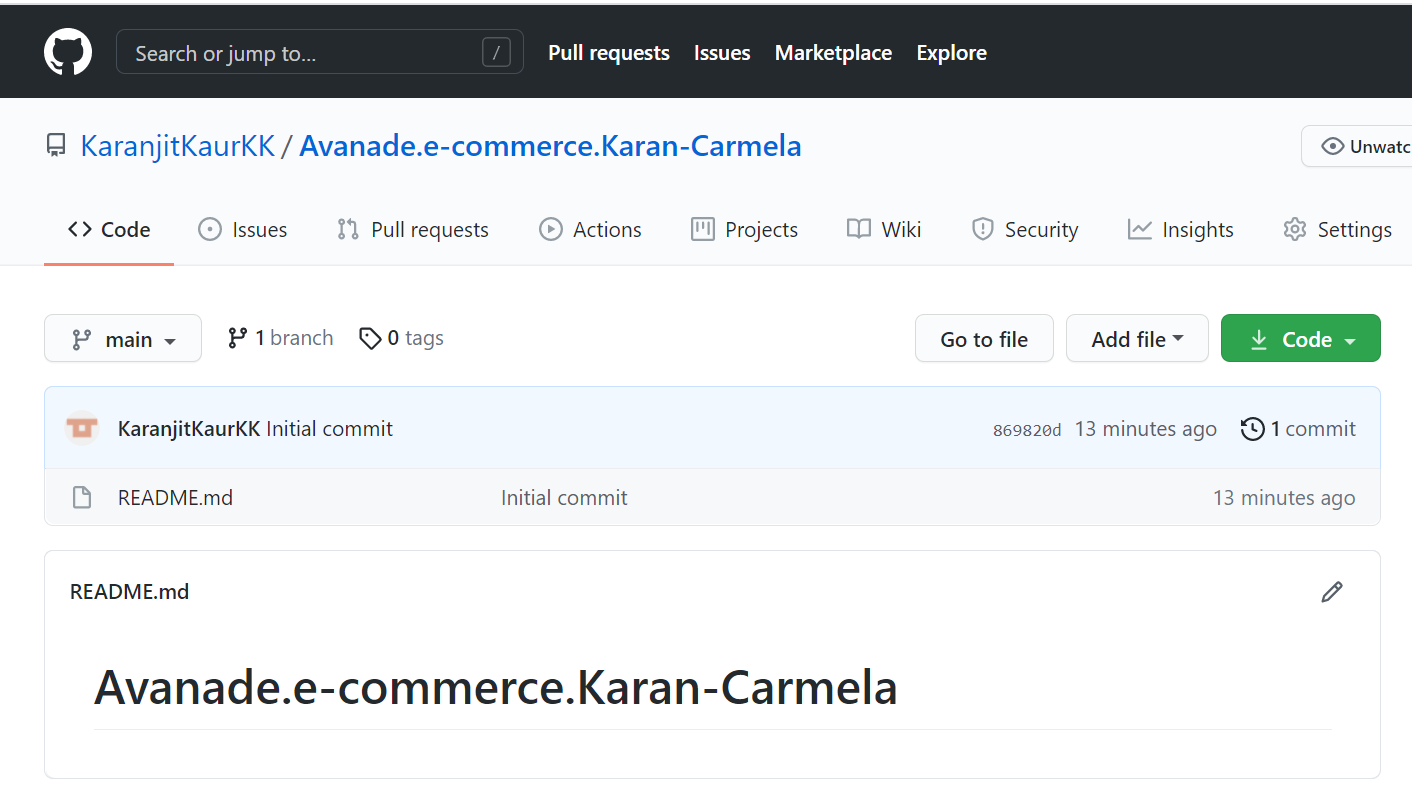
**Beervanade: by Karan & Carmela**

SCENARIO: we are creating the web app for an ecommerce of beers, where the users can find handcrafted beers. The user opens the web app and if he is logged he can see his name and surname, but in this case, for simplicity, the user is already logged. He can navigate thought various pages: catalog where there are products, and he can search with some filters like the input box for the name of beers and the select by category. He can add stuff to the cart, and there is a badge with a counter that display the quantity in the cart.Finally there are also two pages: cart and profile where there are displayed user’s details and cart’s details. (unfortunately these two pages are not implemented).

We started with an empty repository on GitHub (we changed the name of repository in later on in Beervanade\_ecommerce)

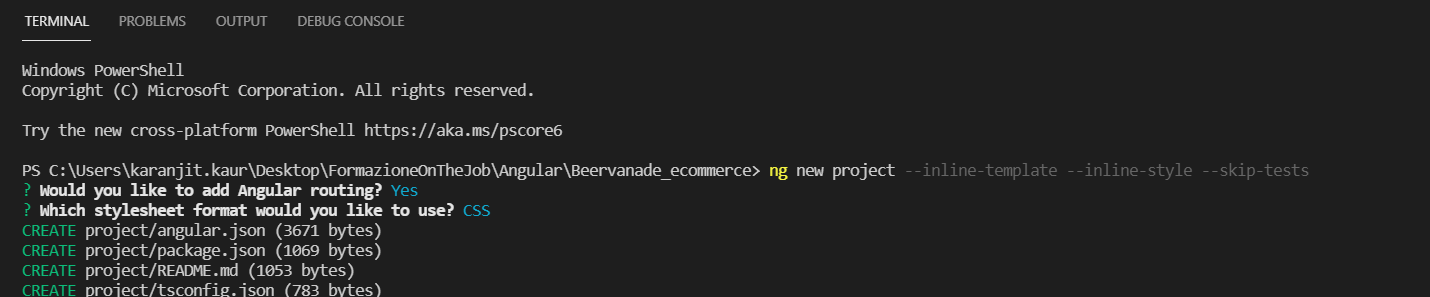


We cloned this repository in just one local pc in order to set the basic structure for all the team members.

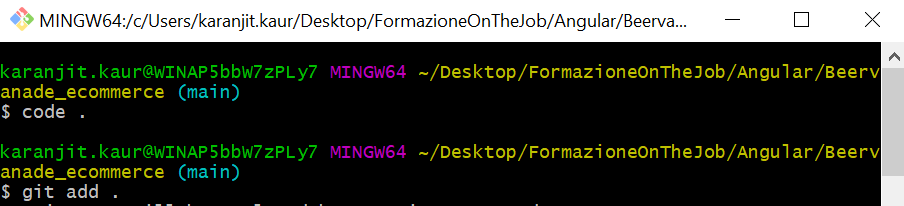
We opened the visual studio code after cloning the repository with the git bash (command: “code .”, and there was an empty project.

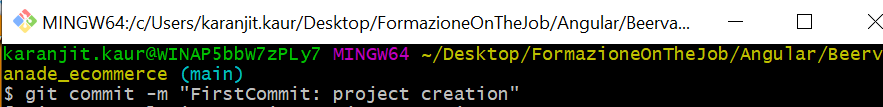


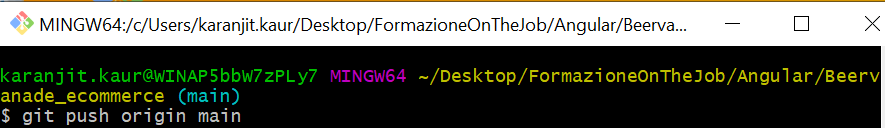
Since we already have installed “node.JS” and “Angular-CLI” we proceeded with the creation of the project with the integrated terminal:



After creating the project, we pushed it on the Git Hub repository



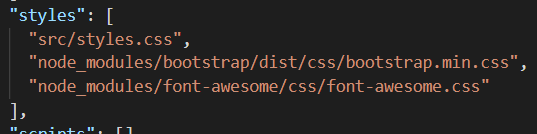




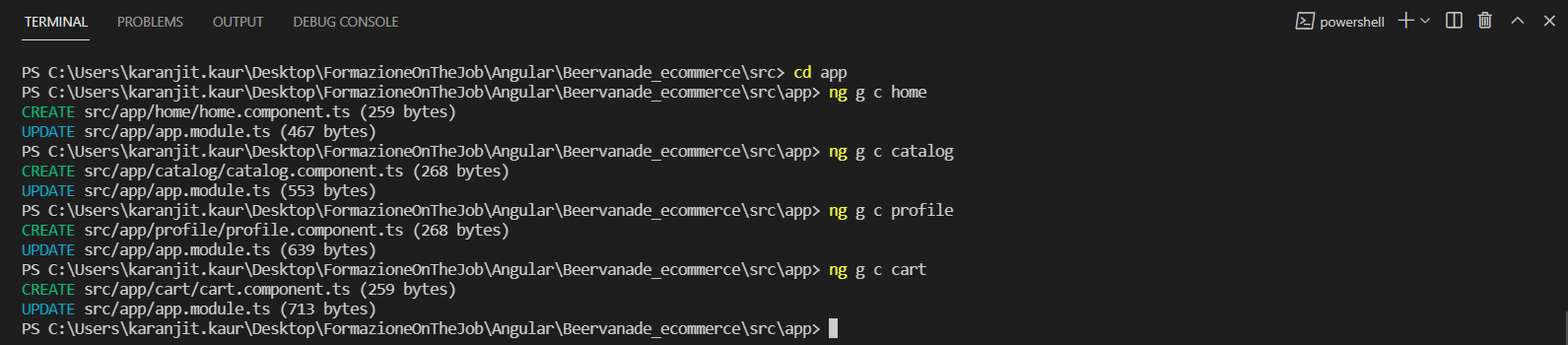
We installed bootstrap and font-awesome into the terminal:



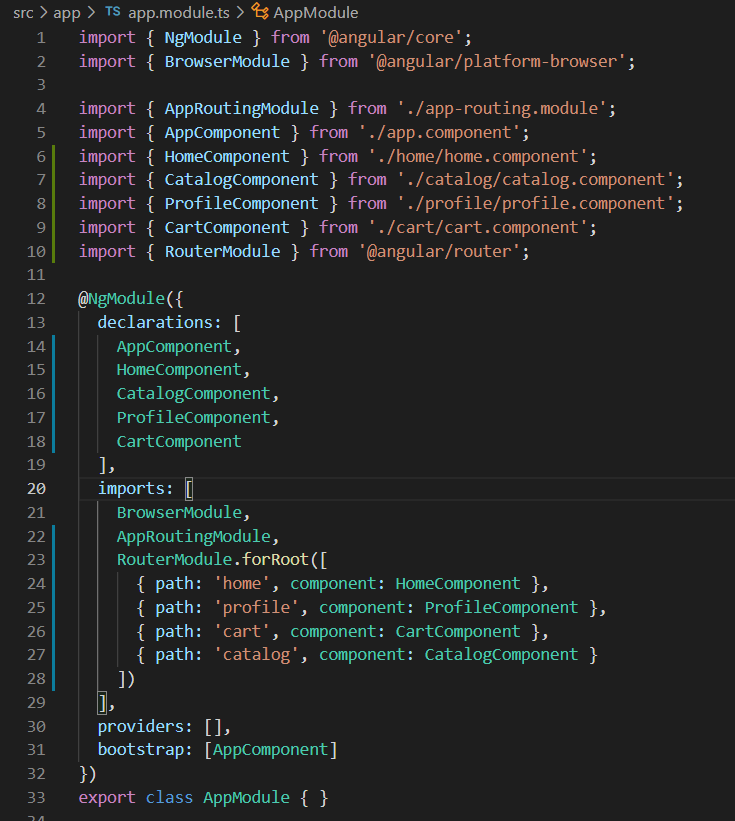
And added these code in the angular.json:



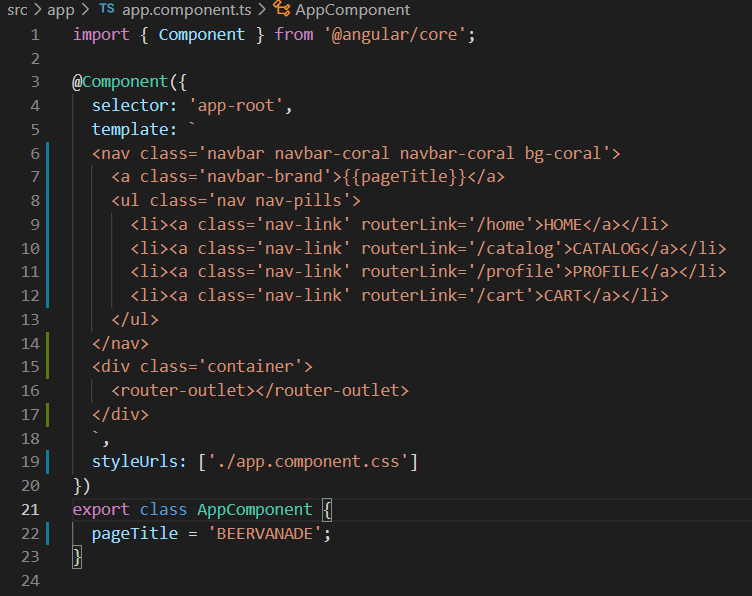
So, we created the components with the “Angular CLI”, moving into the right folder.



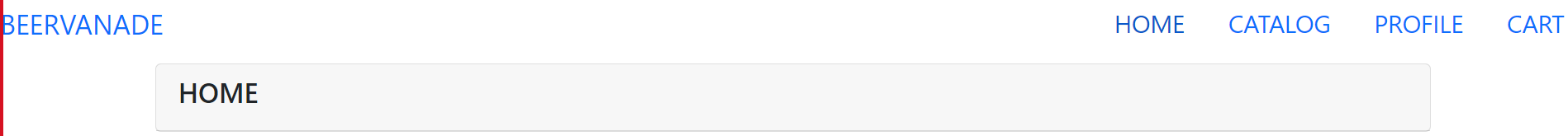
And we modified the app.module.ts in this way, in order to have the routerLink to navigate from one page to another



And in the app.component.ts we added the navbar

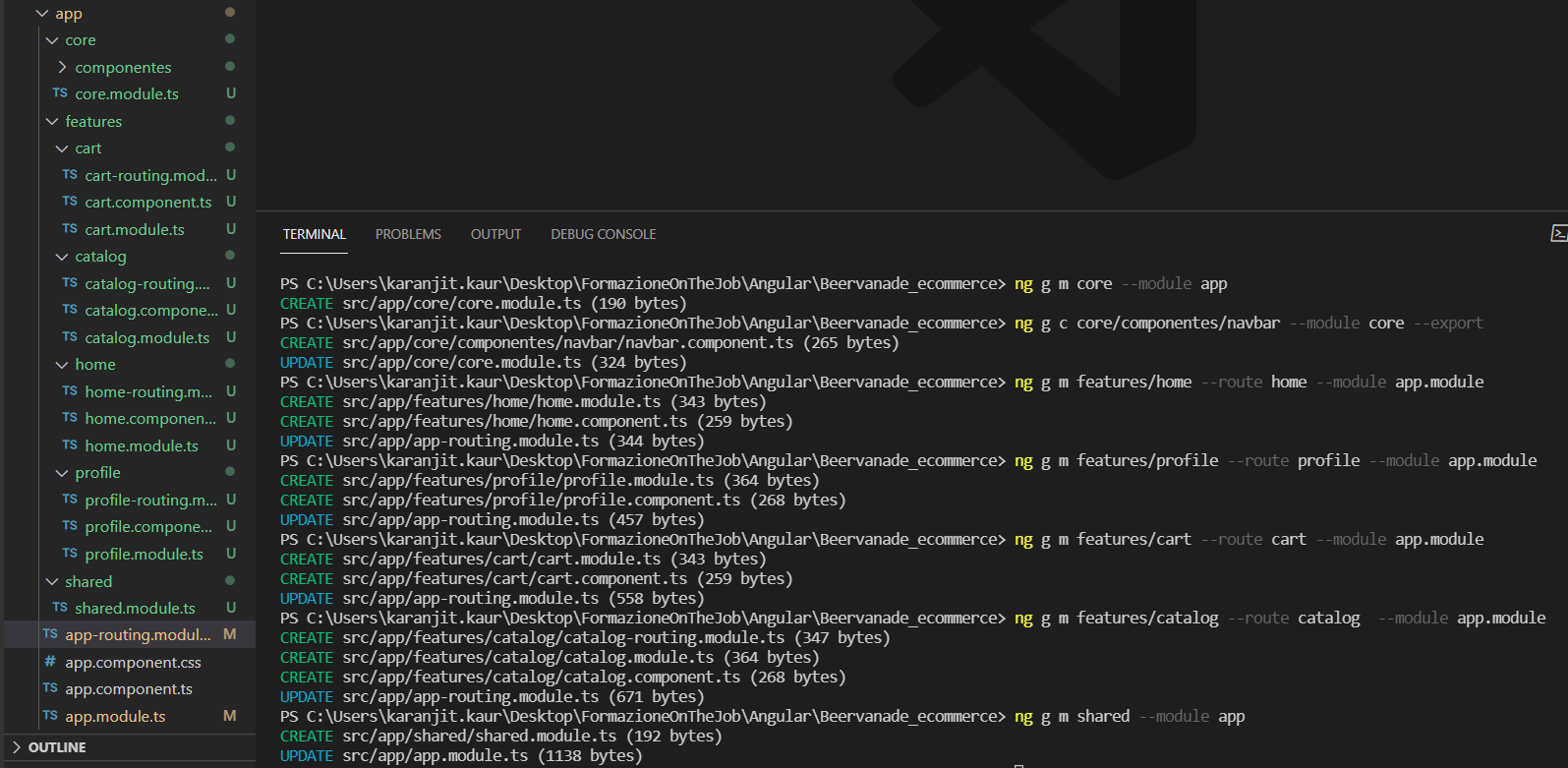


We added the routing in order to navigate from the pages.



This is what we obtained, at the end of first day.

We proceeded to create the architecture into the command line, so we obtained this structure.



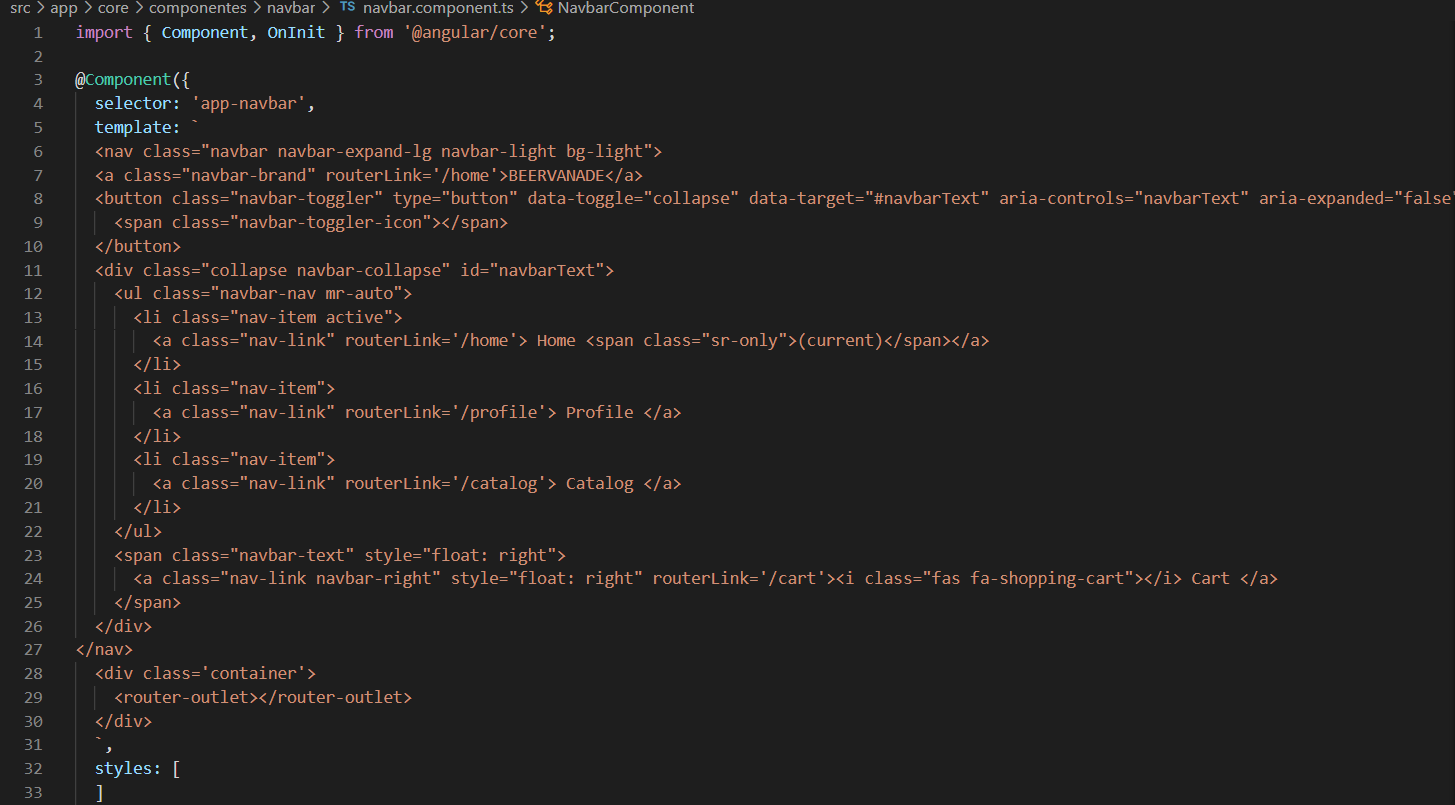
Since we have done yesterday the routing we just adjusted for the new architecture.



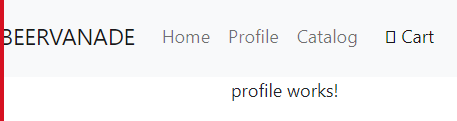
The command line command automatically generated the path for the routing.



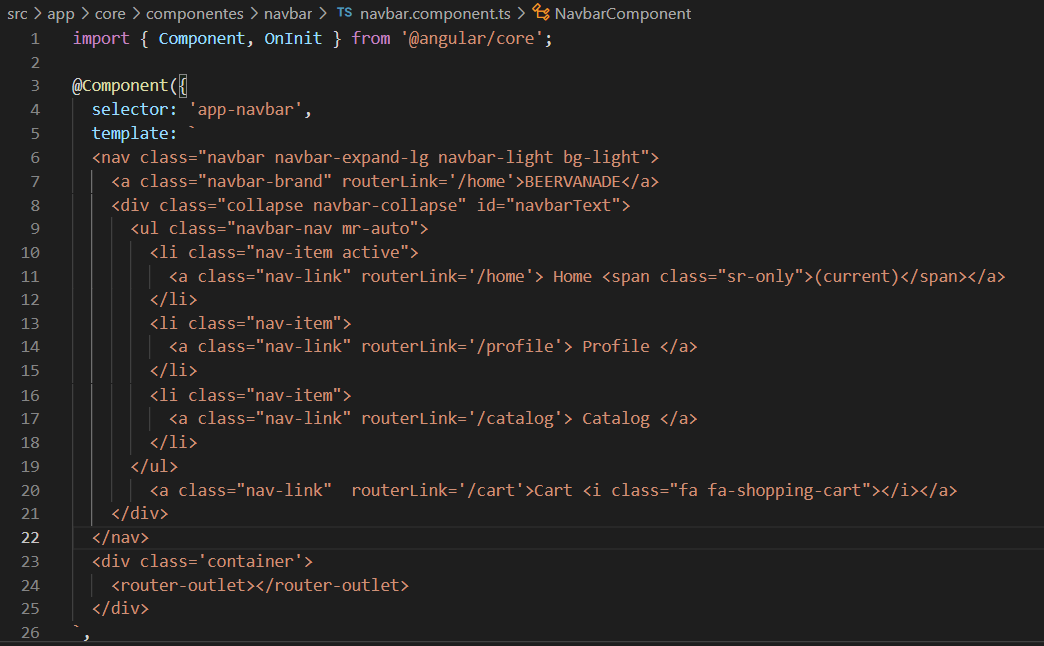
Then we added the header



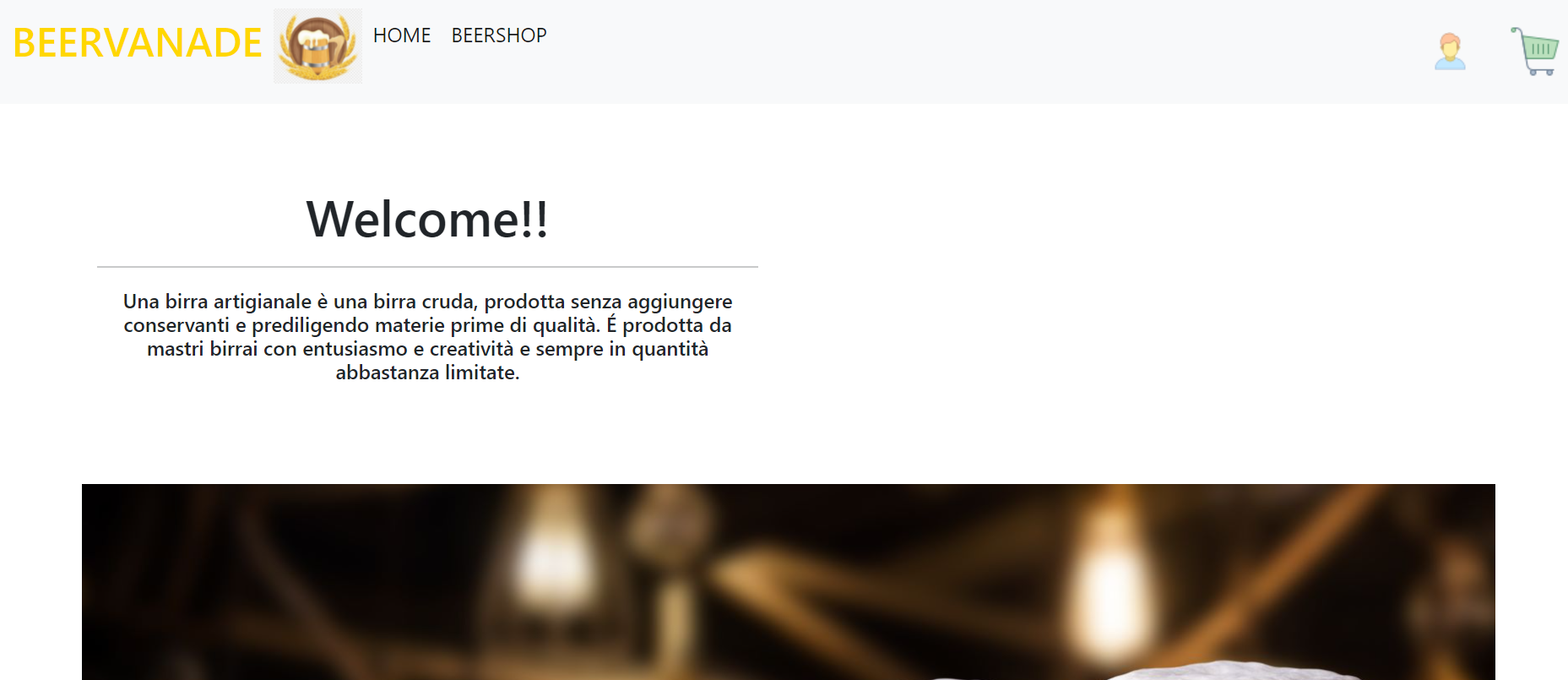
This is what we obtained



We had some issues with the icon and the cart, since we wanted to move to the right the cart.

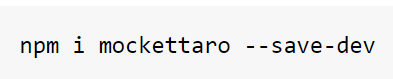


We added some html files and added some logo and image. We edited the home page, working on home.component.ts and [home.component.html](http://home.component.html).

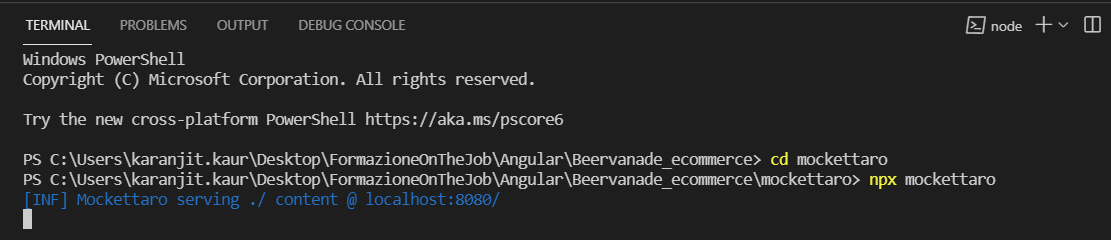


We will fix the layout later on.

Then we installed locally the package to simulate the a back-end server:



We have launched the mockettaro:



We settled the same local host editing the package.json:



And added this new file:

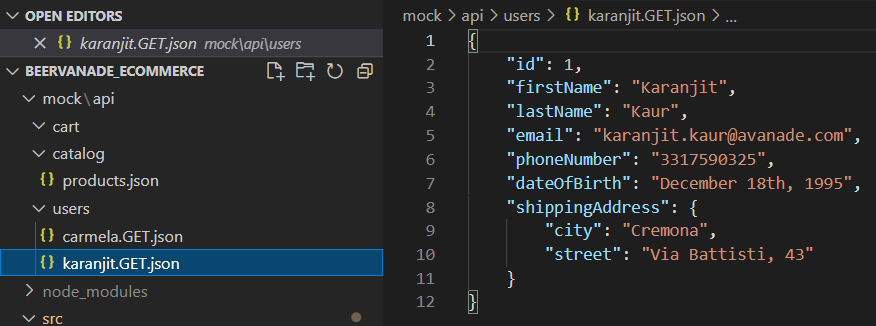


We create some user to simulate a database:

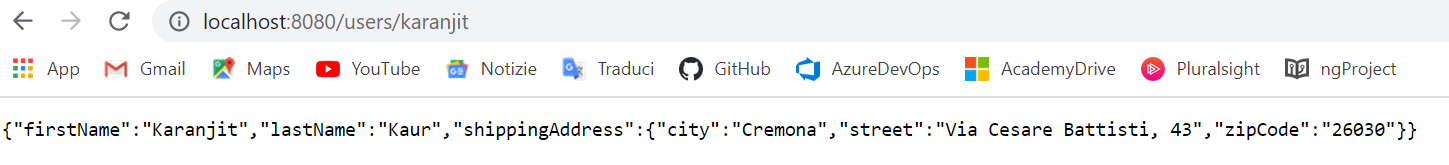


The code writed in the package.json is to lunch in a simultaneous way on the same localhost:4200 the mockettaro and the web app.

We added some users and an array of products:



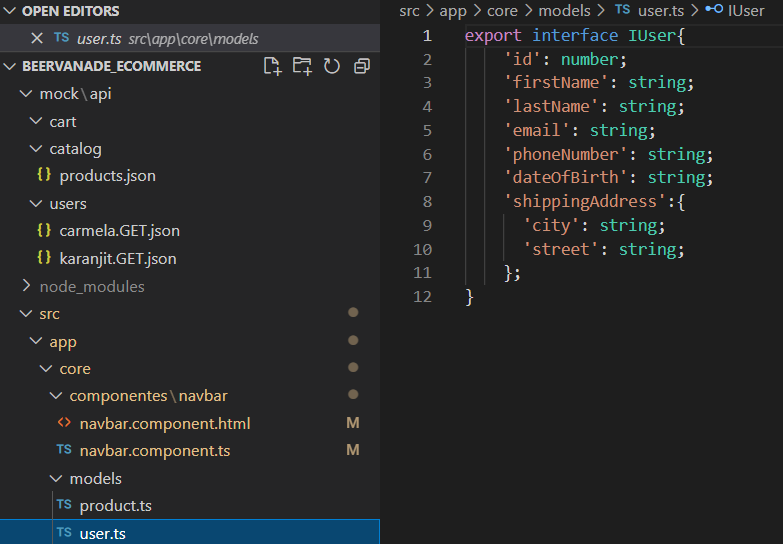
And the result we can see on the browser is this one

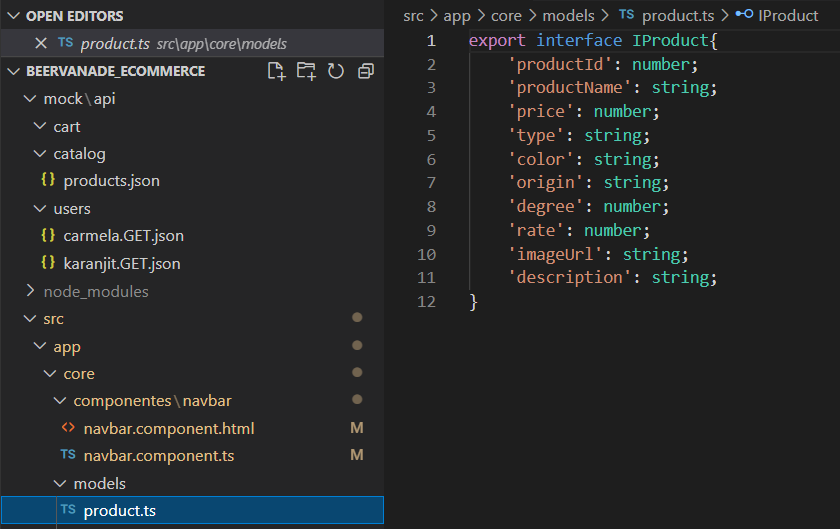


And the array of products is:

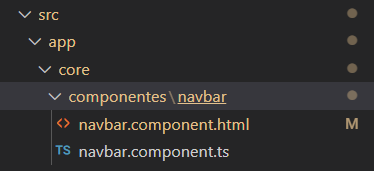


We defined also the two model of the user and the product in the core:

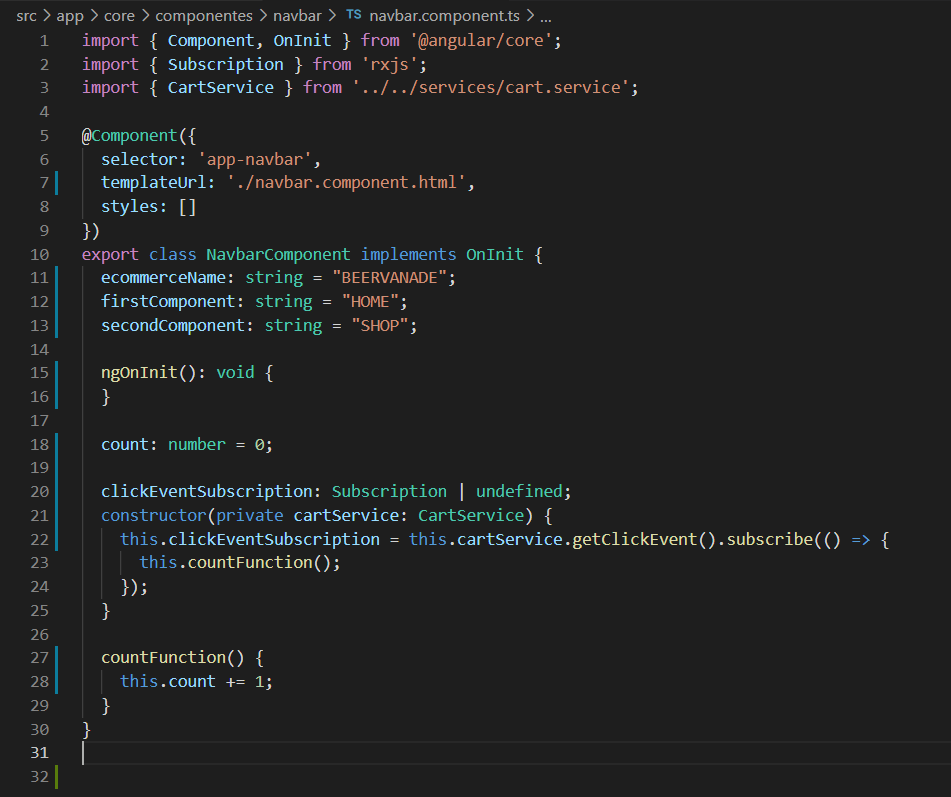




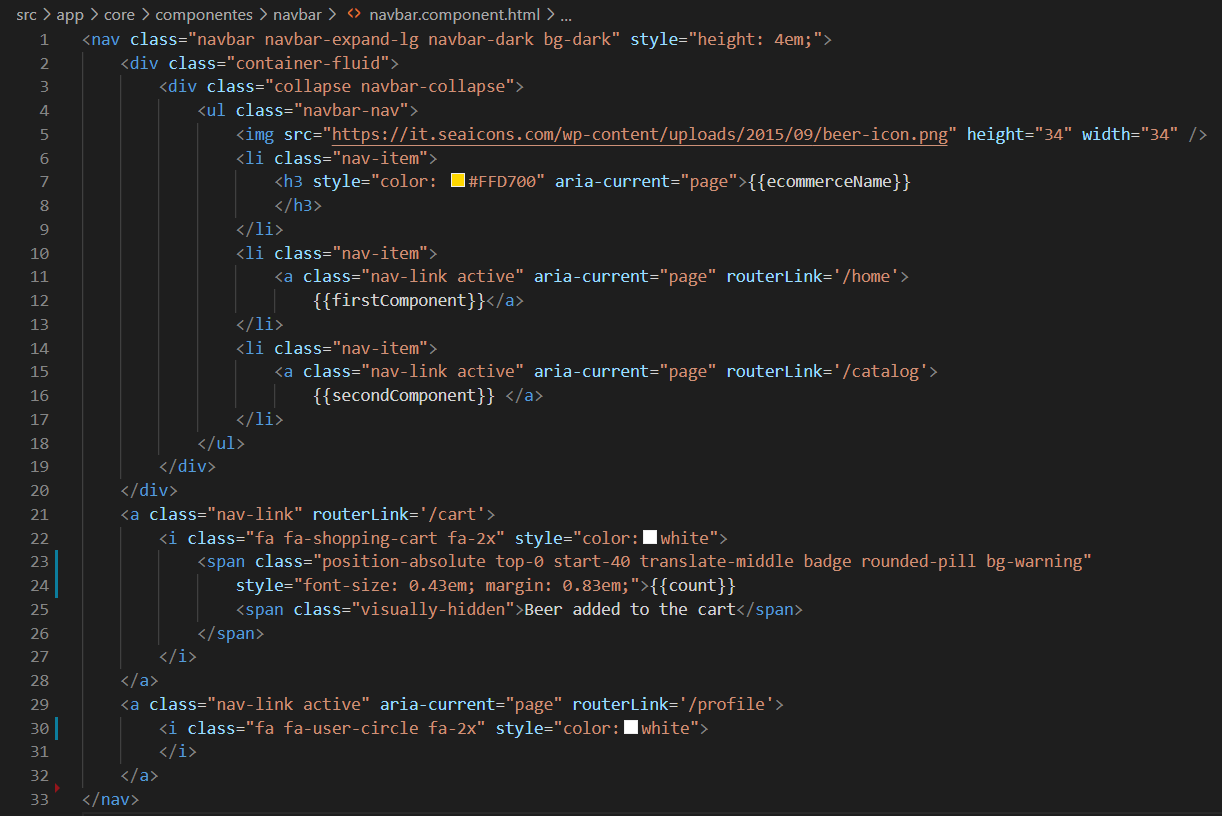
First we started creating and working with the core, so we worked on the navbar.



This is the component of the navbar with the imports, the decorator and the class

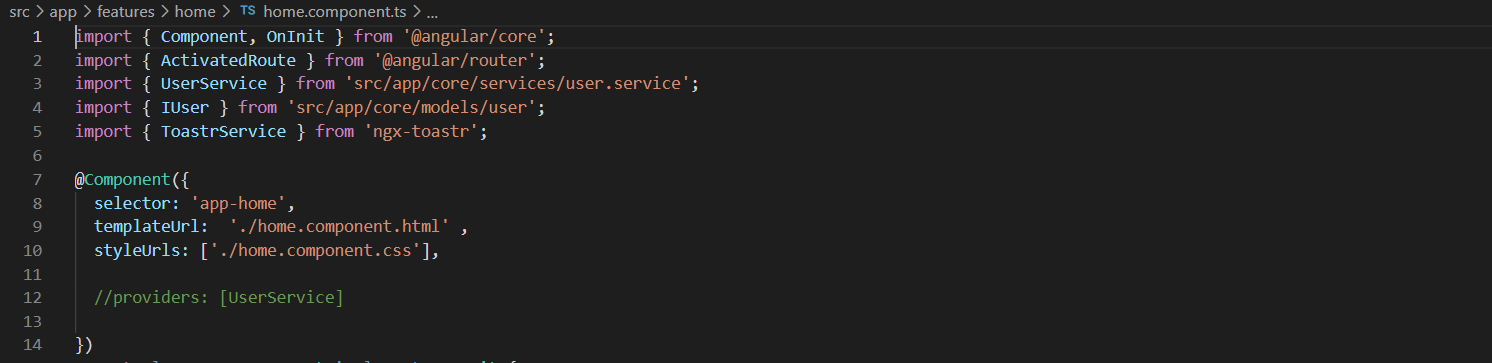


Some functionality, from line 18, are created in order to have a badge that count the items in the cart. We created the link to the html file.

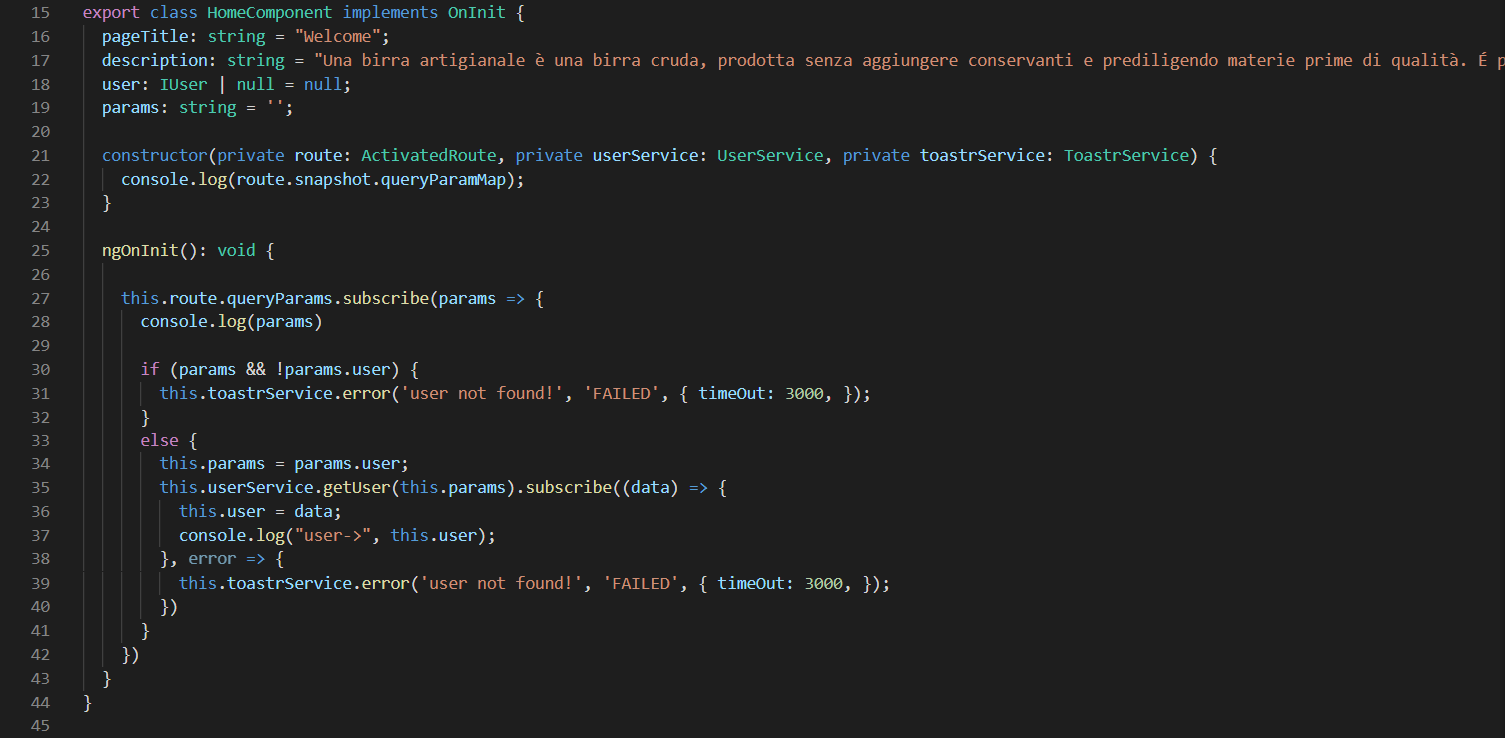


Then we started creating the home page layout with the functionality required.

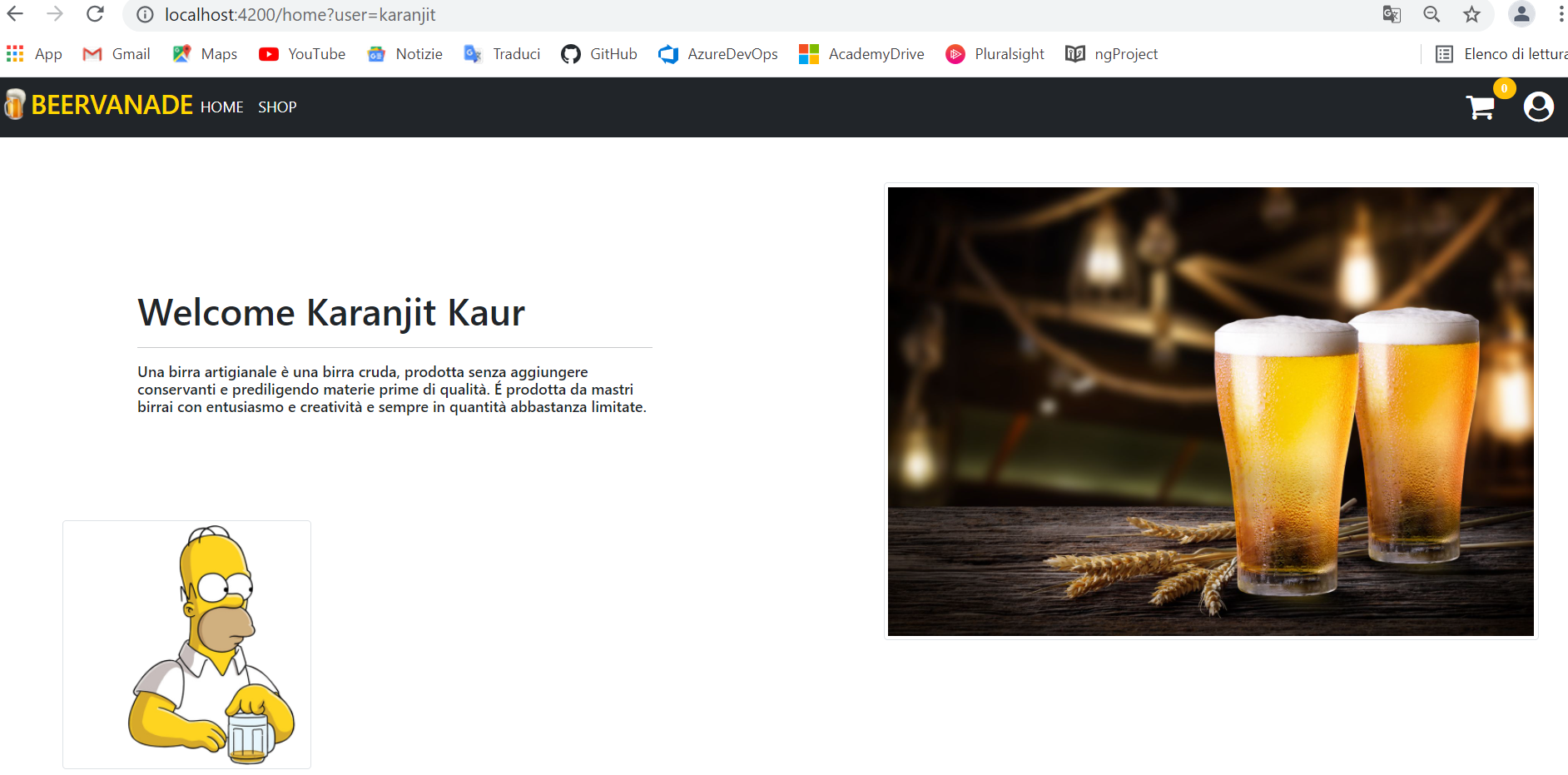
Here we have the [home.component.ts](http://home.component.ts) with the import and the decorator



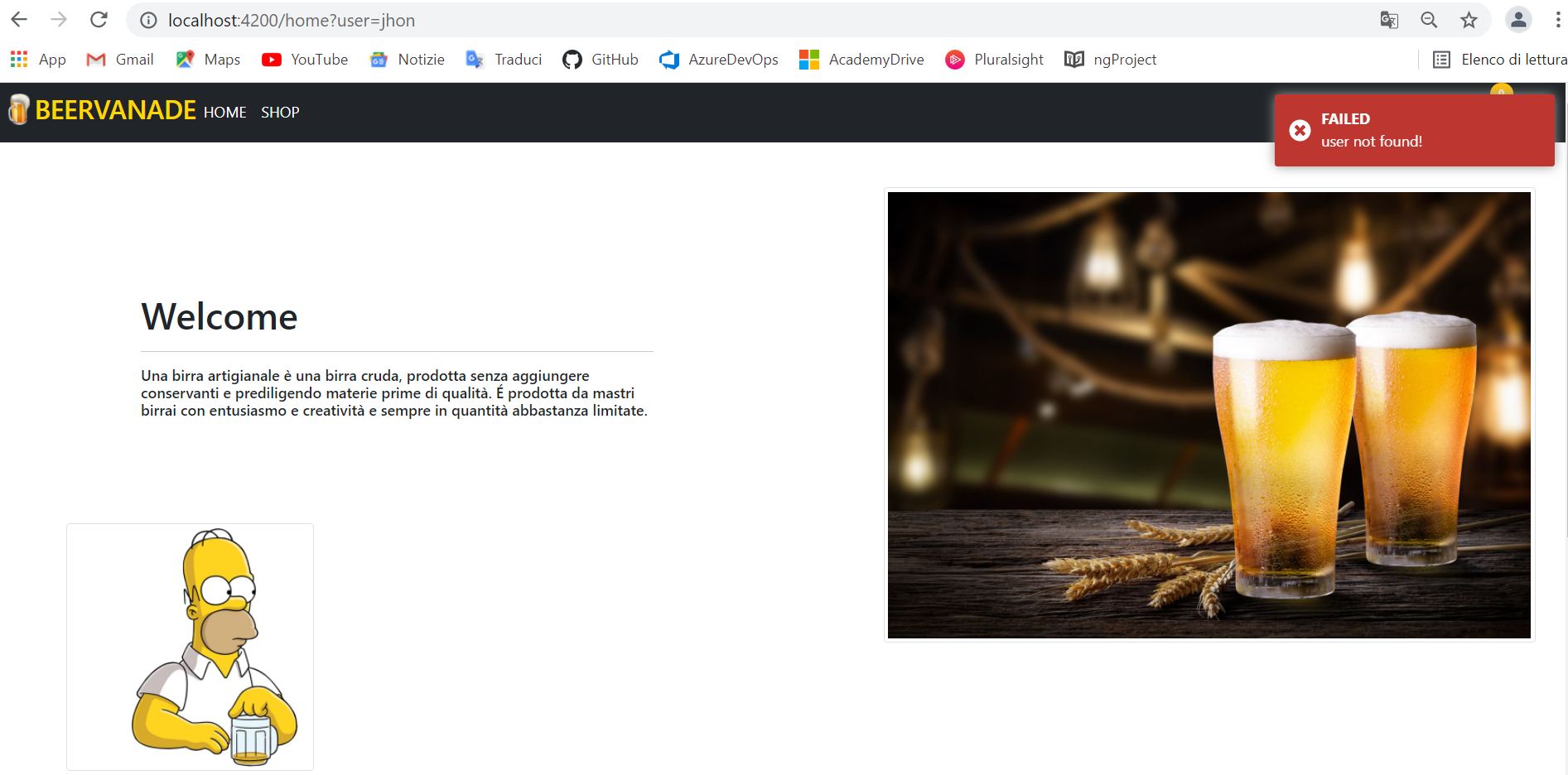
And here we have the class:



In this component we take an user form the querystring (for example: localhost: 4200user?karanjit) and if the user exist in the mock it will return a Welcome message customized with first name and last name.



if the user doesn’t exist it will return a error message: user not found



This error message is created with a particular tool: toaster we installed the package with the CLI and added it to the angular.json the line 59

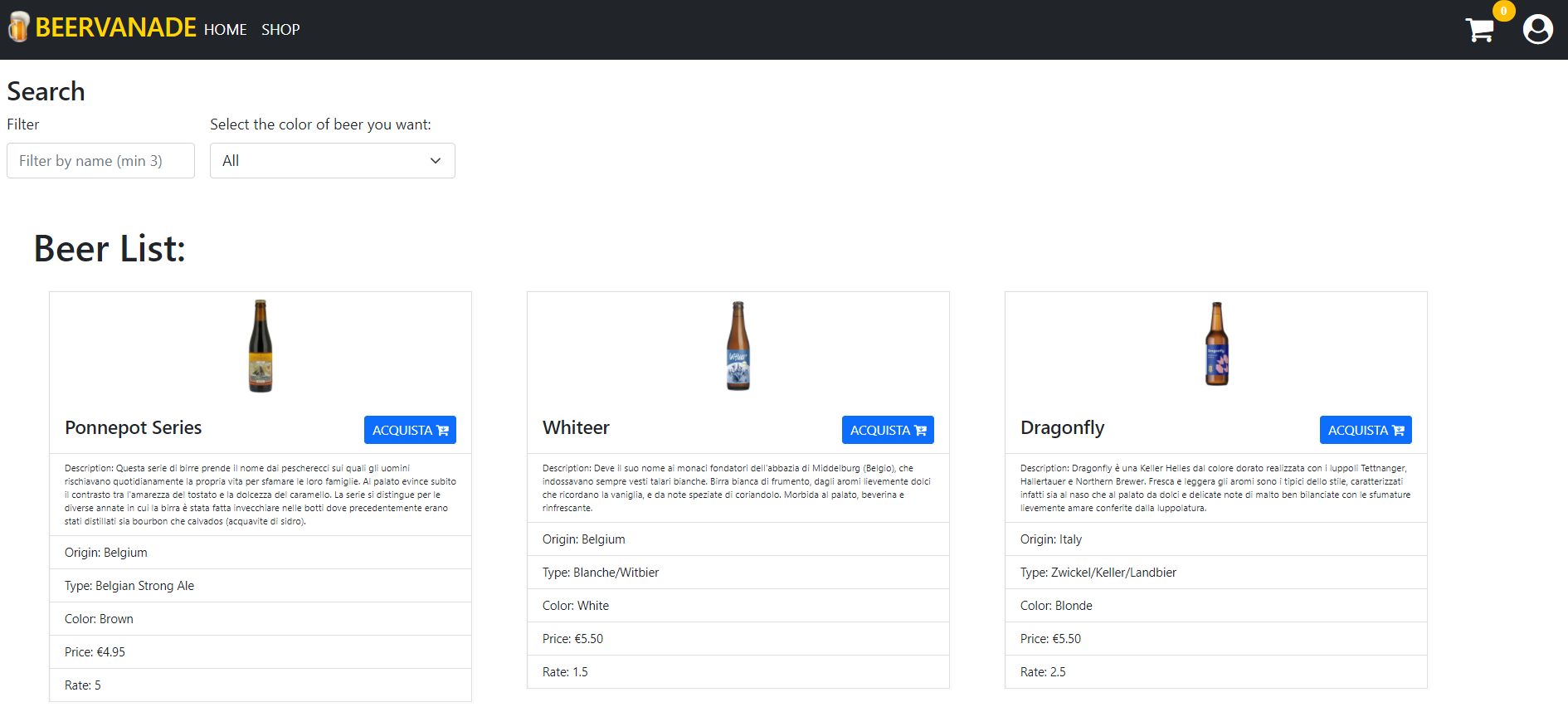


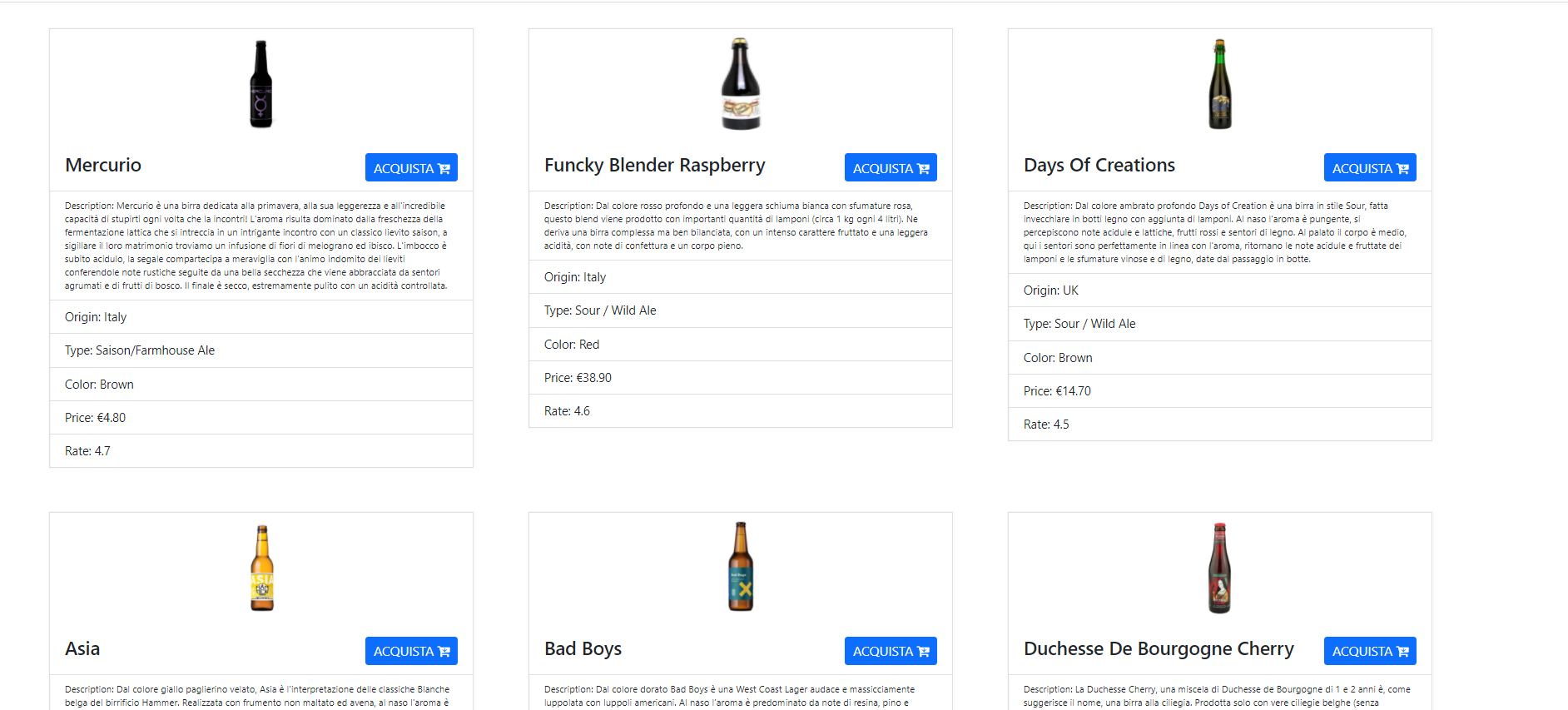
Then we imported the Module in the core.module.ts and in the [home.component.ts](http://home.component.ts), So when a user is found thought the querysting it will return no error and a welcome message with the first name and last name, else it will return the red message of user not found. We also used the user.service.ts

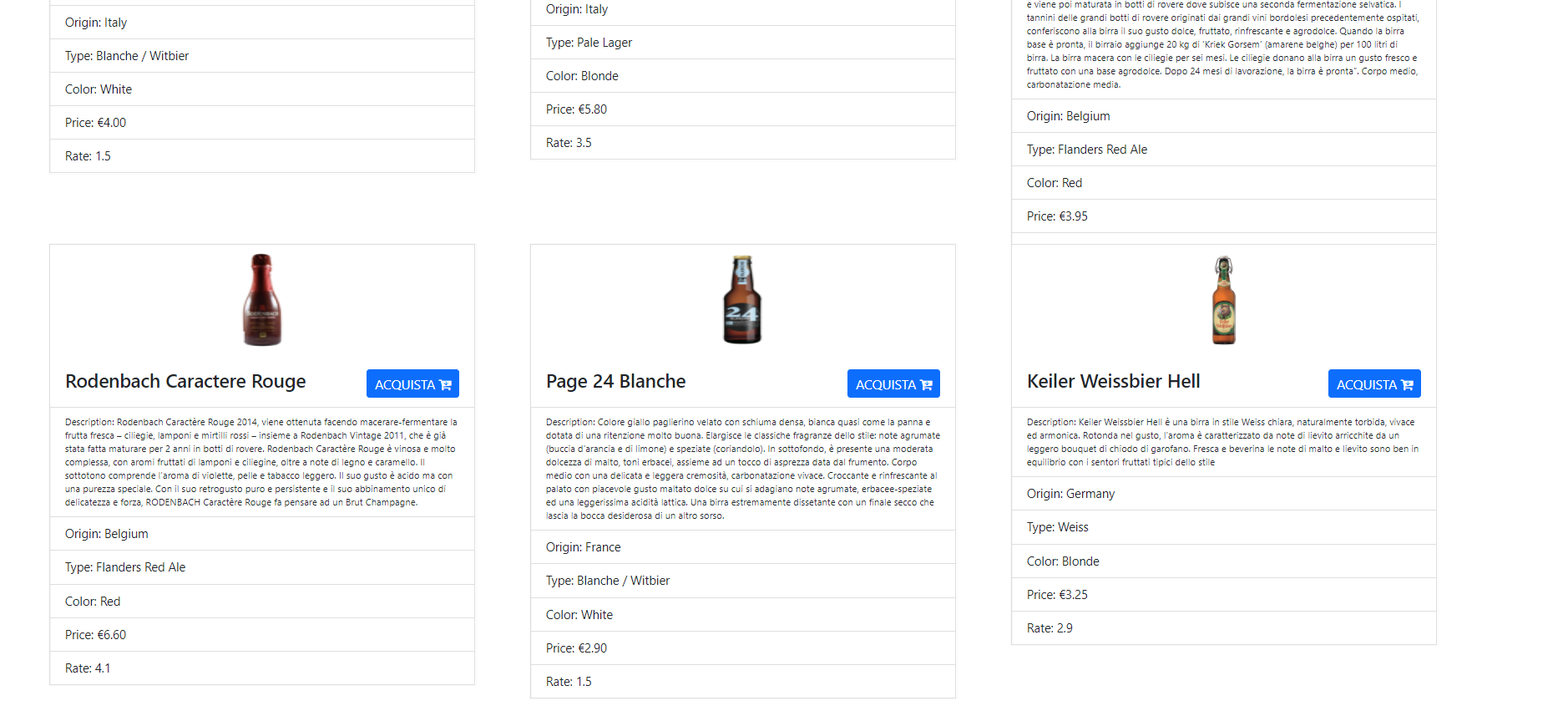


To get the user from the mock and return the User, with the Http

After the home component we worked on the catalog.compoment.ts

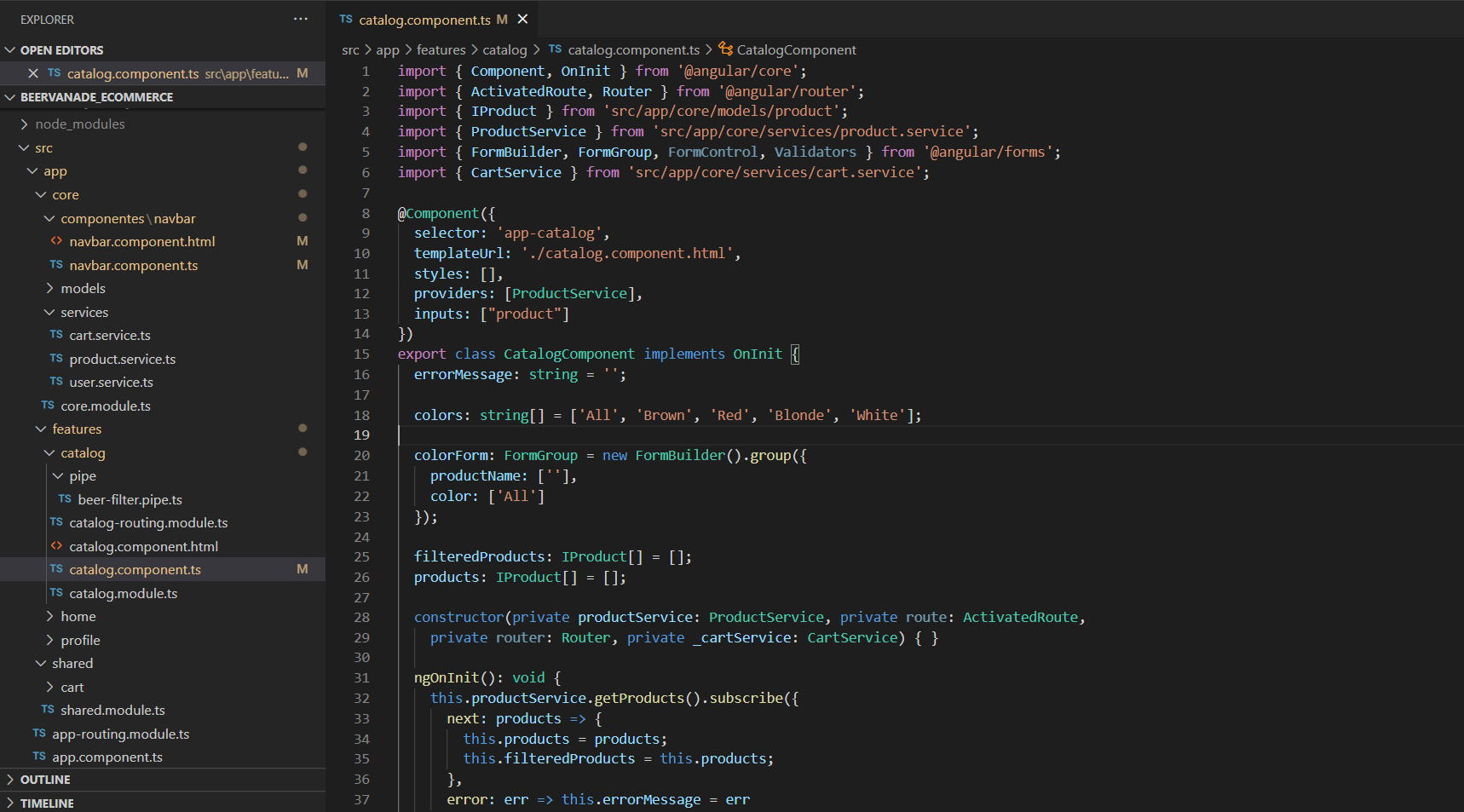


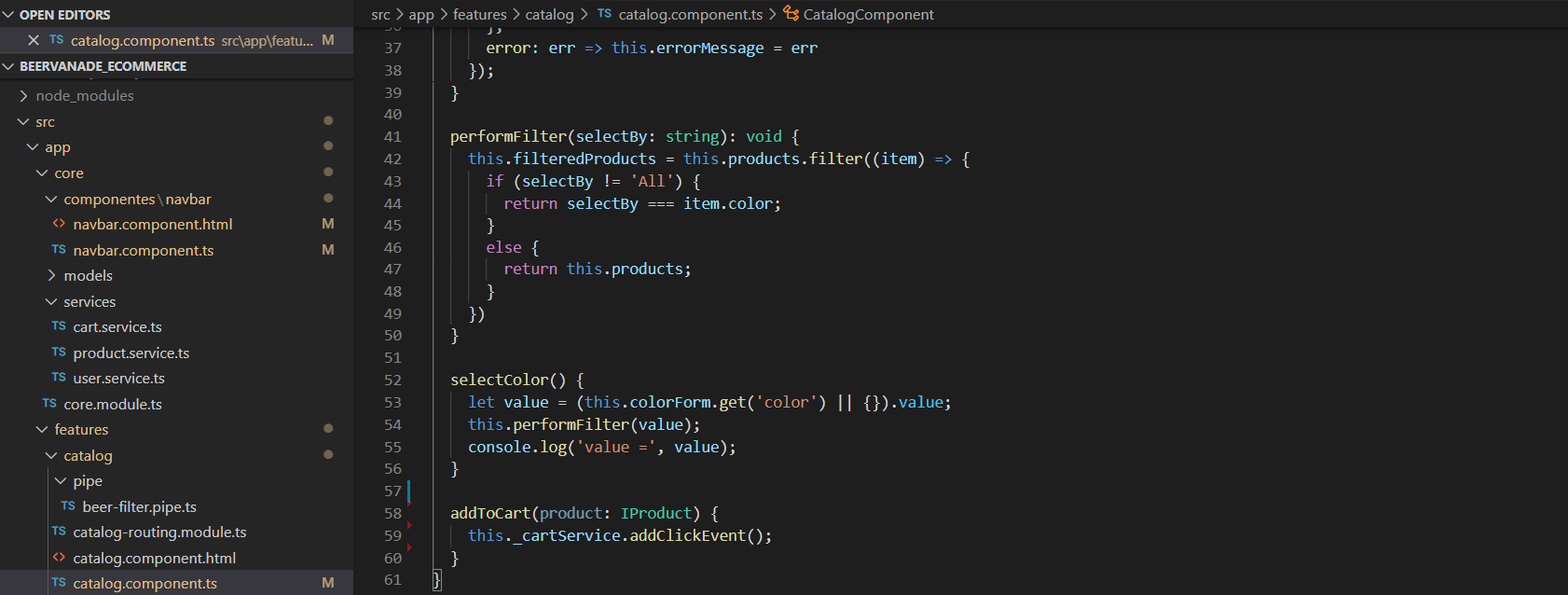




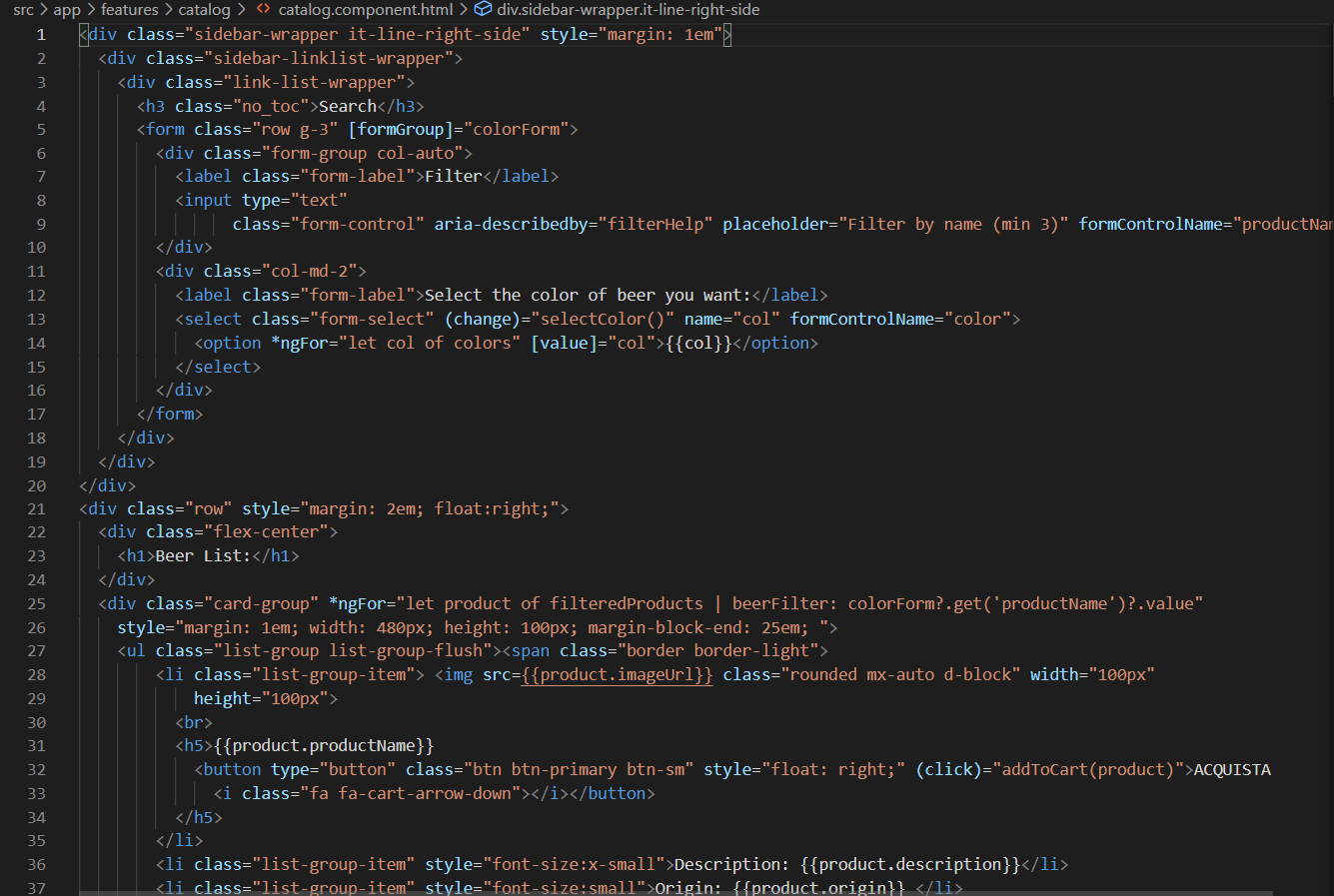
Where is displayed a list of all product in the products’ array.

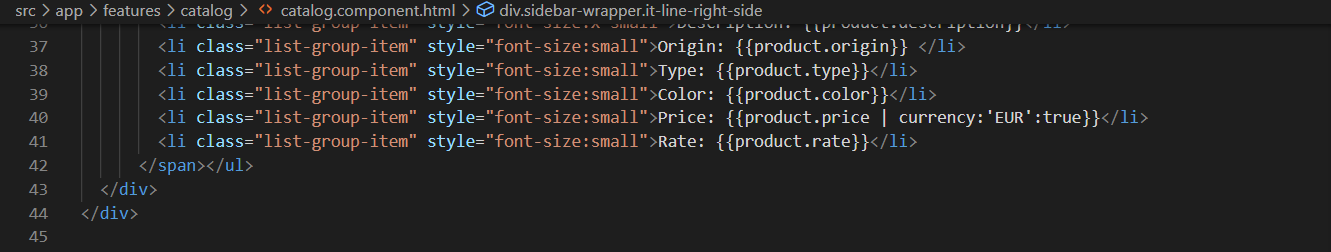
If we look at the catalog.component.ts





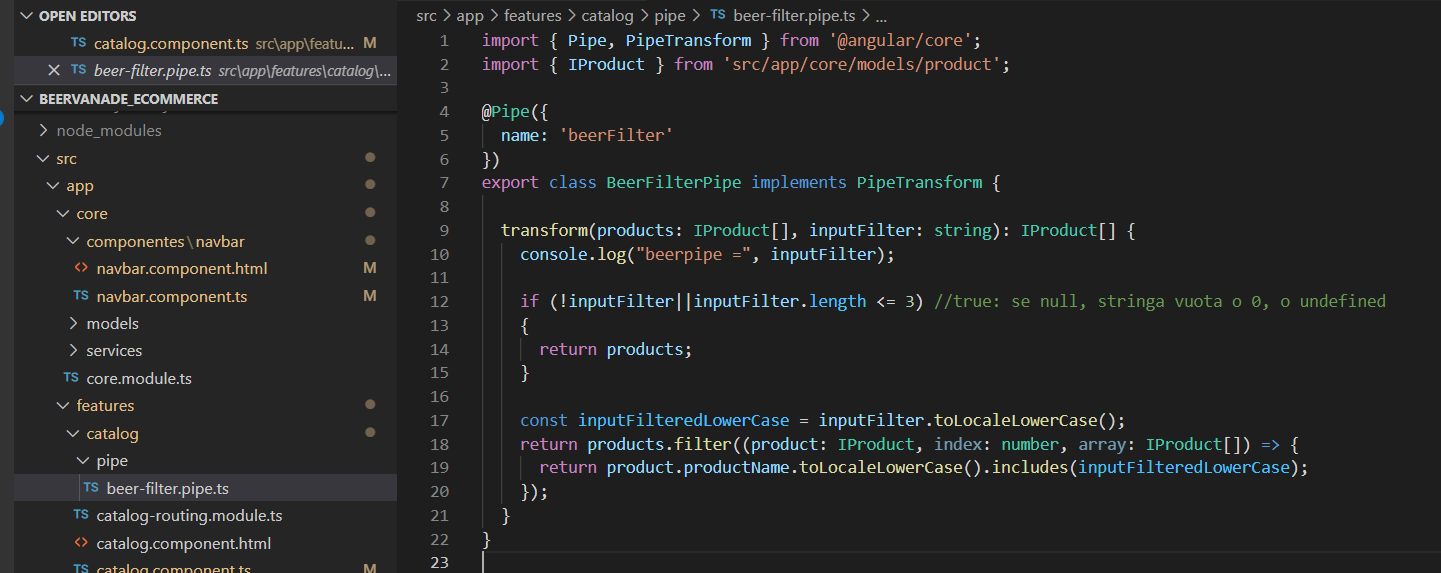
And the html is:





We also used the product.service.ts in the core



To get the product from the mock

In the catalog.componet.ts we have some functionality also for the filter, we filtered the product using the reactive forms. And there are some functionalities for the select, like the array declared “colors”. For the filter we also added a pipe which was generated with the cli.

We also added some functionality in order to perform the badge, and when you click on the buy button it will add a quantity to the badge.

Due to time problem we only implemented only these two component.