Web applications

Group: DHI1V.So

Student name: Iana Lazareva

Student number: 526830

# Content

[Content 2](#_Toc107134857)

[Requirements 2](#_Toc107134858)

[Overview 3](#_Toc107134859)

[Framework choices 4](#_Toc107134860)

[Class diagram 5](#_Toc107134861)

[API specification 5](#_Toc107134862)

[1. GET requests 5](#_Toc107134863)

[2. POST requests 8](#_Toc107134864)

[3. PUT requests 9](#_Toc107134865)

[4. DELETE requests 12](#_Toc107134866)

[Sequence diagrams 14](#_Toc107134867)

[Wireframes 16](#_Toc107134868)

[Screenshots of the project 16](#_Toc107134869)

[Errors handling 20](#_Toc107134870)

[Design choices 22](#_Toc107134871)

# Requirements

− The application must consist of a separate frontend and backend

− The frontend must be written in HTML, CSS and JavaScript

− The backend must be written in Java, using Spring Boot

− The backend model must consist of at least three Entities (model classes) with a relation

between them

− At least one Entity must have a one-to-many relation with another Entity

− The backend must have full CRUD (Create, Read, Update, Delete) functionality for at least

two model classes

− The CRUD functionality must use the correct HTTP Verbs (GET, POST, PUT, DELETE)

− The backend must follow the REST specifications, including correct Status Codes

− The backend must persist its data using the Spring ORM layer (Hibernate)

− At least 3 backend endpoints must use query parameters

− At least 3 backend endpoints must use path parameters

− The frontend must consist of multiple pages (HTML) that are fully styled (CSS) and functional

(JS)

− Your pages must get their data from your own backend, retrieved using the Fetch API

− Your pages must post new data to your own backend, using the Fetch API

− Your Fetch API calls should be error handled. Checking on anything that goes wrong,

including status code errors and other (unexpected) errors

− Your pages must have form validation using JavaScript and built in HTML form validation

− The frontend must be simple enough that it can run by opening index.html and without any

package managers

− The frontend must include your own custom CSS-file that is written by yourself

− The styling can use CSS-only frameworks that do not include JavaScript, as long as you

validate the framework with your teacher

− The JavaScript must be built without any third-party frameworks

− The backend REST specification must be fully documented, including possible status codes

and example requests and responses

− At least two (more complex) requests must be documented, using sequence diagrams

− Design choices, including what frameworks are used or architectural choices made, must be

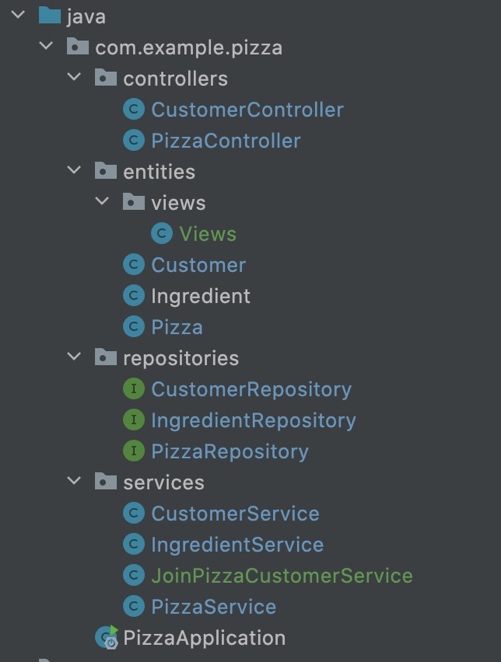
documented.

# Overview

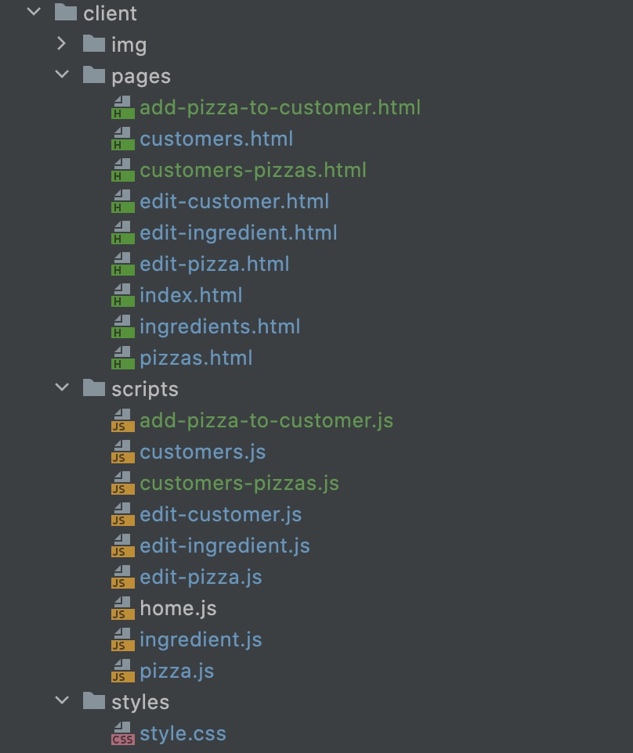
This project is made for the pizza business, where product owner can create, update and delete pizzas, add and edit ingredients to each pizza, see the total price for each pizza, and manage a customer’s (and their pizzas) database.

# Framework choices

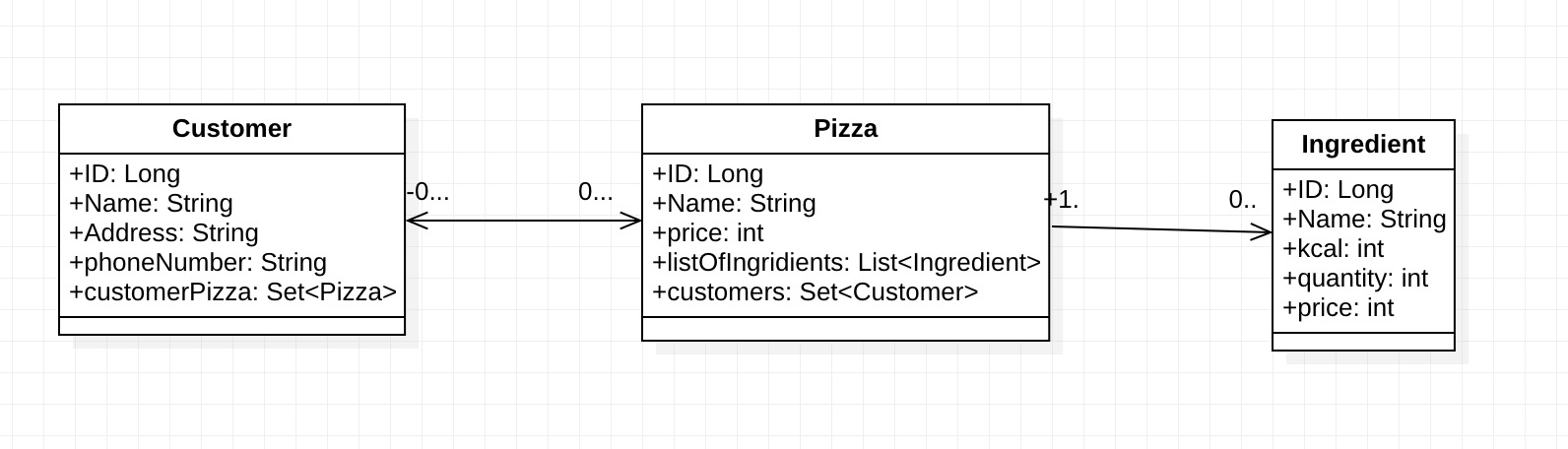
The backend of the application is written in Java using Spring Boot. The backend persists its data using the Spring ORM layer (Hibernate). Also, there are folders for controllers, entities (models), repositories, services, and class PizzaApplication for running an app.



The frontend of the application is written in HTML, CSS, and JavaScript. The frontend consists of 9 HTML pages, that are fully styled with CSS, and 9 functional JavaScript files, which are connected to those HTML pages.



# Class diagram



For this project, I’m using 3 tables.

**Customer.**Contains information about customers, such as id, name, address, phone number, and set of pizzas.

**Pizza**. This table includes rows id, name, price, list of ingredients, and set of customers.

**Ingredient.**The table contains id, name, kcal, quantity, and price columns.

The relationship between pizza and ingredients is “one to many”, which means that a pizza can have plenty of ingredients. Customer and pizza connected with a “many to many” relationship. Each customer can have a lot of pizzas, and each pizza can have a lot of customers.

# API specification

## GET requests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GET** | **/customers** | | | |
| Get a list of all customers | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | |  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | List of customers can be empty | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GET** | **/customers/{id}** | | | |
| Get one customer, based on id | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | Id of the customer to find |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | The found customer based on ID | |
| 404 | Thrown if no customer was found | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GET** | **/customers/{id}/pizzas** | | | |
| Get a list of pizzas for a specific customer, based on id | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | Id of the customer to find |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | List of pizzas for a specific customer | |
| 404 | Thrown if no customer was found | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GET** | **/pizzas** | | | |
| Get a list of all pizzas, independent of customer | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | |  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | List of pizzas, independent on customers. Can be empty | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GET** | **/pizzas/{id}** | | | |
| Get one pizza, based on id | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | Id of the pizza to find |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | The found pizza based on ID | |
| 404 | Thrown if no pizza was found | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GET** | **/ pizzas/{id}/ ingredients** | | | |
| Get a list of ingredients for a specific pizza, based on id | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | Id of the pizza to find |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | List of ingredients for a specific pizza | |
| 404 | Thrown if no pizza was found | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GET** | **/pizzas/{id}/customers** | | | |
| Get a list of customers for a specific pizza, based on id | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | Id of the customer to find |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | List of customers for a specific pizza | |
| 404 | Thrown if no pizza was found | |
|  |  | |
|  |  | |
|  |  | |

## POST requests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **POST** | **/customers** | | | |
| Create new customer | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | Customer | body | The customer that needs to be added |
|  |  |  |
|  |  |  |
|  |  |  |
|  | (path / body) |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 201 | Customer created successfully | |
| 400 | Thrown when something was wrong with customer body | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **POST** | **/pizzas** | | | |
| Create new pizza | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | Pizza | body | The pizza that needs to be added |
|  |  |  |
|  |  |  |
|  |  |  |
|  | (path / body) |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 201 | Pizza created successfully | |
| 400 | Thrown when something was wrong with pizza’s body | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **POST** | **/pizzas/{id}/ingredients** | | | |
| Create new ingredient and assign it to the pizza | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
|  | | id | path | Id of pizza |
| ingredient | Body | The ingredient |
|
|
|
|
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 201 | The ingredient was created successfully | |
| 400 | Thrown when something was wrong with ingredient body | |
| 404 | Pizza id not found | |
|  |  | |
|  |  | |

## PUT requests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PUT** | **/customers/{id}** | | | |
| Update a customer based on id | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | ID of the customer that needs to be updated |
| customer | body | Updated customer |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | Updated customer is retuned if successful | |
| 400 | Thrown if customer body was wrong or customer id in body did not matched customer id in path | |
| 404 | Thrown if customer was not found | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PUT** | **/ customers** **/{customerId}/pizzas/{pizzaId}** | | | |
| Add pizza to customer | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | customerId | path | ID of the customer that needs to be added a pizza |
| pizzaId | path | Id of the pizza that needs to be added to customer |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | Returned if successful | |
| 404 | Thrown if pizza was not found | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PUT** | **/pizzas/{id}** | | | |
| Update a pizza based on id | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | ID of the pizza that needs to be updated |
| pizza | body | Updated pizza |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | Updated pizza is retuned if successful | |
| 404 | Thrown if pizza was not found | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PUT** | **/pizzas/{pizzaId}/customers/{customerId}** | | | |
| Add a customer to pizza | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | pizzaId | path | ID of the pizza that needs to be added to customer |
| customerId | path | ID of the customer that needs to be added to pizza |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | Returned if successful | |
| 404 | Thrown if customer of pizza was not found | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PUT** | **/pizzas/{pizzaId}/ingredients/{ingredientId}** | | | |
| Update an ingredient based on id | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | ingredientId | path | ID of the ingredient that needs to be updated |
| ingredient | body | Updated ingredient |
| pizzaId | path | Id of the pizza with that ingredient |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 200 | Updated ingredient is retuned if successful | |
| 400 | Thrown if ingredient body was wrong or ingredient id in body did not matched ingredient id in path | |
| 404 | Thrown if ingredient was not found | |
|  |  | |
|  |  | |

## DELETE requests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DELETE** | **/customers/{id}** | | | |
| Delete a customer | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | ID of customer to be deleted |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 204 | Returned when deleting of customer was successful | |
| 404 | Thrown if no customer was found with that ID | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DELETE** | **/ customers /{customerId}/pizzas/{pizzaId}** | | | |
| Delete a pizza from customer | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | customerId | path | ID of customer with needed pizza to be deleted |
| pizzaId | path | ID of pizza which needed to be deleted |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 204 | Returned when deleting was successful | |
| 404 | Thrown if no pizza or customer was found with that ID | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DELETE** | **/pizzas/{id}** | | | |
| Delete a pizza (and all its ingredients) | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | id | path | ID of pizza to be deleted |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 204 | Returned when deleting of pizza was successful | |
| 404 | Thrown if no pizza was found with that ID | |
|  |  | |
|  |  | |
|  |  | |

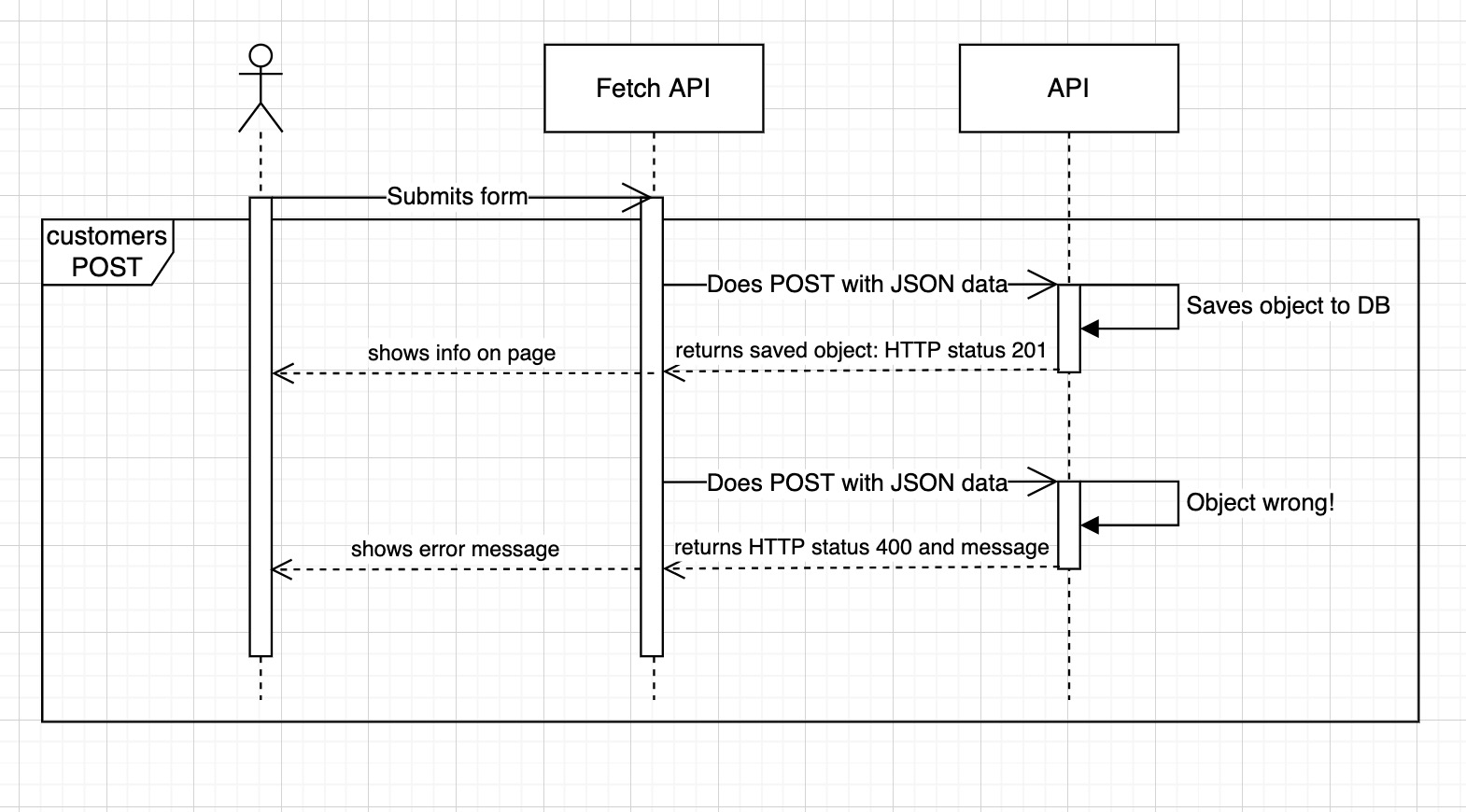
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DELETE** | **/pizzas/{pizzaId}/ingredients/{ingredientId}** | | | |
| Delete an ingredient | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | ingredientId | path | ID of ingredient to be deleted |
| pizzaId | path | ID of pizza with needed ingredient to be deleted |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 204 | Returned when deleting of ingredient was successful | |
| 404 | Thrown if no ingredient was found with that ID | |
|  |  | |
|  |  | |
|  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DELETE** | **/pizzas/{pizzaId}/customers/{customerId}** | | | |
| Delete a pizza from customer | | | | |
|  | | | | |
| **Parameters:** | | **Name** | **Type** | **Description** |
| *Add a \* to the name of required parameters.* | | customerId | path | ID of customer with pizza that needed to be deleted |
| pizzaId | path | ID of pizza that needed to be deleted |
|  |  |  |
|  |  |  |
|  |  |  |
| **Responses:** | | **Code** | **Description / example if successful** | |
|  | | 204 | Returned when deleting was successful | |
| 404 | Thrown if no pizza or customer was found with that ID | |
|  |  | |
|  |  | |
|  |  | |

# Sequence diagrams

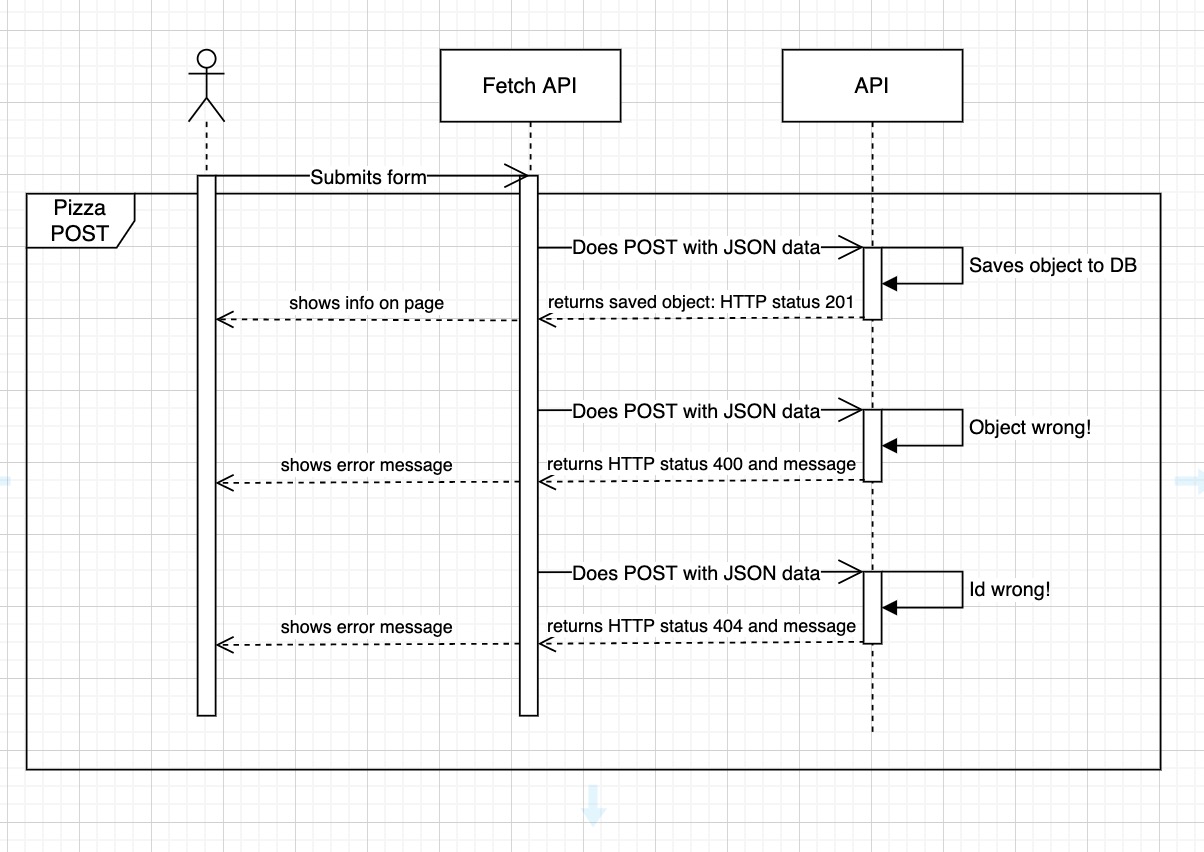
The sequence diagram below shows us a scenario for the customer's POST method.

HTTP status 201 will be returned when JSON data for POST was correct and the customer was created successfully. HTTP status 400 will return an error and a message, that something was wrong with the customer body.



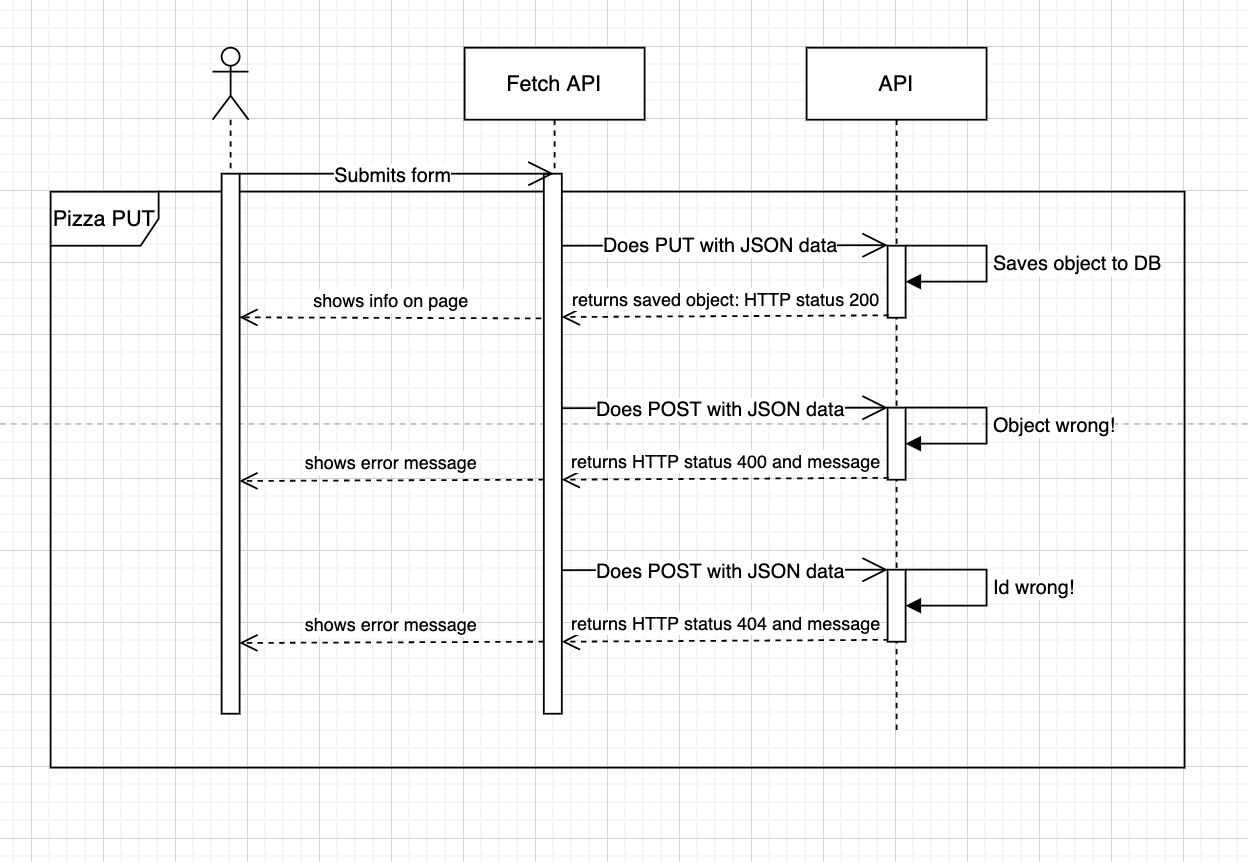
The next sequence diagram shows us a scenario for pizza’s POST method.

HTTP status 201 will be returned when JSON data for POST was correct and pizza has been created successfully. HTTP status 400 will return an error and a message, that something was wrong with the pizza’s body. And the error message and HTTP status 404 will return when pizza ID is not found.



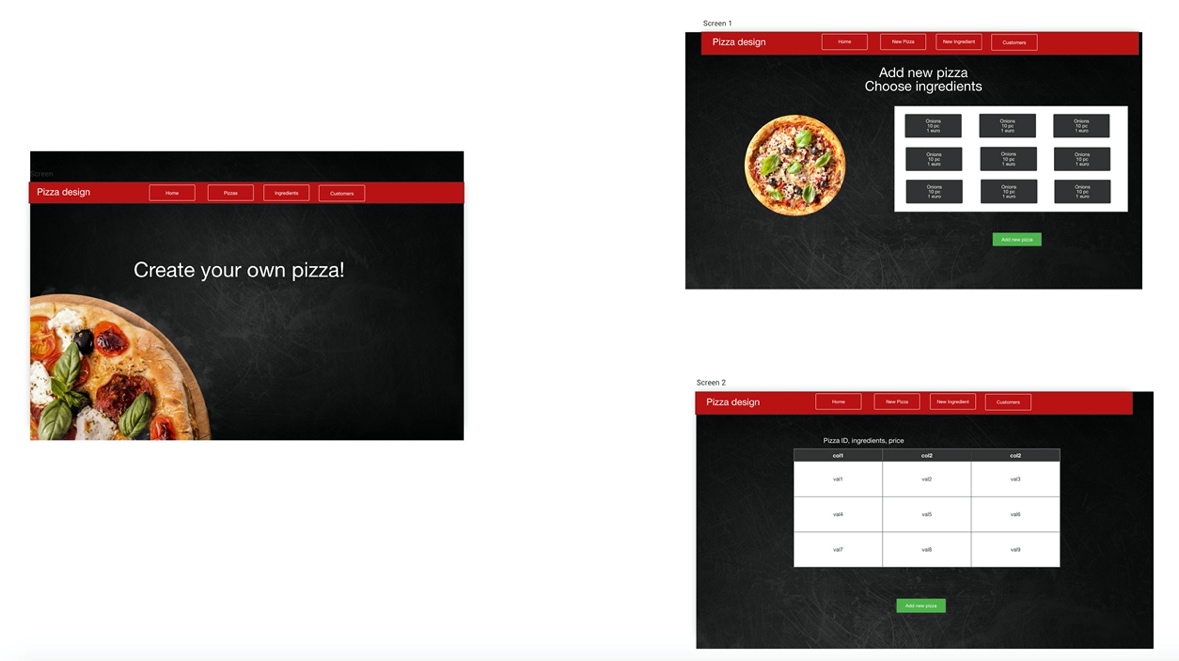
This sequence diagram contains a scenario for pizza’s PUT method.

HTTP status 200 will be returned when JSON data for PUT was correct and pizza has been updated successfully. HTTP status 400 will return an error and a message, that something was wrong with the pizza’s body. And the error message and HTTP status 404 will return when the pizza has not been found.



# Wireframes

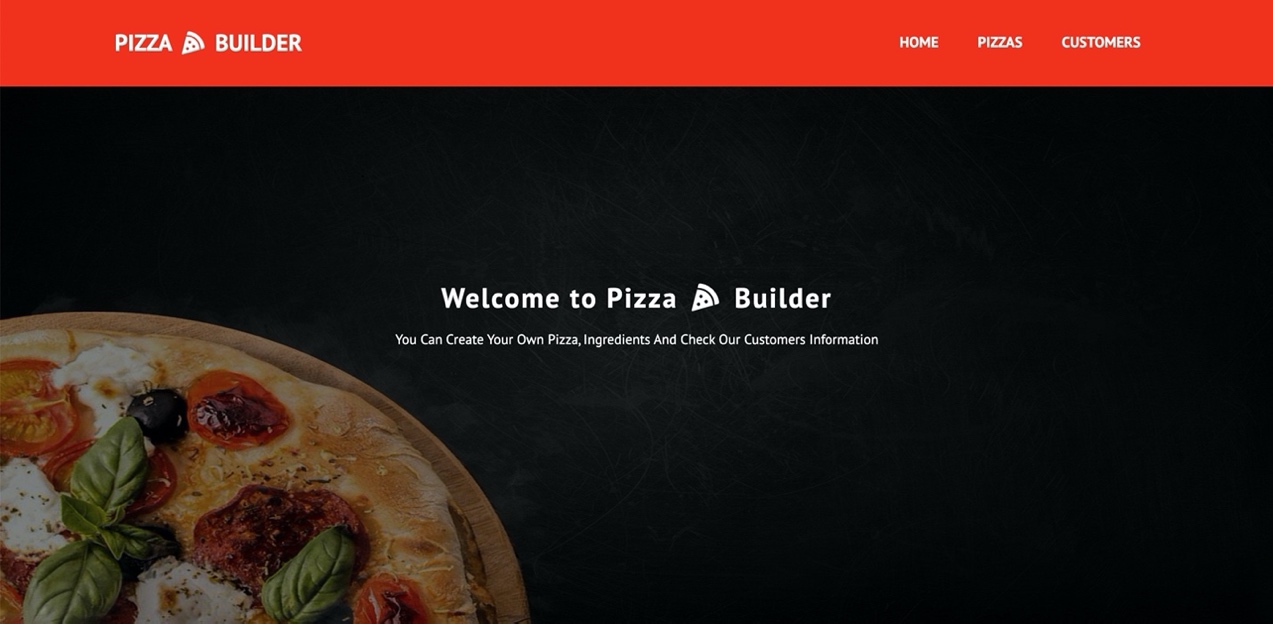
<https://quant-ux.com/#/share.html?h=a2aa10auryqaeEyE0KkbYzhuzhUzuRxKkLcRvzZHSBBuLULuNDgCDniTfIxy>



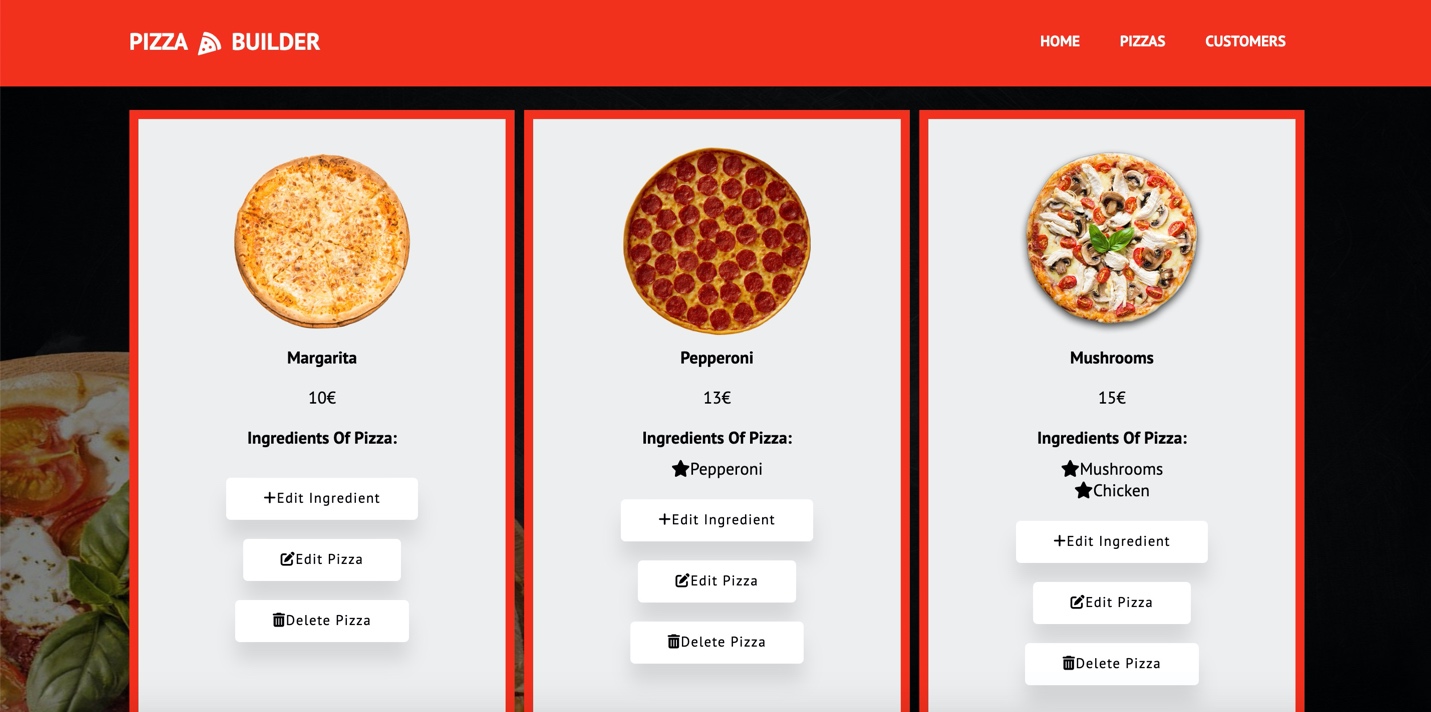
# Screenshots of the project

Home page – index.html

Start page with overall information.

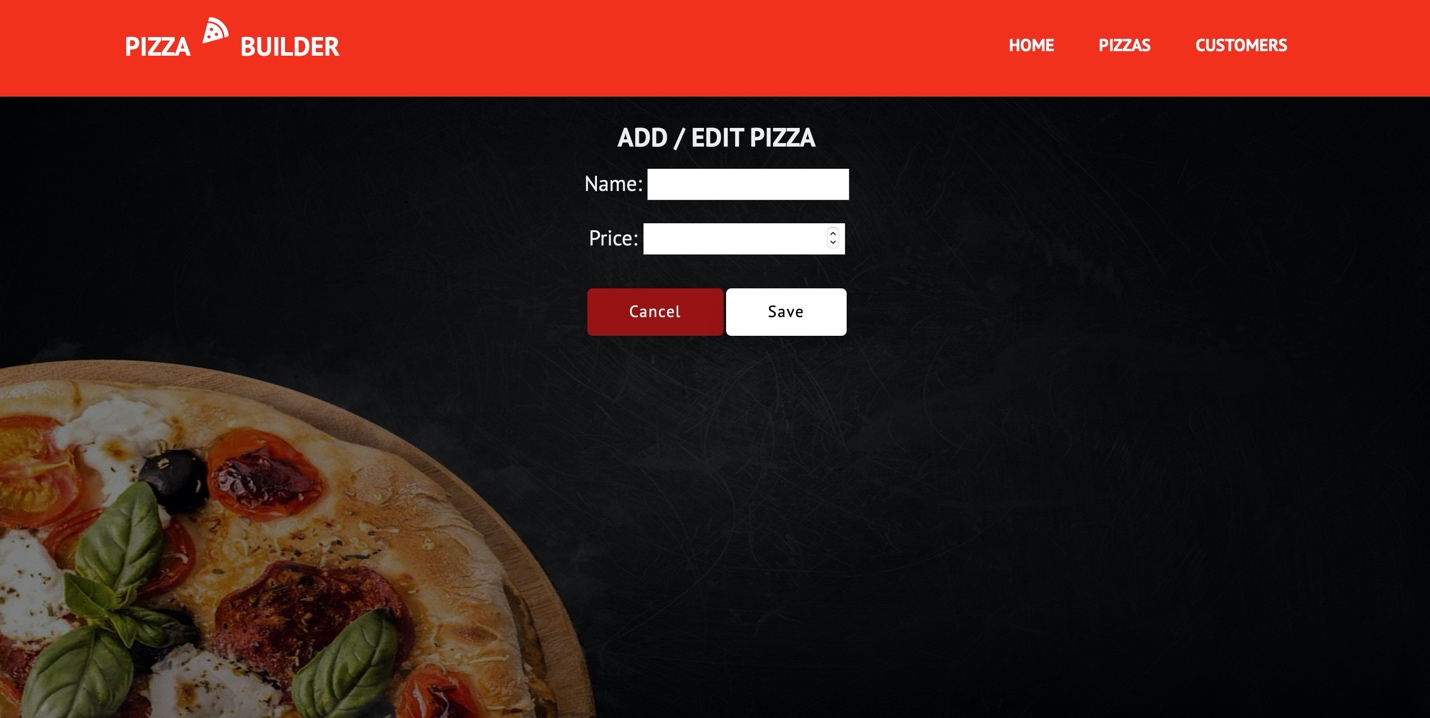


Pizzas page – pizzas.html

Page, where all pizzas from database are shown. From this page, user has access to functions such as “edit ingredient”, “edit pizza”, “delete pizza” and “add new pizza” at the bottom. 

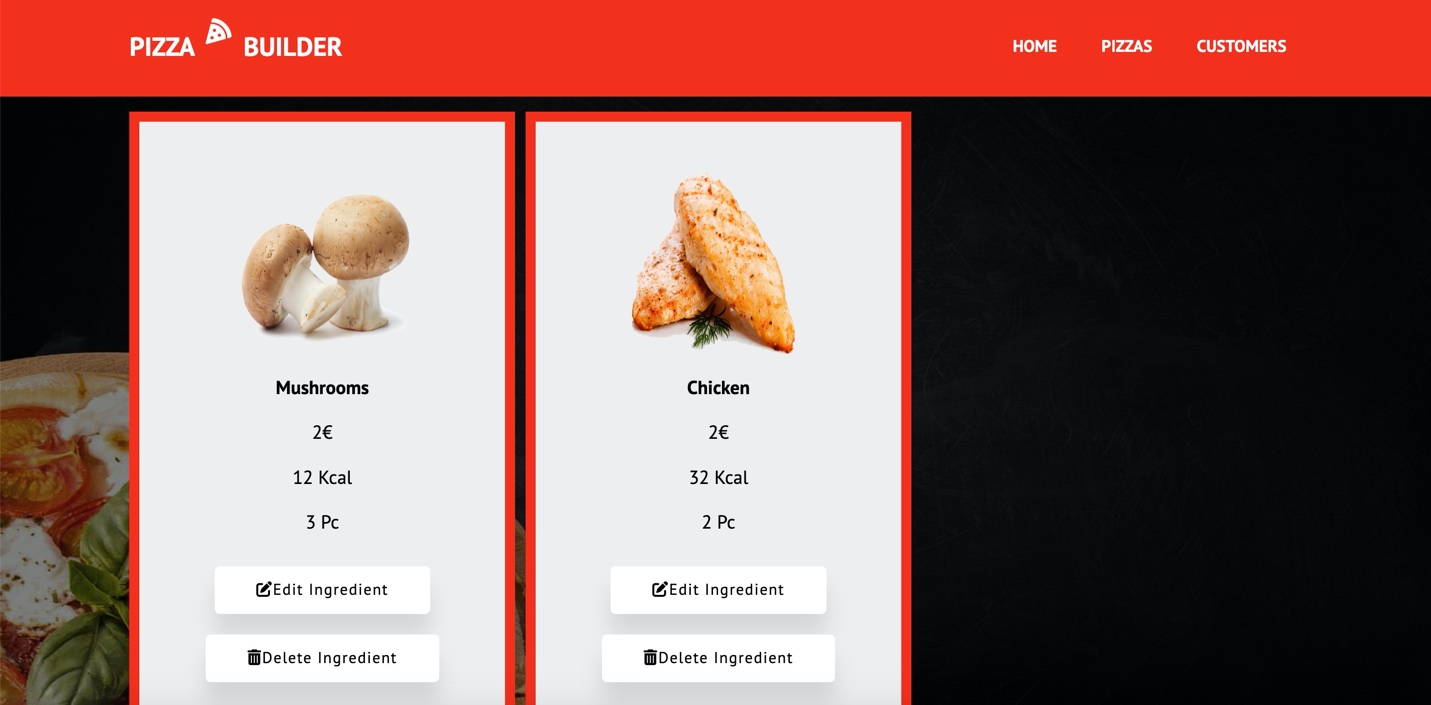
Add or edit pizza page – edit-pizza.html

This page will appear, when the user clicks on the “add new pizza” or “edit pizza” buttons. If the option is “edit pizza” in the URL in params will also appear pizza’s id.



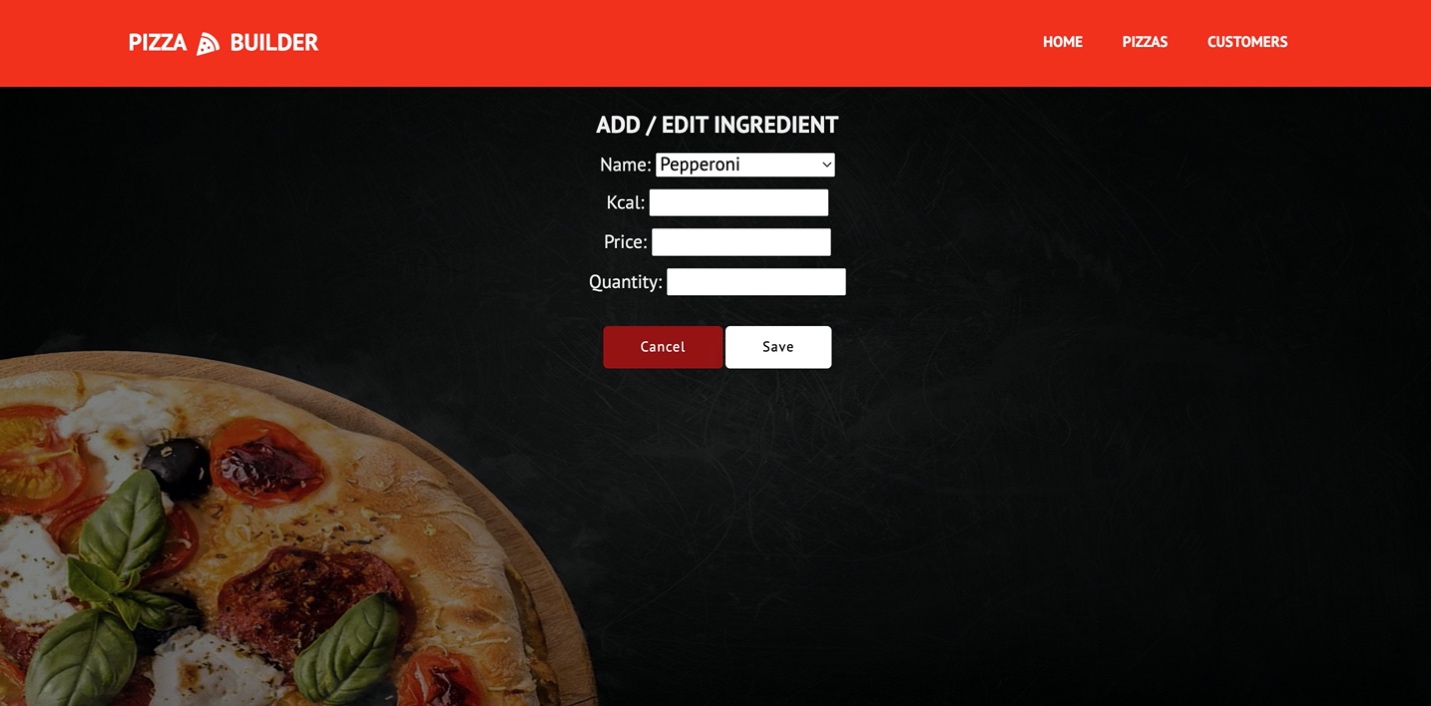
Ingredients page – ingredients.html

The page below will appear when the user clicks on the “edit ingredient” button. If there are no ingredients, there will be just a button “add new ingredient”. Here users can also edit or delete existing ingredients or add new ones.



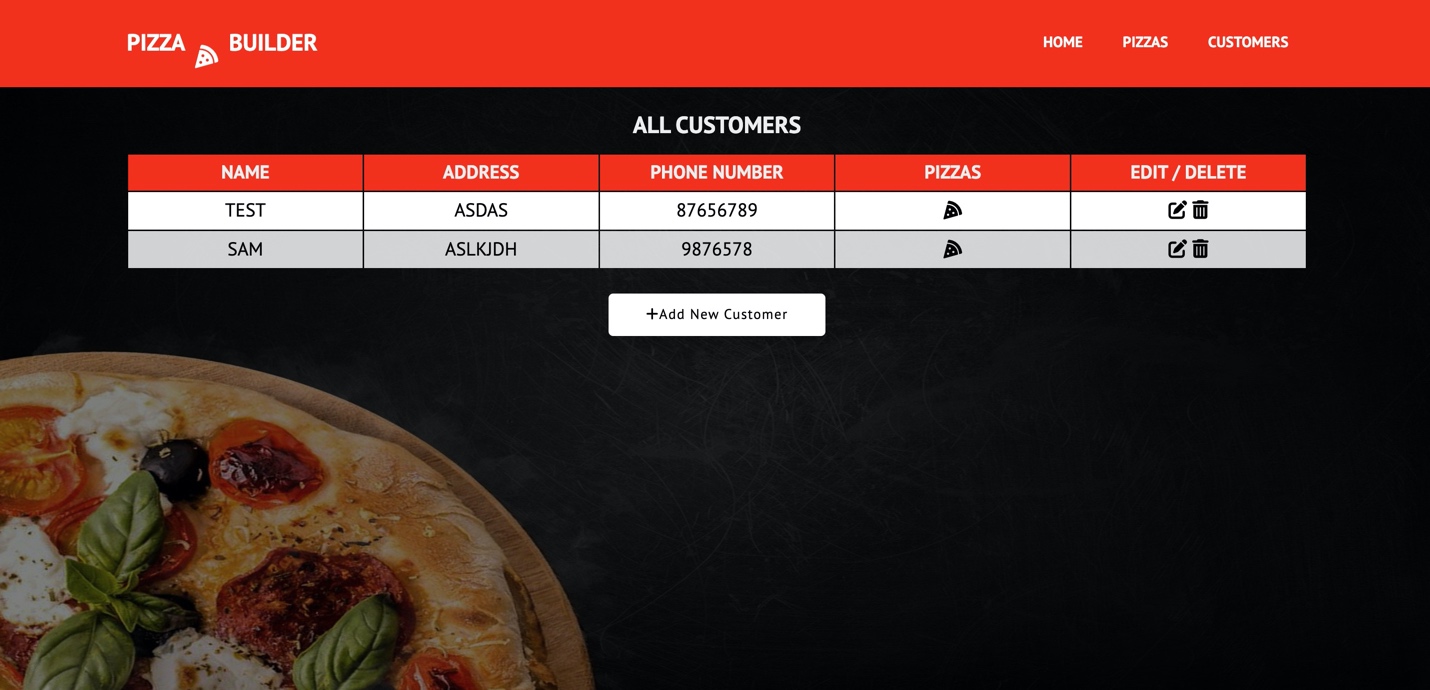
Add or edit ingredients page – edit-ingredients.html

If a user clicks on a button “edit ingredient” in the URL in params will also appear ingredient’s id.



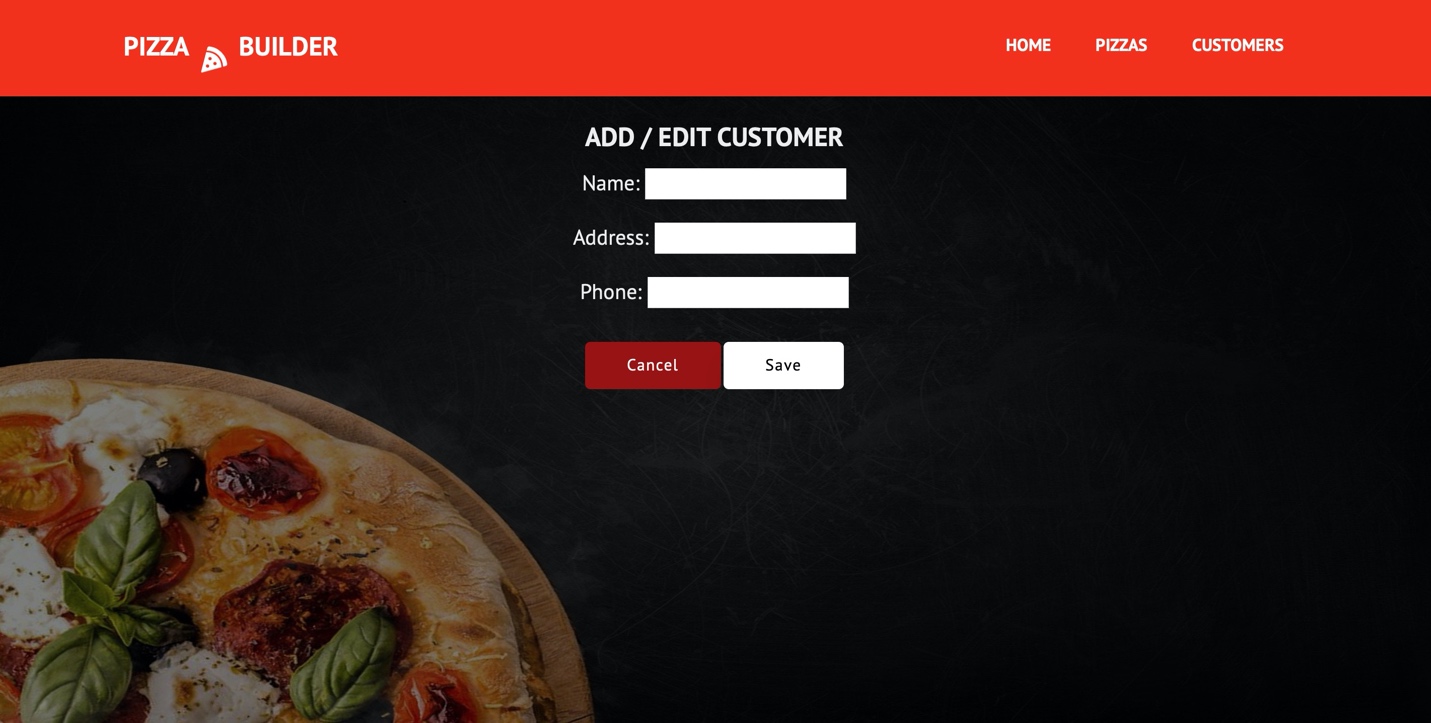
Customers page – customers.html

This page shows all customers from the database. From this page, the user has access to functions such as “edit customer”, “view pizzas”, “delete customer” and “add new customer” at the bottom.



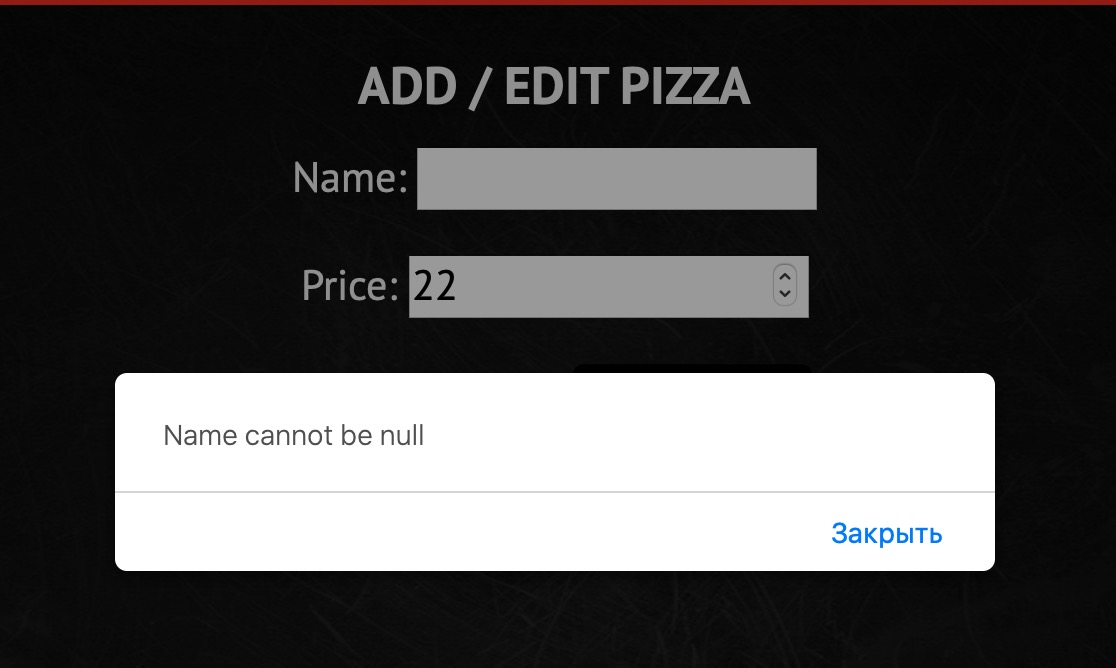
Add or edit customer page – edit-customer.html

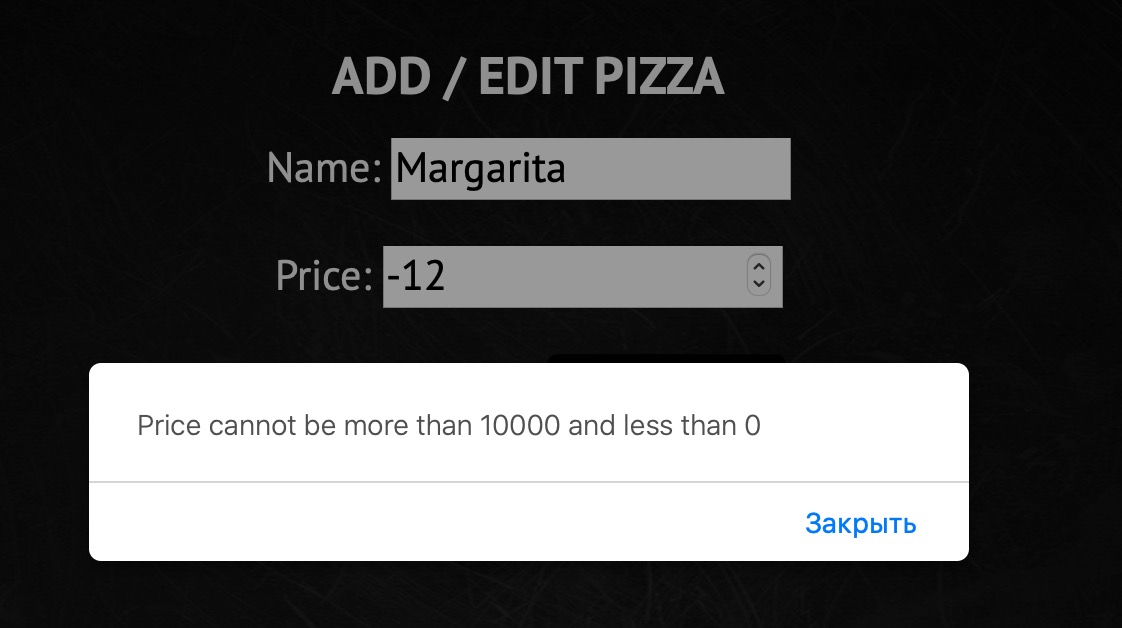
If the user clicks on a button “edit customer” in the URL in params will also appear as the customer’s id.



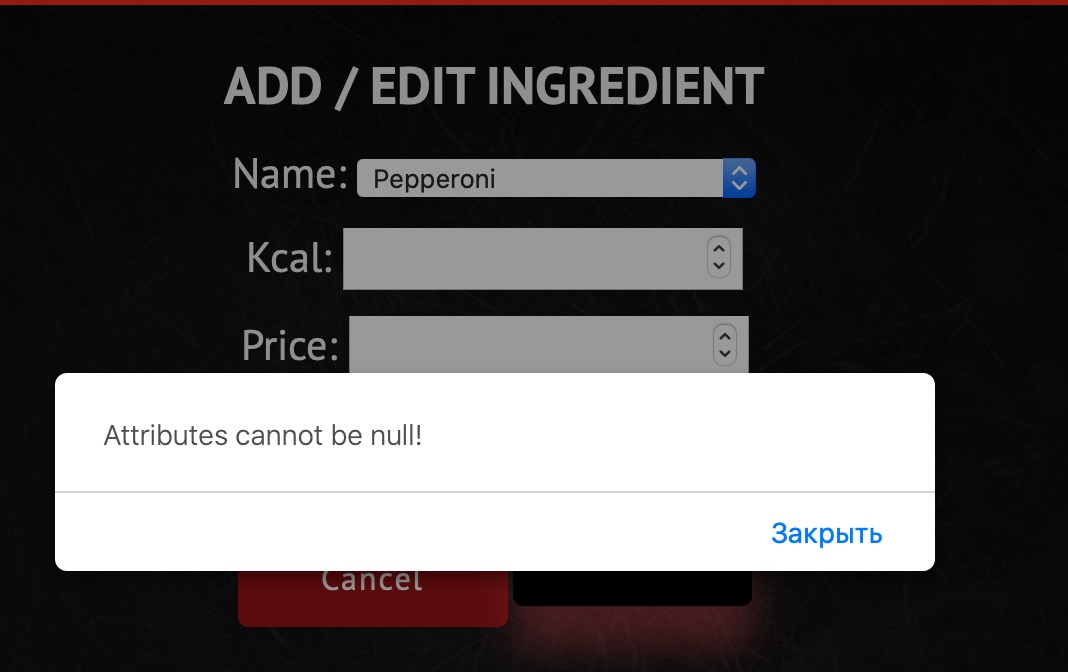
# Errors handling

In the input field, the user cannot enter a null value, also the price should be greater than 0 and lower than 10000.

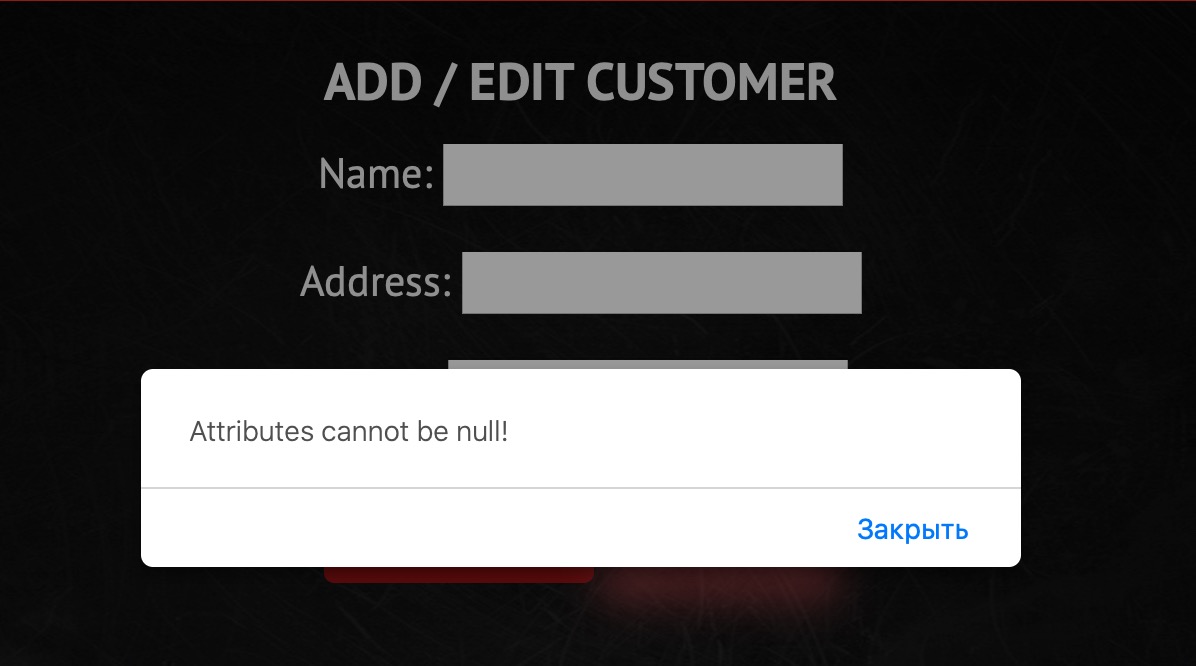




The same checking method includes ingredients.



Also, this method works for customer.

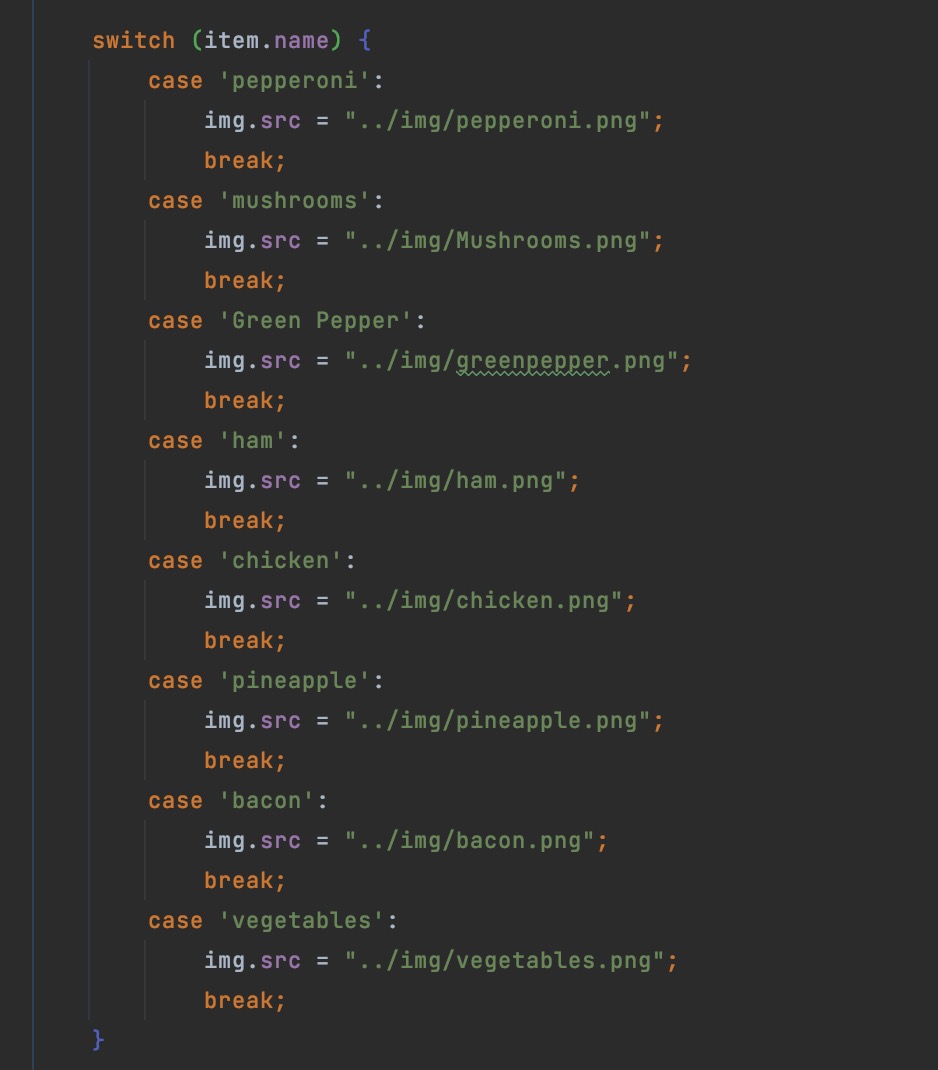


# Design choices

For adding new pizza to the customer user just choose the name of the pizza from the select option panel. The value of each option is the id of the pizza. Pizza’s id will send to fetch the PUT method.



For ingredients pictures, I used a switch, that checks the name of the item and adds an image source depending on it.



For pizzas pictures, I have conditional statements, that depend on the ingredients of the pizza and then assign an image source to the pizza item.



For each pizza I made a fetch GET method with ingredients. Then I create <ul>, <li> elements and add them to each container with pizza.

