Nastavenie a konfigurácia aplikácie v Laraveli

KI/BT/22

Mgr. Dávid Držík – david.drzik@ukf.sk

Laravel

- populárny open-source **PHP framework** určený na vývoj webových aplikácií
- vytvorený s cieľom uľahčiť a zrýchliť vývoj pomocou elegantnej syntaxe a vstavaných nástrojov
- poskytuje robustnú **architektúru MVC** (Model-View-Controller), ktorá podporuje škálovateľnosť a jednoduchú údržbu kódu
- medzi hlavné výhody patrí
  - jednoduchá práca s databázou pomocou Eloquent ORM,
  - intuitívny routing, podpora migrácií a zabudovaná autentifikácia

Požiadavky na spustenie Laravelu

- Laravel je PHP framework, a preto si vyžaduje:
  - **Webový server** – Laravel môže bežať na rôznych webových serveroch, ako napríklad Apache alebo Nginx
  - **PHP** – pre **Laravel 11** je požadovaná verzia **PHP 8.2** alebo vyššia
  - **Composer** – správca balíčkov pre PHP, ktorý sa používa na inštaláciu a správu závislostí Laravelu
  - **Databázový server** – Laravel podporuje rôzne databázové systémy, ako napríklad MySQL, PostgreSQL, SQLite a SQL Server.

Na webový server sme používali XAMPP, ale čo je to Docker?

- Docker je open-source platforma na vývoj, nasadenie a beh aplikácií v **kontajneroch**
  - kontajnery sú ľahké, izolované prostredia, ktoré obsahujú všetky závislosti aplikácie (kód, knižnice, runtime) a umožňujú jej konzistentné spustenie na **rôznych prostrediach** bez ohľadu na operačný systém hostiteľa
- Docker využíva technológiu **kontajnerizácie**, ktorá poskytuje lepšiu efektivitu oproti tradičným virtuálnym strojom (VMs), pretože bežia priamo na jadre operačného systému a nevyžadujú samostatné inštancie OS pre každú aplikáciu

Aký je rozdiel medzi XAMPP a Docker?

- nástroje, ktoré môžeme použiť na spustenie webových aplikácií na lokálnom prostredí
- **XAMPP**
  - predkonfigurovaný softvérový balík, ktorý obsahuje: Apache HTTP Server, MySQL/MariaDB databázu, PHP + Perl a phpMyAdmin
  - vhodnejší pre jednotlivcov a malé teamy na malých projektoch
- **Docker**
  - nástroj, ktorý umožňuje spustiť izolované kontajnery s aplikáciami a ich závislosťami bez ohľadu na konkrétny programovací jazyk

Jednoduchšie vysvetlenie rozdielu

- **XAMPP**
  - XAMPP je ako balík programov, ktorý si nainštalujeme na svoj počítač
  - keď ho spustíme, tak sa zapne Apache2 (webserver), MySQL (databáza) a PHP (jazyk na tvorbu webu)
  - všetko beží priamo na systéme (Windows)
- **Docker**
  - predstavme si ho ako virtuálne škatuľky, kde v jednej beží Apache2, v druhej MySQL a v tretej PHP – ale každá je oddelená
  - nič neinštalujeme priamo do systému – len spustíme kontajner a všetko beží v izolovanom prostredí
  - môžeme mať rôzne verzie Apache2, MySQL a PHP bez konfliktov, ale je to trochu zložitejšie na nastavenie oproti XAMPPu

Dockerfile a docker-compose.yml

- Dockerfile je **textový súbor** obsahujúci inštrukcie, ktoré definujú, ako sa má vytvoriť Docker image
  - tento súbor sa používa pri zostavovaní vlastného obrazu (docker build)
- docker-compose.yml je súbor na definovanie a orchestráciu viacerých Docker kontajnerov
  - používa sa s príkazom **docker compose up** na spustenie aplikácie so všetkými službami

Docker-compose.yml 1/3

```
version: '3.8'  # Docker Compose už nevyžaduje atribút 'version', takže ho môžeš odstrániť

networks:
  teamovy:  # Definícia vlastnej siete s názvom "teamovy", v ktorej budú všetky služby komunikovať

services:

  apache2:
    image: php:8.2-apache  # Použije oficiálny PHP image s Apache serverom, verzia PHP 8.2
    container_name: WebServer  # Pomenuje kontajner ako "WebServer"
    ports:
      - "8080:80"  # Presmerovanie portov: Lokálny port 8080 → Port 80 v kontajneri (Apache beží na 80)
    volumes:
      - ./src:/var/www/html  # Mapovanie lokálneho priečinka "src" do "/var/www/html" v kontajneri
    depends_on:
      mysql:
        condition: service_healthy  # Počká, kým MySQL prejde health checkom (teda kým je skutočne pripravená)
    networks:
      - teamovy  # Pripojenie na sieť "teamovy"
```

Docker-compose.yml 2/3

```
mysql:
  image: mysql:8  # Použije oficiálny MySQL image, verzia 8
  container_name: Database  # Pomenuje kontajner ako "Database"
  restart: unless-stopped  # Kontajner sa automaticky reštartuje iba v prípade, že sa zastaví neočakávane
  tty: true  # Povolenie interaktívneho režimu (užitočné pri ladení)
  ports:
    - "4306:3306"  # Presmerovanie portov: Lokálny port 4306 → Port 3306 v kontajneri (MySQL štandardne beží na 3306)
  volumes:
    - ./config/mysql:/var/lib/mysql  # Uloženie MySQL dát mimo kontajnera, aby sa nestratili po vypnutí
  environment:
    MYSQL_DATABASE: ${MYSQL_DATABASE}  # Premenné pre konfiguráciu MySQL, hodnoty sa berú z .env súboru
    MYSQL_USER: ${MYSQL_USER}
    MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]  # Test, či MySQL správne beží
    interval: 10s  # Kontrola každých 10 sekúnd
    retries: 5  # Po 5 neúspešných pokusoch sa kontajner považuje za nefunkčný
  networks:
    - teamovy  # Pripojenie na sieť "teamovy"
```

Docker-compose.yml 3/3 a ENV

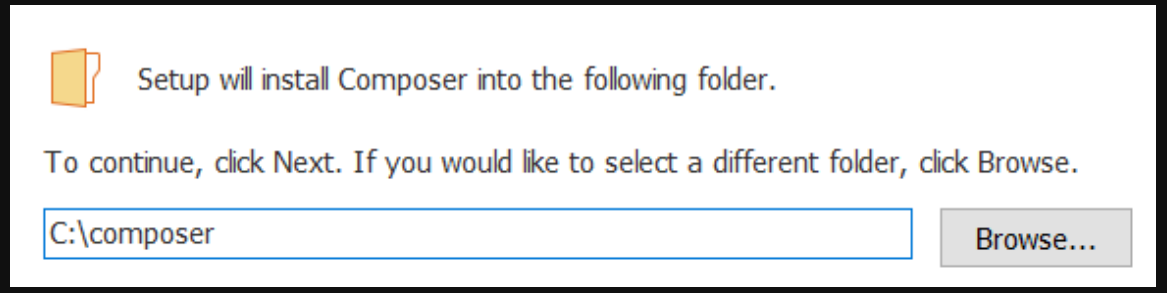
```
phpmyadmin:
  image: phpmyadmin/phpmyadmin  # Použije oficiálny phpMyAdmin image
  container_name: PhpMyAdmin  # Pomenuje kontajner ako "PhpMyAdmin"
  restart: unless-stopped  # Automaticky sa reštartuje, ak sa neočakávane zastaví
  ports:
    - "8081:80"  # Presmerovanie portov: Lokálny port 8081 → Port 80 v kontajneri (phpMyAdmin beží na 80)
  environment:
    PMA_HOST: mysql  # Nastaví MySQL server, ku ktorému sa phpMyAdmin pripája (musí sa zhodovať s názvom služby MySQL)
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}  # Použije heslo MySQL root používateľa z .env súboru
  networks:
    - teamovy  # Pripojenie na sieť "teamovy"
```

```
MYSQL_DATABASE=laravel
MYSQL_USER=laravel
MYSQL_PASSWORD=test
MYSQL_ROOT_PASSWORD=123
```

Použitie XAMPP a Composer

- na uľahčenie vývoja budeme používať **XAMPP**
  - balík, ktorý obsahuje Apache server, MySQL databázu a podporu pre PHP
  - minimum **PHP 8.2**
  - 

- **Composer** je nástrojom na správu dependencies v PHP
  - pomocou neho môžeme rýchlo a efektívne inštalovať Laravel a ďalšie knižnice potrebné pre vývoj aplikácií
  - https://getcomposer.org/download/
  - stiahnuť a nainštalovať
    - najlepšie aktuálnu verziu



Pred inštaláciou Laravelu

- overíme, či všetko funguje ako má

```
C:\Users\admin>php --version
PHP 8.2.12 (cli) (built: Oct 24 2023 21:15:15) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.2.12, Copyright (c) Zend Technologies

C:\Users\admin>composer --version
Composer version 2.8.5 2025-01-21 15:23:40
PHP version 8.2.12 (C:\xampp\php\php.exe)
Run the "diagnose" command to get more detailed diagnostics output.
```

- ak nie, treba hľadať problém
  - najčastejšie OS nevie nájsť PHP alebo composer, treba nastaviť premenné prostredia
  - alebo treba dočasne vypnúť Váš antivírusový program, ktorý blokuje SSL

Inštalácia Laravelu cez CMD

- presuňme sa do adresára XAMPPu, ktorú server vníma ako root zložku
  - cd C:\xampp\htdocs
- nainštalujeme čistý laravel projekt:
  - composer create-project --prefer-dist laravel/laravel **myproject** "**11.***"
- ak inštalácia prebehla úspešne, uvidíme v konzole na záver toto

Názov projektu

Verzia

```
  INFO  Running migrations.

  0001_01_01_000000_create_users_table ..... 34.19ms DONE
  0001_01_01_000001_create_cache_table ..... 4.83ms DONE
  0001_01_01_000002_create_jobs_table ..... 12.25ms DONE


C:\xampp\htdocs>
```

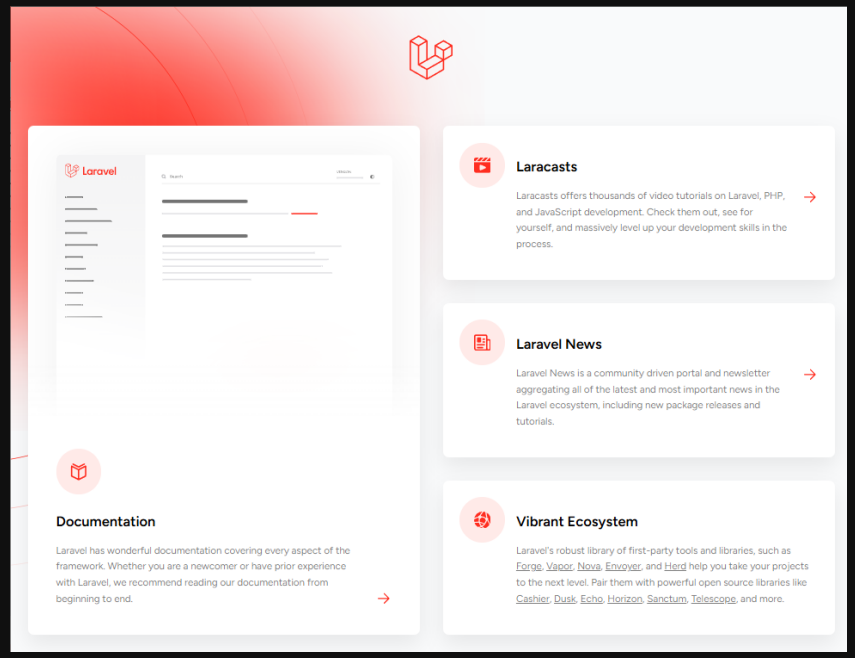
Možné problémy

- ak sme narazili na niečo takéto:
  - *Failed to download nesbot/carbon from dist: The zip extension and unzip/7z commands are both missing, skipping. The php.ini used by your command-line PHP is: C:\xampp\php\php.ini Now trying to download from source - Syncing nesbot/carbon (3.8.4) into cache Failed to download monolog/monolog from dist: The zip extension and unzip/7z commands are both missing, skipping.*
- riešenie je jednoduché:
  - v súbore „C:\xampp\php**php.ini**“ odstránime bodkočiarku na tomto riadku
    - ;extension=zip
  - a reštartujeme XAMPP

IDE (Integrated Development Environment)

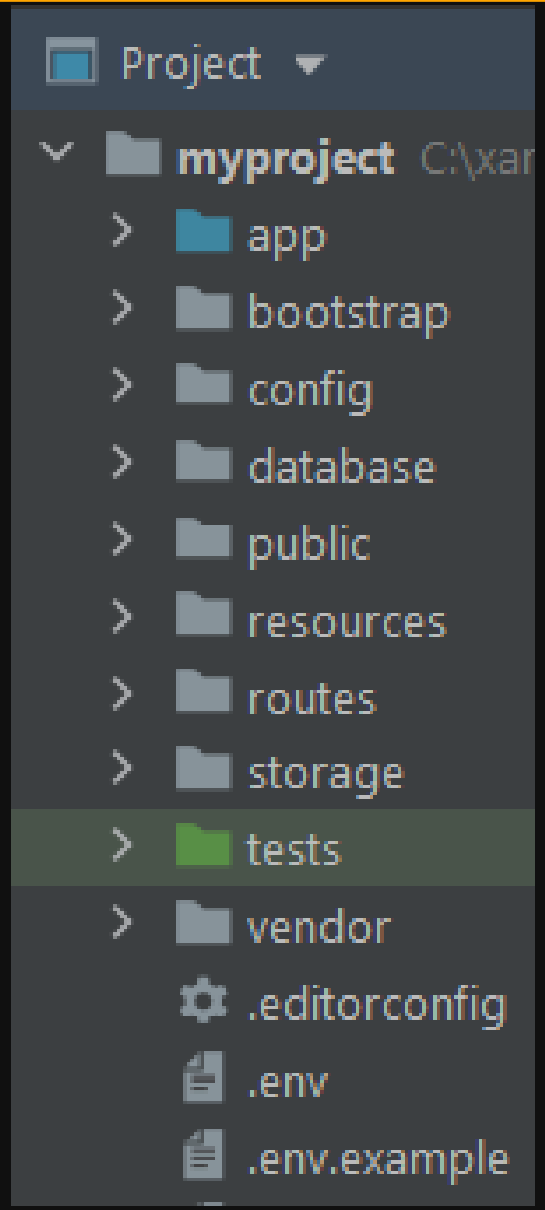
- otvoríme projekt (celý priečinok) v PhpStorme
  - alternatívne je možné využívať aj VS Code
- odteraz budeme využívať konzolu v IDE
- spustíme aplikáciu príkazom:
  - **php artisan serve**

```
Terminal:   Local  ×  +
Microsoft Windows [Version 10.0.19045.5371]
(c) Microsoft Corporation. Všetky práva vyhradené.

C:\xampp\htdocs\myproject>php artisan serve
  forking is not supported on this platform

  INFO  Server running on [http://127.0.0.1:8000].

  Press Ctrl+C to stop the server
```



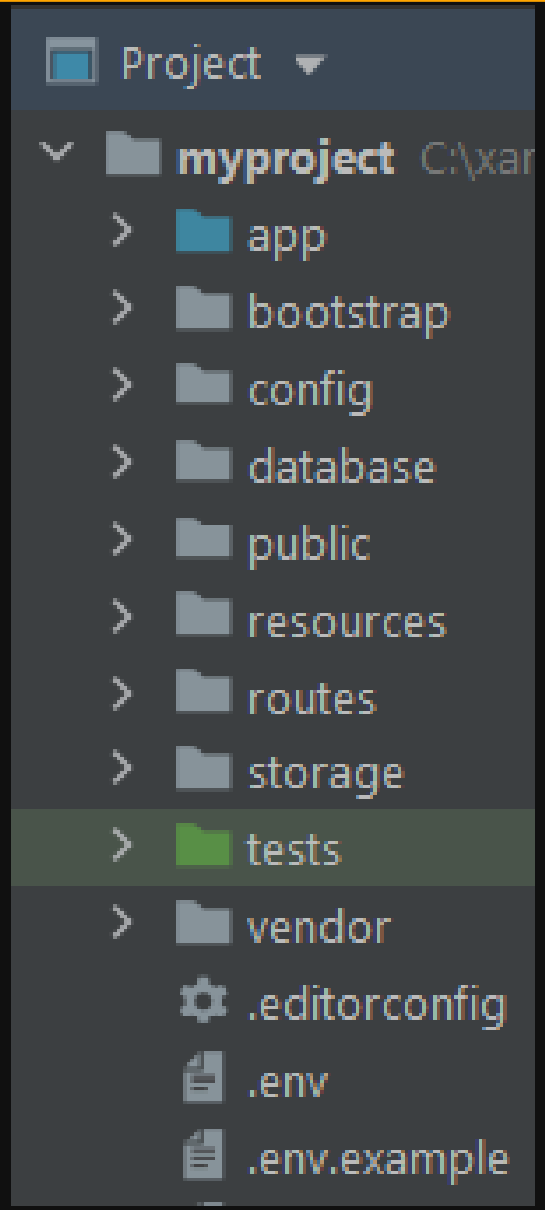
Ak je všetko OK, vidíme toto na http://127.0.0.1:8000/

Štruktúra Laravel projektu 1/7



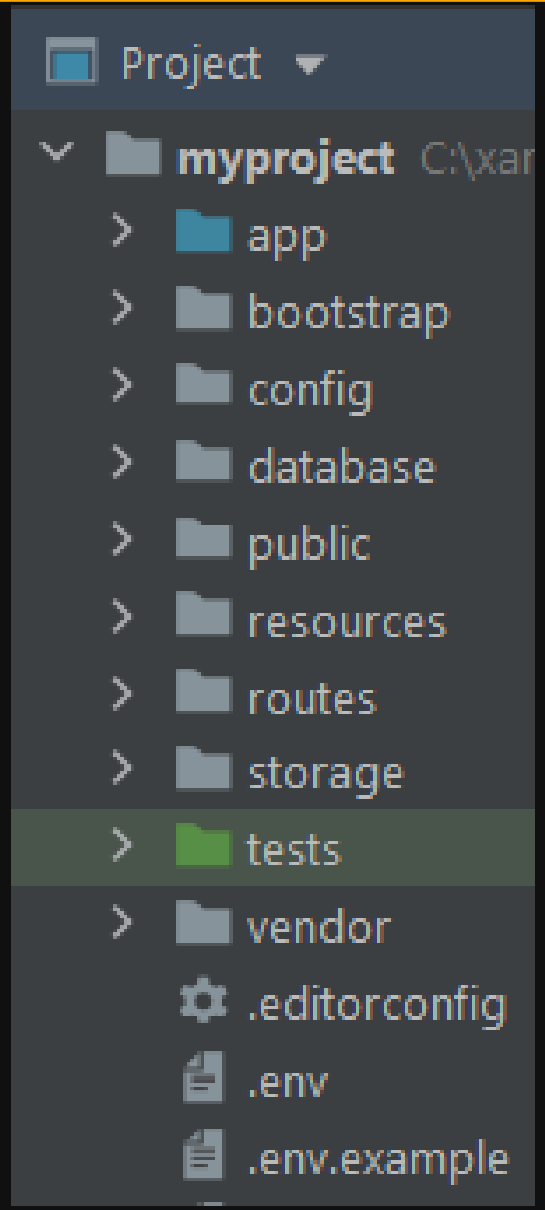
- **1. app/** – obsahuje všetky modely, kontroléry, služby a ďalšie triedy, ktoré tvoria logiku backend aplikácie
  - **Models/** – Obsahuje Eloquent modely (napr. User.php). Modely reprezentujú tabuľky v databáze a poskytujú ORM funkcionalitu.
  - **Http/Controllers/** – Obsahuje kontroléry, ktoré spracúvajú HTTP požiadavky a odovzdávajú údaje pohľadom alebo API odpovediam.
  - **Http/Middleware/** – Obsahuje middleware triedy, ktoré umožňujú manipuláciu s požiadavkami pred ich spracovaním v kontroléri (napr. autentifikácia, CORS).
  - **Providers/** – Laravel Service Providers, ktoré registrujú služby a konfigurácie. Každý Laravel projekt má napr. AppServiceProvider.php, AuthServiceProvider.php.

Štruktúra Laravel projektu 2/7



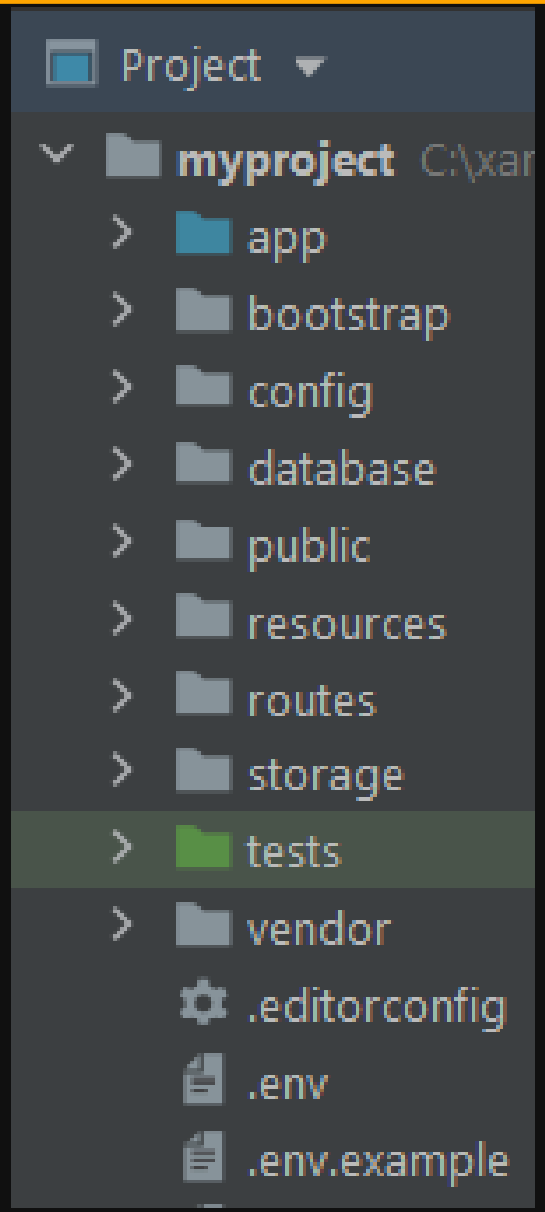
- **2. bootstrap/** – Inicializácia aplikácie
  - obsahuje súbor **app.php**, ktorý spúšťa Laravel aplikáciu
  - nachádza sa tu aj **cache/**, kde sa ukladajú optimalizované verzie konfigurácií a routing informácií
- **3. config/** – Konfiguračné súbory, napr.:
  - **app.php** – základná konfigurácia aplikácie (timezone, locale, debug mode)
  - **database.php** – konfigurácia pripojenia na databázu
  - **queue.php** – nastavenia spracovania frontov
  - **logging.php** – definícia logovacích kanálov (file, Slack, syslog)

Štruktúra Laravel projektu 3/7



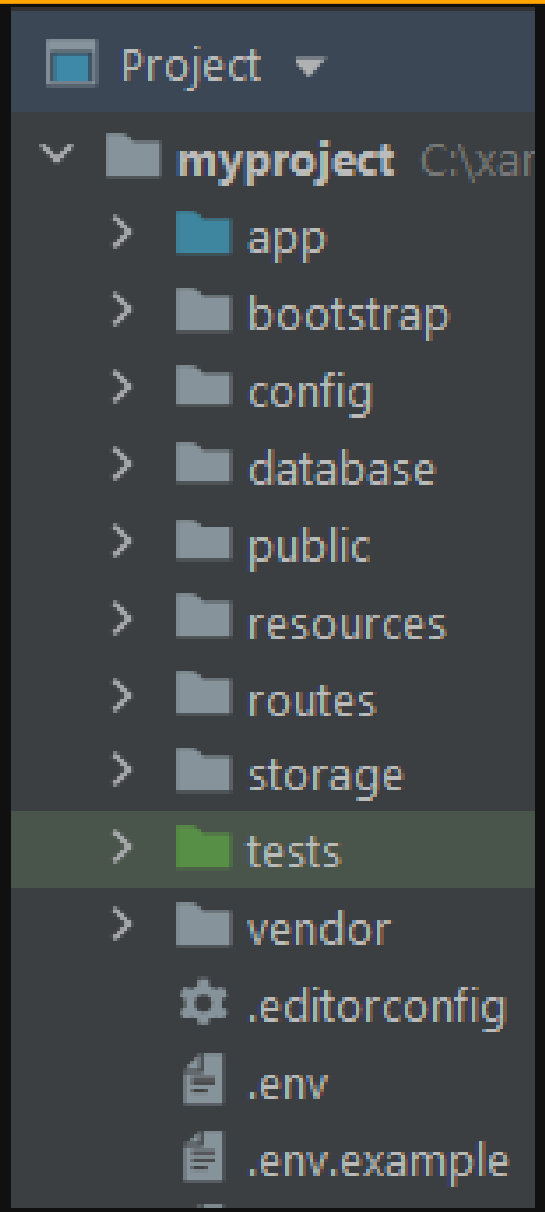
- **4. database/** – obsahuje súbory súvisiace s DB:
  - **migrations/** – definuje štrukturálne zmeny databázy (php artisan migrate)
  - **seeders/** – skripty na naplnenie databázy testovacími dátami (php artisan db:seed)
  - **factories/** – generovanie falošných dát pomocou model factory (php artisan make:factory)

Štruktúra Laravel projektu 4/7



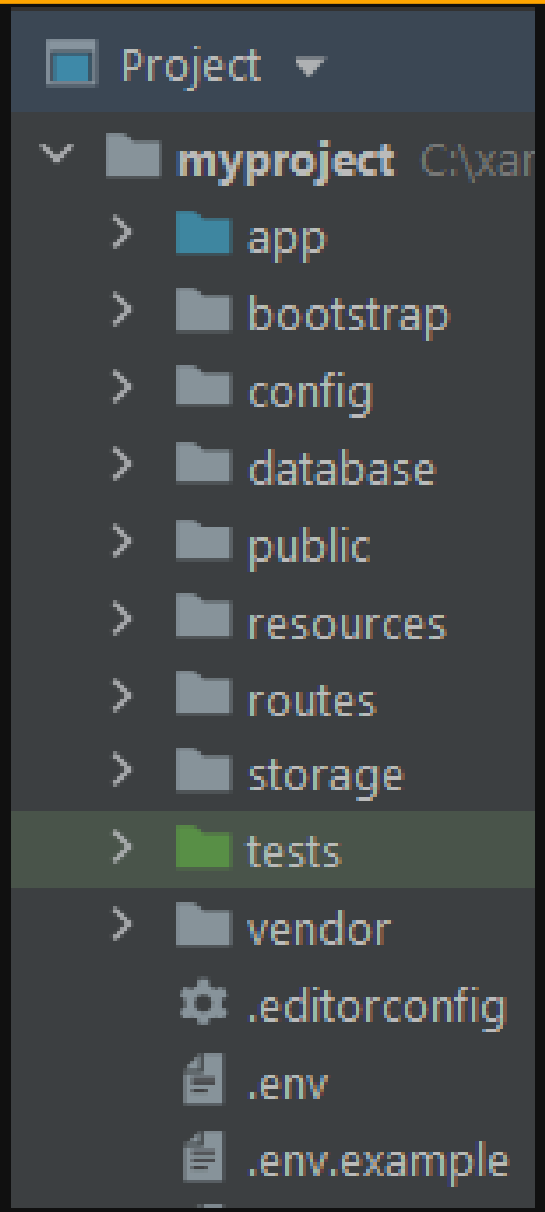
- **5. public/** – koreňový adresár webového servera
  - obsahuje súbor **index.php**, ktorý je vstupným bodom Laravel aplikácie
  - obsahuje **css/, js/, images/** – statické súbory, ak sa nepoužíva frontendový framework
- **6. resources/** – súbory súvisiace s frontendovou časťou Laravel aplikácie:
  - **views/** – Blade šablóny (.blade.php) na vykreslenie HTML
  - **css/** a **js/** – ak Laravel obsahuje aj frontendovú časť
  - **lang/** – Jazykové súbory pre preklady aplikácie

Štruktúra Laravel projektu 5/7



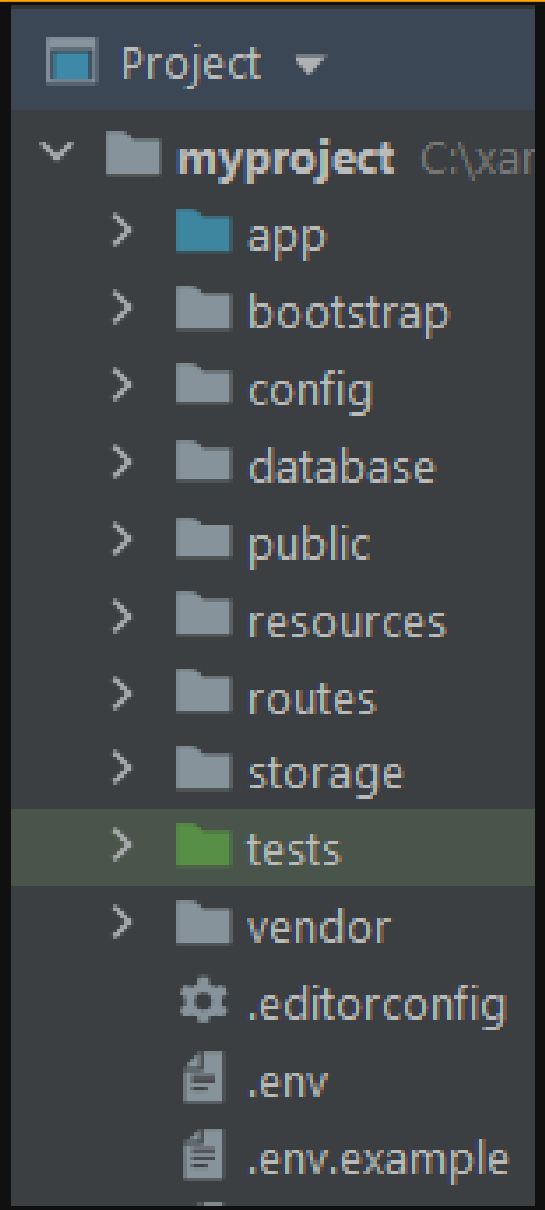
- **7. routes/** – Definícia trás aplikácie
  - Obsahuje súbory na definovanie trás pre aplikáciu:
  - **web.php** – trasy pre webovú aplikáciu (štandardné HTTP požiadavky)
  - **api.php** – trasy pre API (automaticky majú prefix /api)
  - **console.php** – definuje príkazy, ktoré možno spúšťať cez konzolu (php artisan)

Štruktúra Laravel projektu 6/7



- **8. storage/** – súbory generované aplikáciou
  - **app/** – obsahuje súbory nahrané používateľmi
  - **framework/** – Laravel cache, session a view cache súbory
  - **logs/** – logovacie súbory aplikácie (laravel.log)
- **9. tests/** – obsahuje testovacie súbory pre PHPUnit alebo Pest
  - **Feature/** – Testuje funkcie aplikácie (napr. API odpovede)
  - **Unit/** – Testuje individuálne triedy a metódy

Štruktúra Laravel projektu 7/7



- **10. vendor/** – obsahuje Laravel framework a všetky balíčky nainštalované cez Composer (composer.json)
  - **OBSAH vendor/ ZLOŽKY NIKDY NEMODIFIKUJEME!**
- **11. .env** – obsahuje citlivé informácie ako DB pripojenie, API kľúče, mailové nastavenia
  - .env.example – Vzorový .env súbor pre nových vývojárov
  - **zmeny v .env súbore si vyžadujú reštart aplikácie** (php artisan config:clear)

Routing – routes/web.php

- v Laraveli definujeme routes, ktoré určujú, ako aplikácia reaguje na konkrétne URL adresy navštíviteľné cez webový prehliadač

routes/web.php

```php
use App\Http\Controllers\TestController;
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

Route::get('/hello', function () {
    //return view('welcome');
    echo "Hello world!";
});
```

Po načítaní URL / sa zobrazí šablóna **welcome.blade.php**.
Použili sme funkciu view(), ktorá načíta Blade šablónu zo zložky resources/views.

Po načítaní URL **/hello** sa do prehliadača vypíše text "Hello world!". Namiesto zobrazovania šablóny sme použili echo na výpis textu priamo do prehliadača

Routing – routes/web.php

- v Laraveli definujeme routes, ktoré určujú, ako aplikácia reaguje na konkrétne URL adresy navštíviteľné cez webový prehliadač

routes/web.php

```php
use App\Http\Controllers\TestController;
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

Route::get('/hello', function () {
    //return view('welcome');
    echo "Hello world!";
});
```

Takto definované cesty sa nazývajú **closure routes**. Ide o routy, ktoré **neodkazujú** na externé metódy, ale obsahujú celú logiku priamo vo funkcii pri ich definícii.

Majú dva argumenty:
- URI – určuje cestu, na ktorú sa má aplikácia smerovať.
- Closure – definuje, čo sa má stať po príchode na danú cestu.

Úloha: Prvý Controller

- oveľa častejšie je však definovanie routes formou odkazu na metódu v Controlleri
- vytvorte controller **TestController**, ktorý bude obsahovať nasledujúcu metódu a spracovávať HTTP GET požiadavku:
  - **GET /test** → zavolá metódu testAction(), ktorá vypíše text "Funguje to?„

Prvá metóda v Controlleri

- vytvorenie Controllera:
  - **php artisan make:controller TestController**

app/Http/Controllers/TestController.php

```php
<?php

namespace App\Http\Controllers;

class TestController extends Controller{

    public function testAction(){
        return "Funguje to?";
    }

}
```

Definujeme nový route v routes.php.

```
Route::get('/test', [TestController::class, 'testAction']);
```

URL

Controller

metóda

Zdroje

- https://laravel.com/docs/11.x
- https://laravel-news.com/
- https://youtu.be/eUNWzJUvkCA