



UNIVERSITATEA DIN CRAIOVA  
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI  
ELECTRONICĂ  
DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA  
INFORMAȚIEI



DEGREE PROJECT

**Muga Bogdan Marius**

SCIENTIFIC COORDINATOR

**Prof. Dr. Ing. Eugen Ganea**

**SEPTEMBER 2022**

CRAIOVA



UNIVERSITATEA DIN CRAIOVA  
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI  
ELECTRONICĂ  
DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA  
INFORMAȚIEI



## **LUNCHFEST**

**„Web application for Food Delivery”**

**MUGA BOGDAN-MARIUS**

**COORDONATOR ȘTIINȚIFIC**

**Prof. Dr. Ing. Eugen Ganea**

**SEPTEMBER 2022**

**CRAIOVA**

## DECLARAȚIE DE ORIGINALITATE

Subsemnatul **Muga Bogdan-Marius**, student la specializarea **Calculatoare În Limba Engleză** din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoștință de cele prezentate mai jos și că îmi asum, în acest context, originalitatea proiectului meu de licență:

- cu titlul „**LUNCHFEST – Web application for Food Delivery**”,
- coordonată de **Prof. Dr. Ing. Eugen Ganea**,
- prezentată în sesiunea **SEPTEMBRIE 2022**.

La elaborarea proiectului de licență, se consideră plagiat una dintre următoarele acțiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele și referința precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obținute sau a unor aplicații realizate de alți autori fără menționarea corectă a acestor surse,
- însușirea totală sau parțială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situații neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe și indicarea referinței într-o listă corespunzătoare la sfârșitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii și corespunzător în lista de referințe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
- precizarea referințelor poate fi omisă dacă se folosesc informații sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data,

Semnătura candidatului,



UNIVERSITATEA DIN CRAIOVA  
Facultatea de Automatică, Calculatoare și Electronică  
Departamentul de Calculatoare și Tehnologia Informației

Aprobat la data de  
.....  
Șef de departament,  
Prof. dr. ing.  
Marius BREZOVAN

## PROIECTUL DE DIPLOMĂ

Numele și prenumele studentului/-ei:	Muga Bogdan-Marius
Enunțul temei:	Web application for Food Delivery
Datele de pornire:	Every day need for survival – instead of going out to eat, you order it online
Conținutul proiectului:	<ol style="list-style-type: none"><li>1. Introduction</li><li>2. Technologies and tools used – what I used in my development to reach this stage of the project</li><li>3. Architecture Design and Implementation – the whole description/thought of creation about the project itself in a chronological order.</li><li>4. Bibliography</li></ol>
Material grafic obligatoriu:	Prezentare PowerPoint, Documentatie, Cod sursa
Consultații:	Periodice
Conducătorul științific (titlul, nume și prenume, semnătura):	Prof. Dr. Ing. Eugen Ganea
Data eliberării temei:	15.10.2021
Termenul estimat de predare a proiectului:	05.09.2022
Data predării proiectului de către student și semnătura acestuia:	



## REFERATUL CONDUCĂTORULUI ȘTIINȚIFIC

Numele și prenumele candidatului/-ei:

Muga Bogdan-Marius

Specializarea:

Calculatoare În Limba Engleză

Titlul proiectului:

LUNCHFEST – Web application for Food Delivery

Locația în care s-a realizat practica de documentare (se bifează una sau mai multe din opțiunile din dreapta):

În facultate ☐

În producție ☐

În cercetare ☐

Altă locație: [se detaliază]

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input type="checkbox"/>
Tipul proiectului		Cercetare <input type="checkbox"/>	Proiectare <input type="checkbox"/>	Realizare practică <input type="checkbox"/>	Altul [se detaliază]
Aparatul matematic utilizat		Simplu <input type="checkbox"/>	Mediu <input type="checkbox"/>	Complex <input type="checkbox"/>	Absent <input type="checkbox"/>
Utilitate		Contract de cercetare <input type="checkbox"/>	Cercetare internă <input type="checkbox"/>	Utilare <input type="checkbox"/>	Altul [se detaliază]
Redactarea lucrării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input type="checkbox"/>
Partea grafică, desene		Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input type="checkbox"/>
Realizarea practică	Contribuția autorului	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Mare <input type="checkbox"/>	Foarte mare <input type="checkbox"/>
	Complexitatea temei	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă <input type="checkbox"/>
	Analiza cerințelor	Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine <input type="checkbox"/>
	Arhitectura	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă <input type="checkbox"/>
	Întocmirea specificațiilor funcționale	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input type="checkbox"/>

	Implementarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input type="checkbox"/>
	Testarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună <input type="checkbox"/>
	Funcționarea	Da <input type="checkbox"/>	Parțială <input type="checkbox"/>	Nu <input type="checkbox"/>	
Rezultate experimentale		Experiment propriu <input type="checkbox"/>		Preluare din bibliografie <input type="checkbox"/>	
Bibliografie		Cărți	Reviste	Articole	Referințe web
Comentarii și observații					

În concluzie, se propune:

ADMITEREA PROIECTULUI <input type="checkbox"/>	RESPINGEREA PROIECTULUI <input type="checkbox"/>
---	---

Data,

Semnătura conducătorului științific,

## Project Summary

The project represents a web application for a food center company that is populated with all sorts of different foods that can be purchased by a client. The client is allowed to select a number of products that he wants to purchase.

The purpose of the project was to create a User Friendly Interface for the user for a simple task. He can do it online instead of having to go in downtown to eat. The user can just simply order it online and have it delivered.

Regarding the technologies I've used to develop tor this application is mainly from a front-end point of view. Some of the languages that I've used to achieve this are Typescript, HTML, CSS, Bootstrap using the platform Angular.

**Keywords:** Web, application, food, order, online, Angular, TypeScript, Friendly User Interface.

# SUMMARY

<b>1</b>	<b>INTRODUCTION .....</b>	<b>ERROR! BOOKMARK NOT DEFINED.0</b>
1.1	PURPOSE .....	<b>ERROR! BOOKMARK NOT DEFINED.</b>
1.2	MOTIVATION .....	- 1 -
1.3	TASKS.....	- 1 -
<b>2</b>	<b>TECHNOLOGIES AND TOOLS USED .....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>
2.1	TYPESCRIPT LANGUAGE .....	<b>ERROR! BOOKMARK NOT DEFINED.</b>
2.2	HTML(HYPERTEXT MARKUP LANGUAGE) .....	<b>ERROR! BOOKMARK NOT DEFINED.</b>
2.3	CSS(CASCADING STYLE SHEETS).....	- 3 -
2.4	GIT .....	- 3 -
2.5	ANGULAR .....	- 4 -
2.6	VISUAL STUDIO CODE .....	- 4 -
2.7	NODE.JS .....	- 4 -
2.8	BOOTSTRAP .....	- 4 -
2.9	JAVASCRIPT .....	- 4 -
<b>3</b>	<b>ARCHITECTURES DESIGN AND IMPLEMENTATION... ..</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>
3.1	PREREQUISITE.....	<b>ERROR! BOOKMARK NOT DEFINED.</b>
3.1.1	<i>Node.js .....</i>	<i>Error! Bookmark not defined.</i>
3.1.2	<i>@Angular/cli package .....</i>	<i>Error! Bookmark not defined.</i>
3.1.3	<i>@Angular-devkit/built-angular package .....</i>	<i>Error! Bookmark not defined.</i>
3.2	DESIGN AND IMPLEMENTATION .....	- 8 -
3.2.1	<i>Project Level.....</i>	<i>Error! Bookmark not defined.</i>
3.2.2	<i>Source Level(Folder).....</i>	<i>Error! Bookmark not defined.</i>
3.2.3	<i>App Level(Folder).....</i>	<i>Error! Bookmark not defined.</i>
3.2.4	<i>Shared/models level(Folder).....</i>	<i>Error! Bookmark not defined.</i>
3.3	COMPONENT FUNCTIONALITY .....	- 8 -
3.3.1	<i>Header Folder .....</i>	<i>Error! Bookmark not defined.</i>
3.3.2	<i>Home Page.....</i>	<i>Error! Bookmark not defined.</i>
3.3.3	<i>Search Folder/Component.....</i>	<i>Error! Bookmark not defined.</i>
3.3.4	<i>Tags Folder/Component .....</i>	<i>Error! Bookmark not defined.</i>
3.3.5	<i>Services/food folder/service.....</i>	<i>Error! Bookmark not defined.</i>
3.3.6	<i>Services Folder – cart service.....</i>	<i>Error! Bookmark not defined.</i>
3.3.7	<i>Food-page Folder/Component.....</i>	<i>Error! Bookmark not defined.</i>
3.3.8	<i>Cart-pate Folder/Component.....</i>	<i>Error! Bookmark not defined.</i>



3.3.9	Not-found Page/Component.....	<i>Error! Bookmark not defined.</i>
4	REFERENCES.....	ERROR! BOOKMARK NOT DEFINED.
A.	CODUL SURSĂ.....	ERROR! BOOKMARK NOT DEFINED.
C.	CD / DVD.....	- 38 -

# 1 INTRODUCTION

## 1.1 Purpose

The purpose of this project is to provide to the user with an easy to use interface for doing a simple task: to order online. The user is allowed to order whatever the food center can provide and the amount he wants.

## 1.2 Motivation

The motivation to make such an application started from the thought of a daily basis needs of an individual. It is aimed for every user that wants to order online and receive their order without having the necessity to cook it or going out.

## 1.3 Tasks

The tasks that the application is suppose to do is to receive details about the food made in code. Then the user can either select the food he likes or he can filter by one of the tags, or search it by name. Once he selected his food he can then add it to the cart and purchase it.

## 2 TECHNOLOGIES AND TOOLS USED

Most of my technologies and tools used in this project were mainly learned in college, but I've had to study more about in detail to develop the application in a simpler way.

**Technologies/Tools** used are:

### 2.1 TypeScript Language

TypeScript is a free and open source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. It is designed for the development of large applications and transpiles to JavaScript. As it is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs.

TypeScript may be used to develop JavaScript applications for both client-side and server-side execution.

Strictly for my project I've had to study a lot about this language and develop in it, so it was the most used technologies. Later on this document I will describe more about used this language, but as short overview I've used it mainly for function purposes and routing.

## **2.2 HTML(HyperText Markup Language)**

HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

The usage of this language on my project was to create all sorts of elements or add to an existing element a script(function/functionality) that was developed using the TypeScript language and then later styled using CSS.

## **2.3 CSS(Cascading Style Sheets)**

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics; enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, which reduces complexity and repetition in the structural content.

In this project the usage of CSS was to create a style for every element I've made using the HTML. These styles can represent the shape, color, background image, font, margin, how the elements are displayed and so on. Essentially anything that is different (a text for example) from a simple font starting from top-left corner in a straight line is modified using the CSS.

## **2.4 GIT**

Git is free and open source software for distributed version control: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

For my project GIT was very helpful and was also a technology that I had to learn more about since I wasn't familiarized to work with it. But once I worked more with it was so much easier because it helps me store a version of my code and if I had a problem with a code, I can go to a previous version it worked the way I wanted and continue from there with a new implementation.

## **2.5 Angular**

Angular is a TypeScript-based free and open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.

It aimed to simplify both the development and the testing of such applications by providing a framework for client-side model-view-controller (MVC) and model-

view–viewmodel (MVVM) architectures, along with components commonly used in web applications and progressive web applications.

The AngularJS framework worked by first reading the Hypertext Markup Language page, which had additional custom HTML attributes embedded into it. Angular interpreted those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of those JavaScript variables could be manually set within the code or retrieved from static or dynamic JSON resources.

In my project the Angular was the main resource to produce all the components and files that are seen in the project. Those files were first just a skeleton-code and then they had to be develop further with proper functions and adding elements. These components had to be created using a windows terminal or the Visual Studio Code terminal. My choice was to use the Windows CMD(Command Prompt).

## **2.6 Visual Studio Code**

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including Java, JavaScript, GO, Node.js, Python, C++, C.

Visual Studio Code includes basic support for most common programming languages. This basic support includes syntax highlighting, bracket matching, code

folding, and configurable snippets. Visual Studio Code also ships with IntelliSense for JavaScript, TypeScript, JSON, CSS, and HTML, as well as debugging support for Node.js. Support for additional languages can be provided by freely available extensions on the VS Code Marketplace.

Using Visual Studio Code was a personal choice that I preferred and I would recommend it because it is a User Friendly Environment where you can edit the code or can even synchronize it fast with GIT. The highlights (visualization of the color) alone on important keywords are a very important feature because you can develop easier and see things clearer.

## **2.7 Node.js**

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on a JavaScript Engine and executes JavaScript code outside a web browser, which was designed to build scalable network applications.

Node.js lets developers use JavaScript to write command line tools and for server-side scripting - running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

## 2.8 Bootstrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains HTML, CSS and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

Bootstrap is an HTML, CSS & JS Library that focuses on simplifying the development of informative web pages. The primary purpose of adding it to a web project is to apply Bootstrap's choices of color, size, font and layout to that project. As such, the primary factor is whether the developers in charge find those choices to their liking. Once added to a project, Bootstrap provides basic style definitions for all HTML elements.

## 2.9 JavaScript

JavaScript (JS) is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. All major web browsers have a dedicated JavaScript engine to execute the code on users' devices.

JavaScript is a high-level, often just-in-time compiled language. It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programming interfaces for working with text, dates, regular expressions, standard data structures, and the Document Object Model.

JavaScript engines were originally used only in web browsers, but are now core components of some servers and a variety of applications.



## 3 ARCHITECTURE DESIGN AND IMPLEMENTATION

For running this application the instructions in the “README.md” file needs to be followed thoroughly.

### 3.1 Prerequisite

There are few packages/programs that have to be installed to run the application, the steps are on the file “README.md”.

#### 3.1.1 Node.js

This is the mandatory platform that needs to be install on the PC to be able to run the application, and the packages that will be added soon.

#### 3.1.2 @Angular/cli package

This package needs to be run on the CMD(Command Prompt) or Windows Powershell using the following command “npm install -g @angular/cli”. To not have any errors while running this command the previous Prerequisite(3.1.1. ) has to be installed.

```
>npm install -g @angular/cli
```

#### 3.1.3 @Angular-devkit/built-angular package

This package can be optional if the application runs directly from your computer. If it doesn't run you have to install this package as well like the previous package. It needs to run on a CMD that is run from the current folder(the project folder). The instruction to run CMD on current folder is on step 3 from file “README.md”. The

command that needs to be running is “npm install --save-dev @angular-devkit/build-angular --force”.

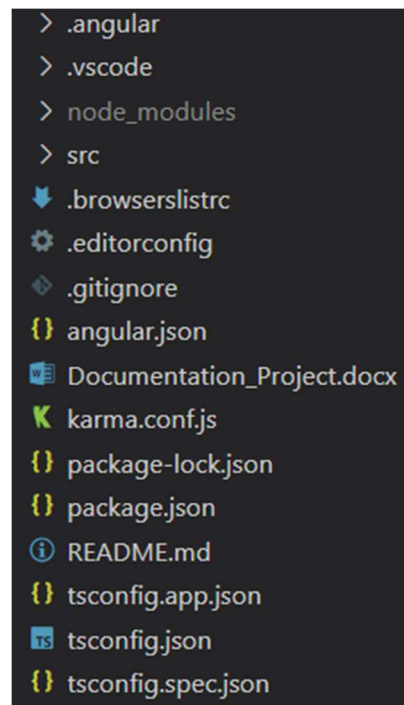
```
npm install --save-dev @angular-devkit/build-angular --force"
```

## 3.2 Design and Implementation

In this section I will present how the application was thought in a chronological order(where I will explain the components), followed by implementation and some visualization of the current page/component(where can be demonstrated).

### 3.2.1Project Level

The following image represents the first level of the project which most of them are mainly an automated generated code that represents dependencies, packages and others.



**node\_modules** – This folder contains all npm and angular packages that were generated when running the command from section 3.1.2. added with the creation of a new workspace “ng new name”

**src** – This is the main folder that will be required to run the application the way it’s suppose to. Most of the contents are made as a “**component**” which is a code skeleton for sub folder, that can be customized and can represent one part of a page or the page itself. This **component** generates a Typescript skeleton-code, HTML file and an empty CSS file.

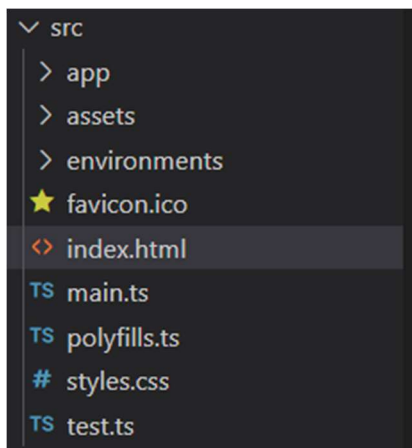
**angular.json** – this file was automatically generated and contains all sorts of information for our current application, such as properties, configuration about our workspace, development tools for the Angular CLI. Some of the configuration that are made are for example the pathing to create and find our files(components that are about to be generated or input files such as images, videos).

**package.json, package-lock.json** – this file shows the version of our tools run for this application(angular, typescript etc)

**README.md** – this file contains the instruction to run the application properly(it can be open with any file editor)

### 3.2.2Source level(folder)

In this folder the files(folders) are generated when the workspace of this project is created when using CMD(or terminal in VS code) using the command ”ng n name”.



app – In this folder contains some sub folders that I will generate it will be modified(and explained in the section above). Also it contains some of the configuration and routing files that will also be modified because some components will be generated.

assets – This folder contains all the input files such as images, videos, audios, etc. In my case I will only have images.

index.html – This file will firstly run the home page component.

test.ts – File that will run if the wants to be debugged.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Lunchfest</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <!-- <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/c
</head>
<body>
  <app-root></app-root>
  <!-- <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/
```

The rest of the files are files that contains the configuration about the following files/folders will be explained in the next section.

### 3.2.3App level(folder)

This folder represents the most important level of the whole application. It has all the implementations for the following components(only sub folders in this case), that are explained chronologically and thoroughly in section 3.3.

```
▼ app
  > cart-page
  > food-page
  > header
  > home
  > not-found
  > search
  > services
  > shared
  > tags
  TS app-routing.module.ts
  # app.component.css
  <> app.component.html
  TS app.component.spec.ts
  TS app.component.ts
  TS app.module.ts
```

cart-page ... tags – Folders that represents a whole component using the command “ng g cart-page” and can be an entire page, or it can be a part(portion) for another page(such as the header folder that is distributed on all pages).

shared/models – In this folder are described the declaration of the classes(properties) that will be used by most of the components that needs to use operations based on this lists.

app-routing-module.ts – This file represents the connection between the components and the routing of what the component will have. The routing, if it required to go to a certain page, it needs to be manually added.

```
import { CartPageComponent } from './cart-page/cart-page.component';
import { FoodPageComponent } from './food-page/food-page.component';
import { HomeComponent } from './home/home.component';
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  {path:'', component:HomeComponent},
  {path:'search/:searchItem', component:HomeComponent},
  {path:'tag/:tag', component:HomeComponent},
  {path:'food/:id', component:FoodPageComponent},
  {path:'cart-page', component:CartPageComponent}
];
```

app.component.html/ts – runs the application with the header component and home.

```
<app-header></app-header>
<router-outlet></router-outlet> <!--Home component load-->
```

app.module.ts – File that contains all the connections and imports of the application between the modules. It automatically updates when a new component is generated.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { HomeComponent } from './home/home.component';
import { RatingModule } from 'ng-starrating';
import { SearchComponent } from './search/search.component';
import { TagsComponent } from './tags/tags.component';
import { CartPageComponent } from './cart-page/cart-page.component';
import { FoodPageComponent } from './food-page/food-page.component';
import { NotFoundComponent } from './not-found/not-found.component';
```

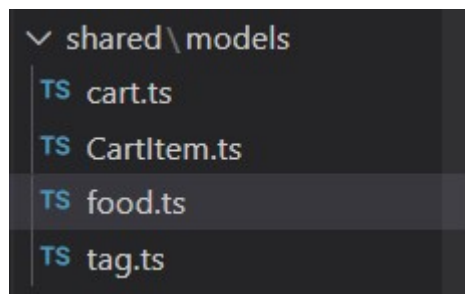
```

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    HomeComponent,
    SearchComponent,
    TagsComponent,
    CartPageComponent,
    FoodPageComponent,
    NotFoundComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule,
    RatingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

### 3.2.4 Shared/models level(folder)

In this folder is where the class declarations are created along with the properties for their respective utilization.



food.ts – This file represents the class that will be used for every detail of a food specification. It will be later initialized in a “cart.service.ts” file with given properties given in the code. In this class the “!” represents a property that for certain assures the compiler that the value is not null or undefined, therefore the “:” attribute needs to be initialized because the compiler doesn’t know if it’s going to have a null value or not, the “?:” represents a property that is optional and can be missed in the initialization, but the value will be undefined.

```
export class Foods{
  id!:number;
  price!:number;
  name!:string;
  favorite:boolean = false;
  star:number = 0;
  tags?:string[];
  imageUrl!:string;
  cookTime!:string;
  origins!:string[];
}
```

tag.ts – This file represents the class that will be used for the tag module and has the amount of times the food appears by the name of tags given in the “Foods” class.

```
export class Tag{
  name!: string;
  count!: number;
}
```

cartItem.ts – This file represents the class where a certain food will be hold temporarily such that it will be later added in the cart. It also computes the price of the item with quantity amount of what the user wants to order.

```
import { Foods } from "../food";
...
export class CartItem{
  food:Foods;
  constructor(food:Foods){
    this.food = food;
  }

  quantity:number = 1;
  get price(): number{
    return this.quantity * this.food.price;
  }
}
```

cart.ts – This file represents an array of items, CartItem(which therefore are Foods products, from food.ts) that will be stored in the cart. A total price will be shown for the whole order.

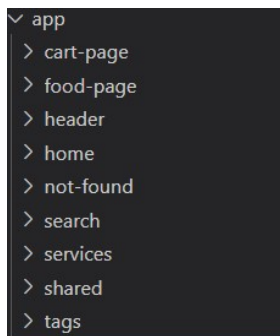
```
import { CartItem } from "../CartItem";

...
export class Cart{
  items:CartItem[] = [];

  get totalPrice(): Number{
    let totalPrice = 0;
    this.items.forEach(item => {
      totalPrice += item.price;
    });
    return totalPrice;
  }
}
```

## 3.3 Component Functionality

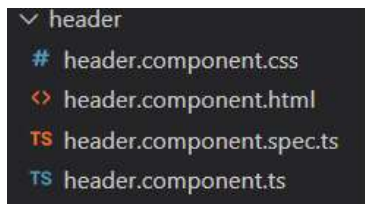
In this section I will present the chronological order of the components created, and implicit the creation of the project as well. These functionalities are made within the folder src/app. Every sub folder from this app folder is a component generated with the command “ng g component” within the internal Terminal from VS Code or CMD(Command Prompt) from Windows that will be modified to look like the page I designed. The routing of the modules/components were explained in the section 3.2.3.



### 3.3.1 Header Folder

This folder was generated with the command from above “ng g header”. This folder represents a component that will be called on every page because it has a generic information that can be used for shortcuts, such as navigation to the home page, or to the cart page. It is build with a HTML, CSS and TS file.





## header.component.html

```
<header>
  <div class="container">
    <a href="#" routerLink="/" class="logo">Lunchfest</a>
    <nav>
      <ul>
        <li>
          <a routerLink="/cart-page">Cart</a>
        </li>
      </ul>
    </nav>
  </div>
</header>
```

## header.component.css

```
header{
  background-image: url('/assets/redBackground.jpg');
  padding: 0;
  border-bottom: 1px solid red;
}
a{
  color: white;
}
a:hover{
  background-color: #e72929;
  color: black;
}
.container{
  margin: auto;
  display: flex;
```

Note how the background image is taken from the “assets” folder.



## 3.3.2 Home Page

This folder is generated like the other folder as well. It is a component that will be run the first time the application is opened. The application can be open using the terminal while within the folder using the following command “ng s -o”.

This component is built with the header component, the normal content of the home component which is a list of foods(details about them), and other components that were added within the HTML file as a module.

```

▼ home
# home.component.css
<> home.component.html
TS home.component.spec.ts
TS home.component.ts

```

```

<app-search></app-search>
<app-tags></app-tags>
<app-not-found [visible]="!foods||foods.length <= 0"></app-not-found>
<ul>
  <li *ngFor="let food of foods">
    <a routerLink="/food/{{food.id}}">
      
      <div class="content">
        <div class="name">
          {{food.name}}
        </div>
        <span class="favorite {{food.favorite?'': 'not'}}">♥</span>
        <star-rating
          [value]="food.star"
          [totalstars]="5"
          checkedcolor="yellow"
          uncheckedcolor="black"
          size="24px"
          [readonly]="true">
        </star-rating>
        <div class="product-item-footer">
          <div class="origins">
            <span *ngFor="let origin of food.origins">
              {{origin}}
            </span>
          </div>
        </div>
      </div>
    </a>
  </li>
</ul>

```

On this html file can be seen how some of the components(folders/modules) are already being used(<app-search></app-search>, <app-tags></app-tags>). These components will be described in the following sections.

Here can also be observed how we iterate through a list of elements with “\*ngFor” every element from the class foods. This operation does insert the food service that will be showed in a later section. But as a short overview, these function will take every attribute from every element of the class, that was already implemented and display it in a proper way.



Quesadilla



Mexic

\$15.00

🕒 25-30

The above picture represents just a type of food that can be ordered, with few details about it. When clicking on one picture the server will redirect us to a route for a food page that contains this specific food(id of food).

The following URL will be followed once you click on one of the pictures from the home page and will go to the food-page component.

```
localhost:4200/food/6
```

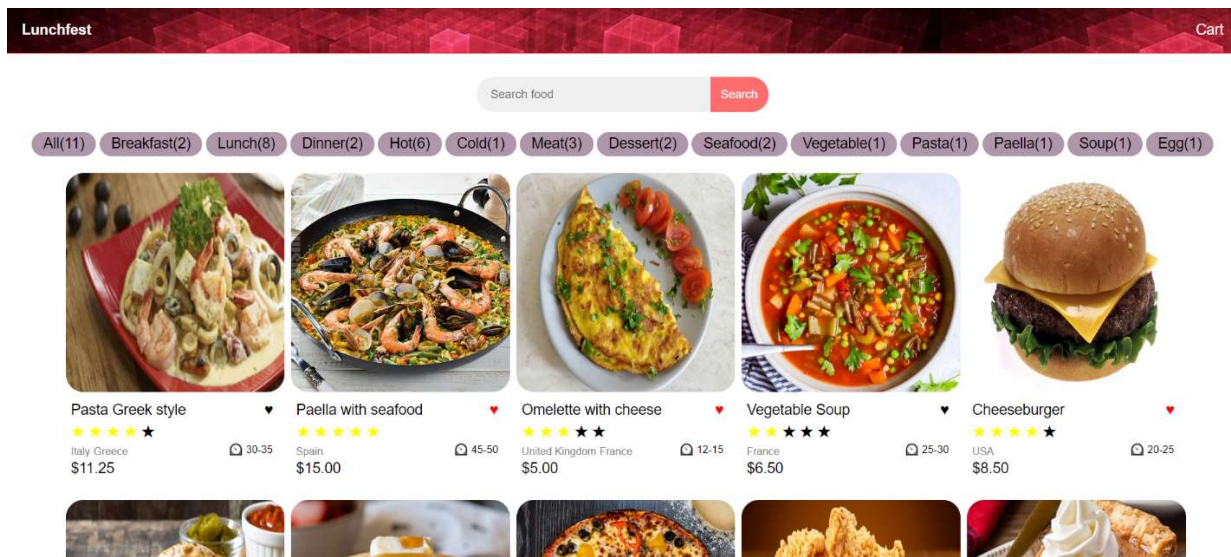
As I said above, there are components that will be used mainly in the home page such as search component and tags component. They are both similar since they both look for an item based on a filter criteria.

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {

  foods:Foods[] = [];
  constructor(private fs:FoodService, private route:ActivatedRoute) { }

  ngOnInit(): void {
    this.route.params.subscribe(params =>{
      if(params['searchItem'])
        this.foods = this.fs.getAll().filter(food => food.name.toLowerCase().includes(params['searchItem'].toLowerCase()));
      else if(params['tag'])
        this.foods = this.fs.getAllFoodByTag(params['tag']);
      else
        this.foods = this.fs.getAll();
    })
  }
}
```

The picture above is just the “home.component.ts” file that logically finds(or calls a function on the first elseif) all the food, searched by the name(it has to be a substring of the actual name of the food, and will go into the “search” component) or selecting a certain tag(which goes into the “tag” component, and it will give all the foods that has that tag).



This is a demonstration of the home page of how it should be visualized, containing all of the items/components explained above.

Also, when the application is opened this is the URL that will be routed to (the home page) and the header will route it as well wherever the user is currently on a different page.

localhost:4200

### 3.3.3 Search Folder/Component

This folder contains the search component, generated again like previous components. It is used in the home component and helps the user to find a product based on an input (food in our case) he gives to the search bar. If the food the user searched is a substring of the food name he wants then he will receive all the food names that contains that substring.

In the following picture it's shown the html file of the search component, I will not enter in detail about the CSS, because it is for styling only, and the demo will be shown at the end of this section. We can see how the HTML file will call the function `search()` that is declared in the respective `search.component.ts` file.

```

<div>
  <input type="text" placeholder="Search food"
    [(ngModel)]="searchItem" (change)="search()">
  <button (click)="search()">Search</button>
</div>

```

In the picture below is the typescript file for this component that searches for the given input from the html file and searches it in the list of the foods. After this operation is done (you click on the search button) you will be redirected to the page “localhost:4200/search/xxxxx” (where “xxxxx” is the input given by the user after clicking on the button and a number of foods will be displayed that have the substring “xxxxx” in their food name).

```

@Component({
  selector: 'app-search',
  templateUrl: './search.component.html',
  styleUrls: ['./search.component.css']
})
export class SearchComponent implements OnInit {

  searchItem:string = '';

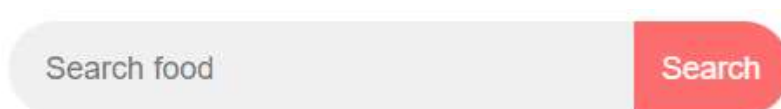
  constructor(private route:ActivatedRoute, private router:Router) {}

  ngOnInit(): void {
    this.route.params.subscribe(params =>{
      if(params['searchItem'])
        this.searchItem = params['searchItem'];
    })
  }

  search(): void{
    if(this.searchItem)
      this.router.navigateByUrl('/search/' + this.searchItem);
  }
}

```

The component can be seen as well on the home page, and can be called from there as well. You can also type in the URL “localhost:4200/search/xxxxx” instead of clicking on the search button for the desired food.



A search bar with a light gray background and a red search button. The search bar contains the placeholder text "Search food". The search button is red with the text "Search" in white.

As a demonstration I will search the food with input “sea” then input “se” to see different results.



localhost:4200/search/sea



sea

Search

Hot(6)

Cold(1)

Meat(3)

Dessert(2)

Seafood(2)



Paella with seafood



Spain

\$15.00

45-50

In the example above we can see that we found only one food that was partially named “sea”.

localhost:4200/search/se



se

Search

Dinner(2)

Hot(6)

Cold(1)

Meat(3)

Dessert(2)

Seafood(2)

Vegetable(1)

Pasta(1)



Paella with seafood



Spain

\$15.00

45-50



Omelette with cheese



United Kingdom France

\$5.00

12-15



Cheeseburger



USA

\$8.50

20-25

In the example above we can clearly see where we found more than one food with our given searches, since all of our results has the input “se” in their food name.

### 3.3.4Tags Folder/Component

Similarly created like the search component, this component is used as well in the home page to look for certain tags for a product. It has another appearance in a food page that will be explained shortly in a section below and can be used in the same purpose that I will explain now.

This component already has some tags given in code, from the foodservice.ts(which is a whole section about this implementation), but will receive the properties from the “shared/models/tag.ts” file(section 3.2.4.).

All(11) Breakfast(2) Lunch(8) Dinner(2) Hot(6) Cold(1) Meat(3) Dessert(2) Seafood(2) Vegetable(1) Pasta(1) Paella(1) Soup(1) Egg(1)

The picture above can also be seen in the home page component.

Once you select one of the tags, the function below is going to be called(from home.component.ts) and will reroute the user to the URL address “localhost:4200/tag/...” with the results of the tag he selected.

```
ngOnInit(): void {
  this.route.params.subscribe(params =>{
    if(params['searchItem'])
      this.foods = this.fs.getAll().filter(food => food.name
    else if(params['tag'])
      this.foods = this.fs.getAllFoodByTag(params['tag']);
    else
      this.foods = this.fs.getAll();
  })
}
```

All the results(products) will be shown in the above URL with having the main home component as a display for the page.

```
<div *ngIf="tags" [style.justifyContent]="justifyContent">
  <a *ngFor="let tag of tags" routerLink="/tag/{{tag.name}}">{{tag.name}}({{tag.count}})</a>
</div>
<div *ngIf="foodPageTags" [style.justifyContent]="justifyContent">
  <a *ngFor="let tag of foodPageTags" routerLink="/tag/{{tag}}">{{tag}}</a>
```

```

import { FoodService } from '../services/food/food.service';
import { Component, OnInit, Input } from '@angular/core';
import { Tag } from '../shared/models/tag';
...
@Component({
  selector: 'app-tags',
  templateUrl: './tags.component.html',
  styleUrls: ['./tags.component.css']
})
export class TagsComponent implements OnInit {
  @Input()
  foodPageTags?:string[];
  @Input()
  justifyContent:string = 'center'
  tags:Tag[] = [];

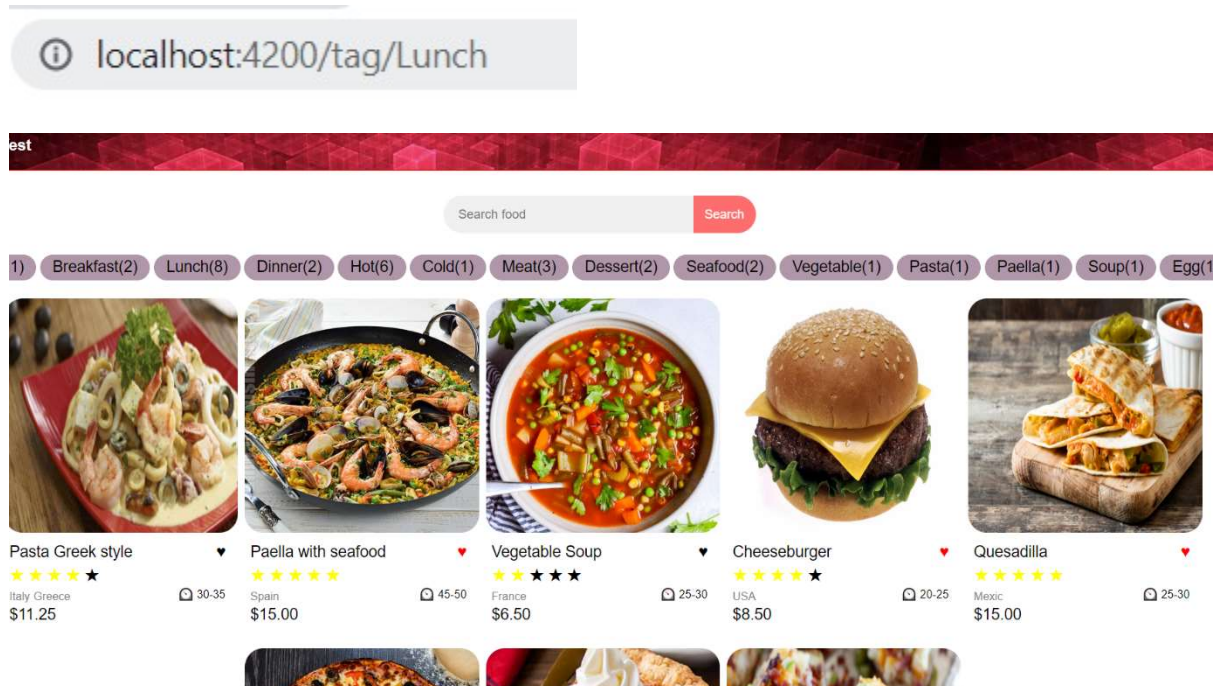
  constructor(private fs:FoodService) { }

  ngOnInit(): void {
    if(!this.foodPageTags)
      this.tags = this.fs.getAllTag();
  }
}

```

In the images above it's shown how the tag component is implemented.

I will show a quick demonstration of how it looks. When you click on the “Lunch” tag you will be redirected to the following URL address “localhost:4200/tag/Lunch” that will show the content from the home components that have the food tag “Lunch” implemented in the code. You can as well select one of the products(foods) and you will be redirected to a new page(food page component).

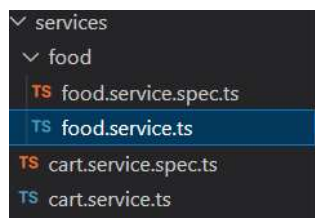




### 3.3.5 Services/food folder/service

In this folder it will be generate a service this time instead of a component using the command line from CMD “ng g s services/food/food”, g for generating a code, s for generating a service, services/food the path where the service will be located, and “/food” the name of the service. Now the difference between those two is that the service can have a multiple purpose in an application and can be used by many other components or services. The service will not have the CSS and HTML files.

As I said the service can have a multiple purpose in an application, which means I can create a lot of functions/methods in my TypeScript file that will have a logical impact over the application where the methods are used properly.



My utility for this service(“food.service.ts”) is to create a lot of functions that return the properties/details about a food(product) or even the tags about the product. Some functions can return all these details by a certain tag or id of the product.

```
@Injectable({
  providedIn: 'root'
})
export class FoodService {

  constructor() { }

  getFoodById(id:number): Foods{
    return this.getAll().find(food => food.id == id)!
  }

  getAllFoodByTag(tag:string): Foods[]{
    if(tag == 'All')
      return this.getAll();
    else
      return this.getAll().filter(food => food.tags?.includes(tag));
  }

  getAllTag():Tag[] { ...
  }
  getAll():Foods[]{
    return [
      {id: 1,
        price: 11.25,
        name: 'Pasta Greek style',
        favorite: false,
        star: 4,
        tags: ['Pasta', 'Lunch', 'Seafood', 'Hot'],
        imageUrl: 'assets/food1.jpg',
```

```

        imageUrl: 'assets/food1.jpg',
        cookTime: '30-35',
        origins: ['Italy', 'Greece']
    },
    {id: 2,
      price: 15,
      name: 'Paella with seafood',
      favorite: true,
      star: 5,
      tags: ['Paella', 'Seafood', 'Lunch', 'Hot'],
      imageUrl: 'assets/food2.jpg',
      cookTime: '45-50',
      origins: ['Spain']
    },
    {id: 3,
      price: 5,
      name: 'Omelette with cheese',
      favorite: true,

```

Besides returning the whole array of the products(food) and the tags the functions “**getAll()**” and “**getAllTag**” these items will also be generated manually with the code, and making sure to give the proper generation.

The other functions “**getFoodById()**” and “**getAllFoodByTag()**” will return a list of food items that will enter the criteria from where they’ve been called(with the given parameters). For example, in the picture below it’s an implementation of the function “**getFoodById()**” from the component FoodPage that will be explained/showed in the following section. This function makes sure that the application will get the correct food when clicked on an item from the home page.

```

export class FoodPageComponent implements OnInit {
    food!: Foods;
    constructor(private activatedRoute: ActivatedRoute, private foodService: FoodService,
      private cartService: CartService, private router: Router) {
      activatedRoute.params.subscribe((params) => {
        if(params['id'])
          this.food = foodService.getFoodById(params['id']);
      });
    }

```

### 3.3.6 Services Folder – cart service

This cart service(cart.service.ts) was created similar like the previous service, with one lower level in the app. Its purpose for this application is to add one particular food that is selected by the user and put it into the cart, to remove an already existent product from the cart, or even change the quantity of one product the user wants to order.

```

@Injectables({
  providedIn: 'root'
})
export class CartService {

  private cart: Cart = new Cart();
  constructor() {}

  addToCart(food: Foods): void {
    let cartItem = this.cart.items.find(item => item.food.id == food.id);
    if (cartItem) {
      this.changeQuantity(food.id, cartItem.quantity + 1);
      return;
    }
    this.cart.items.push(new CartItem(food));
  }

  removeFromCart(food: Foods): void {
    this.cart.items = this.cart.items.filter(item => item.food.id != food.id);
  }

  changeQuantity(foodId: number, quantity: number) {
    let cartItem = this.cart.items.find(item => item.food.id == foodId);
    if (!cartItem) return;
    cartItem.quantity = quantity;
  }

  getCart(): Cart {
    return this.cart;
  }
}

```

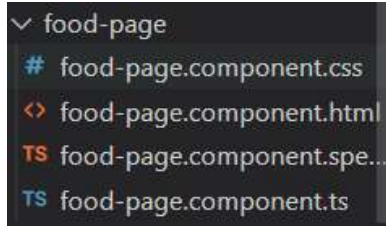
The "addToCart()" method is called when the user will access the food-page component and click on the "Add to Cart" button. To be able to access the food-page component the user has to select a food image from the home page. The method will add this food ID and details about it to the cart. A demonstration will be given in the cart-page section.

The "removeFromCart()" is called from the cart-page component when you will click on the button "Remove" from the page and, the selected food item will be successfully removed from the cart.

The "changeQuantity()" is going to be called from either the cart-page HTML page special element, or from the food-page automatically where the food is already in the cart.

### 3.3.7 Food-page folder/component

This folder contains the food-page component, generated again like previous components. This component is very important since it makes the connection between the food item(selected from the home page component) and the cart-page component. This folder displays the details(or properties) about the food that the user has chosen.



In the “food-page.component.html” file are the elements of the page enumerated, but along those elements are also few elements that need to be highlighted, such as using a specific food item with the Angular directive \*ngIf that takes the current food item from the URL routerLink(which is the ID of the current food) we are on the page. A noticeable element in this page is the “app-tags component” implemented, which takes all the tags of the current product(food) and has the same functionality as the tags from the home component. If clicked on one of the tags you will get redirected to the “localhost:4200/tags/xxxx” and receive the content from there.

Another element that was explained a little bit in the above section is the “Add to Cart” button which will call the method “addToCart()” from food-page.component.ts. This will receive all the information from the current food/product(every food being separated by its ID). Then the current food will be added to the cart, and next will access the cart-page component.

```
<app-not-found [visible]="!food" notFoundMessage="Product Not Found"></app-not-found>
<div *ngIf="food" class="container">
  <img [src]="food.imageUrl" [alt]="food.name">
  <div class="details">
    <div class="header">
      <span class="name">
        {{food.name}}
      </span>
    </div>
    <div class="tag">
      <app-tags [foodPageTags]="food.tags"></app-tags>
    </div>
    <div class="cook-time">
      <span>
        Time to cook: <strong>{{food.cookTime}}</strong> Minutes
      </span>
    </div>
    <div class="price">
      <span>
        {{food.price|currency}}
      </span>
    </div>
    <button (click)="addToCart()">Add to Cart</button>
  </div>
</div>
```

food-page.component.ts

```
@Component({
  selector: 'app-food-page',
  templateUrl: './food-page.component.html',
  styleUrls: ['./food-page.component.css']
})
export class FoodPageComponent implements OnInit {
  food!: Foods;
  constructor(private activatedRoute: ActivatedRoute, private foodService: FoodService,
    private cartService: CartService, private router: Router) {
    activatedRoute.params.subscribe((params) => {
      if(params['id'])
        this.food = foodService.getFoodById(params['id']);
    });
  }

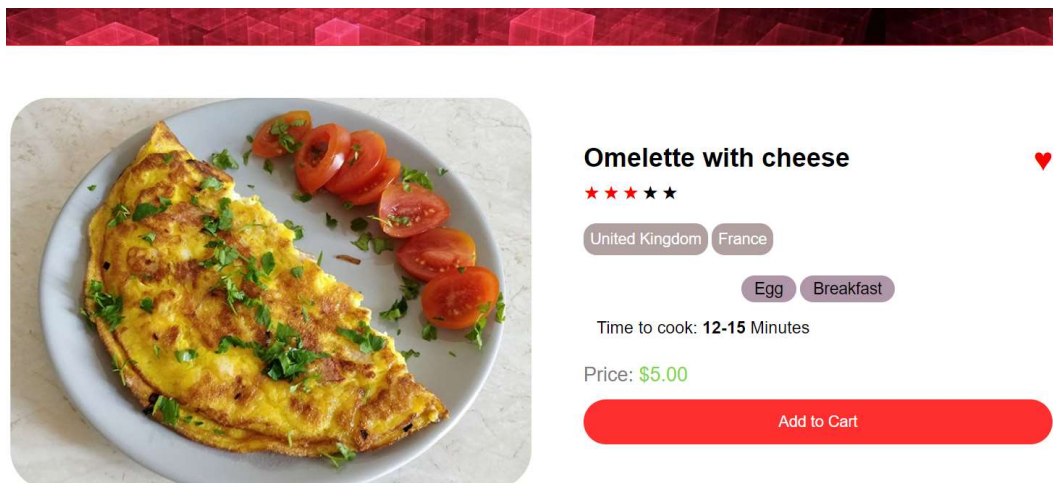
  ngOnInit(): void {
  }

  addToCart(){
    this.cartService.addToCart(this.food);
    this.router.navigateByUrl('/cart-page');
  }
}
```

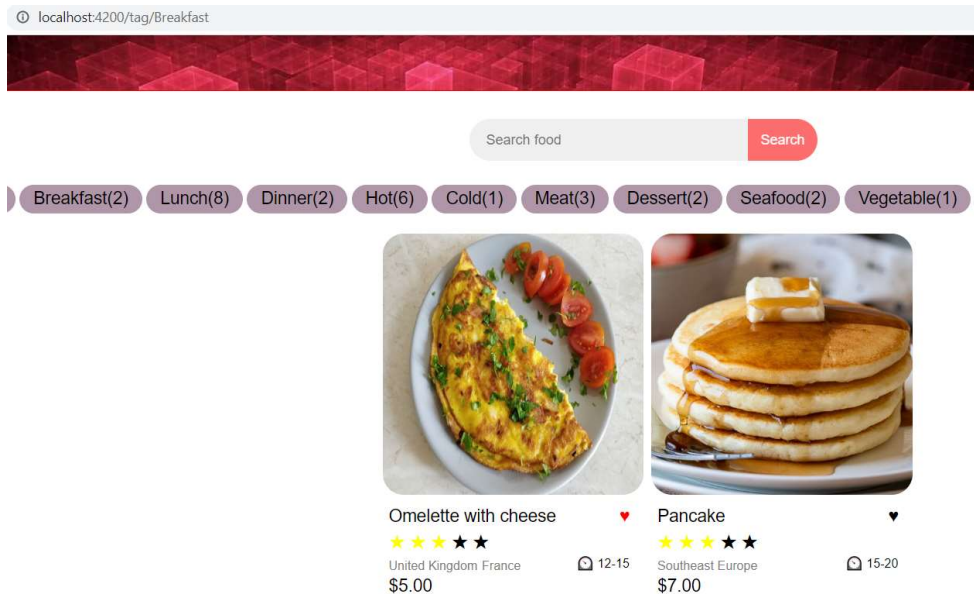
The URL of this page is “localhost:4200/food/id”(where id is the generated id within the code from a specific food).

localhost:4200/food/3

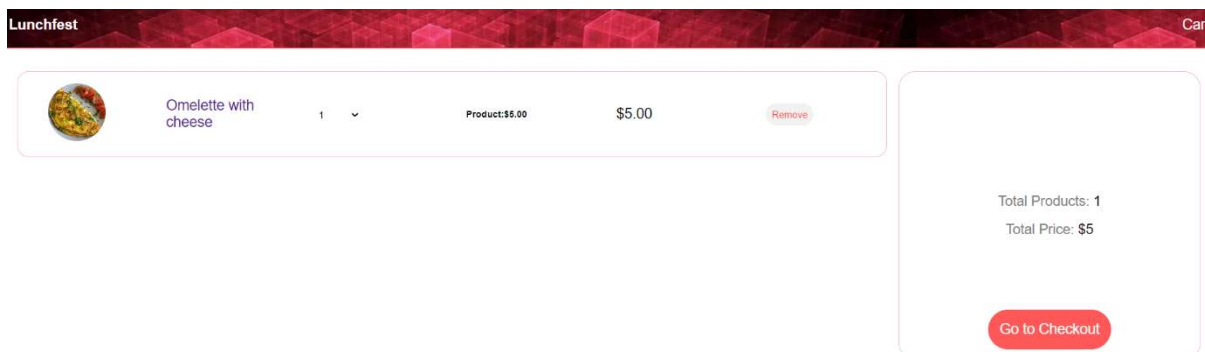
The page of the current food item(with the ID = 3 in our case) should be displayed like this:



The elements explained above(such as tags and the button will do their functionality explained above). For example if you select the “Breakfast” tag you will be redirected to the “localhost:4200/tags/Breakfast” address.



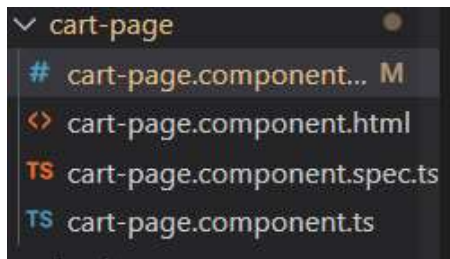
Another functionality demonstrated will be the button explained above, the “Add to Cart” button that will add the current food/product to the cart.



### 3.3.8 Cart-page folder/component

This folder contains the food-page component, generated again like previous components. This component shows the final stage of what the user wants to order. The cart element will contain all the food items that will be added using the last method explained in the food-page component.





In the `cart-page.component.ts` file it will be shown how the services are really important because on every call of these method from the `CartPage` class. Every operation/change that needs to be done will have to call the respective `cartService` methods. Therefore the cart from this page should be updated every time a product(food) is going to be removed or added to/from the cart(`setCart()`, `removeFromCart()` methods). Notice how Adding a food method is not implemented on this component, it was previously called from the `food-page` component.

```
@Component({
  selector: 'app-cart-page',
  templateUrl: './cart-page.component.html',
  styleUrls: ['./cart-page.component.css']
})
export class CartPageComponent implements OnInit {
  cart!: Cart;
  constructor(private cartService: CartService, private foodService: FoodService) {
    this.setCart();
  }

  ngOnInit(): void {
  }
  setCart(){
    this.cart = this.cartService.getCart();
  }
  removeFromCart(cartItem: CartItem){
    this.cartService.removeFromCart(cartItem.food);
    this.setCart();
  }
  changeQuantity(cartItem: CartItem, quantityInString: string){
    const quantity = parseInt(quantityInString);
    this.cartService.changeQuantity(cartItem.food.id, quantity);
    this.setCart();
  }
}
```

In the next 2 following picture from the HTML file it's shown how the page will look logically. For example, it will iterate all items from the class `cart.items`(which contains all the food stored in the cart) and will be shown like a list in the cart. The “select” tag is used for a dropdown effect on our logical thought “quantity” of the product. If the user selects a different option other than the default one then the dropdown value changes to whatever it picks and the quantity amount of the product will change as well along with the price total of the cart order.

The “remove button” from the second picture below will call the function ”removeFromCart()” to remove the food that user wants to remove from the cart and the proper functions to do so will be called along with the operations regarding the price of the card order.

```
<app-not-found [visible]="!cart || cart.items.length <= 0"
notFoundMessage="Cart is empty!" resetLinkText="Go To Homepage">
</app-not-found>
<div *ngIf="cart && cart.items.length > 0" class="container">
  <ul>
    <li *ngFor="let cartItem of cart.items">
      <div>
        <img [src]="cartItem.food.imageUrl" [alt]="cartItem.food.name">
      </div>
      <div>
        <a routerLink="/food/{{cartItem.food.id}}">
          {{cartItem.food.name}}
        </a>
      </div>
      <div>
        <select #quantitySelect (change)="changeQuantity(cartItem, quantitySelect.value)">
          <option>1</option>
          <option>2</option>
          <option>3</option>
          <option>4</option>
          <option>5</option>
        </select>
      </div>
      <div class="perItem">
        Product:{{cartItem.food.price|currency}}
      </div>
    </li>
  </ul>
  <div class="checkout">
    <div class="food-count">
      {{cart.items.length}}
    </div>
    <div class="total-price">
      ${{cart.totalPrice}}
    </div>
    <a routerLink="/checkout">Go to Checkout</a>
  </div>
</div>
```

```
<div>
  <button class="remove-button" (click)="removeFromCart(cartItem)">
    Remove
  </button>
</div>
</li>
</ul>
<div class="checkout">
  <div class="food-count">
    {{cart.items.length}}
  </div>
  <div class="total-price">
    ${{cart.totalPrice}}
  </div>
  <a routerLink="/checkout">Go to Checkout</a>
</div>
```

In the following images I will show a quick demonstration of how the application will look after the user wants to order food.

The user will start first in the home page then he will decide what food he wants to order. He seems to choose the food “Paella with seafood” from Fig. 1. Then he clicks on that image. After he is redirected to the Fig. 2 to the food page he will click on the button “Add to Cart”. Therefore he is going to be redirected to the Fig.3 and then decides to order 4, he will select the dropdown menu and will change the value 4,



from Fig. 4. The user decides that he wants to order more, he wants to order “Pumpkin Pie”. He repeats the same steps above, from Fig. 1 to Fig. 2 and his cart order will be the same as in the Fig. 5. He will then have his final order with the final price on the right side of the screen above the checkout button.

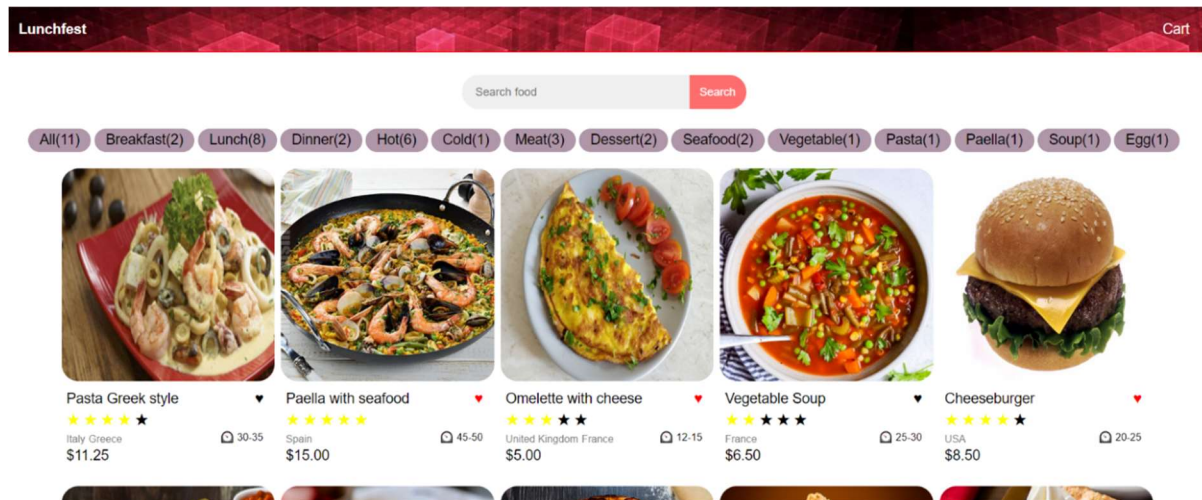


Fig. 1



Fig.2

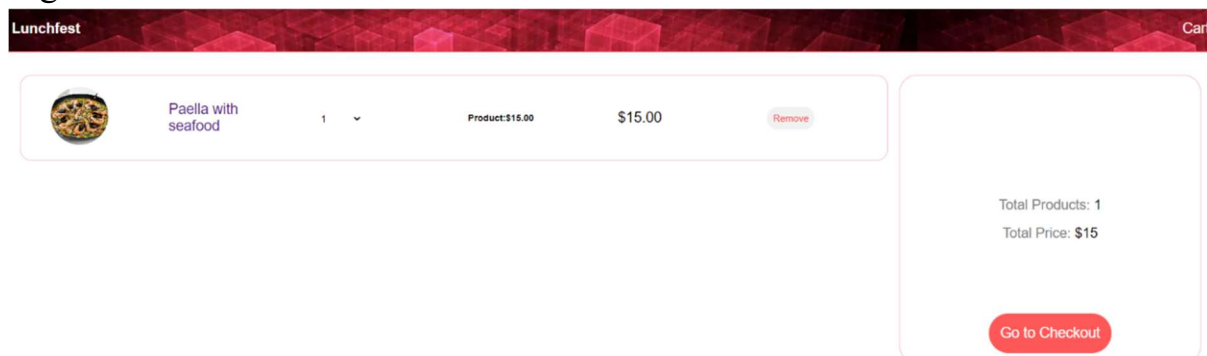


Fig. 3

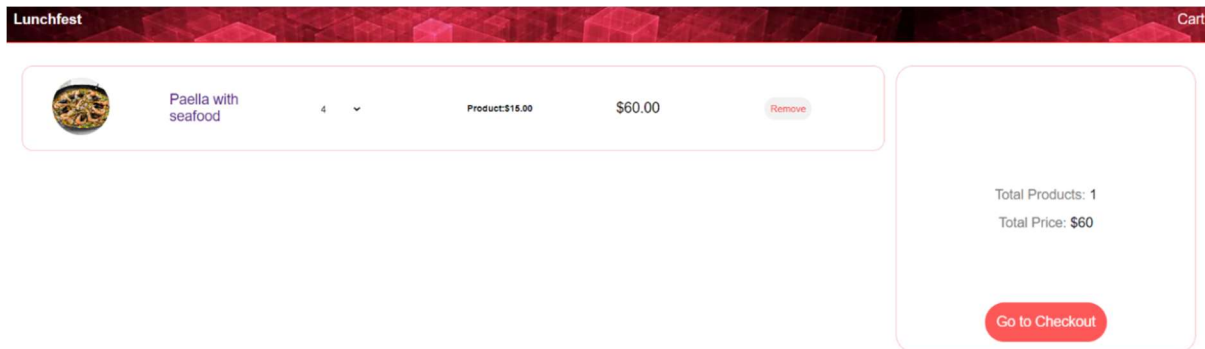


Fig. 4

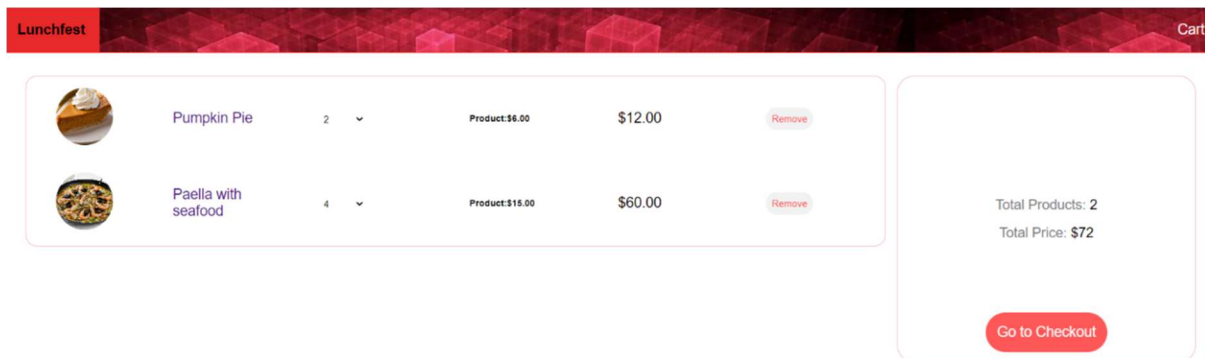


Fig. 5

### 3.3.9 Not-found page/component

This folder contains the not-found component, generated again like previous components. This component was used in certain condition until now and it was shown in the code but I didn't explain it until now. That is because it has a special condition to be met in every scenario. For example it has a basic implementation of how it will look, but it is a partial component that will be added to an already implementation of the other components that is using this component.

```

✓ not-found
# not-found.component.css
<> not-found.component.html
TS not-found.component.spec...
TS not-found.component.ts

```

```

<div *ngIf="visible">
  {{notFoundMessage}}
  <a routerLink="{{resetLinkRoute}}">{{resetLinkText}}</a>
</div>

```

```

@Component({
  selector: 'app-not-found',
  templateUrl: './not-found.component.html',
  styleUrls: ['./not-found.component.css']
})
export class NotFoundComponent implements OnInit {
  @Input()
  visible:boolean = false;
  @Input()
  notFoundMessage:string = 'Not Found!';
  @Input()
  resetLinkText:string = 'Home';
  @Input()
  resetLinkRoute:string = '/'
}

```

The above pictures are the basic implementation of these components(not-found.component.html and not-found.component.ts), they can of course be modified locally(in a different component) which I will show in a bit.

The below example is from the home page, the application will call the component not-found only when there will be no foods to be found. But how can we do that since we already have some foods implemented? Well, this can happen if we call the other component functions, or use them for the wrong reasons. For example, we try to find for a product with the search component that we know it's not going to be found but we will get the basic implementation of the not-found component.

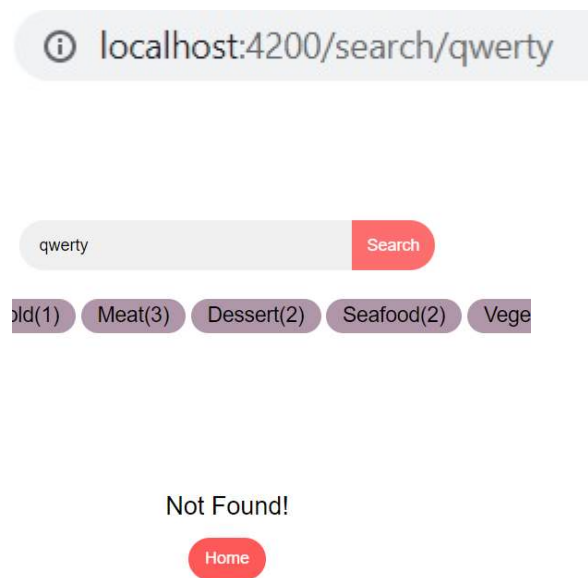
The same can happen with the tag component as well, but to find a non-existent tag can not be called from just clicking on a button from the screen, we need to modify it via the URL(change the URL to "localhost:4200/tag/xd" where xd is a non-existent tag for any of our food products).

```

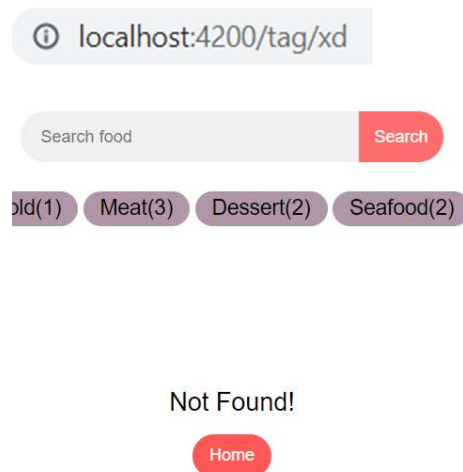
<app-tags></app-tags>
<app-not-found [visible]="!foods||foods.length <= 0"></app-not-found>
<ul>
  <li *ngFor="let food of foods">

```

In the next 2 images will be a demonstration of the search component being wrongly called:



The next 2 images will be a demonstration from the tag component by wrongfully typing the wrong tag name in the URL. There can also be noticed how the placeholder for the search component got reset:



The next 2 images will have a different message shown in the page because I modified it within the HTML page. The next images are from the `cart-page.component.html` file(or component as a whole). The not-found component will

enter in this condition only if the cart is empty(we either removed every item from the cart or we simply started with an empty cart).

```
<app-not-found [visible]="!cart || cart.items.length <= 0"
notFoundMessage="Cart is empty!" resetLinkText="Go To Homepage">
</app-not-found>
```



Cart is empty!

[Go To Homepage](#)

The next 3 images have the same explanation as from the one above. The only differences about them is that the following images are from the food-page.component.html file(or component as a whole) and the condition about it is that the user to introduce a wrong URL, such as “localhost:4200/food/14”. The application will give a not found, because the food class doesn’t have 14 products(food), or to be more precise, the food with ID=14(food.id = 14) does not exist/is implemented/generated.

```
<app-not-found [visible]="!food" notFoundMessage="Product Not Found" resetLinkText="Go To Homepage"></app-not-found>
```

ⓘ localhost:4200/food/14



Product Not Found

[Go To Homepage](#)

## 4 REFERENCES

- [1] “Angular docs” <https://angular.io/docs>
- [2] “QA for Angular, things to avoid”  
<https://www.youtube.com/watch?v=ejjln8hI14M>
- [3] “TypeScript Tutorial” <https://www.w3schools.com/typescript/>
- [4] “TypeScript Tutorial via Youtube”  
<https://www.youtube.com/watch?v=d56mG7DezGs>
- [5] “HTML elements” <https://www.w3schools.com/TAGs/default.asp>
- [6] “CSS tricks” <https://css-tricks.com/>
- [7] “Proper Angular setup” <https://angular.io/guide/setup-local>
- [8] “Angular Components(Directives) and pathing with CMD”  
<https://angular.io/cli>
- [9] “NgModules” <https://angular.io/guide/ngmodules>
- [10] “More in detail Typescript” <https://www.codecademy.com/learn/learn-typescript/modules/typescript-functions>
- [11] “Git Tutorial” <https://phoenixnap.com/kb/how-to-use-git>
- [12] “Git beginner video” <https://www.youtube.com/watch?v=USjZcfj8yxE>

## **A. CODE SOURCE**

<https://github.com/BogdanMariusMuga/food.git>

## **B. CD / DVD**

Autorul atașează în această anexă obligatorie, versiunea electronică a aplicației, a acestei lucrări, precum și prezentarea finală a tezei.

