Web and Internet Engineering

Final Project: *EatKraken - Food delivery system*

CS08

Raffaele Tranquillini
Francesco Piccoli
Giulio Sciortino

Free University of Bolzano - 2020

Table of contents

User stories

- *"As a user, I want to order food from my favourite restaurant so that I can have it delivered at my place in respect of my personal needs and health condition."*

A user can select a city, select one or more dishes from one or more restaurants delivering at their city, check each dish information (allergens, ingredients, nutritional table), add items to cart and finally check out and confirm the order by inserting their personal details and address information.

The user can attach a particular request to the order to some restaurant, and can also decide the delivery type (e.g. booking a place to eat in, takeaway from store and proper home delivery).

The user will then receive an email from the website confirming the placement of the order, plus another email from the restaurant once it accepts or rejects their order. In this email they will receive contact information (telephone number or email) of the concerned restaurant.

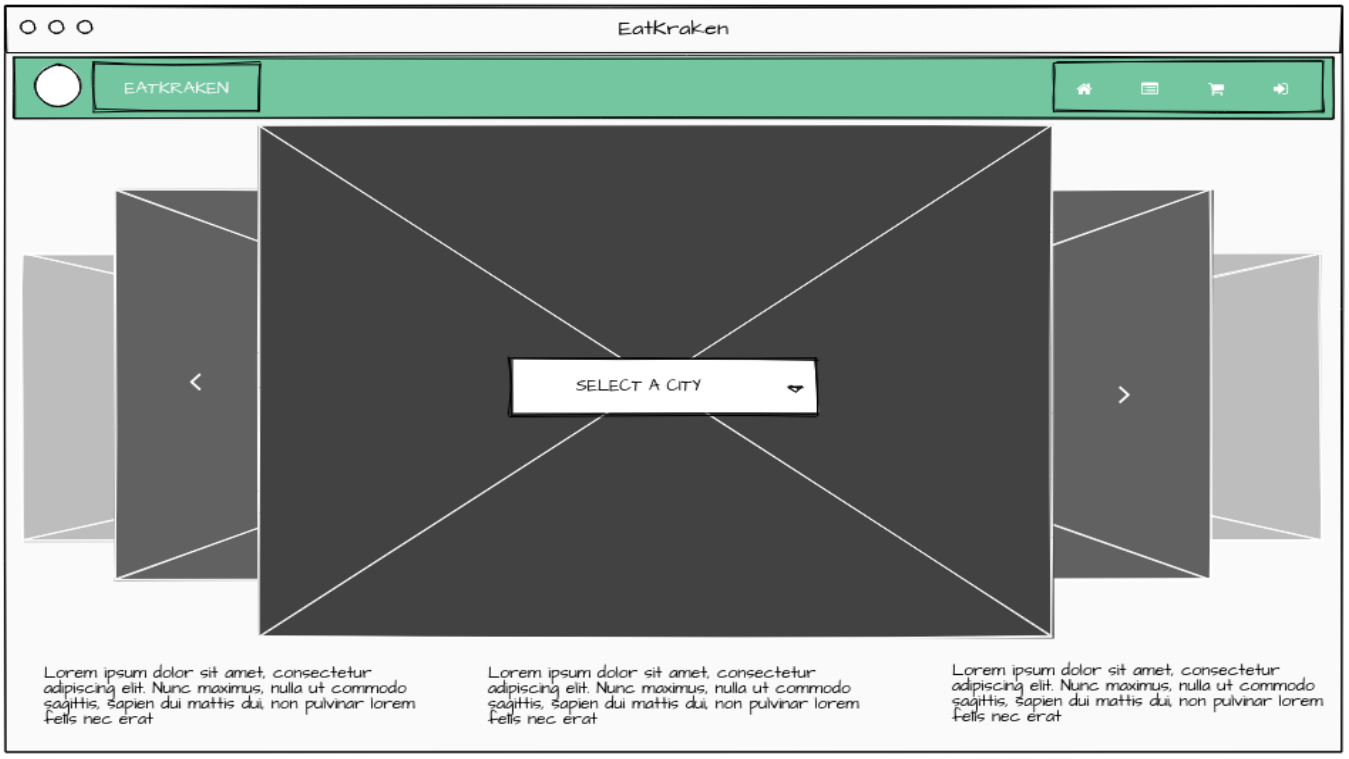
- *"As an owner of a restaurant, I need to receive my clients' orders in an ordered and clear fashion, so that I can serve them in the shortest time possible."*

A restaurant owner is able to add/delete the dishes they serve from the list, check their orders and accept/reject them before their deadline. The owner is able to manage their restaurant's account by logging in with the token that the *EatKraken* team provides to them.

From the *Orders* page owners can see information regarding their customers, contact information, requests/messages to the restaurant, the type of requested delivery and the time left to finish the order on time. In addition, the owner can print a list of the accepted orders.

Wireframes

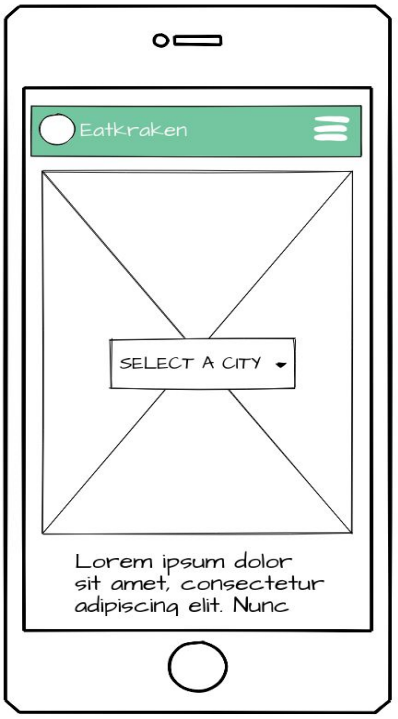
Homepage



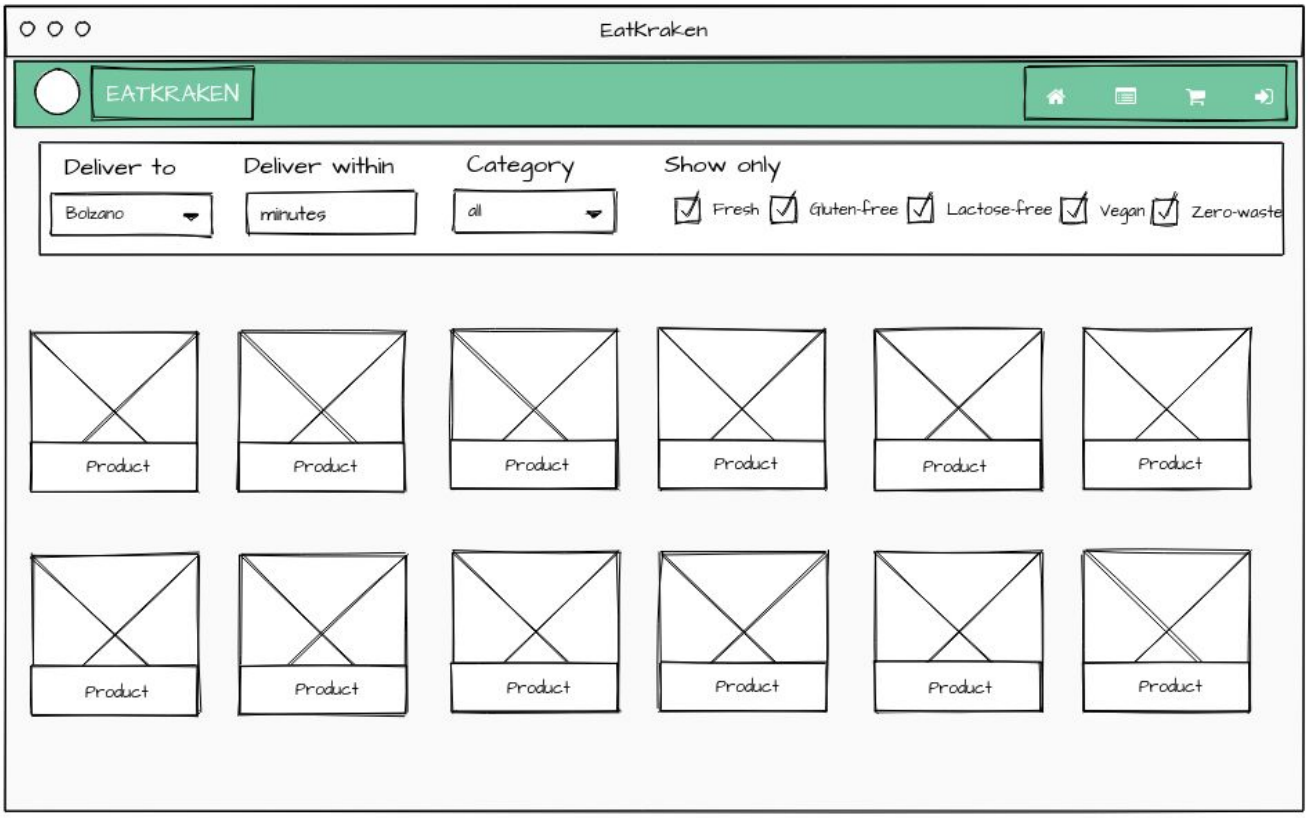
The *homepage* is the first page that the user of the website will see and land on, therefore it should be designed to give a quick idea of what our website does and an easy way to start using it.

Here, a carousel with some images of the available dishes is displayed, and above them a drop-down allows selecting the user's town to see all available dishes quickly.

In addition, some information regarding the features and the ethical values of the restaurant are provided, and scrolling down the page the user is able to see some pictures of the restaurants which work with *Eatkraken* and some (mock) reviews about the website.

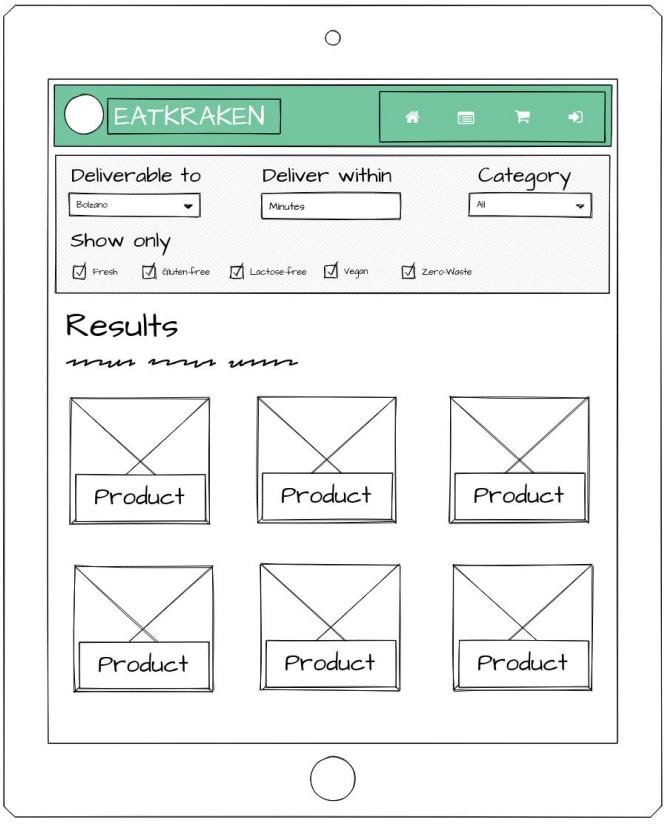


List page

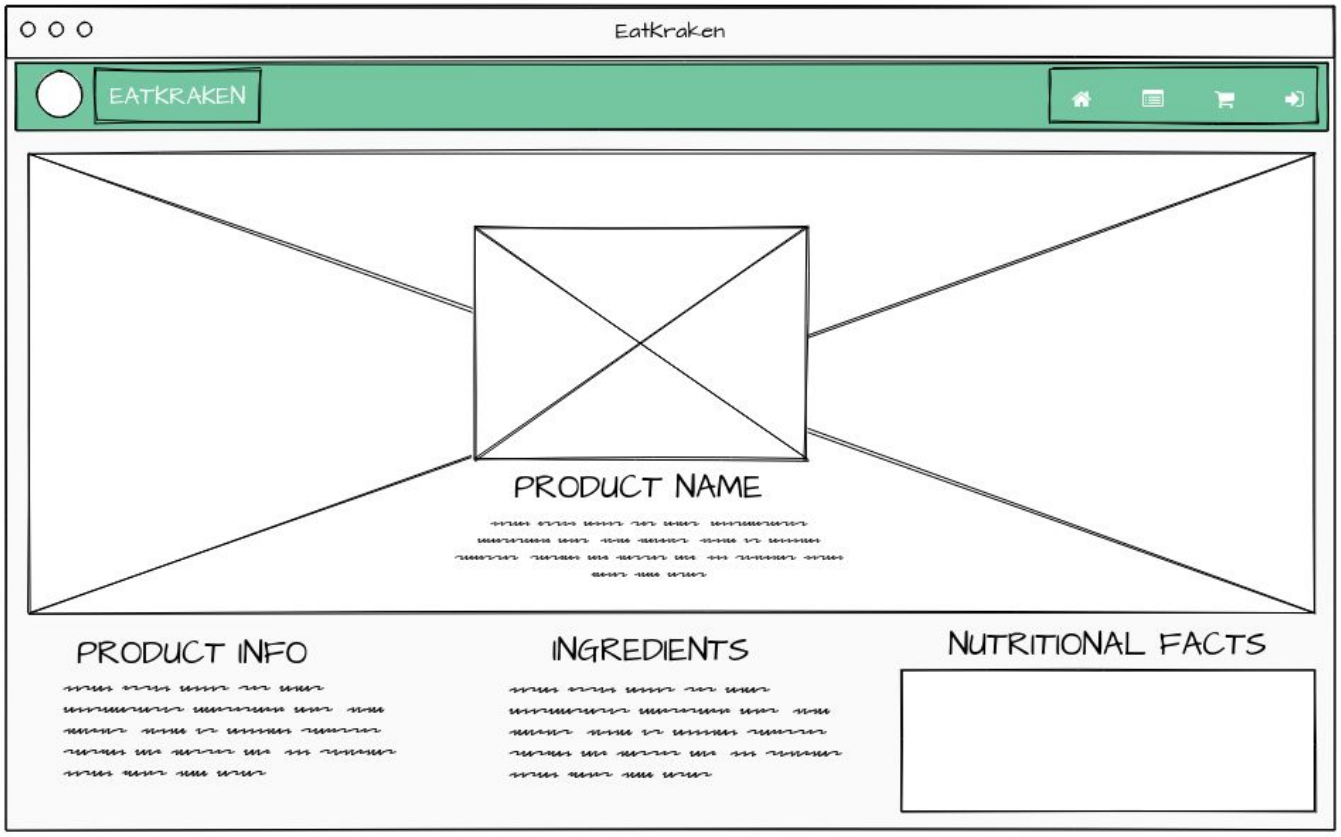


The *list* webpage is where the user can filter, select from all available dishes, and click on a dish to check its information (and possibly add it to cart).

Several possible options are offered to customise your search and filter results, for example, the user can select again the city, the delivery timings, the category, and some allergenes options such as gluten-free or lactose-free options or again vegan products.



Product details page



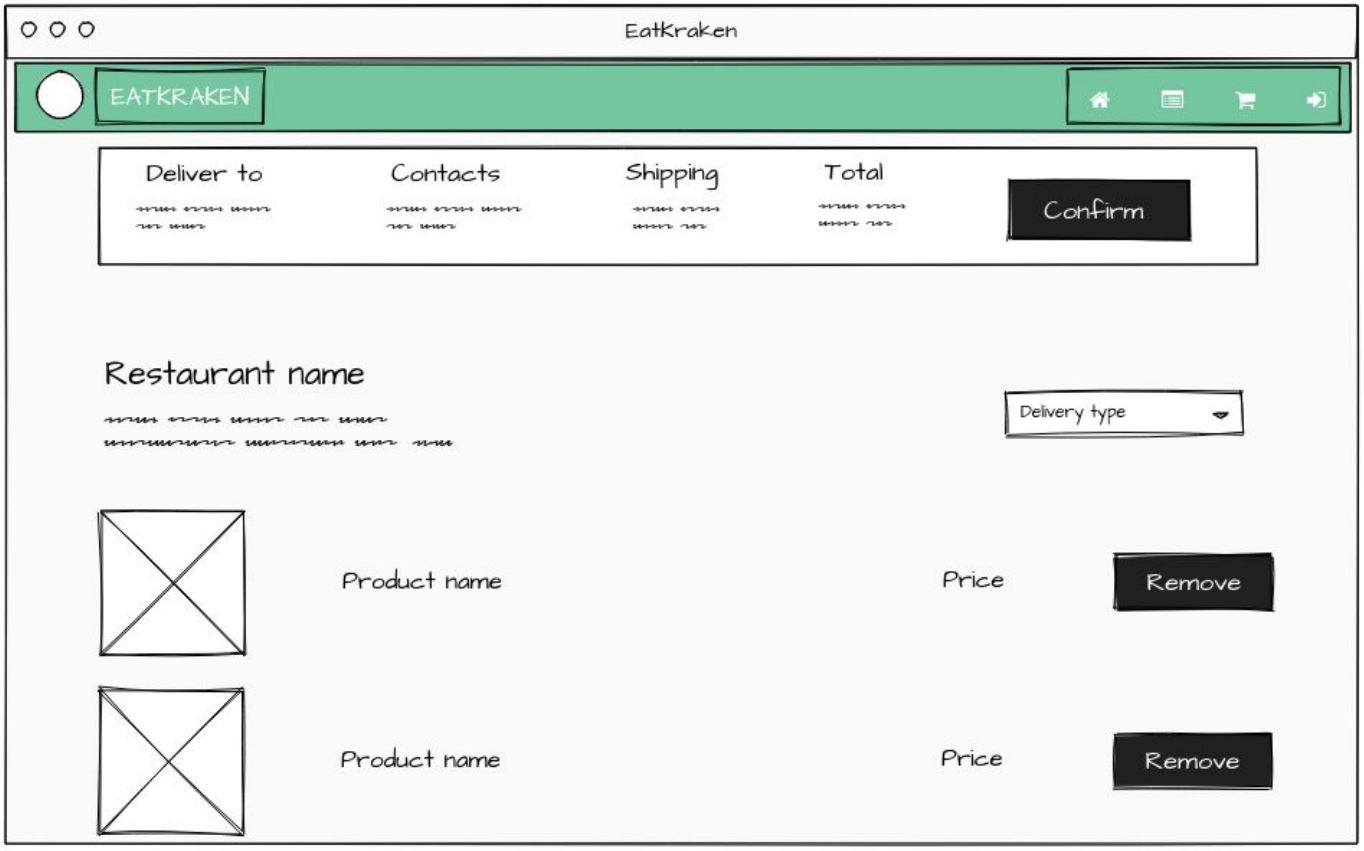
The *product* page displays all available information for one single dish (identified by its *code*), often selected from the list page seen before.

Here in addition to the image of the product displayed in foreground, the user can check some interesting facts regarding the specifics of the product:

- information regarding the product such as vegan, zero-waste, fresh..
- ingredients' list
- nutritional label with kcal, carbs, fats, and proteins per 100g of product

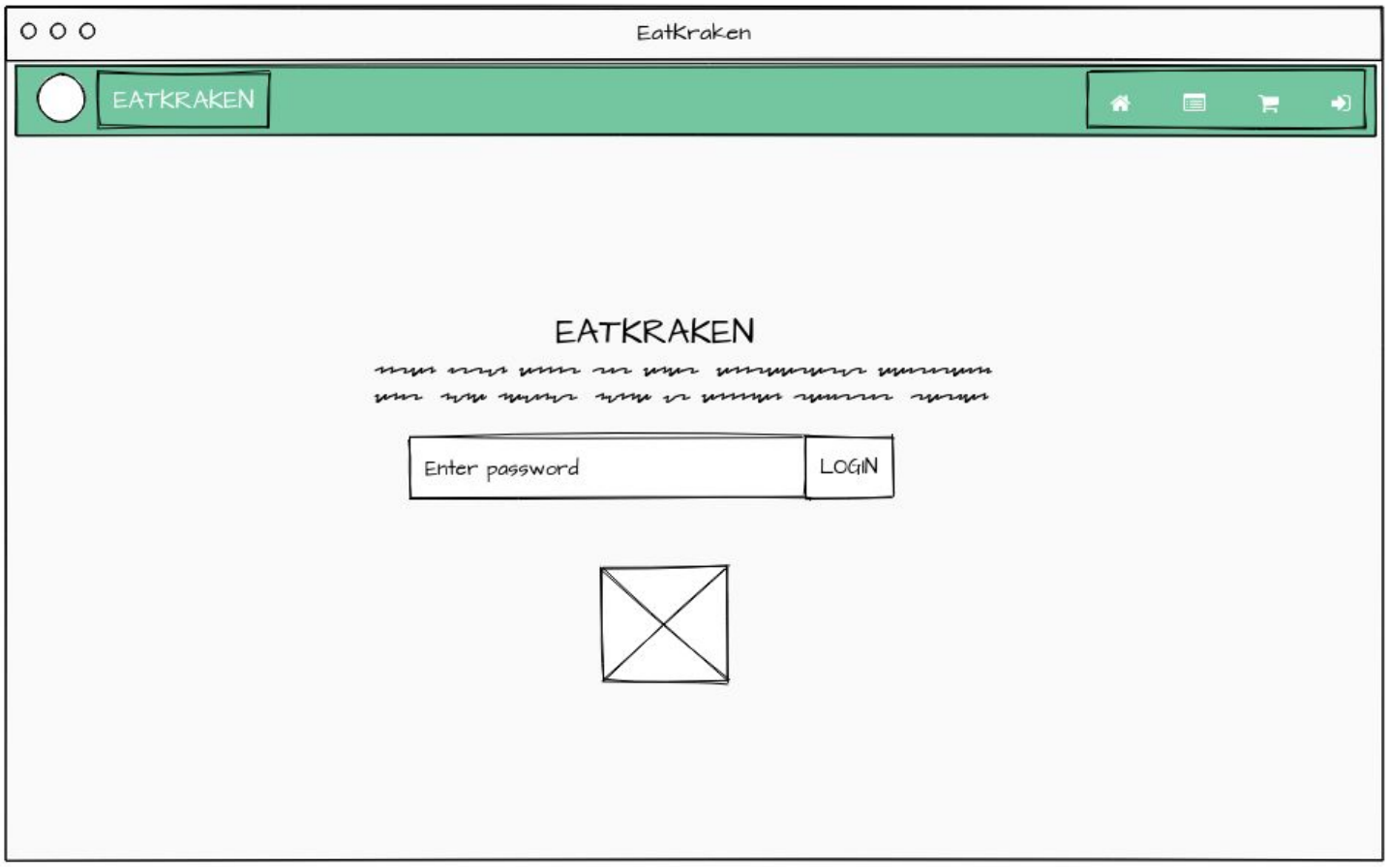
In addition to that, by scrolling down the webpage, the user will see some information regarding the restaurant which prepares the dish, specifically, an image of the restaurant will be displayed along with a short description of the restaurant.

Checkout page



The *checkout* page is the last page shown to the customer in the process of placing an order. Here a summary of their order is displayed, in particular all the products they put in the cart will be shown, and from here the user can remove dishes, choose the delivery type, and finally enter all their personal information to finalize and confirm the order. The total price will be the result of the sum of all the prices of the dishes and the cost for the specified delivery type.

Restaurant dashboard - Login page



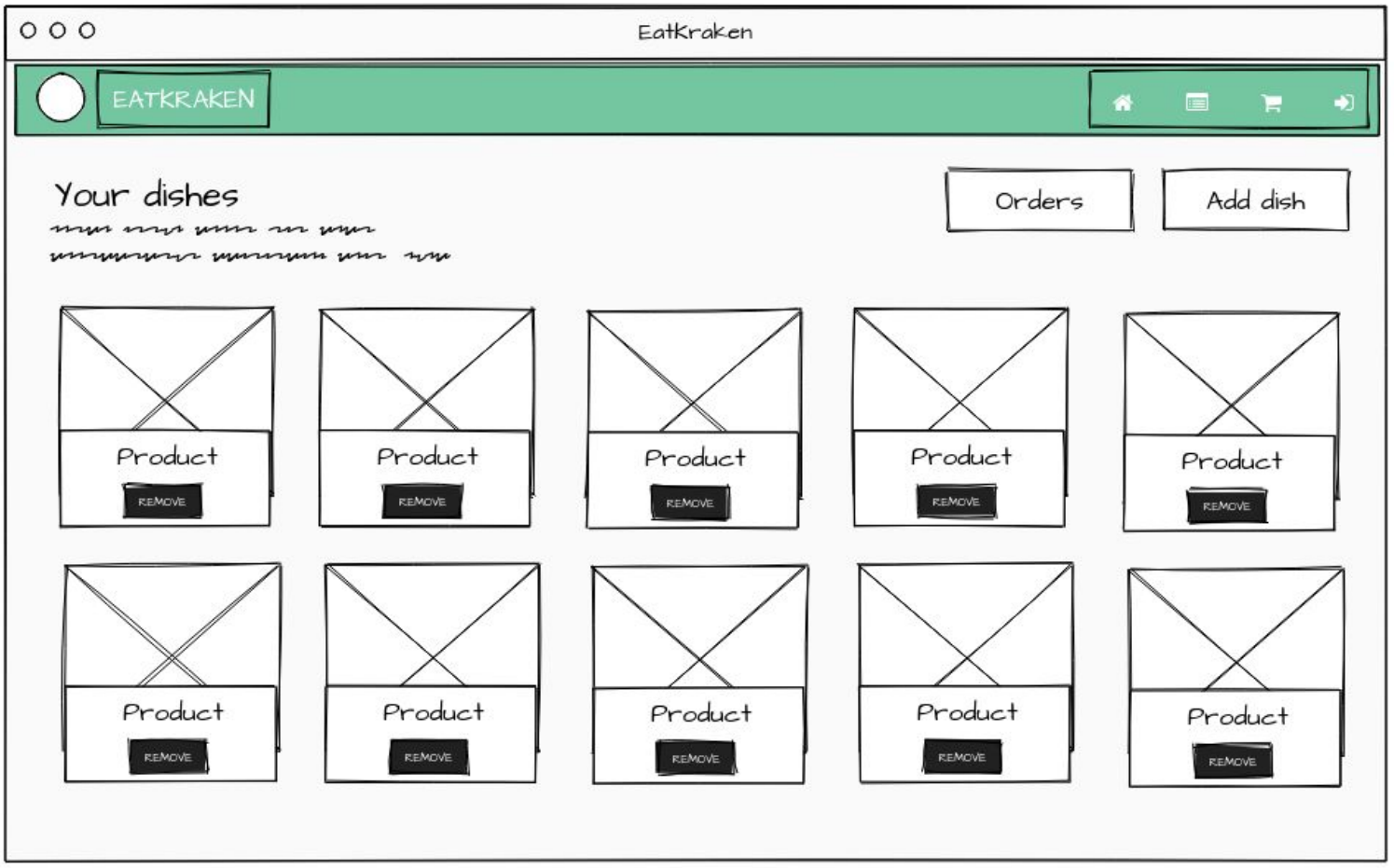
The *login* page is where the restaurant owner starts the real interaction with the website to manage their restaurant, here they will insert the random, 32-character secret token provided to them by the *eatkraken* team when the account was created, and login to their restaurant account to manage orders and dishes.

Navbar

The navbar menu is an unique widget that appears in all the pages of the website, from which the user can do several actions:

- go back to *homepage* either by clicking on the *EatKraken* logo, or by clicking on the home button on the right.
- go to the *list* page directly to select new dishes.
- go to the *checkout* page.
- go to the *login* page where only the restaurant owners can login.

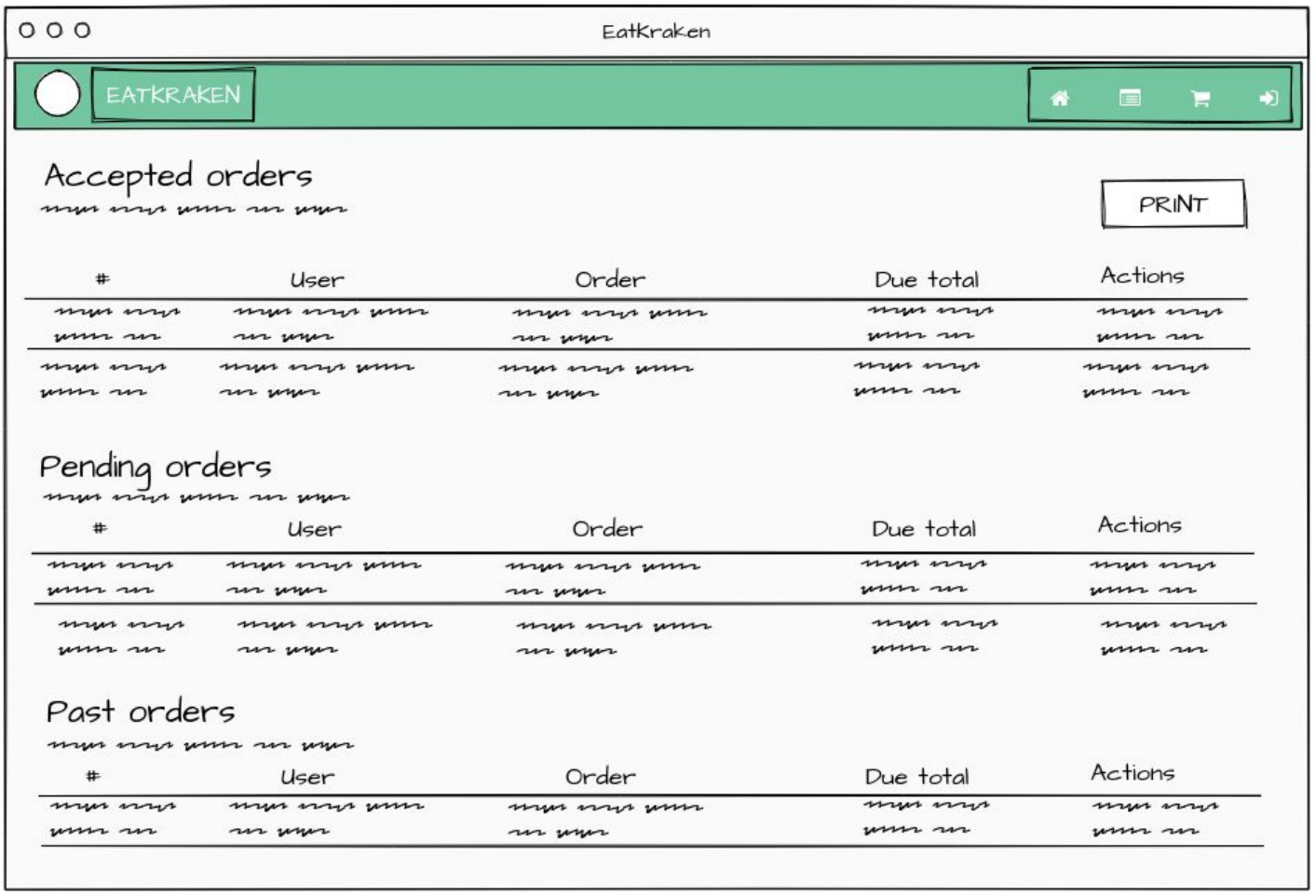
Restaurant dashboard - Manage Dishes page



Once logged, the restaurant owner will land on this page, from which they can

- see all their restaurants' dishes and remove some of them if they are no longer available; instead, by clicking on a dish name, the *product* page of that dish will be open in a new tab
- go to the *orders* webpage by clicking on the orders button
- add a new dish by clicking on the *add dish* button

Restaurant dashboard - Orders page

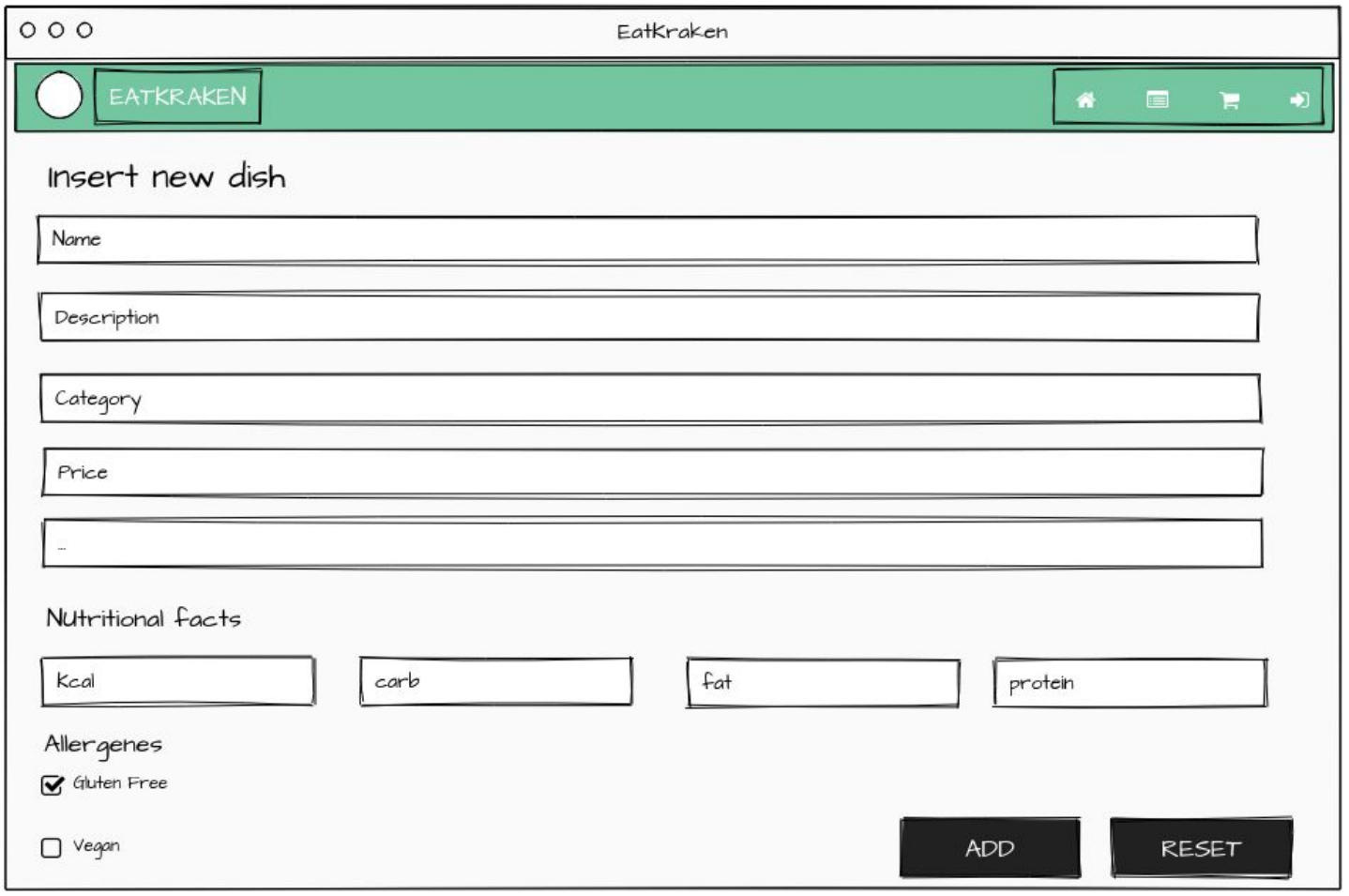


The *orders* page is where the restaurant owner can really manage orders.

Orders will be divided in three categories: accepted ones, pending ones, and past ones. For each order, relevant information such as customer's personal details, delivery type, custom note and time left will be displayed.

The owner can accept or reject orders, by clicking on the respective buttons in the action column (not reported in the picture). In addition the owner can print a minimal list of all the accepted (pending) order details by clicking on the "Print" button.

Restaurant dashboard - Add Dish page



The *add dish* page is a simple form, from which the restaurant owner will be able to insert a new dish by filling all the fields which are necessary for a product.
The dish will be added once the *Add* submit button is clicked (a JavaScript alert message confirming the insertion will be sent too), whereas all fields will be reset if the *Reset* button is clicked.

Technology choice

The platform choice led us to the *HTML5, JavaScript, PHP 7, jQuery 3.x, Bootstrap 3.x and (Postgre)SQL* covered in the course, without the use of any particular backend framework, mostly because none of the teammates was really experienced with web development, so it has been decided to stay on this classic technology model since it is well documented online and it was explained during the lectures and the labs of the course.

jQuery was also used for some minor components (such as the *AJAX* "add to cart" button, which however includes a fallback behaviour for non-JS-enabled browsers as "onclick" is ignored by those). Other prompt and confirm dialogs present in pages like *checkout.php* instead use *"vanilla" JavaScript* prompts.
Finally, the "delivery city" dialog is designed as a *Bootstrap* modal due to the lack of built-in *JS* drop-down dialog prompts.

System architecture

The system architecture follows a simplified *Model-View-Controller* architecture. This choice gave clarity and modularity in the code, and has also eased debugging and corrections all along the process.

All CSS stylesheets are included in one unique style.css file, as having a file for every *view* would have resulted in several, extremely small CSS files which would have added lots of overhead on each page load, while in this way a still very small (currently 11kB) CSS file can be cached on the first page load by browsers without wasting time or requests.

Frontend

To develop the templates of the webpages the *HTML5* markup language was used together with *SCSS* and the *Bootstrap 3* framework to style and embellish pages.
To add some dynamic functionalities (alert messages/confirm messages/ajax/cart animation) *JavaScript* was used.

The *main.js* file is the one and only *JavaScript* core of the website, and includes several small "helper" functions for our app, some of which general (e.g. a function to ask confirmation before submitting a form button) and some specific (e.g. those to modify and send the personal details in *checkout.php*).

Backend

The *EatKraken* backend was entirely developed in *PHP 7* following a procedural structure, with method names starting with the name of the enclosing "library" (e.g. ***db_****get_dishes*). Embeddable views (or "templates") have been stored in the *views* folder, with needed/fillable parameters defined as strings, and later included by the "*controller*" pages in the root and *restaurant* directory.

These "*controller*" pages are those accessed directly by the user, and most of them can handle some sort of input like GET and POST requests, documented briefly in the files, to decouple backend logic from pure visualization.

PDO has been chosen as database client library due to its extremely wide documentation and support for several types of databases (such as *MySQL, PostgreSQL, SQLite*).

Four custom "libraries" have been made to support the app, all included in the *libs* folder:

- *database.php* provides all support features for *EatKraken*'s database schema, with features ranging from SQL injection prevention (via prepared statements), to storage of PDO details (such as the choice to cache the database connection and save much time and bandwidth) and database credentials (username, password etc.)

- *session.php* instead is a support library for all features related to the local session stored in the user browser, including the needed *session_start()* procedure call, several support functions for temporarily saving user's data (such as address and delivery city) to be kept between orders, and functions for restaurant login and cart management

- *simple_email.php* provides configuration details and an abstraction layer to send confirmation or denial emails using the PHPMailer library, which we will discuss later

- *shipping_methods.php* provides a minimal unified storage for supported shipping methods in the system, with small support functions and association between database columns in the model, numerical identifiers for methods and names to be displayed.

Authentication system

We developed an extremely simple, token-based authentication system for restaurants for the sake of simplicity. Each token is supposed to be a unique, secret 32-byte (like MD5 hashes, to keep a migration to a password-based system easier) string generated by EK admins for each restaurant. The restaurant owner can then click the *Login* button in the navigation bar, enter their token and log in to the "dashboard" to manage dishes and orders

E-mails

We used the PHPMailer library and its *simple_email* abstraction layer for the e-mail features, and created an email address on Gmail due to difficulties we encountered in sending emails between each other from our local devices. These e-mails are still very simple and serve only demonstrative purposes.

APIs

A simple, stateful *cart_add* API endpoint has been created for this website to support the AJAX-based cart addition system.

It is worth noticing that, in hindsight, the Laravel framework would have eased several parts of the development of our application, providing an overall stabler backend and less code on our side.
However, the advantage of using "raw" PHP is that we quickly gained a better understanding of how web backends work at a lower level, which we believe will be extremely useful even when developing higher-level, framework-assisted projects in the future.

Database

We used PostgreSQL as relational DBMS thanks to our experience with it from the "*Introduction to Databases"* course from first semester, although we later found out that some features, like auto-increment columns, would have had simpler implementation and integration for MySQL. Nevertheless, the dynamic structure of PDO could make a potential migration of our schema to MySQL almost unnoticeable.

To enhance collaboration with database data, we decided to keep our database in the free online "*ElephantSQL*" database, which proved to be very reliable (with no data corruption or service interruptions), although the free plan provided some *serious* limitations on speed, which shows the huge difference in loading times for some pages, which load in the range of several seconds rather than milliseconds for a normal, small database stored offline on the same machine. A dump of the database contents has been attached in case a the project is to be tested on a locally configured server.

There are several tables in the database, which we can briefly summarize as follows:

- **restaurants**, including information, delivery costs and access tokens for restaurants in the system
- **cities**, associating to each city in the system an unique code
- **delivers_to**, serving as relationship between restaurants and cities they can deliver to
- **orders**, which contains all orders placed by users. These are placed as one per restaurant on checkout, and delivery deadline is computed as the maximum of all items in an order. Delivery address is not strictly mandatory field even for “home delivery” orders, as for example it may have been specified over the phone to the restaurant
- **order_items**, containing name and quantity of each item in a particular order, alongside a custom message to be shown to the restaurant if particular requirements are requested.
- **dishes**, containing data on each product in the database, including nutritional values, allergenes and maximum delivery time
- **categories**, associating supported categories to unique identifiers

Installation and Running

There are two ways to install and test EatKraken: the first is unzipping the *eatkraken* folder into the Apache server root, the other is *cd*-ing to the *eatkraken* folder from terminal and running this command to launch the builtin PHP web server:

```
php -S localhost:8080
```

While no local postgreSQL database is required as long as the online hosted one is working, to configure EatKraken to use a different, local database, PDO configuration variables in *libs/database.php* can be quickly adapted. Full compatibility with local MySQL databases probably exist, but table column types need to be adapted beforehand.

Minimum requirements are a local PHP 7.x installation, an Internet connection for database access and optionally Apache or Nginx. Testing under UNIX environments is recommended, but not strictly mandatory.

For testing purposes, external read/write access to the **postgreSQL** database used in the project is possible using the following credentials:

Host: balarama.db.elephantsql.com
Username: yhqbrujn
Password: vTdT4LC9LlOf_rgw6fA-Uz54Q-_xefB5
Database: yhqbrujn

An alternative “shortcut” can be using the following terminal command in UNIX-like systems:

```
psql postgres://yhqbrujn:vTdT4LC9LlOf_rgw6fA-Uz54Q-_xefB5@balarama.db.elephantsql.com:5432/yhqbrujn
```

Development process

The work on the project started at the end of March, in order to have enough time for development given the relatively complex nature of the project. Usually we worked on the project during the weekends, following a somewhat “agile” organization: we had an online meeting each Friday where we decided how to proceed, and then again each Monday to discuss what we did and what we were not able to do.

In order to share our updates and contributions we created a project on GitLab, which is a platform most of us had little or no experience with, and we gradually learned how to use *git* for collaborative editing.

Regarding the division of work, the frontend was mainly developed by Francesco, with Ajax and JavaScript parts done by Raffaele, while the great majority of the backend was written by Raffaele with some side functions developed by Francesco. Giulio helped populating the database and fixed some HTML and CSS details.

Reflection on development process and technology choice

Our experience was positive, and although we used a pretty “standard” web development method without the aid of backend frameworks, we learned many things on how web development should work, how content should be stored (and secured), and how backend and frontend are supposed to interact in a modern environment.

While, in hindsight, a simpler project may have been more appropriate for our web development experience (none for Francesco and Giulio, a small, single project for Raffaele), in the end it was a useful and engaging experience, and we are satisfied to see that our project went beyond becoming a simple “showcase” of contents we learned throughout this course, and looks more like a real website than a simple experiment. Although Raffaele and Francesco both exceeded the estimated 40 hours of work on this project, we are sure that any time spent on it will turn out to be very useful experience for our future work.

Reference and tooling

With regards to the material used as a reference to develop the website, the main sources were:

- Lecture and lab slides
- The W3schools website came very useful for all the languages used.
- The Bootstrap website which included several examples and excellent documentation for the Bootstrap 3 framework.

As expected, preferences for development environment changed between team members, and could be summarized as follows:

Raffaele:

- Microsoft Visual Studio Code for Linux (no humor intended) as IDE, thanks to its simplicity and great support for plugins
- *gitg (Git for GNOME)* for basic git features, command line for more complex commands; *GitLab* to host the shared project repository
- PHP's nice builtin server (*php -S localhost:8080*) to run the website locally, thanks to its much more versatile nature than Apache and its ease of configuration, given how running the database locally was not needed thanks to the external postgreSQL server
- Moderate intake of Döner Kebab to support the most challenging tasks

Francesco:

- Microsoft Visual Studio Code for macOS Catalina as IDE, thanks to its features and and its user friendly interface
- PSequel,a simple software to manage postgreSQL databases with a nice GUI.
- PHP's built-in webserver (*php -S localhost:8080*) to run a web server locally from the command line without the need for external software
- GitLab to manage teamwork

Giulio:

- Microsoft Visual Studio Code for Linux Ubuntu was my choice as a IDE for writing code, because of its user-friendly interface and because it's very popular among developers
- Beekeeper Studio has been used as GUI for managing postgreSQL database thanks to its semplicity and effectiveness
- PHP's builtin webserver to run the website locally, as it doesn't require external tools
- GitLab to manage team-work