Převod PHP aplikace na projekt ve framework Laravel

Petr Nosek

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr Nosek**
Osobní číslo:	**A21344**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Převod PHP aplikace na projekt ve frameworku Laravel**
Téma práce anglicky:	**Converting a PHP Application into a Laravel Framework Project**

Zásady pro vypracování

1. Seznamte se s vývojovým frameworkem Laravel a popište jeho strukturu a základní nástroje.
2. Popište stávající webovou aplikaci navrženou pro organizaci výuky ve školách z hlediska funkcionality, architektury a kódu.
3. Navrhněte proces převodu aplikace z čistého PHP na projekt vytvořený pomocí frameworku Laravel.
4. V rámci praktické části převeďte stávající aplikaci z čistého PHP na projekt vytvořený pomocí frameworku Laravel.
5. Zhodnoťte proces převodu a výsledný kód z hlediska přehlednosti. Věnujte se také zabezpečení webové aplikace.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. STAUFFER, Matt. Laravel: Up & Running. Third edition. O'Reilly Media, 2023. ISBN 978-109-8153-229.
2. LOCKHART, Josh. Modern PHP, New Features and Good Practices. O'Reilly Media, 2015. ISBN 978-149-1904-992.
3. ZANDSTRA, Matt. PHP 8 objects, patterns, and practice: mastering OO enhancements, design patterns, and essential development tools. Sixth edition. New York, NY, U.S.A.: Apress, 2021. ISBN 978-148-4267-905.
4. Otwell, T.: Oficiální dokumentace frameworku Laravel [online]. 2011–2023 [cit. 2023-11-05]. Dostupné z: https://www.laravel.com/docs
5. PHP: PHP Manual [online]. 2001–2023 [cit. 2023-11-05]. Dostupné z: https://www.php.net/manual/en/
6. Laravel News [online]. 2012–2023 [cit. 2023-11-05]. Dostupné z: https://laravel-news.com
7. Way, J.: Webový portál Laracast [online]. [cit. 2013-11-05]. Dostupné z: https://laracasts.com

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.
- že při tvorbě této práce jsem použil/a nástroj generativní model AI ChatGPT; chatgpt.com za účelem zlepšení formulace textu a generativní model AI Gemini; gemini.google.com za účelem vyhledávání informačních zdrojů. Po použití těchto nástrojů jsem provedl/a kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne 10. května 2024 Petr Nosek, v. r.

podpis studenta

ABSTRAKT

Tato bakalářská práce se zabývá převodem stávající webové aplikace pro organizaci výuky ve školách z čistého PHP na moderní vývojový framework Laravel. Teoretická část popisuje strukturu a základní nástroje frameworku Laravel. Praktická část se věnuje popisu původní aplikace včetně funkcionalit, architektury a struktury kódu. Práce se také zaměřuje na návrh architektury nové aplikace a výběr technologií. Součástí práce je také popis implementace nového projektu zaměřený na nastavení vývojového prostředí, tvorbu datových modelů, migrací, kontrolérů a logiky aplikace. Práce dále zhodnotí tento proces převodu z hlediska přehlednosti a udržitelnosti nového kódu, a také popíše zabezpečení převedené aplikace.

Klíčová slova: PHP, Laravel, framework, převod PHP aplikace, webová aplikace

ABSTRACT

This bachelor thesis deals with the conversion of an existing web application for the organization of teaching in schools from pure PHP to the modern development framework Laravel. The theoretical part describes the structure and essential tools of the Laravel framework. The practical part describes the original application, including functionalities, architecture, and code structure. The work also focuses on the design of the architecture of the new application and the selection of technologies. The thesis also includes a description of the implementation of the new project, which is focused on setting up the development environment, creating data models and migrations, and developing controllers and application logic. The thesis also evaluates this conversion process in terms of the new code's clarity and maintainability and describes the converted application's security.

Keywords: PHP, Laravel, framework, conversion of PHP application, web applications

Rád bych poděkoval vedoucímu své práce Ing. Radku Valovi, Ph.D., za cenné rady a vstřícnost při konzultacích mé bakalářské práce. Mé poděkování patří též mé rodině za podporu během celého studia.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V posledních letech dochází k rychlému vývoji a rozšiřování technologií v oblasti vývoje webových aplikací, kdy při tvorbě těchto aplikací se více využívají vývojové frameworky, které usnadňují vývoj. Tyto frameworky poskytují širokou škálu nástrojů a funkcí, které přispívají k rychlejším vývoji, udržitelnosti a rozšiřitelnosti kódu. Laravel, jakožto jeden z PHP frameworků, je příkladem frameworku, který umožňuje vytvářet komplexní webové aplikace s využitím implementovaných funkcí v Laravelu.

Hlavním cílem této bakalářské práce je převést webovou aplikaci, která původně nevyužívala žádný PHP framework, na projekt ve frameworku Laravel. Původní aplikace, zaměřená na organizaci výuky ve školách, umožňuje uživatelům vytvářet kurzy, ve kterých mohou vytvářet testy, zadávat úkoly a sdílet studijní materiály se studenty. Kvůli složité orientaci v kódu a komplikované implementaci nových funkcí je vhodné převést tuto aplikaci na modernější a strukturovanější projekt s využitím frameworku Laravel. Převedená verze aplikace bude nabízet stejné funkce jako původní aplikace.

Úvod teoretické části této práce se bude věnovat obecným návrhovým vzorům, jejich struktuře a rozdělní, pro pochopení principu návrhových vzorů. Hlavní obsah teoretické části se bude věnovat frameworku Laravel, jeho architektuře a základním nástrojům, které lze při vývoji webových aplikací využít. Pozornost bude také věnována architektonickému vzoru MVC, který je klíčový pro tvorbu aplikací nejen ve frameworku Laravel. Dále bude popsána adresářová struktura Laravelu a jeho další nástroje jako jsou Eloquent ORM, Artisan, Tinker, routování a Blade šablonovací systém. Tato část poskytne teoretický základ potřebný pro následný převod původní aplikace na projekt ve framework Laravel.

Praktická část popíše stávající webovou aplikaci včetně použitých technologií, přehledu funkcí, struktury databáze a kódu. Před samotnou tvorbou nové verze aplikace se praktická část věnuje plánováním převodu aplikace na Laravel, což zahrnuje definici cílů, výběr nástrojů, návrh nové architektury a databáze aplikace. Kapitola o implementaci nového projektu popíše konfiguraci vývojového prostředí, převod databáze, tvorbu modelů a kontrolérů, nastavení rout a vytváření uživatelského prostředí. Závěrečná část je věnována srovnání a hodnocení obou verzí aplikace z hlediska zabezpečení, rozšiřitelnosti a udržitelnosti kódu.

I. TEORETICKÁ ČÁST

1 OBECNÉ NÁVRHOVÉ VZORY

Návrhové vzory, které řeší problémy při tvorbě softwaru, slouží jako šablony pro vyřešení opakujících se problémů. Tyto vzory nelze přímo použít jako hotové funkce nebo knihovny, ale slouží jako řešení pro konkrétní problém kódu, které lze upravit a následně aplikovat na kód projektu. [1] Návrhové vzory představují plán pro vyřešení opakujících se problémů, jako je složitá interakce mezi objekty, který lze opakovaně použít. [2]

Na rozdíl od algoritmů, které představují definici akcí pro dosažení určitého cíle, návrhové vzory poskytují spíše popis pro vyřešení specifického problému, kdy ve výsledku mohou být aplikovány různým způsobem. [1] Kromě algoritmů se mohou návrhové vzory plést s architekturou aplikace. Návrhové vzory řeší konkrétní problémy, které nejsou zaměřené na celkovou strukturu systému, kdežto architektura samotné aplikace se zabývá strukturou celého projektu. [3]

1.1 Struktura návrhového vzoru

Návrhové vzory se skládají ze čtyř částí, které se obvykle vyskytují v popisu vzoru. [1]

1.1.1 Název vzoru

Název návrhového vzoru by měl stručně vystihovat problém, který vzor řeší, jeho řešení a vyplývající důsledky. Zároveň název vzoru by neměl být delší jak dvě slova. Například, u návrhového vzoru *Mediator* název naznačuje, že jeho účelem je vytvořit objekt sloužící jako prostředník pro zjednodušení složité komunikace mezi objekty. [2]

1.1.2 Problém

Problém, který vzor řeší, určuje jeho použitelnost, jelikož poskytuje kontext pro použití vzoru podobně jako sada podmínek, za kterých lze vzor použít. Tyto podmínky specifikují vhodnost vzoru pro řešení existujícího problému. [3]

1.1.3 Řešení

Řešení popisuje prvky, které tvoří konkrétní vzor, jako jsou jejich vztahy a spolupráce. Vzor tedy není konkrétním postupem, ale spíše šablonou, kterou lze v určitých situacích využít. Tato šablona popisuje to, jak by měly být prvky uspořádány pro vyřešení daného problému. [3]

1.1.4 Důsledky

Důsledky zahrnují popis jeho výhod a nevýhod. Zároveň slouží pro zvážení alternativních návrhů a přínosů použití vzoru. Důsledky daného vzoru mohou ovlivnit klíčové atributy systému jako je např. rozšiřitelnost a přenositelnost. [2] [3]

1.2 Rozdělení

Návrhové vzory se dělí do tří kategorií: Creational, Structural a Behavioral Patterns.

1.2.1 Creational Patterns

Tento vzor popisuje tvorbu objektů a vytváření tříd pro snížení složitosti návrhu. [2] Vzory při tvorbě tříd využívají dědičnost, zatímco vzory pro tvorbu objektů používají delegaci. [4] Zároveň je jeho úkolem opakovaně využít stávající kód. [3]

Do Creational Patterns lze zařadit tyto vzory [4]:

- Abstract Factory
- Builder
- Factory Method
- Object pool
- Prototype
- Singleton

1.2.2 Structural Patterns

Strukturální návrhové vzory popisují uspořádání tříd a objektů. [3] Vzory pro vytváření strukturálních tříd využívají dědičnost, zatímco vzory pro objekty definují způsob, jak uspořádat jednotlivé objekty za účelem rozšířit jejich funkce. [4] Strukturální návrhové vzory také usnadňují návrh a umožňují tvořit vztahy mezi různými komponentami systému. [2]

Do této skupiny lze zařadit [4]:

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight

- Private Class Data
- Proxy

1.2.3 Behavioral Patterns

Bahavioral Patterns se primárně zaměřují na komunikaci mezi objekty v rámci tříd. [3] Tyto vzory jsou zvláště důležité pro usnadnění vzájemné interakce mezi objekty [2]. Vzory popisují, jak mají objekty spolupracovat na zadaném úkolu, protože samotné objekty by tento úkol samy neprovedly. [3]

Do Behavioral Patterns patří např. [4]:

- Command
- Null Object
- Obeserve
- Interpreter
- State

2 LARAVEL FRAMEWORK

Tato kapitola se zabývá frameworkem Laravel, jeho historií, architekturou a základními vlastnostmi.

2.1 Framework

Frameworky, v kontextu softwarového inženýrství, představují soubory abstraktních nástrojů a komponent, které slouží k usnadnění a urychlení vývoje softwaru. Poskytují základní strukturu, knihovny, nástroje a funkce, na kterých vývojáři mohou stavět a rozvíjet vlastní aplikace. [5]

PHP frameworky se využívají pro usnadnění vytváření webových aplikací. Mnoho z nich již obsahují knihovnu pro otestování vytvořené webové aplikace. [5]

2.2 Základní informace Laravel Framework

Laravel je jeden z populárních PHP frameworků pro vývoj webových aplikací a od svého vzniku se Laravel vyvinul v nástroj, který nabízí vývojářům komplexní sadu funkcí pro zjednodušení a zrychlení procesu vývoje webových aplikací. Laravel je navržen s důrazem na jednoduchou a přehlednou syntaxi a poskytuje podporu MVC architektury, což je přístup, který odděluje logiku aplikace od uživatelského rozhraní, a tím usnadňuje správu a údržbu kódu. [6] [8]

Klíčovými rysy Laravelu je integrovaný ORM systém Eloquent, který umožňuje vývojářům pracovat s databázemi pomocí PHP kódu a Blade šablonovací engine, který podporuje tvorbu dynamických HTML šablon s PHP. [6]

2.3 Historie a vývoj Laravel Framework

Laravel, PHP framework zaměřený na vývoj webových aplikací, vyvinul Taylor Otwell a poprvé byl představen veřejnosti v roce 2011. Laravel vznikl jako náhrada za framework PHP CodeIgniter, který nenabízel autentifikaci a autorizaci uživatelů. [5] Cílem frameworku Laravel bylo poskytnout vývojářům soubor nástrojů pro rychlejší a efektivnější vývoj webových aplikací a od roku 2011 se Laravel stále vyvíjí a prošel mnoha změnami, díky čemuž se stal jedním z oblíbených PHP frameworků mezi vývojáři. [7]

Tabulka 1. Historie verzí frameworku Laravel [8]

verze	měsíc a rok vydání
Laravel 1	červen 2011
Laravel 2	září 2011
Laravel 3	únor 2012
Laravel 4	květen 2013
Laravel 5	únor 2015
Laravel 6	září 2019
Laravel 7	březen 2020
Laravel 8	září 2020
Laravel 9	únor 2022
Laravel 10	únor 2023
Laravel 11	březen 2024

2.4 Architektura Laravel Framework

Tato kapitola se zaměřuje na architekturu frameworku Laravel, popisuje MVC architekturu, adresářovou strukturu projektu a základní nástroje a balíčky, které tento framework poskytuje.

2.4.1 MVC architektura

Laravel Framework využívá velmi oblíbený architektonický vzor MVC, který je široce využívaným vzorem. Tento vzor je založen na principu oddělení aplikace na tři základní komponenty: model, view (pohledy) a controller (kontroléry), což umožňuje oddělit logickou část od uživatelského prostředí. Cílem MVC architektury je dosáhnout situace, kdy logika aplikace a výstup uživateli jsou striktně oddělené. [9] [10]

Tento vzor je klíčovým prvkem mnoha webových frameworků, včetně Laravelu, Nette, Zendu či ASP.NET. Pro využití zmíněných frameworků je nezbytně důležité pochopit tento architektonický vzor. [10]

2.4.1.1 Model

Model představuje datovou a logickou vrstvu aplikace, kde jsou definována data a pravidla pro jejich manipulaci. To zahrnuje realizaci databázových dotazů, provádění výpočtů, validaci dat a další operace související s daty. Model tedy funguje jako zásadní prostředník mezi aplikací a její databází, umožňující efektivní správu a manipulaci s daty. [9] [10]

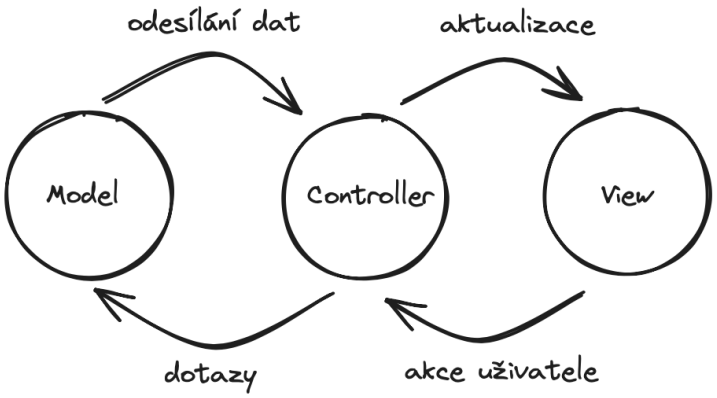
V rámci Laravelu umožňuje nástroj Eloquent ORM intuitivně pracovat s databází v rámci modelu. Díky tomu lze snadno provádět operace CRUD, ale také vytvářet vztahy mezi modely. [9]

2.4.1.2 View

View představuje uživatelské rozhraní aplikace. Jejím primárním cílem je prezentování dat uživateli, která jsou poskytnuta z kontroléru, a zároveň slouží jako prostředník pro získání uživatelských vstupů. Tato vrstva je charakteristická svým omezeným využitím logiky, což je omezeno pouze na nezbytné prvky nutné pro zobrazení výstupů. Zároveň podporuje podmínky a cykly. [9]

2.4.1.3 Controller

Kontrolér plní zásadní roli prostředníka mezi modelovou a prezentační vrstvou aplikace. Jeho hlavním úkolem je přijímat vstupy od uživatelů, zpracovat je a následně předat zpracovaná data příslušné vrstvě *view*. Tímto způsobem kontrolér udržuje logiku aplikace oddělenou od uživatelského rozhraní. [9]



Obrázek 1 – MVC architektura [9]

2.4.2 Jádro frameworku Laravel

Jádro frameworku Laravel tvoří sada komponent, které usnadňují vývoj webových aplikací. Mezi hlavní komponenty tohoto frameworku patří routování, dependency injection, systém pro spouštění událostí a reakci na ně pomocí listenerů, nástroj pro tvorbu a spouštění příkazů z příkazové řádky a především systém pro tvorbu dynamických HTML stránek pomocí šablonovacího systému Blade. [12]

2.4.3 Adresářová struktura

Po otevření adresáře projektu Laravel jsou viditelné následující soubory a adresáře [11]:

- app/
- bootstrap/
- config/
- database
- public/
- resources/
- routes/
- storage/
- tests/
- vendor/
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- readme.md
- server.php
- webpack.mix.js

2.4.3.1 Složky

Tato kapitola popise jednotlivé složky v projektu Laravel a vysvětluje jejich význam. [11] [12]

- *App* – Adresář obsahuje základní kód aplikace společně s modely a kontroléry.
- *Bootstrap* – Obsahuje soubory, které zavádí framework Laravel při každém spuštění.
- *Config* – Místo, kde jsou uloženy všechny konfigurační soubory.
- *Database* – V tomto adresáři se nachází všechny migrace databáze, seedy a factories modelů. Adresář lze využít pro použití SQLite.
- *Public* – V tomto adresáři se nachází soubor index.php, což je klíčový soubor pro přesměrování všech požadavků. Kromě toho jsou zde veškeré veřejně dostupné soubory, včetně obrázků, javascriptových souborů a stylů CSS.
- *Resources* – Složka, kde jsou umístěny především pohledy (views), soubory pro překlady a nezkompilované soubory JavaScript nebo stylů CSS.
- *Routes* – Místo, kde se nachází všechny definice cest (routy) aplikace, a to jak pro trasy HTTP, tak pro konzolové příkazy a příkazy Artisan.
- *Storage* – Místo, kde jsou uloženy caches, logs a zkompilované systémové soubory. Adresář je rozdělen na tři podsložky: *app*, *framework* a *logs*. Do adresáře *app* lze uložit soubory, která vygenerovala webová aplikace. V adresáři *framework* jsou uloženy soubory vygenerované frameworkem a mezipamětí, zatímco složka *logs* obsahuje záznamy o činnosti aplikace.
- *Tests* – Obsahuje všechny jednotkové a integrační testy.
- *Vendor* – V této složce jsou umístěny všechny balíčky, které Composer instaluje.

2.4.3.2 Další soubory v adresáři

- *.editorconfig* – Poskytuje vývojovému prostředí informace ohledně standardů pro kódování Laravel.
- *.env* – Obsahuje konfigurační proměnné nezbytné pro běh aplikace. Tento soubor umožňuje definovat proměnné, které lze snadno upravovat bez nutnosti zasahovat do zdrojového kódu, což zajišťuje flexibilitu při nasazování aplikace v různých prostředích, jako jsou vývoj, testování a produkce.
- *.env.example* – Jedná se o šablonu, kterou lze použít pro vlastní soubor .env.
- *artisan* – Umožňuje spouštět příkazy *Artisan* z příkazového řádku.

- *composer.json a composer.lock* – Konfigurační soubory pro Composer. Composer.json lze upravovat uživatelem, naopak composer.lock upravovat uživatelem nelze.
- package.json – Obsahuje seznam JavaScript knihoven, které má NPM stáhnout.
- *webpack.mix.js* – Soubor je volitelný konfigurační soubor pro Laravel Mix (nástroj sloužící k zjednodušení práce s frontendovými zdroji). [11] [12]

2.5 Základní nástroje a balíčky

Následující kapitola se zabývá klíčovými nástroji a balíčky, které Laravel nabízí pro vývoj webových aplikací.

2.5.1 Eloquent ORM

Eloquent ORM usnadňuje práci s databází v Laravel framework. [12] Každá tabulka v databázi se váže s konkrétním modelem, který umožňuje provádět databázové operace, jako je načítání, vkládání, aktualizace nebo mazání záznamů, aniž by bylo nutné psát SQL kód. [13]

Eloquent ORM je založen na principu *ActiveRecord*, což znamená, že poskytuje jednotné rozhraní pro práci s různými typy databází. Díky tomuto přístupu každá třída Eloquentu nejenže umožňuje interakci s celou databázovou tabulkou (například pomocí metody *User::all(),* která vrátí všechny uživatele), ale také reprezentuje řádek v tabulce, což umožňuje každé instanci třídy spravovat vlastní data. To znamená, že pro uložení nebo smazání záznamu v databázi lze jednoduše použít metody jako *$user->save()* nebo *$user->delete().* [12]

2.5.1.1 Generování tříd modelů

Pro vytvoření nového modelu slouží příkaz Artisan: *php artisan generate:model <nazev>*. Ve složce *app/models* se vytvoří odpovídající model. [13]

2.5.1.2 Názvy tabulek

Při prvním pohledu na Eloquent ORM je možné si všimnout, že není nutné specifikovat, která databázová tabulka odpovídá danému modelu. Laravel využívá standart, podle kterého se pro název tabulky automaticky použije množné číslo názvu třídy modelu ve „snake_case" formátu, pokud není název tabulky určen jinak. To znamená, že pro model *Flight* bude Eloquent předpokládat, že záznamy jsou ukládány v tabulce *flights*. [12] [13]

2.5.1.3 Primární klíč

Eloquent ORM předpokládá, že každá tabulka modelu v databázi má sloupec primárního klíče s názvem *id*. [13]

2.5.2 Artisan

Artisan představuje integrované rozhraní příkazové řádky (CLI) frameworku Laravel, které se nachází v kořenovém adresáři aplikace pod skriptem *artisan*. Toto rozhraní je vybaveno širokou škálou příkazů, které usnadňují proces vývoje webových aplikací. [14] Díky Artisanu mohou vývojáři snadno generovat základní struktury různých komponent aplikace, spravovat databázové migrace, udržovat stav aplikace a řídit frontu úkolů. [12]

2.5.2.1 Příkazy

Pro zobrazení seznamu dostupných Artisan příkazů se používá příkaz *php artisan list*, což umožní uživatelům získat přehled o všech nástrojích, které mohou v rámci svého Laravel projektu využít. Každý příkaz navíc obsahuje podrobnou nápovědu, jež je přístupná pomocí *help* následovaného jménem příkazu. Tento postup poskytuje užitečné informace o použití konkrétního příkazu, jeho parametrech a příkladech použití. [14]

Jeden z příkladů je příkaz *php artisan migrate* k inicializaci nebo aktualizaci struktury databáze na základě databázových migrací, které jsou definovány v aplikaci. [15] Další Artisan příkaz je *php artisan make:model* pro vytvoření nového modelu. Například, pro vytvoření modelu *User* lze využít příkaz *php artisan make:model User.* Tento příkaz nejenže vytvoří novou třídu modelu v příslušném adresáři aplikace, ale současně vytvoří migraci pro nový model pomocí parametru *--migration* nebo zkráceně *-m*. [13]

2.5.3 Tinker

Tinker, nástroj příkazového řádku, umožňuje uživatelům interagovat s databází, tvořit objekty nebo vkládat data. K jeho spuštění se využívá příkaz *php artisan tinker*. [16]

2.5.4 Routování

V rámci struktury projektu Laravel je zásadní definovat webové routy pro aplikaci v adresáři *routes*. Tímto způsobem můžeme specifikovat URL požadavky a jejich mapování na odpovídající kontroléry a metody. Laravel poskytuje podporu pro širokou škálu HTTP metod, jako jsou GET, POST, PUT, DELETE a další. [18]

Pro definici rout specifických pro webovou část aplikace se typicky využívá soubor *web.php*, zatímco routy určené pro API jsou definovány v souboru *api.php*. [17]

2.5.4.1 Základní routování

Nezákladnější routy v Laravelu se vyznačují přijímáním URI spolu s anonymní funkcí. Tímto způsobem lze definovat cesty bez nutnosti složitých konfiguračních souborů pro routování. [18]

```
Route::get('/', function () {
    return 'Hello, World!';
});
```

2.5.4.2 Propojení routy s kontrolérem

Propojení routy s kontrolérem se provádí uvedením názvu kontrolérů a metody, která má být zavolána jako akce routy.

Příklad propojení routy s kontrolérem v Laravelu by mohl vypadat takto:

```
Route::get('/user', [UserController::class, 'index']);
```

V tomto příkladě *Route::get* definuje routu pro HTTP GET požadavek na cestu */users*, přičemž *[UserController::class, 'index']* určuje, že požadavek má být zpracován metodou *index* ve třídě *UserController*. [19]

2.5.4.3 Parametry routy

Parametry routy umožňují dynamickou manipulaci s částmi URL cesty. Tyto parametry, často označované jako proměnné segmenty URL, reagují na uživatelské požadavky a zpracovávají data přímo získaná z URL. Příkladem může být potřeba získání uživatelského ID z URL. [18]

```
Route::get('/user/{id}', function (string $id){...}
```

Občas je nutné zadat nepovinný parametr routy. Stačí za název parametru doplnit *?*. [18]

2.5.4.4 Pojmenování routy

Pojmenování routy je možné realizovat přidáním metody *name()* k definici routy, což umožňuje její jednoduché volání a odkazování v rámci celé aplikace, přičemž název routy musí být unikátní. [18]

2.5.5 Blade šablonovací systém

Blade šablonovací systém představuje nástroj pro vytváření uživatelských rozhraní. Na rozdíl od některých šablonovacích enginů, Blade neomezuje vývojáře ve využívání nativního PHP kódu v šablonách, což umožňuje větší flexibilitu při vytváření uživatelského rozhraní. Soubory šablon Blade mají příponu *.blade.php* a jsou obvykle umístěny v adresáři *resources/views*. [20]

Blade nabízí možnost vracet pohledy přímo z rout nebo kontrolérů pomocí globálního helpera *view*. Také lze do pohledu předat data pomocí druhého argumentu helpera *view*, což umožňuje dynamické generování obsahu stránky. V neposlední řadě Blade podporuje dědičnost šablon, snadnou rozšiřitelnost a umožňuje rozdělit šablony do opakovaně použitelných sekcí. [20] [12]

2.5.5.1 Vypisování dat

Data, která jsou poslána do šablony Blade, lze zobrazit obalením proměnné ve složených závorkách. [20]

```
Hello, {{ $name }}
```

Blade také umožňuje zobrazit výsledky libovolné PHP funkce. [20]

```
{{ time() }}
```

2.5.5.2 Podmínky

Podmínky v šablonovacím systému Blade lze vytvářet pomocí direktiv *@if*, *@elseif*, *@else* a *@endif*. Podmínky fungují stejně jako v jazyce PHP.

Oproti jazyku PHP poskytuje Blade syntaxy *@unless*, která představuje v PHP následující kód: *<?php if (! $condition)*. Syntaxe *@unless* musí být ukončena *@endunless*.

Kromě toho lze také využívat direktivy jako *@isset* nebo *@empty*. [20]

2.5.5.3 Cykly

Kromě podmínek nabízí šablonovací systém Blade také jednoduché direktivy pro tvorbu cyklů. Tyto direktivy nabízejí stejnou funkčnost jako v nativním PHP, ale s tím rozdílem, že zjednodušují syntaxi při implementaci. [20]

```
@for ($i = 0; $i < 10; $i++)
    Aktuální hodnota je {{ $i }}
@endfor

@foreach ($users as $user)
    <p>Toto je uživatel {{ $user->id }}</p>
@endforeach
```

2.5.6 Migrace

Migrace v Laravel frameworku fungují jako systém verzování databáze. Umožňují vývojářským týmům spravovat a sdílet strukturu databáze webové aplikace. Pomocí migrací není potřeba ručního vytváření a úpravy databázových tabulek, což je výhoda hlavně u složitějších aplikací, kdy ruční úpravy by byly časově náročné a vedly k chybám. Migrace umožňují definovat změny struktury databáze v samostatných souborech migrací, které jsou umístěny ve složce *database/migrations*. Po jejich spuštění Laravel automaticky provede jednotlivé migrace v pořadí podle data vytvoření. Novou migraci lze jednoduše vytvořit pomocí Artisan příkazu *make:migration*. [15]

Každý soubor migrace obvykle obsahuje metody *up()* a *down()*, které definují, jaké operace se mají provést při spuštění migrace. Metoda *up()* se využívá pro vytvoření nových tabulek a sloupců v databázi, zatímco metoda *down()* by měla vrátit tuto akci zpět. [12]

2.5.6.1 Spuštění migrací

Pro spuštění všech migrací použijeme příkaz Artisan *migrate: php artisan migrate*. [15]

2.5.6.2 Schema Builder

Migrace ve své podstatě využívají Schema Builder, Laravel třídu poskytující soubor metod pro definici a modifikaci databázových tabulek a sloupců. Výhodou Schema Builderu je, že pracuje se všemi Laravelem podporovanými databázemi. [15] [21]

Pro vytvoření nové tabulky využijeme metodu *create*, která vyžaduje dva argumenty: první je název tabulky a druhý je *closure*, tedy objekt *Blueprint*. Kód pro vytvoření tabulky by mohl vypadat následovně: *Schema::create('users', function (Blueprint $table) {...}*. [15]

Pro modifikaci existujících databázových tabulek se využívá metoda *table* namísto *create*. K přejmenování tabulek slouží metoda *rename*, zatímco pro jejich odstranění se používá *drop* nebo *dropIfExists* (pokud chceme tabulku odstranit pouze v případě, že již existuje). [15]

2.5.7 Seedy

Laravel Framework poskytuje možnost vyplnit tabulky testovacími daty. V projektu aplikace Laravel se nachází složka *database/seeders* s třídou *DatabaseSeeder*. Tato třída standardně obsahuje pouze metodu *run*. Tato metoda je spuštěna při vykonání příkazu Artisan *db:seed*. V rámci metody *run* lze do databáze vkládat data následujícím způsobem: [22]

```
DB::table('posts')->insert([
    'user_id' => 1, // ID uživatele, který příspěvek vytváří
    'course_id' => 1, // ID kurzu, ke kterému příspěvek patří
    'text' => Str::random(50), // Náhodný text příspěvku
]);
```

2.5.8 Factories

Ruční zadávání dat pro každý model je časově náročné, a proto můžeme využít factories pro generování velkého počtu záznamů. [23] Tyto factories se nacházejí ve složce *database/factories* a umožňují definovat výchozí hodnoty atributů pro modely Eloquent. Pro generování reálných dat, jako jsou emaily, jména, příjmení, telefonní čísla nebo různé texty, využívají factories knihovnu *Faker*. [24]

2.6 Rozšíření Laravelu pomocí balíčků

Následující kapitola se věnuje rozšiřovacím balíčkům pro Laravel Framework. Laravel umožňuje vývojářům snadno implementovat různé balíčky, které rozšiřují základní možnosti frameworku nebo přidávají zcela nové funkce.

2.6.1 Laravel Breeze

Laravel Breeze je balíček poskytující základní implementaci pro autentizaci. Laravel Breeze umožňuje rychlé nasazení kompletního systému pro přihlášení a registraci uživatelů, a kromě přihlášení a registrace nabízí Laravel Breeze obnovu hesla, ověřování uživatelů pomocí emailu nebo správu uživatelského profilu. [25] Balíček obsahuje předpřipravené kontroléry a šablony, díky čemuž je možné rychle implementovat autentizační procesy do Laravel aplikace. [27] Podporuje šablony Blade, ale i moderní JavaScriptové frameworky jako Vue nebo React. [26] V základu používá Laravel Breeze CSS framework Tailwind CSS. Součástí Laravel Breeze je i sada testů pro ověření základních funkcionalit, které zjednodušují testování autentizačních procesů. [25]

2.6.1.1 Instalace

Balíček Breeze se instaluje přes composer pomocí příkazu: *composer require laravel/breeze –dev*. Pro dokončení instalace je nutné použít příkaz Artisan: *php artisan breeze:install*. V neposlední řadě je nutné zaktualizovat migrace webového projektu. [25]

2.6.1.2 Struktura

Všechny související routy, kontroléry a šablony jsou umístěny v kořenových složkách Laravel projektu. Specificky se nové routy nacházejí v souboru *routes/auth.php*, které se odlišují od tradičního způsobu registrace rout v Laravelu, neboť jsou tyto routy přímo začleněny do *web.php*. Kontroléry pro autentizaci jsou umístěny v *app/Http/Controllers/Auth/*, zatímco šablony pro Laravel Breeze jsou k dispozici v *resources/views/auth/.* [27]

2.6.1.3 Další autentizační balíčky

Pro Laravel Framework existují další autentizační balíčky. Jedním z nich je Laravel Jetstream, který nabízí komplexnější funkcionalitu oproti Laravel Breeze. Kromě přihlášení a registrace nabízí Laravel Jetstream dvoufázové ověřování, správu relací nebo např. volitelnou správu týmů. [26] Podobně jako Laravel Breeze, Laravel Jetsream využívá pro uživatelské rozhraní CSS framework Tailwind CSS. [25]

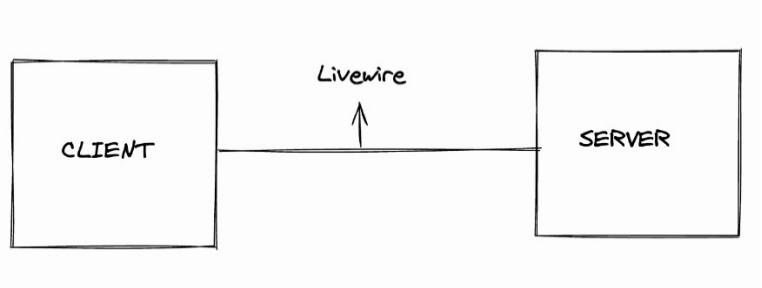
Dalším balíček je Laravel Fortify, který je vhodný pro tvorbu vlastního autentizačního systému a oproti ostatním balíčků nenabízí předpřipravené uživatelské rozhraní. [26]

2.6.2 Livewire

Livewire je framework pro Laravel aplikace, který umožňuje vývojářům vytvářet reaktivní uživatelská rozhraní bez potřeby psaní JavaScriptu. Vývojáři mohou využít Laravel framework a šablon Blade k vytvoření dynamických uživatelských rozhraní. S Livewire je možné reagovat na akce uživatelů, jako je odeslání formulářů, scrollování nebo klikání na tlačítka, a to vše bez nutnosti obnovovat stránku. [28]

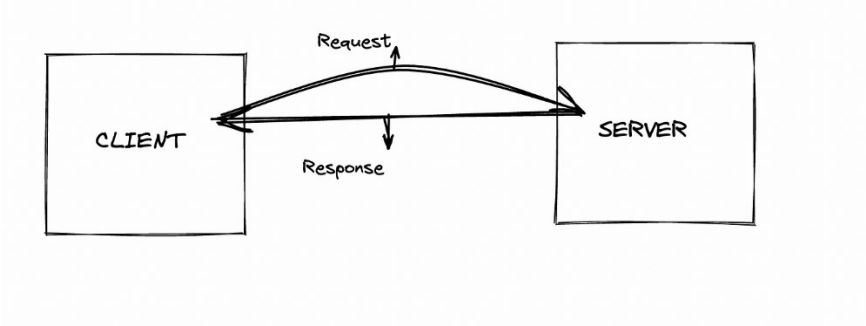
2.6.2.1 Princip fungování

Základní princip Livewire spočívá vtom, že funguje jako most mezi klientem a serverem, který umožňuje komunikaci od klienta k serveru a naopak. [28]



Obrázek 2 – Fungování Livewire [28]

Po prvním načtení stránky, které obsahuje komponentu Livewire, Livewire naváže na své komponenty javascriptové posluchače událostí, které sledují každou uživatelskou interakci. Každá akce je odeslána na server formou asynchronního API požadavku, přičemž součástí tohoto požadavku je stav komponenty nebo její obsah. Server následně akci zpracuje, vytvoří aktualizovanou šablonu a nový stav komponenty, které jsou poté zpětně poslány klientovi. Pomocí Javascriptu je poté na straně klienta aktualizována specifická část stránky, aniž by došlo k její kompletní obnově. [28]



Obrázek 3 – Cyklus požadavku a odpovědi mezi serverem a klientem [28]

2.6.2.2 Základní funkce

Základní funkce Livewire spočívající v několika klíčových funkcionalitách:

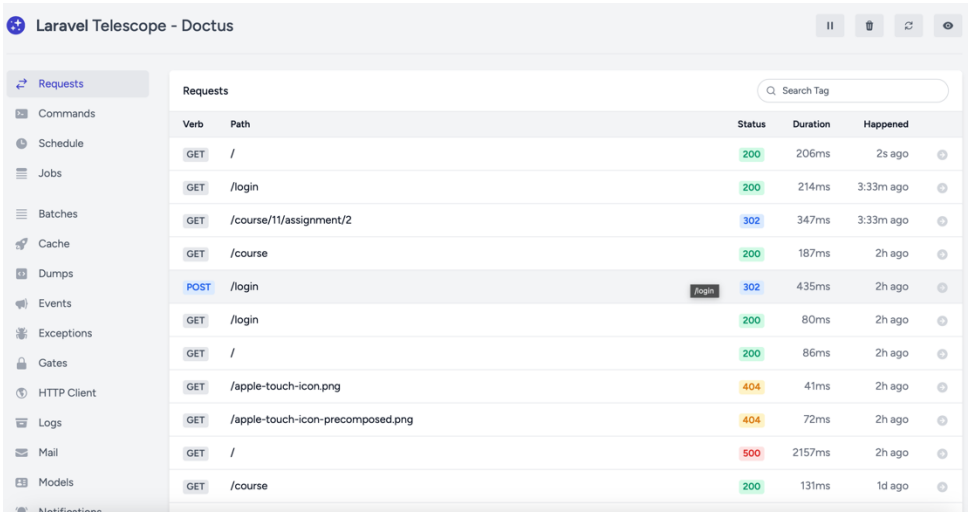
- *Syntaxe* – Livewire nabízí syntaxi pro tvorbu UI komponent s využitím šablonovacího systému Blade. Díky tomu mohou vývojáři pokračovat v psaní kódu pro uživatelské prostředí bez nutnosti se učit novou syntaxi.
- *Aktualizace v reálném čase* – Livewire umožňuje aktualizaci UI komponent v reálném čase bez nutnosti psaní JavaScriptu.
- *Obousměrná vazba dat* – Livewire nabízí dvoucestné vazby dat. Znamená to, že vývojáři mohou snadno synchronizovat data mezi frontendovými a backendovými komponentami.

- *Zpracování formulářů* – Livewire zjednodušuje práci s formuláři tím, že poskytuje vestavěnou validaci formulářů a zpracování chyb. Livewire umožňuje definovat pravidla pro validaci vstupů a následně zobrazit chybové hlášky.
- *Zpracování událostí* – Díky Livewire je možné řešit události vyvolané uživatelem přímo v backendovém kódu, což usnadňuje implementaci reakcí na uživatelské události. Není potřeba definovat události pomocí Javascriptu. [29]

2.6.3 Laravel Telescope

Laravel Telescope představuje ladící nástroj, který nabízí detailní přehled o požadavcích, které přichází do webové aplikace, databázových dotazech, úlohách ve frontě, zachycených výjimkách, odeslaných e-mailech, proměnných a mnoho dalších. Zkrátka Laravel Telescope poskytuje informace o tom, co se děje ve webové aplikaci. [30]

Díky poskytnutí přehledných informací o všech klíčových aspektech webové aplikace, umožňuje Telescope rychlejší ladění, a tím urychluje proces samotného vývoje. Tento nástroj je vhodný jak pro použití během vývoje, tak pro monitorování aplikace v produkčním prostředí. [31]



Obrázek 4 – Prostředí nástroje Laravel Telescope

2.6.3.1 Základní využití

Telescope nabízí širokou škálu funkcí. K těm nejzákladnějším a nejužitečnějším patří následující: [31]

- *Monitorování požadavků* – Lze sledovat příchozí požadavky a jejich odpovědi, což může být užitečné při řešení problémů s cestami (routy).

- *Ladění dotazů* – Zobrazuje detailní přehledy o databázových dotazech, včetně informací o jejich délce a výkonnosti, s podrobnými údaji o jednotlivých dotazech a jejich kontextu.
- *Sledování úloh, e-mailů a událostí* – Umožňuje pozorování úloh ve frontě, odesílaných e-mailů a událostí, poskytuje podrobné informace o těchto procesech, včetně dat o vstupu, výstupu, stavu a selhaných úkolech.
- *Zpracování výjimek* – Přehled o všech výjimkách, které se v aplikaci vyskytují.
- *Optimalizace výkonu* – Profilovací nástroj Telescope pomáhá identifikovat možná slabá místa v aplikaci, které mohou pomoct k jejich optimalizaci a zlepšení celkového výkonu.

2.6.3.2 Instalace a spuštění

Pro nainstalování ladícího nástroje Laravel Telescope využijeme nástroj pro správu knihoven *Composer*: *composer require laravel/telescope.* Následně je nutné aplikovat nástroj Telescope pomocí příkazu *Artisan*: *php artisan telescope:install*. V neposlední řadě je důležité aktualizovat migrace pomocí příkazu *Artisan*. [30]

K přístupu k Laravel Telescope se dostanete navštívením cesty */telescope*. Zde se nachází všechny zmíněné informace o webové aplikace. [30]

2.7 Vývojové prostředí Laravel

Existuje několik možností, jak nastavit vývojového prostředí pro webové aplikace v Laravel framework.

2.7.1 Docker

Docker je platforma, která umožňuje oddělit vývojové prostředí od lokálního systému. To umožňuje jednoduchou instalaci různých nástrojů podle potřeb projektu bez toho, aniž by došlo k různým konfliktům. Pro práci s Laravel aplikacemi nabízí Laravel praktický nástroj *Sail*, který zjednodušuje nastavování a spouštění Laravel aplikací v Dockeru. [32]

2.7.2 Laravel Homestead

Laravel Homestead je vývojové prostředí, které umožňuje okamžité použití bez nutnosti instalace PHP, webového serveru nebo jiného serverového softwaru. Homestead využívá přednastavené virtuální prostředí Vagrant, které nabízí vytváření tzv. Vargant boxů, které jsou

jednorázové, takže pokud dojde k jakémukoli problému, lze box jednoduše smazat a znovu nastavit. [33]

Homestead je kompatibilní s Windows, macOS i Linuxem a obsahuje Nginx, PHP, MySQL, PostgreSQL, Redis, Memcached, Node a další software pro vývoj aplikací v Laravelu. [33]

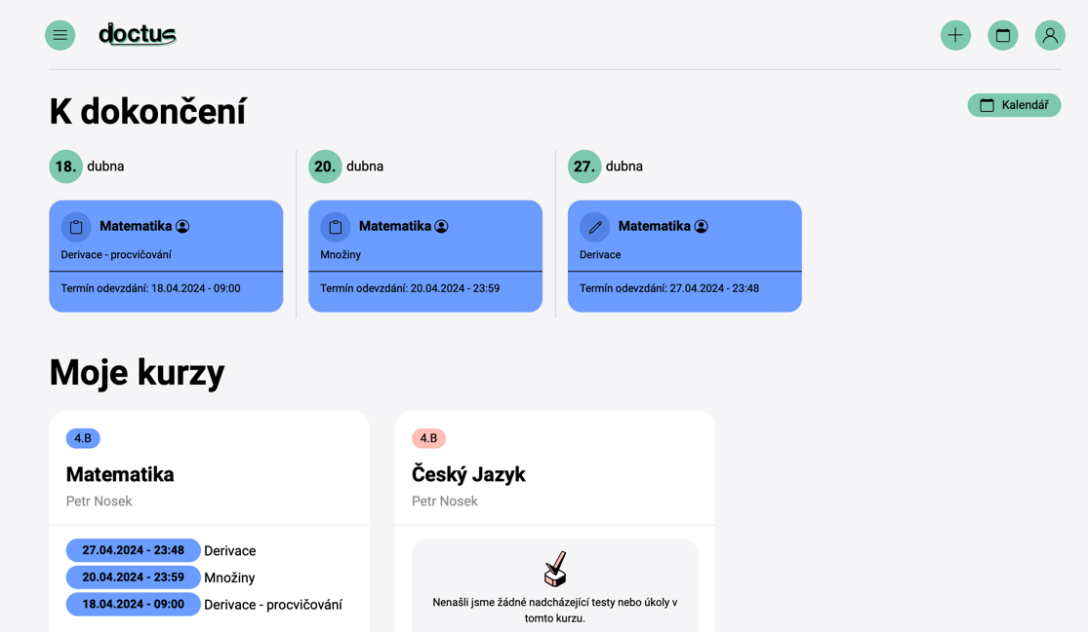
2.7.3 Herd

Laravel Herd je nativní vývojové prostředí pro macOS a Windows, které poskytuje rychlý a pohodlný způsob, jak začít s vývojem v Laravel a PHP. Toto prostředí zahrnuje veškerý potřebný software, včetně PHP a nginx, což umožňuje po instalaci začít pracovat na projektech v Laravel. [34]

II. PRAKTICKÁ ČÁST

3 POPIS STÁVAJÍCÍ APLIKACE

Tato kapitola se věnuje popisu již existující webové aplikace, která představuje výchozí bod pro následný převod s využitím frameworku Laravel. Kapitola popisuje hlavní funkcionality, architekturu, strukturu databáze, uživatelské prostředí, a především identifikuje limitace a problémy původního řešení.



Obrázek 5 – Hlavní přehled původní aplikace

3.1 Použité technologie

Webová aplikace byla původně vyvinuta pomocí skriptovacího programovacího jazyka PHP, přičemž nebyl využit žádný z PHP frameworků. Data jsou ukládána na platformě MySQL a pro interakci s databází se používá databázový wrapper. Pro rozšíření funkcionalit bylo použito několik JavaScriptových knihoven, zejména JQuery a Anime.js. Uživatelské rozhraní bylo realizován s využitím CSS frameworků Bootstrap a UIKit.

3.2 Přehled funkcionality

Webová aplikace je efektivní a snadno použitelný nástroj určený pro učitele a studenty, kteří chtějí lépe zorganizovat svoji výuku a učení. S tímto webovým nástrojem si učitelé vytvoří svůj vlastní kurz, do kterého se studenti jednoduše přihlásí pomocí unikátního kódu kurzu. V rámci kurzu mohou učitelé vytvářet testy, zadávat a hodnotit úkoly, nahrávat a organizovat studijní materiály a komunikovat se studenty formou příspěvků.

3.2.1 Testy

Testy lze snadno vytvářet v jednoduchém editoru, kde je možnost vytvářet otevřené i uzavřené otázky s maximálně čtyřmi možnostmi. Je možné nastavit časový limit pro vyplnění testu a definovat maximální počet pokusů, které mohou studenti využít. Každá odpověď testu může být ohodnocena určitým počtem bodů.

3.2.2 Úkoly

Učitelé kurzů mohou vytvářet úkoly s termínem odevzdání, kde studenti mohou nahrávat jakýkoliv typ souboru. Po odevzdání úkolu vidí učitel v přehledu, kdo a kdy úkol odevzdal a zároveň může ohodnotit úkol a napsat studentovi soukromý komentář. Prostřednictvím soukromých komentářů mohou učitelé a studenti mezi sebou komunikovat ohledně odevzdaných prací.

3.2.3 Materiály

V rámci webové aplikace je možné nahrávat studijní materiály pro studenty. Učitel má možnost snadno organizovat a spravovat materiály, což umožňuje přizpůsobit, které dokumenty, videa nebo prezentace budou pro studenty viditelné.

3.2.4 Příspěvky

Studenti i učitelé mohou vytvářet příspěvky v rámci jednotlivých kurzů. Každý příspěvek může být následně komentován ostatními studenty i učitelem.

3.2.5 Kalendář

Součástí aplikace je integrován přehledný kalendář, který poskytuje přehled všech naplánovaných úkolů a testů.

3.3 Architektura a struktura kódu

Při vývoji původní webové aplikace nebyl aplikován žádný návrhový vzor, což vede k obtížnější orientaci v kódu. Nebyla v projektu uplatněna žádná logika organizace kódu, tudíž struktura působí neuspořádaně a může být náchylná k chybám při budoucích úpravách. Struktura složek projektu je rozdělena následovně:

- *Class* – Složka obsahuje všechny PHP soubory, které se zabývají zobrazením a zpracováním dat týkajících se kurzů.

- *Img* – Tento adresář obsahuje všechny obrázky, které webová aplikace používá. Jedná se o loga, ikony a další grafické prvky.
- *Soubory* – Tato složka slouží jako úložiště všech souborů, které uživatele nahrají. Obsahuje např. nahrané materiály nebo odevzdané úkoly studentů.
- *Test* – Složka obsahuje všechny PHP soubory pro zobrazení a zpracování informací o testech v rámci jednotlivých předmětů.
- *User* – Adresář zahrnuje PHP soubory pro autorizaci uživatelů (přihlášení a registrace), úpravy uživatelských profilů a další pomocné soubory.

Navíc v projektu lze nalézt CSS styly v souboru *style.css* a databázový wrapper.

3.3.1 Struktura kódu

Jak už bylo zmíněno, struktura kódu původní aplikace je velmi nepřehledná a těžko by se přidávaly nové funkce. Pro vývojáře, kteří na projektu nepracovali od samého začátku, by mohla být orientace složitá. Zároveň jsou jednotlivé soubory pojmenovány velmi široce a v některých souborech jsou funkce, které by správně měly být oddělené nebo by se měly nacházet na jiných místech projektu. Zde je několik příkladů.

3.3.1.1 Stránka s kurzem

Vypisování přehledu kurzu probíhá v souboru *showClass.php* ve složce *class.* Tento soubor nemá vlastní kontrolér a všechny funkce a dotazy na databázi jsou umístěny buď na začátku souboru nebo v částech, kde se nachází samotné HTML.

```
<?php
…
// Dotaz pro získávání dat ohledně konkrétního kurzu

$idClass = Db::querySingle("SELECT idpredmetu FROM predmety WHERE kod = ?", $_GET["class"]);
if ($idClass) {
    $kontrola = Db::queryOne("SELECT * FROM uzivatele_predmety WHERE iduzivatele = ? &&
idpredmetu = ?", $_SESSION["idUzivatele"], $idClass);
    if ($kontrola) {
        require_once("classFunctions.php");
        require_once("../test/testFunctions.php");

    } else header("Location:../index.php");
} else header("Location:../index.php");
?>

<!DOCTYPE html>
<html lang="en">
```

Dalším nedostatkem je nepoužívání modelů, což vyžaduje explicitní zadávání SQL příkazů. Tento přístup zvyšuje náchylnost k chybám a komplikují případné změny ve struktuře databáze, protože je třeba aktualizovat SQL příkazy v rámci celého projektu.

3.3.2 Kontrola přihlášení

Kontrola, jestli je uživatel přihlášený nebo ne, probíhá v každém souboru, kde to je nutné. Tento přístup může vést k chybám a případné změny musí být prováděny ve většině souborů projektu.

```
// kontrola jestli je uživatel přihlášený
if (!$_SESSION["idUzivatele"]) {
    header("Location:../user/login.php");
} else {
    ini_set("file_uploads", "on");
    ini_set("upload_max_filesize", "30M");
…
```

3.3.3 Absence šablonovacího systému

Projekt nevyužívá žádný šablonovací systém, a tak je pro každou stránku nutné definovat kompletní HTML strukturu. Komponenty, jako jsou těla stránek nebo nadpisy kurzů, jsou specifikovány samostatně v každém souboru. Proto je jakákoli změna vzhledu nebo struktury těchto komponent časově náročná, protože vyžaduje úpravy v každém souboru, kde jsou tyto komponenty použity.

3.3.4 Odesílání formulářů v kurzu

Většina logiky, která zpracovává formuláře v kurzu, je umístěna v souboru *classFunctions.php*. Tento soubor je velmi obsáhlý a hledání konkrétní funkce je velmi obtížné. Nejsou zde definovány jednotlivé funkce, ale pouze podmínky ověřující, zda jsou proměnné z formuláře nastaveny.

```
// Podmínka pro formulář s editací úkolu

if (isset($_POST["name_ukol_edit"])) {

    $nazev_ukol = $_POST["name_ukol_edit"];
    $body = $_POST["max_bodu_edit"];
    $termin = $_POST["termin_ukol_edit"];
    $termin_cas = $_POST["termin_ukol_cas_edit"];
…
}
```

3.3.5 Funkce

Několik funkcí, jako jsou ty pro přidávání a odebírání soukromých komentářů u úkolů, je definováno přímo v souborech určených pro zobrazení uživatelů, aniž by byly odděleny do samostatných kontrolérů nebo souborů.

```
<!--Funkce pro přidání soukromého komentáře-->
<script>
    function pridatkoment(element) {
        var komu = $(element).data("id");
        var input = $(element.parentElement.parentElement).find(".col-
9").find(".koment_text").val();

        var koment_div =
$(element.parentElement.parentElement.parentElement).find(".col-
12").find(".comments");
…
</script>
```

3.4 Struktura databáze

Databáze původní webové aplikace obsahuje 15 tabulek, ve kterých jsou uloženy všechna potřebná data. Samotná databáze je implementována pomocí technologie MySQL a skládá se z následujících tabulek.

- *uzivatele* – Tato tabulka obsahuje informace o uživatelích systému.
- *uzivatele_predmety* – Tato tabulka slouží k asociaci mezi uživateli a předměty, ve kterých jsou zapsáni.
- *predmety* – Obsahuje informace o vytvořených předmětech (kurzech). Součástí jsou sloupce o jméně předmětu, štítku předmětu, vybrané barvě a ID autora.
- *prispevky* – Tato tabulka uchovává příspěvky jednotlivých předmětů.
- prispevky_komentare – Obsahuje komentáře ke konkrétním příspěvkům. Nevýhodou této tabulky je absence ukládání ID autora komentáře. Místo toho se ukládá pouze jeho jméno a příjmení.
- *ukoly* – Obsahuje informace o vytvořených úkolech v předmětech.
- *ukoly_soubory* – Tato tabulka uchovává informace o souborech přiložené k úkolům. Obsahuje cestu souboru a původní název souboru, protože v úložišti aplikace je soubor pojmenován unikátním názvem.
- *odevzdane_ukoly* – Obsahuje informace o odevzdaných úkolech. Součástí je cesta k odevzdanému souboru. Jednou z nevýhod je možnost odevzdat pouze jeden soubor v rámci jednoho úkolu.
- *komentare* – Tato tabulka uchovává soukromé komentáře mezi studentem a učitelem v rámci úkolu.
- *testy* – Obsahuje informace o testech v systému jako je např. časový limit testu, počet možných pokusů nebo termín odevzdání.
- *odpovedi* – Tabulka uchovává odpovědi studentů na otázky v testu.

- *uzivatele_test* – Tato tabulka obsahuje informace o testech, které studenti absolvovali. Samotné odpovědi jsou uloženy v tabulce *odpovedi*.
- *uzavrene_otazky* – Uchovává data o uzavřených otázkách. Nevýhodou je možnost definovat pouze čtyři možnosti otázky.
- *otevevrene_otazky* – Tato tabulka uchovává otevřené otázky v testech.

Obrázek 6 – Struktura databáze původní aplikace

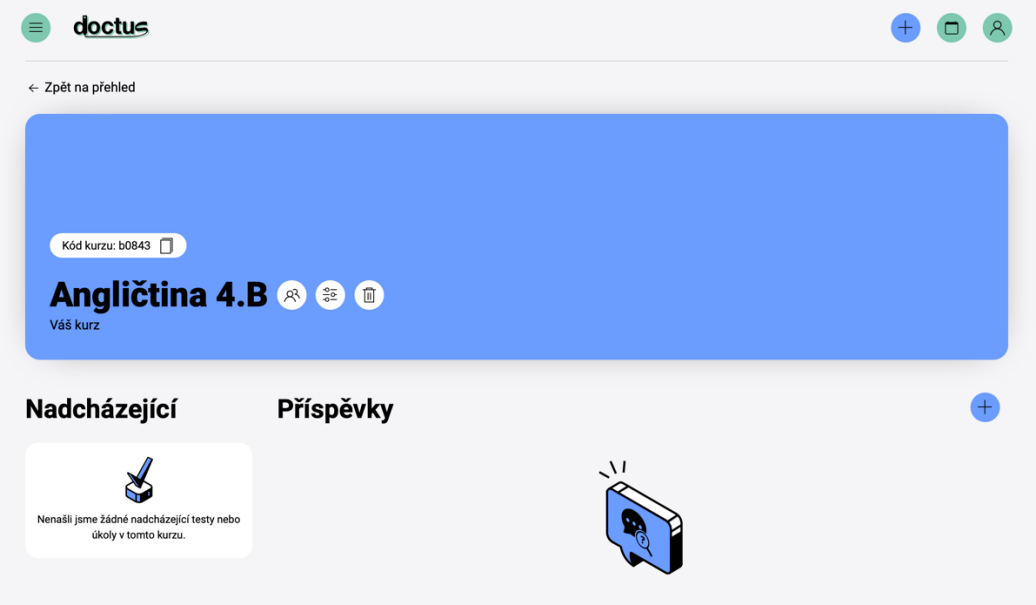
3.5 Uživatelská interakce a zkušenost

Uživatelské rozhraní původní aplikace využívá již zmíněné CSS frameworky Bootstrap a UIKit. Tyto CSS frameworky nabízí moderní, a především responzivní vzhled

jednotlivých komponent. Kromě stylů poskytnuté těmito frameworky byla řada prvků přizpůsobena a doplněna vlastními styly.

Uživatelské prostředí aplikace nabízí jednoduchý a přehledný design, přičemž byla využita předem definována barevná paleta a typografie. Navigace v aplikaci je řešena prostřednictvím menu, odkazů a tlačítek. Pro několik funkcí webové aplikace jsou využity dialogová okna. V neposlední řadě uživatelské rozhraní rovněž umožňuje volbu mezi světlým a tmavým režimem.

Díky responzivitě se uživatelské rozhraní aplikace zobrazuje správně na většině zařízení, včetně mobilních telefonů, tabletů a desktopů.



Obrázek 7 – Přehled kurzu v původní aplikaci

3.6 Identifikace problémů a omezení

Tato kapitola se věnuje popisu problémů a technických omezení, se kterými se původní aplikace potýká.

3.6.1 Škálovatelnost a struktura kódu

Jedním z hlavních problémů, kterému původní webová aplikace čelí, je otázka škálovatelnosti. Aplikace je postavena na čistém PHP bez využití jakéhokoli frameworku nebo návrhového vzoru. Tento přístup může být vhodný pro jednoduché aplikace, ale vede k omezením v budoucí rozšiřitelnosti aplikace.

Struktura kódu je nepřehledná, což komplikuje údržbu a přidávání nových funkcí. Kromě toho nebyl využit žádný návrhový vzor, který by strukturu kód udržel organizovaný.

3.6.2 Struktura databáze

Některé tabulky databáze původní aplikace používají nejednoznačné názvy pro tabulky a sloupce. Kromě toho jsou některé tabulky navrženy příliš složitě. Typickým příkladem jsou tabulky související s testy a jejich otázkami, kde by bylo vhodnější strukturu zjednodušit. Nejenže tento problém ztěžuje manipulaci s daty, ale také zvyšuje nároky na výpočetní výkon při zpracování dotazů.

Dalším významným nedostatkem je absence využívání modelů a migrací.

3.6.3 Uživatelské prostředí

Jedním z problémů uživatelského prostředí je nevyužití šablonovacího systému. Tento problém vede k nadměrnému opakování kódu v rámci aplikace. Jedná se např. o navigační menu, patičky stránky a dalších často používaných komponentech. To zvyšuje náročnost údržby a zpomaluje vývoj, ale také vede k zanedbání aktualizací nebo oprav ve všech částech aplikace.

Dalším výrazným problémem je souběžné využití dvou CSS frameworků. Použití Bootstrapu společně s UIKitem v jedné aplikaci vede ke možným konfliktům mezi CSS třídami a opakování CSS stylů. Překrývání tříd také komplikuje úpravy a vylepšení designu, protože změny v jednom frameworku mohou ovlivnit vzhled prvků, které jsou stylovány druhým frameworkem.

3.6.4 Bezpečnost

Původní aplikace, která nepoužívá přístupy k ochraně databázových dotazů, může být náchylná k SQL injection. Zároveň nejsou všechny vstupy formulářů validovány na straně serveru, ale pouze na straně klienta.

Dále používá původní verze zastaralou verzi PHP. Bez pravidelných bezpečnostních aktualizací se zvyšuje riziko využití známých chyb ze starších verzí PHP.

3.7 Vyhodnocení

Z popisu původní aplikace vyplývá, že struktura kódu je příliš komplikovaná, což ztěžuje orientaci v projektu a přebírání jeho jednotlivých funkcí. Uživatelské prostředí, které

kombinuje více frameworků vede ke složitější údržbě a rozvoji aplikace. Zároveň není využit žádný šablonovací systém a v souborech jsou komponenty uživatelského prostředí definovány vícekrát. Vzhledem k velikost a limitacím původního projektu je rozumnější začít s vývojem projektu zcela od začátku, přičemž nový projekt by se měl držet původních vlastností a cílů, které byly pro původní projekt stanoveny.

4 PLÁNOVÁNÍ PŘEVODU

Následující kapitola se věnuje definici klíčových cílů a požadavků, výběrem vhodných technologií, které zlepší budoucí udržitelnost aplikace, a návrhu nové architektury.

4.1 Definice cílů a požadavků

Nová verze webové aplikace zachovává všechny stávající funkce předchozí verze. Následující body popisují klíčové funkce, které aplikace bude obsahovat.

Registrace a přihlášení – Aplikace poskytuje bezpečný systém pro registraci nových uživatelů a přihlašování již zaregistrovaných uživatelů. Registrace umožňuje uživatelům vytvořit si vlastní účet pomocí e-mailové adresy a hesla.

Editace uživatelského profilu – Aplikace poskytuje uživatelům možnost spravovat a aktualizovat své osobní údaje v rámci svého profilu. Každý uživatel má možnost změnit své jméno, e-mailovou adresu a heslo.

Kurzy – Uživatelé mají možnost vytvářet vlastní kurzy, přičemž mohou nastavit název, štítek a barevné schéma kurzu. Přístup do kurzu je řešen pomocí unikátního přístupového kódu. Správci kurzu mohou upravovat detaily kurzu a spravovat seznam uživatelů, včetně možnosti vyloučení uživatelů z kurzu.

Příspěvky – V rámci kurzu mohou uživatelé publikovat příspěvky a na ně reagovat formou komentářů. Autoři mají možnost své příspěvky a komentáře upravovat nebo odstranit.

Materiály – Správce kurzu může do kurzu nahrávat výukové materiály, které jsou dostupné pro přihlášené uživatele v kurzu. Tyto materiály může dle potřeby skrýt, zobrazit či úplně smazat.

Úkoly – Správci kurzu mohou vytvářet úkoly, definovat jejich názvy, termín odevzdání a maximální počet bodů. Uživatelé mohou k úkolům odevzdávat soubory, které jsou následně hodnoceny správcem kurzu. U úkolů je také možné využívat soukromé komentáře.

Testy – Správce má možnost vytvářet testy s otevřenými či uzavřenými otázkami, nastavovat termíny dostupnosti a odevzdání a definovat maximální počet pokusů pro uživatele. U uzavřených otázek lze vytvořit neomezený počet možností s minimálně jednou správnou odpovědí. Uživatelé mohou testy vyplnit a po odevzdání si prohlédnout výsledky. Správce může upravovat hodnocení otázek a hodnotit otevřené odpovědi.

Kalendář – Aplikace poskytuje kalendář, v němž uživatelé najdou přehled o nadcházejících událostech, jako jsou termíny testů a úkolů.

4.2 Výběr nástrojů a technologií

Nová webová aplikace bude postavena na PHP frameworku Laravel. Výběr Laravel frameworku zajišťuje moderní architekturu aplikace, která bude flexibilní, škálovatelná a bezpečná. Bude použito vývojové prostředí Laravel Herd a bude psána ve vývojovém prostředí PHP Storm.

Databázové řešení bude postaveno na MySQL, přičemž pro vytvoření databázového serveru a snadnou konfiguraci bude použit program DBngin.

Pro zjednodušení autentizačních procesů a základního nastavení uživatelského profilu bude využit Laravel Breeze, který poskytuje startovací balíček pro autentizaci s předkonfigurovanými funkcemi.

Interaktivita uživatelského rozhraní bude řešena pomocí Livewire, který umožňuje vytvářet dynamické uživatelské rozhraní s minimálním použitím Javascriptu. K jeho fungování je potřeba javascriptový framework Alpine.js.

Pro sledování stavu aplikace v reálném čase a ladění budou využívány nástroje jako jsou Laravel Telescope a Laravel DebugBar, které poskytnou informace o dění v aplikaci při potenciálních problémech.

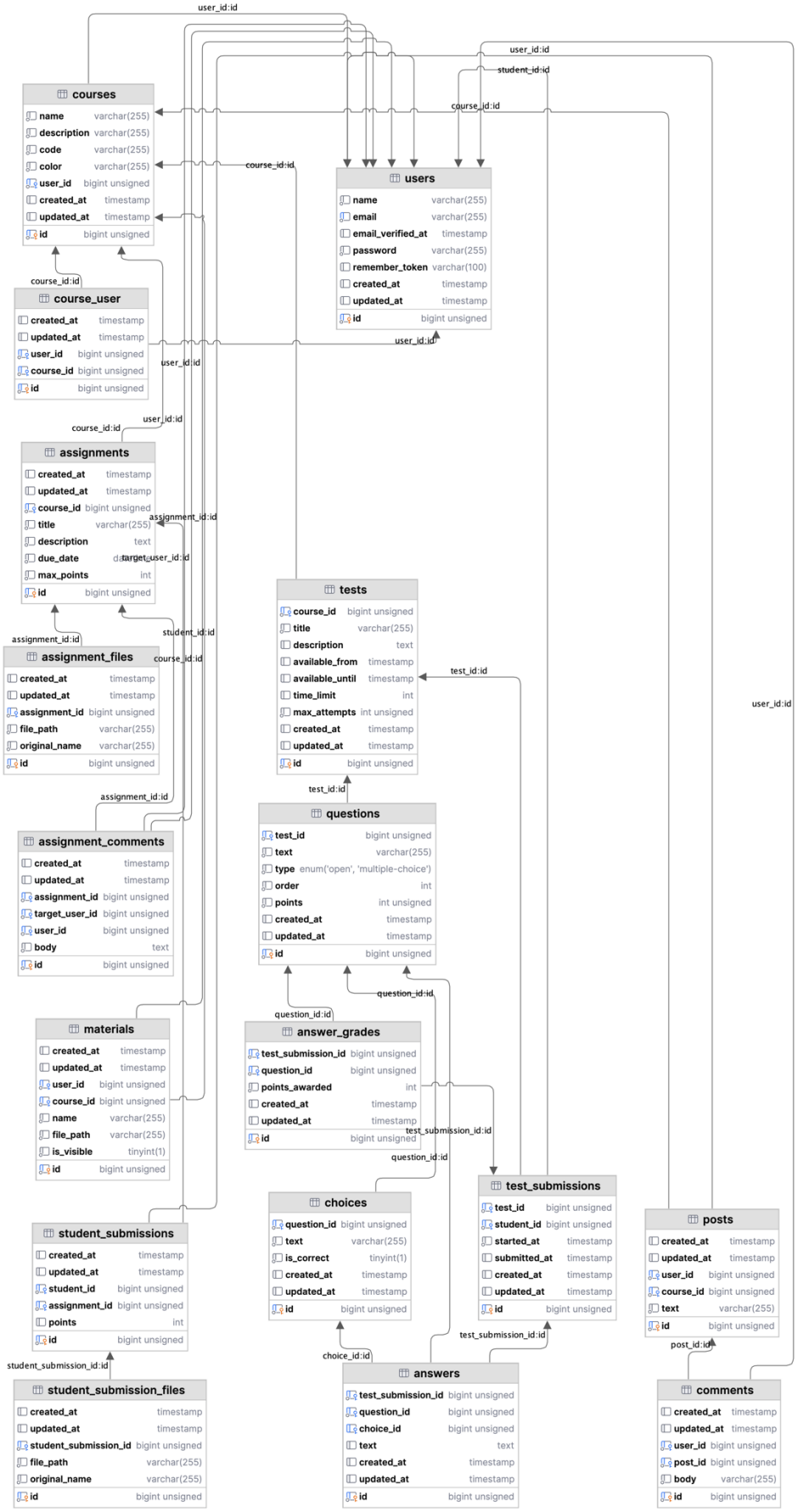
Stylování aplikace bude realizováno pomocí frameworku Tailwind CSS. Důvodem výběru tohoto CSS frameworku je potřeba nahradit původní dva CSS frameworky a vybrat jeden konkrétní. Druhým důvodem je, že Tailwind CSS je již integrován v Laravel Breeze.

4.3 Návrh nové architektury

Tato kapitola se zaměřuje na návrh nové architektury pro novou webovou aplikaci. Popisuje jednotlivé databázové tabulky, modely a příslušné kontroléry, které nová verze webová aplikace využije.

4.3.1 Architektura databáze

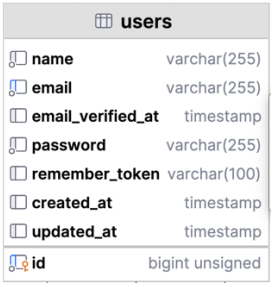
Nová Struktura databáze je navržena na základně struktury databáze původní aplikace. Některé tabulky byly zjednodušeny nebo navrženy tak, aby se s nimi lépe pracovalo. Následuje popis jednotlivých tabulek.



Obrázek 8 – Struktura databáze nové verze aplikace

4.3.1.1 users

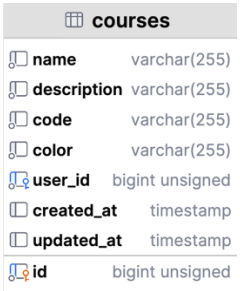
Ukládá informace o uživatelích, včetně jména, hesla a e-mailu. Tabulka se automaticky vytvoří při instalaci Laravel framework.



Obrázek 9 – Tabulka users

4.3.1.2 courses

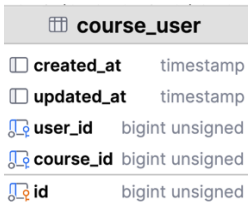
Tato tabulka obsahuje detaily kurzu, jako je název, štítek, ID autora kurzu, barevné schéma kurzu a unikátní kód, kterým se uživatelé přihlásí do kurzu.



Obrázek 10 – Tabulka courses

4.3.1.3 course_user

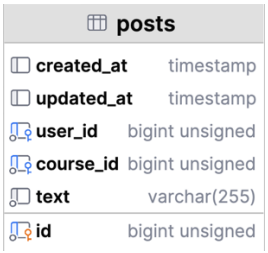
Tabulka vytváří vztah mezi uživateli, kteří se přihlásili do kurzu, a kurzem. Je propojena s tabulkami Users a Courses, čímž umožní vzájemné propojení více uživatelů s více kurzy.



Obrázek 11 – Tabulka course_user

4.3.1.4 Posts

Uchovává data o vytvořených příspěvcích v kurzu. Kromě obsahu příspěvku obsahuje tabulka datum vytvoření a informace o autorovi. Vytváří vztah s tabulkou *courses*, kdy jeden kurz může mít více příspěvků.



Obrázek 12 – Tabulka posts

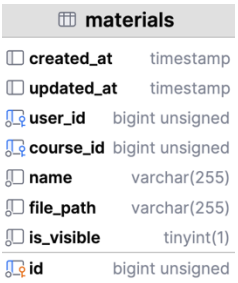
4.3.1.5 comments

Obsahuje komentáře k příspěvkům, včetně textu komentáře a propojení s uživatelem a příspěvkem. Tento vztah umožňuje, aby jeden příspěvek i jeden uživatel mohli mít přiřazeno několik komentářů.

Obrázek 13 – Tabulka comments

4.3.1.6 materials

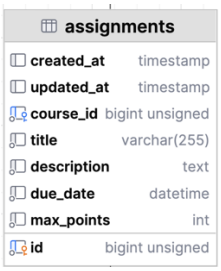
Slouží pro uchování nahraných studijních materiálů kurzu, včetně názvu souboru, cesty k němu a viditelnosti pro uživatele. Vytváří vztah s tabulku *courses*, kdy jeden kurz může mít více příspěvků.



Obrázek 14 – Tabulka materials

4.3.1.7 assignments

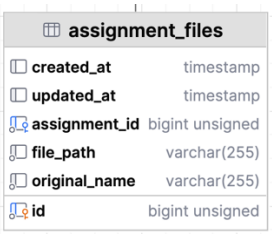
Tabulka ukládá data o úkolech v kurzech. Obsahuje názvy, popisy, termíny odevzdání, maximální počet bodů a vztah k příslušnému kurzu, kdy k jednomu kurzu může být přiřazeno více úkolů.



Obrázek 15 – Tabulka assignments

4.3.1.8 assignment_files

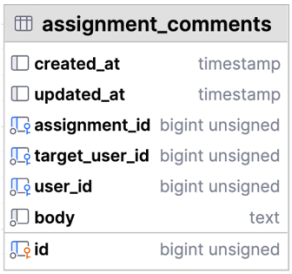
Ukládá data o souborech přiložené k jednomu konkrétnímu úkolu s cestou k souboru a původním názvem.



Obrázek 16 – Tabulka assignment_files

4.3.1.9 assignment_comments

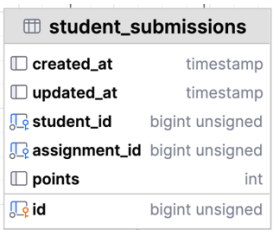
Uchovává soukromé komentáře k úkolům, včetně vztahu k úkolu, cílového uživatele a autora komentáře.



Obrázek 17 – Tabulka assignment_comments

4.3.1.10 student_submissions

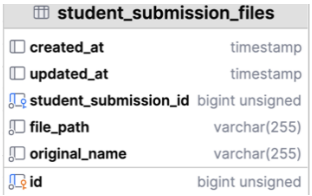
Tato tabulka obsahuje informace o odevzdaných úkolech studentů. Zajišťuje propojení každého odevzdaného úkolu s konkrétním studentem a úkolem.



Obrázek 18 – Tabulka student_submissions

4.3.1.11 student_submission_files

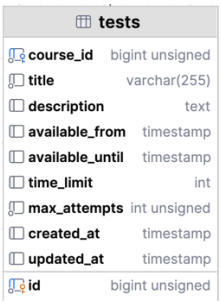
Tabulka obsahuje informace o souborech odeslané studenty jako součást jejich odevzdaných úkolů. Oproti původní verzi tato tabulka umožní studentům odevzdávat více než jeden soubor.



Obrázek 19 – Tabulka student_submission_files

4.3.1.12 tests

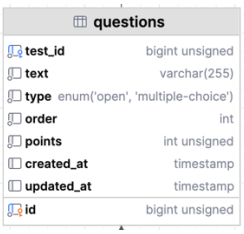
Tabulka obsahuje informace o testech v kurzu, včetně názvu, popisu, maximálního počtu pokusů, datumu odevzdání a limitu času na dokončení.



Obrázek 20 – Tabulka tests

4.3.1.13 questions

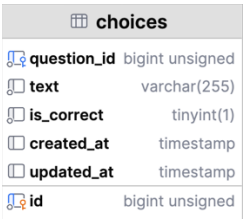
Uchovává otázky testů s typem otázky, počtem bodů a textem otázky. Také tabulka ukládá ID testu, ke kterému je přiřazena.



Obrázek 21 – Tabulka questions

4.3.1.14 choices

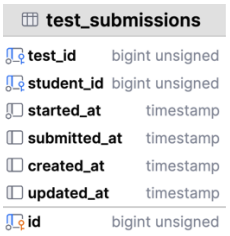
Tabulka obsahuje možnosti odpovědí pro uzavřené otázky, včetně sloupce určeného pro označení správných odpovědí. Stejně jako tabulka otázek (*questions*) obsahuje ID testu, ke kterému otázky patří. Na rozdíl od původní verze, tato tabulka umožňuje vytvořit více než pouze čtyři možnosti pro každou uzavřenou otázku.



Obrázek 22 – Tabulka choices

4.3.1.15 test_submissions

Tabulka obsahuje pokusy testů, časy začátku a odevzdání testu a odkaz na konkrétní test a uživatele, který test vyplňoval.



Obrázek 23 –Tabulka test_submissions

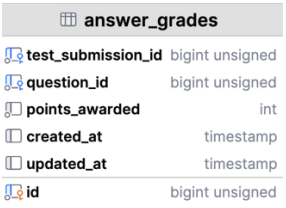
4.3.1.16 answers

Ukládá odpovědi uživatelů na testové otázky, včetně textu odpovědi nebo ID vybrané možnosti. Zároveň tvoří vztah ke konkrétnímu pokusu testu.

Obrázek 24 – Tabulka answers

4.3.1.17 answer_grades

Zaznamenává bodové hodnocení pro jednotlivé odpovědi v testech a spojuje otázky s hodnocenými odpověďmi. Tabulku umožní správci kurzu zpětně ohodnotit jednotlivé odpovědi uživatele.



Obrázek 25 – Tabulka answer_grades

4.3.2 Modely

Tato kapitola se věnuje podrobnému popisu jednotlivých datových modelů v nové verzi aplikace a jejich vztahů mezi nimi. Modely využívají vztahy mezi modely, jako jsou *belongsTo*

(jeden model patří k jinému modelu), *hasMany* (jeden model může obsahovat více instancí jiného modelu) a *belongsToMany* (vztah pro definování mnoho-ku-mnoha).

- **User** – Reprezentuje uživatele aplikace.
  - *hasMany Post*
  - *hasMany Comment*
  - *hasMany StudentSubmission*
  - *hasMany TestSubmission*
  - *hasMany AssignmentComment*
  - *belongsToMany Course* (mnoho uživatelů může být zapsáno do mnoha kurzů)
  - *hasMany* Course (kurzy, které vlastní uživatel)
- **Course** – Reprezentuje kurz, který může obsahovat příspěvky, materiály, úkoly a testy.
  - *hasMany Post*
  - *hasMany Material*
  - *hasMany Assignment*
  - *hasMany Test*
  - *belongsToMany User* (kurz může obsahovat mnoho studentů)
  - *BelongsTo User* (kurz patří jednomu konkrétnímu uživateli)
- **Post** – Příspěvky vytvořené uživateli v rámci kurzu.
  - *belongsTo User*
  - *belongsTo Course*
  - *hasMany Comment*
- **Comment** – Komentáře k příspěvkům uživatelů.
  - *belongsTo User*
  - *belongsTo Post*
- **Material** – Studijní materiály, které jsou součástí kurzu.
  - *belongsTo Course*
  - *belongsTo User*
- **Assignment** – Úkoly zadané v rámci kurzu.
  - *hasMany AssignmentFile*
  - *hasMany AssignmentComment*
  - *hasMany StudentSubmission*

    - *belongsTo Course*
- **AssignmentFile** – Soubory připojené k úkolům.
    - *belongsTo Assignment*
- **AssignmentComment** – Soukromé komentáře k úkolům od uživatelů.
    - *belongsTo User*
    - *belongsTo Assignment*
- **StudentSubmission** – Odevzdané řešení úkolu studentem.
    - *belongsTo User*
    - *belongsTo Assignment*
    - *hasMany StudentSubmissionFile*
- **StudentSubmissionFile** – Soubory připojené k odevzdaným úkolům studentů.
    - *belongsTo StudentSubmission*
- **Test** – Testy vytvořené pro hodnocení studentů v rámci kurzu.
    - *hasMany Question*
    - *hasMany TestSubmission*
    - *belongsTo Course*
- **Question** – Otázky, které jsou součástí testů.
    - *belongsTo Test*
    - *hasMany Choice*
    - *hasMany Answer*
- **Choice** – Možné odpovědi na otázky v testech.
    - *belongsTo Question*
- **TestSubmission** – Představuje pokus studenta o vyřešení testu.
    - *belongsTo Test*
    - *belongsTo User*
    - *hasMany Answer*
    - *hasMany Grade*
- **Answer** – Odpovědi studentů na otázky v rámci testového pokusu.
    - *belongsTo TestSubmission*
    - *belongsTo Question*
    - *belongsTo Choice* (volitelné, pokud je otázka typu "multiple choice")
- **AnswerGrade** – Hodnocení udělené za odpovědi ve vypracovaných testech.
    - *belongsTo Answer*

- *belongsTo TestSubmission*

4.3.3 Kontroléry a Livewire komponenty

Tato část se zaměřuje na klíčovou část nového projektu – kontroléry a Livewire komponenty. Laravel umožňuje pomocí kontrolérů vytvářet logiku a zpracovávat požadavky uživatelů. Některé modely budou využívat komponenty Livewire.

4.3.3.1 Kurzy

Správu kurzů bude spravovat jeden kontrolér, který bude obsahovat funkci pro zobrazení pohledu s výpisem kurzů a pohledu s konkrétním kurzem. Tento kontrolér bude také obsahovat funkce pro editaci kurzu.

Kromě toho bude implementována Livewire komponenta, která uživatelům zobrazí seznam kurzů, do kterých jsou přihlášeni nebo které vlastní. Součástí této komponenty budou také funkce pro vytvoření nového kurzu a funkce pro přihlašování do existujících kurzů.

4.3.3.2 Příspěvky a komentáře

Příspěvky a komentáře budou zpracovávány pomocí jediné Livewire komponenty, jelikož jejich zobrazování probíhá na stránce s přehledem kurzu a není již potřeba vytvářet další kontrolér. Tato komponenta zahrnuje funkce pro výpis, vytváření a úpravy příspěvků a komentářů.

4.3.3.3 Materiály

Materiály budou mít svůj vlastní kontrolér, který bude obsahovat funkci pro zobrazení seznamu materiálů. Podobně jako příspěvky, i materiály budou využívat Livewire komponentu s funkcemi pro výpis, vytváření a mazání materiálů.

4.3.3.4 Úkoly

Úkoly v aplikaci budou spravovány pomocí několika kontrolérů a Livewire komponent. Pro výpis a mazání úkolů bude použita Livewire komponenta. Další kontrolér bude obsahovat funkce pro zobrazení detailů úkolu, jeho vytváření, editaci a odevzdávání. Pro hodnocení a výpis odevzdaných úkolů bude sloužit Livewire komponenta, která usnadní opravování úkolů a umožní vyhledávání studentů.

4.3.3.5 Testy

Testy, podobně jako úkoly, budou spravovány pomocí několika kontrolérů a Livewire komponent. Kontrolér bude sloužit pro zobrazení detailů testu, úpravy a zobrazení pokusů studentů. Jedna Livewire komponenta bude zodpovědná za vytváření testů, další za jejich editaci, zatímco jiná se bude věnovat samotnému vyplňování testu. Další komponenta umožní zobrazení seznamu studentů a jejich pokusů. Poslední Livewire komponenta bude použita pro bodování testů.

4.3.3.6 Kalendář

Kalendář bude rozdělen do dvou částí: celkový kalendář a kalendář pro konkrétní kurz. Pro každý z těchto kalendářů bude vytvořen samostatný kontrolér, který zobrazí příslušný pohled. Také bude využita Livewire komponenta, která bude sloužit pro výpis samotných událostí (termíny testů a úkolů).

4.3.4 Uživatelské prostředí

Nová verze aplikace bude využívat CSS framework Tailwind, který je již integrován do Laravel framework. Uživatelské prostředí bude vycházet z původní verze aplikace, ale bude více zjednodušeno. Pro lepší uspořádání bude využit šablonovací systém.

4.3.5 Zabezpečení

Klíčovým prvkem je důsledná validace všech uživatelských vstupů, která minimalizuje riziko SQL injection. Dále je zabezpečení posíleno použitím policies, které umožní definovat, co mohou uživatelé dělat, čímž se zajistí, že uživatelé mají přístup pouze k těm datům, která mají oprávnění vidět nebo upravovat. Co se týče hesel, ty jsou zabezpečeny pomocí hashování, což už řeší integrovaný modul Laravel Breeze.

5 PROCES PŘEVODU (IMPLEMENTACE)

Následující kapitola se věnuje popisu přípravy vývojového prostředí, tvorbě modelů a jejich migrací. V neposlední řadě je popsána implementace kontrolérů a logiky aplikace.

5.1 Konfigurace vývojového prostředí

Pro vývoj aplikace v rámci frameworku Laravel byl zvolen nástroj Laravel Herd, který poskytuje snadnou a rychlou instalaci Laravelu. Následující kroky popisují, jak byl nový projekt vytvořen a nakonfigurován s použitím Laravel Herd.

5.1.1 Instalace Laravel Herd

Aplikace Laravel Herd je dostupná ke stažení na těchto oficiálních stránkách: https://herd.laravel.com. Instalace zahrnuje nastavení všech potřebných vývojových prostředků pro Laravel, včetně nástrojů jako *php*, *composer* a samotný framework Laravel.

Po úspěšné instalaci lze v rámci aplikace Laravel Herd přidávat nebo upravit cesty, kde jsou uloženy projekty Laravel. Výchozí cesta pro nové projekty je *~/Herd*.

5.1.1.1 Vytvoření nového projektu

Jednoduše přes příkazový řádek vytvoříme nový projekt Laravel s názvem "doctus" s využitím následujících příkazů. Je zásadní, aby projekt byl umístěn v adresáři, který je specifikován v aplikaci Laravel Herd.

```
cd ~/Herd
laravel new doctus
```

Po zadání těchto příkazů lze volit z různých startovacích balíčků, které Laravel nabízí, a zvolit, zda bude projekt využívat verzovací systémem Git. Pro nový projekt nebyl vybrán žádný startovací balíček, protože bude implementován později.

5.1.1.2 Konfigurace nového projektu

V aplikaci DBngin vytvoříme nový databázový server a jako databázový systém zvolíme MySQL. Po vytvoření serveru jej spustíme. Následně se v terminálu přesuneme do složky s naším novým projektem a zadáme následující příkaz.

```
php artisan migrate
```

Po zadání tohoto příkazu budeme vyzváni, zda chceme vytvořit migrace. Zvolíme možnost ano. Nyní by měly být vytvořeny základní databázové tabulky, včetně tabulky pro uživatele a další nezbytné tabulky.

V případě potíží s připojením k novému databázovému serveru lze zkontrolovat soubor *.env* ve složce našeho projektu. V tomto souboru lze upravit parametry pro připojení k databázi. Následně spustíme předchozí příkaz znovu.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=doctus
DB_USERNAME=root
DB_PASSWORD=
```

5.1.1.3 Spuštění nového projektu

V aplikaci Laravel Herd, v záložce Sites, je možné kliknutím na ikonu globusu v řádku s názvem projektu otevřít funkční nový projekt ve webovém prohlížeči. Aby byly správně nahrány a fungovaly všechny JavaScriptové knihovny, je potřeba ještě spustit následující příkaz.

```
npm run dev
```

Nyní je náš nový projekt úspěšně vytvořen.

5.2 Instalace Laravel Breeze

Jak už bylo zmíněno v teoretické části, Laravel Breeze je jednoduchý nástroj pro implementaci autentifikace a autorizace v aplikacích Laravel. Na základě návrhu bude nový projekt tento nástroj využívat.

Pro instalaci tohoto balíčku stačí zadat následující příkazy v kořenové složce našeho projektu.

```
composer require laravel/breeze –dev
php artisan breeze:install
php artisan migrate
npm install
```

V závěrečné fázi ukončíme běh příkazu *npm run dev* a spustíme ho znovu. Po této akci je Laravel Breeze úspěšně nainstalován.

5.3 Instalace Livewire

Instalace samotného Livewire je velmi jednoduchá a stačí zadat následující příkaz.

```
composer require livewire/livewire
```

Protože Livewire i Laravel Breeze využívají javascriptovou knihovnu *Alpine.js* je nutné odstranit volání této knihovny ze souboru *resources/js/app.js*, aby se předešlo jejímu dvojitému načítání.

V závěrečném kroku instalace Livewire je potřeba přidat do souboru *resources/views/layouts/app.blade.php* direktivu *@livewireStyles* uvnitř HTML tagu *<head>* a *@livewireScripts* těsně před uzavřením tagu *<body>*.

5.4 Instalace dalších pomocných nástrojů

Dále v novém projektu budou využity následují nástroje.

5.4.1 Telescope

Pro instalaci ladícího nástroje Telescope, podobně jako u Laravel Breeze, stačí použít několik příkazů. Po jejich zadání bude nástroj úspěšně nainstalován. Funkčnost Telescope můžete ověřit na stránce */telescope*, kde by měly být zobrazeny ladící informace.

```
composer require laravel/telescope
php artisan telescope:install
php artisan migrate
```

5.4.2 Instalace Laravel DebugBar

Instalace Laravel DebugBar je jednoduchý proces, který umožní monitorovat a ladit projekt v Laravel. Zde je příkaz, který nainstaluje tento nástroj.

```
composer require barryvdh/laravel-debugbar --dev
```

Po dokončení instalace se na spodní části stránky projektu objeví lišta DebugBaru.

5.4.3 Livewire Alert

Livewire Alert je jednoduchý nástroj pro vytváření notifikací pro uživatele navržený pro Livewire. Samotná instalace je jednoduchá. V prvním kroku spustíme následující příkaz.

```
composer require jantinnerezo/livewire-alert
```

V dalším kroku je důležité vložit Livewire Alert komponentu a JavaScriptovou knihovnu *SweetAlert2*, kterou tento nástroj využívá, do souboru *resources/views/layouts/app.blade.php* před ukončením HTML tagu <*body*>.

```
<body>
    …
    @livewireScripts
    <script src="//cdn.jsdelivr.net/npm/sweetalert2@11"></script>
    <x-livewire-alert::scripts/>
</body>
```

5.4.4 TinyMCE

TinyMCE je webový textový editor, který umožňuje uživatelům snadno vytvářet a upravovat obsah přímo v prohlížeči, podobně jako v textových editorech. Tento editor bude využit u vytváření nových úkolu.

5.4.4.1 Instalace

Ve složce *resources/views/components* vytvoříme nový soubor pojmenovaný *tinymce-config.blade.php*. Do tohoto souboru pak vložíme skript pro inicializaci TinyMCE a příslušný odkaz na jeho knihovnu.

```
<script src="https://cdn.tiny.cloud/1/no-api-key/tinymce/7/tinymce.min.js"
referrerpolicy="origin"></script>
<script>
    tinymce.init({
        selector: 'textarea#description-editor'
    });
</script>
```

V posledním kroku stačí přidat tuto komponentu do souboru *resources/views/layouts/app.blade.php* uvnitř HTML tagu <*head*>.

```
<head>
    …
    <x-head.tinymce-config/>
</head>
```

5.5 Převod databáze a modelů

V nové verzi webové aplikaci je databáze tvořena pomocí migrací, které definují 25 tabulek pro uložení všech potřebných dat. Několik z těchto tabulek je využíváno ladícím nástrojem Telescope nebo autentizačním balíčkem Laravel Breeze.

5.5.1 Modely

Na rozdíl od původní aplikace, která modely nepoužívala a kde byla logika pro přístup k databázi rozptýlena napříč různými částmi kódu, nová verze aplikace využívá Eloquent ORM. Na základě návrhu vytvoříme příslušné modely a migrace.

5.5.1.1 Vytvoření modelů

Modely v Laravel framework lze vytvořit pomocí příkazu *Artisan*. K tomuto příkazu je možné přidávat další argumenty, které umožňují současně s vytvořením modelu generovat také migraci a kontrolér. Například pro model *Course* je potřeba vytvořit jak migraci, tak kontrolér.

```
php artisan make:model Course -crm
```

Argument *-c* vytvoří s modelem jeho příslušný kontrolér. Díky argumentu *-r* bude tento kontrolér obsahovat metody pro standardní CRUD operace a poslední argument *-m* vytvoří migraci. Vytvořený model lze najít ve složce *app/Models*.

5.5.1.2 Migrace

Pokud migrace byla vytvořena zároveň s modelem, není potřeba spouštět další příkaz. V případě, že potřebujeme vytvořit migraci pro již existující modely, stačí zadat tento příkaz.

```
php artisan make:migration create courses table
```

Vytvořené migrace lze najít ve složce *database/migrations*. Poté upravíme migraci tak, aby odpovídala návrhu nové aplikace. To znamená definování sloupců, jejich názvů a příslušných datových typů. Například migrace pro tabulku s příspěvky by dle návrhu vypadala následovně.

```
public function up(): void
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
        $table->foreignId('user_id')->constrained('users')->cascadeOnDelete();
        $table->foreignId('course_id')->constrained('courses')->cascadeOnDelete();
        $table->string('text');
    });
}
```

U tabulek, které zahrnují cizí klíče, je nutné tyto klíče správně definovat. Například u příspěvků, které jsou přiřazeny k určitému kurzu, je třeba vytvořit cizí klíč pomocí metody *foreignId*. Metoda *cascadeOnDelete()* zajistí, že pokud je kurz smazán, všechny příspěvky spojené s tímto kurzem budou také smazány.

5.5.2 Vztahy mezi modely

Z jednou výhod modelů je definování vztahů mezi dalšími modely. Eloquent ORM v Laravelu poskytuje sadu metod pro práci s těmito vztahy, jako je *hasMany()*, *belongsTo()* a *belongsToMany()*. Vztahy vytvoříme u příslušných modelů podle návrhu modelů nové aplikace.

5.5.2.1 hasMany()

Např. model kurzu může mít více příspěvků, které spadají právě do toho kurzu. Proto je nutné do modelu s kurzem přidat následující funkci s příspěvky. Tato funkce zajistí, že budeme moct jednoduše získávat příspěvky kurzu.

```
class Course extends Model
{
      public function posts(): HasMany
      {
          return $this->hasMany(Post::class);
      }
}
```

5.5.2.2 belongsTo()

Příkladem můžou být příspěvky, které patří do jednoho konkrétního kurzu.

```
class Post extends Model
{
      public function course(): BelongsTo
      {
          return $this->belongsTo(Course::class);
      }
}
```

5.5.2.3 belongsToMany()

Konkrétním příkladem vztahu *belongsToMany* jsou uživatelé, kteří se mohou zapsat do více kurzů, zatímco každý kurz může obsahovat mnoho zapsaných uživatelů.

```
class User extends Authenticatable
{

      public function coursesAsParticipant(): BelongsToMany
      {
          return $this->belongsToMany(Course::class, 'course_user');
      }
}

class Course extends Model
{

      public function users(): BelongsToMany
      {
          return $this->belongsToMany(User::class, 'course_user');
      }
}
```

5.5.3 Protected fillable

U modelů je také nutné specifikovat, které atributy modelu mohou být hromadně nastaveny skrze formuláře. V modelu proto definujeme pole nazvané *fillable* a vněm uvedeme atributy modelu, která mohou být přiřazována. Například pro model kurzu by seznam v poli *fillable* vypadal následovně.

```
class Course extends Model
{
…
    protected $fillable = [
        'name',
        'description',
        'code',
        'color'
    ];
…
}
```

5.5.4 Struktura databáze

Pro aplikování vytvořených migrací zadáme příkaz Artisan.

```
php artisan migrate
```

Pro správné fungování aplikace jsou kromě tabulek modelů nezbytné i další. Migrace těchto tabulek se již implementovali s instalací Laravel a dalších nástrojů pro tento framework.

- *failed_jobs* (uchovává informace o úlohách, které se nepodařilo zpracovat)
- *migrations* (migrace, které byly aplikovány na databázi)
- *password_reset_tokens* (tokeny pro resetování hesel)
- *personal_access_tokens* (správu osobních přístupových tokenů)

- *telescope_entries* (logy a další informace, které zaznamenává Laravel Telescope)
- *telescope_entries_tags* (asociuje tagy s jednotlivými záznamy)
- *telescope_monitoring* (informace o monitorovaných úlohách a procesech v aplikaci)

5.6 Implementace kontrolérů a logiky aplikace

K vytvoření kontrolérů lze využít příkaz *Artisan*, avšak pokud byl kontrolér již vytvořen společně s modelem, není nutné tento příkaz znovu spouštět. V případech, kdy kontrolér nebyl vytvořen společně s modelem, je třeba samostatně vytvořit kontroléry podle návrhu aplikace. Všechny existující kontroléry lze najít ve složce */app/Http/Controllers*.

```
php artisan make:controller CourseController
```

Následně využijeme Livewire komponenty, které vytvoříme tímto příkazem.

```
php artisan make:livewire courses
```

K této komponentě se automaticky vytvoří i pohled, který lze najít ve složce *resources/views/livewire*. Livewire komponenta pak může být vložena do jakéhokoli Blade šablonového souboru pomocí direktivy.

```
<livewire:courses></livewire:courses>
```

Následuje popis částí nového projektu a přehled jejich klíčových funkcí.

5.6.1 Kurzy

Kontrolér kurzu obsahuje pouze dvě funkce. První funkce, *index*, slouží k zobrazení pohledu s Livewire komponentou pro výpis kurzů. Druhá funkce, *show*, je určena pro zobrazení pohledu s detaily konkrétního kurzu.

```
class CourseController extends Controller
{
    //Zobrazení pohledu s výpisem kurzů
    public function index()
    {
        return view('courses.index');
    }
…
}
```

5.6.1.1 Výpis kurzů

Pro získávání a výpis kurzů se stará Livewire komponenta prostřednictvím metody *render()*, která se stará o načítání a zobrazení kurzů ve webové aplikaci. Nejprve je nutné uložit kurzy, ve kterých je uživatel registrován, a kurzy, které uživatel spravuje. Tyto dva seznamy jsou sloučeny do jednoho celkového seznamu. Nakonec metoda *render()* vrátí příslušný pohled, do kterého je předán kompletní seznam všech kurzů.

```
class Courses extends Component
{
…
    public function render()
    {
          // načtení kurzů, které uživatel spravuje
        $coursesAsAuthor = auth()->user()->coursesAsAuthor()->with('user')->get();

          // načtení kurzů, do kterých je uživatel přihlášen
        $coursesAsParticipant = auth()->user()->coursesAsParticipant()->with('user')->get();

          // spojení seznamů
        $allCourses = $coursesAsAuthor->concat($coursesAsParticipant);

          // vrácení pohledu
        return view('livewire.courses', [
            'allCourses' => $allCourses,
        ]);
    }
…
}
```

Následně v pohledu *livewire.courses* vypíšeme jednotlivé kurzy pomocí *foreach*.

```
@foreach($allCourses as $course)
…
@endforeach
```

Tento způsob vypisování a získávání dat využívá většina částí webové aplikace. Výhodou použití tohoto přístupu je, že při každé změně se automaticky data obnoví a překreslí se celá komponenta. Přitom nedochází ke znovunačtení celé stránky.

5.6.1.2 Vytvoření kurzu

Při zpracovávání formulářů pomocí komponenty Livewire je nutné v komponentě definovat proměnné a nastavit tzv. wire modely pro jednotlivé vstupy formuláře. Například u formuláře pro vytváření kurzu, který zpracovává název kurzu, jeho štítek a vybranou barvu, je třeba tento atribut u vstupů správně uvést. Pro definované proměnné lze nastavit validační pravidla, která budou aplikována na jednotlivé vstupy formuláře.

```
class Courses extends Component
{
    // Pravidla a proměnná pro název kurzu
    #[Validate('required', message: 'Please provide a post title')]
    #[Validate('min:3', message: 'This title is too short')]
    #[Validate('max:20', message: 'This title is too long')]
    public string $name;

    // Následují další proměnné a jejich pravidla
    …
}
```

Je klíčové, aby atribut *wire:model* měl stejný název jako proměnná v komponentě. Formulář pro vytváření nového kurzu s jednotlivými vstupy by vypadal následovně.

```
<form method="POST" wire:submit="store">
    @csrf
    …
       <label class="…">Název kurzu</label>
           <input type="text" wire:model="name" class="…">
           <div class="…">@error('name') {{ $message }} @enderror</div>

    <!--Následují další vstupy -->
    …
</form>
```

Atribut *wire:submit* specifikuje, která metoda bude zpracovávat formulář po jeho odeslání. Tato metoda nejprve zvaliduje data, vygeneruje nový unikátní kód kurzu a uloží nový kurz do databáze. V posledním kroku dojde k resetování vstupů. Pokud jsou vstupy zadány nesprávně, uživateli se zobrazí chybová hláška.

```
public function store()
{
    //Validace vstupů
    $validated = $this->validate([
        'name' => 'required|min:3',
        'description' => 'required|min:3',
        'color' => 'required',
    ]);

    // Generování kódu
    $validated['code'] = $this->generateCode();

    // Ukládání kurzu
    auth()->user()->coursesAsAuthor()->create($validated);


    // Resetování vstupů
    $this->reset('name', 'description');
}
```

5.6.2 Příspěvky a komentáře

Jak bylo zmíněno v návrhu, pro komentáře a příspěvky není využíván samostatný kontrolér. Pro výpis, vytváření, mazání a úpravy bude stačit jedna Livewire komponenta, přičemž výpis a vytváření probíhá podobně jako u kurzů.

5.6.2.1 Editace příspěvku a komentářů

Pro editování příspěvků nebo komentářů vytvoříme tlačítko s metodou, které zobrazí formulář pro editaci a uloží do proměnné zvolený příspěvek nebo komentář pro editaci. Tuto metodu pojmenuje např. *selectComment*. Jakou metodu má tlačítko zavolat definujeme atributem *wire:click*.

```
<x-dropdown-link-button wire:click="selectComment({{ $comment }})">
    Upravit
</x-dropdown-link-button>

// Proměnné pro editaci komentáře
public Comment $selectedComment;

// Proměnné pro text komentáře
public $editingCommentText;

public function selectComment(Comment $comment){
    $this->selectedComment = $comment;
    $this->editingCommentText = $comment->body;
    …
}
```

Po zavolání této funkce se uloží komentář, který editujeme a nastaví se jeho text do proměnné *edditingComentText*. Tuto proměnnou následně nastavíme ve formuláři u vstupu pro editaci atributem *wire:model* podobně jako u formuláře pro vytvoření kurzu. To zajistí, že se uživateli načte aktuální text komentáře. Po odeslání se zavolá funkce pro uložení komentáře.

```
public function editComment()
{
    // Kontrola, jestli uživatel má právo komentář upravovat
    auth()->user()->can('update', $this->selectedComment);
    if($this->selectedComment){
        // Validace vstupu
        $this->validate([
            'editingCommentText' => 'required|string|min:3|max:280',
        ]);

        // Ukládání komentáře
        $this->selectedComment->body = $this->editingCommentText;
        $this->selectedComment->save();

        …
    }
    …

}
```

Protože byla využita Livewire komponenta, není potřeba znovu načíst stránku a změněný text se rovnou objeví u zvoleného příspěvku nebo komentáře.

5.6.2.2 Mazání příspěvku nebo komentáře

Funguje podobně jako editace, ale s tím rozdílem, že tlačítko nevolá metodu pro výběr konkrétního příspěvku, ale rovnou funkci pro mazání, kde se předává příspěvek nebo komentář.

```
public function delete(Post $post)
{
    // Kontrola, jestli uživatel má právo příspěvek smazat
    $this->authorize('delete', $post);

    // Smazání příspěvku
    $post->delete();
    …
}
```

5.6.3 Materiály

Na rozdíl od příspěvků, materiály jsou spravovány jedním kontrolérem, který obsahuje funkci pro zobrazování pohledu s Livewire komponentou pro výpis. Tato metoda přijímá model kurzu, aby zobrazila odpovídající materiály. Livewire komponenta vypisuje a maže materiály podobným způsobem jako předešlé modely.

5.6.3.1 Nahrávání materiálů

Materiály se ukládají do předem určené složky. Složka je pojmenována kódem kurzu, což znamená, že každý kurz má svou vlastní složku pro materiály. V tomto případě se cesta skládá ze složky *materials/* a složky s kódem kurzu. Soubor je pak uložen do této složky na lokálním disku. Při vytváření materiálu se zkontroluje nahraný soubor, uloží se na disk a do proměnné se vloží cesta k tomuto nahranému souboru. Do databáze k vytvořenému materiálu se přiřadí cesta k souboru.

```
public function create(){
    …
    $validate = $this->validate();
    if($this->file){
        $courseFolderPath = 'materials/' . $this->course->code;
        $validate['file_path'] = $this->file->store($courseFolderPath, 'local');
    }

    //Následuje ukládání materiálu do databáze
}
```

5.6.3.2 Stahování materiálů

Pro stahování jednotlivých materiálů vytvoříme funkci, která přímá objekt materiálu. Funkce zkontroluje, jestli nahraný soubor existuje a pokud existuje, funkce pak vrací soubor ke stažení. Tato funkce je volána z tlačítka pro stažení, které je umístěno u každého materiálu.

```
public function download(Material $materialToDownload)
{
    if(Storage::disk('local')->exists($materialToDownload->file_path)){
        return Storage::download($materialToDownload->file_path,
        $materialToDownload->name);
    }
}
```

5.6.3.3 Změna viditelnosti

Podobně jako u mazání příspěvku, vytvoříme tlačítko, které zavolá funkci pro změnu viditelnosti studijního materiálu pro studenty. Tato funkce, která přijme ID materiálu, vypadá následovně.

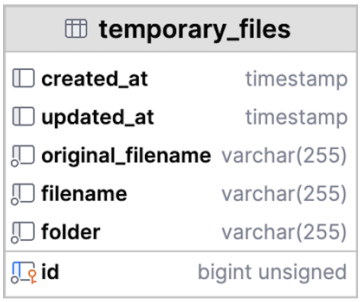
```
public function toggle($toggleId){

    $material = Material::find($toggleId);
    $material->update([
        'is_visible' => !$material->is_visible
    ]);

    …
}
```

5.6.4 Úkoly

Úkoly jsou spravovány kontrolérem, který zahrnuje funkce pro zobrazení, vytváření, úpravu a mazání úkolů. Pro odevzdávání úkolů je použit další kontrolér, který zpracovává nahrané soubory k úkolům. Kromě kontrolérů se pro určité části, jako je hodnocení úkolů, využívají Livewire komponenty.

5.6.4.1 Odevzdávání úkolů

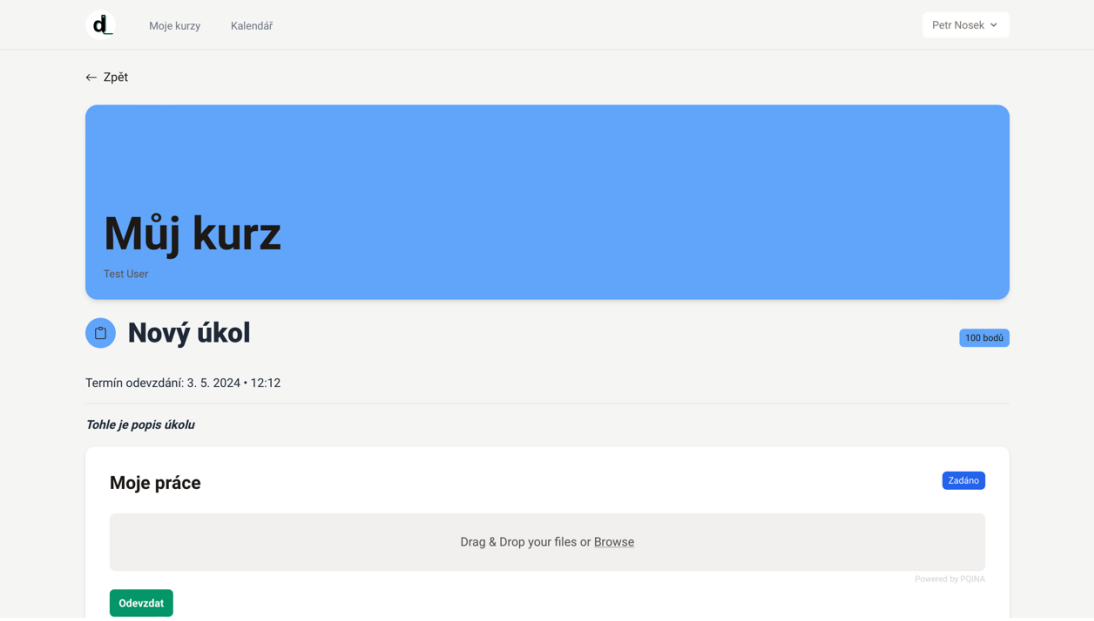
V rámci odevzdávání souborů k úkolům byla implementována javascriptová knihovna *FilePond*, která umožňuje uživatelům snadno nahrávat více souborů. Pro správnou funkčnost této javascriptové knihovny bylo nutné vytvořit databázovou tabulku pro dočasně uložené soubory, která ukládá informace o nahraných souborech do doby, než jsou přesunuty do trvalého úložiště. Vstup z formuláře obsahuje pole složek, které jsou poté zpracovány v kontrolérů pro odevzdávání úkolů.



Obrázek 26 – Tabulka temporary_files

Když uživatel odevzdává soubory k úkolu, kontrolér zpracuje formulář ve funkci *store()*. Nejprve se validuje, jestli pole souborů je skutečně pole. Následně se vytvoří nový záznam

v modelu *StudentSubmission* a přesunou se dočasné soubory do složky konkrétního úkolu. Každý přesunutý soubor je uložen do tabulky *student_submission_files* a dočasný soubor je smazán.



Obrázek 27 – Stránka s odevzdáním úkolů

5.6.4.2 *Hodnocení úkolů*

Hodnocení úkolu pracuje na podobném principu jako úprava příspěvku. Po kliknutí na tlačítko pro hodnocení se vybere odevzdání studenta a zobrazí se formulář pro ohodnocení úkolu. Po odeslání formuláře se v databázi aktualizuje počet bodů získaných za úkol.

```
public function rate()
{
    …

    $this->selectedStudent->submissions()->where('assignment_id', $this->assignment->id)->update([
        'points' => $this->points,
    ]);
    $this->reset('points');
    …
}
```

5.6.4.3 *Vyhledávání studentů*

V příslušném pohledu vytvoříme textové pole pro vyhledávání. V Livewire komponentě definujeme proměnnou *search* a nastavíme u atribut *wire:model.live.debounce.300ms="search"* u textového pole. Tento atribut zajistí, že se obsah Livewire komponenty aktualizuje po každé změně ve vyhledávacím poli s krátkým zpožděním, aby se snížila frekvence dotazů. Metodu *render()* poté upravíme tak, aby z databáze získávala studenty odpovídající zadanému textu.

```
<!-- Pole pro hledání -->
<input wire:model.live="search" type="text" class="…"/>

public function render()
{
    if($this->search != ""){
        // Hledání studentů podle zadaného textu
    }
    else{
        // Získávání všech studentů
    }

    return view('livewire.assignment-feedback', [
        …
    ]);
}
```

5.6.5 Testy

Správu testů probíhá prostřednictvím jednoho hlavního kontrolérů, který obsahuje funkce pro zobrazení seznamu testů, vytváření, zobrazení detailu testu, odstraňování a další akce související se zobrazením pohledu revizí, hodnocení a spuštěného testu. Pro vyplňování testu, editaci, úpravu a udělování hodnocení jsou využívány Livewire komponenty.

5.6.5.1 Vytváření testu

Při vytváření nového testu se funkce *saveTest()* stará o validaci vstupů, zpracování a ukládání dat do databáze. Pole *questions* umožňuje dynamické přidávání otázek, do kterého se ukládají jednotlivé otázky, jejich typ, název a u uzavřených otázek i jejich možnosti. Ve formuláři jsou využity pro vstupy atributy *wire:model.live* pro okamžité zobrazování změn, jako jsou změny typů otázek, možností a jejich případné odstranění.

```
@foreach ($questions as $index => $question)
        …
      <label class="…Typ otázky</label>
        <div class="…">
                <select wire:model.live="questions.{{ $index }}.type">
                        <option value="open">Otevřená</option>
                        <option value="multiple-choice">Uzavřená</option>
                </select>
        @if ($question['type'] === 'multiple-choice')
        <!—Vstupy pro uzavřené otázky -->
        @else
        <!—Vstupy pro otevřené otázky -->
      @endif
@endforeach
```

Po odeslání formuláře funkce *saveTest()* zkontroluje všechny vstupy a vloží všechny otázky a případně jejich možnosti do databáze. V případě chybných vstupů se zobrazí chybová hláška.

```
public function saveTest()
{
    $this->validate([
        // Validace pro test
    ]);
    $test = Test::create([
        // vkládávní testu do databáze
    ]);
    foreach ($this->questions as $questionData) {
        $question = $test->questions()->create([
            // vkládání otázek do databáze
        ]);
        // Pokud je otázka uzavřená a obsahuje možnosti, uložíme je
        if ($questionData['type'] === 'multiple-choice' &&
!empty($questionData['choices'])) {
            // vkládání možností do databáze
        }
    }
}
```

5.6.5.2 *Vyplňování testu*

Pro vyplnění testu je důležité získat všechny otázky a jejich možnosti, které uložíme do pole *questions*. Tyto otázky jsou následně zobrazeny podobným způsobem jako při tvorbě testu, s tím rozdílem, že nyní jednotlivé otázky obsahují vstupy pro odpovědi. Odpovědi uživatelů jsou průběžně ukládány do pole *answers*. Vstupy s odpověďmi využívají atribut *wire:model.live* pro okamžitou aktualizaci pole *answers* a atribut *wire:change* pro zavolání metody po změně hodnoty ve vstupu. Jakmile dojde ke změně, je zavolána metoda *saveAnswers*, která ukládá každou změnu odpovědi do databáze.

```
public function saveAnswer($questionId)
{
    // Vymažeme starou odpověď
    Answer::where('test_submission_id', $this->submission->id)
        ->where('question_id', $questionId)
        ->delete();
    $answer = $this->answers[$questionId];
    if (!empty($answer)) {
        if (is_array($answer)) {
           // Uložíme všechny vybrané možnosti uzavřené otázky

        } else {
            // Uložíme otevřenou odpověď
        }
    }
}
```

Spuštění testu zpracovává funkce *startTest()*, která kontroluje pravidla ohledně počtu pokusů a časového omezení. Na začátku funkce získává všechny pokusy testu z databáze pro daného studenta a zjistí, jestli některý z pokusů není ukončen. Pokud student vyčerpal maximální počet povolených pokusů a neexistují žádné neukončené pokusy, je přesměrován zpět na stránku testu. Pokud existuje nedokončený pokus (chybí čas ukončení), systém načte tento pokus a vypočítá zbývající čas na jeho dokončení. V případě, že žádný nedokončený pokus není, vytvoří se nový záznam pokusu.

Po dokončení vyplňování testu jsou odpovědi znovu uloženy a uzavřené otázky jsou vyhodnoceny. Pro každou otázku je vytvořen záznam v tabulce *answer_grades*, kde se uchovává bodové ohodnocení. U otevřených otázek je také vytvořen záznam, ale sloupec pro body zůstává prázdný, aby mohl být později ohodnocen učitelem.

Obrázek 28 – Stránka s vyplňováním testu

5.6.5.3 Hodnocení testu

V pohledu hodnocení se vypíše vybraný pokus studenta. Otázky jsou uloženy v poli *questions* a odpovědi studenta v poli *answers*. Struktura pro výpis testu je podobná té při vyplňování testu, ale místo vstupů pro odpovědi jsou zobrazeny vstupy pro ohodnocení každé otázky. Také jsou označeny správné a špatné odpovědi studenta. Vstup s body volá při každé změně funkci *saveGrade(),* která zkontroluje vstup a ukládá bodové hodnocení do databáze. Správci kurzu tak nemusí hodnocení ukládat manuálně, protože se vše ukládá automaticky po každé změně.

```
public function saveGrade(Question $question)
{
    // kontrola vstupu
    if ($this->grades[$question->id] < 0 || $this->grades[$question->id] >
$question->points) {
        …
    }
    // uložení hodnocení
    $grade = $this->submission->grades->firstWhere('question_id', $question->id);
    $grade->points_awarded = $this->grades[$question->id];
    $grade->save();
    ...
}
```

5.6.6 Kalendář

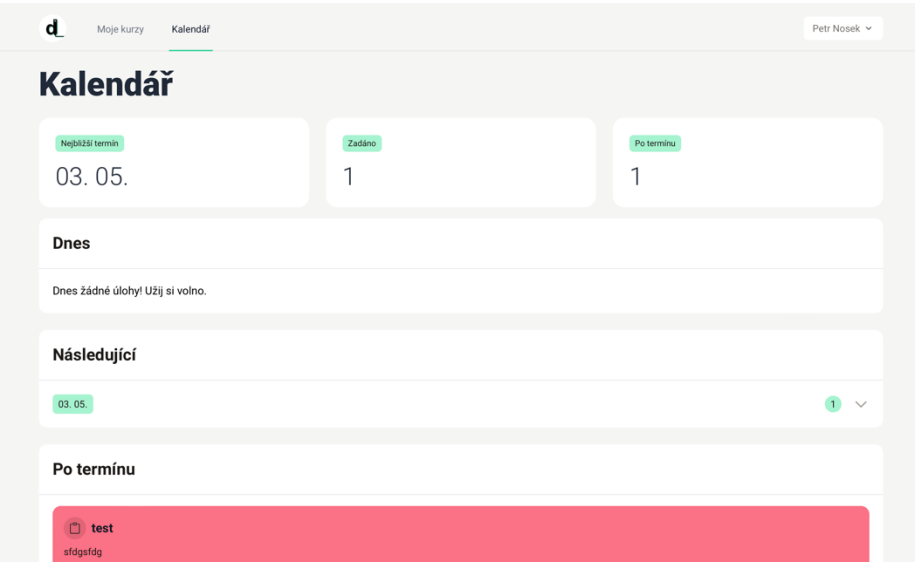
Kalendář je spravován dvěma kontroléry: jeden pro celkový kalendář a druhý pro kalendář kurz. Oba kontroléry obsahují funkci pro zobrazení pohledu, ve kterém je umístěna Livewire komponenta pro získání data.

5.6.6.1 Vypisování kalendáře

Funkce vyhledává úkoly a testy, které patří do kurzů, kde je uživatel zapsán, nebo které učí, a zároveň jsou vyloučeny ty, které uživatel již odevzdal.

Každý úkol a test je poté doplněn o dodatečné informace, jako je datum odevzdání, typ události (úkol nebo test) a informace, zda je uživatel učitelem daného kurzu. Tyto dva seznamy jsou následně sloučeny do jednoho seznamu, který je seřazen podle datumu.

U kalendáře v kurzu funguje vyhledávání událostí na stejném principu, akorát jsou vyhledány úkoly a testy, které patří do aktuálně zvoleného kurzu.



Obrázek 29 – Stránka s celkovým kalendářem

5.7 Vytváření rout

Na rozdíl od původní aplikace, v nové verzi aplikace je potřeba vytvořit jednotlivé routy v souboru *web.php* umístěném ve složce *routes*. Základní routy pro přihlášení, registraci a správu uživatelského profilu jsou již předdefinovány díky instalaci Laravel Breeze.

Pro kontroléry, které využívají základní CRUD operace, je možné využít metodu *resource()*, která automaticky generuje potřebné routy. Tato metoda definuje URL požadavky pro operace jako je zobrazení, vytvoření, aktualizace a mazání záznamů. Pokud však některé kontroléry nevyužívají všechny CRUD operace, můžeme použité funkce specifikovat pomocí metody *only(),* která vytvoří routy pouze pro vybrané funkce. Metodu *middleware()* použijeme u rout, které potřebují ověřit, zda je uživatel přihlášen. Následující příklad ukazuje, jak je definována routa pro kurz, která využívá metody pro zobrazení seznamu všech kurzů, detailu konkrétního kurzu, jeho editaci a aktualizaci dat. Zároveň vyžaduje, aby uživatelé byli přihlášeni.

```
Route::resource('course', CourseController::class)->only(['index','show',
'edit', 'update'])->middleware('auth');
```

V některých případech je nutné definovat další funkce, které nelze pokrýt metodou *resource()*. Pro tyto účely využíváme metodu *get()* pro získávání dat nebo zobrazování stránek a metodu *post()* pro odesílání dat formulářů na server. Pro lepší přehlednost lze routy pojmenovat pomocí metody *name(),* což usnadňuje odkazování na ně v aplikaci.

```
Route::get('/course/{course}/test/{test}/attempt', [TestController::class,
'attempt'])->name('test.attempt')->middleware('auth');
```

Při definování, která routa má být zavolána po odeslání formuláře, se používá atribut *action*. V tomto atributu lze specifikovat název routy pomocí metody *route(),* kde lze případně předat potřebné parametry. U odkazů se postupuje podobně, s tím rozdílem, že místo atributu *action* se využívá atribut *href*.

```
<form method="POST" action="{{ route('assignment.store', $course) }}"
…
</form>
```

5.8 Uživatelské prostředí

Uživatelské prostředí využívá již zmíněný šablonovací systém, který umožňuje vytváření různé komponenty. Ty mohou být opakovaně používány v různých částech webové aplikace. Existující komponenty lze nalézt ve složce *resources/views/components*, kde je možné

vytvářet další komponenty podle potřeby. Tyto komponenty jsou poté snadno použitelné v aplikaci.

```
<x-course-header :course="$course" href="{{route('course.index')}}"/>
```

Pokud je potřeba v komponentě pracovat s konkrétní proměnnou, můžeme je specifikovat pomocí *@props*. Například hlavička kurzu přijímá objekt kurzu a cestu, která určuje, kam má tlačítko vést pro vrácení zpátky. Tyto přijaté proměnné lze následně využívat stejně jako v ostatních částech projektu.

```
@props(
    ['course', 'href', 'hideback' => false]
)

<h1 class="…">{{ $course->name }}</h1>
```

Je možné také vytvořit šablonu pro celou stránku, která obsahuje kompletní strukturu HTML. Klíčovým prvkem je použití proměnné *$slot*, která určuje místo, kam bude vkládán přijatý obsah. Tímto způsobem eliminujeme opakující se strukturu HTML a případné změny se provedou na všech stránkách, které tuto šablonu využívají.

```
<html>
    <head>
        …
    </head>
    <body class="…">
        <!-- Obsah stránky -->
        <main>
          {{ $slot }}
        </main>
    </body>
</html>
```

V jednotlivých pohledech lze využít tuto šablonu stránky následovně.

```
<x-app-layout>
      <!-- Obsah stránky -->
</x-app-layout>
```

5.9 Zabezpečení aplikace

Tato kapitola popisuje bezpečnostní prvky implementované v nové verzi aplikace. Základními prvky jsou autentizace uživatelů, validace vstupů a kontrola, jestli uživatelé mají právo k provádění specifických akcí.

5.9.1 Policies

Policies v Laravel framework slouží pro stanovení pravidel, které určují, co mohou uživatelé v aplikaci dělat. Policies jsou užitečné pro kontrolu přístupu k modelům a jejich operacím, jako je zobrazení, úprava, smazání a další. V novém projektu se policies využívají především pro kontrolu, jestli např. uživatel má právo vytvářet v kurzu nové testy nebo jestli je může opravovat. Pro vytvoření nového policy souboru slouží příkaz Artisan. Vytvořené soubory polices lze najít ve složce *app/policies*.

```
php artisan make:policy PostPolicy --model=Post
```

Po úspěšném vytvořením nového policy souboru je nutné tuto policy zaregistrovat v souboru *app/Providers/AuthServiceProvider.php*. V tomto souboru se nachází pole, které mapuje modely a jejich vytvořené policies. Je vhodné pro každý model vytvořit svůj vlastní policy soubor.

```
protected $policies = [
    \App\Models\Post::class => \App\Policies\PostPolicy::class,
];
```

Ve vytvořeném souboru jsou již předdefinované funkce pro základní operace jako zobrazení, smazání a úpravy. Například funkce, která kontroluje, zda je uživatel autorem příspěvku a má právo jej smazat, by mohla vypadat následovně.

```
public function delete(User $user, Post $post): bool
{
    return $post->user_id == $user->id;
}
```

Pro vyvolání této kontroly stačí před začátkem mazání příspěvku zavolat metodu *authorize()*.

```
public function delete(Post $post)
{
    // Kontrola, jestli uživatel má právo příspěvek smazat
    $this->authorize('delete', $post);

    …
}
```

Kontrolu práva lze také kontrolovat v samotném blade souboru. Příkladem je zobrazení tlačítka pro vymazání příspěvku, pokud je uživatel jeho autorem.

```
@can('delete', $post)
    <button>Smazat příspěvek</button>
@endcan
```

5.9.2 Validace dat

Validace dat na straně serveru je jedna z nejzákladnějším bezpečnostních prvků. Laravel framework poskytuje nástroje pro jednoduchou validaci dat z formulářů, včetně možnosti zobrazování chybových hlášení uživatelům. Validace dat od uživatelů probíhá v kontrolorech jednotlivých modelů. Validace v Laravel framework zahrnuje ověření, zda je vstup vyplněn (*required*), kontrolu délky řetězce (*min, max*) nebo ověření požadovaného formátu vstupu (*string*, *integer*, *date*). Například při přijetí dat z formuláře pro tvorbu úkolu lze použít metodu *validate* takto:

```
public function store(Request $request, Course $course)
{
    $validate = $request->validate([
        'title' => 'required|string|max:255',
        'description' => 'max:1000',
        'date' => 'required|date',
        …
    ]);
…
}
```

Validace v Laravelu zabrání vstupu neplatných dat do systému a zároveň informuje uživatele o chybně zadaných dat. Chybové hlášky lze zobrazit pod vstupem s názvem proměnné, kterou kontrolujeme pomocí bloku *error()* a *enderror*.

```
@error('title')
    <x-input-error :messages="$message" class="mt-2"></x-input-error>
@enderror
```

5.9.2.1 Validace v Livewire

Validace v Livewire funguje podobně jako v Laravelu, ale umožňuje validovat vstupy přímo v Livewire komponentě. To znamená, že jakmile uživatel vyplňuje formulář, validace se provádí na pozadí a uživatel okamžitě vidí chybové zprávy bez nutnosti odeslat celý formulář. Toto se provádí pomocí metod *validate()*, která funguje stejně jako validace v Laravelu. Zároveň Livewire podporuje použití stejných validačních pravidel jako v Laravel framework. Chybové zprávy se vypisují v Blade šablonách stejným způsobem jako v předešlém případě.

6 SROVNÁNÍ A HODNOCENÍ

Tato kapitola se zaměřuje na srovnání původní a nové verzi aplikace. Rozdíly mezi oběma verzemi jsou popsány z hlediska udržitelnosti kódu, zabezpečení a uživatelského prostředí.

6.1 Rozšiřitelnost a udržitelnost kódu

Stará verze aplikace byla napsána v čistém PHP bez využití jakéhokoliv moderního frameworku nebo návrhového vzoru. Tento nedostatek vede k obtížnému pochopení struktury samotného projektu, což komplikuje případnou implementaci nových funkcí, tak opravy chyb v projektu. Rozšiřování takové aplikace nebo vytváření nových komponent často vyžaduje začít zcela od základů.

Na druhou stranu, nová verze aplikace byla vytvořena s použitím frameworku Laravel, který implementuje MVC architekturu. Tato architektura odděluje logiku aplikace od uživatelského rozhraní a není vše zakomponováno v jednom souboru, jak tomu bylo u původní verze. Proto je implementace nových funkcí mnohem jednodušší, což výrazně usnadňuje orientaci v kódu pro vývojáře. Navíc, díky použití Eloquent ORM v Laravelu není nutné explicitně zadávat SQL příkazy a je možné pracovat s více typy databází. Také Eloquent ORM poskytl jednoduchý způsob, jak interagovat s datovými modely.

Porovnání staré a nové verze aplikace jasně ukazuje, že přechod na Laravel framework výrazně zlepšuje udržovatelnost a rozšiřitelnost projektu.

6.2 Zabezpečení aplikace

Původní aplikace využívá vlastní jednoduché řešení pro přihlašování a registraci. Navíc kontrola, zda je uživatel přihlášen, se provádí na každé stránce zvlášť. Dalším problémem je nevalidování všech vstupů formulářů, což může vést k potenciálním útokům.

Na druhé straně, nová verze aplikace využívá Laravel Breeze, který obsahuje nástroje pro autentizace a autorizaci. V novém projektu jsou také ošetřeny všechny vstupy formulářů a zároveň byl implementován *middleware* pro kontrolu přihlášení uživatelů.

6.3 Uživatelské prostředí

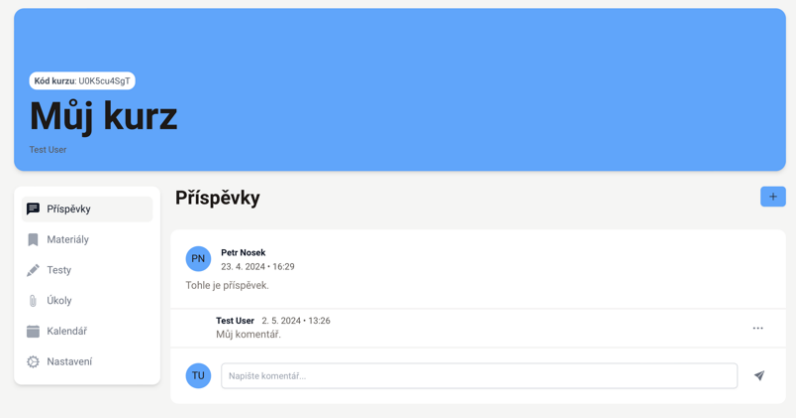
Uživatelské prostředí obou aplikací jsou dobře zpracované. Původní aplikace sice využívá dva CSS frameworky, což není úplně dobré řešení, ale uživatelé tento nedostatek nepoznají. I přes tento nedostatek je původní aplikace dobře strukturovaná. Obě verze aplikace

poskytují responzivní design a podporují světlý i tmavý motiv. Jedním z problémů původní verze je, že u některých formulářů se nezobrazují chybové hlášky o špatně zadaných vstupech.

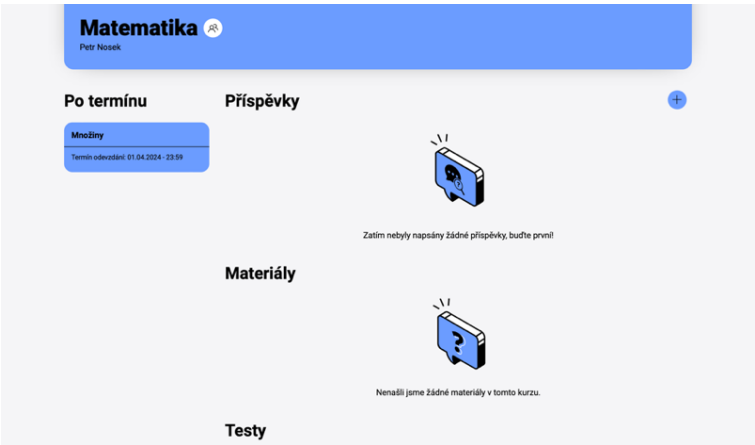
Uživatelské prostředí nové aplikace vychází ze vzhledu původní aplikace, ale jednotlivé části aplikace jsou rozděleny do více částí, aby na jedné stránce nebylo zobrazeno mnoho informací. Příkladem je přehled kurzu, kde na rozdíl od původní verze, kde byly příspěvky, materiály, úkoly a testy zobrazeny na jedné stránce, je v nové aplikaci každá z těchto částí zobrazena na samostatné stránce.

6.3.1 Porovnání uživatelského prostředí

Stránka s přehledem kurzu jsou velmi podobné, ale nová verze aplikace rozděluje jednotlivé sekce kurzu do jednotlivých záložek, zatímco původní aplikace zobrazuje všechny tyto části na jedné stránce. Nyní je stránka kurzu přehlednější a uživatelé se mohou v kurzu lépe orientovat.

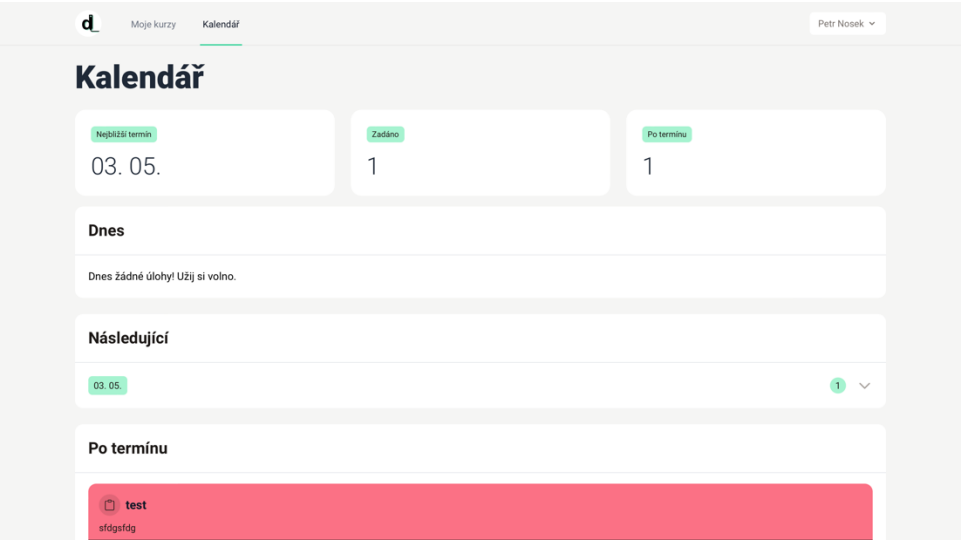


Obrázek 30 – Přehled kurzu nové aplikace

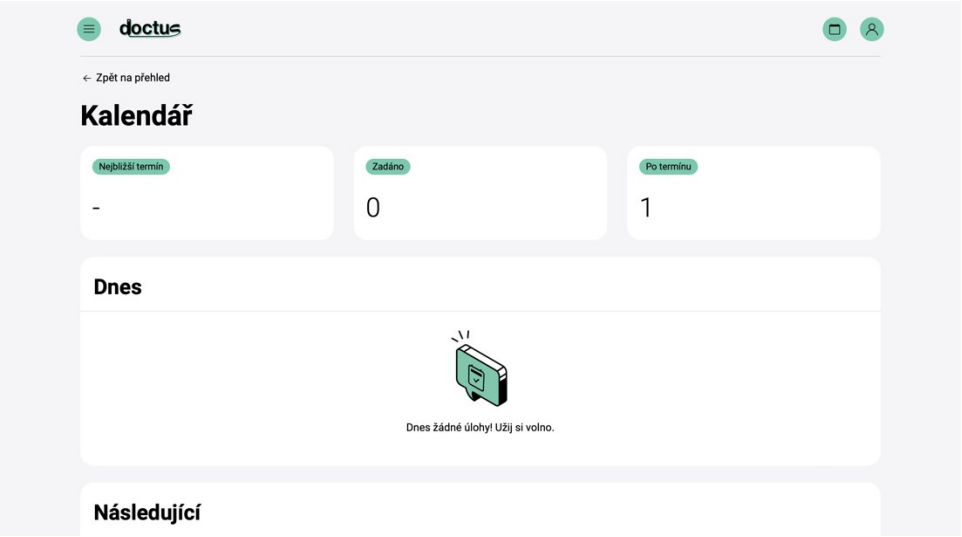


Obrázek 31– Přehled kurzu původní aplikace

Stránka s celkovým kalendářem jsou totožné a nabízí stejný přehled blížících se úkolů a testů. Rozdílem je kalendář v kurzu nové aplikace, který vychází z celkového kalendáře, ale zobrazuje pouze úkoly a testy v kurzu. Původní kalendář v kurzu nerozděluje dnešní události, následující a po termínu. Toto řešení uživatelům poskytuje lepší přehled o blížících se událostech v kurzu.



Obrázek 32 – Celkový kalendář nové aplikace



Obrázek 33 – Celkový kalendář původní aplikace

6.4 Přínosy převodu

Díky využití MVC architektury je kód snadněji rozšiřitelný a lépe se v něm orientuje. Využití této architektury zrychluje vývoj nových funkcí a opravy různých chyb projektu. Díky použití *policies* je snazší definovat uživatelské oprávnění a nemůže dojít k neoprávněnému

zobrazení nebo aktualizaci dat. Samotný vývoj nové verze aplikace byl plynulejší a ušetřil mnoho času. Původní aplikace vznikala nejméně půl roku, zatímco nová verze vznikla za méně než dva měsíce. Nakonec, framework Laravel navíc nabízí různé funkce a metody, které výrazně usnadňují vývoj webových aplikací.

ZÁVĚR

Tato bakalářská práce byla zaměřena na převod webové aplikace z čistého PHP na projekt vytvořený pomocí frameworku Laravel.

Úvod teoretické části práce se věnoval struktuře a rozdělní obecných návrhových vzorů pro pochopení fungování frameworků, které návrhové vzory využívají. Hlavní část se soustředila na samotný framework Laravel, kde byla nejprve stručně zmíněna jeho historie a následně popsána architektura tohoto frameworku. V práci bylo také vysvětleno fungování návrhového vzoru MVC, který tvoří základ Laravelu. Bylo popsáno fungování práce s databází, migracemi a případné naplňování databáze testovacími daty. Teoretická část také popsala základní nástroje a balíčky frameworku Laravel.

Praktická část práce popsala hlavní funkce původní aplikace, strukturu kódu a databáze, přičemž byly zmíněny hlavní problémy spojené s tím, že původní aplikace nevyužívala žádný framework ani návrhový vzor. Dále v praktické části byly stanoveny klíčové cíle a požadavky nové verze aplikace a následně vybrány technologie a nástroje pro tvorbu aplikace. Byla navržena nová architektura aplikace a popsány jednotlivé tabulky a modely společně s jejich vztahy. Dále byla podrobně popsána instalace Laravel frameworku pomocí vývojového prostředí Laravel Herd a dalších souvisejících nástrojů. Praktická část také popsala vytváření modelů, vztahů a migrací vyplývající z návrhu aplikace. Dále se věnovala funkcionalitám aplikace, jejich kontrolérům, logice a popsala zabezpečení převedené aplikace.

V poslední kapitole bylo provedeno srovnání původní a převedené aplikace. Byly popsány rozdíly v zabezpečení, uživatelském prostředí, rozšiřitelnosti a udržitelnosti kódu.

Převod na tento moderní framework přinesl zlepšení v oblasti správy kódu, což značně usnadňuje implementaci nových funkcí a zvyšuje přehlednost a čitelnost kódu. Navíc, nově převedená aplikace klade větší důraz na zabezpečení, kdy oproti původní aplikaci byla definována uživatelská práva a všechny uživatelské vstupy ošetřeny. Kromě toho, nová verze aplikace využívá architekturu MVC, kterou Laravel implementuje. Tato architektura oddělila logiku aplikace od uživatelského rozhraní, na rozdíl od původní verze, kde bylo vše zahrnuto do několika souborů. Díky použití Eloquent ORM v Laravelu nebylo nutné explicitně zadávat SQL příkazy, což zjednodušilo práci s databází. Navíc, na rozdíl od původní verze, která byla vázána na konkrétní typ databáze (MySQL), Eloquent ORM umožnil pracovat s více typy databází.

SEZNAM POUŽITÉ LITERATURY

[1] *What's a design pattern?*, © 2014-2024. Online. Refactoring.guru. Dostupné z: https://refactoring.guru/design-patterns/what-is-pattern. [cit. 2024-03-02].

[2] ANAND, Adarsh; SINGH, Jagvinder; VAIN, Juri; TABREZ QUASIM, Mohammad a JOHRI, Prashant (ed.), 2022. *System Assurances Modeling and Management*. Nizozemsko: Elsevier Science. ISBN 9780323902410.

[3] JAIN, Aaina, 2020. *A Series on Design Patterns*. Online. Medium. Dostupné z: https://medium.com/@aainajain/a-series-on-design-patterns-b6302a6e19cd. [cit. 2024-03-02].

[4] *Design Patterns*, © 2007-2024. Online. Sourcemaking.com. Dostupné z: https://sourcemaking.com/design_patterns. [cit. 2024-03-08].

[5] BROTHERTON, Claire, 2023. *The Most Popular PHP Frameworks to Use*. Online. Kinsta. Dostupné z: https://kinsta.com/blog/php-frameworks/. [cit. 2024-03-01].

[6] OTWELL, Taylor, © 2011-2024. *The PHP Framework for Web Artisans*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/. [cit. 2024-05-03].

[7] *Laravel - History*, © 2009-2024. Online. W3schools.in. Dostupné z: https://www.w3schools.in/laravel/history. [cit. 2024-03-08].

[8] CROZAT, Benjamin, 2024. *A complete history of Laravel's versions (2011-2024)*. Online. Nenalezený vydavatel. Dostupné z: https://benjamincrozat.com/laravel-versions. [cit. 2024-03-15].

[9] HANIF, Fikri, 2023. *Understanding the MVC Architecture in Laravel: A Comprehensive Guide*. Online. Medium. Dostupné z: https://fkrihnif.medium.com/understanding-the-mvc-architecture-in-laravel-a-comprehensive-guide-8f620cc139b6. [cit. 2024-03-10].

[10] HARTINGER, David, © 2024. *MVC architektura*. Online. ItNetwork. Dostupné z: https://www.itnetwork.cz/mvc-architektura-navrhovy-vzor. [cit. 2024-03-10].

[11] OTWELL, Taylor, © 2011-2024. *Directory Structure*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/structure. [cit. 2024-03-11].

[12] STAUFFER, Matt, 2019. *Laravel: up & running*. Second edition. O'Reilly Media. ISBN 9781492041160.

[13] OTWELL, Taylor, © 2011-2024. *Eloquent: Getting Started*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/eloquent. [cit. 2024-03-11].

[14] OTWELL, Taylor, © 2011-2024. *Artisan Console*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/artisan. [cit. 2024-03-12].

[15] OTWELL, Taylor, © 2011-2024. *Database: Migrations*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/migrations. [cit. 2024-03-11].

[16] *Laravel Tinker*, © 2011-2021. Online. Javatpoint.com. Dostupné z: https://www.javatpoint.com/laravel-tinker. [cit. 2024-03-11].

[17] AHMED, Shahzeb, 2022. *An Introduction to Routing in Laravel*. Online. The Official Cloudways Blog. Dostupné z: https://www.cloudways.com/blog/routing-in-laravel/. [cit. 2024-03-15].

[18] OTWELL, Taylor, © 2011-2024. *Routing*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/routing. [cit. 2024-03-15].

[19] OTWELL, Taylor, © 2011-2024. *HTTP Responses*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/responses. [cit. 2024-03-16].

[20] OTWELL, Taylor, © 2011-2024. *Blade Templates*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/blade. [cit. 2024-03-20].

[21] YASSER, Noor, 2023. *Demystifying Laravel's Schema Builder: Everything You Need to Know*. Online. Medium. Dostupné z: https://medium.com/@noor1yasser9/demystifying-laravels-schema-builder-everything-you-need-to-know-c4f65c010300. [cit. 2024-03-27].

[22] OTWELL, Taylor, © 2011-2024. *Database: Seeding*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/seeding. [cit. 2024-03-27].

[23] BONISTEEL, Steve, 2023. *How To Generate and Use Fake Records with Laravel Model Factories*. Online. Kinsta. Dostupné z: https://kinsta.com/blog/laravel-model-factories/. [cit. 2024-03-27].

[24] OTWELL, Taylor, © 2011-2024. *Eloquent: Factories*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/. [cit. 2024-03-28].

[25] OTWELL, Taylor, © 2011-2024. *Starter Kits*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/. [cit. 2024-03-28].

[26] APRODILLES, 2024. *Authentication in Laravel Using Breeze*. Online. Kinsta. Dostupné z: https://kinsta.com/blog/laravel-breeze/. [cit. 2024-03-29].

[27] KLIMČÍK, Jaroslav, 2021. *Laravel Breeze*. Online. LaravelBlog.cz. Dostupné z: https://laravelblog.cz/clanek/laravel-breeze. [cit. 2024-03-30].

[28] ONYEBUEKE, Emmanuel, 2023. *An Overview Of How Livewire Works*. Online. Medium. Dostupné z: https://medium.com/@developer.olly/an-overview-of-how-livewire-works-85395746d10a. [cit. 2024-03-30].

[29] MEDIA, Pbc, 2023. *Laravel Livewire: An In-Depth Exploration of Empowering Interactive Web Applications*. Online. Medium. Dostupné z: https://pbcmedia.medium.com/laravel-livewire-an-in-depth-exploration-of-empowering-interactive-web-applications-7eb6d485492. [cit. 2024-03-30].

[30] OTWELL, Taylor, © 2011-2024. *Laravel Telescope*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/telescope. [cit. 2024-04-04].

[31] KUNDU, Sauvik, 2023. *Unveiling Laravel Telescope: Boost Your Web Development Experience*. Online. Medium. Dostupné z: https://sauvikkundu.medium.com/unveiling-laravel-telescope-boost-your-web-development-experience-8dc3f2b1ddd7. [cit. 2024-04-04].

[32] *Laravel Development Environment: A Complete Introduction*. Online. Nenalezený vydavatel. Dostupné z: https://www.usenimbus.com/post/laravel-development-environment-a-complete-introduction. [cit. 2024-05-01].

[33] OTWELL, Taylor. *Laravel Homestead*. Online. The PHP Framework For Web Artisans. Dostupné z: https://laravel.com/docs/10.x/homestead. [cit. 2024-05-01].

[34] REDMOND, Paul a REDMOND, Paul. *Laravel Herd*. Online. Laravel News. Dostupné z: https://laravel-news.com/laravel-herd. [cit. 2024-05-01].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MVC Model-View-Controller.

ORM Object–Relational Mapping.

PHP Hypertext Preprocessor.

HTML Hypertext Markup Language.

CRUD Create Read Update Delete.

HTTP Hypertext Transfer Protocol.

UI User Interface

CSS Cascading Style Sheets.

NPM Node Package Manager.

SQL Structured Query Language.

CLI Command Line Interface.

API Application Programming Interface.

AJAX Asynchronous JavaScript and XML

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: PŘILOŽENÉ CD

- Text bakalářské práce ve formátu PDF
- Zdrojové kódy převedené webové aplikace
- Návod pro instalaci projektu