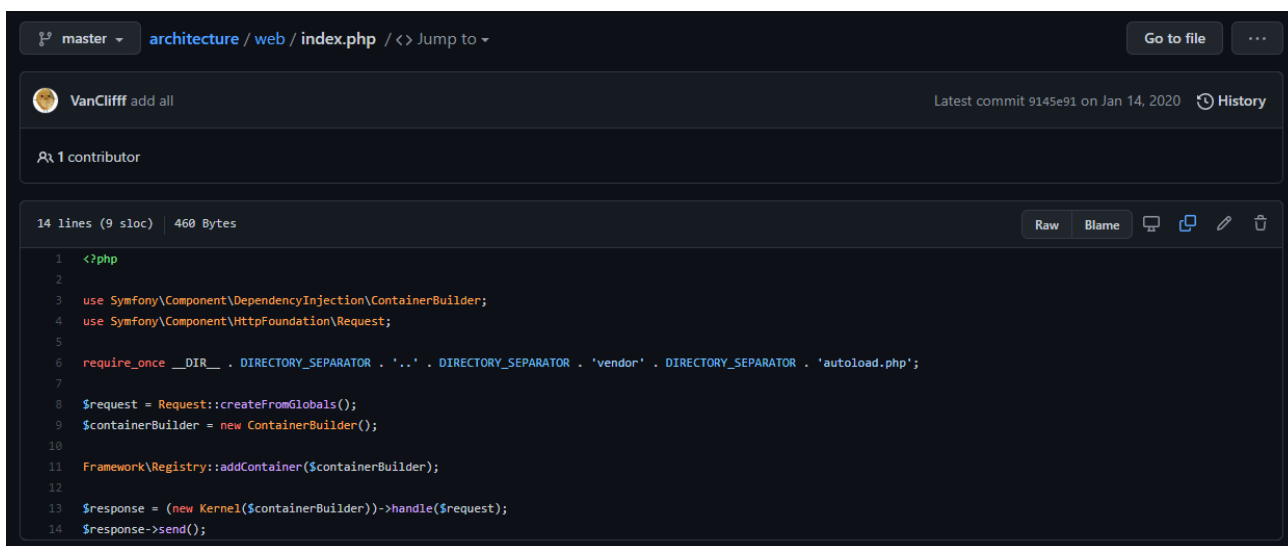


Домашнее задание к Уроку 7. Шаблоны корпоративных приложений

1. Найти и указать в проекте Front Controller и расписать классы, которые с ним взаимодействуют.

В данном проекте в роли Front controller-а выступает файл index.php, который находится в директории **web/**.



The screenshot shows a code editor interface for a file named `index.php` located at `architecture / web /`. The file is 14 lines long, 9 sloc, and 460 Bytes. The code is as follows:

```
1 <?php
2
3 use Symfony\Component\DependencyInjection\ContainerBuilder;
4 use Symfony\Component\HttpFoundation\Request;
5
6 require_once __DIR__ . DIRECTORY_SEPARATOR . '..' . DIRECTORY_SEPARATOR . 'vendor' . DIRECTORY_SEPARATOR . 'autoload.php';
7
8 $request = Request::createFromGlobals();
9 $containerBuilder = new ContainerBuilder();
10
11 Framework\Registry::addContainer($containerBuilder);
12
13 $response = (new Kernel($containerBuilder))->handle($request);
14 $response->send();
```

Это именно та часть проекта, через которую проходят любые возможные запросы. В файле `index.php` также происходит создание экземпляра (инстанции) класса **Kernel**. Инстанция **Kernel**-а, в свою очередь, занимается обработкой запросов и предоставляет ответы (Response) на них. В процессе обработки запроса (метод `handle()`), экземпляр класса **Kernel**, регистрирует роуты (**Routes**) посредством обработки файла routing.php. В данном файле указаны роуты, а также классы контроллеров, которые должны их обрабатывать. В данном проекте существуют следующие классы контроллеров, экземпляры которых могут быть созданы в случае необходимости (в процессе обработки определенного роута):

- **MainController**
- **ProductController**
- **OrderController**
- **UserController**

Также, в файле `index.php` создается экземпляр класса **ContainerBuilder**, который, насколько я понял, предназначен для хранения данных о сервисах, других данных, коллекции роутов,

параметров конфигурации приложения, переменных окружения и тд. Например, **ContainerBuilder** передается в конструктор экземпляра Kernel-а и в нем он будет использован, например, в методе registerRoutes() для хранения коллекции роутов.

Также, в файле index.php, экземпляр класса **ContainerBuilder** передается в качестве аргумента в статический метод класса **Registry addContainer()**.

Registry это пример реализации паттерна **Registry** в данном приложении.

2. Найти в проекте паттерн Registry и объяснить, почему он был применён.

Насколько я смог разобраться, паттерн **Registry** в данном проекте был реализован через класс **Registry**.

```
1  <?php
2
3  declare(strict_types = 1);
4
5  namespace Framework;
6
7  use Symfony\Component\DependencyInjection\ContainerBuilder;
8  use Symfony\Component\Routing\Generator\UrlGenerator;
9  use Symfony\Component\Routing\RequestContext;
10 use Symfony\Component\Routing\RouteCollection;
11
12 class Registry
13 {
14     /** @var ContainerBuilder ...*/
15     private static $containerBuilder;
16
17     /** Добавляем контейнер для работы реестра ...*/
18
19     public static function addContainer(ContainerBuilder $containerBuilder): void{...}
20
21     /** Получаем данные из конфигурационного файла ...*/
22
23     public static function getDataConfig(string $name){...}
24
25     /** Рендерим страницу по названию роута ...*/
26
27     public static function getRoute(string $name, array $parameters = []): string{...}
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
```

Данный класс оперирует данными через одно из своих свойств, статическую переменную **\$containerBuilder**. Через статический метод **addContainer()** данная переменная устанавливается, а через статические методы **getDataConfig()** и **getRoute()** из экземпляра класса **ContainerBuilder**, **\$containerBuilder**, возвращаются данные (посредством использования методов самой переменной).

Таким образом, разные компоненты приложения могут получить данные конфигурационного файла и роут. Так например делает метод **render()**.

3. Добавить во все классы **Repository** использование паттерна **Identity Map** вместо постоянного генерирования сущностей.

Файлы с решением данного задания находятся одной директории с этим документом.

Названия файлов:

- **Product.php** (метод **search()**)
- **User.php** (метод **getById()** и метод **getByLogin()**)