Our First Controller

Symfony uses the MVC structure, so we're going to need controllers.

Creating a Controller

With Symfony, we rarely create controller files manually. We usually use the CLI:

```
symfony console make:controller
# Or specify the controller name directly:
symfony console make:controller ControllerName
```

This command is quite self-explanatory: we're asking Symfony to use the console to generate a controller.

Symfony will then ask you for a name — we'll choose **HomeController** here.

Every controller must include the word "Controller" in its name. If you forget it or make a typo, Symfony will automatically add "Controller" to the name.

Next, it asks whether you want to generate PHPUnit tests (marked as "Experimental").
For now, just say no.

Symfony then displays two lines:

```
created: src/Controller/HomeController.php
created: templates/home/index.html.twig
```

This means it created both the controller and a template.
Let's ignore the template for now and focus on the controller.

Structure of a Controller

Let's break down the controller file line by line:

- First, a namespace.
- Then some `\use` statements to import classes (no need for `\require`; Symfony uses Composer's autoloader).
- Then a class with the same name as the file, extending an abstract class that provides helper methods for controllers.
  - Inside this class is an `index` method. The name isn't very important.
  - Above the method, there's a `Route` attribute (`#[...]` is a PHP 8+ attribute, not a comment).
    - Any method with this attribute becomes its own page (route).
    - You can change the name of the route — just don't change it later once it's in use.
    - The path string (like `/blog`) can be changed anytime — Symfony will update it everywhere automatically.

        - We'll use the route's name to generate all internal links in the project.
    - The method currently does one thing: it calls the `render()` method inherited from `AbstractController`.
        - The first parameter is the view to display (it automatically looks in the `templates/` folder).
        - The second parameter is an associative array of variables passed to the view (other PHP variables will not be accessible in the view).

We'll explore more controller capabilities later. For now, let's focus on the view.

Note: Routes can also be defined in `config/routes.yaml` instead of above the method.

Before we go, let's pass a few variables to the view, including an array and a random number:

```
[
  'controller_name' => 'HomeController',
  "fruits" => ["banana", "tomato", "cherry"],
  "pays" => ["france" => "Bonjour le monde!", "england" => "Hello World!"],
  "chiffre" => rand(0, 10),
  "vide" => [],
  "xss" => "<script>alert('Hi');</script>"
]
```

And let's change the route path to `/` and its name to `app_home`.