**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.

**Note:** To run this application the instruction from the file “README.md” needs to be followed thoroughly.

1. **Introduction**
   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. **Motivation**

The motivation to make such an application started from the though 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. **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.

1. **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:

* 1. **TypeScript Language**

TypeScript is a [free and open source](https://en.wikipedia.org/wiki/Free_and_open_source) [programming language](https://en.wikipedia.org/wiki/Programming_language) developed and maintained by [Microsoft](https://en.wikipedia.org/wiki/Microsoft). It is a strict syntactical [superset](https://en.wikipedia.org/wiki/Superset) of [JavaScript](https://en.wikipedia.org/wiki/JavaScript) and adds optional [static typing](https://en.wikipedia.org/wiki/Static_typing) to the language. It is designed for the development of large applications and [transpiles](https://en.wikipedia.org/wiki/Source-to-source_compiler" \o "Source-to-source compiler) 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](https://en.wikipedia.org/wiki/Client-side) and [server-side](https://en.wikipedia.org/wiki/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.

* 1. **HTML(HyperText Markup Language)**

The HyperText Markup Language or HTML is the standard [markup language](https://en.wikipedia.org/wiki/Markup_language) for documents designed to be displayed in a [web browser](https://en.wikipedia.org/wiki/Web_browser). It can be assisted by technologies such as [Cascading Style Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets) (CSS) and [scripting languages](https://en.wikipedia.org/wiki/Scripting_language) such as [JavaScript](https://en.wikipedia.org/wiki/JavaScript). [Web browsers](https://en.wikipedia.org/wiki/Web_browser) receive HTML documents from a [web server](https://en.wikipedia.org/wiki/Web_server) or from local storage and [render](https://en.wikipedia.org/wiki/Browser_engine) the documents into multimedia web pages. HTML describes the structure of a [web page](https://en.wikipedia.org/wiki/Web_page) [semantically](https://en.wikipedia.org/wiki/Semantic_Web) 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.

* 1. **CSS(Cascading Style Sheets)**

Cascading Style Sheets (CSS) is a [style sheet language](https://en.wikipedia.org/wiki/Style_sheet_language) used for describing the [presentation](https://en.wikipedia.org/wiki/Presentation_semantics) of a document written in a [markup language](https://en.wikipedia.org/wiki/Markup_language) such as [HTML](https://en.wikipedia.org/wiki/HTML) or [XML](https://en.wikipedia.org/wiki/XML). CSS is a cornerstone technology of the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), alongside HTML and [JavaScript](https://en.wikipedia.org/wiki/JavaScript).

CSS is designed to enable the separation of presentation and content, including [layout](https://en.wikipedia.org/wiki/Page_layout), [colors](https://en.wikipedia.org/wiki/Color), and [fonts](https://en.wikipedia.org/wiki/Typeface).[[3]](https://en.wikipedia.org/wiki/CSS#cite_note-3) This separation can improve content [accessibility](https://en.wikipedia.org/wiki/Accessibility); provide more flexibility and control in the specification of presentation characteristics; enable multiple [web pages](https://en.wikipedia.org/wiki/Web_page) 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. Essentiality 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.

* 1. **GIT**

Git is [free and open source software](https://en.wikipedia.org/wiki/Free_and_open_source_software) for [distributed version control](https://en.wikipedia.org/wiki/Distributed_version_control): tracking changes in any set of [files](https://en.wikipedia.org/wiki/Computer_file), usually used for coordinating work among [programmers](https://en.wikipedia.org/wiki/Programmer) collaboratively developing [source code](https://en.wikipedia.org/wiki/Source_code) during [software development](https://en.wikipedia.org/wiki/Software_development). Its goals include speed, [data integrity](https://en.wikipedia.org/wiki/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.

* 1. **Angular**

Angular is a [TypeScript](https://en.wikipedia.org/wiki/TypeScript)-based [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) [web application framework](https://en.wikipedia.org/wiki/Web_framework) led by the Angular Team at [Google](https://en.wikipedia.org/wiki/Google) and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built [AngularJS](https://en.wikipedia.org/wiki/AngularJS).

It aimed to simplify both the development and the [testing](https://en.wikipedia.org/wiki/Software_testing) of such applications by providing a framework for client-side [model–view–controller](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) (MVC) and [model–view–viewmodel](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel) (MVVM) architectures, along with components commonly used in [web applications](https://en.wikipedia.org/wiki/Web_application) and [progressive web applications](https://en.wikipedia.org/wiki/Progressive_web_application).

The AngularJS framework worked by first reading the [Hypertext Markup Language](https://en.wikipedia.org/wiki/Hypertext_Markup_Language) page, which had additional custom [HTML attributes](https://en.wikipedia.org/wiki/HTML_attribute) embedded into it. Angular interpreted those attributes as [directives](https://en.wikipedia.org/wiki/Directive_(programming)) to bind input or output parts of the page to a model that is represented by standard JavaScript [variables](https://en.wikipedia.org/wiki/Variable_(computer_science)). The values of those JavaScript variables could be manually set within the code or retrieved from static or dynamic [JSON](https://en.wikipedia.org/wiki/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 Studii Code terminal. My choice was to use the Windows CMD(Command Prompt).

* 1. **Visual Studio Code**

Visual Studio Code, also commonly referred to as VS Code, is a [source-code editor](https://en.wikipedia.org/wiki/Source-code_editor) made by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) for [Windows](https://en.wikipedia.org/wiki/Windows), [Linux](https://en.wikipedia.org/wiki/Linux) and [macOS](https://en.wikipedia.org/wiki/MacOS). Features include support for [debugging](https://en.wikipedia.org/wiki/Debugging), [syntax highlighting](https://en.wikipedia.org/wiki/Syntax_highlighting), [intelligent code completion](https://en.wikipedia.org/wiki/Intelligent_code_completion), [snippets](https://en.wikipedia.org/wiki/Snippet_(programming)), [code refactoring](https://en.wikipedia.org/wiki/Code_refactoring), and embedded [Git](https://en.wikipedia.org/wiki/Git). Users can change the [theme](https://en.wikipedia.org/wiki/Theme_(computing)), [keyboard shortcuts](https://en.wikipedia.org/wiki/Keyboard_shortcut), preferences, and install [extensions](https://en.wikipedia.org/wiki/Plug-in_(computing)) that add additional functionality.

Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [JavaScript](https://en.wikipedia.org/wiki/JavaScript), [Go](https://en.wikipedia.org/wiki/Go_(programming_language)), [Node.js](https://en.wikipedia.org/wiki/Node.js), [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [C++](https://en.wikipedia.org/wiki/C%2B%2B), [C](https://en.wikipedia.org/wiki/C_(programming_language)).

Visual Studio Code includes basic support for most common programming languages. This basic support includes [syntax highlighting](https://en.wikipedia.org/wiki/Syntax_highlighting), [bracket matching](https://en.wikipedia.org/wiki/Bracket_matching), [code folding](https://en.wikipedia.org/wiki/Code_folding), and configurable snippets. Visual Studio Code also ships with [IntelliSense](https://en.wikipedia.org/wiki/Intelligent_code_completion) for JavaScript, TypeScript, [JSON](https://en.wikipedia.org/wiki/JSON), [CSS](https://en.wikipedia.org/wiki/CSS), and [HTML](https://en.wikipedia.org/wiki/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 an 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.

* 1. **Node.js**

Node.js is an [open-source](https://en.wikipedia.org/wiki/Open-source_software), [cross-platform](https://en.wikipedia.org/wiki/Cross-platform), [back-end](https://en.wikipedia.org/wiki/Front_end_and_back_end) [JavaScript](https://en.wikipedia.org/wiki/JavaScript) [runtime environment](https://en.wikipedia.org/wiki/Runtime_system) that runs on a [JavaScript Engine](https://en.wikipedia.org/wiki/JavaScript_Engine) and executes JavaScript code outside a [web browser](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Server-side_scripting) - running scripts server-side to produce [dynamic web page](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Web_application) development around a single programming language, rather than different languages for server-side and client-side scripts.

* 1. **Bootstrap**

Bootstrap is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source) [CSS framework](https://en.wikipedia.org/wiki/CSS_framework) directed at responsive, [mobile-first](https://en.wikipedia.org/wiki/Responsive_web_design#Mobile_first,_unobtrusive_JavaScript,_and_progressive_enhancement) [front-end web development](https://en.wikipedia.org/wiki/Front-end_web_development). It contains [HTML](https://en.wikipedia.org/wiki/HTML), [CSS](https://en.wikipedia.org/wiki/CSS) and [JavaScript](https://en.wikipedia.org/wiki/JavaScript)-based design templates for [typography](https://en.wikipedia.org/wiki/Web_design#Typography), [forms](https://en.wikipedia.org/wiki/Form_(HTML)), [buttons](https://en.wikipedia.org/wiki/Button_(computing)#HTML), [navigation](https://en.wikipedia.org/wiki/Web_navigation#Local_website_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](https://en.wikipedia.org/wiki/HTML_element).

* 1. **JavaScript**

JavaScript  (JS) is a [programming language](https://en.wikipedia.org/wiki/Programming_language) that is one of the core technologies of the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), alongside [HTML](https://en.wikipedia.org/wiki/HTML) and [CSS](https://en.wikipedia.org/wiki/CSS). All major [web browsers](https://en.wikipedia.org/wiki/Web_browser) have a dedicated [JavaScript engine](https://en.wikipedia.org/wiki/JavaScript_engine) to execute the [code](https://en.wikipedia.org/wiki/Source_code) on [users](https://en.wikipedia.org/wiki/User_(computing))' devices.

JavaScript is a [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), often [just-in-time compiled](https://en.wikipedia.org/wiki/Just-in-time_compilation) language. It has [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing), [prototype-based](https://en.wikipedia.org/wiki/Prototype-based_programming) [object-orientation](https://en.wikipedia.org/wiki/Object-oriented_programming), and [first-class functions](https://en.wikipedia.org/wiki/First-class_function). It is [multi-paradigm](https://en.wikipedia.org/wiki/Programming_paradigm), supporting [event-driven](https://en.wikipedia.org/wiki/Event-driven_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming), and [imperative](https://en.wikipedia.org/wiki/Imperative_programming) [programming styles](https://en.wikipedia.org/wiki/Programming_paradigm). It has [application programming interfaces](https://en.wikipedia.org/wiki/Application_programming_interface) for working with text, dates, [regular expressions](https://en.wikipedia.org/wiki/Regular_expression), standard [data structures](https://en.wikipedia.org/wiki/Data_structure), and the [Document Object Model](https://en.wikipedia.org/wiki/Document_Object_Model).

JavaScript engines were originally used only in web browsers, but are now core components of some [servers](https://en.wikipedia.org/wiki/Server_(computing)) and a variety of [applications](https://en.wikipedia.org/wiki/Application_software).

1. **Architecture Design and Implementation**

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

* 1. **Prerequisite**

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

* + 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.

* + 1. **@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.



* + 1. **@Angular-devkit/build-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”.

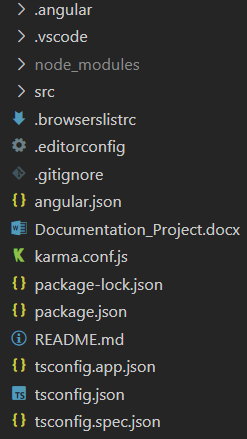


* 1. **De****sign and Implementation**

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).

* + 1. **Project 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.

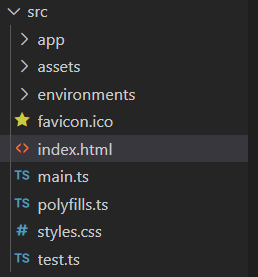
**angular.json** – this file was automatically generated and contains all sorts of information for our current application, such as properties, configuration about our workspace, development tools for the Agular CLI. Some of the configuration that are made are for example the pathing to create and find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

* + 1. **Source 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.

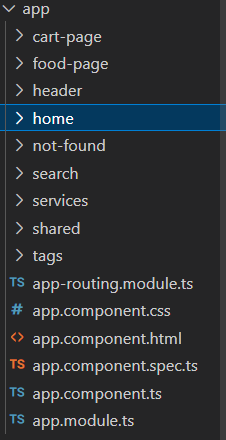
test.ts – File that will run if the wants to be debugged.



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

* + 1. **App level(folder)**

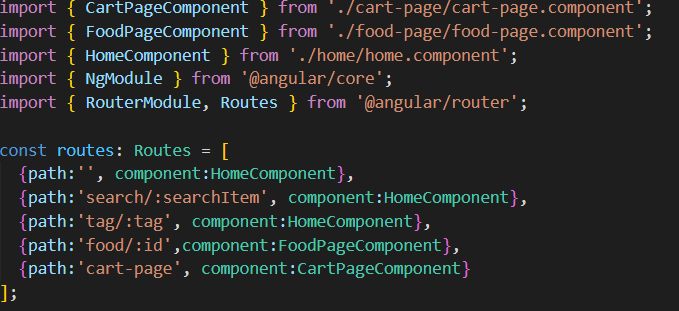
This folder represents the most important level of the whole application. It has all the implementations for the following components(only sub folders in this case), that are explained chronologically and thoroughly in section 3.3.



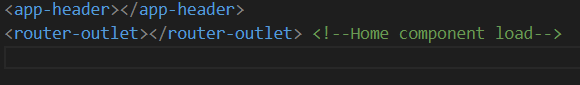
cart-page … tags – Folders that represents a whole component using the command “ng g cart-page” and can be an entire page, or it can be a part(portion) for another page(such as the header folder that is distributed on all pages).

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.

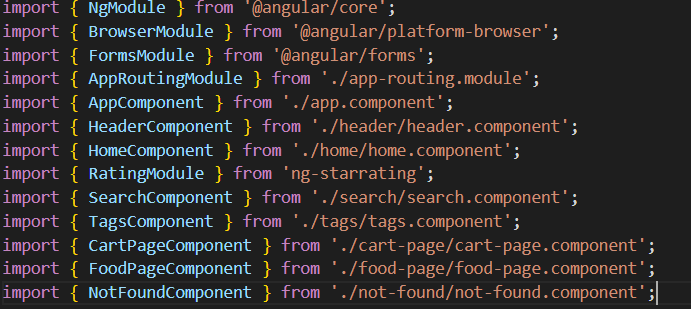
app-routing-module.ts – This file represents the connection between the components and the routing of what the component will have. The routing, if it required to go to a certain page, it needs to be manually added.

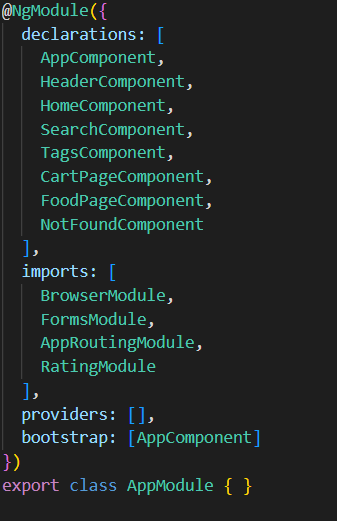


app.component.html/ts – runs the application with the header component and home.



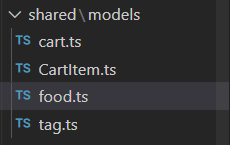
app.module.ts – File that contains all the connections and imports of the application between the modules. It automatically updates when a new component is generated.



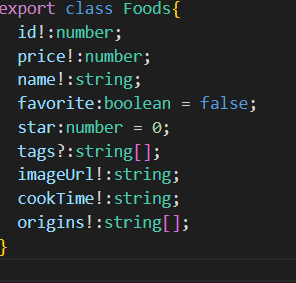


* + 1. **Shared/models level(folder)**

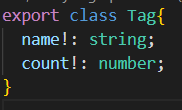
In this folder is where the class declarations are created along with the properties for their respective utilization.



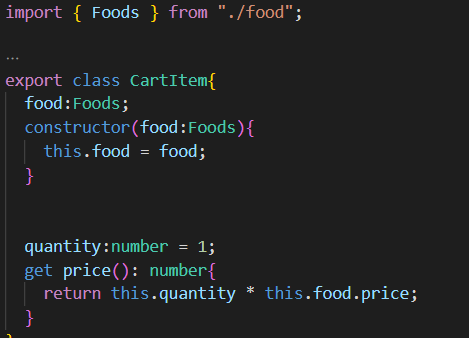
food.ts – This file represents the class that will be used for every detail of a food specification. It will be later initialized in a “cart.service.ts” file with given properties given in the code. In this class the “!:” represents a property that for certain assures the compiler that the value is not 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.



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.



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.

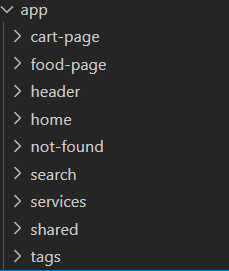


cart.ts – This file represents an array of items, CartItem(which therefore are Foods products, from food.ts) that will be stored in the cart. A total price will be shown for the whole order.



* 1. **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.



* + 1. **Header folder**

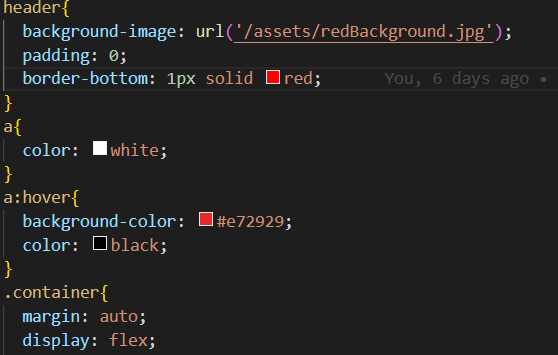
This folder was generated with the command from above “ng g header”. This folder represents a component that will be called on every page because it has a generic information that can be used for shortcuts, such as navigation to the home page, or to the cart page. It is build with a HTML, CSS and TS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



HTML



CSS



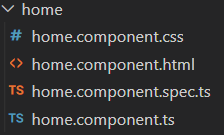
Note how the background image is taken from the “assets” folder.



* + 1. **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”.

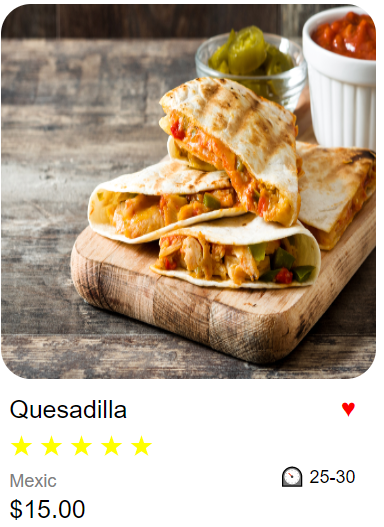
This component is built with the header component, the normal content of the home component which is a list of foods(details about them), and other components that were added within the HTML file as a module.





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.



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).

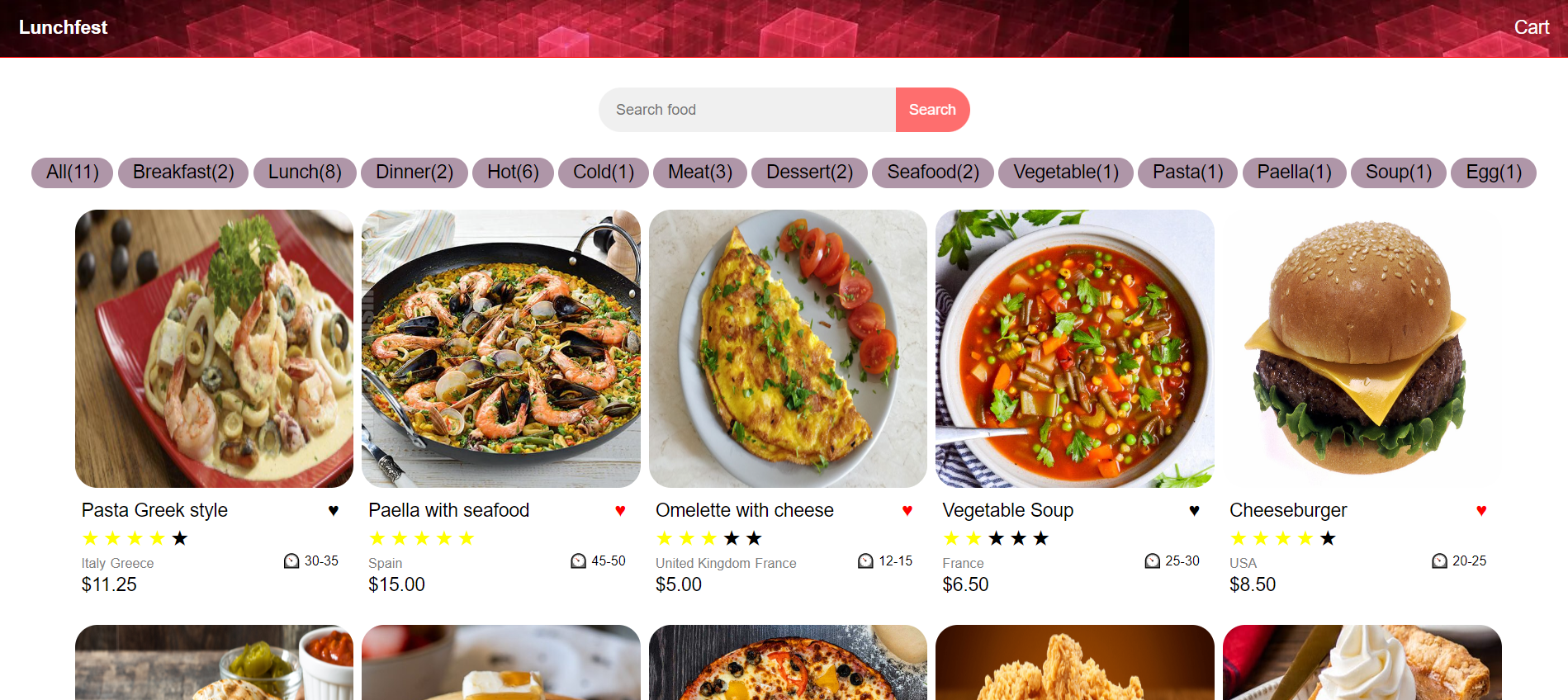
The following URL will be followed once you click on one of the pictures from the home page and will go to the food-page component.



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.



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.

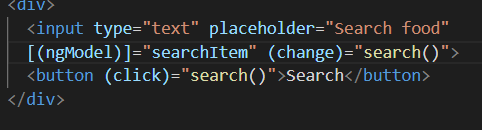
Also, when the application is opened this is the URL that will be routed to(the home page) and the header will route it as well wherever the user is currently on a different page.



* + 1. **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.



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).

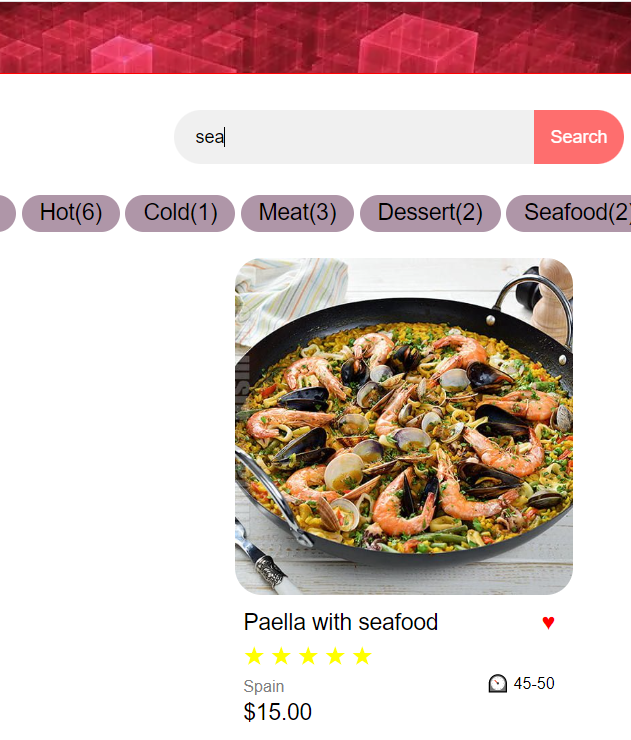


The component can be seen as well on the home page, and can be called from there as well. You can also type in the URL “localhost:4200/search/xxxxx” instead of clicking on the search button for the desired food.



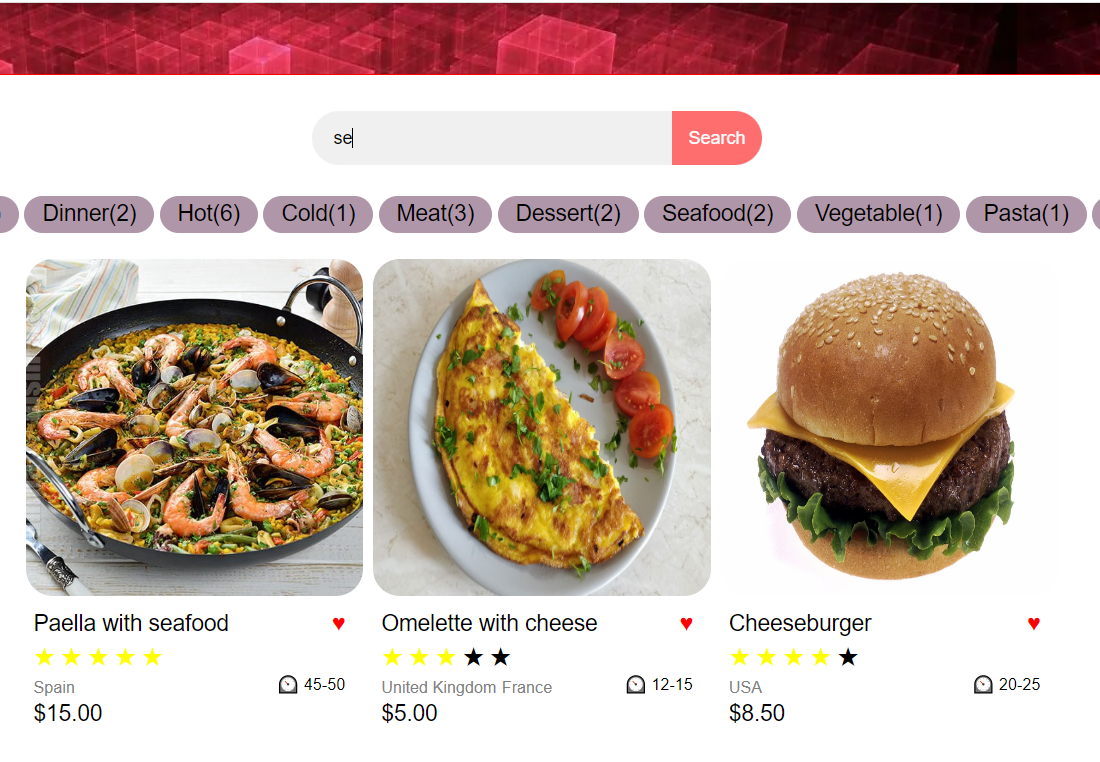
As a demonstration I will search the food with input “sea” then input “se” to see different results.





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





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.

* + 1. **Tags folder/component**

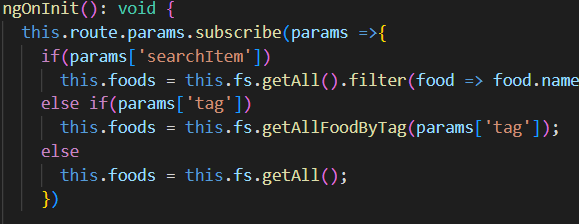
Similarly created like the search component, this component is used as well in the home page to look for certain tags for a product. It has another appearance in a food page that will be explained shortly in a section below and can be used in the same purpose that I will explain now.

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.).

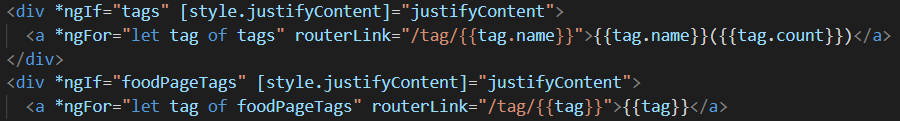


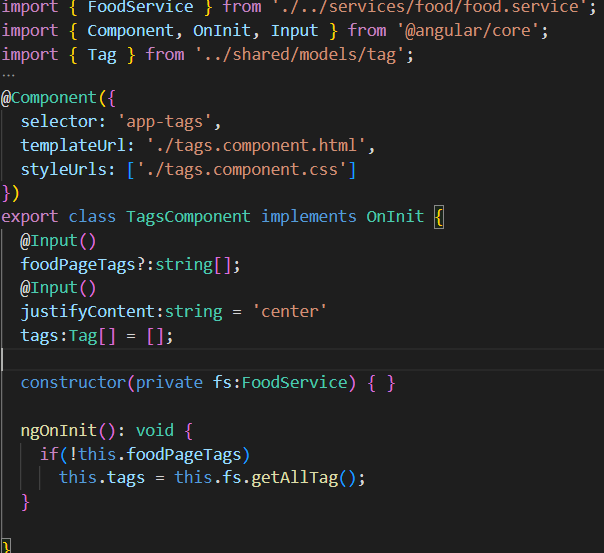
The picture above can also be seen in the home page component.

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.



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

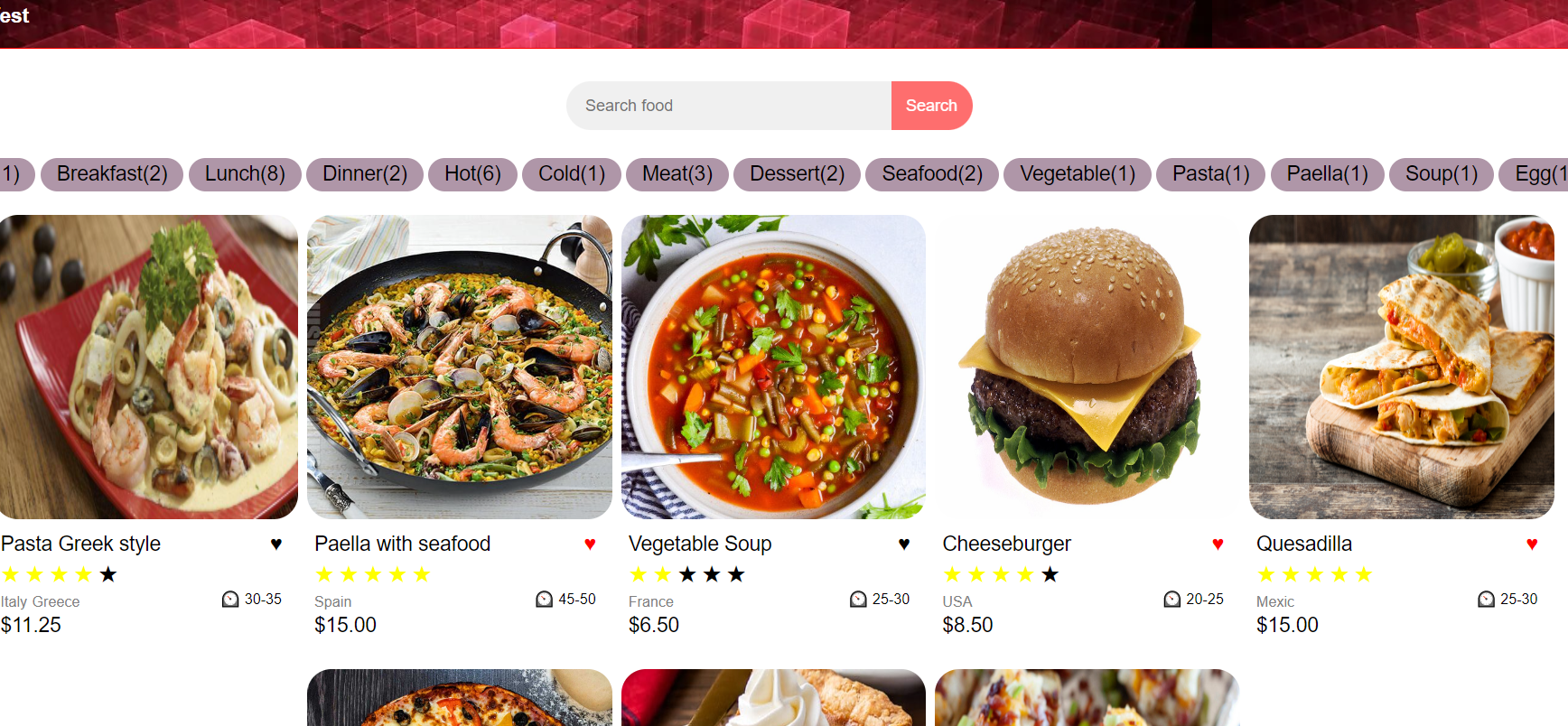




In the images above it’s shown how the tag component is implemented.

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).

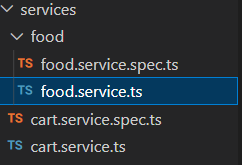




* + 1. **Services/food folder/service**

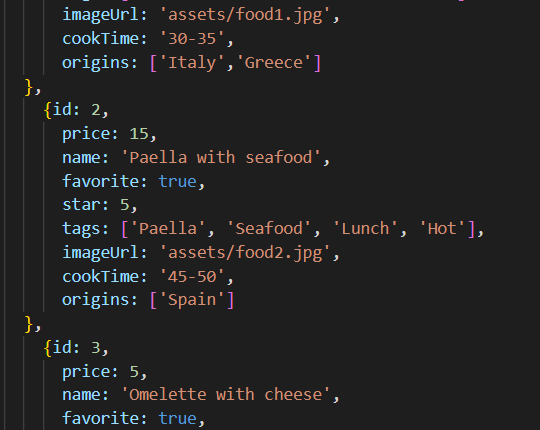
In this folder it will be generate a service this time instead of a component using the command line from CMD “ng g s services/food/food”, g for generating a code, s for generating a service, services/food the path where the service will be located, and “/food” the name of the service. Now the difference 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.



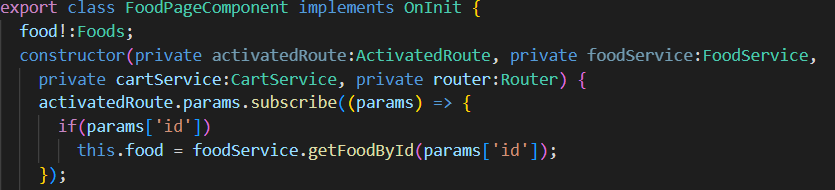
My utility for this service(“food.service.ts”) is to create a lot of functions that return the properties/details about a food(product) or even the tags about the product. Some functions can return all these details by a certain tag or id of the product.





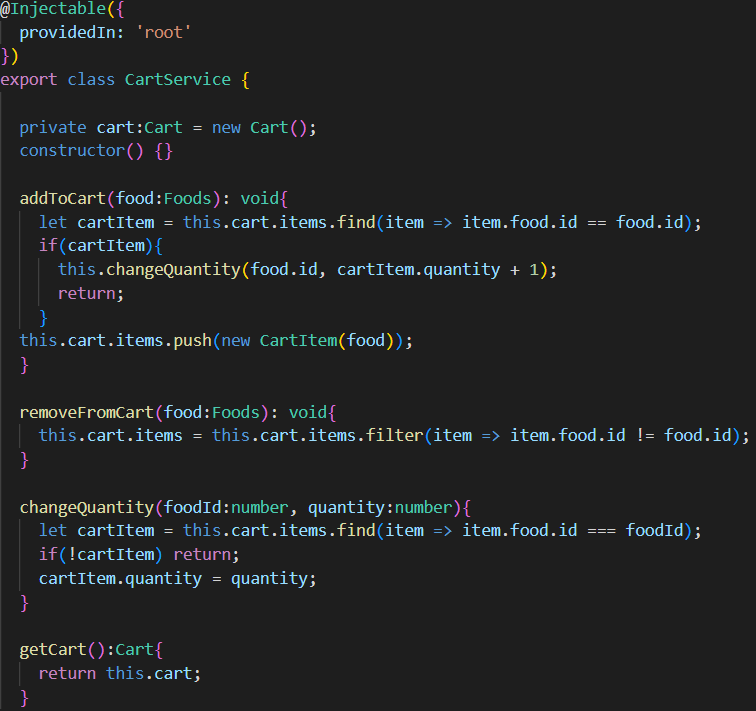
Besides returning the whole array of the products(food) and the tags the functions “**getAll()**” and “**getAllTag**” these items will also be generated manually with the code, and making sure to give the proper generation.

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.



* + 1. **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.



The ”addToCart()” method is called when the 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.

* + 1. **Food-page folder/component**

This folder contains the food-page component, generated again like previous components.