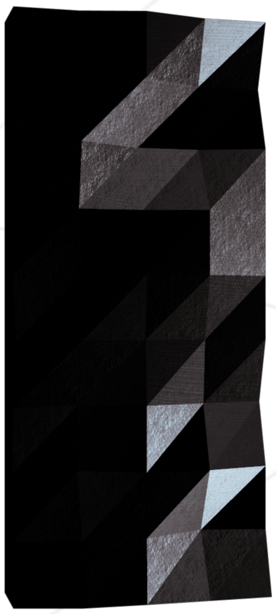


dontpanic_daily

It's not a whale coming from the Magratheans, but you'll be impressed by it nonetheless.

Summary: This project enhances your dontpanic game, adding a new thin slice to make your game cooler and make it easily deployable.

Version: 1.0

Contents

Chapter I

Foreword

“Consider the subtleness of the sea; how its most dreaded creatures glide under water, unapparent for the most part, and treacherously hidden beneath the loveliest tints of azure. Consider also the devilish brilliance and beauty of many of its most remorseless tribes, as the dainty embellished shape of many species of sharks. Consider, once more, the universal cannibalism of the sea; all whose creatures prey upon each other, carrying on eternal war since the world began.

Consider all this; and then turn to the green, gentle, and most docile earth; consider them both, the sea and the land; and do you not find a strange analogy to something in yourself? For as this appalling ocean surrounds the verdant land, so in the soul of man there lies one insular Tahiti, full of peace and joy, but encompassed by all the horrors of the half-known life. God keep thee! Push not off from that isle, thou canst never return!”

- Herman Melville, Moby Dick

Chapter II

Introduction

Using the don't panic game, created earlier, this time we're going to use a database to persist some data along at the same time we use the vibrant technology of containerization. These technologies will help each other, as containerization will facilitate deployment of the application and future additions, as well as this new database.

Knowing that our current game gives 42 as the target number, now we are going to have to randomize the hidden equation that always evaluates to 42 daily. Every day someone accesses the game, a different hidden equation must be the answer. This is the plot necessary for us to implement a super important technology for the world: database. For this, you will have to understand and define how your game will communicate with this new system member in a few ways.

Chapter III

General instructions

- Humans will review your code.
- Up to date containerization is a must. Docker[1] containers to deploy your game with one command.
- Your current dontpanic_baby code should be containerized. We recommend that you try this first.
- You will have to use the database technology you want to learn about. We recommend that you don't be afraid to read documentation regarding this topic in order to be able to work with this technology.
- Since you could use any language, here you can also decide which integrations and libraries will be added so you can connect to your database. Think about your decision. Why use one instead of another available?
- Changes should only be made on the serverside. The clientside must remain the same, with the exception of being containerized.
- We therefore advise you not to be afraid to read a lot of documentation about Docker.

[1] https://www.docker.com/

Chapter IV

Mandatory part

IV.1 Clientside features

- Must be containerized.

IV.2 Serverside features

- Your serverside will connect directly with the database that will be deployed.
- Every day your hidden equation must change.
- The daily hidden equations must be stored in the database.
- Security is also important. Ensure that your application is not vulnerable to SQL Injection or other flaws that may exist once we add a database.
- Your application must still follow REST[1] principles, and use JSON[2] as a data exchange format, as mentioned the last time we saw each other. Any changes made to these points must be updated accordingly.
- The current documentation must be updated according to the new modifications.

[1]https://www.redhat.com/en/topics/api/what-is-a-rest-api
[2]https://www.w3schools.com/whatis/whatis_json.asp

IV.3 Database features

- You must use the database technology you want to learn about, we will suggest some of them:
  - PostgreSQL
  - MariaDB
  - MongoDB
  - ElasticSearch
- You are free to use any data structure in your database, but if using any database, it must follow some data structuring good practices:
  - define and name tables and columns (or equivalently) consistently.
  - use the appropriate data types for each attribute.
  - try to normalize the data as much as possible.
  - and lastly KISS! Keep it as stupidly simple.

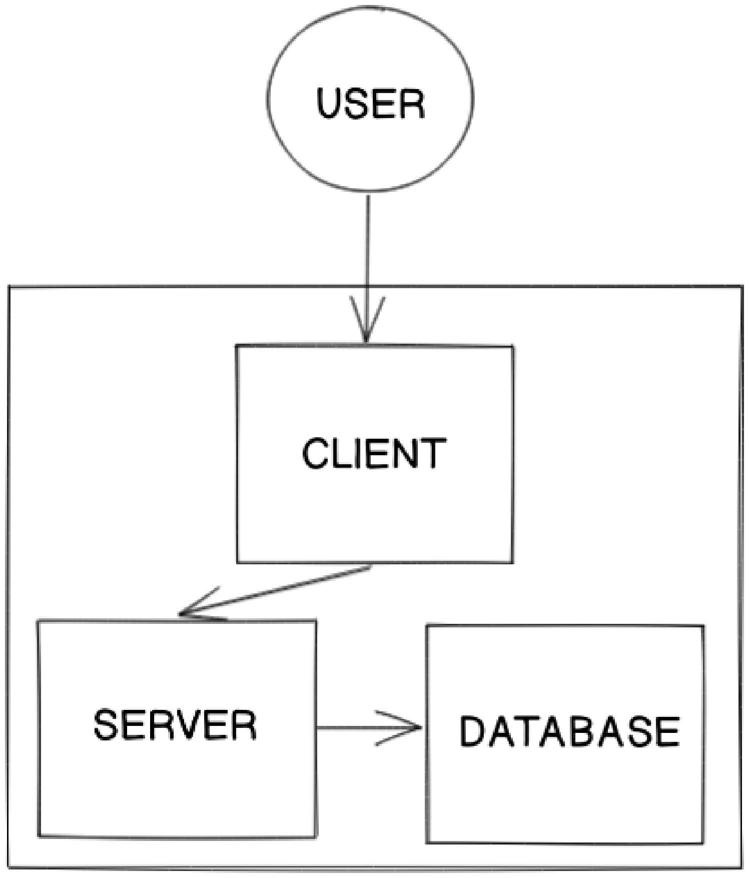
  In this sense of good practices, being naturally subjective, the most important thing is to know why you made the decision to structure your data in that way as well as explain it.

IV.4 Containerization features

- You must use Docker[3].
- Only the clientside service must be open to the world listening on port 8080. You should understand what this means and be able to explain how this happens in your implementation.
- You must have a container for each different service.
- You need to ensure that the necessary services can communicate with each other within containers. For that, you will need to explain how the Docker network works. It is strictly forbidden to use the 'host' network.
- Each container must bear the same name as the service concerned.
- Your containers should stop when an application also stops for some unexpected or expected reason.
- Even if the containers stop, the data must remain, as well as the target number of the day.
- You should always use the 'ubuntu:20.04' image as a base for Dockerfiles written by you.
- Every image must have a Dockerfile written by you, it is forbidden to take already built images. The only exception is the database container and 'ubuntu:20.04' for use as a base image.
- You should understand everything the Dockerfile does and be able to explain it.
- Docker-compose[4] is acceptable and highly recommended, but you should know what shenanigans it does for you, as well explain it.
- The data in the database container must persist even if you delete the container completely and recreate it.

[3] https://docs.docker.com/
[4] https://docs.docker.com/compose/

Here is an example of what you will need to set up:



IV.5 Tests features

- The tests must be updated according to the modifications made: following the same instructions as we saw before and testing the main functions, especially your serverside.

Chapter V

Bonus part

For the bonus to be valid, the mandatory part must be perfect.

When we're working with code, especially in a team of people with their hands on the same codebase, it's often important that we're able to share the same development environment across the team. Being an interesting thing, often not everyone has this luxury. The bonus is to offer this luxury and use a tool to be able to create a consistent development environment that is versioned along with your code.

You should use Vagrant[1] to make that in a command: create an environment where everything needed to work on your code is already there, as well as for your team.

The Vagrantfile file must be at the root of your repository and must use 'ubuntu/-focal64' and use VirtualBox as a provider[2]. For your Vagrantfile, you should be able to explain why you made the decisions you made to write it as you did. Remember that simplicity is the greatest sophistication!

[1]https://www.vagrantup.com/
[2]https://www.vagrantup.com/docs/providers/virtualbox

Chapter VI

Turn-in and peer-evaluation

VI.1 Turn-in

Turn in your work on your repo Git. Only the work included on your repo will be reviewed during the evaluation.

VI.2 Peer-evaluation

Another group will evaluate your game and your code and will positively welcome the quality of it.