WIKI

Vydanie 0.1

Lacike

11. apr 2022

Webové technológie

Toto je technologická dokumentácia /vlastná wiki/ k vývoju webových aplikácii za použitia **FRONT END** technológii a **BACK END** jazyka PHP ako aj ostatné technologocké vychytávky.

Konkrétne technológie sú popísané vlastnými skúsenosťami z vývoja ako aj ucelená postupnosť krokov vývoja webovej aplikácie.

Sumár praktickej dokumentácie a packegov 3. strán nájdeš *TU <https://github.com/chiraggude/awesome-laravel>* a navyse velky zoznam open source projektov pre inspiraciu najdes na *Laravel-Open-Source-Projects <https://world.openfoodfacts.org/>*.

KAPITOLA 1

Bulma css

1.1 Bulma Responsive Tables

Responsive tables for Bulma CSS Framework — Pure HTML & CSS/SCSS.

> Free under MIT License Pure HTML & CSS/SCSS Built for Bulma CSS Framework No js framework dependencies Ready-to-use CSS SCSS sources with variables

Dokumentacia : https://github.com/justboil/bulma-responsive-tables

1.1.1 Instalacia

```
npm i bulma-responsive-tables --save
```

Pre pouzitie SCSS naiportuj do tvojho subora

```
/* Bulma Responsive Tables */
@import "node_modules/bulma-responsive-tables/bulma-responsive-tables";

/* Bulma */
@import "node_modules/bulma/bulma";
```

Pekne free Admin rozhranie: https://justboil.me/

Zaroven pekny package: https://buefy.org/

Lightweight UI components for Vue.js based on Bulma

KAPITOLA 2

Laravel

Laravel je framework pre vývoj web aplikácii s kladeným dôrazom na elegantnú syntax a dokumentáciu, ktorá je dostupná na Laravel dokumentácia.

Pre sťahovanie dependencies slúži Composer a všetky dostupné balíčky resp. repozitár balíčkov je dostupný na Packagist alebo GitHub.

Laravel má aj vlastný adresár balíčkov Packalyst určený pre **Laravel** projekty a teší sa aj veľkému počtu fanúšikov vlastných videotutoriálov Laracast ako aj Scotch.io tutoriálov.

2.1 Inštalácia Via Composer

```
composer create-project laravel/laravel example-app
cd example-app
php artisan serve
```

2.2 Inštalátor Laravel

Alebo si môžete nainštalovať **Laravel Installer** ako globálnu závislosť **Composer**:

```
composer global require laravel/installer
```

Poznámka: Uistite sa, že ste do vašej `$PATH` umiestnili celý systémový adresár dodávateľa bin Composer, aby váš systém mohol nájsť spustiteľný súbor laravel.

Tento adresár existuje na rôznych miestach v závislosti od vášho operačného systému; niektoré bežné miesta však zahŕňajú:

```
macOS: $HOME/.composer/vendor/bin
```

V mojom pripade zafungovalo pridanie nasledujucej adresy do suboru `~/.zshrc`

```
export PATH="/Users/lacike/.composer/vendor/bin"
```

Poznámka: Nasledne este spustit `source ~/.zshrc`

2.3 Laravel Git projekt

Pre pohodlie môže inštalátor Laravel vytvoriť aj úložisko **Git** pre váš nový projekt. Ak chcete označiť, že chcete vytvoriť úložisko Git, zadaj pri vytváraní nového projektu príznak `--git` :

```
laravel new example-app --git
```

2.4 Laravel GitHub projekt

Namiesto použitia príznaku `--git` môžete tiež použiť príznak `--github` na vytvorenie úložiska Git a tiež na vytvorenie zodpovedajúceho súkromného úložiska na **GitHub**:

```
laravel new example-app --github
```

2.4.1 API

Instalacia

```
php artisan make:controller PhotoController --api
```

V pripade ze vieme aj model tak rovno oznacime aj model :

```
php artisan make:controller API/UserController --api --model=User
```

2.4.2 Authentication

Laravel ponúka vlastný vstavaný **autentifikačný** systém, ktorý je sa jednoducho implementoval do verzie **6.x**.

Laravel zároveň vygeneroval aj základný **layout** aplikácie ako aj **login page** a **registráciu**.

Stacilo pouzit príkaz :

```
php artisan make:auth
```

Poznámka: V pripade, ze chcete registraciu zakazat tak v subore `routes/web.php` pridajte `Auth::routes(['register' => false])`

Od verzie **6.x** az po **7.x** Laravel preveroval vo svojej dokumentacii laravel/ui

Laravel ui

Balík **laravel/ui** spoločnosti Laravel poskytuje rýchly spôsob, ako vytvoriť všetky trasy a zobrazenia, ktoré potrebujete na overenie, pomocou niekoľkých jednoduchých príkazov:

```
composer require laravel/ui --dev

php artisan ui vue --auth
```

Detail pre implemnentaciu **Vue.js** do laravel projektu popisujem v sekcii *Vue.js*.

Od verzie **8.x** az po aktualnu verziu je mozne pouzit 2 implemnentacie tzv. Starter kit .

Laravel Breeze

Laravel Breeze je minimálna, jednoduchá implementácia všetkých overovacích funkcií Laravel, vrátane prihlásenia, registrácie, resetovania hesla, overenia e-mailu a potvrdenia hesla. Predvolená vrstva zobrazenia Laravel Breeze sa skladá z jednoduchých šablón Blade v štýle Tailwind CSS.

Najskor je nutne stiahnut package

```
composer require laravel/breeze --dev
```

Nasledne spustit artisan prikazy:

```
php artisan breeze:install
```

Poznámka: Spusti prikaz `npm install && npm run dev` pre instalaciu **NPM** balickov.

Po instalacii NPM balikov nasleduje este spustit migraciu DB

```
php artisan migrate
```

Reaktivny scafolding

```
php artisan breeze:install vue
```

Alebo ...

```
php artisan breeze:install react
```

A nasledne nainstalovat NPM moduly :

```
npm install
npm run dev
php artisan migrate
```

Laravel Jetstream

Zatiaľ čo <Laravel Breeze>_ poskytuje jednoduchý a minimálny východiskový bod pre vytvorenie aplikácie Laravel, **Jetstream** rozširuje túto funkčnosť o robustnejšie funkcie a ďalšie frontendové technologické balíky.

Pre tých, ktorí sú v Laravel úplne noví, odporúčame naučiť sa s Laravel Breeze pred absolvovaním Laravel Jetstream.

Poznámka: Kompletnú dokumentáciu pre inštaláciu Laravel Jetstream nájdete v ‚oficiálnej dokumentácii Jetstream <https://jetstream.laravel.com/2.x/introduction.html>‘_.

2.4.3 Mail

Je funkcionalita pre odosielanie mailov.

Viac na webe Laravelu Mail.

Konfiguracia

Na odosielanie mailov je potrebne nakonfigurovat v subore `.env` potrebne atributy ktore je mozne nastavit aj na FAKE server.

Korektne nastavenie :

Email bude chodit cez zadany SMT server:

Konfiguracia

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.m1.websupport.sk
MAIL_PORT=465
MAIL_USERNAME=postmaster@lvconsult.sk
MAIL_PASSWORD=Ye6cJx9.n|
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=postmaster@lvconsult.sk
MAIL_FROM_NAME="LV consult s.r.o."
MAIL_ENCRYPTION=tls
```

Fake nastavenie

Email bude chodit namiesto priamo userovi do schranky na **FAKE** server. Pre zobrazenie mailov ako aj konfiguracnych udajov pouzi login na mailtrap.io :

```
user = allacino@gmail.com
password =
```

Je mozne vytvarat projekty a subinboxy.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=465
MAIL_USERNAME=69dc9da0f35af5
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
MAIL_PASSWORD=cfdc030754ebed
MAIL_ENCRYPTION=tls
```

Mailablle class

Pomocnu triedu (Mailablle class) na odosielanie mailov je mozne vytvorit nasledujucimi prikazmy:

```
$ php artisan make:mail <Nazov_triedy>                                    // s␣
↪pouzitim Blade view
$ php artisan make:mail <Nazov_triedy> --markdown=<Cesta_k_suboru>        // s␣
↪pouzitim sablony Markdown
$ php artisan vendor:publish --tag=laravel-mail                           // export␣
↪markdown komponent do vlastnej struktury
```

V kazdej vygenerovanej classe metoda **build** vytvara mail a je mozne pouzit parametre ako :

```
-> from
-> subject
-> view
-> attach
```

```
return $this->from('Meno_odosielatela','example@example.com')
                            ->view('emails.orders.shipped');
```

V pripade ze nechceme **MAIL** formatovat tak posleme PLAIN text takto

```
->text('emails.orders.shipped_plain');
```

V pripade ze chceme do **MAIL** sablony posielat aj data , tak mame k dispozicii 2 moznosti :

1. Via **Public Properties**

V konstruktore vytvorenej mailablle triedy musime zadefinovat model z ktoreho budeme citat data:

```
public $order;
public function __construct(Order $order)
{
    $this->order = $order;
}
```

2. Via The **with** Method

V takomto pripade mozeme do VIEW poslat aj pole s datami „**with**“

```
public function build()
{
    return $this->view('emails.orders.shipped')
                ->with([
                    'orderName' => $this->order->name,
                    'orderPrice' => $this->order->price,
                ]);
}
```

Odosielanie priloh

```
public function build()
{
    return $this->view('emails.orders.shipped')
                ->attach('/path/to/file');
}
```

alebo

```
public function build()
{
    return $this->view('emails.orders.shipped')
                ->attach('/path/to/file', [
                    'as' => 'name.pdf',
                    'mime' => 'application/pdf',
                ]);
}
```

Poznámka: Pri tomto pouziti je potrebne mat na pamati ze sa subor musi fyzicky nachadzat.

Odosielanie prilohy PDF

Vygenerovane PDF data vieme poslat v prilohe mailu aj bez potreby ulozenia na disk.

Vtedy pouzijem nasledujucu build metodu :

```
public function build()
{
        return $this->view('emails.orders.shipped')
                                ->attachData($this->pdf, 'name.pdf', [
                                        'mime' => 'application/pdf',
                                ]);
}
```

Poznámka: PDF musi prist cez metodu `$pdf->output()`

Priklad:

```
$pdf = PDF::loadView('pdf.confirmTB', $data);
    $pdf->save('invoice.pdf');

    return $pdf->output();
```

Odoslanie obrazku

```
<body>
 Here is an image:

 <img src="{{ $message->embed($pathToFile) }}">
</body>
```

Markdown mail

Zakladom je mat blade s pouzitim Markdown komponentami. Prikaz na vygenerovanie VIEW :

```
$ php artisan make:mail <Nazov_triedy> --markdown=<Cesta_k_suboru>          // s␣
↪pouzitim sablony Markdown
```

napr.

```
$ php artisan make:mail OrderShipped --markdown=emails.orders.shipped
```

V pripade pouzitia Markdown sablony v metode **build** pouzijeme metodu **markdown** :

```
return $this->from('example@example.com')
            ->markdown('emails.orders.shipped');
```

Kustomizacia Markdown komponentov

V prvom rade musime mat vyexportovane MARKDOWN komponenty do vlastnej struktury :

```
$ php artisan vendor:publish --tag=laravel-mail
```

Po vygenerovani sa komponenty nachadzaju v `resources/views/vendor/mail`

2.4.4 Kustomizacia CSS Markdown komponentov

Vygenerovane komponenty obsahuju defaultny css subor `default.css` pre kazdu temu `resources/views/vendor/mail/html/themes` ktoreho upravou sa zmeny prejavia automaticky.

V pripade ze si chceme vytvorit vlastnu themu, tak ju vytvorime tu `resources/views/vendor/mail/html/themes` ale nesmieme zabudnut na nastavenie temy v configu `config\mail`

Odoslanie mailu

```
Mail::to($request->user())->send(new OrderShipped($order));
```

alebo

```
Mail::to($request->user())
        ->cc($moreUsers)
        ->bcc($evenMoreUsers)
        ->send(new OrderShipped($order));
```

MAIL je mozne odoslat priamo do prehliadaca :

```
Route::get('/mailable', function () {
$invoice = App\Invoice::find(1);

        return new App\Mail\InvoicePaid($invoice);
});
```

Dalsou moznostou je vyrenderovanie Mailu. Metoda **render** vráti vyhodnotený obsah Mailu ako reťazec

```
$invoice = App\Invoice::find(1);

return (new App\Mail\InvoicePaid($invoice))->render();
```

Lokalizovanie jazyka mailu

```
    Mail::to($request->user())->send(
(new OrderShipped($order))->locale('es')
    );
```

Queueing Mail

Keďže odosielanie e-mailových správ môže drasticky predĺžiť čas odozvy vašej aplikácie, mnohí vývojári sa rozhodnú do frontu odosielať e-mailové správy. Laravel to uľahčuje pomocou zabudovaného rozhrania API pre jednotnú frontu. Ak chcete na fronte e-mailovú správu, použite metódu frontu na priečke pošty po zadaní príjemcov správy:

```
Mail::to($request->user())
             ->cc($moreUsers)
             ->bcc($evenMoreUsers)
             ->queue(new OrderShipped($order));
```

Táto metóda sa automaticky postará o stlačenie úlohy na frontu, aby sa správa odoslala na pozadí. Samozrejme, pred použitím tejto funkcie budete musieť nakonfigurovať svoje fronty *Queues <https://laravel.com/docs/9.x/queues>*.

2.4.5 Mail verifikacia

Laraval ma v sebe uz zabudovanu funkcionalitu overovania mailov..

Viac na webe Laravelu - Mail verification

Konfiguracia

Na odosielanie mailov je potrebne nakonfigurovat v subore `.env` potrebne atributy ktore je mozne nastavit aj na FAKE server.

Verify Email

** Postup ** po vygenerovani Auth Scaffolding `php artisan make:auth`

Implementovat rozhranie **MustVerifyEmail** do modelu `User`

```
class User extends Authenticatable implements MustVerifyEmail
{
    use Notifiable;

    // ...
}
```

Do potrebneho modelu pridaj IF `use Illuminate\Contracts\Auth\MustVerifyEmail;`

Pridaj verifikacnu Route do `web.php`

Do suboru `routes >> web.php` pridaj extra parameter

```
Auth::routes(['verify' => true]);
```

Toto enabluje controller s názvom `VerificationController.php`, ktorý je už dodávaný s Laravelom.

3.Pre ostrenie vstupu neverifikovanym pridaj do konstruktoru daneho Controllera nazov middleware **verified**

```
public function __construct()
{
    $this->middleware(['auth', 'verified']);
}
```

alebo /nove riesenie Laravel 7.*/

```
Route::get('profile', function () {
 // Only verified users may enter...

})->middleware('verified');
```

2.4.6 PDF Wrapper

Pre tvorbu **PDF** je mozne vyuzitie viacero PDF modulov. Asi medzi najoblubenejsie package patri DOMPDF

```
composer require barryvdh/laravel-dompdf
```

Po instalacii bolo v minulosti pridat do suboru `config/app.php`

```
Barryvdh\DomPDF\ServiceProvider::class,

'PDF' => Barryvdh\DomPDF\Facade::class,
```

Poznámka: V aktualnej verzii laravelu a instalacii DomPDF nie je nutne modifikovat subor `config/app.php`.

Nasledne je PDF wraper pouzit takto:

```
$pdf = App::make('dompdf.wrapper');
$pdf->loadHTML('<h1>Test</h1>');
return $pdf->stream();
```

Alebo je mozne pouzit fasadu :

```
$pdf = PDF::loadView('pdf.invoice', $data);
return $pdf->download('invoice.pdf');
```

Nastavenia mozeme urobit v konfiguracnom subore `config/dompdf.php` ak pouzijeme prikaz

```
php artisan vendor:publish --provider="Barryvdh\DomPDF\ServiceProvider"
```

Varovanie: Pri pouziti attachment v maili je potrebne do build metody dannej mail triedy pouzit `attachData` pre vlozene surovych dat, ale data musia prist cez `output()`.

2.4.7 Sass

Metóda sass vám umožňuje zostaviť **Sass** do CSS, ktoré sú zrozumiteľné pre webové prehliadače.

Najskor je potrebne nainstalova sass loader :

```
npm install resolve-url-loader@^5.0.0 --save-dev --legacy-peer-deps
```

Metóda sass akceptuje cestu k vášmu súboru Sass ako svoj prvý argument a adresár, do ktorého by sa mal umiestniť kompilovaný súbor, ako svoj druhý argument :

```
mix.sass('resources/sass/app.scss', 'public/css');
```

Vlozenie normalize

```
npm install --save normalize.css
```

2.4.8 Tailwind css

Instalacia TailwindCSS

TailwindCSS sa stal veľmi populárnym nielen u vývojárov **Laravel**, ale aj vývojárov **Vue**.

Tailwind vám umožňuje vytvárať aplikácie a webové stránky bez toho, aby ste museli opustiť svoj html.

Najprv musíme nainštalovať potrebné závislosti:

```
npm install -D tailwindcss postcss autoprefixer
```

Konfiguracia TailwindCSS

Teraz môžeme vytvoriť konfiguračný súbor tailwind, ktorý môžeme použiť na rozšírenie predvolených nastavení **TailwindCSS** :

```
npx tailwindcss init
```

Tým sa v našom projekte vytvorí nový súbor `tailwind.config.js`.

Mix configuration

Potom musíme nakonfigurovať **webpack** na spracovanie inštalácie tailwind:

```
const mix = require("laravel-mix");

mix.js("resources/js/app.js", "public/js")
        .vue()
        .postCss("resources/css/app.css", "public/css", [require("tailwindcss")]);
```

Configure your template paths

Add the paths to all of your template files in your `tailwind.config.js` file :

```
module.exports = {
  content: [
        "./resources/**/*.blade.php",
        "./resources/**/*.js",
        "./resources/**/*.vue",
  ],
  theme: {
        extend: {},
  },
  plugins: [],
}
```

Tailwind directives

Aby ste mohli vo svojom projekte použiť pomocné triedy tailwind, môžeme ich jednoducho pridať do nášho súboru `resources/css/app.css` :

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Tieto direktívy tailwind sú preberané našimi procesmi zostavovania webpacku a triedy obslužných programov tailwind sú umiestnené do výsledného vytvoreného súboru css.

Nakoniec môžeme pridať šablónu so štýlmi na stránku `welcome.blade.php`, aby sme mohli použiť štýly v našom projekte:

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
        <head>
                <meta charset="utf-8">
                <meta name="viewport" content="width=device-width, initial-scale=1">

                <title>Laravel</title>

                <!-- Fonts -->
                <link href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;
↪700&display=swap" rel="stylesheet">
                <link rel="stylesheet" href="{{ asset('css/app.css') }}">
        </head>
        <body>
                <div id="app"></div>
        </body>
        <script src="{{ asset('js/app.js') }}"></script>
</html>
```

Možno budete musieť znova reštartovať vývojový server a/alebo reštartovať proces buildovania.

Teraz môžeme do nášho komponentu pridať niekoľko pomocných tried, aby sme otestovali, že všetko funguje:

```
<template>
        <div class="flex justify-center mt-24">
                <h1 class="text-2xl font-bold text-gray-700">Vue 3 App</h1>
        </div>
</template>
```

2.4.9 Trix editor

Trix je open-source projekt od Basecampu, tvorcov Ruby on Rails. Milióny ľudí dôverujú svoj text Basecamp a my sme vytvorili Trix, aby sme im poskytli čo najlepší zážitok z úpravy.

```
npm install trix
```

Alternativy:

- vue-trix
- vue-trix-editor

Nasledne si trix editor naimportujeme do komponenty kde chceme pouzivat editor :

```
<script>
import trix from 'trix'

export default {
        components: {
                trix,
        },
}
```

Zaroven je potrebne zadat v komponente aj CSS :

```
<style lang="scss" scoped>
        @import '~trix/dist/trix.css';
</style>
```

Nasledne pouzijeme v kode :

```
<form>
        <input id="x" value="Editor content goes here" type="hidden" name="content">
        <trix-editor input="x"></trix-editor>
</form>
```

2.4.10 Vue.js

Vue.js (nebo jen Vue; vyslovuje se stejně jako view) je open-source progresivní JavaScriptový framework pro vytváření uživatelských rozhraní.Začlenění do projektů, které používají jiné JavaScriptové knihovny je s Vue snadné, protože je navržen tak, aby mohl být přijímán postupně. Vue může také fungovat jako webový aplikační framework, na kterém je možné vytvářet pokročilé Single-page applications. Zakladnu dokumentaciu najdes [TU](https://vuejs.org/).

Stavia na štandarde HTML, CSS a JavaScript a poskytuje deklaratívny a komponentný programovací model, ktorý vám pomáha efektívne rozvíjať používateľské rozhrania, či už jednoduché alebo zložité.

VUE 2

Implementacia

Pre implementaciu cisteho **Vue 2** bez dalsich medzi frameworkov je optimalne vykonat starsim sposobom a to nasledovnym balickom :

```
composer require laravel/ui
```

> **Varovanie:** POZOR!!! Plati pre verziu 6.x a vyssiu pre VUE 2 .

Nasledne je nutne si vybrat reaktivny framework a v nasom pripade to je **Vue.js**

```
// Generate basic scaffolding...

php artisan ui bootstrap
php artisan ui vue
php artisan ui react
```

Po nainstalovaniu VUE dependencies je mozne vygenerovat login registration :

```
// Generate login / registration scaffolding...

php artisan ui bootstrap --auth
php artisan ui vue --auth
php artisan ui react --auth
```

Po uspesnej instalacii staci spustit :

```
npm install && npm run dev
```

Layout pouzitie

```
@extends('layouts.app')

@section('content')
        <example-component></example-component>
@endsection
```

Varovanie:

Nasledujuci js prihod do stranky :

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

Struktura projektu

Zakladnym suborom v Laravel aplikacii je `resources\js\app.js` kde registrujeme VUE komponenty ako aj identifikujem zakladnu ROOT kostru vue priestoru.

Je nutne zadeklarovat hlavny HTML pre VUE instanciu nasledovnym sposobom :

```
<div id="vue-app">
        Hello Vue !
</div>
```

K danemu kodu je nutne vytvorit uz zmienenu VUE instanciu :

```
<script type="text/javascript">
        var app = new Vue({
                el: '#vue-app'
        });
</script>
```

Komponenty

```
<?js

Vue.component('example-component', require('./components/ExampleComponent.vue'));

/**
 * Aplikacia musi mat vo VIEW html element s ID <div id="app">
 */

<div id="app">
  Hello {{ name }}.
</div>
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
<script>
const app = new Vue({
        el: '#app',
        data: {
                name: 'Vašo'
        }
});
</script>
```

VUE 3

Instalacia

Ak používate Laravel od **6.x** vyššie, možno ste natrafili na balík **Laravel/ui**, pomocou ktorého by sme mohli nainštalovať Bootstrap, ako aj React alebo Vue.

Pre aktualnu verziu Laravel **9.x** neexistuje žiadny balík pre instalaciu **Vue 3**, ale existuje pomerne jednoduchý spôsob, ako to urobiť.

Najprv nainštalujte závislosti potrebné pre Vue 3:

```
npm install --save vue@next && npm install --save-dev vue-loader@next
```

Konfigurcia

Po instalacii musíme povedať **webpacku**, aby skompiloval aj naše súbory vue.

Môžeme to urobiť otvorením súboru `webpack.mix.js` a pridaním nasledujúceho riadku

```
const mix = require("laravel-mix");

mix.js("resources/js/app.js", "public/js")
        .vue() //new
        .postCss("resources/css/app.css", "public/css", [
                //
]);
```

Nasledne spustím inštaláciu npm :, po ktorej nasleduje npm run dev.

```
npm install && npm run dev.
```

Po vykonani vsetkych predchadzajucich krokoch mam laravel projekt pripraveny s VUE 3 frameworkom.

Projekt

Vytvorime koreňovu komponentu, ktora bude obsahovať celú našu aplikáciu **Vue**.

Vytvorte nový súbor `App.vue` v priečinku `resources/js` s nasledujúcim označením:

```
<template>
        <div>
                <h1>Vue 3 App</h1>
        </div>
</template>
```

Teraz musíme prispôsobiť našu `app.js` v `resources/js`, aby sme mohli využívať náš súbor vue:

```
require("./bootstrap");

import { createApp } from "vue";

import App from "./App.vue";

createApp(App).mount("#app");
```

Najprv importujeme metódu **createApp()**, ktorá je pre vývojárov Vue nová. Pomocou toho môžeme vytvoriť novú inštanciu Vue.

Potom importujeme náš súbor Vue a vytvoríme novú inštanciu Vue a pripojíme ju k prvku s ID „app“.

Taktiez ale mozeme konstantnu premennu ktoru podobne pripojime k prvku s ID ako v predchadzajucom pripade

```
import { createApp } from "vue";

import Home from "./components/Home.vue";
import Dashboard from "./components/Dashboard.vue";

const app = createApp({
  components: {
        Home,
        Dashboard
  }
});

app.mount("#app");
```

Teraz vytvorte prvok, ktorý má toto id. Aby sme to dosiahli, môžeme odstrániť štandardné označenie nachádzajúce sa v našom súbore `welcome.blade.php` a nahradiť ho týmto:

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
        <head>
                <meta charset="utf-8">
                <meta name="viewport" content="width=device-width, initial-scale=1">

                <title>Laravel</title>

                <!-- Fonts -->
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
                <link href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;
↪700&display=swap" rel="stylesheet">
        </head>
        <body>
                <div id="app"></div>
        </body>
        <script src="{{ asset('js/app.js') }}"></script>
</html>
```

Ako môžete vidieť, máme **div** s id *app*, ktorá bude nahradená našou aplikáciou Vue.

Nižšie máme značku skriptu, ktorá odkazuje **na app.js**, ktorú nemáme v `resources/js`, ale v našom priečinku `public/js`. Tento súbor je konečným výsledkom procesu vytvárania nášho webového balíka.

Predtým, ako to urobíme, musíme znova spustiť proces zostavovania, pretože sme v skutočnosti nespracovali súbory Vue, ktoré sme vytvorili.

```
npm run dev
```

Poznámka: Aby sme nemuseli spúšťať `npm run dev` zakaždým, keď vykonáme zmenu, môžeme spustiť `npm run watch`, ktorý nám umožní spustiť proces zostavovania hneď po uložení zmien.

Import CSS from

V pripade ze mame hotovy FE, tak nema zmysel buildovat stale nove CSS a hotovy dizajn si vieme v hlavnom js subore `app.js` importovat

```
import './asset/css/mystyle.css'
```

Vue router

```
npm install vue-router@4
```

Po instalacii samotneho **Vue Router** je potrebne nakonfigurovat zakladnu kostru obdobne ako pri zavedeni **VUE** a to v subore `app.js` alebo `main.js` pripadne podla vlastneho uvazenia.

```
require("./bootstrap");

import { createApp } from "vue";
import Home from "./components/Home.vue";
import Dashboard from "./components/Dashboard.vue";

const app = createApp({
  components: {
        Home,
        Dashboard
  }
});

app.mount("#app");
```

Pri tvorbe SPA aplikacie, je vhodne pri pouziti vsetky <a> elemetoch pouzit <router-link>. Vtedy pri preklikoch sa nerefreshuju stranky.

2.4.11 QR kod

Instalacia

Pre generovanie QR kodov je vhodny jednoduchy QRcode /simple-qrcode/ package

```
composer require simplesoftwareio/simple-qrcode
```

Po instalcia ako byva zvykom je potrebne do konfig suboru `config/app.php` pridat service provider and aliases.

```
//config/app.php

'providers' => [
 ....
 SimpleSoftwareIO\QrCode\QrCodeServiceProvider::class
 ],

'aliases' => [
 ....
 'QrCode' => SimpleSoftwareIO\QrCode\Facades\QrCode::class
 ],
```

Pouzitie

Po uspesnych krokoch vieme uz pouzivat QR code :

```
Route::get('qrcode', function () {
         return QrCode::size(300)->generate('A basic example of QR code!');
 });
```

QR Code with Color

```
Route::get('qrcode-with-color', function () {
         return \QrCode::size(300)
                                        ->backgroundColor(255,55,0)
                                        ->generate('A simple example of QR code');
 });
```

QR Code with Image

```
Route::get('qrcode-with-image', function () {
                $image = \QrCode::format('png')
                                                ->merge('images/laravel.png', 0.5, true)
                                                ->size(500)->errorCorrection('H')
                                                ->generate('A simple example of QR code!
↪');
          return response($image)->header('Content-type','image/png');
 });
```

Email QR code

```
Route::get('qrcode-with-special-data', function() {
         return \QrCode::size(500)
                                    ->email('info@tutsmake.com', 'Welcome to Tutsmake!',
↪'This is !.');
 });
```

QR Code Phone Number

```
QrCode::phoneNumber('111-222-6666');
```

QR Code Text Message

```
QrCode::SMS('111-222-6666', 'Body of the message');
```

KAPITOLA 3

Python

Instalcia …

Tu doplnim pokiny k instalacii …

Podrobnejsie sa viac vsak budeme venovat vyvoju web aplikacii a to 1 z tychto 2 frameworkov :

- Django
- Flask

3.1 Django

Django je pomerne komplexný webový framework, ktorý ponúka všetko, čo budú naši užívatelia potrebovať.

Django za nás prevezme správu databázy, poskytne veľa užitočných funkcií a ponúkne svoju perfektnú administráciu.

3.1.1 Inštalácia

Počítame s tým, že Python máte nainštalovaný. Django nainštalujeme nasledujúcim príkazom, ktorý vložíme do príkazového riadku. Ak nie ste s príkazovým riadkom zoznámenie, otvorte ponuku Štart a napíšte „cmd“. Na nájdenú aplikáciu „Príkazový riadok“ vpravo kliknite myšou. Ak nepoužívate Windows, predpokladám, že viete kde má váš systém terminál :) Zadajte príkaz:

..code-block:: console

    py -m pip install Django==2.0.4

Ak používate Anaconda / MiniConda distribúciu Pythone, upravte si príkazy pre inštalácie ako:

..code-block:: console

    conda install django

Aktuálne číslo verzie, nájdete na https://www.djangoproject.com/

3.1.2 Adresárová štruktúra Django projektu

Obsah zložky vyzerá nasledovne:

```
mysite/
        mysite/
                __init__.py
                settings.py
                urls.py
                wsgi.py
        manage.py
```

Jednotlivé súbory si teraz popíšeme:

- **settings.py** - Tu sa nachádza konfigurácia projektu a tu tiež inštalujeme svoje aplikácie, viď ďalej.
- **urls.py** - Tu je uložené schéma, ktoré hovorí ako sa má spracovať URL adresa, ktorú užívateľ do prehliadača zadá. Definujeme tu aké adresy smerujú na aké časti našej aplikácie. Tomuto mechanizmu sa hovorí routovanie.
- **wsgi.py** - Konfigurácia rozhrania pre nasadenie webové aplikácie na server.
- **manage.py** - O zložku vyššie sa nachádza manage.py, pre nás najdôležitejšia súbor. Umožňuje nám spúšťať rôzne príkazy. Použijeme ho napr. K migrácii databázy, vytváranie aplikácií, zberu statických súborov a ďalším podobným účely.

3.1.3 Spustenie servera

Spustenie servera prevedieme spustením príkazu cez súbor manage.py. K tomu potrebujeme otvoriť príkazový riadok v priečinku mysite, kde sa tento súbor nachádza.

Zadáme príkaz:

```
py manage.py runserver
```

3.1.4 Vytvorenie aplikácie (modulu)

Projekt môže obsahovať viac aplikácií. Tie fungujú ako moduly, typicky máme napr. Zvlášť administráciu od hlavnej aplikácie. Asi ste sa asi skôr stretli s pojmami aplikácie a modul než s projekt a aplikácie, význam je ale úplne rovnaký.

Vytvoríme si teda v projekte aplikácii (modul) s názvom „ahoj_svete“ príkazom:

```
py manage.py startapp ahoj_svete
```

3.1.5 Adresárová štruktúra aplikácie (modulu)

Django za nás všetko potrebné vytvorí, to je jeho výhoda oproti napr. Flask frameworku. V priečinku projektu sa nám vytvorí nový adresár, jeho obsah vyzerá nasledovne:

```
mysite/
        ahoj_svete/
                __init__.py
                admin.py
                apps.py
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
            models.py
            tests.py
            # urls.py
    # Ostatní zůstává stejné
```

Jednotlivé súbory:

- admin.py - Administrácia jednotlivé aplikácie.
- apps.py - Konfigurácia jednotlivej aplikácie.
- models.py - Správa modelov databázy, Django ORM, viď ďalej v kurze.
- tests.py - Testy aplikácie.
- urls.py - Tento súbor sa nevytvorí, musíme ho vytvoriť manuálne. Budú tu odkazy na Python skripty, ktoré požiadavky zaslané cez danej URL adresy spracujú

3.1.6 Inštalácia aplikácie (modulu)

V súbore `NAZOV_PROJEKTU/NAZOV_PROJEKTU/settings.py` aplikáciu „nainštalujeme“. Django musí mať prehľad o tom, kde má svoje aplikácie a koľko ich má. U tejto aplikácie inštalácia nie je nutná, ale je lepšie si zvyknúť inštalovať všetky naše aplikácie, už len z princípu Django frameworku.

Inštaláciu vykonáme pripísaním názvu aplikácie do `settings.py`, do časti označenej komentárom # Application definition :

```
INSTALLED_APPS = [
        'helloWorld',
        ...
        ...
]
```

3.1.7 Routovanie

Dostávame sa do bodu, kedy musíme zabezpečiť, aby otvorenie URL adresy servera otvorilo aplikáciu `helloWorld`. Keďže postup je zložitejší, vysvetlíme si hneď na začiatku čoho potrebujeme docieliť.

Adresu servera (http://localhost:8000/) postupne napojíme zo súboru `urls.py` projektu, cez súbor `urls.py` aplikácie (modulu), až na našu prvú Python metódu, ktorá užívateľovi pošle „Hello world!“ hlášku späť do prehliadača.

Routovanie projektu (modulu)

V súbore `NAZOV_PROJEKTU/NAZOV_PROJEKTU/settings.py` pridáme route na našej aplikácii. Keďže táto ruta bude definovaná v routách aplikácie, teda v súbore `NAZOV_PROJEKTU/helloWorld/urls.py`, tento súbor sem iba naincludujeme, aby sme ju nemuseli písať 2x. Všetky ruty definované v aplikácii sa potom samy prenesú do rout projektu a budú fungovať.

Keďže budeme chcieť, aby sa aplikácia `helloWorld` spustila priamo po otvorení adresy servera `localhost:8000/`, namiesto adresy uvedieme len prázdny reťazec ‚‘. Do súboru `urls.py` si pridajte route na helloWorld.urls. Nezabudnite pridať aj import pre **include**:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
        path('admin/', admin.site.urls),
        path('', include("ahoj_svete.urls")),
]
```

Routovanie aplikácie

Všetky routy budeme už definovať len v priečinku aplikácie. Prejdite do priečinka `mysite/ahoj_svete/` a tu vytvorte súbor `urls.py`. Tam pridáme URL adresu našej prvej stránky. Tu už adresu prekryjeme cez tzv. **View**. To je Python metóda, ktorá požiadavku spracuje a vráti užívateľovi výsledok.

Poznámka: Ak poznáte **MVC** architektúru, aj tu Python používa iné názvoslovie.

Obsah súboru tu `urls.py` bude nasledujúce:

```
from django.urls import path
from . import views

urlpatterns = [
        path("", views.index, name="index"),
]
```

Pridanie view

View je teda posledná časť reťazca, ktorá požiadavku užívateľa zachytí a odpovie na neho. Jedná sa o Python metódu. Rôzne URL adresy napojujeme na rôzne views, teda rôzne metódy. My máme v aplikácii teraz len jednu, aby toho nebolo na začiatok moc.

Vytvoríme si teda view pre zobrazenie hlášky „Ahojte všetci!“.

Prejdeme do súboru `mysite/ahoj_svete/views.py`, kde si vytvoríme metódu index(). Jej parametrom bude request, požiadavka, ktorý prišiel cez URL adresu. Našu hlášku nezobrazíte ako HTML stránku, ale len ako text vrátením inštancia typu HttpResponse. Kód danej metódy je nasledovné, nezabudnite si pridať aj **HttpResponse** na prvom riadku.

```
from django.shortcuts import render, HttpResponse

def index(request):
        return HttpResponse("Ahoj světe!")
```

Všetko si uložte a otvorte adresu aplikácie v prehliadači. Prejdite na http://localhost:8000/.

Varovanie: Nezabudnite, že musíte mať spustený aj server, viď vyššie.

3.1.8 MVC

MVC je veľmi obľúbený architektonický vzor, ktorý sa uchytil najmä na webe, hoci pôvodne vznikol na desktopoch. Je súčasťou populárnych webových frameworkov, akými sú napr. **Zend** alebo Nette pre PHP, Ruby On Rail pre Ruby alebo MVC pre ASP .NET. Osobne si bez neho (alebo nejakého podobného princípu) nedokážem predstaviť zložitejšie web.

Motivácia

Základnou myšlienkou **MVC** architektúry je oddelenie logiky od výstupu. Rieši teda problém tzv. „Špagetového kódu", kedy máme v jednom súbore (triede) logické operácie a zároveň renderovanie výstupu. Súbor teda obsahuje databázové dotazy, logiku (volanie Python príkazov) a rôzne pohádzané HTML tagy. Všetko je zamotané do seba ako špagety.

Kód sa samozrejme zle udržuje, nieto rozširuje. Je zle highlightovaný, pretože si s ním IDE nevie rady, HTML nie je správne naformátované, strácame sa v jeho stromovej štruktúre. Naším cieľom je, aby zdrojový kód s logikou vyzeral ako zdrojový kód (Python) a výstup vyzeral ako HTML stránka s čo najmenšou prímesou ďalšieho kódu.

Komponenty

Celá aplikácia je rozdelená na komponenty 3 typov, hovoríme o modeli, View (pohľadoch) a Controller (kontroléry), od toho MVC. Označenie pohľad sa budem snažiť vyhýbať, pretože mi príde mätúce, že takto preloženej nekorešponduje s označením V. Neexistuje žiadna striktná definícia architektúry a tak sa môžete stretnúť s viacerými výklady. Zameral som sa na ten najrozšírenejší.

Komponenty Model a Controller sú samozrejme triedy. View je HTML šablóna.

Model

Model obsahuje logiku a všetko, čo do nej spadá. Môžu to byť výpočty, databázové dotazy, validácie a podobne. Model vôbec nevie o výstupe. Jeho funkcia spočíva v prijatí parametrov zvonku a vydanie dát von. Zdôrazním, že parametre nemyslím URL adresu ani žiadne iné parametre od užívateľa. Model nevie, odkiaľ dáta v parametroch prišli a ani ako budú výstupné dáta naformátovaná a vypísane.

Keďže budeme používať tzv. ORM (Objektovo-Relačná Mapovanie), naše modely budú priamo korešpondovať s databázovými tabuľkami. Budeme teda mať napr. Model Uzivatel, Komentar alebo Clanok. Instance modelov budú samozrejme obsahovať atribúty z databázy. Instance modelu Uzivatel bude mať napr. Atribút meno. Triede môžeme definovať inštančný metódy, napr. Takú, ktorá vypočíta vek používateľa podľa jeho dátumu narodenia. Metódy týkajúce sa všeobecne užívateľov (teda triedny) často vkladáme do modelu ako statické, napr. Overenie správnej dĺžky a znakov hesla (teda jeho validáciu, pretože heslo overujeme ešte predtým, než je inštancia používateľa vytvorená a zároveň s užívateľom logicky súvisí).

View

> **Varovanie:** V Django v MVC architektúre má VIEW iný význam. V tejto sekcii budeme teda hovoriť o všeobecnej komponente, nie o Django.

Pohľad (View) sa stará o zobrazenie výstupu užívateľovi. Jedná sa o html šablónu, obsahujúci HTML stránku a tagy značkovacieho jazyka Django, ktorý umožňuje do šablóny vkladať premenné, prípadne vykonávať iterácie (cykly) a podmienky. Pohľad uzivatel teda vypíše detaily o používateľovi, pohľad clanek vypíše obsah článku.

Pohľadov máme veľa, napr. Pre funkcionalitu s entitou užívateľa: uzivatel_registrace, uzivatel_prihlaseni, uzivatel_profil a podobne. Pohľad uzivatel_profil je ale už spoločný všetkým užívateľom a sú do neho posielané rôzne dáta, vždy podľa toho, koho zrovna zobrazujeme. Tieto dáta sú potom dosadená do HTML elementov šablóny.

Šablóny možno samozrejme vkladať do seba, aby sme sa neopakovali (šablóna s layoutom stránky, šablóna s menu a šablóna s článkom).

Poznámka: View nie je len šablóna, ale zobrazovač výstupu. Obsahuje teda minimálne množstvo logiky, ktorá je pre výpis nutná (napr. Kontrola, či si užívateľ vyplnil prezývku pred jej vypísaním alebo cyklus s komentármi, ktoré sa vypisujú).

View podobne ako Model vôbec nevie, odkiaľ mu dáta prišla, stará sa len o ich zobrazenie užívateľovi.

Controller

Controller je teraz onen chýbajúci prvok, ktorý osvetlí funkčnosť celého vzoru. Ide o akéhosi prostredníka, s ktorým komunikuje užívateľ, model i view. Drží teda celý systém pohromade a komponenty prepája. Jeho funkciu pochopíme z ukážky životného cyklu stránky. Opäť existuje veľa rôznych prístupov, najčastejšie má každá entita (Resource) jeden controller, máme teda UserController, PostController a tak podobne.

Životný cyklus stránky

Životný cyklus začína používateľ, ktorý zadá do prehliadača adresu webu a parametre, ktorými nám oznámi, ktorú podstránku si želá zobraziť. Budeme chcieť zobraziť detail uzivatele s id 15. Urobme si ukážku URL adresy:

```
http://www.domena.cz/uzivatel/detail/15
```

Požiadavku ako prvý zachytí tzv. **Router**. S ruty sme sa už zoznámili. Ten podľa definovaných rout spozná, ktorý controller voláme. V našom prípade voláme UserController.

Daný controller podľa parametrov spozná, čo sa po ňom chce, teda že má zobraziť detail užívateľa. Zavolá model, ktorý používateľa vyhľadá v databáze a vráti jeho údaje. Ďalej zavolá ďalšiu metódu modelu, ktorá napr. Vypočíta vek používateľa. Tieto údaje si controller ukladá do premenných. Nakoniec vyrenderuje view (Template). Názov pohľadu poznáme podľa akcie, ktorú vykonávame. View sú odovzdané premenné s príslušnými dátami. Controller(view) teda poslúchol užívateľa, obstaral podľa parametrov dotazu dáta od modelu a odovzdal ich view(templetu).

View(templete) prijme dáta od Controlleru(view) a vloží ich do pripravenej šablóny. Hotová stránka je zobrazená užívateľovi, ktorý často o celej tejto kráse ani netušia :)

MVC architektúra nám uľahčuje aj myslenia pri vývoji projektu. Keď píšem logiku, patrí do modelu, formátovanie a štýlovanie výstupu riešim v šablóne, to čo užívateľ chce z parametrov zisťujem v Controlleru. 3 rôzne problémy na 3 rôznych miestach, oddelené tak, aby do seba nezasahovali a nerobili nám vývoj zložitejši.

MVT = MVC

Framework **Django** implementuje MVC architektúru presne tak, ako sme si ju popísali. Jednotlivé komponenty ale nazýva po svojom a bohužiaľ názov jednej prehodil, čo môže byť mätúce.

Modely - Modelom hovorí Django `modely`. **Views** - pohľadom hovorí Django `Templates`, čo je v preklade šablóny. **Controllers** - kontrolérom Django hovorí `Views`.

Nenechajte sa teda zmiasť, ked tvoríme nový **View**, nie je to **HTML šablóna**, ale onen **Controller** medzi Modelom a Šablónou (templetom). V minulej aplikácii sme svoje prvé view implementovati ako metódu index(). Tá ešte nepoužívala model ani šablónu, ale iba vrátila textovú odpoveď užívateľmi.

3.1.9 Statické súbory

Teraz sa vrhneme do **static files**, to jest úložisko súborov, ktoré budeme používať v našich templates. Povedzme, že si budeme chcieť do svojho **index.html** pridať obrázok a uvádzať k nemu absolútnu cestu nie je práve rozumné. Preto má Django podporu pre statické súbory. Každá aplikácia má svoju zložku static, kde sú tieto súbory uložené. Rovnako ako tomu bolo pri template, aj tu Django po spustení servera pristupuje ku zložkám static ako k jednej zložke. Je len na vás do akéj aplikácie si daný obrázok / súbor vložíte, ale ak bude obrázok pre aplikáciu `moviebook`, tak je oveľa lepší obrázok uložiť do `/mysite/moviebook/static/moviebook/obrazek.jpg`.

Poznámka: Zložku so static files e potrebné nastaviť v `/mysite/mysite/settings.py` :

```
STATIC_ROOT = os.path.join(BASE_DIR, "static/")
```

Keď už máme nastavené, je potreba súbory zhromaždiť. Túto akciu vykonáme príkazom:

```
py manage.py collectstatic
```

Teraz si môžeme tento obrázok vložiť do nášho template :

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>MovieBook</title>
</head>
<body>
        {% load static %} <!-- Slouží pro načtení statických souborů -->
        Název filmu: {{nazev_filmu}} <br>
        Žánr: {{zanr}} <br>
        Hodnocení: {{hodnoceni}} <br>
        Náš obrázek: <br>
        <img src={% static 'moviebook/obrazek.jpg' %} alt=""> <!-- Zde vkládáme obrázek -
↪->
</body>
</html>
```

3.1.10 Databáza

Ako databáze budeme používať **SQLite**, ktorá je v Django už prednastavené a nemusí sa na rozdiel od iných databáz inštalovať ani konfigurovať. Ak by ste v budúcnosti vytvárali komplexnejšie aplikácie, je vhodnejšie použiť napr. databázu MySQL https://www.itnetwork.sk/mysql .

Modely

S databázou budeme pracovať pomocou tzv. **ORM** (Objektovo-Relačná Mapovanie). To znamená, že budeme pracovať s objektmi, a toto počínanie nám bude Django na pozadí automaticky prevádzať na databázové príkazy.

K vytvoreniu databázových tabuliek teda nespustíte zakladacia SQL príkazy, ako ste možno boli zvyknutí, ale vytvoríme triedy reprezentujúci databázové entity. Korešpondujúce tabuľky budú neskôr založené automaticky. Vytvorme si entity Film a Zanr.

Prejdeme do súboru `mysite/moviebook/models.py` a upravime jeho obsah do nasledujúcej podoby:

```
from django.db import models

class Film(models.Model):
        nazov = models.CharField(max_length=200)
        recenzie = models.CharField(max_length=180)

class Zaner(models.Model):
        film = models.ForeignKey(Film, on_delete=models.CASCADE)
        nazov_zanru = models.CharField(max_length=80)
```

Migrácia

Úprave databázy tak, aby zodpovedala definícii modelov v našej aplikácii, sa hovorí **migrácie**. Tento proces musíme spustiť zakaždým, keď vykonáme zmenu v definícii dátové štruktúry a potrebujeme, aby **Django** na jej základe databázu upravilo, v našom prípade dokonca vytvorilo.

Databázovú migráciu najprv vytvoríme príkazom:

```
py manage.py makemigrations [nazov aplikacie]
```

Potom migráciu spustíme:

```
py manage.py migrate
```

Django API

Teraz si ukážeme prácu s **Django API**, teda ako do databázy vkladať nové riadky ako objekty a ako objekty z databázy tiež získavať.

Ukážku vykonáme v interaktívnom shellu, ktorý spustíme pomocou:

```
py manage.py shell
```

A do neho napíšeme nasledujúci kód:

```
from moviebook.models import Film, Zanr
muj_film = Film(nazev="Strazci Galaxie", rezie="James Gunn") # Vytvoříme si nový film
muj_film.nazev # Zobrazí název filmu
muj_film.save() # Uloží film do DB

Film.objects.all() # Zobrazí všechny existující filmy
muj_film.zanr_set.all() # Zobrazí všechny žánry k danému filmu
muj_film.zanr_set.create(nazev_zanru="Fantasy/Action") # Vytvoří nový žánr k tomuto filmu
```

Rozšírenie modelov

Upravime naše modely tak, aby nám vracali názov a meno režiséra. Podobne upravíme aj žáner.

```
from django.db import models

class Zaner(models.Model):
        nazov_zanru = models.CharField(max_length=80)

        def __str__(self):
                return "Nazov_zanru: {0}".format(self.nazov_zanru)

class Film(models.Model):
        nazov = models.CharField(max_length=200)
        rezie = models.CharField(max_length=180)
        zaner = models.ForeignKey(Zaner, on_delete=models.SET_NULL, null=True)

        def __str__(self):
                return "Nazov: {0} | Rezie: {1} | Zaner: {2}".format(self.nazov, self.
↪rezie, self.zaner.nazov_zanru)
```

Administrácia databázy

Konečne sa dostávame k administrácii. Superuser máme vytvoreného.

Najprv je potrebné naše modely do administrácie zaregistrovať a to úpravou súboru `/mysite/moviebook/admin.py` :

```
from django.contrib import admin
from .models import Film, Zaner #Importujeme si modely

#Modely registrujeme
admin.site.register(Film)
admin.site.register(Zaner)
```

Nasledne si spustíme server a zamierime si to na adresu `http://localhost:8000/admin/`.

Meta

Meta, slúži na ukladanie / nastavenie informácií navyše, ako je v tomto prípade názov množného čísla entity nastaviteľný pomocou `verbose_name_plural`. Upravime súbor ``/mysite/moviebook/models.py`` :

```
from django.db import models

class Zaner(models.Model):
        nazov_zanru = models.CharField(max_length=80, verbose_name="Žáner")

        def __str__(self):
                return "Nazov_zanru: {0}".format(self.nazov_zanru)

        class Meta:
                verbose_name="Žáner"
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
                verbose_name_plural="Žánre"

class Film(models.Model):
        nazov = models.CharField(max_length=200, verbose_name="Názov Filmu")
        rezie = models.CharField(max_length=180, verbose_name="Režia")
        zaner = models.ForeignKey(Zaner, on_delete=models.SET_NULL, null=True, verbose_
↪name="Žáner")

        def __str__(self):
                return "Nazov: {0} | Rezie: {1} | Zaner: {2}".format(self.nazov, self.
↪rezie, self.zaner.nazov_zanru)

        class Meta:
                verbose_name="Film"
                verbose_name_plural="Filmy"
```

3.1.11 Generic views

Generic views sú predpripravené views pre jednoduché akcie, ktoré sa vo webových aplikáciách často používajú. Práve tie využijeme pre pridávanie a editáciu záznamov v našej databáze, aby sme nemuseli písať všetko znova.

ListView

Ako prvý generic view si vyskúšame **ListView**, ktoré vypíše zoznam položiek. V našom prípade si ním samozrejme necháme vypísať všetky filmy z databázy. Súbor `/mysite/moviebook/views.py` upravíme do nasledujúcej podoby:

```
    class FilmIndex(generic.ListView):

template_name = "moviebook/film_index.html" # cesta k templatu ze složky templates (je␣
↪možné sdílet mezi aplikacemi)
context_object_name = "filmy" # pod tímto jménem budeme volat list objektů v templatu

    # tato funkce nám získává list filmů seřazených od největšího id (9,8,7...)
            def get_queryset(self):
                    return Film.objects.all().order_by("-id")
```

Ako prvé je potrebné naimportovať generic views a samotné modely.

View teraz už nie je tvorené obyčajnou metódou, ale triedou dedičov z generic.ListView, prípadne z iného generického pohľadu. Keď sa nad tým zamyslíte, je to logické, pretože práve dedičnosťou sa nám do view dostane predpripravená funkčnosť.

- Generic view nastavíme šablónu a ako sa má premenná s jednotlivými prvkami zoznamu v šablóne vymenovať.
- Následne definujeme metódu pre získanie všetkých filmov, ktoré si zoradíme od posledne pridaných po tie najstaršie.

Na tento view si vytvoríme odkaz v `/mysite/moviebook/urls.py` :

```
from django.urls import path
from . import views
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
urlpatterns = [
        path("film_index/", views.FilmIndex.as_view(), name="filmovy_index"),
]
```

Template pre náš prvý generic view ListView bude v súbore `/mysite/moviebook/templates/moviebook/film_index.html` s nasledujúcim obsahom:

```
<!DOCTYPE html>
<html lang="cs">
<head>
        <meta charset="UTF-8">
</head>
<body>
        {% for film in filmy %}
                Název: {{film.nazov}} <br>
        {% endfor %}
</body>
</html>
```

DetailView

Teraz by bolo dobré vytvoriť si aj view pre detail vybraného filmu. K tomu nám pomôže **DetailView**, ktoré nám o filme zobrazia všetky podrobnosti.

Na koniec `/mysite/moviebook/views.py` pridáme:

```
class CurrentFilmView(generic.DetailView):

        model = Film
        template_name = "moviebook/film_detail.html"
```

View je opäť trieda, oddděná z generického predka.

Poznámka: V prípade detaile nastavujeme len model a názov šablóny.

Šablóna

Šablónu pre view vytvoríme v `/mysite/moviebook/templates/moviebook/film_detail.html`, ktorý sme v detail view vyššie nastavili. Všimnite si pomenovanie súborov, kedy je názov zložený z názvu entity, podčiarknutia a názvu generic view. Obsah šablóny je nasledujúci:

```
<!DOCTYPE html>
<html lang="cs">
<head>
        <meta charset="UTF-8">
</head>
<body>
        <h1> {{ film.nazov }} </h1><small> {{ film.rezie }} </small>
        <h3> {{film.zaner.nazov_zanru}} </h3>
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
</body>
</html>
```

Každý DetailView potrebuje poznať ID / PK (primárny kľúč) konkrétneho filmu, pre ktorý nám bude zobrazovať všetky informácie, aby si ho mohol z databázy načítať. Odkaz na film preto bude obsahovať aj PK konkrétneho filmu.

Upravíme súbor `/mysite/moviebook/urls.py` :

```
path("<int:pk>/film_detail/", views.CurrentFilmView.as_view(), name="film_detail"),
```

Tým sme Dajngo vysvetlili, že keď niekto zadá URL adresu na detail filmu, je to číslo pred ňou primárny kľúč (ID) tohto filmu.

Teraz odkaz na film pridáme do výpisu filmov, teda do šablóny `/mysite/moviebook/templates/moviebook/film_index.html`.

Obyčajný užívateľ sa totiž zatiaľ stále nemá ako dostať na stránku s informáciami o filme. Do odkazu nezadáme absolútnu adresu, ale použijeme názov URL, ktorý sme v routách uviedli (v tomto prípade name="filmovy_detail"), ako parameter odovzdáme ID / PK. URL je takto jednoduchšie a keby sa adresa pohľadu niekedy zmenila, táto zmena sa prejaví bez nutnosti šablónu upraviť.

Pridajme teda odkaz do `film_detail.html` :

```
<!DOCTYPE html>
<html lang="cs">
<head>
        <meta charset="UTF-8">
</head>
<body>
        {% for film in filmy %}
           <a href={% url "filmovy_detail" film.id %}> Název: {{film.nazev}} <br> </a>
        {% endfor %}
</body>
</html>
```

Formulár

Filmy vieme vypisovať a zobrazovať ich detail. Pridanie filmu urobíme pomocou triedy ModelForm.

Formulár by sme samozrejme mohli vytvoriť oldschool cestou len ako čisté HTML, ako sme to robili v kalkulačke, ale Django tu máme práve preto, aby sme sa naučili ako si s ním uľahčiť prácu.

Vytvoríme si nový modul `/mysite/moviebook/forms.py`, v ktorom sa bude nachádzať náš formulár:

```
from django import forms
from .models import Film

class FilmForm(forms.ModelForm):

        class Meta:
                model = Film
                fields=["nazov", "rezie", "zaner"]
```

Asi vás neprekvapí, že ako ďalší krok si vytvoríme view, ktoré bude stránku s formulárom obsluhovať. Do view `/mysite/moviebook/views.py` pridáme nový import práve na náš formulár:

```
from django.shortcuts import render, redirect, render_to_response
from django.views import generic

from .models import Film
from .forms import FilmForm #Nový import

# ...
```

CreateView

Okrem importu na koniec súboru pridáme obsluhu formulára, ktorú si vzápätí vysvetlíme.

Použijeme pre ňu generic view CreateView. Vidíme ako môžeme z Django prevziať veľa funkcionality, ktorú by sme inak museli implementovať sami.

```
# ...

class CreateFilm(generic.edit.CreateView):

        form_class = FilmForm
        template_name = "moviebook/create_film.html"

# Metoda pro GET request, zobrazí pouze formulář
        def get(self, request):
                form = self.form_class(None)
                return render(request, self.template_name, {"form":form})

# Metoda pro POST request, zkontroluje formulář; pokud je validní, vytvoří nový film;␣
↪pokud ne, zobrazí formulář s chybovou hláškou
        def post(self, request):
                form = self.form_class(request.POST)
                if form.is_valid():
                        form.save(commit=True)
                return render(request, self.template_name, {"form":form})
```

View nastavujeme formulár a šablónu.

Ďalej obsahuje 2 akcie, get() formulár iba zobrazuje a post() ho spracováva v prípade, že už bol odoslaný.

Všimnite si, že v oboch akciách formulár odovzdávame pomocou listu do šablóny, aby sme ho tam mohli vykresliť. Určite by sme sa nemali v metóde post() zabudnúť opýtať, či bol formulár validný vyplnený. Keďže formulár vie aký model spracúva, pre uloženie filmu na ňom stačí zavolať len metódu save() a je hotovo.

Pre nových view a teda novú adresu si ako vždy pridáme route v `/mysite/moviebook/urls.py` :

```
from django.urls import path
from . import views

urlpatterns = [
        path("film_index/", views.FilmIndex.as_view(), name="filmovy_index"),
        path("<int:pk>/film_detail/", views.CurrentFilmView.as_view(), name="filmovy_
↪detail"),
        path("create_film/", views.CreateFilm.as_view(), name="novy_film"),
]
```

Šablóna

Vytvoríme si template `/mysite/moviebook/templates/moviebook/create_film.html` pre náš form :

```
<!DOCTYPE html>
<html lang="cs">
<head>
        <meta charset="UTF-8">
</head>
<body>
        <form method="POST">
                {% csrf_token %} <!-- Django požaduje ověření proti útoku csrf  -->
                {{ form }}
                <input type="submit">
        </form>
</body>
</html>
```

Po odoslaní validného formulára sa vytvorí nový film. Po prechode na http://localhost:8000/moviebook/film_index si môžete vyskúšať, že sa medzi filmami naozaj zobrazí.

3.1.12 Internationalization

Pred vygenerovanim multijazycnych suborov je potrebne upravit subor `settings.py` :

```
LOCALE_PATHS = [
        os.path.join(BASE_DIR, "locale")
]
```

Vytvorte súbor správ:

- Z hlavnej ponuky vyberte Tools | Spustite úlohu manage.py (R) .
- V okne úlohy manage.py zadajte

```
makemessages --locale <názov prostredia>
```

Tento krok zopakujte pre každé miestne nastavenie, ktoré chcete vytvoriť.

Ak sú na lokalizáciu označené reťazce, PyCharm vytvorí adresár a súbor **django.po** pre každé miestne nastavenie:

PyCharm provides a dedicated intention action to wrap strings in Django templates in {%trans%}, or {%blocktrans%} tags.

Pri prvom volani je potrebne do sablony dotiahnut `` {% load i18n%} ``

Compile a message file

Povytvoreni lokalizacnych suborov je potrebne subory skompilovat :

```
py manage.py compilemessages
```

KAPITOLA 4

Sphinx

Sphinx je nástroj, ktorý uľahčuje vytváranie inteligentnej a krásnej dokumentácie, ktorú napísal Georg Brandl a je licencovaná pod licenciou BSD.

Pôvodne bol vytvorený pre dokumentáciu Pythonu a má vynikajúce možnosti na dokumentáciu softvérových projektov v rôznych jazykoch. Samozrejme, táto stránka je tiež vytvorená zo zdrojov reStructuredText pomocou Sphinx! Mali by sa zdôrazniť tieto vlastnosti:

Výstupné formáty:

- HTML (vrátane Windows HTML Help),
- LaTeX (pre tlač PDF verzie),
- ePub,
- Texinfo,
- Plain text

Sphinx používa **reStructuredText** ako svoj značkovací jazyk a mnohé z jeho silných stránok pochádzajú zo sily a priamočiarosti reStructuredText a jeho sady na analýzu a preklad, Docutils.

4.1 reStructuredText Directives

- Direktívy začínajú explicitným označením začiatku (dve bodky a medzera), po ktorom nasleduje typ smernice a dve dvojbodky (spoločne „značka direktívy“).
- Direktívny blok začína bezprostredne za direktívnou značkou a obsahuje všetky nasledujúce odsadené riadky.
- Direktívny blok je rozdelený na argumenty, možnosti (zoznam polí) a obsah (v tomto poradí), z ktorých sa môže objaviť ktorýkoľvek. Podrobnosti o syntaxi nájdete v časti Smernice v špecifikácii značky reStructuredText.

V popisoch nižšie sú uvedené „prvky dokumentu“ (názvy prvkov stromu dokumentov; generické identifikátory XML DTD) zodpovedajúce jednotlivým smerniciam.

Podrobnosti o hierarchii prvkov nájdete v dokumente Docutils Document Treem a Docutils Generic DTD XML definícia typu dokumentu. Podrobnosti o implementácii direktív nájdete v časti Vytváranie direktív reStructuredText.

4.2 Images

Existujú dve obrazové smernice: „image“ a „figure“.

Je na autorovi, aby zabezpečil kompatibilitu formátu obrazových údajov s výstupným formátom alebo užívateľským agentom (LaTeX engine, HTML prehliadač, ODT, ...).

Nasledujúca tabuľka poskytuje prehľad ...

4.2.1 Image

```
.. image:: img/intro.jpg
```

Klikatelny image :

4.3 Integracia Sphinx do Laravel

Integracia je robena cez **websupport** modul a navod je popisany aj v SPHINX doc

sdsd sdasd

4.4 Markdown

Markdown je ľahký značkovací jazyk so zjednodušenou syntaxou formátovania obyčajného textu. Existuje v mnohých syntakticky odlišných príchutiach. Na podporu dokumentácie založenej na Markdown môže Sphinx použiť **MyST-Parser**.

4.4.1 Instalacia

Pre pouzitie markdownu vo vlastnom projekte postupujte nasledovne :

1. Nainstaluj Markdown parser **MyST-Parser**

```
$ pip install --upgrade myst-parser
```

2. Pridaj **MyST-Parser** do extension v konfiguracii projektu `conf.py`

```
extensions = ['myst_parser']
```

Poznámka: MyST-Parser vyzaduje Sphinx 2.1 alebo novsi.

---

3. Ak chcete použiť súbory Markdown s inými príponami ako .md, upravte premennú **source_suffix**.Nasledujúci príklad konfiguruje Sphinx na analýzu všetkých súborov s príponami .md a .txt ako Markdown:

```
source_suffix = {
        '.rst': 'restructuredtext',
        '.txt': 'markdown',
        '.md': 'markdown',
}
```

4. MyST-Parser môžete ďalej nakonfigurovať tak, aby umožňoval vlastnú syntax, ktorú štandardná CommonMark nepodporuje. Viac v dokumentácii MyST-Parser.

4.4.2 Zbierka rozšírení Sphinx

Nástroje v ekosystéme Sphinx na písanie krásnych online kníh a dokumentov:

- sphinx-copybutton
- sphinx-togglebutton
- sphinx-panels

4.5 Reference

V podstate sphinx dokumentacia umoznuje viacero pristupov k odkazovaniu na ci uz interne alebo externe zdroje :

1. Odkazovanie na interny dokument v projekte 1. Odkazovanie na internu sekciu v akomkolvek dokumente 1. Odkazovanie na externu url

4.5.1 Link na dokument

Odkaz na určený dokument :doc:; názov dokumentu môže byť špecifikovaný v absolútnom alebo relatívnom zmysle.

Napríklad, ak sa odkaz :doc:`parrot` vyskytuje v sketches/indexe dokumentu, potom odkaz odkazuje na sketches/parrot.

Ak je odkaz :doc:`/people` alebo :doc:`../people`, odkaz odkazuje na ľudí.

Ak nie je uvedený žiadny explicitný text odkazu (ako zvyčajne: :doc:`Členovia Monty Python </people>`), titulok odkazu bude názvom daného dokumentu.

4.5.2 Link na sekciu

Pri použití **ref** je možné vygenerovať krížový odkaz s definovaným iba názvom, za predpokladu, že je uvedený explicitný názov.

Príklad:

```
See :ref:`this code snippet <this-py>` for an example.
```

Na podporu krížových odkazov na ľubovoľné miesta v akomkoľvek dokumente sa používajú štandardné štítky ReST.

Poznámka: Názvy štítkov musia byť jedinečné v celej dokumentácii.

Referenčné štítky musia začínať podčiarkovníkom. Pri odkazovaní na štítok je potrebné vynechať podčiarknutie (pozri príklady vyššie).

Existujú dva spôsoby, ako môžete odkazovať na štítky:

1. Odkaz na sekciu so stitkom

Ak štítok umiestnite priamo pred názov sekcie, môžete naň odkazovať pomocou :ref:`názov-názvu`.

Napríklad:

```
.. _my-reference-label:

Section to cross-reference
--------------------------

This is the text of the section.

It refers to the section itself, see :ref:`my-reference-label`.
```

Rola :ref: potom vygeneruje odkaz na sekciu s názvom odkazu „Sekcia na krížový odkaz“.

Funguje to rovnako dobre, keď sú sekcia a odkaz v rôznych zdrojových súboroch.

2. Odkaz na nazov sekcie bez stitku

Na štítky, ktoré nie sú umiestnené pred názvom sekcie, možno stále odkazovať, ale odkazu musíte prideliť explicitný názov pomocou tejto syntaxe: :ref:`Názov odkazu <názov-názvu>`.

Použitie **ref** sa odporúča pred štandardnými odkazmi reStructuredText na sekcie (ako je `Názov sekcie`_), pretože funguje naprieč súbormi. Keď sa zmenia nadpisy sekcií, zobrazí sa varovanie ak sú nesprávne.

4.5.3 Link na externu url

Explicit external targets are interpolated into references such as „Python“.

For others, please see:

http://docutils.sourceforge.net/docs/ref/rst/directives.html.

4.6 Sample

V tejte sekcii su zozbrazene veskere ukazky pouzitia.

4.6.1 Code block

Ukazka blokov pomocou direktivy `.. code-block:: **[language]**`.

YAML

```
# Configuration in YAML
```

XML

```
<!-- Configuration in XML -->
```

PHP

```
// Configuration in PHP
```

HTML+PHP

```
<! Test pre HTML !>
```

CONSOLE

```
$ pip install lumache
```

BASH

```
# common.sh
pip install my-dependency==1.2.3
```

PYTHON

```
from celery.contrib import rdb; rdb.set_trace()
```

INI

```
# Configuration for nova-rootwrap
# This file should be owned by (and only-writeable by) the root user

[DEFAULT]
# List of directories to load filter definitions from (separated by ',').
```

Zaroven je mozne aj pouzitie alternativnej direktivy `.. code:: **[language]**`.

```
import sphinx_rtd_theme

extensions = [
    ...
    'sphinx_rtd_theme',
]

html_theme = "sphinx_rtd_theme"
```

Code block s cislovanymi riadkami s podfarbenim a caption ...

Výpis 1: Code Blocks can have captions.

```
1 def some_function():
2     interesting = False
3     print 'This line is highlighted.'
4     print 'This one is not...'
5     print '...but this one is.'
```

4.6.2 GUI label

```
:guilabel:`Some action`
```

Ina ukazka pre block

```
:guilabel:`Some action`
```

Some action

4.6.3 Centered text

You can create a statement with centered text with `.. centered::`

This is centered text!

4.6.4 Admonitions

Výstraha: Directives at large.

Upozornenie: Don't take any wooden nickels.

Nebezpečné: Mad scientist at work!

Chyba: Does not compute.

Rada: It's bigger than a bread box.

Dôležité:

- Wash behind your ears.
- Clean up your room.
  - Including the closet.
  - The bathroom too.
    * Take the trash out of the bathroom.
    * Clean the sink.
- Call your mother.
- Back up your data.

Poznámka: This is a note. Equations within a note: $G_{\mu\nu} = 8\pi G(T_{\mu\nu} + \rho_\Lambda g_{\mu\nu})$.

Tip: 15% if the service is good.

Example
Thing1
Thing2
Thing3

Varovanie: Strong prose may provoke extreme mental exertion. Reader discretion is strongly advised.

And, by the way…

You can make up your own admonition too.

4.7 Download Links

`Stiahni JPG`

4.7.1 Target

Implicitny odkaz na nadpis bez stitka *Glossary*

4.8 Blocks

4.8.1 Literal Blocks

Literal blocks are indicated with a double-colon („::“) at the end of the preceding paragraph (over there `-->`). They can be indented

```
if literal_block:
    text = 'is left as-is'
    spaces_and_linebreaks = 'are preserved'
    markup_processing = None
```

Or they can be quoted without indentation:: >> Great idea! > > Why didn’t I think of that?

4.8.2 Sidebar

A code example

With a sidebar on the right.

4.9 Glossary

Documentation Provides users with the knowledge they need to use something.

Reading The process of taking information into ones mind through the use of eyes.

4.10 HTML Theming

Sphinx poskytuje množstvo hotovych tem pre HTML a formáty založené na HTML.

Poznámka: Táto časť poskytuje informácie o používaní už existujúcich tém HTML. Ak si chcete vytvoriť vlastnú tému, pozrite si tému Vývoj témy HTML.

Sphinx podporuje zmenu vzhľadu svojho HTML výstupu prostredníctvom tém. Téma je kolekcia HTML šablón, štýlov a iných statických súborov. Okrem toho má konfiguračný súbor, ktorý určuje, z ktorej témy sa má dediť, aký štýl zvýraznenia sa má použiť a aké možnosti existujú na prispôsobenie vzhľadu a štýlu témy.

Témy sú navrhnuté tak, aby nezohľadňovali projekt, takže ich možno bez zmeny použiť pre rôzne projekty.

4.10.1 Použitie témy

Použitie témy v Sphinx je jednoduché. Keďže ich nie je potrebné inštalovať, stačí nastaviť konfiguračnú hodnotu html_theme.

Ak chcete napríklad povoliť klasickú tému, pridajte do `conf.py` nasledovné:

```
html_theme = "classic"
```

Pre danu temu môžete tiež nastaviť možnosti špecifické pre tému pomocou konfiguračnej hodnoty `html_theme_options`. Tieto možnosti sa vo všeobecnosti používajú na zmenu vzhľadu a štýlu témy.

Ak chcete napríklad umiestniť bočný panel na pravú stranu a čierne pozadie pre panel vzťahov (panel s navigačnými odkazmi v hornej a dolnej časti stránky), pridajte nasledujúci `conf.py` :

```
html_theme_options = {
"rightsidebar": "true",
"relbarbgcolor": "black"
}
```

4.10.2 Oblubene témy

Medzi najoblubenejsie temy patria nasledujuce :

Read the Docs Sphinx Theme

Táto téma Sphinx bola navrhnutá tak, aby používateľom dokumentácie na stolných aj mobilných zariadeniach poskytla skvelú čitateľskú skúsenosť.

Táto téma sa bežne používa s projektmi na **Read the Docs**, ale môže pracovať s akýmkoľvek projektom Sphinx.

Dokumentacia je dostupna TU.

Instalacia a aktivacia témy

Nainštaluj balík (alebo ho pridajte do svojho súboru `requirements.txt`):

```
$ pip install sphinx_rtd_theme
```

Vloz do suboru `conf.py` :

```
import sphinx_rtd_theme

extensions = [
        'sphinx_rtd_theme',
]

html_theme = "sphinx_rtd_theme"
```

Poznámka: Pridanie tejto témy ako rozšírenia umožňuje lokalizáciu reťazcov tém vo vašom preloženom výstupe.

Ak tieto reťazce nie sú preložené vo vašom výstupe, buď nám chýbajú lokalizované reťazce pre vaše miestne nastavenie, alebo používate starú verziu témy.

Via Download

Varovanie: Inštalácia priamo zo zdroja úložiska je zastaraná a neodporúča sa. Od vydania 3.0.0 nebudú do úložiska zahrnuté statické aktíva.

Stiahni balik z repozitára sphinx_rtd_theme/sphinx_rtd_theme do vašej dokumentácie na docs/_themes/sphinx_rtd_theme a potom pridajte do svojho súboru Sphinx `conf.py` nasledujúce dve nastavenia:

```
html_theme = "sphinx_rtd_theme"
html_theme_path = ["_themes", ]
```

Konfiguracia

Nasledujúce možnosti možno definovať v súbore `conf.py` vášho projektu pomocou konfiguračnej možnosti **html_theme_options**.

Napríklad:

```
html_theme_options = {
        'analytics_id': 'G-XXXXXXXXXX',  #  Provided by Google in your dashboard
        'analytics_anonymize_ip': False,
        'logo_only': False,
        'display_version': True,
        'prev_next_buttons_location': 'bottom',
        'style_external_links': False,
        'vcs_pageview_mode': '',
        'style_nav_header_background': 'white',
        # Toc options
        'collapse_navigation': True,
        'sticky_navigation': True,
        'navigation_depth': 4,
        'includehidden': True,
        'titles_only': False
}
```

Table of contents options

Nasledujúce voľby menia spôsob, akým direktívy toctree generujú navigáciu v dokumentácii.

Book Theme

Téma ** Book Theme Sphinx** s čistým dizajnom, podporou interaktívneho obsahu a moderným vzhľadom a štýlom ako z knihy.

Dokumentacia je dostupna TU.

Get started

Poznámka: Táto dokumentácia a nižšie uvedené príklady sú napísané pomocou **MyST Markdown**, čo je forma markdown, ktorá funguje so Sphinx.

Blizsie informacie su popisane na stranke *Konfiguracia markdown*

Predpoklady

Mali by ste byť relatívne oboznámení s ekosystémom Sphinx a mali by ste ho mať nainštalovaný lokálne vo svojom počítači.

Instalacia a aktivacia témy

Najprv si nainštalujte tému **sphinx-book-theme** s pip:

```
$ pip install sphinx-book-theme
```

Potom aktivuj tému v konfigurácii Sphinx `conf.py` :

```
html_theme = "sphinx_book_theme"
```

Tým sa aktivuje téma Book Sphinx pre vašu dokumentáciu.

Poznámka: Možno budete musieť zakomentovať svoju konfiguráciu html_theme_options v závislosti od vašej predchádzajúcej témy.

Zdrojové úložisko

Pridajte do svojej témy tlačidlo zdrojového úložiska.

Existuje niekoľko spôsobov, ako si môžete prispôsobiť tému Book Sphinx.

V tomto priklade ukazeme pridanie úložiska GitHub.

Ak chcete pridať tlačidlo, ktoré odkazuje na úložisko vašej dokumentácie, pridajte nasledujúcu konfiguráciu do `conf.py`.

```
html_theme_options = {
        "repository_url": "https://github.com/{your-docs-url}",
        "use_repository_button": True,
}
```

Bočný panel

Existuje niekoľko hlavných oblastí stránky, ktoré si môžete prispôsobiť podľa témy knihy. Najbežnejší je primárny bočný panel (štandardne na ľavej strane stránky).

Prvky bočného panela sú definované ako šablóny HTML podľa Sphinx, rozšírení Sphinx a aktuálnej témy. Pomocou konfigurácie `html_sidebars` môžete určiť, ktoré stránky by mali obsahovať prvky bočného panela. Tento krok v tomto návode preskočíme, ale detaili su popisane v Control the left sidebar items .

Prispôsobme si logo našej stránky

Za týmto účelom upravíme možnosti konfigurácie `html_logo` a `html_title`.

Ak chcete pridať logo lokality, postupujte takto:

1. Vložte obrázok do koreňového adresára dokumentácie (napr. myimage.png).
2. Pridajte do súboru `conf.py` riadok, ktorý vyzerá takto:

```
html_logo = "path/to/myimage.png"
```

Ďalej prispôsobíme názov stránky.

Ak to chcete urobiť, pridajte do súboru `conf.py` riadok, ktorý vyzerá takto:

```
html_logo = "html_title = "My site title""
```

Okraje stranky

Existuje niekoľko špeciálnych druhov obsahu, ktoré táto téma podporuje, prevažne inšpirovaných štýlmi tém Tufte CSS.

Jeden druh obsahu je obsah na okraji. To vám umožní „vysunúť“ obsah na okraj stránky, na okraje.

Pridajte na stránku nejaký obsah okrajov pridaním nasledujúcej smernice (napísanej pomocou *MyST Markdown*).

```
{margin} Look, some margin content!
On wider screens, this content will pop out to the side!
```

Na širokouhlých obrazovkách sa obsah okrajov vysunie na stranu stránky a umožní obsahu pod ním pohybovať sa nahor.

To vám umožní poskytnúť ďalšie informácie bez prerušenia toku informácií.

Existuje mnoho ďalších vecí, ktoré môžete robiť s témou knihy Sphinx. Teraz, keď ste začali, pozrite si ďalšie časti vľavo, kde sa dozviete viac o tom, ako ho používať.

4.11 Tabulky

Su dostupne 3 moznosti :

4.11.1 Table

Tabuľka 1: Truth table for „not“

A	not A
False	True
True	False

Rozpoznávajú sa nasledujúce možnosti:

- **align** - „left“, „center“, „right“
- **width** - length , percentage
- **widths** - „auto“, „grid“, list of integers

4.11.2 CSV table

Tabuľka 2: Frozen Delights!

Treat	Quantity	Description
Albatross	2.99	On a stick!
Crunchy Frog	1.49	If we took the bones out, it wouldn’t becrunchy, now would it?
Gannet Ripple	1.99	On a stick!

Možnosti:

- align - „left“, „center“, or „right“

4.11.3 List table

Tabuľka 3: Frozen Delights!

Treat	Quantity	Description
Albatross	2.99	On a stick!
Crunchy Frog	1.49	If we took the bones out, it wouldn’t be crunchy, now would it?
Gannet Ripple	1.99	On a stick!

Možnosti:

KAPITOLA 5

Tailwind css

KAPITOLA 6

Vue.js

Vue.js (nebo jen Vue; vyslovuje se stejně jako view) je open-source progresivní JavaScriptový framework pro vytváření uživatelských rozhraní.Začlenění do projektů, které používají jiné JavaScriptové knihovny je s Vue snadné, protože je navržen tak, aby mohl být přijímán postupně. Vue může také fungovat jako webový aplikační framework, na kterém je možné vytvářet pokročilé Single-page applications. Zakladnu dokumentaciu najdes [TU](https://vuejs.org/).

Stavia na štandarde HTML, CSS a JavaScript a poskytuje deklaratívny a komponentný programovací model, ktorý vám pomáha efektívne rozvíjať používateľské rozhrania, či už jednoduché alebo zložité.

Poznámka: Detail pre implemnentaciu **Vue.js** do laravel projektu popisujem v sekcii *Vue.js*.

6.1 Vseobecny postup instalacie NPM Packages

1. Nainstaluj komponentu cez NPM

```
npm install --save "NAZOV_KOMPONENTY"
```

2. Registruj komponentu v subore `app.js`

   Registrovat kompenentu mozeme podla potreby :

GLOBALNE

– globalnu registraciu robime v zakladnom subore `resources/assets/js/app.js`

```
// import nainstalovaneho pluginu

import VueFormWizard from 'vue-form-wizard'
import 'vue-form-wizard/dist/vue-form-wizard.min.css'

// registracia komponenty
Vue.use(VueFormWizard)
```

LOKALNE

– robime priamo v inej komponente

```
import {FormWizard, TabContent} from 'vue-form-wizard'
import 'vue-form-wizard/dist/vue-form-wizard.min.css'

//component code
components: {
  FormWizard,
  TabContent
}
```

6.2 Existujuce VUE form komponenty

** Multisteps formulare**

- vue-form-wizard
- vue-good-wizard

** Flash spravy**

- vue-flash-message
- Vuex Flash

Mnou vytvorene komponenty

VUE dirctives v-show - pouziva sa pre zobrazenie elementu v-model - pouziva sa pre previazanie input pola s nejakou premennou @click - odchytava event na kliknutie, zvykne sa odkazovat na metodu @submit - potvrdenie formulara a odoslanie dat

..note:

```
Pre zrusenie defaultne funkcionality pri odoslani formulara je potrebne pouzi ``@submit.
↪prevent``
```

6.2.1 Webrebel

Vue 2 vs. Vue 3

Zakladne rozdiely medzi Vue2 a Vue3 :

- vytvorenie instancie VUE (create App)
- eventBus
- zrusili sa filtre pouzivane cez pipe | (staci prerobit na methods)
- inline template (toto som este nepouzival)

Zakladna kostra VUE 3

```
<script>

        import VueComponent from 'cesta_k_suboru/VueComponent.vue';

        export default {

                // vsetky pouzite komponenty vo VUE komponente
                components: {
                        VueComponent,
                },

                // data ktore pouzivana v komponente
                data {
                        return {
                                legend: 'Meno Legendy',
                                newObject: '',
                        }
                },

                // sekcia pri vzniknutom komponnete
                created() {
                        window.eventBus.on('NAZOV_GLOBALNEHO_EVENTU', (event) => (this.
↪newObject = event) )
                },

                // metody a funkcie pouzite v komponente
                methods: {
                        showAlert() {
                                alert('Legenda : ' + this.legend)
                        },
                        formSubmited() {
                                this.$emit('new-object', this.newObject)
                        },
                },

        }
</script>
```

Events

Eventy si vstavane direktivy vo vue a pouziva sa prefix @

@input

Udalost ked nastane zmena na input fielde

Zrusenie refreshu prehliadaca

```
@submit.prevent
```

Nastavenie metody po potvrdeni form

```
@submit.prevent="newMethod"
```

Nasledne je potrebne vytvorit vo VUE komponente aj metodu

```
<script>
        export default {
          methods: {
                newMethod() {
                        alert('FORM submited!')
                },
          },
        }
</script>
```

Kumunikacia z rodica na potomka

Data z rodica do podradenej komponenty (potomka) sa data posielaju cez `props`

```
<vue-component :myName="Meno v premennej myName" />
```

Kumunikacia z potomka na rodica

V pripade odchytenie eventu potomok vie tuto udalost resp. po odchyteni poslat spravu rodicovi cez `$emit`

```
<script>
        export default {
          methods: {
                newMethod(e) {
                        this.$emit('NAZOV_VLASTNEHO_EVENTU', data)
                },
          },
        }
</script>
```

Po udalisti sa zavola metoda, na ktoru dokaze rodic reagovat po pridani takeho eventu do svojej implementacie komponentu

```
<vue-component @NAZOV_VLASTNEHO_EVENTU="data = @event" />

<p>
{{ data }}
</p>
```

Kumunikacia cez EventBus

Kuminikacia cez na seba nezavisle komponenty.

Vo Vue 3 si zrusilo pouzivanie **EventBus** a odporuca sa pouzit externy package.

Napriek tomu pre velke aplikacie sa odporuca pouzivat **VUEX** na menezovanie stavov a pod cross velku aplikaciu.

Poznámka: Urcite pri budovani vacsej aplikacie by chcelo pouzit **VUEX**

EventBus mitt

Na ukazku pouzijeme externu kniznicu **mitt**

Nainsatlovat

```
npm insatll mitt
```

Do `app.js` alebo `main.js` je potrebne importovat mitt

```
import mitt from 'mitt'

window.eventBus = mitt()
```

Nasledne u podobne ako pri komunikaii s rodicom vieme poslat udalost do Globalneho eventBus-u

```
window.eventBus.emit('NAZOV_GLOBALNEHO_EVENTU', data)
```

mixins

V pripade ze sa nas kod bude v aplikacii opakovat 2 a viac krat , je vhodne pre zachovanie REUSABLE pouzit mixins .

Mixins je v podstate vlastna kniznica s datami a funkciami, ktoru vieme jednoducho pouzitv akomkolvek VUE komponente.

Na ukazku jednoducheho filter mixin:

```
export default {
        data() {
                return {
                        data: [],
                        search: ''
                }
        },
        computed: {
                filteredItems() {
                        return this.data.filter( item => {
                                return item[this.searchColumn]
                                        .toLowerCase()
                                        .includes(this.search.toLowerCase())
                        })
                }
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
        },
}
```

Varovanie: Nezabudnite importovat svoje Mixins do komponenty

```
mixins: [filterMixin]
```

Moment js

Uprava resp. formatovanie datumu na FE tzn. ze ked zo servera nepride naformatovany cas napr. packagom Carbon. N

Nutne doinstalovat package :

```
npm install moment
```

Potom staci do svojeho nejakeho Mixin prihodit funkciu na formatovanie napr. :

..code-block:

```
    import moment from 'moment'

methods: {
            niceDay(dateTime) {
                    return moment(dateTime).fromNow()
            }
    },
```

Nasledne v komponente kde mame dotiahnuty mixin , tak staci vo vypise hodnoty zabalit do funkcie `{{ niceDay( VALUE ) }}`

6.2.2 Best practice

Form

Pouzitie elementu <form> je mozne aj vo vue ale je takmer nevyhnute pridat osetrenie na refresh formulara

```
<form @submit.prevent="MOJA_METODA"
```

REF

Referenciu pouzivama na identifikaciu konkretneho elementu, a to tak ze do elementu pridame tag `ref=VLASTNY_NAZOV` a potom vieme vo funkciach pouzit :

```
this.$refs.VLASTNY_NAZOV.value = ''
this.$refs.VLASTNY_NAZOV.focus()
```

KAPITOLA 7

Online marketing

KAPITOLA 8

Web browser games

KAPITOLA 9

Node.JS

KAPITOLA 10

Vyvoj e-shopu

KAPITOLA 11

October CMS

KAPITOLA 12

Codeigniter

KAPITOLA 13

Ruby on Rails

KAPITOLA 14

Mopje projekty

KAPITOLA 15

Windows

KAPITOLA 16

Apple swift

KAPITOLA 17

Smart Home

Index