FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání bakalářské práce

| | |
|---|---|
| **Název:** | Kvíz – státní rozpočet |
| **Student:** | Vojtěch Sillik |
| **Vedoucí:** | PhDr. Ing. Tomáš Evan, Ph.D. |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Webové a softwarové inženýrství, zaměření Softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | do konce letního semestru 2022/2023 |

## Pokyny pro vypracování

Navrhněte a implementujte webovou aplikaci, která uživateli nabídne sadu otázek, při jejichž vyplňování bude interaktivně zobrazen dopad do státního rozpočtu. Např. se ukáže cena navýšení výdajů pro armádu, valorizace důchodů apod. Na začátku bude uživateli položeno několik sociologických otázek pro následné vyhodnocování například na jeho věk, vzdělání, bydliště apod. a na konci zobrazen celkový výsledek hospodaření při realizaci požadovaných kroků. Interně proběhne vyhodnocení toho, jak které skupiny uživatelů odpovídají na jednotlivé otázky.

1. S vedoucím práce formalizujte požadavky na webovou aplikaci.
2. Diskutujte a zvolte vhodnou implementační platformu.
3. Metodami softwarového inženýrství proveďte návrh a implementaci.
4. Aplikaci vhodně otestujte a dokumentujte.

---

*Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 14. prosince 2021 v Praze.*

Bakalářská práce

# KVÍZ – STÁTNÍ ROZPOČET

**Vojtěch Sillik**

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: PhDr. Ing. Tomáš Evan, Ph.D.
12. května 2022

# Obsah

# Seznam obrázků



# Seznam tabulek



# Seznam výpisů kódu

*Děkuji vedoucímu práce panu PhDr. Ing. Tomáši Evanovi, Ph.D. a panu Ing. Marku Sušickému za pomoc, rady a konzultace v průběhu tvorby této práce. Dále děkuji rodině, přítelkyni a přátelům za podporu v průběhu studia a při psaní této práce.*

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 12. května 2022 .....

# Abstrakt

Cílem této práce je tvorba kvízové aplikace pro simulaci státního rozpočtu. Při tvorbě analyzuji současný stav řešení a provádím průzkum vhodných dostupných technologií pro implementaci. Následně provádím návrh uživatelského prostředí a databáze.

Výsledkem práce je webová aplikace, která na základě odpovědí na kvízové otázky provádí simulaci státního rozpočtu. V administrační části je možnost vyplnění údajů pro dotazník, počátečního stavu simulace, definování otázek a jejich odpovědí včetně vlivu odpovědí na stav simulace. Hlavní technologie pro implementaci jsou Laravel pro programovací jazyk PHP a React pro programovací jazyk JavaScript. Pro design aplikace je použita technologie Boostrap pro Cascading Stylesheets.

Výsledek této práce umožňuje vlastní provozování této aplikace bez poplatků. Na závěr práce jsou popsány možnosti rozšíření aplikace o další funkce.

**Klíčová slova** webová aplikace, simulace státního rozpočtu, PHP, JavaScript, Laravel, React

# Abstract

The aim of this thesis is to create a quiz application for the simulation of the government budget. I analyse suitable technologies for implementation of this application. Subsequently, I design the user interface and database.

The result of the thesis is a web application which, based on the answers to the quiz questions, performs a government budget simulation. It is possible to fill in data for the questionnaire, the initial state of the simulation, defining the questions and their answers, including the influence of the answers on the state of the simulation. The main technologies used for implementation are Laravel for the PHP programming language and React for the JavaScript programming language. Bootstrap is used for the design of the user interface.

The result of this thesis allows the actual operation of this application free of charge. At the end of the thesis are described the possibilities of extending the application functionalities.

**Keywords** web application, government budget simulation, PHP, JavaScript, Laravel, React

# Seznam zkratek

| | |
|---|---|
| API | Application Programming Interface |
| CSRF | Cross-Site Request Forgery |
| CSS | Cascading style sheets |
| HDP | Hrubý domácí produkt |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| JS | JavaScript |
| JSX | JavaScript XML |
| JSON | JavaScript Object Notation |
| PHP | PHP: Hypertext Preprocessor |
| REST | Representational state transfer |
| SASS | Syntactically Awesome Stylesheet |
| SPA | Single page application |
| SQL | Structured Query Language |
| XML | Extensible Markup Language |

# Úvod

Státní rozpočet je velice důležitým tématem v dnešní společnosti, je předmětem debat politiků, novinářů, obyvatel... Státní rozpočet ovlivňuje úplně každého obyvatele státu, bez ohledu na to, o jaký stát se jedná. Je důležité usilovat o udržitelný stav rozpočtu, což se ne vždy daří, jelikož občané velkou mírou mohou vyvíjet tlak na politiky, a tedy prosadit změnu, která nemá pozitivní dopad na udržitelnost rozpočtu. Pro vyrovnání rozpočtu se musí takovéto rozhodnutí kompenzovat, například zvýšením příjmů pomocí zvýšení daní, což se ale zajisté nebude líbit skupinám, kterých se toto dotkne. Tyto důsledky nemusí být vždy vidět v očích široké veřejnosti. Také některé návrhy politiků sice mohou znít dobře pro politickou kampaň, ale ve výsledku mohou vést ke zbytečnému zadlužení státu.

Výsledek práce je určen pro širokou veřejnost, protože je důležité, aby si lidé mohli zkusit nasimulovat své rozhodnutí ve věcech státního rozpočtu a viděli dopady těchto rozhodnutí, případně si mohou ověřit, jak reálné jsou některé návrhy politiků. Tímto si také mohou ověřit svou finanční gramotnost. Z těchto důvodů jsem si vybral téma práce tvorbu kvízové webové aplikace pro státní rozpočet.

Cílem této práce je vývoj webové aplikace, která provede simulaci státního rozpočtu založenou na odpovědích na kvízové otázky. Aplikace bude umožňovat nastavení počátečního stavu simulace, definování otázek a odpovědí k ním. Před vyplněním kvízu bude uživatel vyzván k vyplnění informací o sobě – dosažené vzdělání, věk, kraj bydliště a preferenci politické strany, aplikace bude umožňovat tyto údaje také nastavit.

V kapitole 1 se zabývám analýzou pojmů týkajících se státního rozpočtu, rešerší existujících řešení, stanovením požadavků na aplikaci a v neposlední řadě analýzou vhodných technologií pro implementaci této aplikace. V kapitole 2 se zabývám návrhem architektury pro aplikaci, výběrem technologií na základě analýzy vhodných technologií, návrhem uživatelského prostředí, návrhem databáze a také návrhem základních ukázkových dat pro aplikaci. V kapitole 3 se zabývám potřebným nastavením prostředí pro zprovoznění aplikace při vývoji, tvorbou jednotlivých částí aplikace a úskalím při implementaci. V kapitole 4 se věnují otestování a dokumentaci vyhotovené aplikace.

# Kapitola 1

# Analýza

*V této kapitole se věnuji analýze domény problému státního rozpočtu, jelikož tuto analýzu využiji při návrhu řešení aplikace, dále se věnuji průzkumu již existujících aplikací na trhu pro simulaci státního rozpočtu, analýze uživatelských požadavků, případů užití a na závěr kapitoly se věnuji prozkoumání vhodných technologií pro vývoj webové aplikace.*

## 1.1 Veřejné finance

Dle [1] jsou základem ekonomie peněžní vztahy mezi alespoň dvěma subjekty. označovány jsou jako finance. Živelnost znamená, že tyto tržní transakce nejsou nikým centrálně plánovány nebo organizovány a jsou důsledkem mnoha svobodných rozhodnutí. Tržní vztahy subjektů jsou dobrovolné. Právě živelnost a dobrovolnost jsou hlavními rysy tržních vztahů a jsou typické pro finance domácností a podnikové. Veřejných financí, tedy těch peněžních vztahů, kterých se účastní stát, se tyto rysy netýkají, platí pro ně principy nenávratnosti, neekvivalence a nedobrovolnosti.

Princip nenávratnosti znamená, že plátce finanční částky nemá nárok na jakoukoliv protislužbu, nebo navracení (případně reklamování) této částky. Principem nedobrovolnosti se rozumí nucenost finanční vztahy ve veřejných financí na základě daných zákonů, při porušení těchto zákonů hrozí sankce. Podíl odchozích plateb občana a příchozích plateb od státu zpravidla není vyrovnaný, označuje se jako princip neekvivalence. [1, 2]

### 1.1.1 Normativní pohled na veřejné finance

Normativní pohled je založen na zhodnocení výsledků, na základě určených kritérií, je to srovnání aktuálního stavu s ideálním. Tento pohled na základě teorie tržního selhání stanovuje základní funkce veřejných financí – alokační, redistribuční a stabilizační. Funkce alokační je shromažďování financí na realizaci veřejných statků. Funkce redistribuční snižuje nerovnoměrnost důchodů a bohatství ve společnosti. Funkce stabilizační má za cíl stabilizovat ekonomiku státu s pomocí finanční stimulace. [1, 3]

### 1.1.2 Nenormativní pohled na veřejné finance

Nenormativní (pozitivní) pohled je založen na faktické zhodnocení jevů, bez dalšího subjektivního zhodnocení následků, má za cíl zachovat nezávislost tohoto hodnocení. Vychází z teorie veřejné volby. [1, 3]

## 1.2 Fiskální politika

Fiskální politika je využívání výdajů státního rozpočtu, daně, cla apod. Původně fiskální politika měla za cíl shromažďovat finance pouze pro pokrytí potřeb státu. Státní rozpočet měl být neutrální a neovlivňovat trh. Nyní má fiskální politika spíše aktivní podobu, kdy ovlivňuje trh svými zásahy. Tato aktivní funkce je předmětem sporů mezi keynesovsky a neoklasicky zaměřenými ekonomy. [4]

Dle [3, 5] je fiskální politika nástrojem pro realizaci programových prohlášení vlády, a ke udržení stabilního ekonomického vývoje státu. Za základní funkce fiskální politiky se dle [3] považují právě funkce alokační, redistribuční a stabilizační.

### 1.2.1 Typy fiskální politiky

Rozlišují se dva základní typy fiskální politiky – expanzivní a restriktivní. Expanzivní politika má za cíl růst agregátní poptávky (souhrnu výdajů všech subjektů v ekonomice), snaží se o růst soukromých investic a spotřeby (například s pomocí snižování daní nebo zvyšováním účelných dotací), tato politika má za cíl zvýšení produktu a snížení nezaměstnanosti, aplikuje se v době, kdy je ekonomika v recesi (je pod svými možnostmi). Restriktivní politika je opak expanzivní politiky, zaměřuje se na omezení, případně i snížení agregátní poptávky, používá se, když je ekonomika na hranici svých možností. [5, 3]

### 1.2.2 Nástroje fiskální politiky

Dle [5] využívá fiskální politika k plnění svých cílů různé nástroje. Vláda může vytvořit zvýšenou poptávku pomocí například investic nebo daní. Nástroje se mohou dělit dle charakteru na diskreční (vědomá) opatření – tyto opatření reagují na krátkodobé výkyvy ukazatelů ekonomiky, tyto opatření se realizuji jednorázově například na základě souhlasu Poslanecké sněmovny, řadí se zde například změny daňových sazeb. Dalším druhem opatření dle charakteru jsou vestavěné stabilizátory, tyto opatření působí automaticky, aby stabilizovaly ekonomiku, patří zde například pojištění pro případ nezaměstnanosti. [5, 4]

### 1.2.3 Neoklasický a keynesiánský pohled na fiskální politiku

Zastánci neoklasického pohledu na fiskální politiku zastávají názor, že zásahy státu nejsou vždy nutné, že trh je schopen regulovat sám sebe, a tím ekonomika dosáhne dlouhodobé rovnováhy. Dle zastánců keynesiánského pohledu na fiskální politiku jsou potřeba zásahy ze strany státu, pro dosažení této rovnováhy. [4]

## 1.3 Státní rozpočet

Státní rozpočet je plán hospodaření státu s financemi na období určitého rozpočtového roku, je hlavní součástí veřejných financí, obsahuje souhrn příjmů a výdajů definovaných v zákoně stanovující státní rozpočet pro daný rozpočtový rok. Rozdíl příjmů a výdajů se nazývá saldo, pokud je kladný, poté se označuje jako přebytek, pokud je záporný, tak se označuje jako schodek. Příjmy a výdaje se dělí do skupin, zvaných kapitoly, určené jsou zákonem. [6, 7]

### 1.3.1 Příjmy státního rozpočtu

Dle [8, 9] je nejčetnější dělení příjmů státního rozpočtu na daně, odvody, poplatky, cla, evropské fondy. Daně jsou povinnou platbou fyzických a právnických osob v určené lhůtě. Odvody jsou povinné platby právnických osob, platí zejména pro právnické osoby, které jsou alespoň částečně ve státním vlastnictví, část zisku se poté přerozděluje do příjmů státního rozpočtu.

### 1.3.2 Výdaje státního rozpočtu

Dle [9] výše výdajů závisí hlavně na HDP (Hrubý domací produkt). Dle [4] je HDP součet všech finančních hodnot dokončených výrobků a služeb, které jsou vyprodukovány během jednoho roku prostředky umístěnými v dané zemi.

Výdaji se financuje například nákup služeb a zboží pro zajištění veřejných statků, investiční i neinvestiční potřeby neziskových organizací, které jsou zřízeny státem, peněžní transfery obyvatelstvu (například výplaty dávek sociální podpory, příspěvky v nezaměstnanosti, výplaty důchodů), dotace (například pro podporu ekologie), poskytovaní dotací do dalších rozpočtů (například pro územní samosprávy). Největší dílem na výdajích se podílejí mandatorní výdaje (příjemci platby mají na ně ze zákona nárok). Mimo mandatorních výdajů existují také kvazimandatorní výdaje, které jsou určeny zákonem, ale stát může jejich výši regulovat (například příspěvek pro penzijní připojištění). [8]

### 1.3.3 Odhad státního rozpočtu na více let dopředu

Dle zákona [6] je vláda povinna předkládat spolu s návrhem státního rozpočtu také střednědobý výhled, který předkládá prognózu vývoje příjmů a výdajů státního rozpočtu alespoň na 2 roky dopředu.

## 1.4 Analýza existujících řešení

V této podkapitole se věnuji již existujícím řešením, které se zabývají simulací státního rozpočtu na základě odpovědí uživatele na předem definované otázky. Většina webových aplikací, které jsem nalezl a uvádím zde, je určena pro trh USA. Jelikož žádné z těchto řešení není volně dostupné, tak žádnou z implementaci již hotových řešení nemohu využít.

### 1.4.1 Federal Budget Challenge

Federal Budget Challenge [10] je pro mou práci hlavní inspirace. Na hlavní stránce si uživatel může přečíst, že tato stránka je výzvou, která spočívá ve vyplnění odpovědí na otázky za cílem snížení státního dluhu, je zde vidět také přehled příjmů a výdajů členěných dle skupin.

Po zahájení výzvy stránka požaduje zadání základních informací o uživateli, například PSČ, věk, příslušnost k politickém spektru a email. Poté se již přistoupí k samotnému odpovídání na otázky, kde uživatel odpovídá nejprve na otázky týkající se výdajů a poté na otázky týkající se příjmů. U každé z otázek je na výběr z několika odpovědí, žádná odpověď nemusí být povinně vyplněná. U každé odpovědi je uveden vliv na státní rozpočet (jestli zvýší, nebo sníží výdaje či příjmy), kolik procent lidí pro konkrétní možnost hlasovalo a argumenty pro a proti této volbě. Po zodpovězení všech otázek se uživateli zobrazí souhrn dopadů jeho odpovědí a o jakou částku se mu podařilo snížit deficit.

### 1.4.2 Federal Balancing Act 2022

Federal Balancing Act 2022 [11] je taktéž simulace státního rozpočtu. Na hlavní stránce uživatel vidí přehled aktuálního stavu státního rozpočtu, stav deficitu a udržitelnost rozpočtu.

Stránka má opět rozdělené sekce pro příjmy a výdaje. U každé sekce je dělení výdajů/příjmů do skupin. Každé z těchto skupin může uživatel přidávat, nebo ubírat financování, případně provést doplňující volbu, například úpravu věkového limitu pro starobní důchod. Před odesláním odpovědí je uživatel vyzván k doplnění informací – jakou stranu by volil, věk, pohlaví a PSČ. Po odeslání odpovědí se uživateli zobrazí změny v jednotlivých skupin příjmů a výdajů a výhled dlouhodobého deficitu v procentech HDP na základě zvolených změn.

Tato webová aplikace vychází ze systému Balancing Act [12]. Tento systém umožňuje simulaci státního rozpočtu, stejně jako je na stránce Federal Balancing Act 2022. Další možností je vytvoření simulace osídlení obcí, systém pro výběr projektu k realizaci na základě hlasování veřejnosti a systém pro přehled výdajů územních celků financovaných z daní. Vzhledem k existenci záložky ceník usuzují, že tento systém je zpoplatněn. Ceník není veřejný, pro zjištění konkrétní ceny je potřeba zaslat poptávku.

### 1.4.3 Kalkulačka veřejných příjmů a výdajů státního rozpočtu

Kalkulačka veřejných příjmů a výdajů státního rozpočtu [13] nevyužívá rozhraní webové aplikace, ale je to soubor pro aplikaci Microsoft Excel. Z tohoto důvodu není dostupná pro tak velkou skupinu uživatelů jako webová aplikace, tato volba aplikace také snižuje uživatelskou přívětivost.

Soubor je rozdělen na 4 tabulky, v první tabulce je nápověda pro kalkulačku veřejných příjmů, v druhé tabulce je samotná kalkulačka veřejných příjmů, v třetí tabulce je nápověda pro kalkulačku veřejných příjmů a ve čtvrté tabulce je samotná kalkulačka veřejných příjmů. V každé z těchto kalkulaček je možné upravovat výdaje respektive příjmy určitých položek v procentech. Chybí zde například propojení těchto dvou kalkulaček pro kompletní simulaci státního rozpočtu. Přehlednost této aplikace je velice nízká.

## 1.5 Analýza uživatelských požadavků

Na základě rešerše existujících řešení, požadavků ze zadání práce a konzultací s vedoucím práce a Ing. Markem Sušickým jsem sestavil následující seznam uživatelských požadavků na tvořenou webovou aplikaci.

### 1.5.1 Funkční požadavky

Funkční požadavky jsou požadavky na funkcionalitu aplikace.

F1 Uživatel před přístupem do kvízu musí vyplnit své základní údaje – věk, dosažené vzdělání, kraj ve kterém bydlí a jakou stranu by volil.

F2 Uživatel bude mít možnost vyplnit kvíz, postupně se mu budou zobrazovat otázky, u každé otázky musí vybrat právě jednu odpověď. Odpovědi uživatele se budou průběžně ukládat.

F3 Po zvolení odpovědi uživatelem se zobrazí stav simulace rozpočtu, každá odpověď může ovlivnit výsledek zvýšením, nebo snížením příjmů či výdajů.

F4 Při odeslání hotového kvízu uživatelem se zaznamená informace o kompletním dokončení kvízu.

F5 Po odeslání hotového kvízu se uživateli zobrazí výsledky simulace státního rozpočtu, na základě jeho odpovědí, se mu také zobrazí porovnání s výsledky ostatních uživatelů na základě zadané preference politické strany.

F6 Všem uživatelům budou přístupné výsledky dosud dokončených kvízů.

F7 Administrátorská část bude dostupná pouze po přihlášení administrátora.

F8 V administrátorské části bude možné vytvářet a upravovat otázky a odpovědi k nim. U každé odpovědi administrátor vyplní kterou kapitolu státního rozpočtu ovlivňuje, o kolik zvýší příjmy, nebo výdaje této kapitoly v aktuálním roce, následujícím roce a přespříštím roce.

F9 V aplikaci bude umožněna evidence kapitol státního rozpočtu, na tyto evidované kapitoly se bude možné odkazovat při tvorbě odpovědí u otázek a při zadávání stavu státního rozpočtu.

F10 Do aplikace bude možné zadat aktuální stav státního rozpočtu dle kapitol výdajů a příjmů, z toho stavu se vždy bude vycházet pro simulaci státního rozpočtu.

### 1.5.2 Nefunkční požadavky

Nefunkční požadavky jsou požadavky, které nespadají pod funkční požadavky.

N1 Dostupnost přes webové rozhraní.

N2 Aplikace se bude přizpůsobovat šířce obrazovky, tak aby se dala ovládat i z mobilních zařízení.

N3 Intuitivní ovládání.

N4 Aplikace bude v češtině.

## 1.6 Případy užití

Níže uvádím hlavní případy užití této webové aplikace. Mezi aktéry patří systém, uživatel a administrátor. Administrátor je speciální uživatel, který přihlášením získává dodatečná oprávnění.

U1 Vyplnění kvízu – úplné

Uživatel vstoupí na webovou stránku, odstartuje vyplňování kvízu, po vyplnění svých základních údajů se mu zobrazí postupně jednotlivé otázky, pro pokračování na další otázku musí zvolit odpověď (ta ovlivní aktuální ukazatel simulace státního rozpočtu). Pokud uživateli zbývá pouze jedna otázka k odpovězení, systém ho vyzve k dokončení kvízu. Systém následně zaznamená tento kvíz jako dokončený (tedy může sloužit pro následné vyhodnocování výsledků), a zobrazí analýzu odpovědí v souvislosti s ostatními uživateli.

U2 Vyplnění kvízu – neúplné

Uživatel bude postupovat podobně jako v U1, ale kvíz nedokončí, případně jej ukončí dříve než na poslední otázce. V takovém případě systém nebude evidovat tento kvíz jako dokončený.

U3 Zobrazení analýzy všech dokončených kvízů

Uživatel si v aplikaci nechá zobrazit analýzu kvízů, systém mu zobrazí souhrnný přehled odpovědí ze všech vyplněných a již dokončených kvízů.

U4 Vytvoření otázky

Administrátor po přihlášení si zvolí možnost vytvoření otázky, u ní vyplní číslo otázky, titulek a popis.

U5 Upravení otázky

Administrátor bude postupovat stejně jako v U4, ale místo vytvoření nové otázky zvolí úpravu již existující otázky, u které upraví libovolnou z dostupných vlastností otázky.

U6 Vytvoření odpovědi k otázce

Administrátor po přihlášení má možnost u otázky (buď při vytvoření, nebo úpravě) vytvořit jednotlivé odpovědi, u nich nastaví vliv na vybraný výchozí stav státního rozpočtu.

U7 Úprava odpovědi k otázce

Administrátor bude postupovat stejně jako v U6, ale místo vytvoření nové odpovědi k otázce zvolí úpravu již stávající otázky, poté vyplňuje stejné údaje jako při vytváření odpovědi.

U8 Vytvoření kapitoly státního rozpočtu

Administrátor po přihlášení zvolí možnost vytvoření kapitoly státního rozpočtu, zde vyplní číselné označení kapitoly a její název.

U9 Zadání aktuálního stavu státního rozpočtu

Administrátor po přihlášení zvolí možnost změny aktuálního stavu státního rozpočtu, zde vyplní pro různé kapitoly státního rozpočtu (ty, které jsou již v aplikaci vytvořené), zadá příjmy a výdaje pro aktuální rok, následující rok a přespříští rok.

### 1.6.1 Pokrytí funkčních požadavků

V tabulce 1.1 je vyobrazeno pokrytí funkčních požadavků případy užití.

| | U1 | U2 | U3 | U4 | U5 | U6 | U7 | U8 | U9 |
|---|---|---|---|---|---|---|---|---|---|
| F1 | + | + | | | | | | | |
| F2 | + | + | | | | | | | |
| F3 | + | + | | | | | | | |
| F4 | + | | | | | | | | |
| F5 | + | | | | | | | | |
| F6 | | | + | | | | | | |
| F7 | | | | + | + | + | + | | |
| F8 | | | | + | + | + | + | | |
| F9 | | | | | | + | + | + | |
| F10 | | | | | | | | | + |

**Tabulka 1.1** Pokrytí funkčních požadavků případy užití

## 1.7 Analýza technologií

### 1.7.1 PHP

Jako hlavní programovací jazyk jsem zvolil PHP (zkratka z PHP: Hypertext Preprocessor), jelikož je to jeden z nejpopulárnějších programovacích jazyků pro tvorbu webových stránek. Dle [14] je odhadováno, že PHP využívá okolo 77 % stránek, u kterých se povedla identifikace programovacího jazyka, ve kterých byly napsány.

PHP je skriptovací jazyk pro obecné užití, který je určen pro běh na serveru, ale hlavně je zaměřen na vývoj webových stránek.[15]

V následujících podkapitolách popisuji dle [16] populární programové rozšíření (nadstavby) pro PHP.

#### 1.7.1.1 Laravel

Laravel je nadstavba s elegantní syntaxí, dává k dispozici strukturu a základ pro tvorbu aplikace, řeší detaily implementace, tak aby se vývojář mohl soustředit na implementaci hlavní myšlenky aplikace. Poskytuje velkou řadu nástrojů pro usnadnění vývoje. Cílí na všechny skupiny vývojářů (jak začátečníky, tak i pokročilé). K vývoji tohoto nástroje přispívá komunita, pro vývojáře jsou k dispozici návody formou dokumentace i videí. [17]

#### 1.7.1.2 Symfony

Symfony je sada nástrojů pro tvorbu webových aplikací, jakoukoliv část této nadstavby lze použít nezávisle na použití celku. Je to světoznámé a stabilní vývojové prostředí. Výhodou této nadstavby je právě možnost volby určitých komponent, bez nutnosti používání všech komponent. [18]

#### 1.7.1.3 CodeIgniter

CodeIgniter je nástroj pro vytváření webových stránek s pomocí PHP, hlavním cílem použití tohoto nástroje je zrychlení vývoje takové aplikace. Dává k dispozici sadu funkcí a jednoduchý přístup jak k ním přistupovat. Minimalizuje rozsah potřebného kódu pro vytvoření aplikace. Autoři se snažili o zachování maximální možné flexibility, tak aby vývojář nebyl nucen využívat určitý způsob vývoje, ale zvolil si ten, který mu vyhovuje. Tento nástroj se snaží poskytovat potřebné funkce, aniž by je vynucoval. [19]

### 1.7.2 Cascading Style Sheets

CSS (zkratka Cascading Style Sheets) je základní technologie pro webový design, s její pomocí je možné popsat vizualizaci HTML (zkratka z Hypertext Markup Language) nebo XML (zkratka z Extensible Markup Language) dokumentu do požadovaného stylu. HTML je nejzákladnější stavební blok webové stránky, definuje význam a strukturu jednotlivých částí webové stránky, za pomocí značek (anglicky tags). XML je jazyk podobný HTML, ale na rozdíl od něj nedefinuje žádné značky pro obsah, ale je možné je vytvořit pro svou konkrétní potřebu. [20, 21, 22]

#### 1.7.2.1 Bootstrap

Bootstrap je populární nadstavba pro HTML, CSS a JavaScript. Používá se pro vytváření responsivních (přizpůsobujících se šířce obrazovky) stránek. Používá se pro rychlý a snadný vývoj designu webové stránky. Obsahuje komponenty pro snadnou tvorbu jednotlivých části stránek. [23, 24]

### 1.7.3 JavaScript

JS (zkratka z JavaScript) je populární programovací jazyk, využívá se hlavně pro webový vývoj, ale je využitelný i pro oblast mimo webové aplikace. Umožňuje vývoj komplexních funkcí webových aplikací, nejčastěji nějaké interaktivní zobrazení části stránky. Je to třetí ze základních technologií pro tvorbu webových stránek (zahrnují také CSS a HTML). [25, 26]

Níže popisuji populární nadstavby pro tvorbu uživatelského prostředí s pomocí JS.

#### 1.7.3.1 Angular

Angular je vývojová platforma, která se zaměřuje na SPA (single page application – aplikace, které tvoří pouze jedna stránka, která se přepisuje, místo toho, aby se načítalo více stránek prohlížečem). Je založen na komponentách, do kterých se člení části aplikace, poskytuje širokou sadu funkcí a sadu vývojových nástrojů pro ulehčení vývoje. [27]

#### 1.7.3.2 React

React je nástroj pro snadnou tvorbu uživatelského prostředí. Za pomocí návrhu jednoduchých šablon pro komponenty je možné efektivně aktualizovat a vykreslovat pouze potřebné data. Komponenty jsou důležitou součástí Reactu, s pomocí nich se člení kód do jednotlivých logických celků. Lze jej použit a integrovat s libovolnou jinou technologií. [28]

### 1.7.4 Databáze

Databáze slouží k uchovávání strukturovaných dat. Relační databáze navíc uchovává vztahy mezi těmito daty. Zvolil jsem databáze podporující SQL (Structured Query Language), což je jazyk pro manipulaci s daty, definování vztahů mezi nimi, a tvorbu struktury databáze, také může nastavovat oprávnění pro přístup k jednotlivým druhům dat. Data jsou seskupovány do tabulek, každá tabulka má sloupce (reprezentující vlastnosti dat) a řádky (reprezentující samotná data). [29]

#### 1.7.4.1 MySQL

MySQL je populární systém pro databáze, který je open-source. Nicméně po převzetí firmou Oracle jsou některé funkcionality zpoplatněny, v reakci na tuto skutečnost vznikl projekt MariaDB, který se zakládá na MySQL, ale zakládá se na zachování open-source politiky spolu se zachováním všech funkcionalit zadarmo. [29, 30]

#### 1.7.4.2 PostgreSQL

PostgreSQL je open-source systém pro databáze, může být využíván pro ukládaní jak strukturovaných (objektů), tak i nestrukturovaných dat. Je vyvíjen od roku 1986, nabízí velkou řadu funkcionalit, a dokáže uchovávat velké množství dat. [29, 31]

# Kapitola 2
# Návrh

*V této kapitole se věnuji výběru technologií, návrhu architektury, návrhu uživatelského prostředí, návrhu databáze a také návrhu ukázkových dat pro aplikaci.*

## 2.1 Výběr technologií

Technologie pro vyvíjenou aplikaci jsem vybíral na základě kapitoly 1, průzkumu studií a doporučení od Ing. Marka Sušického (člen OpenDataLab laboratoře na Fakultě Informačních Technologii Českého Učení Technického v Praze). Pro PHP jsem zvolil technologii Laravel, pro CSS Bootstrap, pro JavaScript React a pro databází jsem zvolil systém PostgreSQL.

### 2.1.1 Laravel

Rozhodl jsem se pro PHP nadstavbu Laravel, jelikož podle je to jeden z nejpopulárnějších nástrojů pro PHP. Dle [32, 33] zvládne pojmout nejvíce požadavků za sekundu, využívá nejméně operační paměti a má nejmenší dobu odezvy z programových rozšíření probíraných ve 1.7.1.

### 2.1.2 Bootstrap

Pro CSS jsem se rozhodl pro rozšíření Bootstrap, jelikož je to populární způsob tvorby designu webových stránek, který klade důraz na responsivitu a obsahuje řadu komponent pro úpravu designu jednotlivých části stránek.

### 2.1.3 React

U technologie pro JavaScript jsem se rozhodl pro React, jelikož je to populární nástroj pro tvorbu interaktivního uživatelského rozhraní, a je pouze na vývojáři kde všude jej použije a s jakými technologiemi jej zkombinuje.

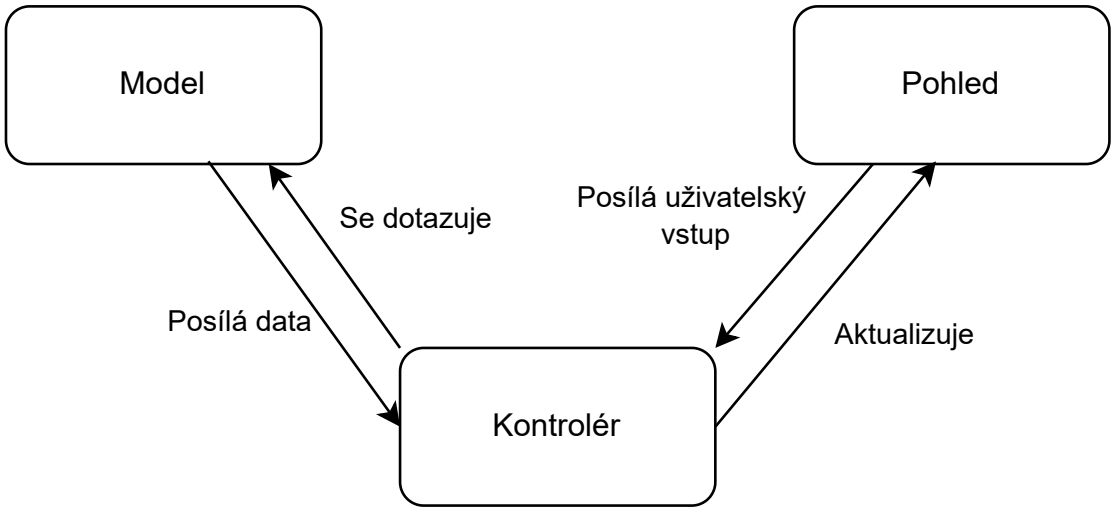
### 2.1.4 PostgreSQL

Rozhodl jsem se pro ukládaní dat v databázi systému PostgreSQL, jelikož je to populární systém s dlouholetým vývojem a byl mi také doporučen Markem Sušickým.

## 2.2 Architektura

Laravel má postaven základ na softwarové architektuře MVC (Model View Controller). MVC se skládá ze tří částí – modelu, kontroléru (anglicky controller) a pohledu (anglicky view), rozdělením kódu do těchto tří částí se snaží oddělit hlavní aplikační logiku od uživatelského prostředí, tímto se docílí lepší udržitelnost a přehlednost kódu. Model zaštiťuje základní logiku týkající se dat (například změnu informace v datech) a jejich reprezentaci v aplikaci. Pohled vytváří uživatelské prostředí na základě dat z modelu. [34]

Komunikace tří částí v MVC architektonickém návrhu je znázorněna na obrázku 2.1. Jelikož je Laravel základem této aplikace, bude tedy základem pro aplikaci také architektura MVC, okolo které budu navíc budovat další pomocné části aplikace (stejně, jako to dělá samotný Laravel). Jelikož pohled se stará o uživatelské rozhraní, tak v této části aplikace bude hlavní výchozí bod pro umístění kódu pro HTML, Bootstrap (respektive CSS) a React (respektive JavaScript).




**Obrázek 2.1** Komunikace komponent v architektonickém návrhu MVC, převzato z [34], překresleno a přeloženo autorem

Aplikace bude členěna na administrátorskou a uživatelskou část. Administrátorská část bude dostupná pouze po přihlášení uživatelským jménem a heslem. V administrátorské části bude možnost vložit data aplikace, popis dat je v sekci 2.4. Uživatelská část bude přístupná všem uživatelům bez přihlášení, bude zde popis aplikace, samotný kvíz a výsledky všech vyplněných a dokončených kvízů. Vzhled těchto částí navrhuji v sekci 2.3.

Komunikace mezi uživatelem a aplikací bude realizována architekturou klient-server. Dle [35] klient-server je architektura, která zahrnuje dvě aplikace (případně počítače) – klienta a server, klient odesílá požadavky na server, server je vyhodnotí a pošle klientovi odpověď.

Pro komunikaci se využívá protokol HTTP (Hypertext Transfer Protocol), který využívá architektury klient-server. Klient zahájí komunikaci odesláním požadavku (anglicky request) na server, ten jej zpracuje a odešle odpověď (anglicky response). Požadavek specifikuje HTTP metodu (GET, POST, PUT...), která indikuje požadovanou akci, která se má provést, dále obsahuje adresu (většinou relativní vůči serveru na který se dotazuje) kam požadavek směruje, verzi HTTP protokolu, dále mohou následovat různé hlavičky (doplňující informace) a případně tělo požadavku (využívá se například při metodě POST, pro posílání dat na server). Odpověď obsahuje verzi protokolu, stavový kód (200, 403, 404, 422...), krátký slovní popis významu stavového kódu, hlavičky (další informace) a (nepovinně) tělo odpovědi, kde jsou vložena data (například HTML stránka). [36]

### 2.2.1 Začlenění technologie React

React použiji pro zobrazení kvízu, jelikož je důležité, aby tato část byla interaktivní, což React umožní, tím, že se nebude muset načítat při každé odpovědi celá stránka znovu, ale pouze se překreslí obsah aktuální stránky na základě dat.

Jelikož React bude fungovat v prohlížeči uživatele, tak musí získávat data ze serveru, kde budou uložena v databázi (více v sekci 2.4). Laravel bude poskytovat API (Application Programming Interface) za pomocí architektury klient-server právě pro tyto účely. Toto API bude možno využívat různými klienty, tím se zajistí lepší rozšířitelnost do budoucnosti, například API může sloužit jako zdroj dat pro mobilní aplikaci. Axios [37] je nástroj pro komunikaci se serverem za pomocí JS, tento nástroj bude spuštěn v prohlížeči uživatele. Axios je již začleněn v základu aplikace pro Laravel. Axios odešle požadavek na server, server podle vyslaného požadavku pošle v odpovědi požadované data, a tyto data poté Axios předá Reactu, který je může dále zpracovávat.

Dle [38] API je programové rozhraní, které definuje jakým způsobem lze získat obsah ze systému (v tomto případě data z aplikace) a jak budou formátovaná. Laravel bude sloužit pro API jako server a React jako klient, ale API nebude nijak omezeno na typ klienta, kdokoliv se k tomuto API bude moct připojit, tohle zajistí možné rozšíření do budoucna například o mobilní aplikaci.

Server bude posílat odpovědi ve formátu JSON (JavaScript Object Notation). JSON je populární formát pro přenos dat, jelikož je jednoduchý a čitelný jak pro stroje, tak i pro lidi, je nezávislý na programovacím jazyku, i když by se mohlo zdát, že podle názvu je závislý na JavaScriptu, ale pouze vychází z notace zápisu objektů v tomto programovacím jazyce. [39]

## 2.3 Návrh uživatelského prostředí

Jak jsem již zmínil v sekci 2.2 aplikace bude členěna na dvě části – administrátorskou a uživatelskou. Obě části budou mít podobný vzhled.

### 2.3.1 Uživatelská část

Uživatelská část bude mít v horní části stránky záhlaví s názvem aplikace (formou odkazu na hlavní stránku), a odkazy na jednotlivé podstránky – hlavní stránku, kvíz a výsledky. Pod záhlavím bude prostor pro hlavní obsah stránky (dle konkrétní zvolené podstránky), pod tímto prostorem bude zápatí, které bude obsahovat copyright, informaci o autorovi aplikace a odkaz na administraci.

Na hlavní stránce bude popis aplikace, na stránce s kvízem bude nejprve zobrazen formulář pro vyplnění věku, nejvyššího dosaženého vzdělání

### 2.3.2 Administrátorská část

Administrátorská část bude dostupná přes odkaz v patičce z libovolné podstránky. Pokud uživatel při přístupu do administrace nebude přihlášen, tak bude vyzván k přihlášení.

V horní částí stránek v administraci bude záhlaví obdobné tomu z uživatelské části, ale místo názvu aplikace zde bude napsán text *Administrace* pro snadné odlišení od uživatelské části a odkazy na jednotlivé podstránky administrace – správu krajů, politických stran, vzdělání, kapitol rozpočtu a otázek, a dále odkaz pro odhlášení. Pod záhlavím bude opět prostor pro hlavní obsah jednotlivých podstránek, a dále pod prostorem pro obsah bude opět zápatí obdobné uživatelské části, ale s odkazem právě na tuto část aplikace.

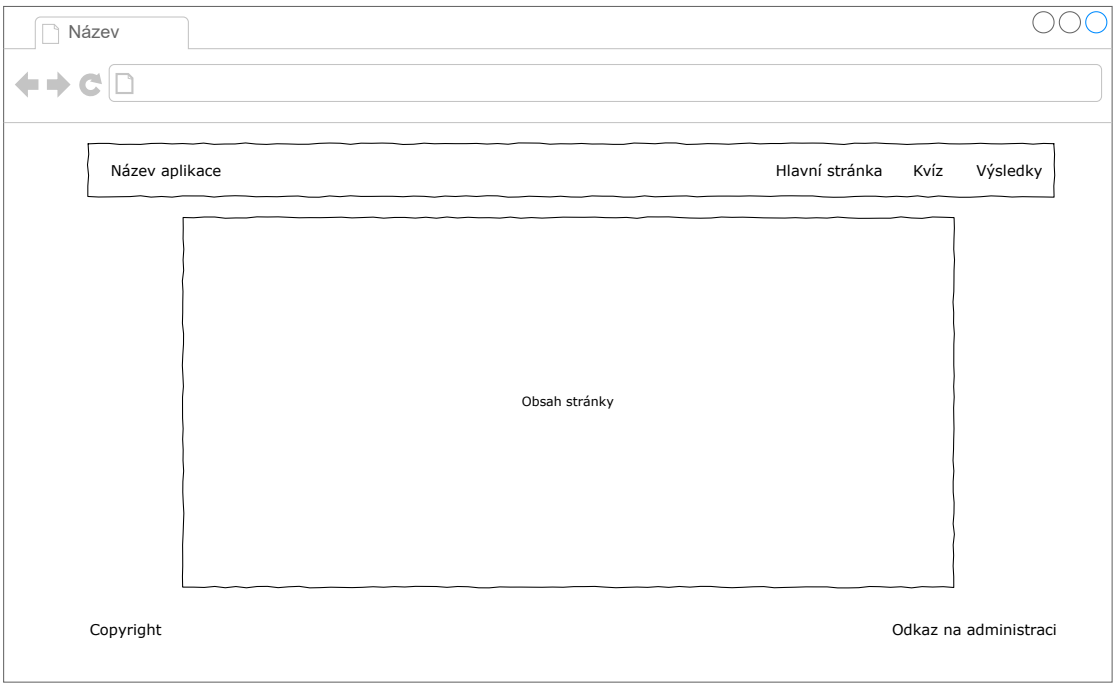
Na úvodní stránce bude statistika počtu zahájených kvízů a počtu dokončených kvízů. Pro správu krajů, politických strany a vzdělání bude využito zobrazení v tabulce, kde u každého

řádku bude výpis jednotlivých dat a možnost jejich editace nebo smazání, dále nad tabulkou bude tlačítko s odkazem na stránku pro přidání nových dat. Správa kapitol rozpočtu bude zobrazena obdobně jako například správa krajů, ale bude zde navíc u každého záznamu v tabulce odkaz pro zobrazení či přidání aktuálního stavu pro danou kapitolu. Správa otázek bude vypadat obdobně jako správa krajů, ale bude zde navíc odkaz pro zobrazení konkrétní otázky, v tomto zobrazení bude možnost přidat, upravit a smazat odpovědi k dané otázce, u každé odpovědi bude možné definovat změnu stavu rozpočtu pomocí odkazu v seznamu odpovědí.

### 2.3.3 Návrh vzhledu

Hlavní barvou pozadí bude bílá, modrá barva bude sloužit pro odkazy v menu, hlavní barva textu bude černá. Chci docílit minimalistického vzhledu, tak aby stránka působila přehledně. Návrhy jednotlivých částí aplikace jsem vytvořil za pomocí nástroje [40].

Na obrázku 2.2 je vyobrazen návrh vzhledu pro uživatelskou část, v záhlaví je uveden název aplikace a jednotlivé odkazy na podstránky, uprostřed je prostor pro obsah jednotlivých podstránek, v zápatí je umístěn copyright a odkaz na administraci.

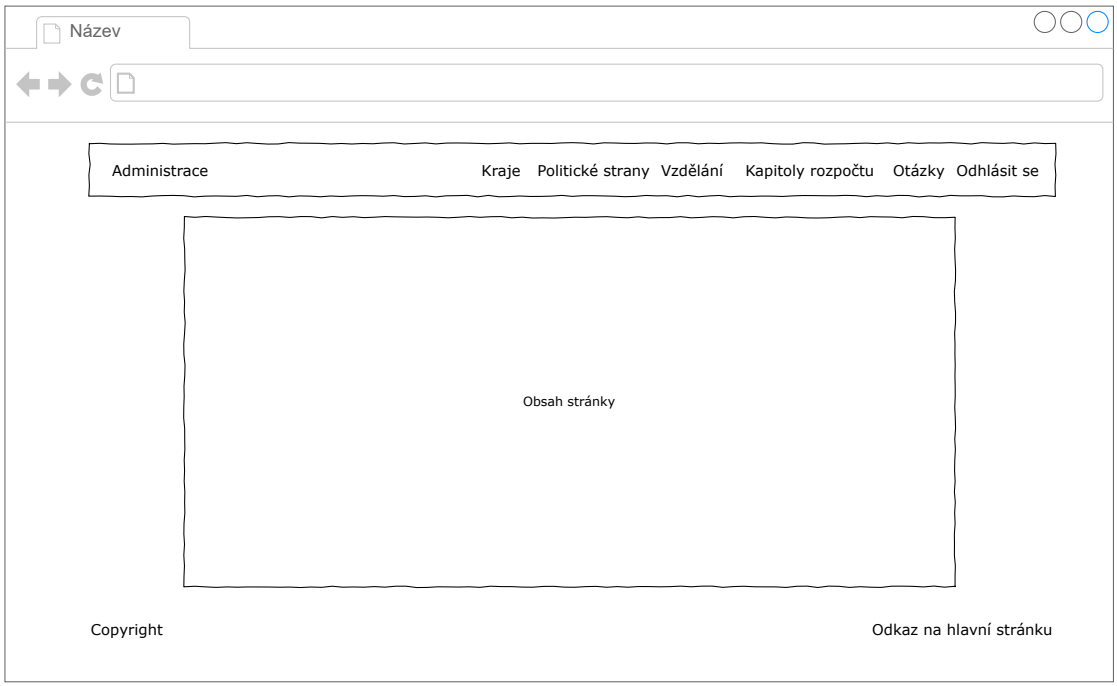



**Obrázek 2.2** Návrh vzhledu uživatelské části aplikace

Na obrázku 2.3 je vyobrazen návrh vzhledu pro administrátorskou část, je velmi podobný předchozímu návrhu. V záhlaví místo názvu aplikace bude informace, že se uživatel nachází v administraci, v menu budou odkazy na jednotlivé podstránky pro správu dat aplikace. V zápatí bude odkaz na hlavní stránku aplikace.

## 2.4 Návrh databáze

Na obrázku 2.4 je vyobrazeno konceptuální schéma databáze, schéma je vytvořeno za pomocí aplikace [40]. Jednotlivé tabulky databáze jsou vyobrazeny jako obdélníky, v horní části obdélníku je název tabulky, ve spodní části poté jednotlivé atributy (sloupce). Mezi jednotlivými tabulkami



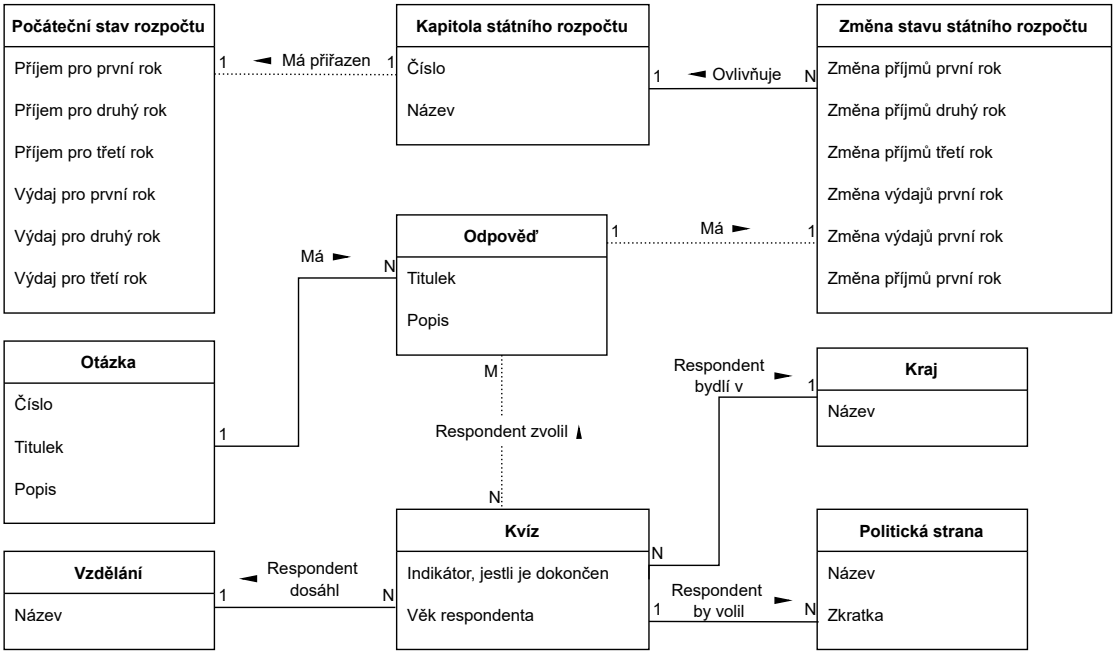

**Obrázek 2.3** Návrh vzhledu administrátorské části aplikace

(obdélníky) jsou vedeny přímky, které symbolizují vztah mezi entitami (jednotlivými záznamy v tabulce), pokud je tento vztah povinný poté je znázorněn plnou čárou, jinak přerušovanou. U každého vztahu je popisek a šipka ke kterému směru se popisek vztahuje. Navíc na každé straně přímky je uvedeno buď číslo 1 nebo písmeno N (případně M, pokud na jedné straně je již písmeno N, ale význam zůstává stejný) – číslo 1 vyjadřuje, že záznam z tabulky na opačném konci je v tomto vztahu jedenkrát, písmeno N (respektive M) označuje, že záznam z tabulky na opačném konci je v tomto vztahu libovolně-krát. Například vztah mezi entitami v tabulce *Změna stavu státního rozpočtu* a v tabulce *Kapitola státního rozpočtu* znamená, že změna stavu státního rozpočtu ovlivňuje kapitolu státního rozpočtu (v tomto vztahu může figurovat právě pro jednu kapitolu) a naopak kapitola státního rozpočtu může být ovlivňována více změnami stavu státního rozpočtu.

Jednotlivé názvy tabulek a sloupců budou v implementaci přeloženy do angličtiny, kvůli konzistenci s kódem (který bude obsahovat také názvy v angličtině), a přibudou zde další položky, které souvisí s implementačními detaily. Výsledné schéma databáze je vygenerováno z databáze a lze jej nalézt v příloze A.

## 2.5 Data pro aplikaci

Připravil jsem základní ukázková data pro aplikaci, pro demonstraci funkčnosti. Seznam krajů jsem připravil dle [41]. Nejvyšší dosažené vzdělání budou 5 typů – základní, střední, vyšší odborné, vysokoškolské a položka jiné, pro ostatní vzdělání. Pro seznam stran jsem využil výsledky voleb do parlamentu v roce 2021 dle [42], odkud jsem zvolil 10 stran s nejvíce hlasy, dále zde bude jedna možnost jiné pro zahrnutí ostatních stran. Kapitoly rozpočtu a počáteční stavy rozpočtu pro rok 2022 jsem čerpal ze [43, 44], počáteční stavy pro roky 2023 a 2024 jsem čerpal ze střednědobého výhledu rozpočtu vlády [45].




**Obrázek 2.4** Konceptuální schéma databáze

### 2.5.1 Otázky

Pro aplikaci jsem také připravil 3 základní otázky, spolu s odpověďmi a jejich vlivy na kapitoly státního rozpočtu.

#### 2.5.1.1 Slevy na jízdném

Jako téma jedné z otázek jsem zvolil problematiku slev na jízdném pro studenty a seniory, jelikož tohle téma bylo v poslední době často probíráno jak v médiích, tak i na sociálních sítích. Dle [46] vláda schválila snížení slevy jízdného na 50 % ze 75 %, což by mělo přinést úsporu 1,9 miliardy Kč a celkovou útratu za slevy na jízdném ve výši 3,9 miliardy korun. Kdyby ministerstvo zmenšilo slevu na 25 %, tak by vydalo celkově za slevy na jízdném 2 miliardy korun. Dle [47] ve schváleném rozpočtu, že kterého čerpám je již toto snížení slev na jízdném zahrnuto ve snížených výdajích.

Proto pro aplikaci je považován výchozí stav pro 50% slevu, tedy nepřináší žádnou změnu do státního rozpočtu, 75% sleva by znamenala zvýšení výdajů o 1,9 miliardy korun, a 25% sleva by přinesla úsporu výdajů ve výši 1,9 miliardy korun. Z vyznění prohlášení vlády usuzuji, že tohle zvýšení / snížení výdajů se promítne skoro stejně i v dalších letech, pro simulaci využiji stejné částky pro všechny 3 zamýšlené roky v aplikaci.

#### 2.5.1.2 Výdaje pro armádu

Dle [48] vláda má v plánu do roku 2025 zvýšit výdaje za kapitolu ministerstva obrany na výši 2 % HDP, momentálně je ve výši 89,1 miliardy, což je cca 1,35 % HDP.

Pro aplikaci jsem tedy připravil 4 možnosti pro odpovědi – první zahrnuje zrychlené zvýšení plnění tohoto plánu, v roce 2023 zvýšení výdajů na 1,6 % HDP a v roce 2024 na 1,8 % HDP. Druhá odpověď plánuje pozvolnější zvýšení nejprve v roce 2023 na 1,5 % HDP a v roce 2024 na 1,65 % HDP. Třetí odpověď odráží střednědobý výhled, takže nezvyšuje výdaje kapitoly ministerstva obrany (pro rok 2023 je to cca 1,49 % HDP a pro rok 2024 cca 1,57 % HDP – využil jsem odhadu

výdajů ministerstva obrany a výše HDP dle střednědobého výhledu rozpočtu). Poslední odpověď zahrnuje výši aktuální relativní hodnoty výdajů vůči HDP, tedy 1,35 %.

K odhadu konkrétních částek jsem využil střednědobého výhledu rozpočtu [45], kde je odhad hodnoty HDP (pro rok 2023 je to 6 692 miliard korun a pro rok 2024 je to 6 958 miliard korun). Pro jednotlivé odpovědi a jednotlivé roky jsem vypočítal postupně jednotlivé hodnoty výdajů armády a vypočítal rozdíl, oproti předpokladu těchto výdajů. Pro první odpověď je změna pro rok 2023 navýšení výdajů cca o 7,1 miliardy korun, pro rok 2024 vychází navýšení výdajů o cca 15,7 miliardy korun. Pro druhou odpověď vychází navýšení výdajů pro rok 2023 o cca 0,4 miliardy korun a pro rok 2024 o cca 5,3 miliardy korun. Třetí odpověď nijak nemění odhadované výdaje. A čtvrtá odpověď snižuje výdaje pro rok 2023 o cca 9,6 miliardy korun a pro rok 2024 cca o 15,6 miliardy korun.

#### 2.5.1.3 Valorizace důchodů

Další téma otázky jsem zvolil valorizaci důchodů, vláda dle [49] zavede v září 2022 mimořádnou valorizaci důchodů, kvůli vysoké míře inflace. Valorizace se dotkne všech státních důchodů (starobních, invalidních i pozůstalostních). Procentní výměra důchodů vzroste o 5,2 %, což zvýší výdajovou stránku kapitoly ministerstva práce a sociálních věcí o cca 8 miliard korun.

Uživateli budou nabídnuty dvě možnosti – zda-li by schválil takovouto valorizaci, nebo ne. Předpokládám, že v dalších letech se tato konkrétní valorizace projeví na výdajích přibližně stejně, a tedy pro všechny 3 roky simulace budou zvýšeny výdaje cca o 8 miliard korun.

# Kapitola 3
# Implementace

*V této kapitole popisuji jaké vývojové prostředí jsem využil, jak a jaké technologie jsem při implementaci využíval, jaké kroky jsem při implementaci činil a úskalí implementace.*

## 3.1 Vývojové prostředí

Pro vývoj aplikace používám nástroje Docker pro běhové prostředí aplikace, Git pro udržování historie kódu aplikace a vývojové prostředí PhpStorm, ve kterém píšu kód aplikace. Pro správu závislostí na ostatních technologiích (používá se pro instalaci balíčků rozšíření a nadstaveb programovacích jazyků) používám Composer pro PHP a npm pro JavaScript.

### 3.1.1 Docker

Docker je platforma, která umožňuje vytváření kontejnerů – standardizované spustitelné komponenty, obsahující programy a aplikace, lze je spustit v libovolném prostředí. Docker umožňuje správu těchto kontejnerů, jejich vytvoření, aktualizaci, spuštění a zastavení. Obraz kontejneru je šablona pro kontejner, která definuje software obsažený v kontejneru, nastavení tohoto prostředí a přidružení souborů ke kontejneru. Spuštěním obrazu se z něj stává kontejner. Kontejner neobsahuje svůj operační systém, tak jako virtuální počítač, a je tedy tím pádem menší a snadněji udržovatelný, jednotlivé kontejnery jsou od sebe odděleny, a tím je zajištěna vysoká bezpečnost. [50, 51]

#### 3.1.1.1 Dockerfile

Dockerfile je textový soubor, který slouží k definování obrazu kontejneru na platformě Docker. Tento soubor se poté spustí za pomocí nástrojů platformy Docker, aby se vytvořil požadovaný kontejner.

#### 3.1.1.2 Tvorba kontejneru pro aplikaci

Laravel poskytuje nástroj Sail, který zaobaluje práci s kontejnerem Dockeru, v dokumentaci [52] je návod, pro stažení obrazu kontejneru, který již má zabudovaný Sail, tohle se mi ale nezdařilo, a protože jsem chtěl využít Docker, tak jsem postupoval dle [53], kde je vypsaný základní Dockerfile s PHP podporující Laravel a PostgreSQL databázi. Tento ukázkový soubor jsem upravil dle potřeb této aplikace.

Využil jsem souboru pro Docker Compose, což je nástroj, který dá do skupiny více kontejnerů, které mohou mezi sebou komunikovat, tohoto využiji, jelikož jeden kontejner bude databáze

a druhý bude Apache, což je HTTP server, na kterém bude spuštěna aplikace. Docker Compose se definuje pomocí *docker-compose.yml* souboru.

Struktura souborů je následující:

- `docker` ..... adresář pro soubory týkající se Dockeru
  - `apache` ..... adresář pro konfigurační soubory serveru Apache
    - `default.conf` ..... soubor pro konfiguraci serveru Apache
  - `php` ..... adresář pro konfigurační soubory týkající se nastavení kontejneru pro PHP
    - `Dockerfile` ..... soubor pro konfiguraci obrazu Dockeru
  - `postgres` ..... adresář pro konfigurační soubory týkající se PostgreSQL
    - `data` ..... adresář pro uložení dat databáze
    - `.env` ..... konfigurační soubor pro kontejner s PostgreSQL
    - `.env.example` ..... ukázkový konfigurační soubor pro kontejner s PostgreSQL
- `docker-compose.yml` ..... soubor pro Docker Compose

V adresáří *php* je soubor *Dockerfile*, který výchází ze základního obrazu pro Apache server s PHP ve verzi 8 a nainstaluje potřebný software pro Laravel (včetně Composeru). Na tento soubor se odkazuje ze souboru *docker-compose.yml*, který definuje dva kontejnery - jeden pro databázi a jeden pro Apache, u databázového kontejneru je připojen konfigurační soubor *.env*, který výchází ze souboru *.env.example*, v něm je uloženo nastavení hesla, názvu účtu a názvu databáze, které se použije při instalaci PostgreSQL serveru. V souboru *default.conf* je nastavení pro povolení přepisu adresy webové aplikace (čehož využívá Laravel), což umožňuje tvoření hezčích odkazů.

Vybudování obrazu do kontejneru jsem provedl pomocí nástroje Docker Compose. Prvotní nastavení kontejneru trvá delší dobu, jelikož se instaluje všechen potřebný software, ale každé další spuštění je již poměrně rychlé.

### 3.1.2 Git

Verzovací systém udržuje záznamy o změnách souborů v čase, je možné kdykoliv se k těmto jednotlivým verzím souborů vrátit. Distribuovaný verzovací systém je systém, kde uživatel má lokální změny u sebe ve svém počítači, a na vzdáleném serveru je uložena centralizovaná kopie těchto souborů, za pomocí synchronizace těchto dvou míst je možné dosáhnout distribuce kopií kódu s historií více lidem, kteří mohou tak na kódu pracovat zároveň. Git je distribuovaný verzovací systém, soubory které spravuje sdružuje do repozitářů, tyto repozitáře mohou být jak pouze lokální (tedy nejsou synchronizovány se žádným serverem), tak i vzdálené (u nich je možné si stáhnout jejich lokální kopii a průběžně synchronizovat změny se serverem). V rámci jednoho repozitáře je možné oddělit spravované soubory do více větvi – paralelně existující změny souborů, tyto větve je možné slučovat dohromady. Jednotlivé změny se sdružují do bloků (commit), tyto bloky lze pojmenovávat pro jednodušší dohledání jednotlivých verzí. [54]

Využívám server GitHub pro uložení kódu aplikace, v [55] je repozitář pro tuto aplikaci veden pod laboratoří OpenDataLab Fakulty Informačních Technologií ČVUT.

### 3.1.3 Composer

Composer je nástroj pro správu závislostí balíčků rozšíření v PHP pro aplikace, umožňuje jejich snadnou instalaci a aktualizaci. [56] Soubor *composer.json* definuje jednotlivé závislosti na balíčcích rozšíření, tento soubor je využíván při instalaci a aktualizaci těchto závislostí ve zdrojovém kódu aplikace, při aktualizaci (či prvotní instalaci) se vytvoří soubor *composer.lock*, který definuje jaké závislosti a v jakých verzích se mají instalovat, to zajišťuje stabilitu aplikace, jelikož se při instalaci těchto závislostí jiným uživatelem nestane, že by použil jejich nekompatibilní verzi. [57] Composer také přináší funkci automatického načítání tříd požadovaných aplikací a jednotlivých balíčků, načítání provádí na základě vestavěné funkce PHP pro automatické načítání tříd. [57, 58]

### 3.1.4 Npm

Npm je stejně jako Composer nástroj pro správu závilostí balíčku rozšíření, ale narozdíl od Composeru, npm spravuje balíčky pro Javascript. V souboru *package.json* se uchovává seznam jednotlivých závislostí a požadovaných verzí. [59]

## 3.2 Základ aplikace

Nainstaloval jsem Laravel dle [52] s pomocí nástroje Composer pro základ aplikace. Poté jsem provedl konfiguraci připojení, názvu aplikace a potřebných proměnných v souboru .env a patričně upravil soubor .env.example. K tvoření jednotlivých tříd jsem využíval Laravel Artisan, což je nástroj v rámci Laravelu, který přijímá příkazy z příkazové řádky a vykonává je. V základu podporuje generování běžných struktur programu.

### 3.2.1 Životní cyklus požadavku

Příchozí požadavek je dle [60] předán souboru *public/index.php*, který zajistí načtení automatického načítání tříd za pomocí Composeru a vytvoří instanci Laravel aplikace ze souboru *bootstrap/app.php*. Laravel vytvoří službu (service container) pro poskytování závislostí pro jednotlivé metody/třídy (dependency injection). Používání tohoto principu umožňuje dynamické vkládání závislostí na jednotlivé potřebné místa, umožňuje také například snadnější testování, kde místo třídy s reálnou implementací se dodá testovací třída, u které je možno ověřit testované vlastnosti.

Požadavek je poté odeslán do kernelu aplikace, buď je to HTTP kernel (pro HTTP požadavky), nebo kernel konzole (pro požadavky z konzole). Kernel zajistí načtení a konfiguraci potřebných komponent pro aplikaci, než se vykoná samotný požadavek, například poskytovatele služeb, kteří zajišťují načítání komponent samotného Laravelu. HTTP kernel také definuje seznam HTTP middleware, přes které musí požadavek projít, zajišťují přídavnou funkcionalitu aplikace, může sloužit jako filtr nebo bližší prozkoumání jednotlivých požadavků.

Po načtení potřebných komponent se požadavek předá směrovači, který zajistí předání konkrétní metodě představující cestu pro směrovač, nebo kontroléru. Jakmile se zde zpracuje požadavek, tak se vytvoří odpověď, která je zpět odeslána přes middleware, kernel a zpět do souboru *public/index.php*, který tuto odpověď odešle klientovi.

### 3.2.2 Adresářová struktura

Základní adresářovou strukturu [61] jsem upravil pro potřeby této aplikace následovně (některé složky jsou vynechány, tyto složky lze dohledat ve zmíněném zdroji, pro tuto vizualizaci nejsou důležité).

- `app` ..... adresář obsahující hlavní kód aplikace
  - `Console` ..... adresář týkající se zpracování požadavků z příkazové řádky
    - `Commands` ..... adresář pro vytváření příkazu pro příkazovou řádku
  - `Http`
    - `Controllers` ..... adresář, který obsahuje kontroléry
      - `Admin` .... adresář, který obsahuje kontroléry specifické pro administrátorskou část
      - `Api` ..... adresář, který obsahuje kontroléry specifické pro API aplikace
    - `Requests` ..... adresář, který obsahuje validaci příchozích požadavků
      - `Admin` ..... obsahuje validaci požadavků specifických pro administrátorskou část
    - `Resources` ..... obsahuje pomocné třídy pro korektní transformaci dat pro API
  - `Models` ..... v tomto adresáři jsou uloženy modely
  - `View`

- `Components` ..... v tomto adresáři jsou uloženy třídy, které inicializují jednotlivé komponenty pohledů

- `bootstrap` ..... adresář obsahující soubory Laravelu, které slouží ke spuštění aplikace
- `config` ..... adresář obsahující soubory nastavení aplikace
- `database` ..... adresář obsahující soubory týkající se nastavení databáze
  - `migrations` ..... adresář obsahující migrace
  - `seeders` ..... adresář obsahující soubory pro vložení základních dat do databáze
- `lang` ..... adresář obsahující jazykové soubory
- `node_modules` ..... adresář obsahující soubory npm
- `public` ..... v tomto adresáři jsou soubory, které jsou určené k veřejnému přístupu
- `resources` ..... adresář obsahující zdrojové soubory pro Javascript, CSS a pohledy
  - `js` ..... v tomto adresáři jsou soubory týkající se programovacího jazyka Javascript
    - `components` ..... v tomto adresáři jsou komponenty pro React
  - `sass` ..... v tomto adresáři jsou soubory pro SASS
  - `views` ..... v tomto adresáři jsou uloženy všechny pohledy a jejich komponenty
- `routes` ..... adresář obsahující soubory pro určení cest pro směrovač
- `storage` ..... adresář určený pro úložiště vnitřních souborů Laravelu a aplikace
- `tests` ..... adresář obsahující testy
- `vendor` ..... adresář obsahující soubory Composeru

## 3.3 Reprezentace dat

Jako další krok po instalaci a konfiguraci aplikace jsem zvolil tvorbu reprezentace dat v aplikaci. Jednotlivé záznamy (entity) v tabulkách databáze budou reprezentovány formou modelů. Po tvorbě modelů jsem vytvořil migrace pro databázi, které slouží k definování struktury databáze. A poté jsem upravil výchozí třídu pro vkládání výchozích dat (Seeder) do databáze, který slouží pro vložení výchozích dat do databáze.

### 3.3.1 Tvorba modelů

Laravel pro práci s databází umožňuje dva přístupy – buď přímo použít metody, které budují jednotlivé SQL dotazy, nebo použít Eloquent ORM, což je nástroj, který umožňuje namapovat jednotlivé vztahy mezi záznamy z relační databáze na vztahy mezi jednotlivými instancemi tříd v PHP a navíc tento nástroj umožňuje na jednotlivých modelech používat metody, které také budují SQL dotazy. Na základě názvu modelu Eloquent ORM odvodí název tabulky, poté podle jednotlivých sloupců v tabulce doplní do modelu dynamické atributy, které reprezentují tyto sloupce, lze do nich přiřazovat data a také je z nich číst. Tyto modely mají metodu *save*, která uloží změny, případně vytvoří tento záznam v tabulce. [62, 63] Zvolil jsem Eloquent ORM.

Za pomoci nástroje Artisan jsem vytvořil jednotlivé modely, ty jsem doplnil o dokumentační komentáře. U modelu *Education* Eloquent nemohl odhadnout správný název tabulky, jelikož se jedná o nepravidelné podstatné jméno, a tedy jsem musel specifikovat proměnnou *table*, která definuje pro jakou tabulku je daný model určen.

Dle [64] se jednotlivé vztahy mezi entitami definují pomocí funkcí, které zároveň slouží také jako definování dodatečných dynamických atributů. V těchto funkcích se spouští metody zděděné z obecného modelu, tyto funkce se spouští podle typu vztahu, argumentem těchto funkcí je vždy třída reprezentující druhou stranu ve vztahu, tedy druhý model. V této aplikaci jsou vazby 1:1, 1:N a N:M. Vazba 1:1 je nejjednodušší ze zmíněných vazeb, stačí u entity v této metodě spustit funkci *belongsTo*, která označuje že tento model patří v daném vztahu k jinému modelu a v druhém modelu spustit funkci *hasOne*, která označuje, že daný model má ve vztahu k sobě jinou funkci. Vazba 1:N se definuje obdobně, akorát u jednoho z modelů se místo funkce *hasOne* spouští funkce *hasMany*, která indikuje, že daný model má ve vztahu k sobě více jiných modelů.

**Výpis kódu 3.1** Kód modelu Answer

```
class Answer extends Model
{
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = ['title', 'description'];

    /**
     * Get the question that this answer belongs to
     */
    public function question()
    {
        return $this->belongsTo(Question::class);
    }

    /**
     * Get the budget state change, that this answer makes
     */
    public function budgetStateChange()
    {
        return $this->hasOne(BudgetStateChange::class);
    }

    /**
     * Get all quizzes in which this answer is chosen
     */
    public function quizzes()
    {
        return $this->belongsToMany(Quiz::class, 'quiz_answers');
    }
}
```

Vazba M:N je specifická v tom, že je symetrická (není zde žádná entita který by byla ve slabší roli) a realizována tabulkou, která je mimo dvě definované entity, u obou modelů se v dané metodě spouští funkce *hasMany*.

Za pomocí těchto principů jsem postupně vytvořil modely pro odpovědi, kapitoly rozpočtu, počáteční stavy rozpočtu, změny stavů rozpočtu, vzdělání, politické strany, otázky, odpovědi, kvízy a kraje dle návrhu databáze ze sekce 2.4. Názvy modelů jsem přeložil z češtiny do angličtiny, jelikož Laravel je psán v angličtině, tak jsem chtěl zachovat konzistenci pojmenování v kódu. Navíc jsem upravil model pro uživatele (*User*), který využiji při tvorbě administrace a základ tohoto modelu byl již obsažen v základu aplikace. Ve výpisu kódu 3.1 je implementace jednoho z modelů. Metody *question*, *budgetStateChange* a *quizzes* implementují právě výše zmíněné vztahy mezi entitami s pomocí volání funkcí, atribut *fillable* reprezentuje atributy entity (sloupce v databázi), kterým je možno přiřadit hodnotu hromadně. Hromadné přiřazení umožňuje snadnější vytváření více entit najednou, ve výchozím stavu nejsou žádné atributy povoleny, kvůli zvýšení bezpečnosti, jelikož by škodlivý kód od uživatele mohl nečekaně změnit citlivé informace v databázi. Tuto možnost sice nikde nevyužívám, ale je to základ pro možné rozšíření do budoucna.

**Výpis kódu 3.2** Úryvek kódu migrace

```
Schema::create('budget_state_changes', function (Blueprint $table) {
    $table->id();
    $table->timestamps();
    $table->bigInteger('income_first_year');
    $table->bigInteger('expense_first_year');
    $table->bigInteger('income_second_year');
    $table->bigInteger('expense_second_year');
    $table->bigInteger('income_third_year');
    $table->bigInteger('expense_third_year');
    $table->foreignId('budget_chapter_id')
        ->constrained()
        ->onUpdate('cascade')
        ->onDelete('cascade');
    $table->foreignId('answer_id')
        ->constrained()
        ->onUpdate('cascade')
        ->onDelete('cascade');
});
```

### 3.3.2 Tvorba migrací

Migrace slouží k verzování struktury databáze, umožňuje jednoduše tvorbu struktury databáze v rámci aplikace. Migrace se uchovávají v adresáři *database/migrations*. Každá migrace je reprezentována třídou, která má dvě hlavní metody *up* a *down*. Metoda *up* slouží pro zavedení změn ve struktuře databáze, metoda *down* slouží k navrácení těchto změn. [65]

Vytvořil jsem jednu hlavní migraci pro zavedení základní struktury databáze pro aplikaci. V této migraci je obsažen kód pro vytvoření všech potřebných tabulek pro tuto aplikaci. Ve výpisu kódu 3.2 je vyobrazena část migrace, která vytváří tabulku *budget_state_changes* odpovídající tabulce *Změna stavu státního rozpočtu* na obrázku 2.4. Nejprve se přidá sloupec pro identifikátor, poté se přidají sloupce *created_at* a *updated_at* za pomocí příkazu *timestamps*, tyto dva sloupce slouží pro uchování data vložení a aktualizace záznamu, jsou automaticky vyplňovány za pomocí technologie Eloquent. Další sloupce jsou stejné atributy jako jsou definovány na výše zmíněném obrázku, akorát přeloženy do angličtiny, všechny mají typ *bigInteger*, což zajistí velký rozsah čísel se kterým je možno pracovat. Dále tato část migrace přidává odkaz na kapitolu rozpočtu a odpověď na otázku, při smazání těchto záznamů se automaticky smaže i tento záznam (toto indikuje příznak *cascade*).

### 3.3.3 Úprava třídy pro vkládání výchozích hodnot

V základu aplikace je připravena třída pro vkládání výchozích hodnot do databáze (Seeder). Soubor s touto třídou je uložen v adresáři *database/seeders*. Tyto data se vkládají do databáze za pomocí budování SQL dotazů díky funkcionalitám technologie Laravel. Zde jsem vložil data pro aplikaci, která jsem navrhl v sekci 2.5. Ve výpisu kódu 3.3 je úryvek z tohoto souboru, který vkládá do databáze data pro nejvyšší dosažené vzdělání, nastavuje čas vytvoření a aktualizace vytvářeného záznamu na aktuální čas.

**Výpis kódu 3.3** Úryvek kódu pro vložení dat do databáze

```
DB::table('educations')->insert([
    ['created_at' => 'now()', 'updated_at' => 'now()',
     'name' => 'Základní vzdelání'],
    ['created_at' => 'now()', 'updated_at' => 'now()',
     'name' => 'Strední vzdelání'],
    ['created_at' => 'now()', 'updated_at' => 'now()',
     'name' => 'Vyssí odborné vzdelání'],
    ['created_at' => 'now()', 'updated_at' => 'now()',
     'name' => 'Vysokoskolské vzdelání'],
    ['created_at' => 'now()', 'updated_at' => 'now()',
     'name' => 'Jiné']
]);
```

## 3.4 Vytváření jednotlivých podstránek

Pro vytváření jednotlivých podstránek, jak v rámci administrace, tak i uživatelské části, jsou potřeba stejné základní stavební bloky – kontroléry, cesty pro směrovač a pohledy. U administrátorské části navíc využívám třídy pro validaci požadavků.

### 3.4.1 Tvorba kontrolérů

Kontroléry řídí jednotlivé akce, které se mají vykonat s daným požadavkem, na vstup dostane požadavek od uživatele, po vykonání předem definované akce vrací odpověď. [66]

Artisan umí vygenerovat jak běžné kontroléry, tak i kontroléry pro zdroje dat, u kterých rovnou vygeneruje metody pro zobrazení, vytvoření, úpravu a smazání těchto dat, nicméně kód těchto metod je potřeba doplnit ručně.

### 3.4.2 Tvorba cest pro směrovač

Cesty pro směrovač pro webovou stránku jsou definovány v souboru *routes/web.php*, cesty definované v tomto souboru jsou automaticky přiřazené do skupiny *web*. Jednotlivé cesty se definují pro různé HTTP metody (*GET*, *POST*, *PATCH*, *DELETE*...), poté směrovač vybere konkrétní cestu která splňuje dané pravidla. Směrovací cesty (route) se mohou seskupovat do skupin a přiřazovat jim společné vlastnosti, seskupit se dají manuálně do ručně vytvořené skupiny, nebo pomocí předpony v adrese podstránky, případně dle kontroléru. Pokud se seskupí jednotlivé cesty dle kontroléru, usnadní se tím odkazování na jednotlivé metody kontroléru. Je možné tyto cesto také seskupit pro kontroléry pro zdroje dat, tohle zajistí vygenerování odpovídajících cest pro směrovač – je možné také některé nechtěné generované cesty vynechat. Jednotlivé cesty je možné pojmenovávat, což usnadní generování odkazů. Jak skupinám, tak i jednotlivým cestám lze přiřazovat middleware, přes který musí požadavek projít.[67]

#### 3.4.2.1 Ochrana před CSRF útoky

Pro cesty směrovače v souboru *routes/web.php* je povolen middleware, který přináší ochranu před útokem typu CSRF (Cross–Site Request Forgery). Tento útok se odehrává tak, že jiná webová aplikace vytvoří například formulář, který se automaticky odešle po načtení stránky, pokud by tento formulář směroval na stránku se zranitelností proti CSRF útokům, tak by mohl změnit například email uživatele, bez jeho vědomí, stačilo by akorát, že by uživatel byl přihlášený, protože tento požadavek by se vykonal jeho prohlížečem. Laravel tohle řeší s pomocí sezení (sessions),

za pomocí kterých se ukládají data mezi požadavky (jelikož HTTP je ve výchozím stavu bezestavový – nepřenáší stav mezi jednotlivými požadavky, tedy neví například nic o přihlášení uživatele), a tedy ukládá se zde například jak údaje o přihlášení, tak i údaje o klíči proti CSRF útoku. Tento klíč se poté vkládá do jednotlivých formulářů a požadavků uživatele a automaticky se zkontroluje, že je shodný s klíčem uloženým v datech o uživateli (sezení – session). [67, 68, 69]

### 3.4.3 Tvorba pohledů

Pohledy se vytváří ve složce *resources/views*, pohledy z této složky lze vracet jako odpovědi jak v kontrolérech, tak přímo v definicích jednotlivých směrovacích cest za pomocí funkce *view*. [70]

Pohledy v technologii Laravel využívají šablonový systém Blade, což je systém, který usnadňuje tvorbu pohledů, v základu to jsou makra psána v PHP, která se při požádaní konkrétního souboru zkompilují do obyčejného PHP souboru (pokud již není vygenerován, případně tento soubor byl změněn od poslední kompilace). Pohledy se mohou členit do komponent – znovupoužitelných celků. Tyto celky se poté dají využít na různých místech stránky. Je možné udělat buď jednoduchou komponentu, která má pouze soubor šablony, nebo také pokročilejší komponentu, která má navíc i třídu, která ji inicializuje, zde může proběhnout zpracování vstupních dat. Jednotlivé komponenty mají atributy, které se předávají při požadavku na vykreslení jednotlivé komponenty a také sloty, které reprezentují obsah který se má vypsat uvnitř komponenty. Tyto sloty je také možné specifikovat pomocí názvu, pokud komponenta tento slot bude podporovat, tak jej může například vypsat na speciálním místě. [71]

Komponenty pohledů využívám pro tvorbu hlavního společného vzhledu stránky, jelikož se jednotlivé stránky liší pouze vnitřním obsahem, ale také i pro jednotlivé části stránky, například odkazy v hlavičce stránky, případně tlačítka v administraci.

#### 3.4.3.1 Vložení CSS a Javascriptu do pohledů

Pro vzhled a rozšíření funkcionality je potřeba integrace CSS a Javascript souborů v pohledech.

#### 3.4.3.2 Sass a Bootstrap

Rozhodl jsem se pro použití Sass (Syntactically Awesome Stylesheet), což je dle [72] rozšíření pro CSS, které pomáhá psát CSS kód efektivněji. Bootstrap se distribuuje mimo jiné ve formátu Sass a proto jsem se pro toto rozšíření rozhodl. Tohle rozšíření používá soubory přípony *.scss*, což indikuje, že tyto soubory se musí před použitím v pohledu zkompilovat do souborů CSS, tento proces popíšu v sekci 3.4.3.4.

Bootstrap jsem nejdříve nainstaloval za pomocí nástroje npm. Poté jsem jej integroval do souboru *app.scss*, pomocí příkazu *@import*. Tohle umožní používání technologie Boostrap, tam kde bude vložen tento soubor. Bootstrap definuje CSS třídy, které se přiřazují jednotlivým HTML elementům a tím se mění jejich vzhled.

#### 3.4.3.3 Javascript

Pro interaktivitu aplikace je potřeba Javascript, který se umisťuje do složky *resources/js*. V tomto adresáři je soubor *app.js*, který je obecným souborem, který vkládám na každou stránku, zde již byl integrován základní soubor, ve kterém se konfiguruje Axios, přidal jsem zde integraci Javascript části technologie Bootstrap.

#### 3.4.3.4 Kompilace

Tyto jednotlivé zdroje Javascript a Sass je potřeba kompilovat do výsledné použitelné podoby. Webpack je dle [73] nástroj, který vytváří balíčky Javascript souborů. Tohoto využívá Laravel

**Výpis kódu 3.4** Nastavení Laravel Mix

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

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

**Výpis kódu 3.5** Základ vzhledu pro administraci

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{{ config('APP_NAME') }}</title>

    <link rel="stylesheet" href="{{mix('css/app.css')}}" type="text/css">
    <link rel="stylesheet" href="{{mix('css/admin.css')}}" type="text/css">
    <script src="{{mix('js/app.js')}}"></script>
</head>
<body>
    {{$slot}}
</body>
</html>
```

Mix, což je technologie která využívá právě Webpack, definuje metody, které využívají jednotlivé sestavovací a kompilovací procesy technologie Webpack. [74]. Kompilace se spouští pomocí nástroje npm. Laravel Mix umí zkompilovat jak obyčejné Javascript soubory, tak i Javascript soubory pro React a Sass soubory. Nastavení nástroje Laravel Mix je ukazáno ve výpisu kódu 3.4, zde je například vidět nastavení pro kompilaci souboru *resources/js/app.js*, kde se výsledný soubor uloží do složky *public/js*, obdobně se zkompiluje i soubor *react.js*, ke kterému se ale navíc přidá podpora pro kompilaci souborů psaných pro React. Dále je zde kód pro kompilaci jednotlivých Sass souborů.

## 3.5 Tvorba administrační části

Poté co jsem měl vytvořen základ pro přístup k datům aplikace, jsem mohl vytvořit administraci aplikace. Administrace je dostupná pouze po přihlášení, takže jsem potřeboval vytvořit vynucení pro přihlášení uživatele, což poskytuje již implementovaný middleware *Auth*. Dále jsem potřeboval vytvořit společnou šablonu pro celou administraci, ve které se bude vkládat potřebné zkompilované Javascript a CSS soubory. Tento základ vzhledu je realizován formou komponenty pro systém Blade a je vyobrazen ve výpisu kódu 3.5, titulek stránky je nastaven na hodnotu z konfiguračního souboru aplikace, jednotlivé soubory CSS a Javascript se vkládají do HTML stránky s pomocí funkce *mix*, která určí správné umístění souboru po zpracování nástrojem Laravel Mix, v proměnné *slot* je uložen obsah komponenty při použití. Tento základ se používá pro přihlašovací stránku a jako základ pro administrátorskou sekci po přihlášení.

**Výpis kódu 3.6** Úryvek z kódu pro definici směrovacích cest

```
Route::controller(LoginController::class)->group(function() {
    Route::post('admin/login', 'login');
    Route::get('admin/login', 'show')->name('admin.login');
    Route::get('admin/logout', 'logout')->name('admin.logout');
});
```

### 3.5.1 Konfigurace administrátorského účtu

Název a heslo administrátorského účtu se definuje v *.env* souboru aplikace. Zde se zadává heslo, které je zakódováno, tak aby nebylo snadno odhadnutelné z tohoto souboru. Pro to, aby se tohle heslo konfigurovalo snadno, jsem vytvořil příkaz pro Artisan příkazovou řádku, který generuje kód hesla zadaného na vstupu, který se pak zadá do tohoto konfiguračního souboru. Tento příkaz jsem vytvořil za pomocí souboru *HashPassword* ve složce *app/Console/Commands*, Laravel je schopen tento příkaz automaticky načíst, a pokud se shoduje definovaný předpis s tímto příkazem, pak je vykonán, tento příkaz zakóduje vstupní heslo za pomocí standardních funkcí dostupných v technologii Laravel.

### 3.5.2 Přihlašování do administrátorského účtu

Vytvořil jsem pohled pro přihlašovací formulář na základě oficiální šablony přihlašovacího formuláře pro Bootstrap, do kterého jsem přidal (jako u ostatních formulářů v aplikaci) Blade příkaz *@csrf*, který vygeneruje HTML vstup vyplněný s požadovaným klíčem pro ochranu před CSRF útokem. Dále jsem vytvořil kontrolér *LoginController*, který má tři metody. Metoda *show* slouží pro zobrazení přihlašovacího formuláře. Metoda *login* slouží pro samotné přihlášení uživatele, tato metoda kontroluje, že se zadaný název účtu a heslo shoduje s údaji v administraci, pokud se shodují, tak se využívá vestavěné funkce technologie Laravel pro přihlášení uživatele, po přihlášení se vygeneruje nové sezení (session) pro uživatele, aby se zamezilo útoku Session fixation, který spočívá ve znalosti identifikátoru sezení, a poté se útočník může vydávat za tohoto uživatele. [75]. Další metodou je *logout*, ve které se uživatel odhlásí pomocí vestavěné funkce v Laravelu, a následně se zneplatní sezení (session) a poté se vygeneruje nový klíč pro CSRF, tak jak je doporučeno v [75].

Když jsem měl vytvořený kontrolér, tak jsem mohl vytvořit jednotlivé směrovací cesty v souboru *web.php*. Tohle je vyobrazeno ve výpisu kódu 3.6. Zde je vidět definovaná cesta pro HTTP metodu *POST*, která zajišťuje odeslání formuláře, a dvě metody *GET* – jedna pro zobrazení formuláře a druhá pro odhlášení administrátora.

### 3.5.3 Správa dat aplikace

Po vytvoření přihlášení jsem mohl začít vytvářet správu jednotlivých tabulek v databázi. Vytvořil jsem kontroléry postupně pro všechny modely, pro každý kontrolér jsem také vytvořil potřebné pohledy. Navíc jsem vytvořil třídy pro validaci jednotlivých požadavků, tyto třídy definují metodu pro pravidla pro validaci a chybové hlášky pokud validace selže. Pokud požadavek tyto pravidla splní, tak postoupí dále v životním cyklu požadavku, jinak se uživatel přesměruje na předchozí stránku s chybovou hláškou. Vynucení této validace se docílí předepsáním typu požadavku (který symbolizuje právě validaci) v metodě kontroléru.

Správu dat by se dalo rozdělit na dvě části – jednodušší (ta nemá vazby na další modely) a složitější (vytváří vazby s dalšími modely). Mezi jednodušší správu dat vybírám ku příkladu správu kapitol státního rozpočtu. Pro tuto správu jsem vytvořil kontrolér *BudgetChapterController*, který má metodu *index*, která zobrazí pohled s výpisem všech vytvořených kapitol v tabulce.

**Výpis kódu 3.7** Metoda store kontroléru BudgetChapterController

```
public function store(BudgetChapterRequest $request)
{
    $validated = $request->validated();

    $budgetChapter = new BudgetChapter;

    $budgetChapter->number = $validated['budget_chapter_number'];
    $budgetChapter->name = $validated['budget_chapter_name'];

    $budgetChapter->save();

    return redirect()->route('admin.budget_chapters.index');
}
```

**Výpis kódu 3.8** Úryvek z kódu pro definici cesty pro směrovač pro správu kapitol státního rozpočtu

```
Route::prefix('admin')->name('admin.')
   ->middleware('auth')->group(function () {
   ...
   Route::resource('budget_chapters', BudgetChapterController::class)
    ->except('show');
   ...
});
```

Dále má metodu *create*, která slouží pro zobrazení formuláře pro vytvoření nové kapitoly, využívá pohledu, který je stejný jak pro vytvoření, tak úpravu kapitoly rozpočtu – pohled rozliší o jakou akci se jedná podle toho, jestli je mu předána existující kapitola nebo neexistující formou instance modelu *BudgetChapter*. Další metodou tohoto kontroléru je *store*, která přidává na základě dat z formuláře novou kapitolu. Kód této metody je vyobrazen ve výpisu kódu 3.7, kde je možné vidět, že se předává metodě *BudgetChapterRequest*, což je validovaný požadavek uživatele, dále se získá validovaný vstup, vytvoří se nová kapitola, do které se vloží data, uloží se a uživatel je přesměrován na výpis kapitol (kód byl pro vizualizační účely zkrácen). Metoda *edit* zobrazuje formulář pro úpravu již existující kapitoly státního rozpočtu. Metoda *update* slouží pro aktualizaci dat existující kapitoly rozpočtu na základě formuláře pro úpravu kapitoly. A v neposlední řadě metoda *destroy* slouží pro smazání záznamu o kapitole státního rozpočtu v databázi. Cesta pro směrovač je definována v rámci skupiny *admin*, ke které je přidělen middleware pro kontrolu přihlášení uživatele, využívá se zde generování cest pro zdroje dat, jak je znázorněno na výpisu kódu 3.8, je zde vynechána metoda *show*, která by sloužila pro zobrazení jednotlivé kapitoly rozpočtu, což mi ale přišlo zbytečné, jelikož zobrazení pro tento případ je dostatečně přehledné.

Správa krajů, politických stran a vzdělání je implementována obdobně jako správa státních kapitol. Správa počátečních stavů rozpočtu na základě kapitol byla mírně složitější na implementaci, využil jsem při definování cesty pro směrovač složených cest [67], kde se v adrese nejdříve uvede identifikátor kapitoly, a až poté jednotlivé operace nad počátečními stavy rozpočtu. Toto se promítne do definice metody například pro vytvoření nového počátečního stavu rozpočtu, jak je vyobrazeno na výpisu kódu 3.9, ve vynechané části kontroluji, že se nevytváří stav rozpočtu u kapitoly, která má již nějaký stav rozpočtu přiřazen, dále se zde obdobně jako u správy kapitol státního rozpočtu. Dále se vytváří nový stav státního rozpočtu, kde se přiřadí data dle dat vyplněných ve formuláři a následně se přiřadí do vztahu ke kapitole s pomocí uložení přes vztah vyobrazený na výpisu kódu 3.10, který přiřadí tento stav rozpočtu ke konkrétní kapitole.

Správa otázek je obdobná správě kapitol rozpočtu, ale navíc je zde také zobrazení jednotlivých

**Výpis kódu 3.9** Úryvek z kódu metody store kontroléru BudgetStateController

```
public function store(BudgetStateRequest $request,
                      BudgetChapter $budgetChapter)
{
    ...

    $validated = $request->validated();
    $budgetState = new BudgetState;

    ...

    $budgetChapter->budgetState()->save($budgetState);

    ...
}
```

**Výpis kódu 3.10** Metoda budgetState modelu BudgetChapter

```
public function budgetState()
{
    return $this->hasOne(BudgetState::class);
}
```

otázek, kde se zobrazuje také výpis přiřazených odpovědí ke konkrétní otázce, možnost jejich přidání, úprava a smazání. Správa odpovědí u otázky je obdobná jako správa počátečních stavů rozpočtu, jednotlivé odpovědi lze zobrazit a přejít na změnu stavu simulace rozpočtu kterou provádí. Správa změn stavů rozpočtů je obdobná jako správa odpovědí, ale je závislá jak na otázce, tak i odpovědi. U metod *create* a *update* ověřuji, že daná odpověď ještě nemá žádnou změnu stavu rozpočtu přiřazenou. Jelikož se u každé změny rozpočtu volí jakou kapitolu rozpočtu ovlivňují, musel jsem vyřešit jak zařadit tuto nově vytvářenou entitu do dvou vztahů najednou, jelikož by se jinak porušila referenční integrita databáze (neplatily by omezení vztahů mezi tabulkami), toho jsem docílil v metodě *store* nejprve vytvořením změny rozpočtu, ke které jsem nejprve přiřadil kapitolu státního rozpočtu a poté jsem tuto změnu státního rozpočtu přiřadil k odpovědi. Tento postup je zobrazen na výpisu kódu 3.11. Stejně jsem postupoval při aktualizaci již existující změny stavu rozpočtu v metodě *update*, jelikož se zde může změnit přiřazení ke kapitole státního rozpočtu.

## 3.6 Tvorba uživatelské části

Stejně jako pro administrátorskou část, tak i pro uživatelskou část vytvářím základ vzhledu, ve kterém definuji základní prvky vzhledu (zde záhlaví, blok pro obsah a zápatí), a vkládám potřebné CSS a JS soubory.

### 3.6.1 Tvorba kvízu

Pro uživatelskou část je důležitá technologie React, ve které je realizován samotný kvíz. Jednotlivé části kvízu jsou realizovány pomocí komponent v technologii React. Tyto komponenty jsou zapouzdřením jednotlivých částí uživatelského rozhraní, obecně zpracují vstup a vrátí výstup, který by se měl vykreslit.

**Výpis kódu 3.11** Metoda store kontroléru BudgetStateChangeController

```
public function store(BudgetStateChangeRequest $request,
                      Question $question, Answer $answer)
{
    ...

    $validated = $request->validated();
    $budgetStateChange = new BudgetStateChange();

    ...

    $budgetStateChange->budgetChapter()
        ->associate($validated['budget_state_change_chapter_id']);
    $answer->budgetStateChange()->save($budgetStateChange);

    ...
}
```

#### 3.6.1.1 React

Výstup v Reactu může být definován buď pomocí HTML elementů, nebo pomocí JSX (JavaScript XML) elementů, což je syntaktické rozšíření pro JavaScript, JSX poté vyprodukuje element pro React, který se může vykreslit. [76] Jednotlivé elementy jsou neměnné, tedy nedá se měnit jejich obsah, tím symbolizují záznam výstupu v určitém okamžiku, pro změnu obsahu se musí vygenerovat nový element, při vykreslení tohoto elementu, se v prohlížeči překreslí pouze změněná část HTML elementu, takže se nevykreslují zbytečné části. [77]

Jak jsem již zmínil, komponenty jsou důležitou stavební jednotkou pro React, jelikož zapouzdřují jednotlivé části uživatelského prostředí včetně jejich vzhledu a funkcionalit. Jejich vstup se nazývá *props*, a je dostupný skrz proměnnou stejného názvu, tento vstup je neměnný, tedy si udržuje svůj stav pořád stejný. Pro definici komponenty se může použít jak funkce (používá se spíše pro jednoduché komponenty), nebo třída, metoda *render* poté slouží pro získání výstupu, který se má vykreslit. [78].

Komponenty mohou mít také svůj vlastní stav, který dynamicky ovlivňuje vykreslovaný element. Tento stav se definuje u komponenty realizované třídou v konstruktoru definicí atributu *state*. Pro změnu tohoto stavu se používá funkce *setState*, díky čemu React ví, kdy se vnitřní stav komponenty změnil, a tedy může v příhodný čas vykreslit komponentu znovu. Komponenty definované pomocí tříd také definuje dvě metody, které se spouští v průběhu životního cyklu komponenty – *componentDidMount*, která se spustí při ukotvení komponenty na stránce, a *componentWillUnmount*, která se spustí při odstranění této komponenty ze stránky. [79]

#### 3.6.1.2 Tvorba kontroléru

Před samotným tvořením komponent pro React jsem nejprve potřeboval vytvořit zázemí pro tuto část v technologii Laravel. Tohle zázemí je tvořeno kontrolérem pro získaní potřebných dat pro dotazník (seznam krajů, politických stran a vzdělání), seznam otázek jejich odpovědí a vliv odpovědí na státní rozpočet, a počáteční stav státního rozpočtu pro simulaci státního rozpočtu. Dále tento kontrolér zaznamenává zahájení vyplňování kvízu a zaznamenává jednotlivé odpovědi u konkrétního kvízu.

Vytvořil jsem tedy kontrolér *QuizController* ve složce *Http/Controllers/Api*, jelikož se jedná o API aplikace. Tento kontrolér vždy vrací odpověď na požadavek formou JSON formátu. V metodě *data* získává za pomocí modelů potřebná data, které transformuje za pomocí tříd pro transformací zdrojů z databáze do PHP polí, v této transformaci se definuje jaké atributy se mají

**Výpis kódu 3.12** Metoda store kontroléru Api/QuizController

```
public function store(QuizRequest $request)
{
    $validated = $request->validated();

    $hash = Hash::make(time() . $request->ip() . random_int(-1000, 1000));

    $quiz = new Quiz;

    $quiz->hash = $hash;
    $quiz->age = $validated['quiz_age'];

    $quiz->region()->associate($validated['quiz_region']);
    $quiz->party()->associate($validated['quiz_party']);
    $quiz->education()->associate($validated['quiz_education']);

    $quiz->save();

    return [
        'hash' => $hash
    ];
}
```

zobrazit a jak mají být pojmenovány. Laravel poté tohle pole automaticky transformuje na JSON. Metoda *store* je vyobrazena na úkazce kódu 3.12, přijímá validovaný požadavek pomocí validátoru *QuizRequest*, který kontroluje zda jsou všechny potřebné data zadána korektně v požadovaném formátu (například že je věk v určitém rozmezí, kraj existuje v databázi). Po validaci vstupu vytváří kód pro identifikaci konkrétního kvízu, za pomocí pomocné třídy *Hash* (která se mimo jiné využívá také pro zakódování hesel), vstupem je aktuální čas, IP adresa uživatele a náhodné číslo, což by mělo zajistit dostatečnou náhodnost tohoto kódu. Důvodem pro zavedení tohoto kódu je, že je to identifikátor pro konkrétní kvíz, kdybych použil například číselný identifikátor, tak by poté kdokoliv mohl vyplnit odpovědi ke kvízu jinému uživateli, což by bylo nechtěné. Poté se vytvoří nový záznam pro kvíz, vyplní se odpovídající data a přiřadí se ke kvízu odpovídající zvolené položky z dotazníku, nakonec se uživateli odešle identifikační kód tohoto kvízu. Metoda *answer* slouží ke zaznamenání odpovědi ke kvízu, vstupuje zde validovaný požadavek skrze validátor *QuizAnswerRequest*, který kontroluje existenci vstupních dat (kód kvízu, vybraná odpověď u konkrétní otázky) a že tato odpověď již nebyla zvolena a že tato otázka nebyla již zodpovězena. Pokud počet odpovědí bude po zodpovězení této otázky stejný jako počet otázek, poté se označí kvíz jako dokončený, smaže se u něj identifikační kód, a v odpovědi se odešle identifikační číslo pro zobrazení výsledků kvízu.

#### 3.6.1.3 Tvorba komponent pro React

V této aplikaci používám pouze React komponenty realizované třídami. Hlavní komponentou je *QuizBase*, která v metodě *componentDidMount* za pomocí Axios získá data přes API aplikace. Dále obsahuje pomocné funkce na nastavení chybového stavu komponenty (*error*), a aktualizaci identifikačního kódu kvízu (*hashUpdated*), těmto metodám se musí v konstruktoru přiřadit význam klíčového slova *this*, jelikož v jiném kontextu volání by mohlo *this* odkazovat na nečekanou hodnotu. Tohle přiřazení jednoznačného významu proměnné *this* je využito i v ostatních komponentách. Tato komponenta vykresluje na základě svého stavu buď komponentu pro zobrazení chyby, nebo indikátor načítání, nebo dotazník, nebo kvíz.

Komponenta pro dotazník *Questionnaire* představuje formulář, který je spravován přes Re-

act. Jednotlivé změny vstupů se propíšou za pomocí události do stavu komponent a jakékoliv změny stavu vstupů v rámci komponenty se propíšou do formuláře, tedy komponenta je s tímto formulářem plně provázána. Při odeslání dotazníku se pošle HTTP požadavek na API HTTP metodou *POST*, pokud je všechen vstup validní, server pošle odpověď s identifikačním kódem nově vytvořeného kvízu, tento kód se propíše do komponenty *QuizBase* za pomocí metody, která je předána jako vstup do komponenty *Questionnaire*. Pokud nastanou jakékoliv chyby při validaci vstupu, tak se jednotlivé chyby vypíšou k odpovidajícím polím ve formuláři.

Komponenta pro kvíz *Quiz* představuje samotný kvíz včetně simulace státního rozpočtu. V konstruktoru se nejdříve nastaví výchozí stav této komponenty, včetně výpočtu počátečního stavu simulace rozpočtu na základě předtím získaných dat přes API. Metoda *selectAnswer* je vyvolána při zvolení odpovědi, nastaví stav této komponenty na odpovídající zvolenou odpověď a její vliv na rozpočet. Metoda *submitAnswer* je spuštěna při pokusu o odeslání odpovědi, pokud žádná odpověď není zvolena, tak je uživatel vyzván pro vybrání nějaké odpovědi. Za pomocí Axios se odešle požadavek na API pro zaznamenání odpovědi, pokud je po zaznamenání této odpovědi kvíz již hotový, tak proběhne přesměrování uživatele na výsledky tohoto kvízu, jinak se promítne změna rozpočtu zvolené do aktuálního stavu simulace státního rozpočtu. V metodě *render* definuji dvě pomocné proměnné pro formátovaní částek v kvízu – jedna slouží pro formátování pro českou měnu, druhá navíc zobrazí znaménko (toho se využívá při promítnutí vlivu aktuální zvolené odpovědi do simulace rozpočtu). Ve výpisu kódu 3.13 je vyobrazeno vytvoření elementů za pomocí JSX, ve kterém se definuje vazba na komponenty *Question* a *BudgetSimulation*, těmto komponentám se předávají potřebná data, také je zde vyobrazeno podmíněné zobrazení chybové hlášky, za pomocí konstruktu *hodnota A ZÁROVEŇ hodnota*, tento konstrukt funguje tak, že když levá hodnota se vyhodnotí na pravdivou hodnotu, tak se vykoná zároveň i pravá hodnota, jinak výsledek celého tohoto bloku je nepravdivá hodnota, kterou React při vykreslování ignoruje. Dále je zde vidět přiřazení události pro tlačítko na uložení odpovědi pomocí atributu elementu.

Komponenta *Question* reprezentuje jednotlivou otázku, která se vykreslí spolu se svými odpovědmi, a přehledem jejich vlivu na simulaci státního rozpočtu. Tato komponenta také řídí volbu odpovědi, poté tuto zvolenou odpověď propaguje do komponenty *Quiz* za pomocí metody definované v proměnné *props*.

Komponenta *BudgetSimulation* zobrazuje aktuální stav simulace státního rozpočtu formou tabulky, ve sloupcích jsou souhrnné hodnoty výdajů, příjmů a výsledku státního rozpočtu. V jednotlivých řádcích jsou poté vypsány aktuální hodnoty pro jednotlivé roky, a navíc také změny simulace na základě zvolené odpovědi (pokud odpověď tuto změnu definuje).

#### 3.6.1.4 Zobrazení kvízu

Pro zobrazení kvízu využívám metodu *index* kontroléru *QuizController* (odlišný od kontroléru pro API), tato metoda je navazána na směrovací cestu definovanou v souboru *web.php*. V této metodě se definuje zobrazení pohledu pro zobrazení kvízu. Definice tohoto pohledu je vyobrazena na výpisu kódu 3.14. Tento pohled definuje základní element pro komponentu *QuizBase*, která se na tento element naváže. Dále se vkládá zkompilovaný soubor za pomocí Laravel Mix, který obsahuje tuto základní komponentu včetně všech jejich dílcích komponent a závislostí. Soubor *react.js* který vstupuje do kompilace je vyobrazen na výpisu kódu 3.15. V tomto souboru se nejprve načte komponenta *QuizBase*, poté React a jeho část pro vykreslování komponent. Poté se definuje kořenový element pro komponenty (tento element je definován právě ve výše zmíněném pohledu), a vykreslí se komponenta *QuizBase* do tohoto kořenového elementu. Kompilace těchto komponent je usnadněná díky Laravel Mix a metodě *react*, která přidává podporu pro kompilaci JSX syntaxe.

**Výpis kódu 3.13** Úryvek z metody render komponenty Quiz

```
...

return (
    <div className="quiz">
        <Question question={question}
            onSelectAnswer={this.selectAnswer}
            signFormatter={signFormatter}/>

        {this.state.error_message && (
            <div className="error_message alert alert-danger">
                {this.state.error_message}
            </div>
        )}

        <a href="#"
            className="btn btn-primary mb-5" onClick={this.submitAnswer}>
            {this.state.next_question_btn_text}
        </a>

        <BudgetSimulation budget={this.state.budget}
                          current_budget_state_change=
                          {this.state.current_budget_state_change}
                          formatter={formatter}
                          signFormatter={signFormatter} />
    </div>
);
...
```

**Výpis kódu 3.14** Pohled pro zobrazení kvízu

```
<x-layout>
    <h1>Kvíz</h1>
    ...
    <div id="quizReact"></div>

    <script src="{{ mix('js/react.js') }}"></script>

</x-layout>
```

**Výpis kódu 3.15** Obsah souboru react.js

```
import QuizBase from "./components/QuizBase";
import React from "react";
import ReactDOM from "react-dom/client";

const root = ReactDOM.createRoot(document.getElementById('quizReact'));
root.render(<QuizBase />);
```

**Výpis kódu 3.16** Získání nejčetnějších odpovědí na jednotlivé otázky pro dokončené kvízy

```
$answersByTopCount = Answer::withCount(['quizzes' =>
    function(Builder $builder) {
        $builder->where('is_finished', true);
    }])->orderBy('quizzes_count', 'desc')->get();

$questionIds = [];

$topAnswers = $answersByTopCount->filter(
    function ($answer, $key) use (&$questionIds) {
        if (in_array($answer->question_id, $questionIds)) {
            return false;
        } else {
            $questionIds[] = $answer->question_id;
            return true;
        }
    }
);

$questionToAnswer = [];

foreach ($topAnswers as $top_answer) {
    $questionToAnswer[$top_answer->question_id] = $top_answer;
}
```

### 3.6.2 Zobrazení výsledků

Aplikace zobrazuje výsledky za pomocí kontroléru *QuizController* a metody *quizResults* pro zobrazeni výsledku konkrétního kvízu a metody *results* pro zobrazení výsledků všech dokončených kvízů.

#### 3.6.2.1 Zobrazení výsledku vyplněného kvízu

Pro vizualizaci výsledku jsem zvolil zobrazení výsledků státního rozpočtu po aplikování všech změn na základě odpovědí na počáteční stavy rozpočtu. Dále zde jsou zobrazeny v tabulce odpovědi na jednotlivé otázky a výpis nejčastější odpovědi respondentů na tuto otázku.

Metoda *quizResult* realizuje toto zobrazení výsledků, požaduje na vstupu model kvízu, ten je předán této metodě za pomocí směrovače na základě číselného identifikátoru. Zde již nevadí, že je možné změnit identifikátor kvízu na libovolný vyplněný kvíz, jelikož zde nejsou zobrazovány žádné citlivé data a funkci aplikace to nijak neovlivní. V této metodě se nejprve kontroluje jestli nalezený kvíz je dokončen, pokud není, aplikace vygeneruje chybovou hlášku s HTTP kódem 404 – nenalezeno. Nejčetnější odpovědi se počítají tak, že se nejdříve vyberou z databáze všechny odpovědi, které jsou přiřazeny v dokončených kvízech, vypočítá se počet výskytů těchto odpovědí zaznamenaných jako zvolené odpovědi ve kvízech, a postupně se seřadí od nejvyššího počtu. Tyto odpovědi se vyfiltrují tak, aby ke každé otázce zůstala pouze jedna nejčetnější odpověď. Poté se vytvoří mapování mezi identifikátorem otázky a identifikátorem nejčastější odpovědi, pro snadnější využití dále. Tento postup je vyobrazen na výpisu kódu 3.16. Poté se získají odpovědi uživatele na jednotlivé otázky a spočítá se konečný stav simulace státního rozpočtu pro všechny 3 roky. A tyto data se výpišou za pomocí pohledu.

**Výpis kódu 3.17** Zpracování dat pro zobrazení výsledků kvízů dle politických stran

```
$parties = Party::all();
$partyToAnswersCount = [];
foreach ($parties as $party) {
    $answers = Answer::withCount(['quizzes' =>
        function(Builder $builder) use ($party) {
            $builder->where('is_finished', true)
                ->where('party_id', $party->id);
    }])->get();

    foreach ($answers as $answer) {
        $partyToAnswersCount[$party->id][$answer->id] =
            $answer->quizzes_count;
    }
}
```

#### 3.6.2.2 Zobrazení výsledků všech kvízů

Pro zobrazení všech výsledků jsem zvolil vyhodnocení počtu odpovědí dle jednotlivých preferencí politických stran respondentů. Pro každou politickou stranu jsou vypsány všechny otázky a u každé otázky všechny odpovědi spolu s počtem zvolených odpovědí respondentů, kteří by volili konkrétní politickou stranu.

Metoda *results* realizuje zobrazení všech výsledků. Nejprve získá z databáze všechny politické strany evidované v aplikaci, ke každé z nich poté vypočítá počet odpovědí kvízů, ve kterých uživatel vyplnil, že by volil konkrétní stranu. Poté použiji dvojité mapování počtu odpovědí na odpověď, pro snadný přístup – nejprve dle identifikátoru pro politickou stranu a poté podle identifikátoru identifikátoru dané odpovědi, tohle řešení usnadní přístup k počtům odpovědi za pomocí dělení podle politických stran. Tento kód je vyobrazen na výpisu kódu 3.17.

# Kapitola 4
# Testování a dokumentace

*V této kapitole popisuji, jak jsem aplikaci testoval a dokumentoval.*

## 4.1 Testování

Aplikaci jsem testoval jak za pomocí automatizovaného testování, ale také i manuálně. Manuálně jsem zkontroloval všechny součástí aplikace. Jelikož jsou lidé náchylní na chyby, tak jsem také aplikaci doplnil o programové testy.

Laravel podporuje tvorbu testů a jejich spouštění za pomocí nástroje PHPUnit, a tedy při testování je možné použít jakoukoliv funkcionalitu tohoto nástroje. Podporuje dva typy testů – jednotkové, které testují izolovanou část kódu (většinou metodu), při těchto testech není k dispozici žádná část technologie Laravel, druhým typem testů jsou funkční, které testují část funkcionality systému. Laravel dává k dispozici nástroje pro testování HTTP požadavků na aplikaci, tyto požadavky se ve skutečnosti neprovádí skrze HTTP protokol, ale vsunou se aplikaci tvářící se jako takovýto požadavek. Další užitečnou funkcí pro testování je práce s databází, za pomocí použití rozšíření třídy *RefreshDatabase* se po každém provedení testu uvede databáze do původní podoby před zahájením testu. [80, 81, 82]

### 4.1.1 Jednotkové testování

Jednotkové testy jsou uloženy ve složce *tests/Unit*. Jednotkové testování testuje pouze část kódu, a nespouští při tom žádné služby dostupné pro aplikaci, tak jsem pro testování vybral pouze model *Party*, kterému jsem v průběhu implementace přidal funkcionalitu pro získání jména politické strany, jak by se mělo zobrazovat napříč aplikací – ve formátu *(zkratka) – název strany*, případně pokud není zkratka strany k dispozici, tak jen ve formátu *název strany*. Tento test je realizován v souboru *PartyTest*. Ukázka testu je zobrazena na výpisu kódu 4.1.

### 4.1.2 Funkční testování

Funkční testy jsou uloženy ve složce *tests/Feature*. Za pomocí funkčních testů jsem otestoval komplexnější funkcionalitu aplikace. Testoval jsem přístup za pomocí požadavku na aplikaci, kde jsem přistoupil na konkrétní adresu a zkontroloval odpověď. Laravel umožňuje kontrolu, zda v odpovědi jsou chyby, které vznikly při validaci, toho jsem využíval.

Vytvořil jsem testy postupně pro otestování správy (vytvoření, úpravu, případné zobrazení a smazání) krajů, politických stran, vzdělání, kapitol státního rozpočtu a jejich počátečních stavů, otázek, odpovědí a jejich vlivu na státní rozpočet. Pro testování adres, které spadají pod

**Výpis kódu 4.1** Jednotkové testování modelu Party

```
public function test_display_name_with_short_name()
{
    $party = new Party();
    $party->name = 'Test';
    $party->short_name = 'TST';

    $this->assertTrue($party->displayName() === '(TST) - Test');
}
```

**Výpis kódu 4.2** Metoda setUp třídy AdminTestCase

```
public function setUp(): void
{
    parent::setUp();

    $this->user = new User();
    $this->user->username = 'testing user';
    $this->user->password = 'test';
    $this->user->save();
    $this->actingAs($this->user);
}
```

administrační část, jsem vytvořil základ *AdminTestCase*, ze kterého jsem tvořil jednotlivé třídy pro testy. Metoda *setUp* se spouští před každým testem, slouží pro nastavení prostředí pro každý test. V tomto základu jsem tedy využil této metody, pro vytvoření uživatelského účtu, který bude simulovat přihlášení pro jednotlivé požadavky, tím docílím simulace přihlášení do administrace. Tato metoda je vyobrazena ve výpisu kódu 4.2. V tomto základu také definuji, že po každém spuštění testu se mají vymazat data v databázi.

Testování spravování dat je provedeno ve všech testech obdobně, každý test se liší v tom, s jakými daty pracuje a jaké vlastnosti ověřuje. Například testovací třída *BudgetChapterTest* která slouží pro testování správy kapitol státního rozpočtu, v metodě testující jestli aplikace opravdu neumožní vytvoření kapitoly státního rozpočtu s číslem kapitoly, které by nebylo číslo. Toho se docílí pomocí simulace HTTP metody *PUT* s patričnými daty, jak je vidět ve výpisu kódu 4.3.

Provádění těchto testů má destruktivní vliv na databázi aplikace, tedy po provedení těchto testů je nutné obnovit data v databázi za pomocí *DatabaseSeeder*.

**Výpis kódu 4.3** Test nečíselné hodnoty čísla kapitoly státního rozpočtu

```
public function test_creation_non_numeric_number()
{
    $response = $this->post('/admin/budget_chapters', [
        'budget_chapter_number' => 'abcd',
        'budget_chapter_name' => 'Test'
    ]);

    $response->assertStatus(302);
    $response->assertSessionHasErrors(['budget_chapter_number']);
}
```

## 4.2 Dokumentace

Složitější části kódu jsem opatřil komentáři, které dokumentují danou funkcionalitu. Snažil jsem se dodržovat standardní zvyky (popsané v dokumentaci Laravelu) pro aplikace vytvářené za pomoci technologie Laravel, aby kód aplikace byl co nejvíce samo-popisný a zřetelný.

Také jsem vypracoval návod pro zprovoznění aplikace, který je v angličtině k dispozici na [55], nebo v češtině v příloze B.

# Závěr

Cílem této práce bylo vytvořit aplikaci, která by umožnila simulaci státního rozpočtu formou kvízu. Kde jednotlivé odpovědi na otázky ovlivní simulovaný výsledek hospodaření státu.

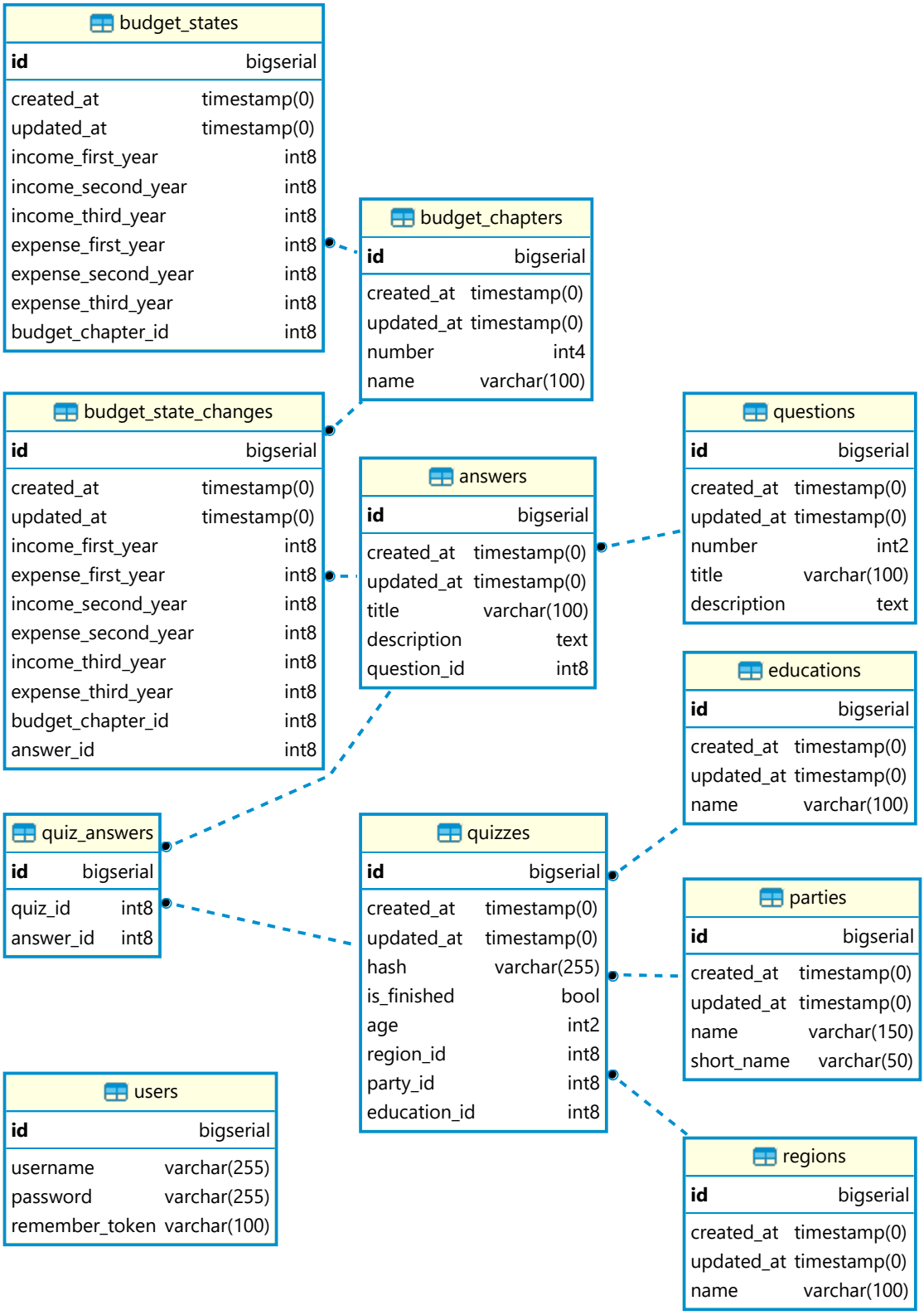
Výsledkem je webová aplikace členěná do administrátorské a uživatelské části. Administrátorská část je dostupná pouze po přihlášení do administrátorského účtu. Umožňuje nastavení výchozího stavu státního rozpočtu na 3 roky pro simulaci dle jednotlivých kapitol, tvorbu otázek a odpovědí pro kvíz, u každé odpovědi se nastavuje jakou kapitolu rozpočtu ovlivňuje. Uživatelská část je dostupná bez přihlášení, uživatel si zde může vyplnit již zmiňovaný kvíz po vyplněních svých základních údajů. V této části je také možnost zobrazení analýzy všech vyplněných odpovědí u dokončených kvízů. Pro výběr vhodných technologií jsem provedl analýzu. Aplikace je napsána v technologii Laravel pro programovací jazyk PHP, je doplněna technologií React pro programovací jazyk JavaScript, pro zvýšení interaktivity aplikace. Odpovídaní na kvízové otázky je realizováno s pomocí technologie React, která odesílá asynchronní dotazy na API vytvořené v technologii Laravel. Základ uživatelského rozhraní je tvořen pomocí HTML a stylizován pomocí technologie Bootstrap. Databáze je vytvořena v technologii PostgreSQL podle vytvořeného schématu databáze na základě analýzy.

Vhodné rozšíření pro aplikaci by mohlo být doplnění o mobilní aplikaci, jelikož je již dostupné API pro hlavní uživatelskou část, čehož by tahle mobilní aplikace mohla využit. Dále by bylo možné například doplnit další způsoby vyhodnocení vyplněných odpovědí uživatelů.

Příloha A

# Schéma databáze vygenerované aplikací

Schéma databáze aplikace je vygenerováno za pomocí programu DBeaver.


budget_states
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
income_first_year int8
income_second_year int8
income_third_year int8
expense_first_year int8
expense_second_year int8
expense_third_year int8
budget_chapter_id int8
budget_chapters
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
number int4
name varchar(100)
budget_state_changes
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
income_first_year int8
expense_first_year int8
income_second_year int8
expense_second_year int8
income_third_year int8
expense_third_year int8
budget_chapter_id int8
answer_id int8
answers
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
title varchar(100)
description text
question_id int8
questions
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
number int2
title varchar(100)
description text
educations
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
name varchar(100)
quiz_answers
id bigserial
quiz_id int8
answer_id int8
quizzes
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
hash varchar(255)
is_finished bool
age int2
region_id int8
party_id int8
education_id int8
parties
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
name varchar(150)
short_name varchar(50)
users
id bigserial
username varchar(255)
password varchar(255)
remember_token varchar(100)
regions
id bigserial
created_at timestamp(0)
updated_at timestamp(0)
name varchar(100)

# Příloha B

# Návod pro zprovoznění aplikace

Tento návod předpokládá nainstalovaný Docker.

1. Ve složce *app/docker/postgres* zkopírujte soubor *.env.example* jako soubor *.env*.
2. Ve složce *app* spusťte příkaz v příkazové řádce `docker-compose up -d`, tohle vytvoří Docker kontejner.
3. Ve složce *app/src* zkopírujte soubor *.env.example* jako soubor *.env*.
4. Otevřete příkazovou řádku pro Docker kontejner *php-apache*, tohoto je možno docílit za použití příkazu v příkazové řádce `docker-compose exec php-apache /bin/bash`.
   a. Spusťte příkaz `composer install`.
   b. Spusťte příkaz `npm install`.
   c. Pokud chcete změnit výchozí administrátorské heslo, tak spusťte příkaz `php artisan password:hash VAŠE_HESLO_ZDE` a zkopírujte vygenerovaný text do souboru *src/.env* k položce *ADMIN_PASSWORD_HASH*.
   d. Spusťte příkaz `php artisan key:generate`.
   e. Spusťte příkaz `php artisan migrate:fresh --seed`
5. Pokud aplikace hlásí problémy s oprávněním, tak spusťte v Docker kontejneru *php-apache* příkaz `chmod -R 755 storage`.

# Bibliografie


1. DVOŘÁK, Pavel. *Veřejné finance, fiskální nerovnováha a finanční krize*. V Praze: C.H. Beck, 2008. ISBN 978-80-7400-075-1.
2. MAAYTOVÁ, Alena. *Veřejné finance*. Praha: Vysoká škola ekonomie a managementu, 2012. ISBN 978-80-86730-90-5.
3. MAAYTOVÁ, Alena; OCHRANA, František; PAVEL, Jan. *Veřejné finance v teorii a praxi*. Praha: Grada Publishing, 2015. ISBN 978-80-247-5561-8.
4. JUREČKA, Václav. *Makroekonomie*. 3., aktualizované a rozšířené vydání. Praha: Grada Publishing, 2017. ISBN 978-80-271-0251-8.
5. OCHRANA, František; PAVEL, Jan; VÍTEK, Leoš. *Veřejný sektor a veřejné finance: financování nepodnikatelských a podnikatelských aktivit*. Praha: Grada, 2010. ISBN 978-80-247-3228-2.
6. *Zákon č. 218 / 2000 ze dne 27. června 2000 o rozpočtových pravidlech a o změně některých souvisejících zákonů (rozpočtová pravidla)*. 2000. ISSN 1211-1244. Dostupné také z: `http://aplikace.mvcr.cz/sbirka-zakonu/ViewFile.aspx?type=c%5C&id=3453`.
7. ŽÁKOVÁ, Pavlína et al. *Státní rozpočet 2014 v kostce: Informační příručka Ministerstva financí ČR* [online]. Praha: Ministerstvo financí, 2014 [cit. 2022-04-16]. ISBN 978-80-85045-60-4. Dostupné z: `https://www.mfcr.cz/assets/cs/media/Informacni-letak_2014_Statni-rozpocet-v-kostce_II.pdf`.
8. TEJMLOVÁ, Naděžda. *Vývoj fiskální politiky u nás po roce 1989*. Praha, 2014. Diplomová práce. AMBIS vysoká škola, a.s.
9. VAŇKOVÁ, Eliška. *Fiskální politika a hospodářský cyklus v České republice*. Praha: ČVUT, 2022. Diplomová práce. České vysoké učení technické v Praze, Masarykův ústav vyšších studií.
10. *Federal Budget Challenge* [online]. 2020 [cit. 2022-04-16]. Dostupné z: `https://www.federalbudgetchallenge.org/`.
11. *Federal Balancing Act 2022: An Interactive Budget Simulation* [online] [cit. 2022-04-17]. Dostupné z: `https://us.abalancingact.com/2022-federal-budget`.
12. *Balancing Act* [online] [cit. 2022-04-16]. Dostupné z: `https://abalancingact.com/`.
13. *Národní rozpočtová rada: Kalkulačka veřejných příjmů a výdajů státního rozpočtu* [online] [cit. 2022-04-17]. Dostupné z: `https://unrr.cz/`.
14. Q-SUCCESS. *Web Technology Surveys: Usage statistics of PHP for websites* [online]. 2022 [cit. 2022-04-24]. Dostupné z: `https://w3techs.com/technologies/details/pl-php`.

15. THE PHP GROUP. *PHP: What is PHP?* [Online]. 2022 [cit. 2022-04-24]. Dostupné z: https://www.php.net/manual/en/intro-whatis.php.
16. *Raygun: 10 Popular PHP frameworks for web developers to consider in 2021* [online]. 2018 [cit. 2022-04-24]. Dostupné z: https://raygun.com/blog/top-php-frameworks/.
17. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://laravel.com/docs/9.x.
18. SYMFONY SAS. *Symfony, High Performance PHP Framework for Web Development: Symfony explained to a Developer* [online] [cit. 2022-05-04]. Dostupné z: https://symfony.com/explained-to-a-developer.
19. CODEIGNITER FOUNDATION. *CodeIgniter 4.1.9 documentation: Welcome to CodeIgniter4* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://codeigniter.com/user_guide/intro/index.html.
20. MOZILLA FOUNDATION. *MDN: CSS: Cascading Style Sheets* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS.
21. MOZILLA FOUNDATION. *MDN: HTML: HyperText Markup Language* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.
22. MOZILLA FOUNDATION. *MDN: XML: Extensible Markup Language* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/XML.
23. *Bootstrap: The most popular HTML, CSS, and JS library in the world.* [Online] [cit. 2022-05-04]. Dostupné z: https://getbootstrap.com/.
24. GAIKWAD, Suraj Shahu; ADKAR, Pratibha. A Review Paper On Bootstrap Framework. *Iconic Research And Engineering Journals*. 2019, roč. 2, č. 10, s. 349–351. ISSN 2456-8880. Dostupné také z: https://irejournals.com/formatedpaper/1701173.pdf.
25. MOZILLA FOUNDATION. *MDN: JavaScript* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
26. MOZILLA FOUNDATION. *MDN: What is JavaScript?* [Online]. 2022 [cit. 2022-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.
27. GOOGLE LLC. *Angular: What is Angular?* [Online]. 2022 [cit. 2022-05-04]. Dostupné z: https://angular.io/guide/what-is-angular.
28. META PLATFORMS, INC. *React: A JavaScript library for building user interfaces* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://reactjs.org/.
29. BUSH, Josephine. *Learn SQL Database Programming: Query and Manipulate Databases from Popular Relational Database Servers Using SQL*. Birmingham: Packt Publishing, Limited, 2020. ISBN 978-1-83898-476-2.
30. MARIADB FOUNDATION. *MariaDB.org* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://mariadb.org/.
31. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL: The World's Most Advanced Open Source Relational Database* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://www.postgresql.org/.
32. ABUTALEB, Hamza; TAMIMI, Abdelfatah; ALRAWASHDEH, Thamer. Empirical Study of Most Popular PHP Framework. *2021 International Conference on Information Technology (ICIT)*. 2021, s. 608–611. ISBN 978-1-6654-2870-5. Dostupné z DOI: 10.1109/ICIT52682.2021.9491679.
33. LAAZIRI, Majida; BENMOUSSA, Khaoula; KHOULJI, Samira; KERKEB, Mohamed Larbi. A Comparative study of PHP frameworks performance. *Procedia Manufacturing*. 2019, roč. 32, s. 864–871. ISSN 2351-9789. Dostupné z DOI: 10.1016/j.promfg.2019.02.295.

34. PUSHER LTD. *How Laravel implements MVC and how to use it effectively* [online]. 2018 [cit. 2022-05-04]. Dostupné z: https://blog.pusher.com/laravel-mvc-use/.
35. OLUWATOSIN, Haroon Shakirat. Client-Server Model. *IOSR Journal of Computer Engineering*. 2014, roč. 16, č. 1, s. 67–71. ISSN 2278-8727. Dostupné z DOI: 10.9790/0661-16195771.
36. MOZILLA FOUNDATION. *MDN: An overview of HTTP* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.
37. *Axios* [online] [cit. 2022-05-06]. Dostupné z: https://axios-http.com/docs/intro.
38. RED HAT, INC. *Red Hat: What is a REST API?* [Online]. 2020 [cit. 2022-05-05]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.
39. FOUNDATION, Mozilla. *MDN: JSON* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/JSON.
40. JGRAPH. *Diagram Software and Flowchart maker* [online]. 2021 [cit. 2022-05-03]. Dostupné z: https://www.diagrams.net/.
41. ČESKÝ STATISTICKÝ ÚŘAD. *Český statistický úřad: ZKRATKY KRAJŮ A OKRESŮ REGIONS AND DISTRICTS - ABB* [online]. 2014 [cit. 2022-05-07]. Dostupné z: https://www.czso.cz/csu/czso/13-2199-04-2004-regions_and_districts___abbreviations.
42. ČESKÝ STATISTICKÝ ÚŘAD. *Volby do Poslanecké sněmovny Parlamentu České republiky konané ve dnech 8.10. – 9.10.2021* [online]. 2021 [cit. 2022-05-07]. Dostupné z: https://www.volby.cz/pls/ps2021/ps2?xjazyk=CZ.
43. *Zákon č. 57 / 2022 ze dne 10. března 2022 o státním rozpočtu České republiky na rok 2022*. 2022. ISSN 1211-1244. Dostupné také z: https://www.mfcr.cz/assets/cs/media/Zak_2022-057_Zakon-o-statnim-rozpoctu-Ceske-republiky-na-rok-2022.pdf.
44. WEISS, Tomáš. *Ministerstvo financí České Republiky: Sněmovna schválila nový rozpočet s deficitem 280 mld. Kč, o 97 mld. Kč menším než plánovala minulá vláda* [online]. 2022 [cit. 2022-05-07]. Dostupné z: https://www.mfcr.cz/cs/aktualne/tiskove-zpravy/2022/snemovna-schvalila-novy-rozpocet-s-defic-46455/.
45. MINISTERSTVO FINANCÍ ČESKÉ REPUBLIKY. *Ministerstvo financí České Republiky: Vládní návrh zákona o státním rozpočtu České republiky na rok 2022 a střednědobého výhledu na roky 2023 a 2024* [online]. 2021 [cit. 2022-05-07]. Dostupné z: https://www.mfcr.cz/cs/legislativa/legislativni-dokumenty/2021/vladni-navrh-zakona-o-statnim-rozpoctu-c-43195.
46. CAFOUREK, Tomáš. *Školáci a senioři si připlatí za dopravu, od dubna budou mít slevu 50 procent* [online]. 2022 [cit. 2022-05-11]. Dostupné z: https://www.idnes.cz/ekonomika/doprava/vlaky-autobusy-slevy-seniori-studenti.A220119_110718_eko-doprava_cfr.
47. VLÁDA ČR. *Vláda schválila návrh státního rozpočtu na letošní rok a odsouhlasila rozvolnění protiepidemických opatření* [online]. 2022 [cit. 2022-05-11]. Dostupné z: https://www.vlada.cz/cz/media-centrum/aktualne/vlada-schvalila-navrh-statniho-rozpoctu-na-letosni-rok-a-odsouhlasila-rozvolneni-protiepidemickych-opatreni-194275/.
48. KORDÍK, Jiří. *Vláda plánuje miliardy navíc pro armádní rozpočet. Dvě procenta HDP na obranu chce uzákonit* [online]. 2022 [cit. 2022-05-11]. Dostupné z: https://ct24.ceskatelevize.cz/domaci/3467736-vlada-planuje-miliardy-navic-pro-armadni-rozpocet-dve-procenta-hdp-na-obranu-chce.
49. BUREŠ, Michal. *Další valorizace důchodů bude v září 2022, o kolik vzrostou důchody?* [Online]. 2022 [cit. 2022-05-11]. Dostupné z: https://www.finance.cz/542054-valorizace-duchodu-zari-2022/?_fid=w38x%5C#survey-place.

50. IBM CLOUD EDUCATION. *Docker* [online]. 2021 [cit. 2022-05-08]. Dostupné z: https://www.ibm.com/cloud/learn/docker.
51. DOCKER INC. *Docker: Use containers to Build, Share and Run your applications* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://www.docker.com/resources/what-container/.
52. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Installation* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/installation.
53. OLUSUSI, Oluyemi. *How to Get Started With Docker and Laravel* [online]. 2021 [cit. 2022-05-08]. Dostupné z: https://www.twilio.com/blog/get-started-docker-laravel.
54. SOMASUNDARAM, Ravishankar. *Git: version control for everyone: Beginner's guide : the non-coder's guide to everyday version control for increased efficiency and productivity.* Birmingham: Packt, 2013. ISBN 978-1-84951-752-2.
55. OPENDATALABCZ. *GitHub: budget_quiz* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://github.com/opendatalabcz/budget_quiz.
56. *Composer: Introduction* [online]. 2022 [cit. 2022-05-09]. Dostupné z: https://getcomposer.org/doc/00-intro.md.
57. *Composer: Basic usage* [online]. 2021 [cit. 2022-05-09]. Dostupné z: https://getcomposer.org/doc/01-basic-usage.md.
58. MAGDI, Mustafa. *PHP Composer … The Autoloader* [online]. 2018 [cit. 2022-05-09]. Dostupné z: https://medium.com/tech-tajawal/php-composer-the-autoloader-d676a2f103aa.
59. SINGH, Sunny. *What Is NPM?: A Simple English Guide to Truly Understanding the Node Package Manager* [online]. 2020 [cit. 2022-05-09]. Dostupné z: https://medium.com/swlh/what-is-npm-a-simple-english-guide-to-truly-understanding-the-node-package-manager-41e82f6c5515.
60. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Request Lifecycle* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/lifecycle.
61. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Directory Structure* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/structure.
62. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Database: Getting Started* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/database.
63. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Eloquent: Getting Started* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/eloquent.
64. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Eloquent: Relationships* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/eloquent-relationships.
65. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Database: Migrations* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/migrations.
66. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Controllers* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/controllers.
67. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Routing* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/routing.
68. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: HTTP Session* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://laravel.com/docs/9.x/session.

69. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: CSRF Protection* [online]. 2022 [cit. 2022-05-08]. Dostupné z: `https://laravel.com/docs/9.x/csrf`.

70. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Views* [online]. 2022 [cit. 2022-05-08]. Dostupné z: `https://laravel.com/docs/9.x/views`.

71. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Blade Templates* [online]. 2022 [cit. 2022-05-08]. Dostupné z: `https://laravel.com/docs/9.x/blade`.

72. *Sass* [online]. 2022 [cit. 2022-05-10]. Dostupné z: `https://sass-lang.com/`.

73. *Webpack: Concepts* [online]. 2022 [cit. 2022-05-10]. Dostupné z: `https://webpack.js.org/concepts/`.

74. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Compiling Assets (Mix)* [online]. 2022 [cit. 2022-05-08]. Dostupné z: `https://laravel.com/docs/9.x/mix`.

75. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Authentication* [online]. 2022 [cit. 2022-05-08]. Dostupné z: `https://laravel.com/docs/9.x/authentication`.

76. META PLATFORMS, INC. *React: Introducing JSX* [online]. 2022 [cit. 2022-05-04]. Dostupné z: `https://reactjs.org/docs/introducing-jsx.html`.

77. META PLATFORMS, INC. *React: Rendering Elements* [online]. 2022 [cit. 2022-05-04]. Dostupné z: `https://reactjs.org/docs/rendering-elements.html`.

78. META PLATFORMS, INC. *React: Components and Props* [online]. 2022 [cit. 2022-05-04]. Dostupné z: `https://reactjs.org/docs/components-and-props.html`.

79. META PLATFORMS, INC. *React: State and Lifecycle* [online]. 2022 [cit. 2022-05-04]. Dostupné z: `https://reactjs.org/docs/state-and-lifecycle.html`.

80. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Testing: Getting Started* [online]. 2022 [cit. 2022-05-08]. Dostupné z: `https://laravel.com/docs/9.x/testing`.

81. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: HTTP Tests* [online]. 2022 [cit. 2022-05-08]. Dostupné z: `https://laravel.com/docs/9.x/http-tests`.

82. LARAVEL LLC. *Laravel – The PHP Framework For Web Artisans: Database Testing* [online]. 2022 [cit. 2022-05-08]. Dostupné z: `https://laravel.com/docs/9.x/database-testing`.

# Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `app` ..... zdrojové kódy aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF