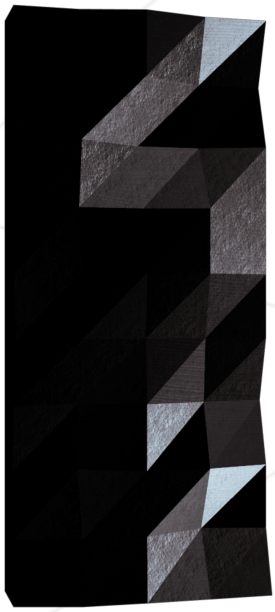


dontpanic_baby

The hidden equation may be closer than you think.

Summary: This project is about slicing big stuff into small deliverable stuff. It will contain the most basic implementation of the main technologies we can use, end-to-end.

Version: 1.0

Contents

Chapter I

Foreword

Vegan cupcake recipe:

I.1 Ingredients

150ml almond or soy milk
½ tsp cider vinegar
110g vegan butter or sunflower spread
110g caster sugar
1 tsp vanilla extract
110g self-raising flour
½ tsp baking powder

For the buttercream

125g vegan butter
250g icing sugar
1¼ tsp vanilla extract
a few drops of vegan food colourings (check the label)

I.2 Method

Step 1

Heat the oven to 180C/160C fan/gas 4. Line the holes of a 12-hole cupcake tin with paper cases. Stir the milk and vinegar in a jug and leave to thicken slightly for a few mins.

Step 2

Beat the butter and sugar with an electric whisk until well combined. Whisk in the vanilla, then add the milk a splash at a time, alternating with spoonfuls of the flour. Fold in any remaining flour, the baking powder and a pinch of salt until you get a creamy batter. Don't worry if it looks a little curdled at this stage.

STEP 3

Divide between the cupcake cases, filling them two-thirds full, and bake for 20 - 25 mins until golden and risen. Leave to cool on a wire rack.

STEP 4

To make the buttercream, beat the butter, icing sugar and vanilla with an electric whisk until pale and creamy. Divide between bowls and colour with different food colourings until you get desired strength. Spoon or pipe onto the cooled cupcakes.

```
Big cakes are too much for us.  We do prefer cupcakes.
```

Chapter II

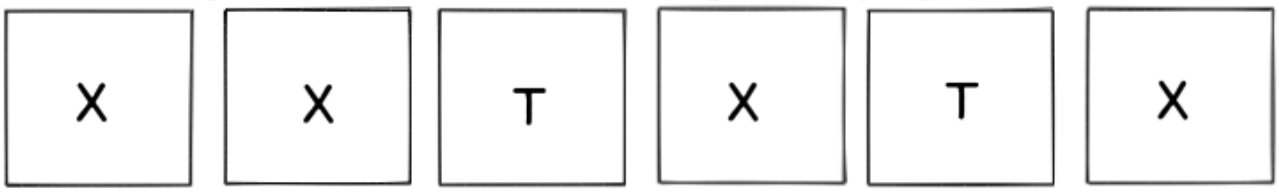
Introduction

In this project you will have to create a fun game that will make you think and learn about the main concepts and technologies in today's web world. Also, often overlooked, learning to write and use unit tests.

The game consists of guessing the hidden equation that results in "42". Every guess must be equal to this target number (42) and after each guess, tip characters will show how close you are. Let's take a look at an example:

There's a potential equation that can lead us to 42:

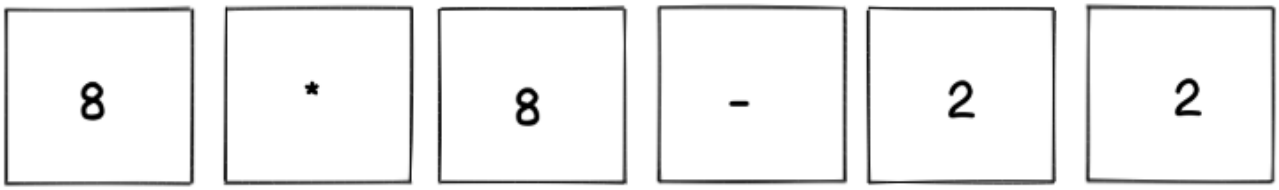
Unfortunately, this equation is not the hidden one that leads us to 42. So, gently the game show how close we are to the hidden equation:



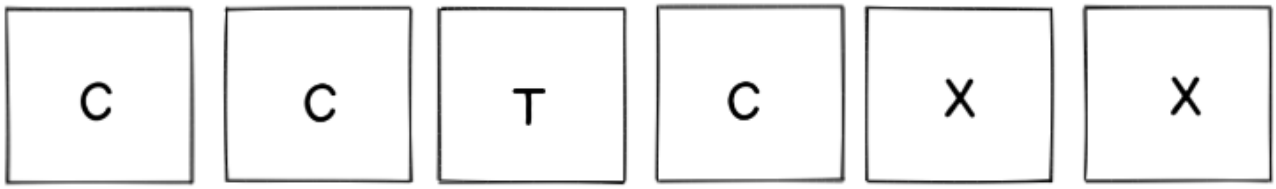
- The "T" means the character exists in the hidden equation, but is not in the correct spot.

- The "X" means the character is not in the hidden equation

With this information, I can now try again

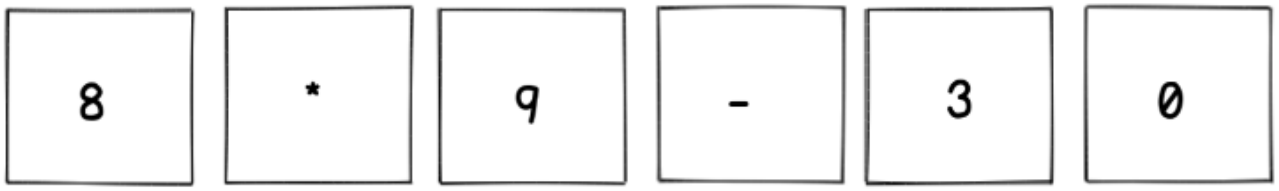


This is also not the hidden equation we are looking for, and we can see this analyzing what the game presented us after this try:



- The "C" means the character exists in the hidden equation, and it is in the right position
- The "T" and the "X" you already know what they mean, right?

With this new information, I now try:



And...

Great Scott! In this move I managed to find out the hidden equation needed to lead me to 42. That's it, I won.

Now that you know the mechanics of this game, you can proceed to implement it.

Chapter III

General instructions

- This project will be reviewed by humans.
- For this project you are free to use the language you want.
- Clientside, you will have to use HTML, CSS, and Javascript but only with browsers natives API[1]. No frameworks or libraries are acceptable, like the DOM manipulation library jQuery or React. KISS: Keep it stupidly simple!
- HTML code must not use "<table>" elements. It must be built exclusively from a grid or flexbox layout.
- Your web application must have a decent layout: at least a header, a main section and a footer.
- Your back-end must use a technology stack as close as possible to the technologies you want to learn about. Take care to not bring more complexity than needed. Over-engineering is the root of all evil.
- Still in serverside, if you need some inspirations, we will suggest as main languages:
    - Sinatra for Ruby.
    - Express for Node.
    - Flask for Python.
    - Scalatra for Scala.
    - Slim for PHP.
    - Nickel for Rust.
    - Goji for Golang.
    - Spark for Java.
    - Crow for C++.

[1] https://developer.mozilla.org/en-US/docs/Web/API

- Your web application must produce no errors, no warning or log line in any console, server side and client side. Nonetheless, due to the lack of HTTPS, any error related to that are tolerated.

- Your web application should be at least be compatible with Firefox (>= 91) and Chrome (>= 96).

- You are free to use any webserver you want, like NGINX or even the built-in webserver.

- You must write unit tests that cover the main functions. (see below)

Chapter IV

Mandatory part

IV.1 Common features

- The application must receive an input that will be sent by the user and that must have exactly 6 characters.
- When the equation is sent, the application must return to the user if it is the correct one, otherwise it must display how close the user is to the correct answer.
- The hidden equation is always the same (it is still a game to play just once).
- All equations must result in the target number (42), otherwise the equation will be rejected and no answer tip should be provided.
- Last attempts and hints about the hidden equation must always be present on the same screen.
- Of course you will have to handle the inputs in a safe way, so validations will need to exist for the input. Tests will help you with that.
- The accepted inputs are numbers (0-9), *, /, + and -.
- The screen must present somewhere a link pointing to a README.md file in the project repository root explaining the game.

IV.2 Serverside features

- All rules and definitions must be on serverside. The serverside is the reference, the representation of the truth.

- You should create an API that follows the principles of REST. You will have to be able to justify your choice and explain its characteristics.

- You should endorse JSON for your exchange format with the API. You will have to be able to justify your choice and explain its characteristics.

- You must temporarily store any necessary data in local memory. This means that there is no data persistence.

- You should use the main API documentation tool for your chosen programming language to document serverside and its routes.

IV.3 Clientside features

- The clientside must be able to connect with the serverside and respect it's definitions. Back-end's address must be configured on the clientside.

- It should contain an input to type characters to discover the hidden equation.

- With the characters placed, the user should be able to send his attempt to the serverside.

- It should be straight to the point and easy to understand what the user should do the first time accessing it.

IV.4 Tests features

The objective of the tests are:

- Increase the reliability of the delivered versions.
- To reduce the "time to market" by facilitating / automating the recipe for each new version.
- Reinforce customer satisfaction and the relevance of the delivered versions by allowing longer development cycles.
- Ensure that any changes or introductions of new functionality do not break any part of your code, thinking that you can test all parts of your code over and over again and verify that everything is working fine.

Regardless of the language you choose to do this, the testing tools ecosystem is already quite mature. This is a proposal for you to become familiar with unit tests, having to cover the main functions of your code.

Among the ways to cover your code with tests, the simplest way is by defined functions[1]. The idea being very simple, you need to call the functions defined in your code. That way, you already have functions covered by unit tests. Another thing is to ensure that the input sent to the function (thinking about parameterized functions) causes the function to return what it should. If what should be returned was not returned then the test failed and you should understand why and look for ways to solve this possible bug.

For testing, likewise, if you are uninspired, here are some tools to use:

- Shell script.
- Pytest for Python.
- Standard library testing for Golang.
- Catch2 for C++.
- Jest for JavaScript.

Using TDD[2] you'll be able to test in a good way, avoiding Big Design Up-Front (BDUF) and designing your code in a test oriented fashion.

[1] https://en.wikipedia.org/wiki/Code_coverage
[2] https://en.wikipedia.org/wiki/Test-driven_development

Chapter V

Turn-in and peer-evaluation

V.1 Turn-in

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

V.2 Peer-evaluation

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