Brain Fog Blog

Author: Daniele Campagnoli

Index

[Introduction 2](#_Toc97400243)

[Project Repository Structure 3](#_Toc97400244)

[System Architecture 4](#_Toc97400245)

[Docker Compose Setup 5](#_Toc97400246)

[Configure a Debian machine with docker 10](#_Toc97400247)

[How CCS is structured 11](#_Toc97400248)

[Grid system 11](#_Toc97400249)

# Introduction

The Brain Fog Blog (BFB) is a blog abut technology and generative art.

The main scope of this project is to demonstrate my skills on software engineering.

# Project Repository Structure

The project is hosted at the following git repository:

<https://github.com/DanieleCampagnoli/brainfog>

The repository is divided in 2 folders:

1. docs: contains the project documentation
2. sources: contains the project source code

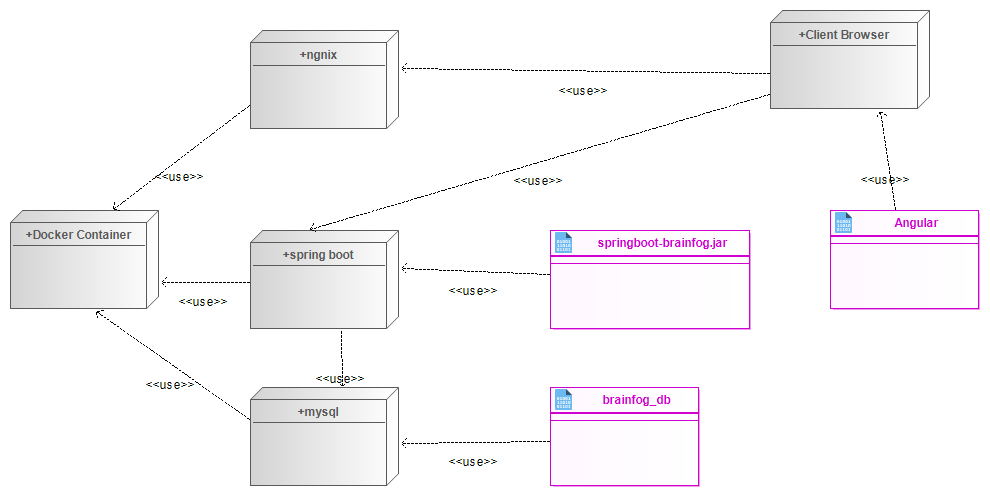
The docs folder is organized with the following files

|  |  |
| --- | --- |
| file | description |
| documentation.docx | The project documentation (this document). |
| modelio.zip | A modelio project with UMLs.  The tool can be downloaded from this website  <https://www.modelio.org/> |
| brain\_fog.xd | This file contains a mockup of the website made with Adobe XD. |
| posts | Folder with a word file for every post. |

The source folder is organized in the following way:

|  |  |
| --- | --- |
| file | description |
| angular-brainfog | Angular application source code (UI) |
| springboot-brainfog | Spring boot application that contains the business logic of the application. This application exposes a rest API that is consumed by the Angular application. |
| AngularBrainfog.Dockerfile | Docker configuration for the angular-brainfog application. |
| docker-compose.yml | This docker configuration file is used to configure and startup all the docker containers in order to start the overall system. |
| MysqlDB.Dockerfile | Docker configuration of MySql ( Persistence). |
| SpringBootBrainfog.Dockerfile | Docker container for springboot-brainfog. |

# System Architecture



The application follows the classical 3 tier architecture:

1. presentation tier: a single page application developed with angular
2. application tier: a set of algorithms that are exposed via a rest api developed with spring-boot
3. data tier: a mysql relational database

**Version 1.0.0**

Every blog post is mainly implemented in angular. The blog text is hardcoded in the html page.

The spring backend and the persistence is called only to feed the UI with the algorithm output.

# Docker Compose Setup

**Main idea**

Angular will be deployed on a dedicated web server on a dedicated docker container.

Spring will be deployed on a dedicated web server with the standard spring boot setup and it will use one docker container.

MySQL will be deployed also on a dedicated docker container.

**Angular And Docker**

Angular And Docker Deployment taken from the following tutorial

https://www.indellient.com/blog/how-to-dockerize-an-angular-application-with-nginx/

Command to build the angular docker image

docker build -t angular-brainfog -f AngularBrainfog.Dockerfile .

Command to start the angular docker image

docker run -d -p 8080:80 angular-brainfog

**Spring Boot And Docker**

<https://spring.io/guides/topicals/spring-boot-docker>

docker build -t springboot-brainfog -f SpringBootBrainfog.Dockerfile .

docker run -d -p 8080:8080 springboot-brainfog

**MySQL and Docker**

The configuration is specified by the docker-compose.yml. For now, I’m not using any special configuration.

**Docker Compose**

Start all the applications required by the system: db, application server and web server with the angular application.

Run the following commands from the project folder.

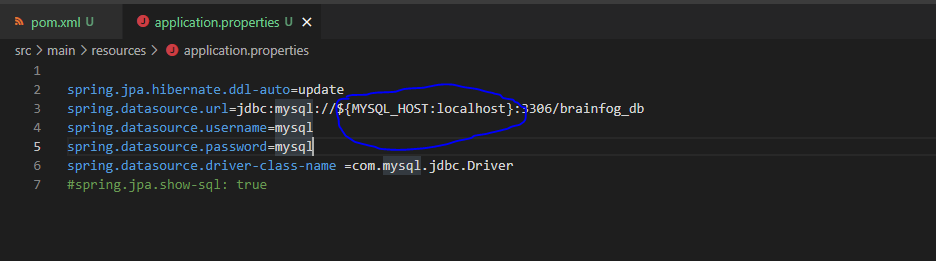
docker-compose build

docker-compose up

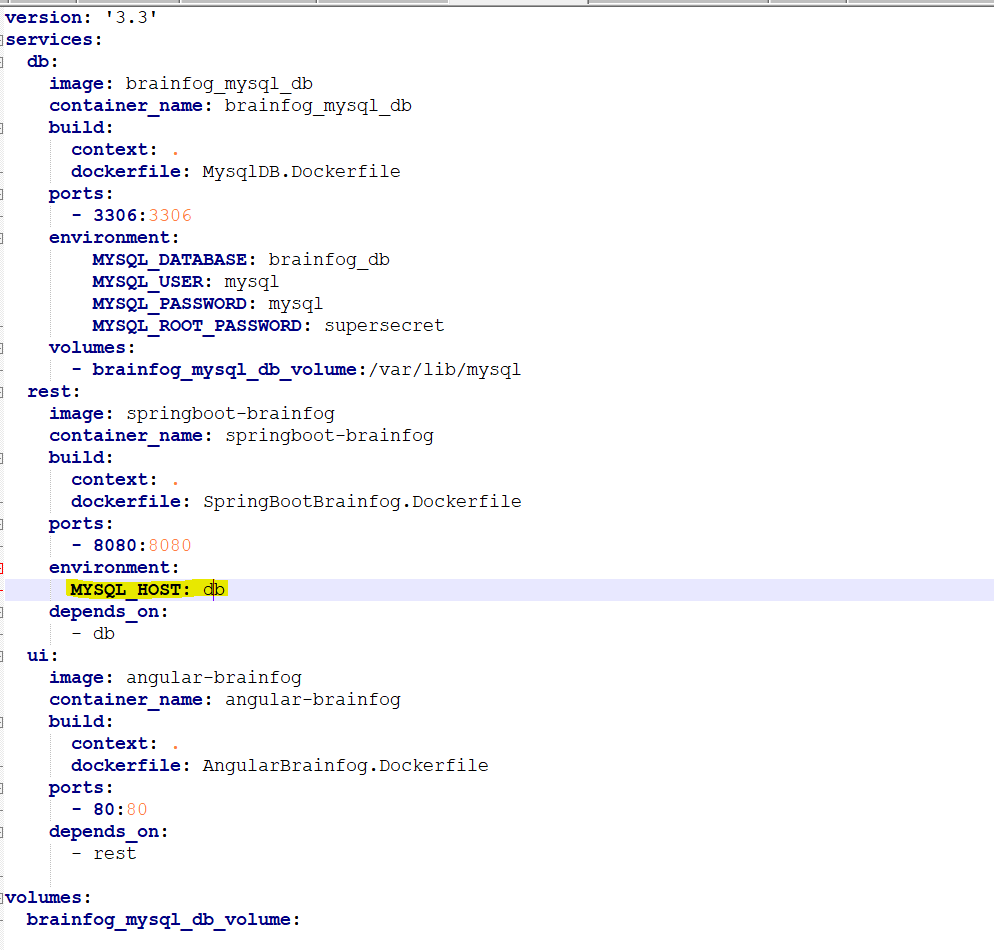
The docker compose creates a virtual network.

<https://docs.docker.com/compose/networking/>

In order to make it work there is a special configuration to implement on Spring.



Docker compose configuration

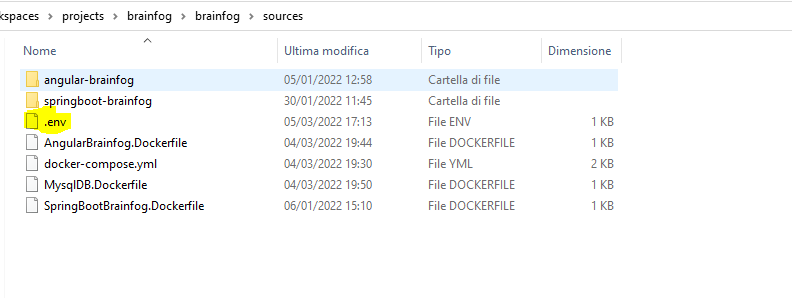


**Password Management**

The application is published as a public project on github.

For this reason, the passwords as specified inside system variables and are populated with a script.

The easiest way to achieve this is to use the standard docker-compose .env file.



The docker-compose command reads by default that configuration value at every execution and its cross platform, so it’s not necessary to produce different scripts based on the operative system where the application is deployed.

This file will not be committed on the git repository.

**Notes on how to deploy on the production machine**

The idea is to download a set of docker image and run them without doing much configuration on the machine.

configuration to deploy on cloud machine

1- MSQL and Spring are managed by docker compose via the virtual network

2- Angular must point to the spring instance via DNS or IP

Angular supports the build configuration via the commands

ng build --configuration=development

ng build --configuration=production

reference [**https://angular.io/guide/build**](https://angular.io/guide/build)

We can wrap that argument inside the docker compose configuration with the command line argument

brainfog-build-config= development | production

docker-compose build --build-arg brainfog-build-config=development

docker-compose up

**how to tag the docker compose image on docker hub**

docker compose push

**how to deploy on the production environment**

The docker compose is particularly powerful with this simplified setup.

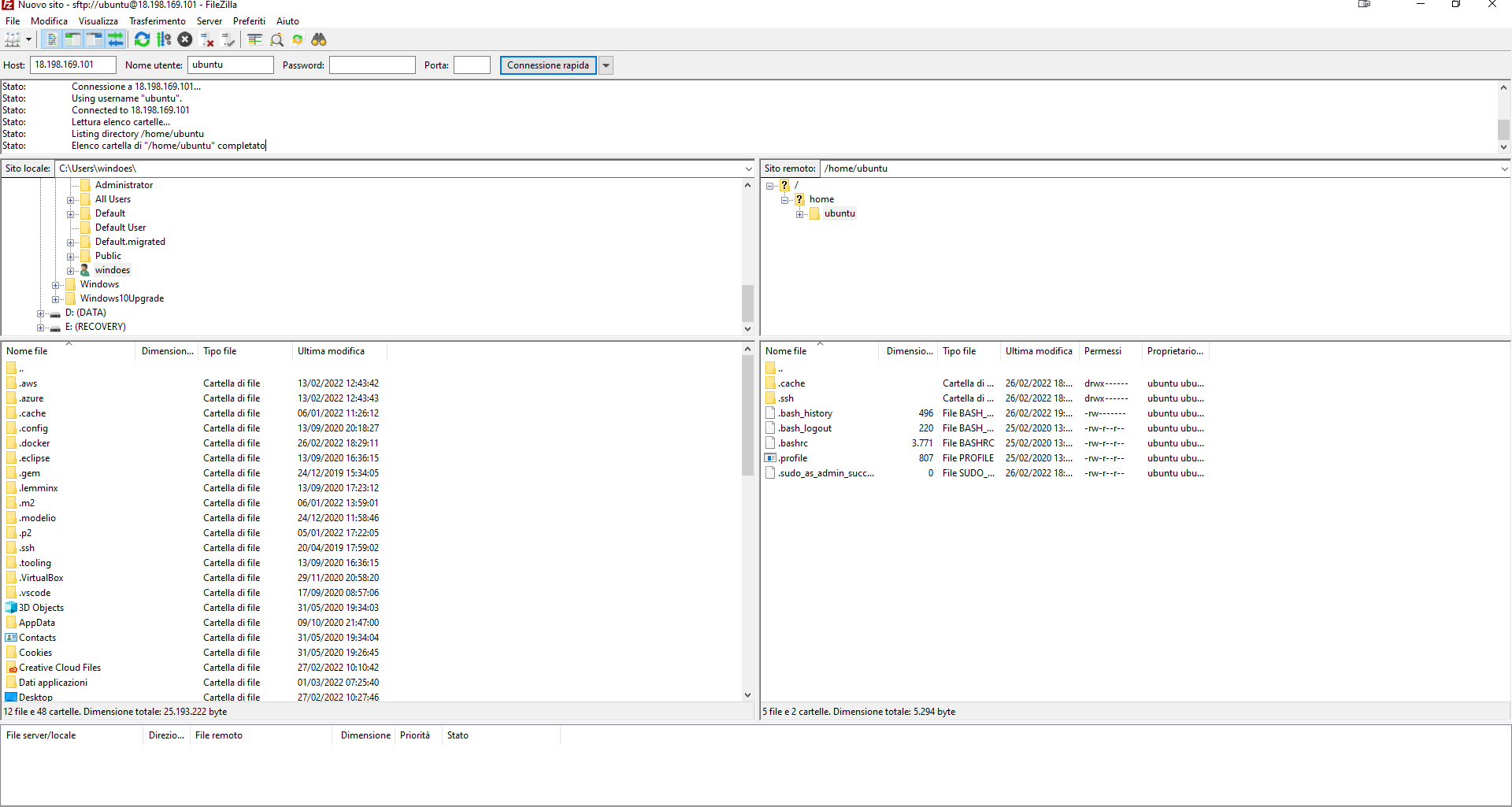
You have just to downloaded the docker-compose.yml from the git repository and run the following commands

Docker-compose pull -> download the images from docker hub

Docker-compose up -> start the containers

**how to connect to the host machine**

Use filezilla, File -> gestione siti -> File Chiave -> select the donwloaded key file from amazon aws



## Configure a Debian machine with docker



# How CCS is structured

The css follows the BEM notation <http://getbem.com/naming/>.

The file src\app\app.component.css contains the global css rules of all the components.

On top of the file we have css custom properties

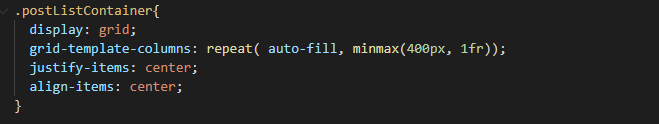
<https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties>

They are variables that can be called from other css classes. This is useful to define colors and other utility values like fonts and so on.

## Grid system

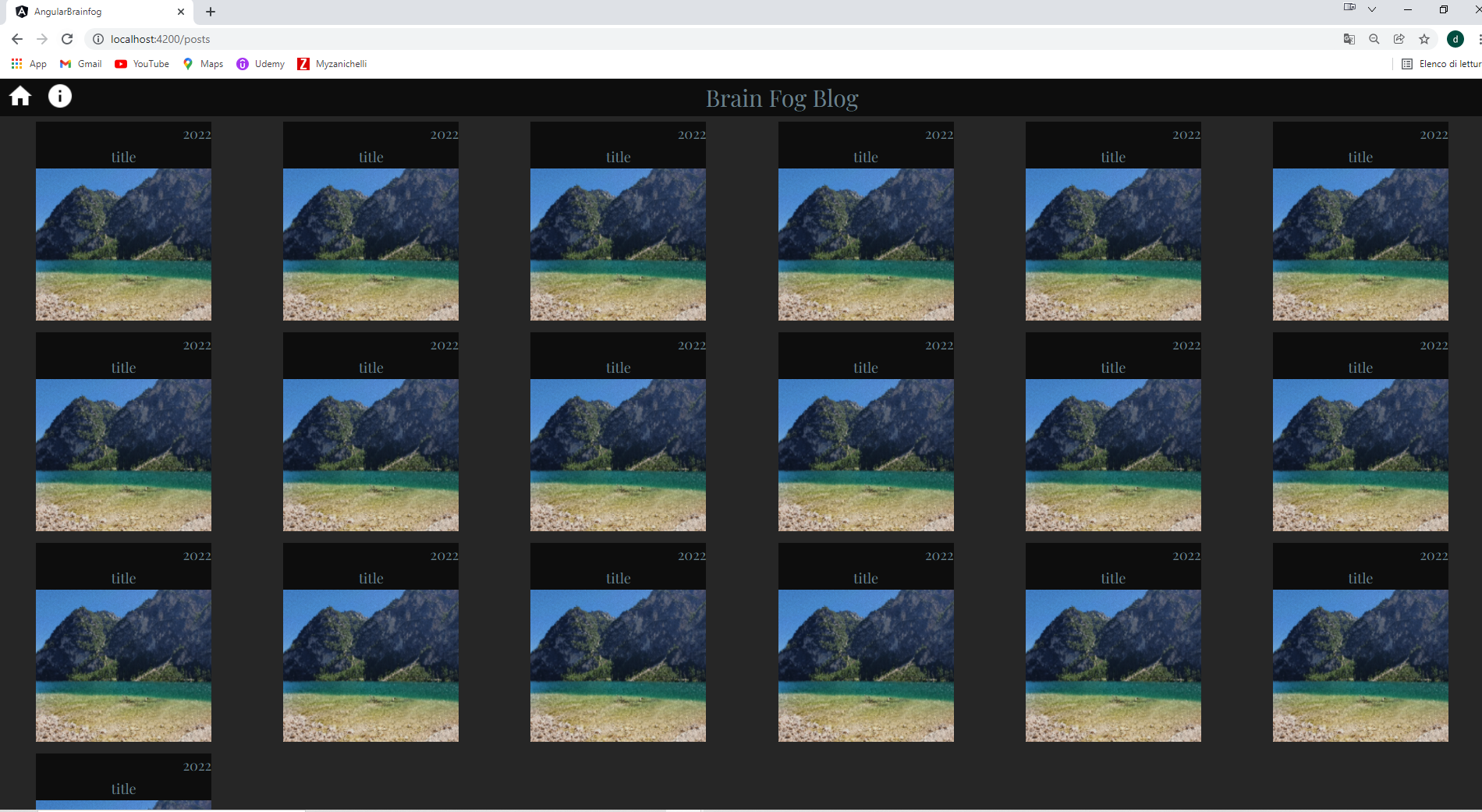
The application is using the standard CCS grid system.

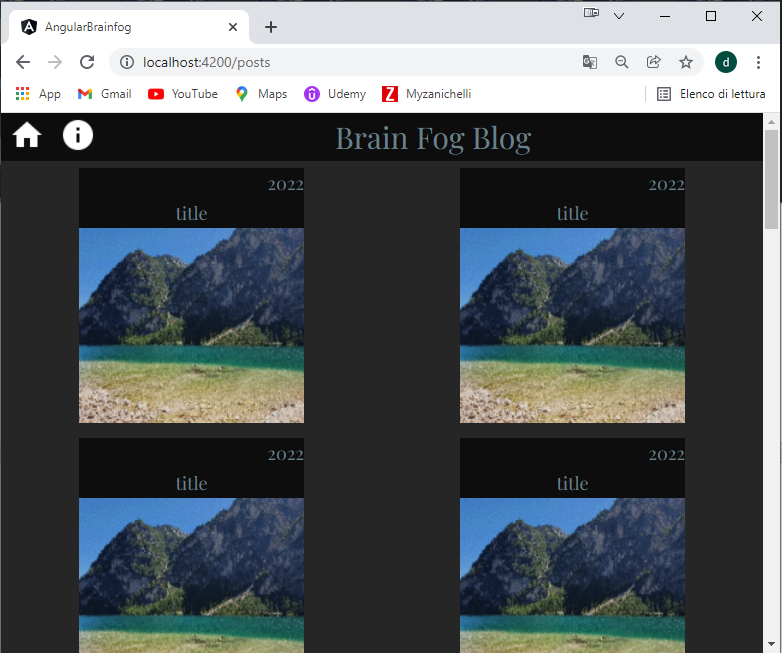
The grid system is also able to arrange the items on the page in a responsive way.



This snippet of code defines a grid with columns that are minimum 400 px wide or they have all the same dimension on the page (1fr).

This allows the list to shrink if needed.





As you can notice from the two screenshots the grid has adjusted the number of columns from 6 to 2 based on the window dimension.