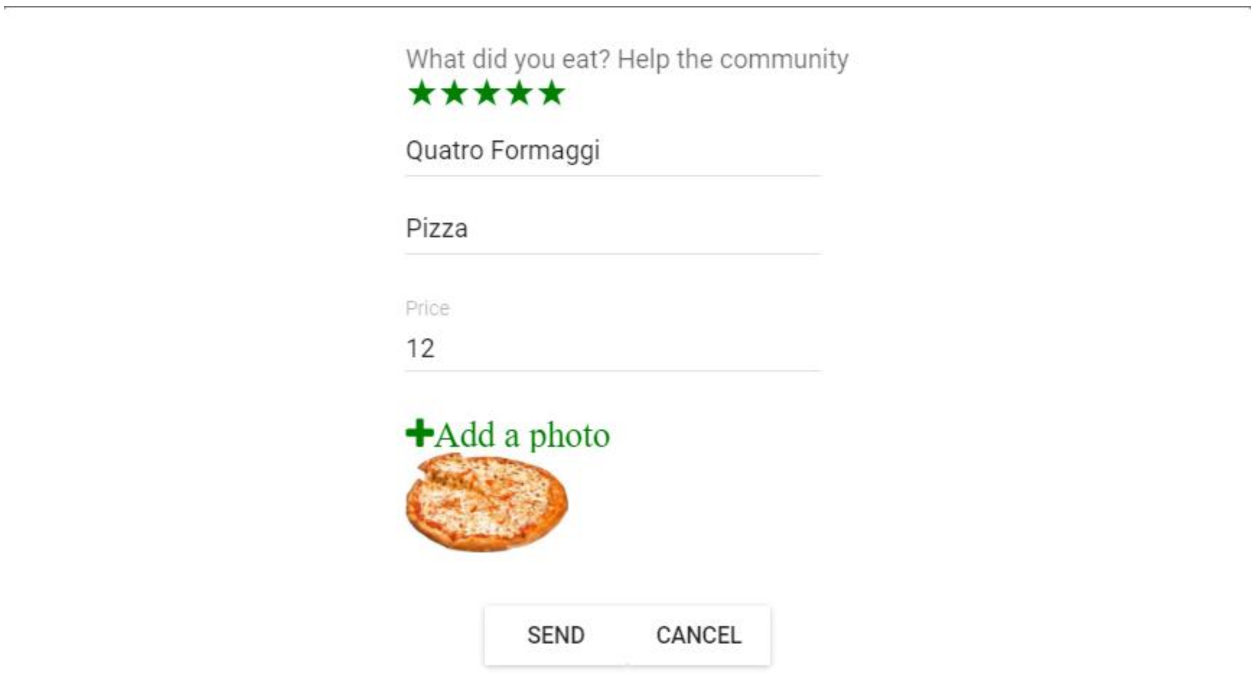


Figure 24 - Add a dish in menu 2

- Recommend a dish from menu

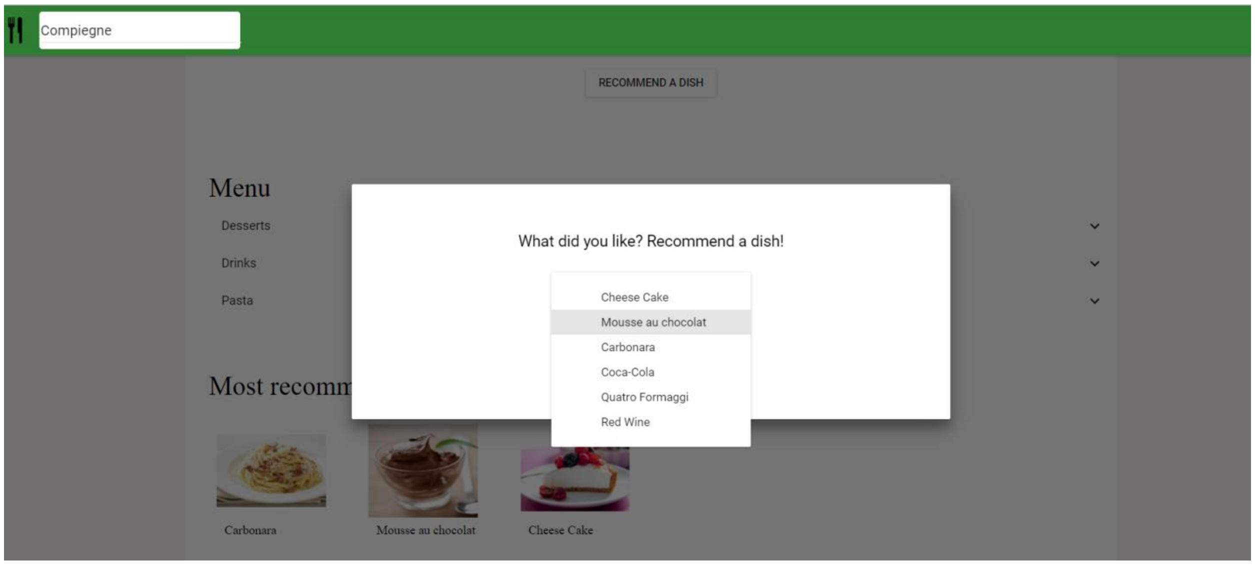


Figure 25 - Recommend a dish from menu

Testing

This phase includes testing and bug fixing. Although this is presented like a separated phase, it was a continuous process.

MEMORAe

"With the MEMORAe approach, we wanted to model and design a web platform to manage all the heterogeneous knowledge resources circulating in an organization. The platform, of the same name as the approach, was designed and developed to facilitate organizational learning and the capitalization of knowledge from semantic modeling. It exploits the power of new technologies supporting collaboration (web 2.0 technologies, touch tables, etc.) and relies on semantic web standards. The heart of innovation is the organization around a knowledge map of all the private or shared resources, resulting from a formal or informal process within a group of individuals (team, service, project, organization, etc.).

The use of a semantic map makes it possible to define a common repository in which it is possible to navigate to access the resources capitalized in different spaces. These spaces are visible in parallel and facilitate the transfer of knowledge between individuals.

General Principles:

- Integrated platform - One of the strengths of the MEMORAe platform is its complete integration of all the functionalities needed to host a collaboration and knowledge capitalization server. It includes all its features within the same application: it is not an association of different software offering the desired features.
- Knowledge card - The knowledge map is the graphical representation of the definition of the shared referential of the chosen organization. You can navigate on this map and select a focus node (click on a node). This node then takes the status of active index, the one that is considered to make visible the resources in the sharing spaces. By passing the cursor on link, we can see the nature / description of the relationship, it is the same for the nodes.
- Sharing spaces - A sharing space makes it possible to make visible a set of resources shared by the members of this space. The same resource can be visible in different spaces, but it remains stored in a single place. Each user has a sharing space accessible only by himself (private). Visualization of shared spaces in parallel facilitates the transfer of resources from one space to another and therefore sharing: drag and drop.
- User box - The user box appears on the right at the top of the interface. It concerns the user-specific information. In the first place, the user is asked to choose the organization for which he wishes to use the collaboration platform. Once selected, the associated knowledge map of the chosen organization is displayed.
- Organization of resources - The organization of the resources is done:

    - around the common repository and allows different access to resources according to their semantic description, any resource can be indexed by its content (the concepts it processes) or its container (author, date of creation, etc.)
    - according to the access rights of the user on sharing spaces for viewing.
- Resources –
    - Modeling of documentary resources and / or from a social process (pdf file, semantic forum, semantic chat, semantic wiki, etc.).
    - Human resource modeling: who knows what?
    - Annotation of resources, parts of resources or concepts (components of the common repository); The annotation itself is considered a resource. It is therefore accessible in the sharing spaces as a resource or via the annotated resource."[11]

Web Ontology Language

"The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies. Ontologies are a formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains: the nouns representing classes of objects and the verbs representing relations between the objects. Ontologies resemble class hierarchies in object-oriented programming but there are several critical differences. Class hierarchies are meant to represent structures used in source code that evolve fairly slowly (typically monthly revisions) whereas ontologies are meant to represent information on the Internet and are expected to be evolving almost constantly. Similarly, ontologies are typically far more flexible as they are meant to represent information on the Internet coming from all sorts of heterogeneous data sources. Class hierarchies on the other hand are meant to be fairly static and rely on far less diverse and more structured sources of data such as corporate databases.

The OWL languages are characterized by formal semantics. They are built upon the World Wide Web Consortium's (W3C) XML standard for objects called the Resource Description Framework (RDF). OWL and RDF have attracted significant academic, medical and commercial interest."[12]

[11] Qu'est ce que MEMORAe? at http://memorae.hds.utc.fr/
[12] Web Ontology Language at https://en.wikipedia.org/wiki/Web_Ontology_Language

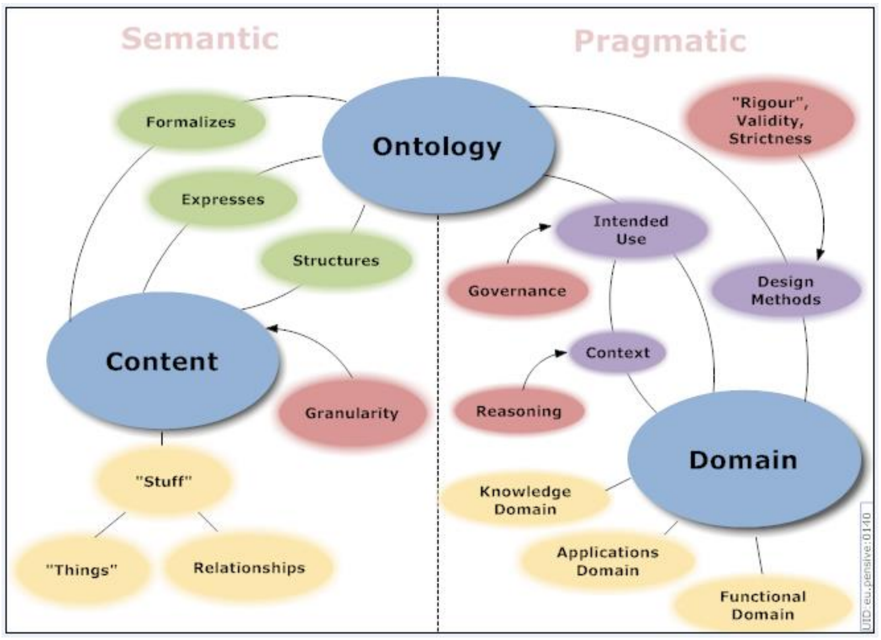


Figure 26 - Web ontology schema

Protégé

“Protégé is a free, open source ontology editor and a knowledge management system. Protégé provides a graphic user interface to define ontologies. It also includes deductive classifiers to validate that models are consistent and to infer new information based on the analysis of an ontology. Like Eclipse, Protégé is a framework for which various other projects suggest plugins. This application is written in Java and heavily uses Swing to create the user interface. Protégé recently has over 300,000 registered users. According to a 2009 book it is "the leading ontological engineering tool".

Protégé is being developed at Stanford University and is made available under the BSD 2-clause license. Earlier versions of the tool were developed in collaboration with the University of Manchester.”[13]

[13] Protégé (software) at https://en.wikipedia.org/wiki/Prot%C3%A9g%C3%A9_(software)

Integration

This phase is where the integration with MEMORAe platform has been made.

In the first phase, the ontology of the application was required as this was to be stored in the MEMORAe database knowledge. The ontology can be visualized below. The representation is developed for both Cooldown Cooking and also for a trip-advisor like application, as both of them had to be integrated together.

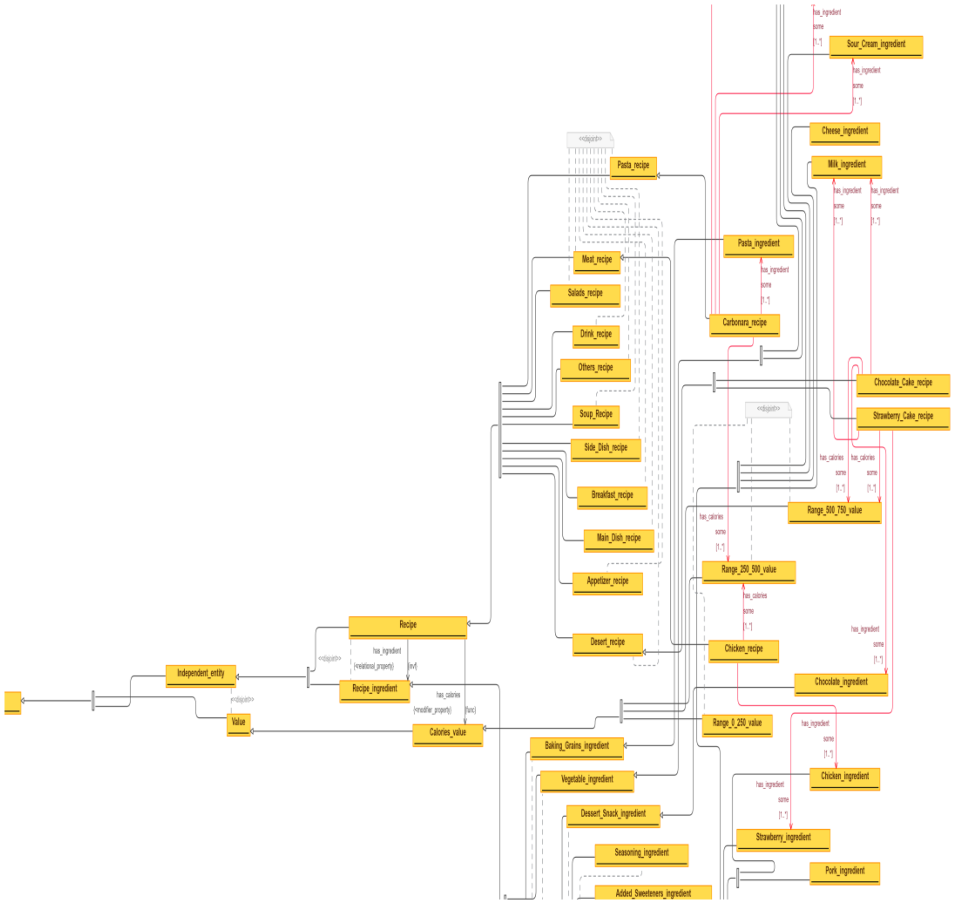


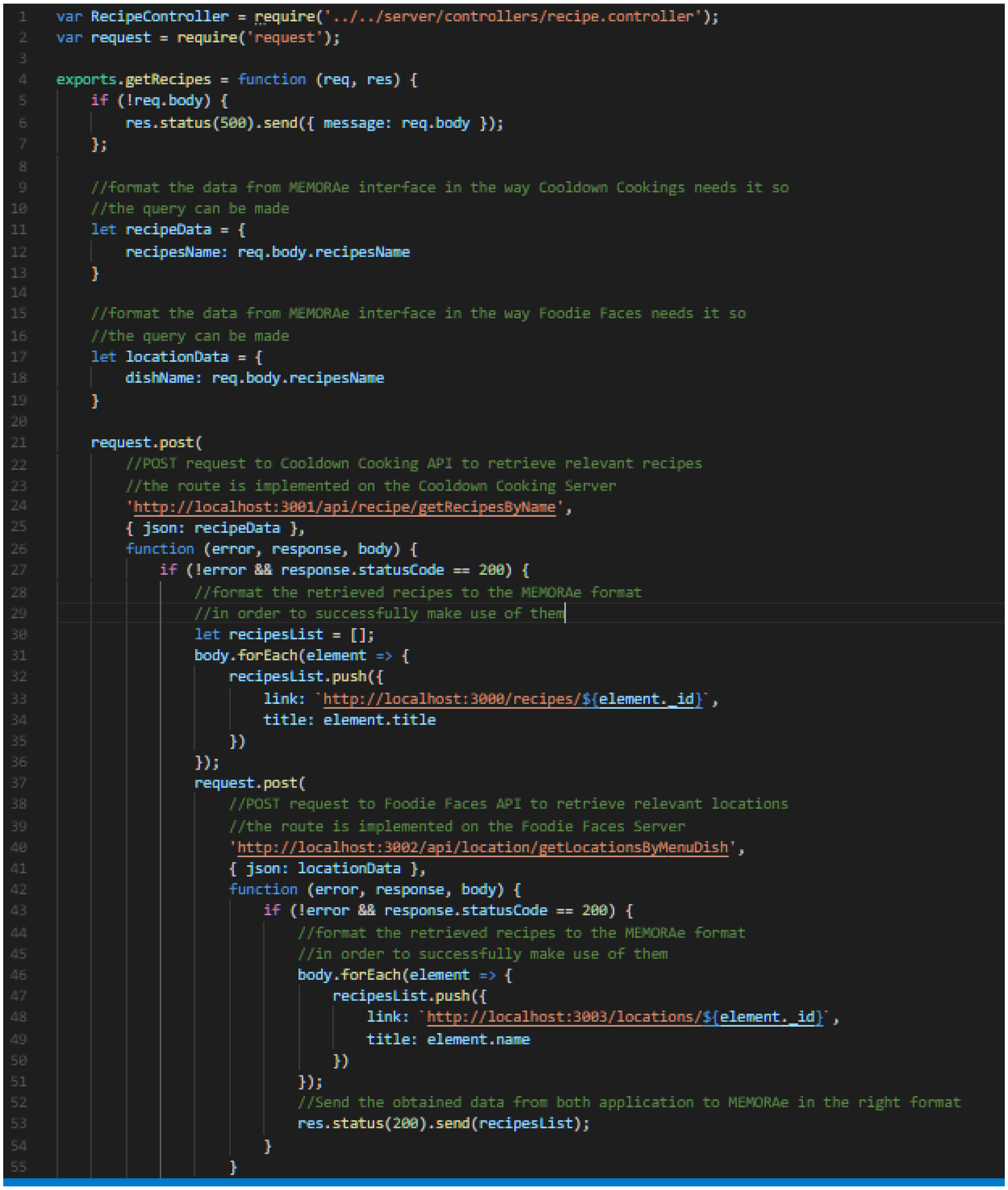
Figure 27 - Web ontology schema for CCFF

In order to have that integration successfully completed, a MEMORAe understanding is being required.

The link between MEMORA and both of the applications has been made using a second layer of methods on the server-side parts which intermediates the communication between the platform and the developed applications.

This layer is built as an externalized API, which has access to both of the developed applications server-side routes and controllers.

The API takes input from the MEMORA interface and propagates it in two directions: first it queries Cooldown Cooking using its implemented routes to get data about the searched string. Once the response is being received on this second layer, it fires another request, but this time to Foodie Faces to obtain relevant pieces of data. Once all the data was received, the API starts the second phase: the JSON formatting. As MEMORA needs a specific JSON format to display the search results, the API will format the obtained data, in order to have compatibility in terms of communication format with MEMORA.



```
var RecipeController = require('../../server/controllers/recipe.controller');
var request = require('request');

exports.getRecipes = function (req, res) {
    if (!req.body) {
        res.status(500).send({ message: req.body });
    };

    //format the data from MEMORAe interface in the way Cooldown Cookings needs it so
    //the query can be made
    let recipeData = {
        recipesName: req.body.recipesName
    }

    //format the data from MEMORAe interface in the way Foodie Faces needs it so
    //the query can be made
    let locationData = {
        dishName: req.body.recipesName
    }

    request.post(
        //POST request to Cooldown Cooking API to retrieve relevant recipes
        //the route is implemented on the Cooldown Cooking Server
        'http://localhost:3001/api/recipe/getRecipesByName',
        { json: recipeData },
        function (error, response, body) {
            if (!error && response.statusCode == 200) {
                //format the retrieved recipes to the MEMORAe format
                //in order to successfully make use of them
                let recipesList = [];
                body.forEach(element => {
                    recipesList.push({
                        link: `http://localhost:3000/recipes/${element._id}`,
                        title: element.title
                    })
                });
                request.post(
                    //POST request to Foodie Faces API to retrieve relevant locations
                    //the route is implemented on the Foodie Faces Server
                    'http://localhost:3002/api/location/getLocationsByMenuDish',
                    { json: locationData },
                    function (error, response, body) {
                        if (!error && response.statusCode == 200) {
                            //format the retrieved recipes to the MEMORAe format
                            //in order to successfully make use of them
                            body.forEach(element => {
                                recipesList.push({
                                    link: `http://localhost:3003/locations/${element._id}`,
                                    title: element.name
                                })
                            });
                            //Send the obtained data from both application to MEMORAe in the right format
                            res.status(200).send(recipesList);
                        }
                    }
```

Figure 28 - Application MEMORAe code integration

APPLICATION SETUP

Before going to the step-by-step instructions, the user should have the following prerequisite installed:

- MongoDB v3.6 or newer – the downloading link can be found in here https://www.mongodb.com/download-center#community
- Node package manager (npm) – the downloading link can be found in here https://www.npmjs.com/get-npm
- Postman – the downloading link can be found in here https://www.getpostman.com/

In order to locally setup the application, the following steps are to be completed:

- Unzip the foodie-faces.rar file
- In the newly created folder with the unzipped files, open a command prompt. This can be done by pressing *ctrl + shift + right click* and choose *Open Command Prompt/Power Shell here*
- Now the following npm command should be written in the prompt in order to install all the dependencies modules for the server side: *npm install –save*
- After the installation is completed, the user should change the directory, from the root folder to the client folder. This can be done by typing the following command: *cd client*
- In the newly selected folder, type *npm install --save*. This will ensure the dependencies modules for the client side are also installed.
- Once the installation is completed, navigate back to the root folder with: *cd..*

After successfully completed the above steps, the application is good to go.

To run the application, complete the following steps:

- Navigate to MongoDB folder – this can usually be found in *C:\Program Files\MongoDB\Server\3.6\bin*
- From here, open *mongod.exe* – this will ensure the database is up and running
- Navigate back to the root folder of the unzipped foodie-faces.rar

- In here, open a new command prompt and write the following command: *npm start*

- Now the user should wait until the configuration is being made – this will usually takes up to 30 seconds. After this, a new tab will be opened in your browser – the application is finally ready to be used.

TIP: in order to have a populated database with locations, the user should run the following query in Postman:

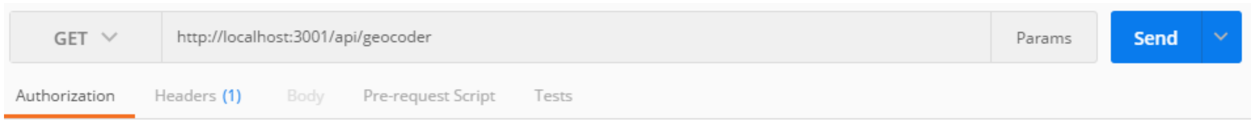


Figure 29 - GET method in order to populate database

Wait for up to one minute, press cancel and now, you have the locations for Paris. If you want to add locations for Lyon and Compiegne, you need to go in the project folder **/server/controllers/utils/utils.controller.js** and replace '/assets/locationsData/parisCrawledFormatted.json' with '/assets/locationsData/compiegneCrawledFormatted.json' or '/assets/locationsData/lyonCrawledFormatted.json', in **line 9** like in the example:

```
9      let locations = JSON.parse(fs.readFileSync('assets/locationsData/compiegneCrawledFormatted.json', 'utf8'));
```

Press cancel, and now, enter the next query:

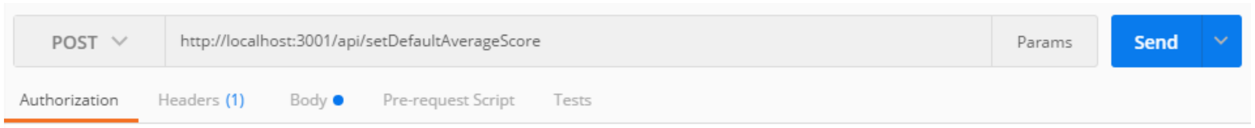


Figure 30 - POST method in order to set default average score

Now wait for up to 30 seconds. Now the database should be also ready

CONCLUSIONS

This project was a challenge for me because I have realized that I can develop and complete a task by myself, just using the internet resources. I have discovered how to find and select information in a useful way and the most important – how to use my programming skills in order to solve a humanity problem.

I have learned what actually means to develop a project from the beginning – create an idea, find an answer to solve it, in order to fulfill the people needs, design a proper database, to fit your methods, and a simple, intuitive interface for users. Covering all the phases, these steps actually help to improve my organizing and professional abilities for my future career.

The most difficult part for me was to plan all the application phases, respect the schedule and finding a great solution for people needs.

Also, the design part, speaking about how the application looks, was a difficult step for me, because in the past, I focused on the application functionality, and less on the visual styles. Fortunately, in this problem, I have found Material-UI which is an open-source project that features React components that implement Google's Material Design and it fits my needs, in order to create simple style for my application.

REFERENCES

1. https://www.studytonight.com/mongodb/introduction-to-mongodb
2. https://medium.freecodecamp.org/introduction-to-mongoose-for-mongodb-d2a7aa593c57
3. https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1
4. https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5
5. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction
6. https://medium.freecodecamp.org/going-out-to-eat-and-understanding-the-basics-of-express-js-f034a029fb66
7. https://medium.freecodecamp.org/everything-you-need-to-know-about-react-eaedf53238c4
8. https://www.tutorialspoint.com/reactjs/reactjs_overview.htm
9. https://developer.telerik.com/featured/introduction-to-the-react-javascript-framework/
10. https://medium.com/material-ui/material-ui-v1-is-out-e73ce13463eb
11. https://medium.freecodecamp.org/meet-your-material-ui-your-new-favorite-user-interface-library-6349a1c88a8c
12. https://hackernoon.com/mean-stack-development-for-developers-4d88c40c4103
13. https://cloudinary.com/documentation/upload_images
14. https://en.wikipedia.org/wiki/Web_Ontology_Language
15. https://en.wikipedia.org/wiki/Prot%C3%A9g%C3%A9_(software)
16. http://scraping.pro/turn-any-interactive-website-into-api-with-parsehub/
17. http://memorae.hds.utc.fr/