

Clean Architecture em PHP

Elton Minetto

@eminetto

O que é **Clean Architecture**?

- <https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html>
- <https://www.amazon.com/Clean-Architecture-Craftsmans-Software-Structure/dp/0134494164>

Premissas

- Independente de frameworks
 - Testável
 - Independente de UI
 - Independente de Database
- Independente qualquer agente externo

**Divide nosso código em 4
camadas**

- **Entities:** representam as entidades das regras de negócio
- **Use Cases:** as regras de negócio da aplicação

- **Controller:** adaptam e convertem os dados do formato usado pelas entidades e use cases para agentes externos como bancos de dados , web, etc
- **Framework & Driver:** frameworks e ferramentas como bancos de dados, frameworks web, etc

Clean architecture em PHP

No namespace *entity* estão nossas entidades


```
ls -lh src/Bookmark/Entity
```

```
-rw-r--r-- 1 eminetto staff 223B May 28 21:12 Bookmark.php
```

```
<?php
declare(strict_types=1);

namespace Bookmark\Entity
;
class Bookmark
{
    public $id;
    public $name;
    public $description;
    public $link;
    public $tags = [];
    public $favorite;
    public $createdAt;
}
```

No namespace *UseCase* temos a definição das interfaces dos *Use Case*

```
ls -lh src/Bookmark/UseCase
```

```
-rw-r--r--  1 eminetto  staff   954B Jun  4 21:41 Service.php
```

```
-rw-r--r--  1 eminetto  staff   425B Jun  4 21:41 UseCaseInterface.php
```

```
<?php

namespace Bookmark\UseCase;

use Bookmark\Entity\Bookmark;
use Bookmark\Driver\RepositoryInterface;

interface UseCaseInterface
{
    public function __construct(RepositoryInterface $repository);

    public function search(string $query);

    public function findAll();

    public function store(Bookmark $bookmark): int;

    public function delete(int $id) : bool;
}
```

O *Service.php* é a implementação dos *Use Case*

```
<?php
namespace Bookmark\UseCase;

use Bookmark\Entity\Bookmark;
use Bookmark\Driver\RepositoryInterface;

class Service implements UseCaseInterface
{
    private $repository;

    public function __construct(RepositoryInterface $repository)
    {
        $this->repository = $repository;
    }

    public function search(string $query)
    {
        return $this->repository->search($query);
    }

    public function findAll()
    {
        return $this->repository->findAll();
    }

    public function store(Bookmark $bookmark): int
    {
        $bookmark->createdAt = new \Datetime();
        return $this->repository->store($bookmark);
    }

    public function delete(int $id) : bool
    {
        $b = $this->repository->find($id);
        if ($b->favorite) {
            return false;
        }
        return $this->repository->delete($id);
    }
}
```

No namespace *Driver* temos a camada correspondente, neste caso os repositórios onde as entidades serão armazenadas


```
ls -lh src/Bookmark/Driver
```

```
-rw-r--r--  1 eminentto  staff   857B Jun  4 21:40 InmemRepository.php  
-rw-r--r--  1 eminentto  staff   320B Jun  4 21:40 RepositoryInterface.php  
-rw-r--r--  1 eminentto  staff  2.9K Jun  4 21:40 SqliteRepository.php  
-rw-r--r--  1 eminentto  staff   406B Jun  4 21:47 SqliteRepositoryFactory.php
```

```
<?php

namespace Bookmark\Driver;

use Bookmark\Entity\Bookmark;

interface RepositoryInterface
{
    public function find(int $id) : Bookmark;

    public function search(string $query);

    public function findAll();

    public function store(Bookmark $bookmark): int;

    public function delete(int $id) : bool;
}
```

Nos arquivos *InmemRepository.php* e *SqliteRepository.php* temos implementações da interface

```
<?php

namespace Bookmark\Driver;

use Bookmark\Entity\Bookmark;

class SqliteRepository implements RepositoryInterface
{
    private $conn;

    public function __construct(\PDO $conn)
    {
        $this->conn = $conn;
        $this->conn->exec("CREATE TABLE IF NOT EXISTS bookmarks (
            id INTEGER PRIMARY KEY,
            name TEXT,
            description TEXT,
            link TEXT,
            tags TEXT,
            favorite integer,
            created_at integer)");
    }

    public function find(int $id) : Bookmark
    {
        $result = $this->conn->query("SELECT * FROM bookmarks where id =$id");
        $b = new Bookmark;
        $b->id = $m[0]['id'];
        $b->name = $m[0]['name'];
        $b->description = $m[0]['description'];
        $b->link = $m[0]['link'];
        $b->tags = explode(",", $m[0]['tags']);
        $b->favorite = $m[0]['favorite'];
        return $b;
    }
}

//restante do arquivo oculto para facilitar a leitura do slide
```

No diretório *Controller* temos a implementação da camada correspondente

```
ls -lh src/Bookmark/Controller
```

```
-rw-r--r--  1 eminetto  staff   509B Jun  4 21:43 HandlerFactory.php
```

```
-rw-r--r--  1 eminetto  staff   638B Jun  4 21:40 IndexHandler.php
```

```
-rw-r--r--  1 eminetto  staff  1.0K Jun  4 21:40 PostHandler.php
```

```
<?php

declare(strict_types=1);

namespace Bookmark\Controller;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\RequestHandlerInterface;
use Zend\Diactoros\Response\JsonResponse;
use Bookmark\UseCase\UseCaseInterface;

class IndexHandler implements RequestHandlerInterface
{
    private $service;

    public function __construct(UseCaseInterface $service)
    {
        $this->service = $service;
    }

    public function handle(ServerRequestInterface $request) :
ResponseInterface
    {
        $all = $this->service->findAll();
        return new JsonResponse($all);
    }
}
```

Podemos também ter diferentes controllers, como a
linha de comando


```
ls -lh cli
```

```
-rw-r--r-- 1 eminetto staff 357B Jun 4 21:42 search.php
```

```
<?php
```

```
require 'vendor/autoload.php';
```

```
$container = require 'config/container.php';
```

```
$repo = $container->get(Bookmark\Driver\SqliteRepository::class);
```

```
$service = new Bookmark\UseCase\Service($repo);
```

```
$result = $service->search($argv[1]);
```

```
foreach ($result as $key => $value) {
```

```
    printf("ID: %s Name: %s URL: %s \n", $value->id, $value->name, $value->link);
```

```
}
```

Podemos facilmente testar nossos pacotes, camada a camada

```
cd test/BookmarkTest ; tree
.
|_----Driver
|  |_----SQLiteRepositoryFactoryTest.php
|  |_----SQLiteRepositoryTest.php
|_----Controller
|  |_----IndexHandlerTest.php
|  |_----HandlerFactoryTest.php
|  |_----PostHandlerTest.php
|_----UseCase
|  |_----ServiceTest.php
```

```
<?php

declare(strict_types=1);

namespace BookmarkTest\Driver;

use Bookmark\Driver\SqliteRepository;
use PHPUnit\Framework\TestCase;
use Bookmark\UseCase\Service;
use Bookmark\Entity\Bookmark;

class SqliteRepositoryTest extends TestCase
{
    private $conn;
    private $repo;

    public function setUp()
    {
        $this->conn = new \PDO('sqlite::memory:');
        $this->repo = new SqliteRepository($this->conn);
    }

    public function testStore()
    {
        $b = new Bookmark;
        $b->name = 'Elton Minetto';
        $b->description = 'Minettos page';
        $b->link = 'http://www.eltonminetto.net';
        $b->tags = ["golang", "php", "linux", "mac"];
        $b->createdAt = new \Datetime();
        $b->favorite = true;

        $id = $this->repo->store($b);
        $this->assertEquals(1, $id);
    }

    //restante do arquivo oculto para facilitar a leitura do slide
}
```

```
<?php

declare(strict_types=1);

namespace BookmarkTest\UseCase;

use Bookmark\Driver\InmemRepository;
use PHPUnit\Framework\TestCase;
use Bookmark\UseCase\Service;
use Bookmark\Entity\Bookmark;

class ServiceTest extends TestCase
{
    private $repo;
    private $service;

    public function setup()
    {
        $this->repo = new InmemRepository;
        $this->service = new Service($this->repo);
    }

    public function testStore()
    {
        $b = new Bookmark;
        $b->name = 'Elton Minetto';
        $b->description = 'Minettos page';
        $b->link = 'http://www.eltonminetto.net';
        $b->tags = ["golang", "php", "linux", "mac"];
        $b->favorite = true;

        $id = $this->service->store($b);
        $this->assertEquals(1, $id);
    }

    //restante do arquivo oculto para facilitar a leitura do slide
}
```

```
<?php

declare(strict_types=1);






namespace BookmarkTest\Controller;

use Bookmark\Controller\IndexHandler;
use PHPUnit\Framework\TestCase;
use Psr\Container\ContainerInterface;
use Psr\Http\Message\ServerRequestInterface;
use Zend\Diactoros\Response\JsonResponse;
use Bookmark\UseCase\UseCaseInterface;
use Bookmark\Driver\SqliteRepository;
use Bookmark\Driver\RepositoryInterface;

class IndexHandlerTest extends TestCase
{
    public function testReturnsJsonResponse()
    {
        $container = $this->prophesize(ContainerInterface::class);
        $container
            ->get(SqliteRepository::class)
            ->willReturn($this->prophesize(RepositoryInterface::class));
        $service = $this->prophesize(UseCaseInterface::class);
        $service->findAll()->willReturn([]);
        $indexPath = new IndexHandler($service->reveal());
        $response = $indexPath->handle(
            $this->prophesize(ServerRequestInterface::class)->reveal()
        );

        $this->assertInstanceOf(JsonResponse::class, $response);
    }
}
```

Premissas

- Independente de frameworks 
- Testável 
- Independente de UI 
- Independente de Database 
- Independente qualquer agente externo 

Exemplo completo

<https://github.com/eminetto/clean-architecture-php>

Perguntas

<http://eltonminetto.net>
@eminetto

<http://coderochr.com>

<http://codenation.com.br>

<http://asemanaphp.com.br>