Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh a Implementace webové aplikace pro rozhodování debat

Jan Velebný

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jan Velebný

Informatika

Název práce

Návrh a Implementace webové aplikace pro rozhodování debat

Název anglicky

Design and implementation of web application for debate judging

---

Cíle práce

Cílem práce je analýza, návrh a implementace webové aplikace s architekturou MVC, sloužící k usnadnění rozhodování soutěžních debat formátu Karl Popper. Aplikace bude disponovat přehledným uživatelským rozhraním spolu se systémem pro pomoc při hodnocení jednotlivých řečníků v debatě. Aplikace bude také umět zobrazovat rady z pravidel a tisk listu rozhodčího (ballotu) v PDF.

Metodika

Na základě studia odborných zdrojů bude formou literární rešerše popsána doména vývoje webových aplikací, dále bude provedena a popsána analýza uživatelských požadavků a v souladu s doporučenými postupy softwarového inženýrství bude vytvořen návrh webové aplikace, který bude poté implementován a otestován.

Implementace bude provedena pomocí PHP frameworku Lumen pro backendovou část a JavaScriptového frameworku Quasar pro frontendovou část.

Poznatky z vývoje a testování budou popsány a zhodnoceny.

Doporučený rozsah práce

60-80 stránek

Klíčová slova

Webová Aplikace, Debata, HTML, PHP, JavaScript, MySQL

Doporučené zdroje informací

BUTLER, Tom. PHP & MySQL: Novice to Ninja. 7th edition. Richmond(Australia): SitePoint Pty., 2022. ISBN ISBN 978-1-925836-47-9.

HAVERBEKE, Marijn. Eloquent JavaScript: A Modern Introduction to Programming. 3rd edition. San Francisco: No Starch Press, 2019. ISBN 978-1-59327-950-9.

MARTIN, Robert C. Clean architecture: a craftsman's guide to software structure and design. 3rd edition. London, England: Prentice Hall, 2018. Robert C. Martin series. ISBN 978-0-13-449416-6.

Vue.js: up and running : building accessible and performant web apps. Sebastopol: O'Reilly, 2018. Robert C. Martin series. ISBN 978-1-491-99724-6.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Petr Hanzlík, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 27. 2. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 2. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 30. 03. 2024

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh a Implementace webové aplikace pro rozhodování debat" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2024 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) vedoucímu této práce Ing. Petru Hanzlíkovi Ph.D. za jeho ochotu a trpělivost. Také bych chtěl poděkovat Kryšpínu Varyšovi za nemálo nápomocných rad a vlídných slov a Matěji Němcovi za mentální podporu.

Návrh a Implementace webové aplikace pro rozhodování debat

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací webové aplikace pro zjednodušení rozhodování akademických debat formátu Karl Popper. Jedná se o aplikaci, který má jako primární účel zefektivnění tvorby listu rozhodčího, tzv. ballotu při debatě. Teoretická část práce se zabývá problematikou vývoje webových aplikací z hlediska postupů softwarového inženýrství. Vysvětluje metodiky vývoje software, jazyk UML a potřebné diagramy stejně jako praktiky objektově orientovaného programování. Dále jsou v teoretické části shrnuty všechny ve vývoji užité technologie a paradigmata spolu s tématem akademické debaty. Tyto jsou poté dopodrobna vysvětleny. Praktická část práce se pak zabývá samotnou implementací aplikace, pokrývá fázi analýzy a vytyčení funkčních požadavků, tvorbu diagramů, implementaci backendu aplikace jako REST API ve frameworku Lumen v jazyce PHP za využití SQL databáze a poté návrh, prototypování a implementaci frontendu aplikace za využití frameworku Quasar v jazyce JavaScript. Výsledkem je funkční webová aplikace, která byla podrobena testování a objevené nedostatky byly opraveny. Aplikace splňuje všechny požadavky a cíle vytyčené v této práci.

Klíčová slova: Webová Aplikace, Debata, HTML, PHP, JavaScript, MySQL

Design and implementation of web application for debate judging

Abstract

This diploma thesis deals with the design and implementation of a web application for simplification of the decision making of academic debates of the Karl Popper format. It is an application that has the streamlining of the creation of the judge's sheet, the so-called ballot in debate as its primary purpose. The theoretical part of the thesis deals with the issue of web application development in terms of software engineering practices. It explains software development methodologies, the UML language and the diagrams needed as well as object-oriented programming practices. Furthermore, the theoretical part summarizes all the technologies and paradigms used in development along with the topic of academic debate. These are then explained in detail. The practical part of the thesis then deals with the actual implementation of the application, covering the phase of analysis and delineation of functional requirements, the creation of diagrams, the implementation of the backend of the application as a REST API in the Lumen framework in PHP using a SQL database and then the design, prototyping and implementation of the frontend of the application using the Quasar framework in JavaScript. The result is a working web application that has been tested and the discovered deficiencies have been fixed. The application meets all the requirements and objectives set out in this diploma thesis.

Keywords: Web Application, Debate, HTML, PHP, JavaScript, MySQL

Obsah

1 Úvod

V dnešní digitální době je debatní kultura a soutěžení na středoškolské i univerzitní úrovni stále populárnější. Debata není jenom prostředkem k výměně názorů, ale také efektivním prostředkem k rozvoji kritického myšlení, schopnosti argumentace a veřejného projevu. V jako i jiných odvětvích se i při organizaci debatních soutěží a událostí setkáváme se situacemi které nám komplikují život. Tyto situace přitom často vznikají, protože se stále používá tužka a papír, když by je mohla zastoupit digitální alternativa. A zatímco pokud jde o organizaci turnajů samotných, ať už se bavíme o přihlášky a ubytování nebo správu výsledků debat, tak existuje povedený digitální systém, který tyto úlohy usnadňuje. Ne všechny disciplíny debatní sféry mají však takové štěstí. Hovořím o samotném rozhodování debat, kdy rozhodčí musí hlídat časy jednotlivých řečníků a musí sepsat detailní list rozhodčího neboli ballot. Existuje sice digitální verze ballotu jako dokument, ale ten práci rozhodčího neusnadňuje dostatečně a včasné vyplňování ballotů je na debatních turnajích zpravidla boj. Aplikace, která je předmětem této diplomové práce má tento nedostatek eliminovat.

Tato diplomová práce se zabývá vývojem webové aplikace, která má za cíl usnadnit proces hodnocení a sledování časů řečníků rozhodčím v debatních soutěžích. Aplikace vychází z potřeby modernizovat a zefektivnit tradiční postupy vyplňování ballotů a správy časů a informací v debatách. Snaží se toho docílit využitím moderních webových technologií a spolehlivosti spolu s intuitivním a uživatelsky přívětivým rozhraním.

V následujících kapitolách práce budou podrobně popsány detaily problematiky a teoretická východiska stejně jako procesy vývoje aplikace, použité technologie, funkcionalita aplikace, testování a výsledky dosažené v průběhu vývoje. Nakonec budou diskutovány přínosy a budoucí směřování této aplikace v rámci debatní komunity.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je analýza, návrh a implementace webové aplikace s architekturou MVC, sloužící k usnadnění rozhodování soutěžních debat formátu Karl Popper. Aplikace bude disponovat přehledným uživatelským rozhraním spolu se systémem pro pomoc při hodnocení jednotlivých řečníků v debatě. Aplikace bude také umět zobrazovat rady z pravidel a tisk listu rozhodčího (ballotu) v PDF.

2.2 Metodika

Na základě studia odborných zdrojů bude formou literární rešerše popsána doména vývoje webových aplikací, dále bude provedena a popsána analýza uživatelských požadavků a v souladu s doporučenými postupy softwarového inženýrství bude vytvořen návrh webové aplikace, který bude poté implementován a otestován. Implementace bude provedena pomocí PHP frameworku Lumen pro backendovou část a JavaScriptového frameworku Quasar pro frontendovou část.
Poznatky z vývoje a testování budou popsány a zhodnoceny.

3 Teoretická východiska

Tato kapitola obsahuje všechny potřebné teoretické podklady k pochopení praktické části práce.

3.1 Softwarové Inženýrství

Softwarové inženýrství je inženýrská disciplína, která se zabývá praktickými problémy vývoje software. Celý tento proces zahrnuje velké množství faktorů sahajících od hardwarového vybavení, přes aspekty jako organizační strukturu a ekonomické možnosti nebo znalosti z oblasti požadavků na software, jeho analýzu, návrh, implementaci a testování, až po řízení lidských zdrojů a schopnosti efektivního využití všech zmíněných zdrojů a procesů, aby se dosáhlo vytvoření produktu požadované kvality.
(Procházka, Klimeš, 2009)

V softwarovém inženýrství je cílem dosáhnout vytvoření kvalitního softwarového produktu s maximální efektivitou. Softwaroví inženýři během své práce postupují systematicky a organizovaně, využívají přitom metody a nástroje, které podporují nebo automatizují jejich činnosti, jako jsou vývojová prostředí, CASE nástroje a nástroje pro testování.

Softwarové inženýrství se zaměřuje na praktické problémy a využívá výsledky z jiných vědních oblastí. V tomto kontextu hraje informatika klíčovou roli, která se mimo jiné zabývá teoretickými aspekty využívání počítačů a softwarových systémů. Zatímco pro informatiku jsou např. programovací jazyky předmětem zájmu, pro softwarové inženýrství jsou spíše nástrojem pro návrh a implementaci řešení. Bez širokého spektra poznatků z informatiky by softwaroví inženýři nemohli úspěšně vykonávat svou profesi.

(Sklenář, 2007)

3.1.1 Metodiky vývoje software

Metodika vývoje software je soubor činností, které se provádějí, když má být vytvořen nějaký softwarový produkt. V kontextu softwarového inženýrství není tento proces pevně daným předpisem pro to, jak postavit počítačový software. Spíše se jedná o přizpůsobitelný návod nebo postup, který umožňuje lidem provádějícím práci (týmu softwarových inženýrů) vybrat si vhodnou sadu pracovních akcí a úkolů. Cílem je vždy dodat software včas a s dostatečnou kvalitou, aby uspokojil ty, kteří jej financovali, a ty, kteří jej budou používat. Každá metodika zahrnuje malý počet rámcových činností, které jsou použitelné pro všechny softwarové projekty bez ohledu na jejich velikost nebo složitost.
Jedná se o:

- **Komunikaci (Specifikace požadavků)** – Před zahájením samotného vývoje softwaru je nesmírně důležitá komunikace a spolupráce se zákazníkem (a dalšími zúčastněnými stranami). Záměrem je porozumět cílům zainteresovaných stran v rámci projektu a shromáždit funkční požadavky, které pomohou definovat vlastnosti a funkce softwaru.
- **Plánování (Analýza)** – Dalším krokem je využití požadavků od zákazníka k vytvoření plánu softwarového projektu, který popisuje technické úkony, které je nutno provést, pravděpodobná rizika, potřebné zdroje, výsledné produkty a pracovní harmonogram.
- **Modelování (Návrh)** – V této etapě se využívá mnoha modelovacích nástrojů (např. UML) aby se dosáhlo lepšího pochopení požadavkům softwaru, který se bude vytvářet a návrhu, který má tyto požadavky splnit.
- **Konstrukci (Implementace)** – Když je vše naplánováno a rozvrženo, je načase software vytvořit. To zahrnuje ruční nebo automatizované generování kódu a testování, které je nutné k odhalení chyb.
- **Předání (Nasazení)** – Když je software kompletní (nebo etapa jeho vývoje), je dodán zákazníkovi a ten zhodnotí, zda je v pořádku a poskytne zpětnou vazbu.

V závislosti na zvolených metodikách vývoje se tyto rámcové činnosti aplikují po celou dobu vývoje software nebo iterativně. To znamená, že jsou aplikovány opakovaně, v řadě iterací projektu. (Pressman, Maxim, 2015)

V následujících podkapitolách budou uvedeny příklady některých metodik vývoje software.

3.1.1.1 Vodopád

Jedná se o jednu z historicky nejstarších metodik vývoje software, která vznikla v 70. letech 20. století. Tato metodika zahrnuje systematický a sekvenční přístup k vývoji software, který se skládá z několika etap s tím, že jedna etapa nemůže započít, dokud neskončí ta předchozí. Celý proces začíná u zákazníka specifikací požadavků a postupuje přes plánování, modelování, konstrukci a nasazení, přičemž vrcholem je průběžná podpora dokončeného softwaru. Tato metodika se ukázala jako značně problematická, protože v praxi se málokdy nějaká etapa vývoje doopravdy dovede do úplného konce. Častokrát se například mohou změnit uživatelské požadavky a celý software se pak musí pozměnit. (Pressman, Maxim, 2015)

3.1.1.2 Inkrementální metodiky

Inkrementální metodiky kombinují prvky lineárního a paralelního procesního toku. Používají lineární sekvence postupně v průběhu času. Každá lineární sekvence produkuje "inkrementy" softwaru, které se dodávají zákazníkovi. Tyto metodiky vývoje softwaru jsou praktické v situacích, kdy jsou počáteční požadavky dobře definovány, ale celkový rozsah vývojového úsilí vylučuje čistě lineární proces. Také jsou užitečné v situacích, kdy je naléhavá potřeba poskytnout zákazníkovi omezený soubor softwarových funkcí rychle a poté tyto funkce upřesnit a rozšířit v pozdějších verzích softwaru.

V praxi se často používá inkrementální přístup, kde inkrement je produktem, obsahující pouze základní funkcionality dle požadavků, ale mnoho doplňkových funkcí zde zůstává nedodáno. Tento produkt je pak používán a testován zákazníkem, a na základě jeho zpětné vazby je vypracován plán na další inkrement. Tento plán zahrnuje modifikace základního produktu a dodání dalších funkcí a možností, tak aby lépe vyhovoval potřebám zákazníka. Tento proces se opakuje po dodání každého inkrementu, dokud není vytvořen kompletní produkt. (Pressman, Maxim, 2015)

3.1.1.3 Iterativní metodiky

Iterativní metodiky vývoje softwaru jsou metodiky, které se zaměřují na opakované cykly vývoje, kde je proces vývoje rozdělen do sérií menších iterací. Každá iterace projde celým vývojovým cyklem, který zahrnuje analýzu, návrh, implementaci a testování. Hlavním principem iterativního modelu je schopnost rychlého získávání zpětné vazby od zákazníka a pružného reagování na změny během vývoje.

V iterativním modelu je plánování a design základním kamenem každé iterace. Během těchto fází jsou identifikovány klíčové požadavky a definovány cíle pro danou iteraci. Poté následuje implementační fáze, kde jsou navržené funkcionality a vlastnosti implementovány a integrovány do existujícího systému. Nakonec je provedeno testování, kde se ověřuje funkčnost a kvalita nově přidaných funkcí.

Jednou z klíčových vlastností iterativního modelu je jeho schopnost adaptace a flexibility. Díky opakovaným cyklům vývoje je možné rychle reagovat na změny požadavků, neboť nové informace a zpětná vazba od zákazníka jsou integrovány do dalších iterací. Tímto způsobem je dosaženo postupného zlepšování a zdokonalování softwarového produktu.

Výhody iterativního modelu zahrnují možnost rychlého dodání funkčních částí produktu, snadnou adaptaci na změny a vylepšení kvality díky opakovanému testování a zpětné vazbě. Nicméně, náležité plánování a řízení je nezbytné pro úspěšnou implementaci iterativního modelu, ať už ve formě Scrumu, extrémního programování nebo jiných agilních metodik. (Pressman, Maxim, 2015)

3.1.1.4 Metodika prototyp

Prototypování jde použít i jako paradigma, využitelné i v jiných procesech vývoje software, nejedná se striktně jen o jasně definovatelnou metodiku vývoje. Metodika to však je také a používá se, aby pomohla ilustrovat, co má být vlastně vytvořeno, pokud jsou požadavky nejasné. Prvním krokem při prototypování je komunikace se zákazníkem a všemi zúčastněnými stranami. Definují se cíle softwaru, identifikují se známé požadavky a nastíní se oblasti, kde je další definice nezbytná. V dalším kroku se vytvoří rychlý prototyp, který se soustředí hlavně na ty části softwaru, se kterými interaguje uživatel

(například rozvržení uživatelského rozhraní nebo formáty výstupního zobrazení). Tento prototyp je poté nasazen a zhodnocen zákazníkem. Zpětná vazba je potom použita k dalšímu zpřesnění požadavků. Ideálně by prototyp měl sloužit jako mechanismus pro identifikaci požadavků na software, který se potom má vytvořit. (Pressman, Maxim, 2015)

3.1.1.5 Spirálová metodika

Spirálová metodika vývoje software kombinuje iterativní povahu prototypování s řízenými a systematickými aspekty vodopádového modelu. Její použití vede k rychlému vývoji stále komplexnějších a kompletnějších verzí softwaru. Princip spočívá v opakovaném cyklickém zvyšování detailu a implementace softwaru, přičemž se současně snižuje riziko a dosahuje se klíčových milníků, jež zajišťují závazek všech zúčastněných stran k proveditelným a vzájemně uspokojivým řešením. Pomocí spirálového modelu se software vyvíjí v řadě verzích. Během počátečních iterací může být software model nebo prototyp a v pozdějších iteracích jsou pak produkovány stále kompletnější verze softwaru. Během každé iterace se zvažují rizika a její konec je značen milníkem. První iterace může vést k vytvoření specifikace produktu; následné iterace mohou být použity k vytvoření prototypu a pak postupně sofistikovanějších verzí softwaru. Každá etapa plánování vede ke změnám v projektovém plánu. Náklady a harmonogram se upravují na základě zpětné vazby získané od zákazníka po dodání. Navíc projektový manažer upravuje plánovaný počet iterací potřebných k dokončení softwaru. Výhodou je, že na rozdíl od jiných metodik, které končí po dodání softwaru, lze spirálový model přizpůsobit tak, aby platil po celou dobu životního cyklu počítačového softwaru. (Pressman, Maxim, 2015)

3.1.1.6 RUK

RUK (Rational Unified Process) je inkrementální a iterativní metodiku vývoje software, která vznikla v devadesátých letech 20. století. Jedná se o komerční verzi metody UP (Unified Process), která vznikla ve firmě Rational (nyní součást IBM), při vzniku jazyka UML. Životní cyklus produktu se zde dělí na čtyři fáze:

- *Zahájení (Inception)* – Kdy se stanovuje rozsah projektu a posuzují se podmínky pro jeho realizaci. Určují se aktéři a případy užití, možná architektura a rizika a ujasní se cena, nástroje, zdroje a časový plán.

- *Rozpracování (Elaboration*) – Kdy se zpracovávají klíčové požadavky a řeší se hlavní problémy s vytvářením softwaru a implementují se základní prvky architektury.
- *Konstrukce (Construction)* – Kdy se detailně navrhnou, implementují a testují všechny zbývající části softwaru. Dojde k Beta nasazení a shromáždí se zpětná vazba.
- *Nasazení (Transition)* – Kdy se hotový software odevzdá zákazníkovi a zajistí se jeho stabilní a korektní fungování v reálném použití.

Každá fáze může mít několik iterací, ale jedna iterace může být obsazena pouze v jedné fázi a výsledkem bude vždy nová verze softwaru. Jedna iterace má také vždy pevně stanovenou délku. (Sklenář, 2007)

3.1.1.7 Agilní metodiky

Agilní metodiky jsou moderní přístupy k vývoji softwaru, které se zaměřují na flexibilitu a schopnost rychle reagovat na změny během vývoje. Tyto metodiky rozpoznávají nejistotu v plánování a dávají přednost adaptivnímu přístupu před pevnými plány.

Jedním z klíčových principů agilních metodik je dodávání funkčního softwaru v krátkých iteracích, známých jako inkrementy. Tímto způsobem se minimalizuje riziko a umožňuje se pružně reagovat na měnící se požadavky. Zákazník je aktivně zapojen do vývojového procesu a poskytuje zpětnou vazbu, což pomáhá vytvářet software, který lépe vyhovuje jeho potřebám.

Agilní metodiky také kladou důraz na spolupráci a komunikaci v týmu. Ruší tradiční rozdělení mezi vývojovým týmem a zákazníkem a místo toho podporují interakci a spolupráci. Tím se odstraňuje bariéra ve formě mentality "my" a "oni", přetrvávající v mnoha softwarových projektech a vytváří se silnější a efektivnější týmová dynamika. (Pressman, Maxim, 2015)

Mezi agilní metodiky vývoje software patří například:

3.1.1.7.1 Extrémní programování (XP)

jedná se o metodiku vývoje softwaru, která si klade za cíl dosáhnout rychlého vývoje softwaru vysoké kvality, který plně odpovídá potřebám zákazníka. XP se vyznačuje pravidelnou a účinnou komunikací se zákazníkem, který je aktivně zapojen do procesu vývoje a spolupracuje s vývojáři na definování požadavků, známých jako "User Stories".

Zdůrazňuje se zde kontinuální cyklus vývoje, kde se neustále a paralelně během celého vývoje provádějí 4 činnosti: plánování, návrh, programování a testování, přičemž důležitým prvkem je také práce v malých inkrementech, které se pravidelně předávají zákazníkovi, aby poskytl zpětnou vazbu. Tento iterativní přístup umožňuje vývojovému týmu reagovat na změny a upravovat vývojové procesy podle potřeby.

V rámci XP je také kladen velký důraz na kvalitu kódu a testování. Testy jsou psány současně s kódem a pravidelně spouštěny k ověření funkčnosti a kvality softwaru. Kód je udržován co nejminimalističtější a provádí se častý refaktoring, aby se zlepšila jeho čitelnost a udržitelnost.

Extrémní programování je také známý svou prací v párech, kde dva programátoři spolupracují na psaní kódu a kontrolují kvalitu vzájemně, čímž se zvyšuje produktivita a zlepšuje kvalita kódu. (Pressman, Maxim, 2015)

3.1.1.7.2 Scrum

Název Scrumu pochází z Rugby a jedná se o situaci, kdy hráči obou týmů tvoří těsný kruh kolem míče na zemi a snaží se jej získat (často násilně).

Jedná se o iterativní a inkrementální metodiku vývoje, což znamená, že zákazník pravidelně obdrží funkční části produktu. Scrum zahrnuje tým lidí pracujících na vývoji produktu, který je rozdělen na role jako je Scrum Master a Product Owner.

Princip této metodiky spočívá v rozdělení vývoje do tzv. sprintů, což jsou krátké časové úseky, jejichž délka je obvykle 2 až 4 týdny, ale může se lišit v závislosti na potřebách projektu. Během sprintů se Scrum Team (samo

organizující se team specialistů a vývojářů) soustředí na dokončení určitého množství práce. Na konci každého sprintu tým prezentuje hotové části produktu. Během toho se vývojový tým schází každý den na krátkém „Stand Up“ meetingu, kde se sdílí, co se udělalo včera, co se bude dělat dnes a jaké překážky je nutno překonat.

Scrum Master je pak zodpovědný za dodržování Scrum procesu a pomáhá týmu odstraňovat překážky ve vývoji a zlepšovat svoji produktivitu a Product Owner je zodpovědný za správu a prioritizaci produktového backlogu, což je seznam všech požadavků na produkt. (Pressman, Maxim, 2015)

3.1.2 UML

Unified Modeling Language (UML) je standardizovaný jazyk používaný v oblasti softwarového inženýrství k vizualizaci, návrhu a dokumentaci softwarových systémů.

Jedná se o rodinu grafických notací, podložených jediným metamodelem, které pomáhají popisovat a navrhovat softwarové systémy, a to zejména ty, které jsou vytvořené pomocí objektově orientovaného programování.

UML jako takové je relativně otevřený standard, který je řízený skupinou OMG (Object Managment Group), která je otevřené konsorcium společností. OMG byla vytvořena k vytváření standardů, které podporují interoperabilitu objektově orientovaných systémů. OMG je možná nejlépe známá svými standardy CORBA (Common Object Request Broker Architecture).

UML vznikl v roce 1997 ve firmě Rational sloučením mnoha objektově orientovaných grafických modelovacích jazyků, které prosperovaly koncem 80. a začátkem 90. let. (Fowler, 2004)

Jak již bylo řečeno, UML je velmi mocný grafický nástroj, určený k vizualizaci návrhu softwaru. V zásadě se dá použít třemi způsoby:

- *Jako náčrt (sketch)* – UML diagramy disponují velmi vysokou mírou abstrakce, a proto jsou vhodné pro tvorbu modelů ve fázi analýzy návrhu software. Usnadňují tvorbu uživatelských požadavků a komunikaci se zákazníky a snižují riziko, že bude software špatně navržen.
- *Jako plán (blueprint)* – Když už se vývoj přesune od fáze analýzy, je možné využít UML k detailnímu plánování implementace pro programátory. Diagramy mohou usnadnit komunikaci mezi vývojáři a zlepšit orientaci ve vyvíjeném softwaru. Navíc po implementaci slouží jako dokumentace softwaru.
- *Jako programovací jazyk* – V některých případech je možné detailní diagramy použít k vygenerování automatické šablony kódu, která pak může sloužit jako základ pro implementaci. (Hartinger, 2024)

3.1.2.1 Diagram tříd

Diagram tříd popisuje typy objektů v systému a různé druhy statických vztahů, které mezi nimi existují. Diagramy tříd také zobrazují vlastnosti a operace třídy a omezení, která se vztahují na způsob propojení objektů. V UML se používá termín feature (vlastnost) jako obecný termín, který zahrnuje vlastnosti a operace třídy. (Fowler, 2004)

Při popisování třídy pomocí diagramu tříd se nejprve uvede její název a poté seznam jejích atributů s datovými typy. Před každým je krom toho ještě uveden modifikátor přístupu, který může mít 4 různé hodnoty:

- - *(mínus)* – Privátní atribut (private)
- \+ *(plus)* – Veřejný atribut (public)
- # *(hash nebo mřížka)* – Chráněný atribut (protected)
- ~ *(tilda)* – Atribut viditelný v rámci balíku (package)

Mezi název atributu a datový typ se pak píše dvojtečka. (Hartinger, 2024)

Dalším způsobem jako určovat vlastnosti třídy jsou asociace a je pomocí nich možné zobrazit stejnou informaci jako výpisem atributů, jen jiným způsobem. Asociace je plná čára mezi dvěma třídami, která směřuje od zdrojové třídy k cílové třídě. Na cílovém konci asociace se nachází název vlastnosti a její mohutnost. Cílový konec asociace odkazuje na třídu, která je typem vlastnosti.

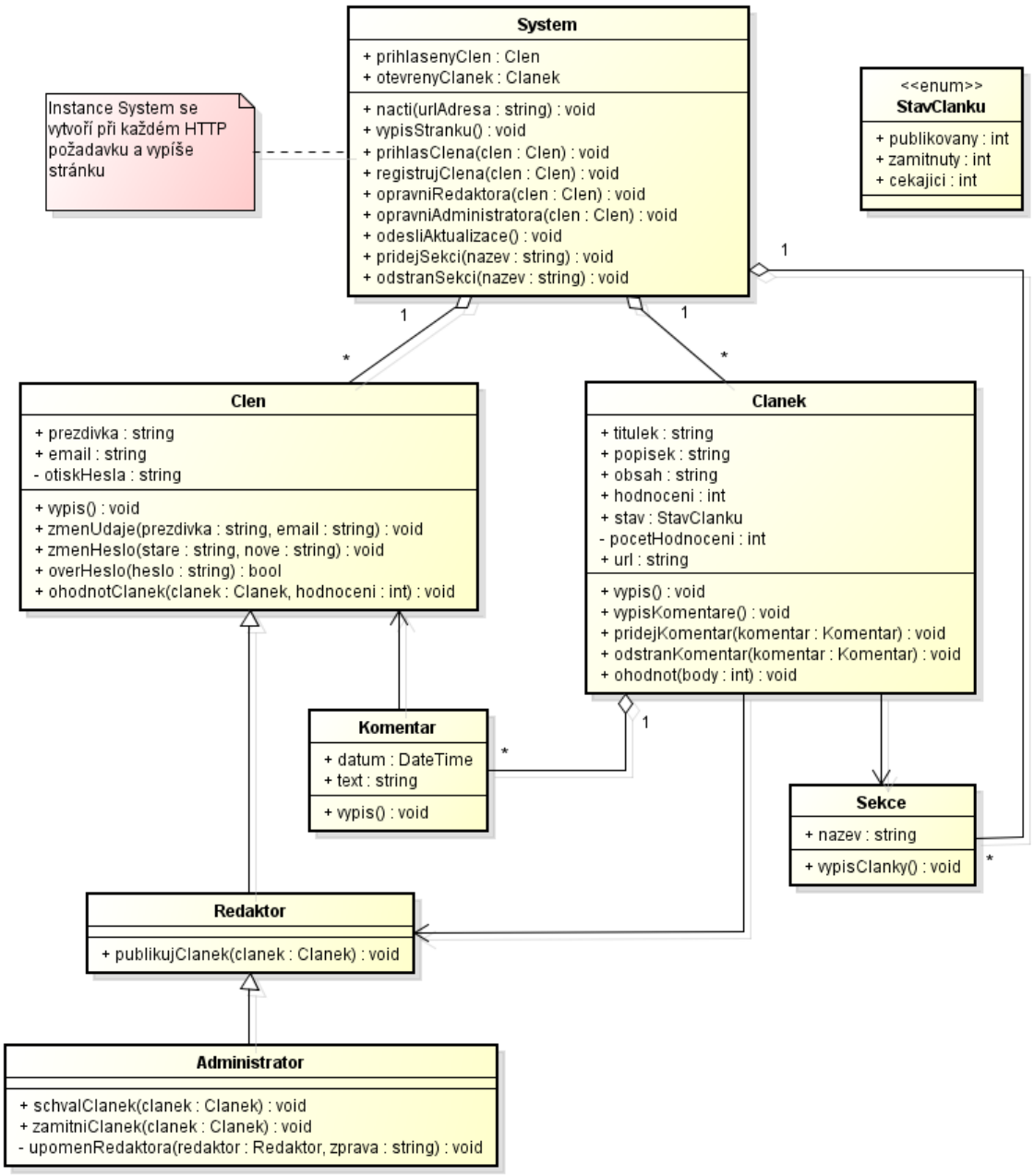
Ačkoli se v obou zápisech objevuje většina stejných informací, některé položky se liší. Asociace se zejména liší tím, že mohou zobrazovat mohutnost na obou koncích řádku.

Nejčastější mohutnosti, které se při tvorbě tříd používají jsou:

- **1** (Instance třídy může být propojená právě s jednu další instanci)
- **0..1** (Instance třídy může být propojená s jednou nebo žádnou další instancí)
- ***** (Instance třídy nemusí mít žádné propojení nebo může být propojená s jakýmkoliv množstvím dalších instancí)

Obecně se mohutnost definuje pomocí spodní a horní hranice. Spodní hranice může být libovolné kladné číslo nebo nula; horní hranice je libovolné kladné číslo nebo * (pro neomezené). Pokud je spodní a horní hranice stejná, lze použít jedno číslo; 1 je tedy ekvivalentní 1..1. * je pak zkratka pro 0..*. (Fowler, 2004)

Jedna z mnoha dalších vlastností diagramu tříd jsou např. realizace. Jedná se o vztah mezi rozhraním a třídou, která ho implementuje. Třída, která reprezentuje rozhrání se nazývá stereotyp a píše se do dvojitých špičatých závorek. Třída, jež implementuje toto rozhraní je na něj napojená vazbou podobnou dědičnosti, pouze s přerušovanou čárou. (Hartinger, 2024)



Obrázek 1 - Příklad UML Diagramu tříd (Hartinger, 2024)

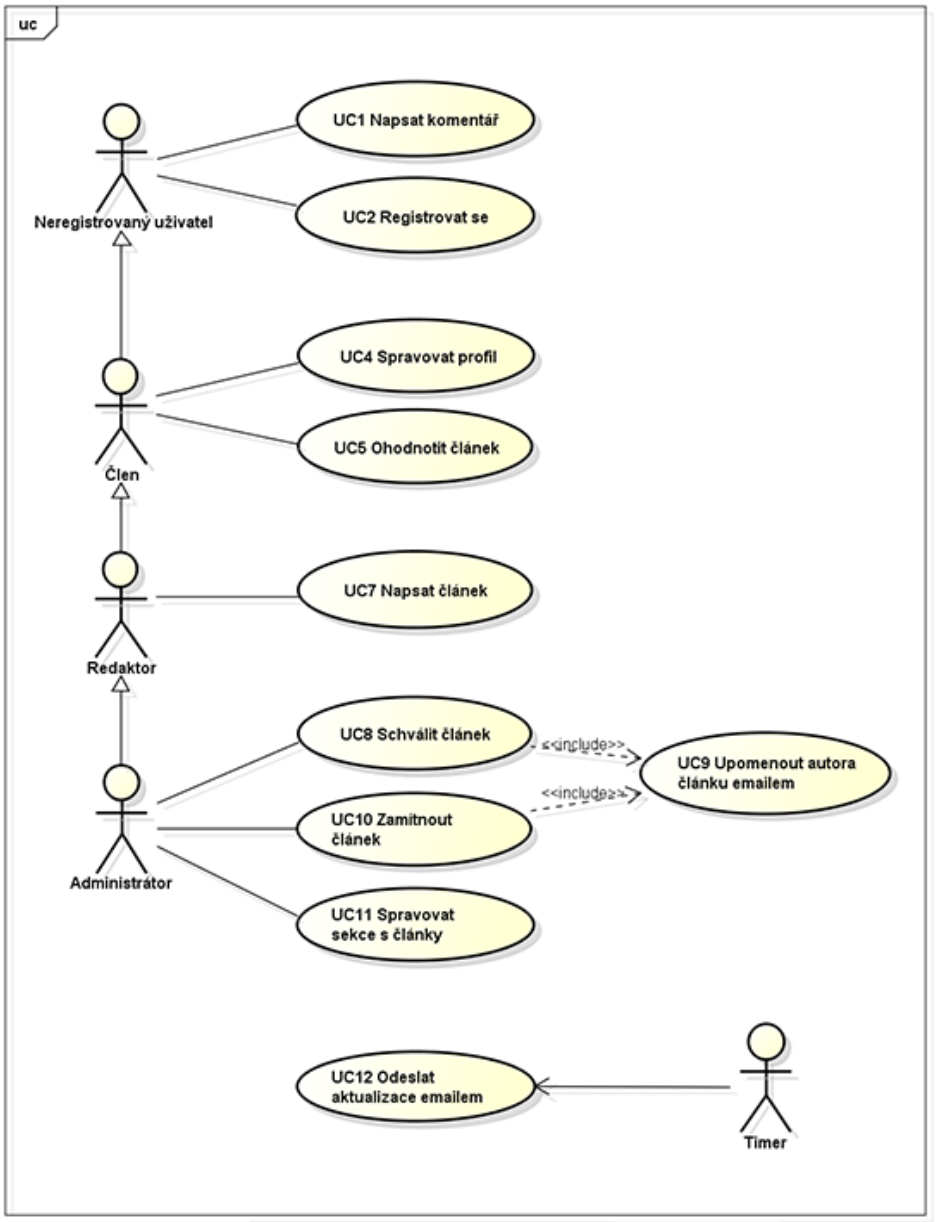
3.1.2.2 Diagram případu užití (Use Case)

Případy užití jsou technikou pro zachycení funkčních požadavků systému. Fungují tak, že popisují typické interakce mezi uživateli systému a samotným systémem, poskytují informace o tom, jak je systém používán.

S diagramem užití se velice úzce pojí i scénář užití, který se dá popsat jako sekvence kroků popisující interakci mezi uživatelem a systémem, které vedou k dosažení nějakého cíle. Případ užití je tedy soubor scénářů spojených společným uživatelským cílem.

Ve vyjádření případů užití jsou uživatelé označováni jako aktéři. Aktér je role, kterou uživatel hraje ve vztahu k systému a která vykonává případy užití. Jednotlivý aktér může provádět mnoho případů užití; naopak, jeden případ užití může mít několik aktérů, kteří jej vykonávají. Aktér také nemusí být nutně člověk. Pokud například systém poskytuje službu pro jiný počítačový systém, tento druhý systém je taktéž aktérem.

Diagram případů užití pak zobrazuje jednotlivé aktéry jako postavy z čar s názvem, napsaným pod ní a jednotlivé případy užití jako elipsy s názvem uvnitř. Samotný diagram pak znázorňuje aktéry, případy užití a vztahy mezi nimi, tedy to, jací aktéři vykonávají, které případy užití a které případy užití obsahují jiné případy užití. (Fowler, 2004)



Obrázek 2 - Příklad UML diagramu případu užití (Hartinger, 2024)

3.1.3 Další diagramy

Ne všechny diagramy, které jsou užitečné pro navrhování a dokumentaci softwaru jsou součástí jazyka UML. V této krátké podkapitole budou proto zmíněny některé další důležité diagramy.

3.1.3.1 Databázový ER diagram.

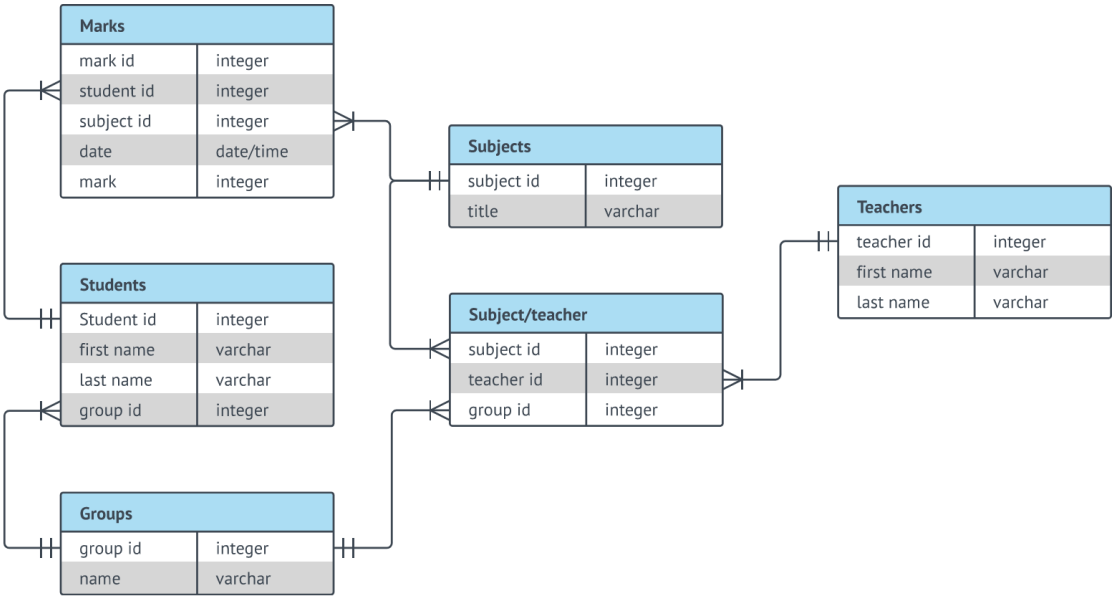
ER diagram neboli Entitně relační diagram se používá k znázornění struktury databází, které jsou velmi často součástí webových aplikací. Tento diagram graficky znázorňuje entity v databázi a vazby mezi nimi, včetně jejich kardinality a parciality. V kontextu databází je Entita objektem našeho pozorování, je to jakákoliv informace, kterou potřebujeme evidovat a v databázi se pak většinou jedná o jednu tabulku. Tyto entity pak mají mezi sebou vazby definované pomocí primárních a cizích klíčů. Kardinalita pak znázorňuje násobnost této vazby a může mít 3 podoby:

- *1:1* – znázorňuje, že jeden záznam může být spojen pouze s právě jedním jiným záznamem a graficky je znázorněna pomocí rovné čáry.
- *1:N* – znázorňuje, že jeden záznam může být spojen s libovolným počtem jiných záznamů a graficky je znázorněna také pomocí rovné čáry, může jich však vést z jednoho záznamu více.
- *M:N* – znázorňuje, že libovolný počet záznamů může být spojen s libovolným počtem jiných záznamů a graficky je znázorněna čárou, ze které na konci vystupují další dvě čáry ve tvaru obrácené šipky.

Parcialita poté pouze určuje povinnost tohoto spojení, tedy určuje minimální počet vztahů, který tabulka může mít a to buď 1 nebo 0. Znázorňuje se buď přeškrtnutou vztahovou čárou pro 1 nebo elipsou zapsanou kolmo přes vztahovou čáru pro 0.
(Skřivan, 2008)

Principy fungování databází budou podrobněji vysvětleny v kapitole, určené pro databáze.

Jak je možné si všimnout, ER diagram je svým principem velmi podobný diagramu tříd. Hlavní rozdíl je ale v tom, že ER diagram zachycuje strukturu relačních tabulek v databázi a ty nemusí vždy mít odpovídající třídu v objektovém modelu aplikace. Je proto nutné zobrazovat strukturu databáze a její objektovou reprezentaci v aplikaci odděleně.



Obrázek 3- Příklad ER diagramu (Lucid Software Inc., 2024)

3.1.3.2 Workflow diagram

Workflow diagram neboli Diagram pracovního postupu poskytuje vizuální přehled obchodního procesu. Pomocí standardizovaných symbolů a tvarů ukazuje, jak pracovní postup probíhá od začátku do konce a kdo je v každém kroku za daný pracovní postup zodpovědný. Při návrhu pracovního postupu je nejdříve nutné provést důkladnou analýzu, která odhalí potenciální slabiny a umožní definovat, standardizovat a identifikovat klíčové aspekty tohoto procesu.

Pracovní postupy také pomáhají zaměstnancům lépe porozumět svým rolím a pořadí, v jakém mají práci vykonávat, a přispívají k větší soudržnosti mezi různými odděleními. Původně byly pracovní postupy používané v průmyslovém sektoru, dnes se však tyto diagramy využívají v různých odvětvích včetně programování a softwarového inženýrství.

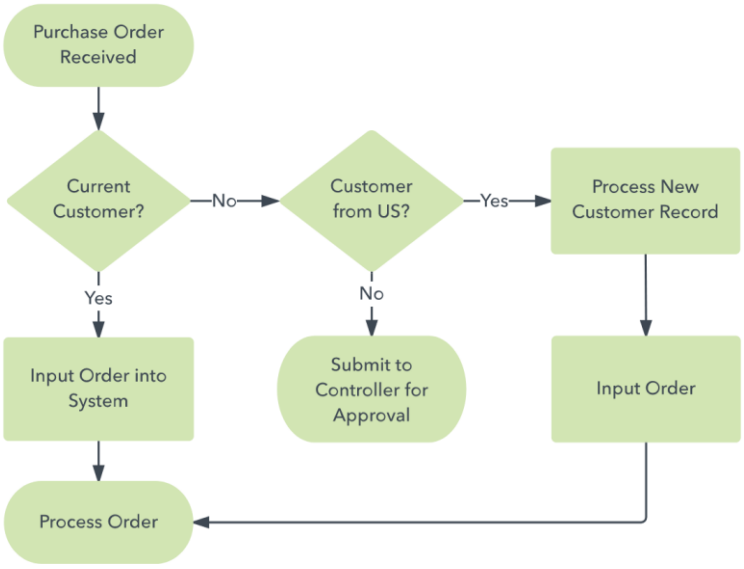
Pokud jde o vizuální reprezentaci, lze o workflow diagramu říci, že každý jeho prvek je navržen tak, aby ilustroval tok mezi jednotlivými kroky. Každý krok pak zahrnuje jeden ze tří parametrů:

- *Vstup* – práce, kapitál, časová dotace, zařízení nebo informace, které jsou potřebné k dokončení kroku.

- *Transformace* – Změny, které vytvářejí výstup, jako je změna polohy, fyzikální charakteristiky, změna vlastnictví nebo účel.
- *Výstup* – Výsledek transformace.

(Lucid Software Inc., 2024)

Jedná se o jeden ze základních diagramů, které je dobré provést před zahájením vývoje k ilustraci množství práce a nutné časové dotace.



Obrázek 4 - Příklad Workflow diagramu (Lucid Software Inc., 2024)

3.1.4 Uživatelský zážitek (UX)

Návrh UX (User Experience neboli uživatelský zážitek) je disciplína (nejen) softwarového inženýrství, která se zabývá navrhováním a optimalizací uživatelských interakcí se softwarem. Cílem je zajistit, aby uživatelé měli pozitivní, efektivní a uspokojivý zážitek při používání daného produktu. Při návrhu UX se zvažuje, kdo bude produkt vlastně používat, proč bude produkt používat a jak ho bude používat. Celý proces návrhu se točí okolo uživatele. V praxi to pak vypadá tak, že se provádí uživatelské testování a průzkumy, tvoří se persony a jinými způsoby se UX designéři snaží porozumět potřebám, preferencím a chováním uživatelů. Na základě poznatků z výzkumu se v rámci návrhu UX pak navrhuje i UI neboli uživatelské rozhraní (User Interface), které by mělo být intuitivní, snadno použitelné a esteticky přitažlivé, tvorba UI zahrnuje návrh drátových modelů (wireframes),

prototypů a samotného grafického designu. Dále se provádí interakční design, kdy se zkoumá, jak budou uživatelé interagovat s prvky rozhraní a jak budou tyto interakce uspořádány, tato etapa návrhu jde často ruku v ruce s testováním s reálnými uživateli, kdy se zjišťuje jak efektivní a příjemné je doopravdy užívání navrhnutého rozhraní. Na základě zpětné vazby se potom design upravuje, aby potřebám uživatelů vyhovoval co nejlépe. (Interaction Design Foundation, 2016)

V následujících podkapitolách se zaměřím na některé elementy návrhu UX trochu podrobněji.

3.1.4.1 Persony

Persony v rámci UX jsou fiktivní postavy nebo uživatelské profily, které reprezentují různé typy uživatelů, kteří mohou používat daný produkt nebo službu. Cílem person je lépe porozumět potřebám, cílům, preferencím a chování uživatelů, což umožňuje UX designérům lépe zaměřit své návrhy a rozhodnutí na konkrétní uživatelské skupiny. Dalo by se říct, že se jedná o ucelený popis ideálního zákazníka. Persony jsou pak obvykle vytvářeny na základě reálných dat z uživatelského výzkumu, jako jsou rozhovory s uživateli, dotazníky, pozorování a analýzy dat.

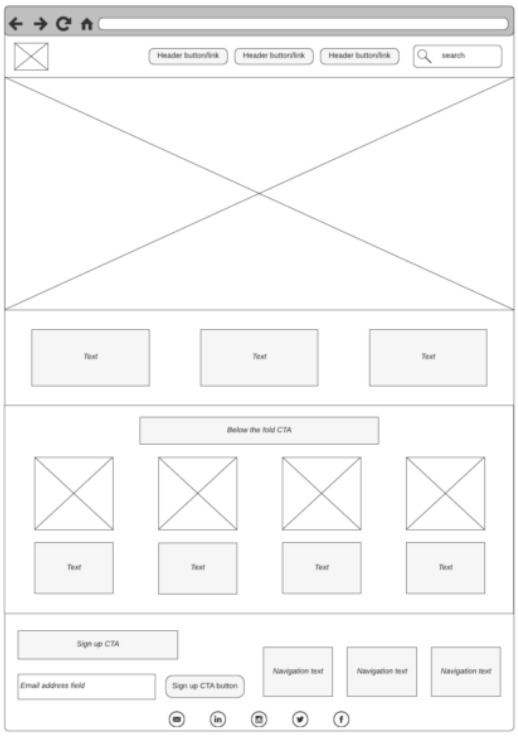
Persony obsahují mnoho informací, jako bychom popisovali reálného člověka. Vytváří se pro ně demografické údaje jako je jméno a příjmení, věk, pohlaví, zaměstnání, děti, rodinný stav nebo třeba i fotografie. Čím se persona více podobá konkrétnímu reálnému člověku, tím lépe se s ní dá pracovat. Údaje persony je pak vhodné doplnit i o rozšiřující informace, jako je například ekonomický status nebo životní úroveň, vlastnosti nebo motivace a zájmy nebo oblíbená místa dotyčného, či typické chování nebo strachy. Tyto všechny údaje se personám většinou dávají, abychom z nich mohli vyvodit jejich pravděpodobný přístup k vyvíjenému produktu.

Kromě person, které jsou ideálním zákazníkem se tvoří také negativní persony, což jsou naopak persony, které jsou velmi nepravděpodobným zákazníkem nebo dokonce zákazníkem naprosto nevhodným. Tyto persony většinou nejsou tak detailní jako ostatní a

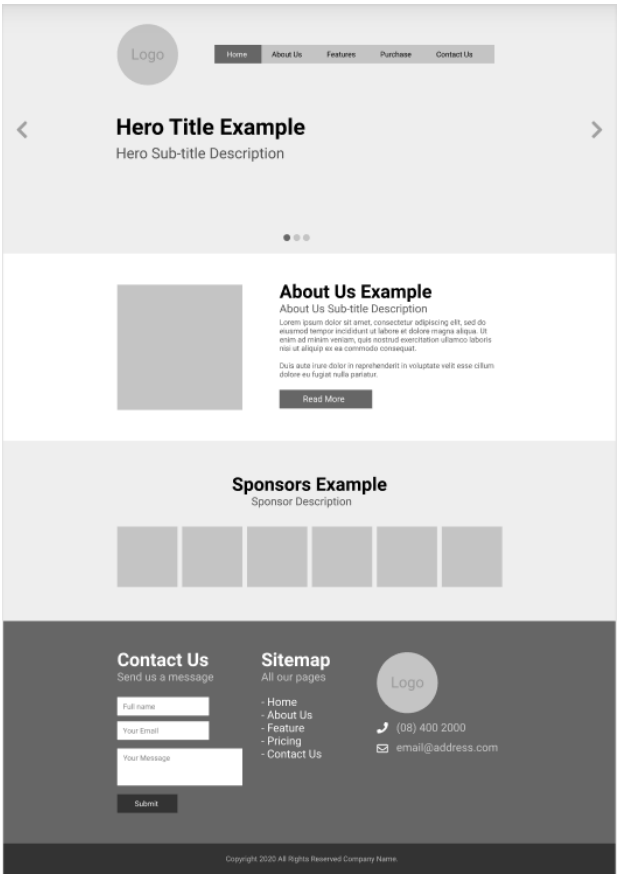
existují, protože by se produkt neměl nikdy tvořit tak, aby vyhovoval každému. (Růžičková, 2024)

3.1.4.2 Návrh UI

V procesu návrhu UX je velkou součásti i návrh UI (Uživatelské rozhraní). Většinou započíná návrhem informační architektury, kdy se zvažuje organizace obsahu a navigační struktura rozhraní, tato etapa také často zahrnuje kreslení prvních návrhů rozhraní tužkou na papír. Dalším krokem pak často bývá tvorba drátových modelů neboli wireframes. Jedná se o jakýsi první návrh, hrubý obrys, který prezentuje pouze ty nejdůležitější elementy UI a bývá pouze černobílý. Dalším krokem návrhu pak je design vizuálního stylu, který zahrnuje volbu barev, typografie, ikon a grafiky. Slouží k tomu, aby se vytvořil jednotný a atraktivní vzhled uživatelského rozhraní. Konečná fáze před samotnou implementací a testováním uživatelského rozraní je prototypování. V této fázi návrhu se vytváří prototyp, který by už měl reprezentovat potenciální finální vzhled webové stránky a měl by sloužit jako simulace, která uživatelům umožňuje testovat funkčnost a uživatelskou přívětivost rozhraní. Prototypy mohou být statické (např. obrázky) nebo interaktivní (klikací prototypy). (Hannah, 2024)



Obrázek 5 - Příklad drátěného modelu (wireframe) (Looka, 2024)



Obrázek 6 - Příklad ranného prototypu (Figma, 2024)

3.2 Programování

Programování je proces tvorby a psaní instrukcí, které počítač interpretuje a provádí. Tyto instrukce jsou obvykle psány v programovacím jazyce a určují, jakým způsobem má počítač vykonávat určité úkoly nebo operace. Cílem programování je vytvořit software, který řeší konkrétní problémy nebo poskytuje určité funkce nebo služby. Programování může být jak složité, tak i kreativní, a zahrnuje různé úrovně složitosti od jednoduchých skriptů až po komplexní aplikace a systémy.

Programovací jazyk je formální jazyk, který umožňuje programátorům psát instrukce pro počítačové programy. Tyto instrukce jsou psány v podobě kódu, který je následně interpretován nebo kompilován do strojového kódu, který počítač může provádět. Existuje mnoho různých programovacích jazyků, z nichž každý má svou vlastní syntaxi, sémantiku a použití. Některé z nejběžnějších interpretovaných programovacích jazyků zahrnují Python, Javascript, Ruby, PHP a z překládaných jazyků C++, Java nebo C#. Každý

programovací jazyk má své vlastní výhody a nevýhody a je vhodný pro různé typy projektů a aplikací. (Navone, 2022)

3.2.1 Objektově orientované programovaní (OOP)

Po vzniku programovacího jazyka Algol v roce 1958 se začalo programovat strukturálně, tedy programovalo se tak, že se psaly jednotlivé příkazy v pořadí, v jakém šly za sebou s využitím cyklů a větvení a funkcí. Využíval se zde princip tzv. funkcionální dekompozice, kdy se problém rozloží na více podproblémů a podle toho se vytvoří funkce, které je řeší. Problém ovšem je, že zde není možnost kód používat znovu s lehkými obměnami, pro každou novou funkcionalitu se musí přidávat nové funkce a může se stát, že se program stane tak složitým, že už se ho nevyplatí dále rozšiřovat. (Hartinger, 2024)

3.2.1.1 Historie

Strukturální postup vývoje byl ale brzy zastíněn novým paradigmatem a když říkám brzy, myslím tím počátky 60. let, kdy se poprvé objevily myšlenky, na kterých je Objektově orientované programování postavené. Tehdy vzniknul programovací jazyk Simula 67, který byl určen pro tvorbu diskrétních simulací. To pak vedlo ke tvorbě programovacího jazyka Smalltalk ve firme Xerox v Palo Alto v 70. letech. Smalltalk stavěl na simulačních principech Simuly a dále je rozvíjel a pomohl rozšířit povědomí o OOP, hlavně na univerzitách. K opravdovému nárůstu popularity ale přispělo až vytvoření C++ v Bellových laboratořích v roce 1983. Nejdřív se jmenoval pouze C with classes a byl vytvořen, protože byl zkrátka potřeba objektově orientovaný programovací jazyk, který by byl rychlejší než Smalltalk. Největší boom ale OOP zažilo až v roce 1995, kdy vyšla kniha „Design Patterns – Elements of Reusable Object-Oriented Software“, která představila 23 návrhových vzorů a přinesla revoluci vytváření softwaru. (Pecinovský, 2014)

3.2.1.2 Filozofie OOP

V čem je Objektový přístup k programování vlastně tak revoluční? Hlavní myšlenka je taková, že se snažíme dosáhnout vyšší abstrakce a jakési simulace reálného světa, aby nám principy programování byly bližší a přirozenější. Základním stavebním kamenem je objekt. Jako i v reálném světe, všechny interakce jsou způsobené objekty, které mají nějaké vlastnosti a nějak na sebe působí. Zatímco strukturovaný program je jakýsi strohý postup řešení dané úlohy, objektově orientovaný program je množina objektů a zpráv, které si

mezi sebou tyto objekty předávají. Při návrhu strukturovaného programu se vymýšlí postupy a datové struktury, na které se pak budou tyto postupy aplikovat a při návrhu objektově orientovaného programu se programátor snaží nalézt jednotlivé účastníky a pak definovat zprávy, které si mezi sebou budou posílat. Na nejnižší úrovni se vlastně jedná o to samé, objektově orientovaný model ale poskytuje vyšší míru abstrakce, která je snadnější na pochopení. (Pecinovský, 2014)

3.2.1.3 Vlastnosti objektu

Objekt obvykle obsahuje jiné objekty. Myslím tím vlastnosti onoho objektu a narážím tím na to, že v OOP je v podstatě všechno objekt, včetně vlastností, událostí, stavů a dějů. Vlastnosti objektu obvykle jsou atributy a metody. Atribut je jakási vlastnost objektu, může se jedna o proměnné s nějakými hodnotami nebo o další objekty, které tento objekt definují a dávají mu jeho význam v programu. Metoda je potom definice toho, jak bude objekt reagovat na zprávy. Většinou se neříká, že se posílá objektu nějaká zpráva, ale že se volá metoda objektu. Jedná se tedy o jakousi funkci definující chování objektu.
Před tím, než můžeme vytvořit objekt, musíme vytvořit třídu, která je jakousi šablonou, která definuje, jak se budou tvořit objekty, které se pak označují jako instance dané třídy. Třída definuje, jaké budou mít její instance atributy a metody. (Pecinovský, 2014)

3.2.1.4 Principy OOP

Paradigma objektově orientovaného programování s sebou přináší několik základních principů, které ho nadále odlišují od strukturovaného programování. Jedná se především o:

- *Polymorfismus* – V moderním programování je kladen velký důraz, aby jednotlivé části programu, tedy jednotlivé objekty, byly izolované a aby zbytek programu neměl přístup dovnitř objektu nebo k jeho vlastnostem, pokud to není nutné pro funkci programu. Velkým pomocníkem je v tomto ohledu rozhraní. Rozhraní je taková další šablona nad třídou. Zatímco třída je šablonou pro objekt a definuje jeho vlastnosti. Rozhraní definuje chování, které musí třída implementovat, ale neposkytuje konkrétní implementaci tohoto chování. Jinými slovy rozhraní určuje, co musí třída dělat a třída pak definuje, jak to dělá. A tomuto principu se právě říká polymorfismus neboli mnohotvarost. V zásadě to znamená, že jedna metoda může mít více podob, které se liší podle konkrétní implementace.

- *Zapouzdření* – Jedná se o praktiku, kdy se atributy a metody, které s těmito atributy pracují, umisťují pohromadě pouze do definice třídy a zároveň se znemožňuje zbytku programu manipulovat s těmito daty jakýmkoliv jiným způsobem než prostřednictvím těchto metod. Použití zapouzdření jde ruku v ruce s použitím rozhraní. Oba tyto principy zvyšují izolaci a samostatnost jednotlivých bloků kódu, čímž se zvyšuje efektivnost vývoje a spolehlivost programu. Zjednodušeně by se dalo říct, že zapouzdření zajišťuje, že k datům objektu je možné přistupovat pouze způsobem definovaným v jeho rozhraní.
- *Dědičnost* – je mechanismus, který umožňuje vytvářet nové třídy na základě již existujících tříd s tím, že nová třída (potomek) dědí atributy a metody ze své nadřazené třídy (rodič) a může je rozšiřovat nebo upravovat. Dědičnost s sebou nese dva problémy a to ty, že potomkům je umožněno pracovat s daty rodiče a rodič musí potomkovi "prozradit" detaily o své implementaci, čímž se porušují principy zapouzdření. Obecně tedy platí, že dědičnost by se měla použít jenom, pokud je jakékoliv jiné řešení výrazně méně efektivní a zároveň pokud chceme, aby kód zůstal stabilní, potomek by měl být vždy speciálním případem předka. Částečně se dá problémům s porušováním zapouzdření předejít implementací vhodného rozhraní.
- *Abstrakce* – Jedním z pravidel OOP je programovat proti rozhraní, a ne proti implementaci, aby se dosáhlo co největší obecnosti kódu. Programátor se snaží, aby se jednotlivé objekty chovaly pouze jako izolované černé skříňky a navzájem nevěděly o svých atributech a metodách, pokud nejsou veřejné. Tomuto principu se říká abstrakce. Obecně se jedná o proces zaměřený na vytvoření zjednodušeného pohledu na složité systémy. V OOP je abstrakce dosaženo pomocí již zmíněných rozhraní a také abstraktních tříd, které také definují vlastnosti, které by třída měla implementovat, ale mohou i nemusí konkrétní implementaci poskytnout, také, na rozdíl od rozhraní se od Abstraktní třídy dá dědit. Obecně se používá, když je potřeba sdílet společné chování mezi několika třídami.

(Pecinovský, 2014)

3.2.2 Návrhové vzory

Současné programování bylo výrazně ovlivněné příchodem návrhových vzorů. Jejich použití zvyšuje efektivitu vývoje, robustnost výsledných programů a spolehlivost. Zvyšuje také jejich spravovatelnost a umožňuje mnohem rychlejší reakce na změny v zadání. Jedná se o jakési osvědčené vzorce řešení situací, které se v programování často vyskytují. Použití návrhových vzorů má za následek rychlejší vývoj, protože se u častých problémů nemusí znovu vymýšlet nové řešení, také to znamená, že vývoj je kvalitnější, neboť odpadá možnost tvorby chyb při vymýšlení unikátního řešení. Dále se vývoj zlevňuje, protože použití návrhových vzorů produkuje znovupoužitelný kód, který se dá použít i napříč více programy a zlepšuje se i komunikace mezi programátory. (Pecinovský, 2014)

Existuje mnoho různých typů návrhových vzorů, které pokrývají různé oblasti softwarového návrhu, včetně správy objektů, vytváření a správy relací mezi objekty, rozložení funkcí a zodpovědností mezi komponentami systému, zpracování dat a další. Mezi známé návrhové vzory patří například Singleton, Factory Method, Observer, Strategy, Decorator a další. Každý z těchto vzorů řeší určitý typ problému a může být aplikován v různých situacích při návrhu softwaru. (Neckář, 2016)

3.2.3 SOLID principy

SOLID je akronym, který označuje pět základních principů objektově orientovaného návrhu softwaru. Tyto principy byly navrženy tak, aby zlepšily modularitu, flexibilitu a udržitelnost softwarových systémů. Každý jednotlivý princip má svůj vlastní účel a přispívá k dosažení dobrého návrhu softwaru. Zde jsou jednotlivé SOLID principy:

- *Single Responsibility Principle (Princip jedné odpovědnosti)* - Tento princip říká, že každá třída by měla mít pouze jednu zodpovědnost, tedy měla by být odpovědná pouze za jeden aspekt funkcionality softwaru. To znamená, že třída by měla mít jen jeden důvod ke změně, tedy jedna třída by měla mít pouze jeden účel.
- *Open/Closed Principle (Princip otevřenosti/zavřenosti)* - Podle tohoto principu by měly být třídy otevřeny pro rozšíření, ale uzavřeny pro úpravy. To znamená,

že by mělo být možné měnit chování tříd pouze pomocí dědičnosti a rozhraní, aniž by bylo nutné upravovat jejich existující vlastnosti.

- *Liskov Substitution Principle (Liskovova substituční zásada)* - Tento princip říká, že instance potomků tříd by měly být schopny být nahrazeny instancemi rodičovských tříd bez změny očekávaného chování programu. Jinými slovy, by mělo být možné použít potomky tříd tam, kde jsou očekávány instance rodičovských tříd, aniž by to narušilo integritu systému. S tímto principem souvisí, že v dědění by potomek měl vždy být speciálním případem rodiče.
- *Interface Segregation Principle (Princip segregace rozhraní)* - Tento princip říká, že žádná třída by neměla být nucena implementovat metody, které nepotřebuje. Místo toho by měla existovat specifická rozhraní pro specifické účely, která budou implementována pouze těmi třídami, které je skutečně potřebují, tedy místo jednoho všeobjímajícího rozhraní by se mělo tvořit více specializovaných.
- *Dependency Inversion Principle (Princip inverze závislostí)* - Tento princip říká, že moduly by neměly záviset na konkrétní implementaci jiných modulů, ale na jejich abstrakcích. To znamená, že by měly být závislosti mezi třídami směřovány od abstrakcí k implementacím, nikoli naopak, tedy programovat by se mělo proti rozhraní, a ne proti implementaci. (Oloruntoba, 2024)

3.3 Webová stránka a aplikce

V této kapitole budou vysvětleny základní principy webových stránek a aplikací spolu s jejich architekturou.

3.3.1 Webové stránka

Webové stránky jsou v zásadě digitální entity dostupné na internetu. Jedna webová stránka je dokument, který lze zobrazit ve webovém prohlížeči po zadání unikátní adresy, kterou má každá stránka, která je dostupná na internetu. Jeden web se může skládat z více propojených webových stránek, které sdílí jedno doménové jméno. Webové stránky jsou tvořeny v jazyce HTML a mohou obsahovat také informace o stylu, skripty a média.

Informace o stylu, je tvořena v CSS a určuje, jak webová stránka vypadá. Skripty se pak programují v JavaScriptu a přidávají do stránek interaktivitu a média jsou soubory jako obrázky, zvuky nebo videa. (Mozilla, c1998–2024)

Webové stránky pak můžeme ještě rozdělovat podle toho, jestli jsou statické nebo dynamické.

- *Statické webové stránky* pouze zobrazují pevně daný obsah, který je vždy stejný pro všechny uživatele. Jedná se o nejstarší způsob, jak dělat web a dnes se statické webové stránky používají pro malé weby, které mají zobrazovat nějaké informace, jako například osobní portfolia, životopisy nebo brožury.
- *Dynamické webové stránky* pak jsou více komplexní a mohou generovat nebo měnit svůj obsah v reálném čase. Většinou se tohoto efektu docílí pomocí JavaScriptu. Dnes je většina webů dynamických, může se jednat například o stránky, které přizpůsobují obsah podle potřeb nebo preferencí každého uživatele.

(Novikova, c2016-2024)

3.3.2 Webová aplikace

O webové aplikaci by se potom dalo říci, že je to speciální případ dynamické webové stránky. Vyznačuje se vysokou úrovní interaktivity a funkcionalitou, která se často podobá klasickým desktopovým aplikacím s tím rozdílem, že nevyžaduje žádnou lokální instalaci a používá se prostřednictvím prohlížeče, stejně jako jakákoliv jiná webová stránka. Webové aplikace mohou provádět složité operace, pracovat s databázemi, přenášet data mezi uživatelem a serverem a více.

Jelikož webové aplikace často provádějí celkem komplexní operace, jejich architektura může být mnohdy dost komplikovaná. Webové aplikace jsou obvykle navrhovány s různými vrstvami. Nejběžnější návrhový model zahrnuje tři vrstvy: prezentační (webový

prohlížeč), aplikační (server) a datovou (databáze). Prezentační vrstva předává uživatelská data aplikační vrstvě, která je zpracuje a může provádět různé akce, včetně jejich ukládání do datové vrstvy.

Některé webové aplikace mohou dosáhnout takové složitosti, že třívrstvový model přestane dostačovat. V takových případech může být potřeba přidat do architektury další vrstvy. Například zavedení integrační vrstvy mezi aplikační a datovou vrstvou může poskytnout jednotné rozhraní pro přístup k datům, což umožní izolaci aplikační vrstvy od změn, které se mohou objevit v implementaci vrstvy datové.

Typickým příkladem webové aplikace jsou třeba sociální sítě nebo e-shopy.
(Codeacademy, 2024)

3.3.2.1 MVC

MVC (Model-View-Controller) je architektonický vzor často používaný při návrhu softwarových aplikací, zejména webových aplikací. Jedná se o strukturu rozdělení zodpovědností mezi různé části aplikace s cílem zvýšit modularitu, usnadnit údržbu a zlepšit škálovatelnost. MVC rozděluje aplikaci do tří částí:

- *Model* – reprezentuje datový model aplikace. Obsahuje logiku pro manipulaci s daty, jako jsou operace čtení, zápis, aktualizace a mazání.
- *View* – je zodpovědné za zobrazení dat uživateli. Může to být uživatelské rozhraní, které vidí uživatel, jako je webová stránka, formulář nebo grafické rozhraní. View získává potřebná data z modelu (obvykle prostřednictvím controlleru) a zobrazuje je uživateli ve vhodné formě.
- *Controller* – je prostředníkem mezi modelem a view, obsahuje veškerou programovou logiku. Zpracovává uživatelské vstupy a volá odpovídající metody v modelu pro získání nebo aktualizaci dat. Poté rozhoduje, které view má být zobrazeno uživateli na základě výsledků operací provedených na modelu.

(Mozilla, c1998–2024)

3.3.3 Webový Server

Webový server je softwarová aplikace nebo hardwarové zařízení, které zajišťuje distribuci webových stránek a dalšího webového obsahu po internetu. Jeho hlavní funkcí je zpracování a odpovídání na HTTP požadavky, které přicházejí od uživatelů přes internetový prohlížeč.

Webový server běží na serverovém počítači a obsahuje konfigurovatelné nastavení, které určuje, jak bude obsluhovat požadavky klientů. Když uživatelé zadají URL do svého prohlížeče, prohlížeč odešle HTTP požadavek na webový server. Webový server poté zpracuje tento požadavek, vyhledá požadovaný soubor nebo stránku, tu přečte a zpracuje podle potřeby a odešle odpovídající HTTP odpověď zpět uživateli.

Stejně jako webové stránky, i webový server může být statický nebo dynamický. Statický webový server zkrátka odesílá uživateli webovou stránku takovou, jak se nachází v jeho úložišti bez dalšího zpracování. Dynamický webový server obsahuje software navíc jako aplikační server nebo databázi. Tento typ serveru zpracuje a změní podobu webové stránky předtím, než jí odešle uživateli. Nejčastěji jí naplní daty z databáze.

Mezi známé webové servery patří například Apache nebo Nginx. (Mozilla, c1998–2024)

3.3.4 Webové Frameworky

Když se bavíme o vývoji webových aplikací, je důležité zmínit také webové frameworky. Framework je softwarový nástroj nebo sada nástrojů, které poskytují standardizovanou strukturu a přístup k vytváření webových aplikací. Tyto frameworky zjednodušují proces vývoje tím, že nabízejí opakovaně použitelný kód a hotová řešení pro běžné úkoly, jako je manipulace s daty, směrování, správa stavu aplikace a vytváření uživatelského rozhraní. Mezi populární frameworky patří například React, Angular, Vue.js pro frontend a Express.js, Django, Ruby on Rails pro backend. Backendové frameworky řídí serverovou stranu aplikace a zajišťují přístup k databázím a zpracování dat. Frontendové frameworky pomáhají vytvářet esteticky přitažlivé webové aplikace pomocí stylování a formátování prvků uživatelského rozhraní. (Spittel, c2016-2024)

3.4 Backend

Backend je část softwarového systému, která je zodpovědná za zpracování a uchovávání dat, logiku aplikace a komunikaci s databází, webovými službami a dalšími serverovými technologiemi. Jedná se o "základní" část aplikace, která není přímo viditelná pro uživatele, ale poskytuje nezbytnou funkcionalitu pro správné fungování. Nachází se na straně serveru a poskytuje funkcionalitu a podporu pro frontendovou (klientskou) část aplikace. (Wintemute, 2024)

3.4.1 API

API neboli Application Programming Interface (Rozhraní pro programování aplikce) je sada procedur, funkcí, protokolů a nástrojů, které umožňují softwarovým aplikacím komunikovat a mezi sebou interagovat. V Praxi se API používá k zajištění komunikace mezi dvěma platformami, aby si mohly mezi sebou vyměňovat data, což v rámci vývoje webů pak umožňuje například oddělit frontendovou a backandovou část aplikace. API se nepoužívá jenom ve webovém vývoji, i když je zde velmi populární. Hojně se používá např. i při vývoji mobilních aplikací. Je užitečně ale kdekoliv, kde se počítá s tím, že si dvě aplikace budou předávat nějaká data. Může se jednat třeba o desktopovou aplikaci, komunikující s operačním systémem. V zásadě tedy API funguje jako jakýsi prostředník mezi uživatelským klientem a databází nebo daty. Web nebo aplikace odešle žádost, kterou API přijme a prohledá databázi a pošle zpět výstup ve specifickém formátu, nejčastěji v JSONu. (Koďousková, 2024)

3.4.1.1 REST API

Pokud se bavíme v kontextu vývoje webových aplikací, REST Api je asi nejpoužívanější řešení backendu. REST znamená „Representational State Transfer" a byl poprvé zaveden v roce 2000 s cílem zlepšit výkon, škálovatelnost a jednoduchost tím, že ukládá konkrétní limity na API. REST se stal populární díky možnosti jej využít pro velké množství aplikací. V praxi REST API funguje tak, že klientovi poskytují textovou reprezentaci svých dat (prostředků). Klient si tyto prostředky může vyžádat pomocí bez stavového HTTP protokolu. HTTP requesty obvykle obsahují koncový bod v API, metodu jako je put, update, get nebo delete, pak také obshují hlavičku, která obsahuje informace o těle

requestu nebo informace o autentizaci, a nakonec samotné tělo s obsahem requestu. REST API se od ostatních typů API liší tím, že se řídí několika základními principy:

- *Bezstavová komunikace:* Každý request od klienta obsahuje všechny informace potřebné k provedení požadované akce. Server nemá žádný kontext o stavu klienta mezi jednotlivými requesty, což umožňuje serveru nebo klientovi zpracovat každou zprávu i když neznají žádné předchozí.
- *Jednotné rozhraní:* Rozhraní mezi klientem a serverem by mělo být jednotné a snadno srozumitelné. To zahrnuje standardní HTTP metody pro manipulaci s daty (GET pro čtení, POST pro vytvoření, PUT pro aktualizaci a DELETE pro smazání) a používání URI pro identifikaci zdrojů. Další princip jednotného rozhraní stanoví, že zprávy by měly být samo popisné. Musí být srozumitelné pro server, aby určil, jak s nimi naložit (například typ requestu, mime typy atd.)
- *Klient-server architektura:* Klient a server jsou odděleny a mohou být vyvíjeny a nasazovány nezávisle na sobě. Je možné měnit klienta, aniž by se tím narušila činnost serveru stejně jako změny na straně server neovlivní funkčnost klienta. To umožňuje větší modularitu a škálovatelnost aplikace.
- *Ukládání do mezipaměti:* Odpovědi ze serveru by měly být označeny jako cachovatelné nebo ne, což umožňuje klientům ukládat odpovědi do mezipaměti a snížit tak zátěž na server. Tento princip dává serveru více možností škálování díky menší zátěži. Mezi paměť zvyšuje rychlost načítání stránky a umožňuje přístup k dříve prohlíženému obsahu bez internetového připojení.
- *Vrstvený systém:* Architektura by měla být vrstvená, což znamená, že mezi klientem a serverem mohou existovat další servery nebo prostředníci, které zprostředkovávají komunikaci. To umožňuje větší flexibilitu a rozšiřitelnost systému.

(Husar, 2024)

3.4.2 Databáze

Databáze jsou strukturované soubory dat, které umožňují efektivní organizaci, ukládání a manipulaci s informacemi. Tyto informace mohou zahrnovat různé typy dat, jako jsou textové řetězce, čísla, obrázky nebo dokonce kompletní dokumenty. Databáze umožňují uživatelům ukládat a vyhledávat data podle různých kritérií a provádět operace jako vkládání, aktualizaci, mazání a dotazování se na data. (Oracle, 2024)

Existují různé typy databázových systémů, včetně relačních databází, které ukládají data do tabulek a používají relace mezi tabulkami k organizaci dat, a NoSQL databází, které používají jiné modely pro ukládání dat, jako jsou dokumenty, grafy nebo páry klíčů a hodnot. (Oracle, 2024)

Databáze sahají až do šedesátých let, tedy do éry sálových počítačů, kdy firma IBM vyvinula tzv. hierarchickou databázi, která umožňovalo spojovat pouze záznamy 1:1, proto se pak v 80. letech objevila síťová databáze, která tento nedostatek řeší, nakonec byla ale také překonána příchodem relační databáze v 60. letech. Existují ještě objektové databáze, které místo tabulek a relací používají objekty a umožňují úzkou spolupráci s objektově orientovanými programovacími jazyky. (Můčka, 2024)

3.4.2.1 Relační Databáze

Jedná se o dnes nejpoužívanější typ databází. Relační databáze jsou typem databázového systému, který ukládá data ve formě tabulek, které jsou navzájem propojeny pomocí vztahů. Tyto tabulky jsou pak strukturovány do řádků a sloupců, přičemž každý řádek představuje jeden záznam a každý sloupec reprezentuje jednotlivý atribut nebo pole dat.

V relačních databázích jsou vztahy mezi různými tabulkami definovány prostřednictvím klíčů, což jsou atributy, které jsou unikátní identifikátory záznamů. Primární klíče identifikují jedinečné záznamy v jedné tabulce, zatímco cizí klíče propojují odpovídající záznamy v jiné tabulce. Primární klíče musí být vždy nenulové a neměnné.

Relační databáze jsou základním kamenem mnoha aplikací a systémů díky své schopnosti efektivně ukládat a manipulovat s daty, a to prostřednictvím standardního dotazovacího jazyka nazývaného SQL (Structured Query Language). SQL umožňuje uživatelům provádět různé operace nad daty, včetně vkládání, aktualizace, mazání a dotazování se na data.
V porovnání s nerelačními databázemi lze říci, že oba typy databází umožňují vertikální škálování, což zahrnuje přidání hardwarových zdrojů jako RAM, kapacity disku nebo počtu procesorových jader. Avšak u nerelačních databází je také možné škálování horizontální, což zahrnuje přidání samostatných serverů. Tento přístup umožňuje flexibilnější přidání zdrojů bez potřeby předem definované architektury, jakou vyžadují relační databáze. Relační databáze jsou konzistentní a mají vysokou integritu dat, ale jsou méně flexibilní, protože vyžadují pevné schéma atributů a řádků. Naopak, nerelační databáze umožňují vložení libovolného počtu a typu parametrů, což je činí mnohem flexibilnějšími, ale méně konzistentními. (Můčka, 2024)

3.4.2.2 SŘBD

Systém řízení báze dat (SŘBD), jak už napovídá jeho název, je systém určený k řízení relačních databází. SŘBD umožňuje uživateli vytvářet jednu nebo více databází a souvisejících objektů, jako jsou tabulky, vztahy, pohledy a dotazy. Dále podporuje základní údržbu, jako je zálohování databáze a obnova záložního souboru. Kromě toho zajišťuje bezpečnost, aby uživatelé a skupiny měli přístup pouze k databázím a datům, ke kterým mají oprávnění. Většina komerčních SŘBD nabízí nad rámec těchto základních funkcí ještě mnoho dalších. Většina zahrnuje nástroje pro monitorování a optimalizaci výkonu svých databází a mnoho z nich také obsahuje služby pro generování reportů a prezentaci výsledků dotazů. Některé dokonce zahrnují složité balíčky pro analýzu obchodních trendů a vzorů. (Conger, 2012)

3.4.2.3 SQL

Původně vzniknul v 70. letech ve firmě IBM ve spolupráci s firmou Oracle. SQL (Structured Query Language) je standardizovaný jazyk pro manipulaci s relačními databázemi. Používá se pro vytváření, modifikaci a dotazování dat v databázových

systémech. SQL umožňuje provádět různé operace, jako jsou vkládání nových záznamů do tabulek, aktualizace existujících záznamů, mazání záznamů, vytváření a úpravy tabulek a provádění složitých dotazů pro získání potřebných informací z databáze.

SQL je deklarativní jazyk používaný pro manipulaci s daty v relačních databázích. Na rozdíl od procedurálních jazyků, jako jsou C++, Java nebo C#, SQL nevyžaduje specifikaci konkrétních kroků, ale spíše definuje, co má být provedeno. SQL nerozlišuje velikost písmen, ale běžně se klíčová slova SQL píší velkými písmeny pro lepší čitelnost. SQL také ignoruje většinu bílých znaků, což umožňuje volnější formátování příkazů. SQL příkazy jsou navíc svojí syntaxí velmi podobné mluvené angličtině. SQL lze rozdělit do dvou hlavních oblastí funkcionality: Jazyka manipulace s daty (DML neboli Data Manipulation Language), který zahrnuje příkazy pro výběr (Select) a manipulaci s daty (Insert, Delete, Update), a Jazyka definice dat (DDL neboli Data Definition Language), který zahrnuje příkazy pro vytváření (Create), úpravu (Alter) a odstraňování (Drop) objektů v databázi. (Conger, 2012)

3.4.2.4 Normalizace Databáze

Normalizace databáze je proces strukturování relačních tabulek tak, aby se minimalizovala redundance dat a možnost vzniku anomálií během manipulace s daty. Existuje několik forem normalizace, z nichž každá řeší specifické anomálie ve struktuře databáze.

První tři normální formy (1NF, 2NF, 3NF) jsou základní a nejčastěji používané při návrhu databází. Cílem první normální formy je eliminovat opakující se skupiny nebo pole v tabulkách a zajistit, aby každý atribut obsahoval pouze jednu hodnotu jednoho typu. Druhá normální forma se zabývá odstraněním funkčních závislostí mezi atributy a klíčem tabulky. Třetí normální forma pak odstraňuje transitivní závislosti, kde jeden atribut závisí na jiném atributu, který není primárním klíčem.

Proces normalizace pomáhá zajistit integritu dat a minimalizovat redundanci, což vede ke zlepšení výkonnosti a správnosti operací s databází. Je důležité provést normalizaci s ohledem na konkrétní potřeby aplikace a optimalizovat strukturu databáze pro konkrétní použití.

Kromě prvních třech normálových forem existují ještě Boyce-Codd Normalálová Forma (BCNF), 4NF, 5NF a Normální forma klíče domény (DK/NF). Tyto další normální formy se obvykle používají k dalšímu zdokonalení struktury databáze v závislosti na konkrétních potřebách a požadavcích aplikace. Jejich cílem je eliminovat další možné anomálie a zlepšit celkovou kvalitu a efektivitu databázového návrhu.

(Conger, 2012)

3.4.2.5 Integrita dat v databázi

Integrita dat je základním pilířem databázových systémů, který zajišťuje konzistenci, validitu a spolehlivost uložených informací. Bezpečná a důvěryhodná data umožňují efektivní rozhodování, ochranu před chybami a útoky a mnohdy i splnění právních a regulačních požadavků. Zajištění integrity dat je tedy klíčové pro úspěšné a bezproblémové fungování databáze.

Integritní omezení jsou:

- *Entitní Integrita:* Zajišťuje, že každá entita v databázi je správně identifikována a udržována v souladu s definovanými pravidly a omezeními, například pomocí jedinečných identifikátorů nebo primárních klíčů.
- *Referenční integrita:* Zajišťuje, že odkazy mezi různými entitami v databázi jsou konzistentní a platné. Například, pokud entita A odkazuje na entitu B pomocí cizího klíče, referenční integrita zabraňuje vytvoření odkazu na neexistující entitu v entitě B. Jinými slovy, toto omezení zabraňuje záznamům odkazovat na jiné neexistující záznamy. K tomu může dojít například po smazání záznamu.
- *Doménová Integrita*: Zajišťuje, že hodnoty uložené v jednotlivých sloupcích nebo polích databáze odpovídají definovaným pravidlům a omezením. Například, pokud je sloupec určen pro ukládání čísel, integrita domény zabraňuje uložení řetězce nebo jiného datového typu.

(Naeem, 2024)

3.4.3 MySQL

MySQL je světově nejpopulárnější open-source relační databáze, která je využívána v mnoha klíčových aplikacích, jako jsou sociální sítě, streamovací služby, e-commerce platformy a další. Je známá pro svou spolehlivost, rychlost a jednoduchost použití. V logu má delfína, který se jmenuje Sakila.

Díky své otevřenosti je MySQL často preferovanou volbou pro vývojáře, kteří si mohou zdarma stáhnout a případně i upravovat její zdrojový kód. MySQL podporuje mnoho programovacích jazyků a poskytuje širokou škálu funkcí, jako je například podpora pro transakce, indexování, zálohy a obnova dat, správa uživatelů a mnoho dalšího.

Jednou z významných vlastností MySQL je její schopnost škálovat se podle potřeb aplikace. Díky možnosti využití v cloudových prostředích a podpory pro replikaci dat může MySQL snadno zvládnout zpracování velkých objemů dat a podporovat rozsáhlé databáze.

MySQL HeatWave je další úroveň vývoje MySQL, která přináší plně řízenou databázovou službu s akcelerátorem dotazů v paměti. Tato služba umožňuje rychlé provádění komplexních analýz dat a strojového učení přímo v databázi, což vede k významnému zrychlení výpočtů a snížení nákladů na infrastrukturu. (Oracle, 2024)

3.4.3.1 Stručná historie MySQL

Vývoj MySQL sahá až do roku 1979, kdy Monty Widenius ve společnosti TcX vytvořil nástroj pro generování reportů napsaný v jazyce BASIC. Postupně byl nástroj přepsán do jazyka C a portován na systém Unix. Tento nástroj se nazýval Unireg a byl primárně nízkoúrovňovým úložným strojem s uživatelským rozhraním pro generování reportů. V 90. letech začali zákazníci TcX požadovat SQL rozhraní k jejich datům, což vedlo k vytvoření MySQL v roce 1996. MySQL byl původně uvolněn pod speciální licencí, která umožňovala komerční použití. Během následujících let byl MySQL postupně rozšiřován, až se stal jedním z nejrozšířenějších open source relačních databázových systémů na světě. V roce 1999–2000 byla založena samostatná společnost MySQL AB, která začala rozvíjet MySQL verze 4.0 a 4.1 s podporou transakcí a dalšími pokročilými funkcemi. V roce 2005 byla vydána stabilní verze 5.0, která přinesla další vylepšení, jako je členění dat tabulek,

replikace na základě řádků a plánovač událostí. MySQL se stále aktivně vyvíjí, přičemž nové funkce jsou integrovány do budoucí verze 5.2. (O'Reilly Media, 2024)

3.4.4 PHP

PHP (rekurzivní zkratka pro PHP: Hypertext Preprocessor) je široce používaný skriptovací jazyk obecného určení, který se často využívá při vývoji webových aplikací a umožňuje jeho vložení do HTML stránek. PHP umožňuje integraci HTML kódu s jeho vlastním skriptovacím kódem, který je obklopen speciálními počátečními a koncovými značkami <?php a ?>.

V porovnání s klientským JavaScriptem se kód PHP spouští na straně serveru, tam se jako jeho výsledek generuje HTML kód, který se poté odesílá klientovi. Klient obdrží výsledky tohoto PHP skriptu, ale není schopen vidět samotný kód, který byl použit. Díky této vlastnosti může být PHP použito také k zpracování všech HTML souborů na serveru, což poskytuje vyšší úroveň zabezpečení a značnou flexibilitu.

PHP je známý svou jednoduchostí pro začátečníky, ale zároveň nabízí širokou škálu pokročilých funkcí pro profesionální programátory. Může být použit pro různé účely, nejen pro webové aplikace, což jej činí velmi univerzálním a flexibilním nástrojem v oblasti softwarového vývoje. (The PHP Group, 2024)

```php
<!DOCTYPE html>
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
            echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

Obrázek 7 - Příklad PHP kódu (The PHP Group, 2024)

Existuje několik hlavních oblastí, kde se PHP skripty běžně používají:

- *Serverové skriptování*: PHP je často využíván pro generování dynamického obsahu webových stránek. Pro tento účel je nezbytné mít nainstalovaný PHP parser, webový server a prohlížeč.
- *Skriptování z příkazové řádky:* PHP lze použít pro spouštění skriptů přímo z příkazové řádky, což je užitečné pro automatické úlohy nebo zpracování textových souborů.
- *Psaní desktopových aplikací***:** I když PHP není primárně určen pro vývoj desktopových aplikací, s využitím PHP-GTK lze psát i desktopové aplikace s grafickým uživatelským rozhraním.

PHP podporuje všechny hlavní operační systémy a je kompatibilní s mnoha webovými servery. Jeho schopnosti nejsou omezeny pouze na generování HTML, ale zahrnují i práci s různými datovými formáty, komunikaci se serverovými službami a práci s databázemi.

(The PHP Group, c2001-2024)

3.4.4.1 Krátká historie PHP

PHP je výsledkem vývoje produktu nazvaného PHP/FI, který v roce 1994 vytvořil Rasmus Lerdorf. Původně se jednalo o sadu jednoduchých skriptů pro rozhraní CGI napsaných v jazyce C, které byly používány pro sledování návštěv Rasmuseva online životopisu. Postupně se funkčnost rozšiřovala a Rasmus přepsal kód, což vedlo k vytvoření bohatší implementace schopné interakce s databázemi. V roce 1995 byl zveřejněn zdrojový kód PHP/FI pro veřejnost a tím byla umožněna jeho další modifikace a vylepšování ze strany uživatelů. PHP postupně získávalo popularitu, ačkoli bylo omezeno primárně na UNIXové systémy. Díky své snadné adoptaci pro vývojáře s vědomostmi v jazycích jako C nebo Perl se stalo atraktivní volbou pro tvorbu dynamických webových aplikací.

PHP 3 byla první verzí, která připomínala PHP tak, jak ho známe dnes. V roce 1997 Andi Gutmans a Zeev Suraski zahájili úplné přepsání parseru, aby vylepšili výkon a modulárnost kódu. Spolupracovali s Rasmusem, aby vytvořili nový, nezávislý programovací jazyk, který se nyní nazýval jednoduše 'PHP'. PHP 3 nabízelo silnou

rozšiřitelnost a podporu objektově orientovaného programování, což vedlo k jejímu ještě většímu úspěchu. Krátce po oficiálním vydání PHP 3 v roce 1998 začali Andi Gutmans a Zeev Suraski pracovat na přepsání jádra PHP, což vedlo k vydání PHP 4 v roce 2000 s novým jádrem Zend Engine 2.0 a mnoha dalšími funkcemi. PHP 5 bylo vydáno v červenci 2004 s novým jádrem Zend Engine 2.0, novým objektovým modelem a dalšími novinkami.

(The PHP Group, c2001-2024)

3.4.4.2 Composer

Composer je nástroj pro správu závislostí v PHP projektech. Umožňuje snadnou instalaci a aktualizaci knihoven a balíčků třetích stran potřebných pro správný chod aplikace. Nejedná se o správce balíčků v tradičním smyslu, ale o nástroj zaměřený na správu závislostí v rámci jednoho projektu, inspirující se nástroji jako npm pro node.js a bundler pro Ruby. Používá se pro efektivní řízení balíčků a minimalizaci problémů s verzováním a konflikty závislostí v PHP projektech. (Composer Community, 2024)

3.4.5 Laravel a Lumen

Laravel je PHP webový aplikační framework pro vytváření moderních webových aplikací. Nabízí širokou škálu funkcí, jako je správa závislostí, abstrakce databáze, fronty a plánované úlohy, testování a další. Framework je vhodný jak pro začátečníky v PHP, tak pro zkušené vývojáře, a nabízí možnost postupného rozšiřování do komplexnějších projektů. Laravel je také snadno škálovatelný a podporuje různé typy databází. (Laravel Holdings, c2011-2024)

Laravel vznikl jako výsledek vývoje a iniciativy Taylora Otwell, s cílem poskytnout alternativu ke staršímu PHP frameworku jménem CodeIgniter. Důvodem byla neexistující podpora pro důležité funkcionality jako vestavěná autentizace uživatelů a správná autorizace. V červnu 2011 byla vydána první beta verze Laravelu a krátce poté, ve stejném měsíci, byl uveden na trh Laravel 1. Laravel je v době psaní této práce ve verzi 11 a obsahuje značné množství přídavných funkcí, jako je objektově relační mapování, podpora pro middleware, templatovací engine Blade a mnoho dalších. (W3Schools, c2009-2024)

3.4.5.1 Lumen

Lumen je mikro-framework vyvinutý tvůrcem Laravelu, Taylorem Otwellem, který nabízí menší, rychlejší a jednodušší alternativu k plnohodnotnému webovému aplikačnímu frameworku. Jeho hlavním zaměřením jsou mikroslužby a menší komponenty aplikací, které vyžadují rychlé a lehké řešení. Navzdory základní podobě s frameworkem Laravel má Lumen menší flexibilitu a konfigurovatelnost, což ho činí ideálním pro projekty, které potřebují rychlý backend bez zbytečných složitostí plnohodnotného frameworku. (Stauffer, 2024)

3.4.5.2 Eloquent ORM

Důležitou součástí frameworku Laravel pro práci s databází je Eloquent, jedná se o ORM, tedy Objektově relační mapování, což znamená, že umožňuje vývojářům pracovat s databázovými tabulkami a záznamy jako s objekty v PHP. Eloquent zjednodušuje práci s databází a umožňuje psát čistý a čitelný kód pro manipulaci s daty, přičemž poskytuje širokou škálu funkcí pro dotazování, vytváření, aktualizaci a mazání dat. Eloquent také poskytuje podporu pro definici vztahů mezi databázovými tabulkami a automatické provedení běžných operací, což výrazně zjednodušuje vývoj webových aplikací. (Laravel Holdings, c2011-2024)

3.4.5.3 Artisan

Další důležitou součástí Laravel frameworku je Artisan, jedná se o nástroj příkazového řádku, který umožňuje provádět různé úkoly spojené s vývojem a správou aplikace. Umožňuje generování kódu, tedy vytváření základních prvků aplikace, jako jsou kontrolery, modely, views, migrace databáze nebo middleware. Dále umožňuje správu databáze včetně spouštění databázových migrací, vytváření seederů pro naplnění databáze testovacími daty a generování SQL skriptů. Artisan také poskytuje nástroje ke konfiguraci, ladění, testování a nasazení aplikace. Artisan tedy zvyšuje produktivitu vývojářů a umožňuje jim efektivněji řešit běžné úkoly při práci na projektech v Laravelu. (Laravel Holdings, c2011-2024)

3.5 Frontend

Frontend je v rámci softwarového inženýrství a webového vývoje termín označující části aplikace, které interagují přímo s uživatelem. Jedná se o část aplikace, kterou uživatel vidí a s níž interaguje prostřednictvím uživatelského rozhraní. To zahrnuje veškeré viditelné a interaktivní prvky v prohlížeči nebo jiném uživatelském rozhraní, jako jsou například webové stránky, grafické prvky, formuláře, tlačítka a další interaktivní prvky. Frontend se obvykle skládá z HTML pro strukturu obsahu, CSS pro design a formátování a JavaScriptu pro interaktivní funkce a dynamický obsah. Je to protiklad backendu, který se stará o zpracování dat a logiku aplikace na serverové straně. (Lemonaki, 2024)

3.5.1 Webové Technologie

V této podkapitole budou krátce nastíněny základní technologie, na kterých stojí v podstatě všechny webové stránky.

3.5.1.1 HTML

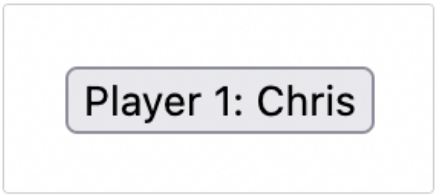
HTML (HyperText Markup Language) je základním stavebním kamenem internetu, který definuje strukturu a význam webového obsahu. Jedná se o standardizovaný značkovací jazyk a jeho hlavním účelem je označovat text, obrázky a další prvky pro zobrazení ve webovém prohlížeči pomocí značek, které se nazývají HTML elementy. Tyto elementy jsou součástí značkovacího jazyka HTML a zahrnují různé položky jako <head>, <title>, <body> a mnoho dalších. HTML umožňuje tvorbu odkazů mezi webovými stránkami a propojení obsahu, což je základní princip hypertextu. Přestože se HTML nezabývá prezentací ani funkcionalitou, je nezbytnou součástí tvorby webových stránek a je často kombinován s dalšími technologiemi, jako je CSS pro vzhled a JavaScript pro chování. (Mozilla, c1998–2024)

3.5.1.1.1 Krátká historie HTML

HTML byl vytvořen Sir Timem Berners-Lee koncem roku 1991, ale oficiálně nebyl vydán. Byl publikován v roce 1995 jako HTML 2.0. HTML 4.01 byl publikován koncem roku 1999 a byl hlavní verzí HTML. Současná verze HTML je verze pátá a byla publikována v roce 2012. Od svých prvních vydání se HTML velmi měnilo a vyvíjelo. Jedny

z nejdůležitějších vylepšení v nejnovější verzi je podpora multimediálního obsahu a zlepšená podpora formulářů. (W3Schools, c2009-2024)

```
<button type="button">Player 1: Chris</button>
```



Obrázek 8 - Příklad použití HTML (Mozilla, c1998–2024)

3.5.1.2 CSS

CSS („Cascading Style Sheets" neboli Kaskádové styly) je jazyk stylopisu používaný k definici prezentace dokumentů psaných v HTML nebo XML. Jeho hlavním účelem je popsat, jak mají být jednotlivé prvky webové stránky zobrazeny na různých zařízeních a médiích. CSS je standardizován organizací W3C a dříve existovaly verze jako CSS1, CSS2.1 a CSS3, ale nyní se všechny nové funkce začleňují do jednoho standardu bez číselného označení. Specifikace CSS se dále vyvíjí pomocí oddělených modulů, které mají vlastní verze. CSS umožňuje nastavovat vlastnosti jako barvy, velikosti písma, pozice prvků, rozložení stránky a mnoho dalšího pomocí selektorů a deklarací. (Mozilla, c1998–2024)

3.5.1.2.1 Krátká historie CSS

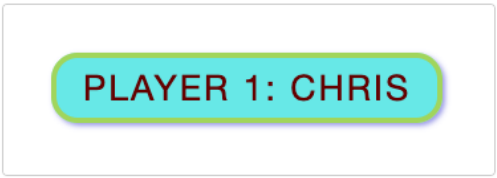
Vývoj CSS započal v roce 1994, kdy Håkon Wium Lie pracoval v CERNu a web začínal sloužit jako platforma pro elektronické publikování. Nedostatek možností stylování dokumentů na webu vedl k nutnosti vytvořit jazyk pro kaskádové styly. Přestože myšlenka oddělení struktury dokumentu od jeho vizuální prezentace existovala už od počátku HTML, žádný z předchozích prohlížečů neumožňoval uživatelům ovlivnit styl dokumentů dostatečně. V roce 1995 Håkon přišel s návrhem Kaskádových HTML stylů (CSS), což vedlo k založení pracovní skupiny W3C pro vývoj CSS. Po několika letech vyvíjení se CSS1 stává v prosinci 1996 oficiálním doporučeným standardem W3C. V následujících

letech se pak objevily další verze CSS, podpora CSS v prohlížečích jako Internet Explorer, Netscape Navigator a Opera, a rostoucí využití CSS pro tisk a e-knihy. (Bos, 2024)

3.5.1.2.2 SASS

Samotné CSS může být ještě rozšířeno o nadstavby jako je SASS (Syntactically Awesome Stylesheets) nebo SCSS (Sassy CSS). Oba jsou to pre-processory pro CSS, což znamená, že se jejich kód kompiluje do CSS a oba přinášejí pozměněnou syntaxi a nové funkce, umožňující psát CSS s větší flexibilitou a organizací. Mezi funkce, které SASS a SCSS přidávají do syntaxe CSS patří například proměnné, vnořené selektory, mixin-y, dědičnost a mnoho dalších. Tyto funkce usnadňují psaní a údržbu složitých stylů a pomáhají vytvářet responzivní, modulární a dobře strukturované styly pro webové stránky a aplikace. (Sass team, c2006-2024)

```css
button {
  font-family: "helvetica neue", helvetica, sans-serif;
  letter-spacing: 1px;
  text-transform: uppercase;
  border: 2px solid rgb(200 200 0 / 60%);
  background-color: rgb(0 217 217 / 60%);
  color: rgb(100 0 0 / 100%);
  box-shadow: 1px 1px 2px rgb(0 0 200 / 40%);
  border-radius: 10px;
  padding: 3px 10px;
  cursor: pointer;
}
```



Obrázek 9 - Příklad použití CSS (Mozilla, c1998–2024)

3.5.1.3 Javascript

JavaScript (JS) je lehký interpretovaný programovací jazyk s funkcemi první třídy, používaný jak na webových stránkách, tak i mimo ně, v prostředích jako Node.js nebo Apache CouchDB. Jedná se o dynamický jazyk s prototypově orientovanou, multi-paradigmatickou povahou, podporující různé programovací styly včetně objektově orientovaného, imperativního a deklarativního (funkcionálního) programování. Mezi jeho

dynamické schopnosti patří například vytváření objektů za běhu, proměnlivé seznamy parametrů a introspekce objektů. Standardy pro JavaScript jsou stanoveny v ECMAScript Language Specification (ECMA-262) a ECMAScript Internationalization API specification (ECMA-402). (Mozilla, c1998–2024)

3.5.1.3.1 Krátká historie Javascriptu

JavaScript vznikl jako reakce na potřebu validace vstupů a dynamických a interaktivních webových stránek. V roce 1995 byl vyvinut Brendanem Eichem ve společnosti Netscape Communications jako doplněk k prohlížeči Netscape Navigator. Původně nazýván Mocha a později LiveScript, byl nakonec pojmenován JavaScript. S jeho úspěchem a následným uvedením do standardu ECMAScript v roce 1997 se stal hlavním jazykem pro programování interaktivity webových stránek. Jazyk nyní vyvíjí Mozilla Foundation a Ecma International. (International JavaScript Institute, c2015-2022)

```
const button = document.querySelector("button");

button.addEventListener("click", updateName);

function updateName() {
  const name = prompt("Enter a new name");
  button.textContent = `Player 1: ${name}`;
}
```

Obrázek 10 - Příklad použití JavaScriptového kódu (Mozilla, c1998–2024)

3.5.2 Vue.js

Vue je JavaScriptový framework zaměřený na tvorbu uživatelských rozhraní. Jeho základem jsou standardní technologie webu, tedy HTML, CSS a JavaScript. Framework poskytuje deklarativní a komponentově orientovaný programovací model, který značně usnadňuje vývoj.

Dva nejdůležitější aspekty Vue jsou Deklarativní renderování a reaktivita. Deklarativní renderování rozšiřuje standardní HTML o šablonovou syntaxi umožňující deklarativní

popis výstupu HTML na základě JavaScriptového kódu. Reaktivita znamená automatické sledování změn stavu v JavaScriptu na stránce a efektivní aktualizaci dokumentové struktury stránky, jakmile se tyto změny odehrávají.

Vue je frameworkem a ekosystémem, který pokrývá většinu běžných potřeb ve frontendovém vývoji. Je navržen tak, aby byl flexibilní a inkrementálně přizpůsobitelný podle konkrétních potřeb. Může být využíván různými způsoby, jako například k vylepšení statického HTML bez nutnosti kompilace, k vložení na jakoukoli stránku jako webová komponenta, k vytvoření jednostránkové aplikace (SPA neboli Single-Page Application), k implementaci fullstacku či serverově renderované (SSR neboli Server-Side Rendering) aplikaci, či k použití v kontextu Jamstacku nebo statické generace webových stránek (SSG neboli Static Site Generating).

Většina projektů využívajících Vue a jeho nástroje pro sestavování zpracovává komponenty Vue pomocí speciálního formátu souborů nazývaného „Jednosouborová komponenta“ (SFC neboli Single-File Component). Tento formát zahrnuje logiku komponenty (JavaScript), šablonu (HTML) a styly (CSS) do jediného souboru, což usnadňuje práci a udržuje kód strukturovaný a přehledný.

(Vue.js, c2014-2024)

3.5.3 Quasar

Quasar je open-source framework založený na Vue.js, který nabízí rozsáhlé možnosti pro rychlé vytváření responzivních stránek a aplikací. Tento framework je licencován MIT licencí a umožňuje vytvářet aplikace v několika různých podobách, včetně jednostránkových aplikací (SPA), aplikací se serverovým vykreslováním (SSR), progresivních webových aplikací (PWA), prohlížečových rozšíření (Browser Extension), mobilních aplikací pro platformy jako Android a iOS pomocí Cordova nebo Capacitor, a také desktopových aplikací pomocí Electronu.

Jednou z hlavních výhod Quasaru tedy je, že je možné napsat kód jednou a poté jej nasadit jako webovou stránku, mobilní aplikaci a/nebo Electronovou aplikaci. To znamená, že

vývojáři mohou vytvářet aplikace s jediným zdrojovým kódem, který může být nasazen na různé platformy, což vede k výraznému zrychlení vývoje aplikací. Quasar poskytuje pokročilé CLI (Command Line Interface neboli Rozhraní příkazové řádky) a nabízí širokou škálu webových komponent, které jsou navrženy s ohledem na výkon a responzivitu.

Při použití Quasaru není potřeba dalších obsáhlých knihoven, jako jsou Hammer.js, Moment.js nebo Bootstrap, protože Quasar obsahuje všechny potřebné nástroje interně a s malým footprintem. Quasar se zaměřuje na usnadnění vývoje tím, že poskytuje osvědčené postupy hned po instalaci a podporuje širokou škálu funkcí, včetně podpory pro rozšíření aplikací, plnou podporu RTL (right to left neboli Zprava doleva, jedná se o zobrazování textu pro některé jazyky), podporu pro postupnou migraci existujících projektů a automatizované testování.

Quasar také nabízí širokou škálu podporovaných platforem, včetně Google Chrome, Firefox, Edge, Safari, Opera, iOS, Android, MacOS, Linux a Windows. K dispozici je také více než 40 jazykových balíčků, které mohou být snadno přidány do projektu.

(PULSARDEV SRL, c2015-2024)

3.6 Git a GitHub

Při tvorbě softwaru je velmi důležité používat software na zálohování a verzování kódu. K těmto účelům se používá mimo jiné právě Git a GitHub. Následující podkapitoly je stručně popíší.

3.6.1 Git

Git je open source distribuovaný systém správy verzí (SCM neboli Source Code Management) vyvinutý Linusem Torvaldsem v roce 2005, který vyniká svým modelem větvení verzí kódu, což je zásadní funkce, která ho odlišuje od ostatních SCM. Tento model umožňuje vytváření a práci s více lokálními větvemi kódu, které mohou existovat nezávisle na sobě a umožňují rychlé vytváření, slučování a mazání různých vývojových linií. Díky tomu lze provádět operace jako je bezproblémové přepínání kontextu, správa

kódu podle rolí nebo vykonávat pracovní postupy založené na funkcích a experimentování s odstranitelnými větvemi kódu.

Git je také známý pro svou rychlost, protože většina operací se provádí lokálně, což mu dává výraznou výhodu oproti centralizovaným systémům, které neustále komunikují se vzdáleným serverem. Již od svého počátku byl navržen tak, aby efektivně zvládal velké repozitáře a je napsán v jazyce C, což snižuje nároky na běhové prostředí spojené s vyššími programovacími jazyky.

Další významnou vlastností Gitu je jeho distribuovaný charakter, který umožňuje každému uživateli mít plnou zálohu hlavního serveru. To znamená, že i když se používá centralizovaný pracovní postup, každý uživatel má vlastní kopii repozitáře a může ji použít k obnovení hlavního serveru v případě havárie nebo poškození.

Git také poskytuje "staging area" nebo "index", což je prostředek pro přípravu změn v commitu, což umožňuje oddělené zahrnutí změn do commitu a usnadňuje revizi a formátování commitů. Jedná se o užitečnou funkci pro vývojáře, kteří chtějí mít kontrolu nad tím, které změny budou součástí commitu. (Chacon, Straub, 2014)

3.6.1.1 Stručná historie Gitu

Git začal jako reakce na potřeby komunity kolem vývoje jádra Linuxu, která hledala alternativu po rozpadu vztahu s komerčním nástrojem BitKeeper v roce 2005. Hlavními cíli nového systému byla rychlost, jednoduchý design, podpora ne-lineárního vývoje a plně distribuované řešení. Od té doby se Git vyvinul v rychlý a efektivní verzovací systém vhodný pro práci s velkými projekty, jako je jádro Linuxu. (Chacon, Straub, 2014)

3.6.2 GitHub

GitHub je webová platforma a služba pro hostování softwarových projektů. GitHub je založen na verzovacím systému Git, který umožňuje sledovat změny v kódu a spravovat jej v centrálním repositáři, což usnadňuje spolupráci mezi vývojáři a umožňuje jim přispívat do společných projektů.

GitHub vychází z Gitu a poskytuje webové rozhraní a sociální platformu pro správu a sdílení projektů. Umožňuje uživatelům vytvářet repozitáře, což jsou místa pro uložení všech souborů souvisejících s určitým projektem. Dále umožňuje tzv. "forking", což je vytvoření kopie existujícího projektu za účelem jeho dalšího vývoje nezávisle na původním projektu. GitHub rovněž umožňuje vytváření "pull requests", což jsou žádosti o začlenění provedených změn zpět do původního projektu.

Díky sociální síti integrované v GitHubu mohou uživatelé diskutovat o změnách, přispívat k projektům a sledovat práci ostatních. To vede ke zlepšení komunikace a spolupráce mezi vývojáři a umožňuje projektům růst a inovovat. GitHub také poskytuje prostor pro sledování změn v projektech a vytváření záznamů změn, což usnadňuje správu a udržování projektů. (How-To Geek, 2024)

3.7 Akademická debata

Akademická debata je formální diskusní metoda, která rozvíjí schopnost kritického myšlení, argumentace a znalostí o kontroverzních tématech. Soutěžní debaty jsou řízeny psanými pravidly a probíhají formou argumentačního duelu mezi týmy, kteří obhajují nebo kritizují danou tezi. Tyto soutěže slouží jako prostředek setkávání mládeže bez ohledu na sociální status a podporují rozvoj jazykových dovedností a kultivovaného vyjadřování. V rámci soutěží se debatuje v českém nebo anglickém jazyce podle stanovených pravidel. Debaty mohou probíhat podle různých forem, jako je například Karl Popper Debate Program (KPDP). (Asociace debatních klubů, 2023)

Soutěže mají vzdělávací charakter a hodnotí se především účast, nikoliv pouze výsledek. Mezi nejvýznamnější soutěže v České republice patří Debatní pohár a Debatní liga, které organizují regionální debatní kluby nebo Asociace debatních klubů. Debatní pohár umožňuje volnou účast týmů, které si sami volí témata a partnery. Vítězem poháru se stává tým s nejvyšším počtem bodů získaných během soutěží. Naopak v Debatní lize se debatuje na vyhlášené oficiální teze, na které se utkávají týmy ve stabilních sestavách. Nejlepší týmy postupují do celostátního finále, kde soupeří o titul Mistra Debatní ligy. (Asociace debatních klubů, 2023)

3.7.1 Pravidla debaty formát Karl Popper

Formát debaty Karl Popper Debate Program (KPDP) je specifický typ strukturované debaty, který je založen na myšlenkách Karla Poppera, rakouského filozofa. V tomto formátu se dva týmy střetávají v diskusi o kontroverzním tématu. Každý tým má tři řečníky s pevně definovanou rolí. Formát KPDP má následující strukturu:

1. *Příprava*: U nepřipravených tezí mají týmy určitý čas na přípravu svých argumentů. Obvykle je to 30–60 minut.
2. *Řečnické časy*: Každý řečník má přidělený časový limit pro svou řeč. Typicky je to 6 minut pro prvního řečníka a druhého řečníka a 5 minut pro třetího řečníka. Křížový výslech trvá 3 minuty.
3. *Průběh debaty*:
    - *První řečníci* prezentují základní argumenty svého týmu a definují téma debaty.
    - *Druzí řečníci* rozvíjejí argumenty svého týmu a reagují na argumenty protistrany, nepřinášejí však již nové argumenty.
    - *Třetí řečníci* shrnují argumenty svého týmu, a prezentují poslední přesvědčivé body. Snaží se přesvědčit rozhodčího o vítezství jejich týmu.
4. *Křížový výslech*: Po každé řeči má druhý tým možnost klást otázky řečníkovi soupeřícího týmu. Toto je čas, kdy se týmy mohou vzájemně konfrontovat a vyvracet argumenty nebo osvětlovat nejasnosti.
5. *Posouzení*: Po skončení debaty rozhodčí zhodnotí výkony obou týmů a vybírá vítěze na základě kvality argumentů, logiky, reakcí na argumenty protistrany, stylu a celkové přesvědčivosti vystoupení.

(Asociace debatních klubů, 2021)

3.7.2 Asociace debatních klubů

Asociace debatních klubů, z.s. (ADK) je nevládní nezisková organizace, která se již třicet let zabývá vzděláváním dětí, mládeže a pedagogů prostřednictvím soutěžního akademického debatování. Její hlavní programy, Debatní liga a Debate League pro střední školy a Debatiáda pro základní školy, nabízejí mladým lidem unikátní vzdělávací

prostředí, které rozvíjí klíčové dovednosti potřebné pro úspěšné fungování v dnešní společnosti. Cílem ADK je propagovat debatní metodu vzdělávání a integrovat ji do školních osnov jako alternativu k tradičním metodám vzdělávání v České republice. (Asociace debatních klubů, 2023)

4 Vlastní práce

Cílem této práce jest navrhnout a naprogramovat webovou aplikaci pro rozhodčí debaty ve formátu Karl Popper. Tuto aplikaci autor nazval Balloter, neboť se zabývá primárně tvorbou ballotů, což je list rozhodčího se záznamem z debaty. Aplikace toho však umí mnohem víc. Podporuje přihlašování a správu uživatelských profilů a umožňuje zobrazovat, přidávat a spravovat jednotlivé teze a balloty. Aplikace je navíc dvojjazyčná a obsahuje i temný režim spolu s moderním uživatelským rozhraním.

Tato kapitola se soustředí na aplikování znalostí nabytých v teoretické části a popis samotné tvorby aplikace od sběru uživatelských požadavků až po samotné vytvoření a testování.

4.1 Přípravná fáze

Před započetím vyvíjení aplikace bylo nutné provést řádnou analýzu a rozvrhnout, co vše bude nutné udělat. Jako součástí přípravné analýzy se prováděly konzultace se zadavatelem, vypracovaly se funkční požadavky a vytvořily se patřičné diagramy.

Během konzultací se zadavatelem bylo vypracováno zadání se třemi etapami. V první etapě bylo nutné udělat formulář na tvorbu ballotů, který umí před vyplňovat některé údaje. V druhé nepovinné etapě se měla do formuláře zabudovat tabulka nazývaná BOTA neboli bodovací tabulka, ve formě nápovědy. Nakonec ve třetí dobrovolné etapě zabudovat do této nápovědy systém vypočítávání bodů podle kategoriích v tabulce.

4.1.1 Funkční požadavky

Na základě tohoto zadání byly vypracovány následující funkční požadavky:

- *Registrace a přihlašování:* Aplikace bude umožňovat přístup pouze přihlášeným uživatelům a bude umožňovat jejich registraci.
- *Offline funkcionalita:* Aplikace bude fungovat nezávisle na připojení na internetu.
- *Přehledné rozhraní:* Aplikace bude mít přehledné uživatelské rozhraní, může být podobné webu debatovani.cz

- *Možnost přepínání jazyků (čeština a angličtina)*: Aplikace bude podporovat přepínaní jazyků, a to konkrétně češtinu a angličtinu.
- *Export hotových ballotů do PDF:* Aplikace bude podporovat exportování hotových ballotů do dokumentu formátu PDF.
- *Funkce ukládání jména uživatele:* Aplikace si bude pamatovat jméno uživatele, aby ho mohla vkládat do ballotu automaticky.
- *Komplexní formulář pro vyplňování ballotů:* Nejdůležitější funkce celé aplikace, komplexní formulář, který bude přehledný a bude zjednodušovat vyplňování ballotů.
- *Funkce debatní časomíry ve formuláři:* Formulář pro vyplňování ballotů bude mít stopky pro všechny řeči a přípravné časy v debatě.
- *Automatické vkládání data a jména rozhodčího, řečníků a týmu do formuláře:* Formulář bude umět z databáze načíst všechna potřebná data a pomoci je rozhodčímu předvyplnit.
- *Kontrola počtu znaků v textových polích ve formuláři:* Formulář bude hlídat počet zapsaných znaků v osobních komentářích u řečníků a také v konečném rozhodnutí debaty a bude rozhodčího upozorňovat bude-li jich bude příliš málo.
- *Kontrola povolených hodnot u bodování ve formuláři:* Formulář bude hlídat, že zadané počty bodů náleží do rozmezí definovaného v pravidlech debaty.
- *Systém dopočítávání bodů za křížový výslech ve formuláři:* Formulář bude umět rozpoznat bodovací systém, který rozhodčí používá pro křížové otázky a pokud se jedná o zastaralý systém, bude umět zadané body matematicky převést na systém nový.
- *Implementace nápovědy z bodovací tabulky ve formuláři:* Formulář bude rozhodčímu nabízet možnost prohlédnout si relevantní nápovědu z bodovací tabulky podle toho, jaký typ bodování zrovna vyplňuje. (Například políčko pro body za styl bude nabízet zobrazení nápovědy bodování stylu)
- *Systém pro dopočítávání bodů podle nápovědy z bodovací tabulky ve formuláři:* Formulář bude umět dát v patřičné kategorii nápovědy rozhodčímu možnost vybrat bodové rozmezí podle výkonu řečníka v dané kategorii a podle toho co rozhodčí vybere bude aplikace umět dopočítat výsledné body.

Již v přípravné fázi bylo ale zřejmé, že aplikace nebude fungovat v režimu offline kvůli nutnosti přihlašování a vývoj se dále soustředil na to využít funkcí plnohodnotné webové aplikace, s tím, že pak bude v budoucnu možné vyvinout offline verzi aplikace pro desktopy s omezenými funkcemi. Aplikace byla vyvíjena s potenciálem jí poté sloučit s existujícími webovými službami, které ADK provozuje. Ukládání jména uživatele se proto změnilo na komplexnější uživatelský profil a přidala se funkcionalita ukládání hotových ballotů a tezí do databáze, což umožňuje v nich vyhledávat a zpětně je zobrazovat v aplikaci.

4.1.2 Persony

Jako jedna z důležitých částí analýzy zadání je si uvědomit ideální uživatele aplikace, která se bude vytvářet. Proto byly v rámci analýzy vytvořeny následující persony. Aplikace má sloužit v podstatě dvěma tipům uživatelů. Rozhodčím, kteří pomocí ní budou moci tvořit nové balloty a pohodlněji rozhodovat debaty a také debatérům, kteří se budou moci podívat na balloty z debat, kterých se účastnili v minulosti.

4.1.2.1 Persona Rozhodčí

Jméno: Kateřina Nováková

Věk: 32 let

Profese: Učitelka anglického jazyka na střední škole

Zájmy:

- *Pedagogika*: Kateřina miluje svou práci jako učitelka anglického jazyka a věnuje se neustálému vzdělávání v oblasti moderních výukových metod a technologií.
- *Debata*: Od studentských let se Kateřina aktivně účastnila debatních soutěží a stala se vášnivou zastánkyní soutěžního akademického debatování. Nyní se podílí na organizaci debatního klubu na své škole a pomáhá studentům rozvíjet dovednosti argumentace a kritického myšlení.

- *Cestování*: Kateřina ráda cestuje a objevuje nové kultury a zvyklosti. Během volna často navštěvuje různé destinace po celém světě, což jí poskytuje inspiraci pro výuku anglického jazyka a mezikulturního porozumění ve třídě.
- *Literatura*: Kateřina je nadšená čtenářka a miluje literaturu v anglickém jazyce. Často doporučuje svým studentům knihy k četbě a využívá literárních děl jako základ pro diskusi a kreativní psaní ve třídě.

Potřeby:

- *Moderní vzdělávací metody*: Kateřina hledá nové inspirace pro svou výuku a je otevřená novým výukovým metodám a technologiím, které mohou zlepšit zážitek ze vzdělávání pro své studenty.
- *Podpora debatního klubu*: Kateřina se snaží podporovat své studenty v jejich zájmu o debatování a hledá způsoby, jak jim poskytnout dostatek příležitostí k rozvoji jejich dovedností v argumentaci a veřejného vystupování.
- *Mezikulturní porozumění*: Kateřina se zajímá o mezikulturní výměnu a hledá způsoby, jak integrovat mezikulturní učení do své výuky anglického jazyka, aby pomohla svým studentům lépe porozumět světu kolem sebe.
- *Literární inspirace*: Kateřina hledá nové způsoby, jak integrovat literární díla do své výuky anglického jazyka a podnítit zájem svých studentů o čtení a literaturu.

Důvod užívání aplikace: Kateřina se v rámci svého zájmu o debatování angažuje v organizaci debatních turnajů a často na nich vystupuje jako rozhodčí. Proto by ocenila, kdyby mohla mít aplikaci, která jí usnadní vyplňovat její list rozhodčího a zároveň poslouží jako časomíra.

4.1.2.2 Persona Debatér

Jméno: Lukáš Novák

Věk: 17 let

Studium: Středoškolák na gymnáziu

Zájmy:

- *Debatování*: Lukáš se aktivně účastní debatních soutěží od svých raných středoškolských let a je vášnivým zastáncem akademické debaty. Má silné dovednosti v argumentaci, kritickém myšlení a veřejném vystupování.
- *Politika a společenské vědy*: Lukáš se zajímá o politické události, společenské otázky a mezinárodní vztahy. Sleduje aktuální události a rád diskutuje o politických tématech a výzvách.
- *Literatura a psaní*: Lukáš má rád četbu a psaní a často se zapojuje do literárních aktivit na své škole. Rád zkoumá různé literární žánry a autory a čte širokou škálu literárních děl.
- *Sport*: Lukáš je aktivní sportovec a rád se účastní různých sportovních aktivit, jako je fotbal a tenis. Sport mu pomáhá udržovat fyzickou kondici a poskytuje mu potřebné odreagování.

Potřeby:

- *Rozvoj dovedností v debatování*: Lukáš hledá příležitosti k tréninku a zdokonalování svých dovedností v debatování, včetně argumentace, výzkumu, rétoriky a týmové spolupráce.
- *Aktuální informace*: Lukáš potřebuje přístup k aktuálním informacím a zdrojům, které mu pomohou lépe porozumět debatovaným tématům a připravit se na turnaje.
- *Podpora týmu*: Lukáš hledá podporu od svých spoluhráčů a trenérů v debatním týmu, aby mohl lépe spolupracovat a dosáhnout úspěchu ve svých debatních vystoupeních a posunout se na vyšší příčky v debatní lize.
- *Vyvážení zájmů*: Lukáš potřebuje najít rovnováhu mezi debatováním, studiem, sportem a dalšími zájmy, aby si udržel celkový zdravý životní styl a dosáhl úspěchu ve všech oblastech svého života.

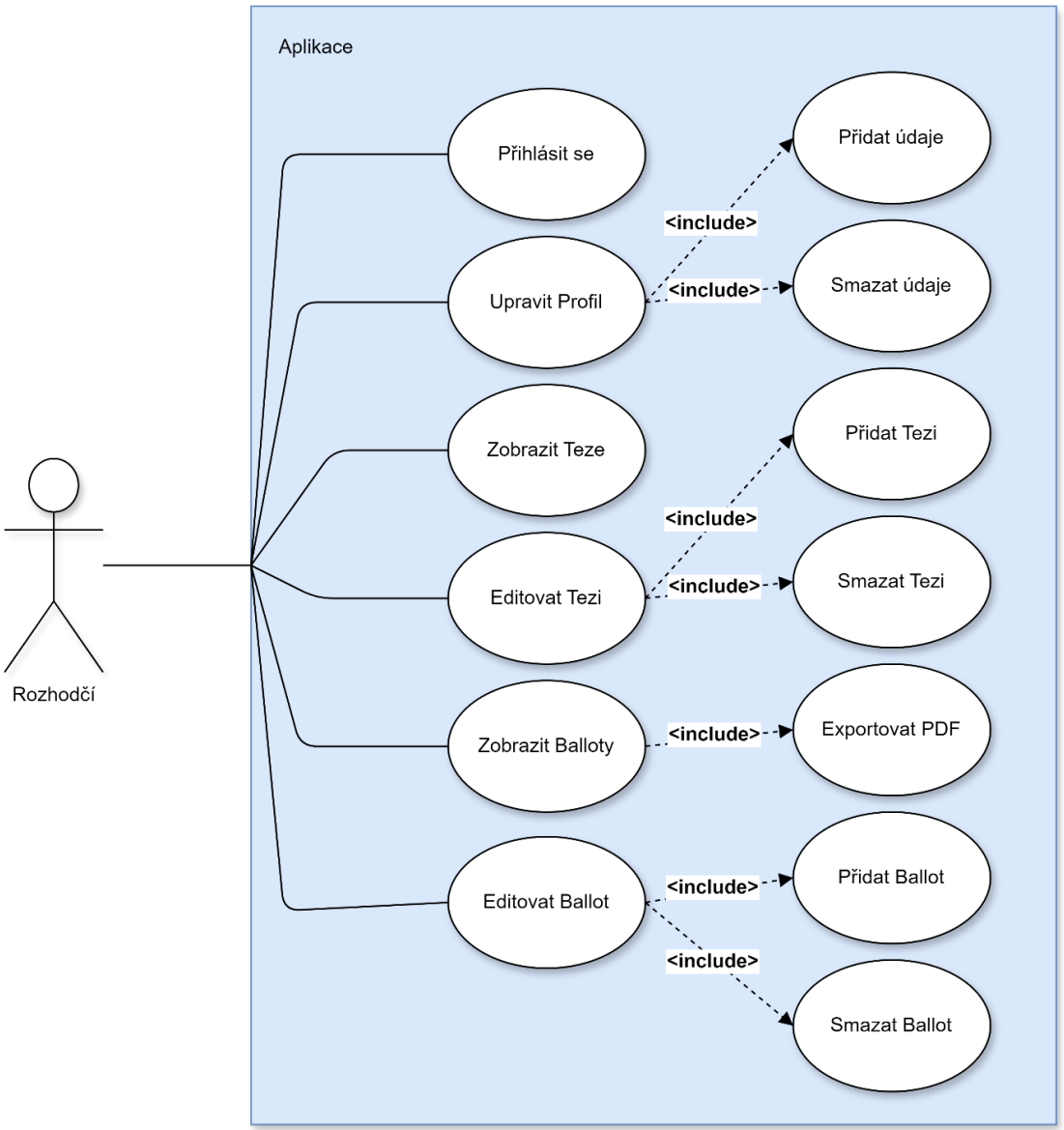
Důvod užívání aplikace: Lukáš je zapálený debatér a všechny svoje balloty z turnajů schraňuje, a dokonce si je i skenuje a ukládá do společného úložiště se svým týmem. Velmi by proto ocenil, kdyby měl přístup do aplikace, která by mu umožnila vidět všechny jeho absolvované debaty a balloty z nich na jednom místě.

4.1.3 Diagramy

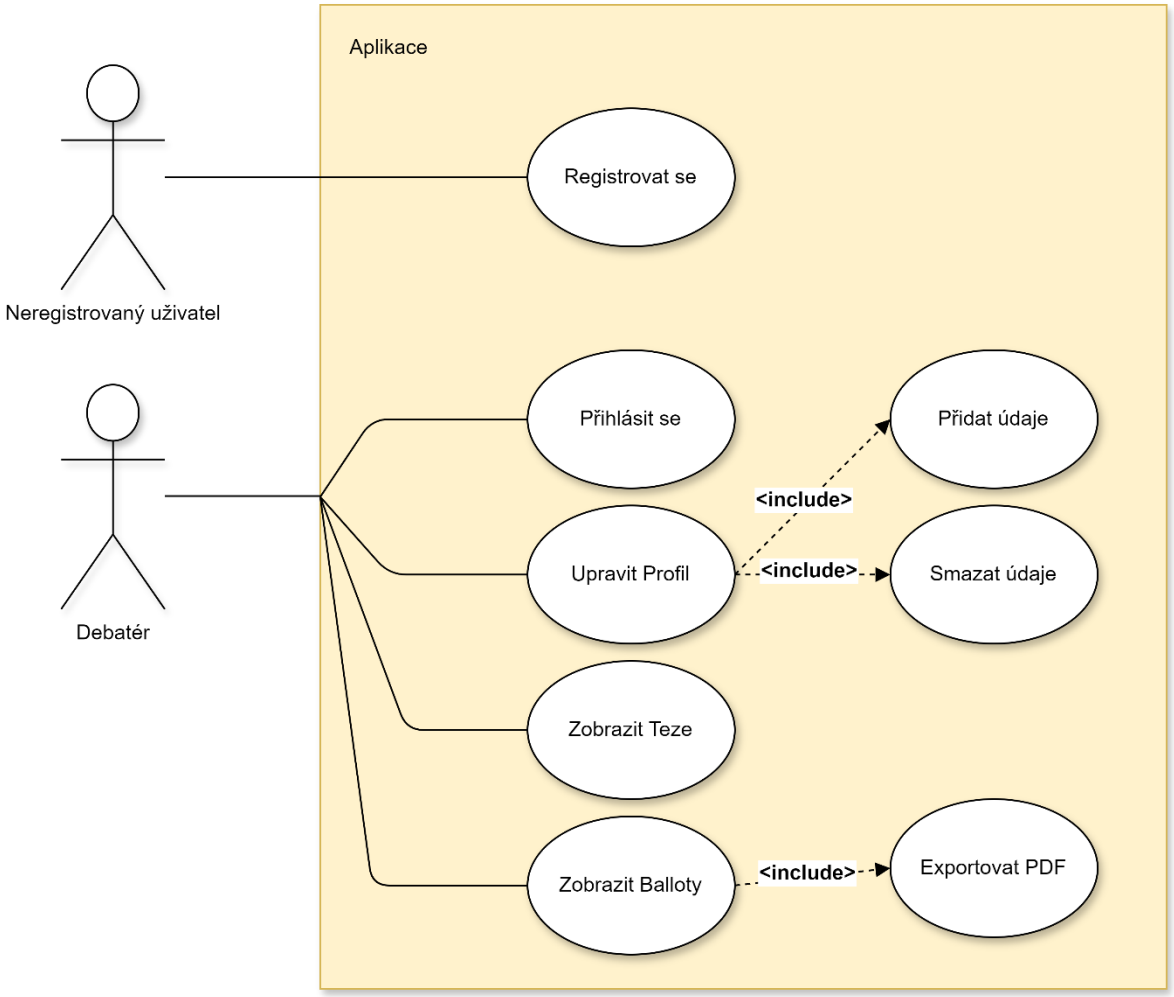
V této podkapitole budou popsány všechny diagramy vytvořené k ilustraci funkčnosti aplikace.

4.1.3.1 Diagram užití

První diagram je diagram užití z pohledu debatéra a z pohledu rozhodčího, pro větší přehlednost byl rozdělen na dva. Navíc je v diagramu z pohledu debatéra přidán i případ neregistrovaného uživatele.



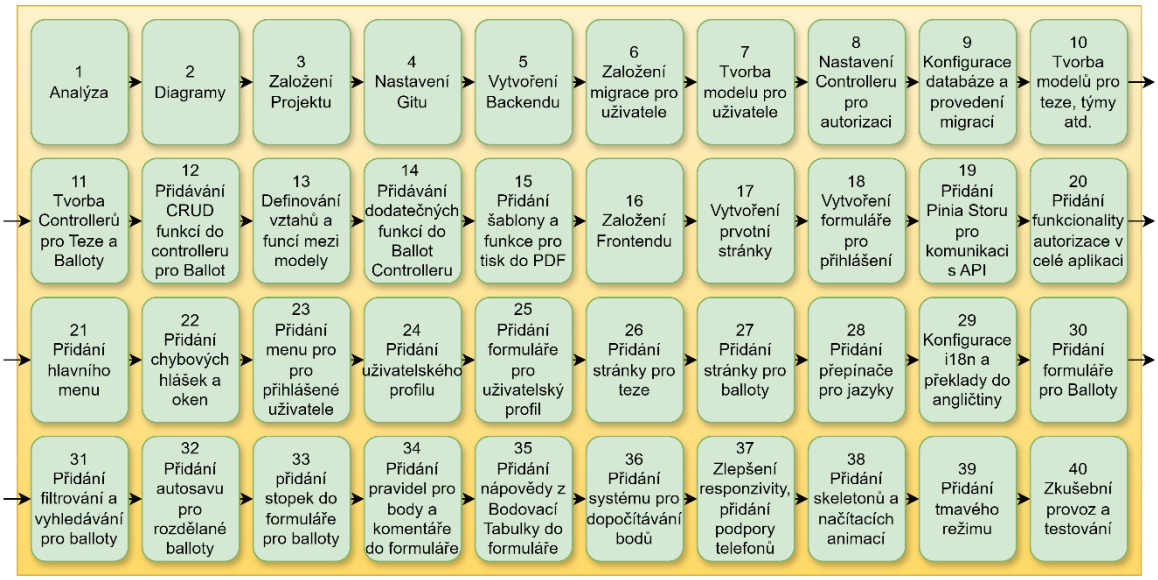
Obrázek 11 - Diagram Užití z pohledu rozhodčího



Obrázek 12 - Diagram Užití z pohledu debatéra

4.1.3.2 Workflow diagram

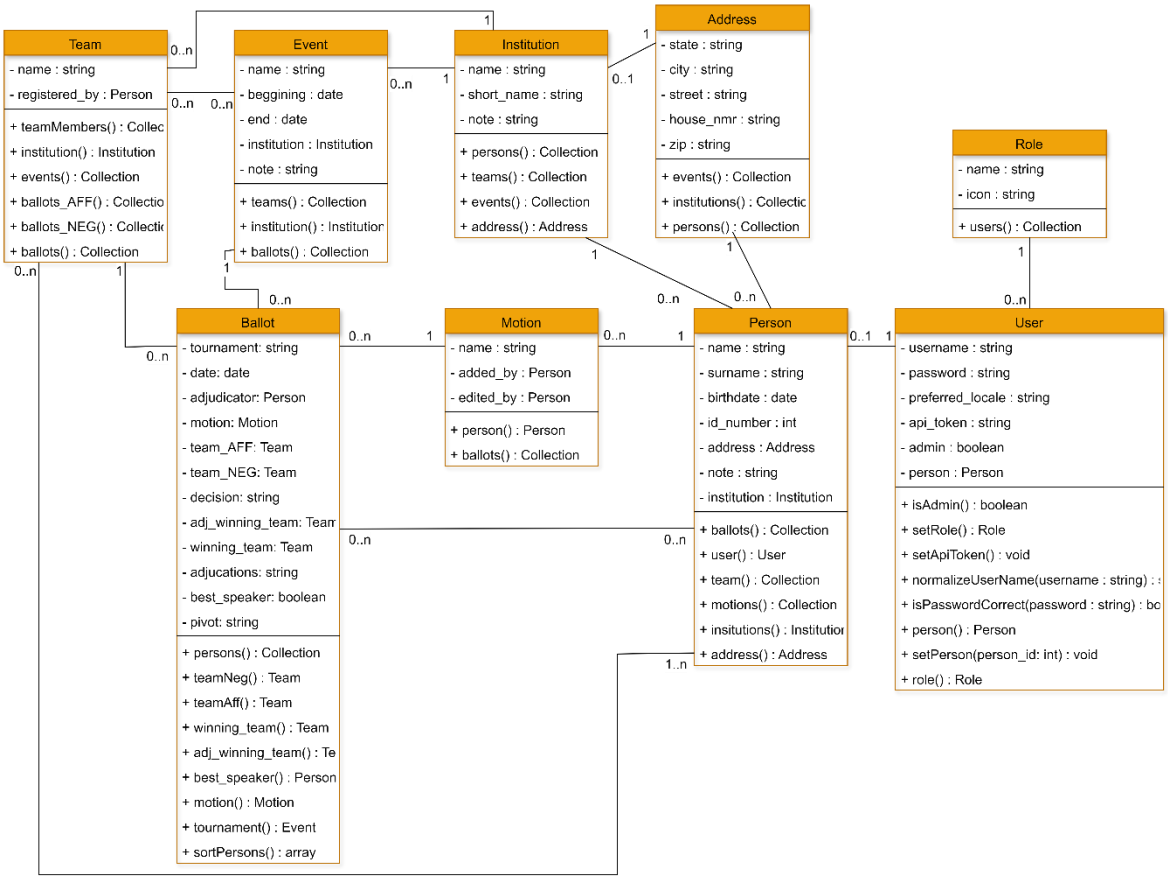
Jedná se o jeden z prvních diagramů, které se vytváří při analýze projektu. Zobrazují se v něm nutné kroky k implementaci aplikace spolu s časovou dotací a zdroji. Jelikož se ale jedná o menší projekt vyvíjený pouze autorem, diagram zobrazuje spíše posloupnost jednotlivých kroků, provedených při implementaci aplikace.



Obrázek 13 - Workflow Diagram

4.1.3.3 Diagram tříd

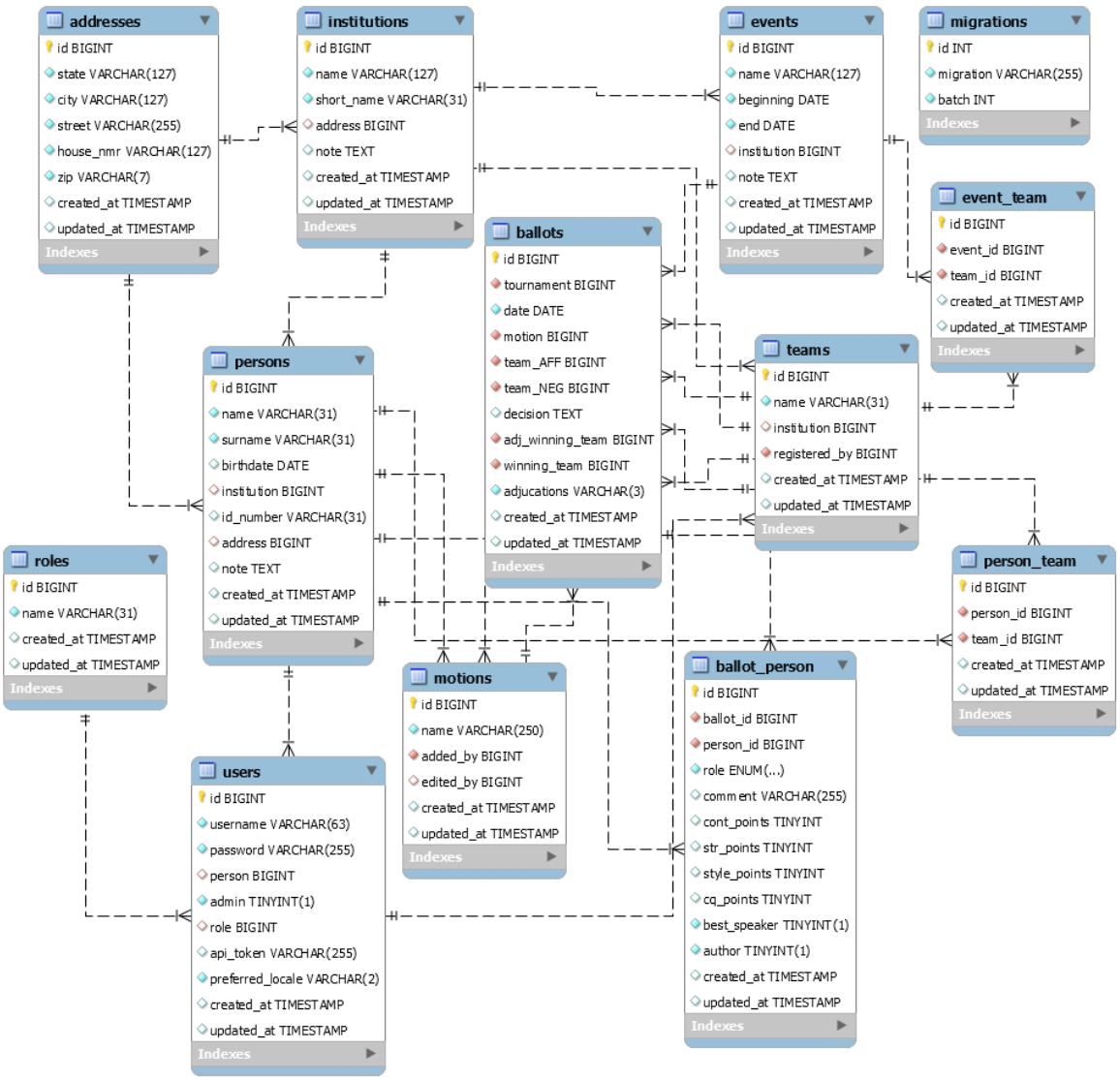
Následující diagram zobrazuje třídy nutné pro chod aplikace, spolu s jejich atributy. Zobrazují se pouze třídy na straně backendu neboť frontend má velmi specifickou strukturu, která neodpovídá klasickému objektovému programu. Kvůli tomu, jak ve frameworku Laravel fungují Modely může být také následující diagram lehce matoucí. Děje se tak kvůli tomu, že atributy nejsou nutně atributy, ale definice vlastností, do kterých se dají ukládat data zvenčí. Metody jsou zde pak použity k definování vztahů mezi modely, častokrát vrací ty modely, se kterými je daný model spojený a v kódu se nevolají jako klasické metody, ale tváří se jako proměnná.



Obrázek 14 - Diagram tříd

4.1.3.4 Databázový ER diagram

Nakonec je zde popsán diagram, který ilustruje strukturu databáze a rozložení jednotlivých entit, jejich atributy a jednotlivé vztahy mezi nimi. Je možné si povšimnout, že se odlišuje od diagramu tříd a je to tím, že Eloquent si občas některé vlastnosti databáze přizpůsobí vnitřním potřebám frameworku. Navíc jsou zde zobrazeny již i mezi-vazební tabulky, které se v objektové reprezentaci frameworky nedefinují explicitně.



Obrázek 15 - ER Diagram databáze

4.2 Vývoj aplikace

V této části vývoje se již přistoupilo k samotné implementaci. Jako vývojové prostředí bylo zvoleno Visual Studio Code od firmy Microsoft. Jako server byla zvolena aplikace Laragon, která poskytuje kompletní řešení spolu s webovým serverem Apache, databázovým serverem MySQL se SŘBD v podobě HeidiSQL. Dále byl pro testování API použit nástroj Postman a pro návrh diagramů bylo použito Draw.io spolu s MySQL workbench. Wireframy pro návrh frontendu byly poté vytvořeny ve webové apikaci wireframe.cc a prototypy byly vytvořeny v programu Figma. Všechny Tyto programy jsou volně k dispozici a zdarma k použití.

Dále byl vytvořen repozitář na webové stránce GitHub a byl spárován s lokální instalací Gitu pro lepší verzování a zálohování softwaru.

Ve vývoji softwaru jak pro backendovou část, tak i pro frontendovou část byl kladen velký důraz na dodržování všech postupů objektově orientovaného programování spolu s postupy SOLID.

Samotný vývoj pak probíhal v jisté formě prototypu, tedy autor provedl prvotní analýzu a sběr požadavků od zadavatele a poté vytvořil funkční prototyp, který otestoval a nadále opravoval a doplňoval o další funkce, podle potřeby. Komunikace se zadavatelem nebyla tak častá, jak by si auto býval byl přál, ale aplikace se i tak povedlo dovézt do zdárného konce.

4.2.1 Backend

Jako první krok se autor rozhodl naprogramovat backend aplikace, tedy postarat se o databázi a datovou logiku. Backend byl tvořen ve Frameworku Lumen a byl tvořen jako REST API.

4.2.1.1 Implementace modelů

Prvním krokem tvorby backendu bylo rozvržení databáze a následná tvorba potřebných modelů a migrací. Autor započal tuto etapu tvorbou uživatele a osoby, které jsou nezbytné

pro fungování zbytku aplikace včetně přihlašování. Následovala pak samotná implementace modelu pro ballot, jež ovšem ukázala, že bude nutné přidat mnoho dalších modelů, které budou mezi sebou propojeny. Tyto modely byly pak také navrženy tak, aby byly co nejvíce kompatibilní s existující architekturou webu ADK debatovani.cz a greybox.debatovani.cz

4.2.1.1.1 Model User

Model user je nejdůležitější model pro funkci registrace a přihlašování. Jedná se o uživatelský profil každého uživatele. Obsahuje jeho přihlašovací jméno a heslo, preferovaný jazyk a přiřazenou osobu, stejně jako API token, který je přidělován každému uživateli po přihlášení nebo informaci, zda je tento uživatel admin. Na tomto profilu bude demonstrována tvorba modelů ve frameworku Laravel. Na následujícím obrázku je možné si povšimnout deklaraci modelu a parametrů, se kterými operuje. Jedná se však o proměnné, do který se dají vkládat přímo hodnoty z requestu. Do hidden vlastností se potom přímo hodnota vkládat z requestu nedá, logika aplikace s ní však může operovat.

```php
<?php

namespace App\Models;

use Illuminate\Auth\Authenticatable;
use Illuminate\Contracts\Auth\Access\Authorizable as AuthorizableContract;
use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Laravel\Lumen\Auth\Authorizable;
use Illuminate\Support\Facades\Hash;

class User extends Model implements AuthenticatableContract, AuthorizableContract
{
    use Authenticatable, Authorizable, HasFactory;

    /**
     * The attributes that are mass assignable.
     *
     * @var string[]
     */
    protected $fillable = [
        'username', 'password', 'preferred_locale', 'api_token', 'admin', 'person',
    ];

    /**
     * The attributes excluded from the model's JSON form.
     *
     * @var string[]
     */
    protected $hidden = [
        'password', 'admin',
    ];
```

Obrázek 16 - Příklad deklarace modelu User

Na dalším screenshotu je pak možno si povšimnout deklarací metod pro model uživatele. Pro vnitřní objektovou logiku frameworku se tyto funkce používají pro definování vztahů mezi modely. Lze však samozřejmě zakládat i funkce potřebné pro definování chování jednotlivých modelů. V této části kódu je například možné si povšimnout funkce setApiToken, která uživateli přiřadí token po přihlášení, používá se šifra sha1 a jako šifrovaný údaj se bere aktuální čas. Další funkcí je pak normalizeUserName, která převádí uživatelské jméno do malých písmen. A Nakonec je zde funkce isPasswordCorrect, která kontroluje, zda zadané heslo je stejné jako uložené heslo. Uložené heslo je samozřejmě zašifrované, takže i zadaná hodnota se nejdřív musí zašifrovat a porovnávají se pak tyto šifry. Funkce person se potom ve zbytku programu chová jako proměnná a slouží k definování vztahu mezi uživatelem a osobou. Zde se například definuje vztah, kdy uživatel náleží jedné osobě a tato funkce poté vrací tu osobu, ke které je tento uživatel přiřazený.

```php
/**
 * Sets Api Token parameter and saves to the DB
 */
public function setApiToken()
{
    $this->api_token = sha1($this->id.time());
    $this->save();
}

/**
 * Converts username to lowercase
 * @param string $username
 * @return string
 */
static public function normalizeUserName(string $username): string
{
    return strtolower($username);
}

/**
 * Check password
 * @param $password
 * @return bool
 */
public function isPasswordCorrect($password): bool
{
    return Hash::check($password, $this->password);
}

public function person()
{
    return $this->belongsTo(Person::class, 'person');
}
```

Obrázek 17 - Příklad deklarování funkcí v modelu User

4.2.1.1.2 Model Person

Dalším důležitým modelem, je model pevně svázaný s userem. Jedná se o osobu, tedy reálnou reprezentaci daného uživatele. Tento model uchovává jméno, příjmení, datum narození, adresu, identifikační číslo, instituci (nejčastěji školu uživatele) a textovou poznámku. Všechny tyto parametry si uživatel nastavuje sám a jsou zde pro snadnější vyplňování potřebných údajů do ballotu, pro zjištění konkrétní struktury tohoto a dalších modelů je možné nahlédnout do diagramu tříd výše.

4.2.1.1.3 Model Ballot

Toto je ten nejdůležitější model, který jest centrem celé aplikace. Uchovává všechny potřebné údaje o dané debatě. Má v sobě atributy jako tournament, date, adjudicator, motion, team_AFF, team_NEG, decision, adj_winning_team, winning_team, adjucations, best_speaker a pivot. Všechny tyto valstnosti jsou vyplňovány z formuláře pro tvorbu ballotů a jedná se o hodnoty jako turnaj, na jakém daná debata probíhá, aktuální datum, osobu rozhodčího, debatovanou tezi, team afirmace a team negace a tak dále. Vlastnost pivot je pak objekt, který reprezentuje mezi-vazební tabulku mezi ballotem a dalšími osobami, kteří jsou účastníci debaty. Jsou zde ukládáni jednotliví řečníci, jejich role a jednotlivé počty bodů. Tato tabulka je poté definována v migracích pro databázi, ale nemá žádnou explicitně definovanou objektovou reprezentaci.

4.2.1.1.4 Model Address

Toto je pomocný model, který existuje primárně pro to, aby se mohla dodržet první normálová forma v databázi. Jedná se o objekt, který ukládá adresy rozdělené na vlastnosti stát, město, ulice, číslo popisné a PSČ. Tento model byl vytvořen primárně proto, že je nutné ukládat adresy pro jednotlivé uživatele nebo instituce nebo třeba turnaje. A může se stát, že se někde nějaká adresa vyskytne dvakrát, a proto je lepší mít adresu zvlášť, aby se v databázi nevyskytovaly duplicitní data.

4.2.1.1.5 Model Event

Tento model je vcelku jednoduchým pomocným modelem. Existuje hlavně proto, aby se mohly balloty třídit podle toho, z jakého jsou turnaje. Tento model tedy reprezentuje jednotlivé debatní turnaje a akce a uchovává data jako název, datum začátku a konce, instituci a poznámku. Je dále spojen s týmy, neboť jednotlivé týmy se registrují na turnaje

a je pak možné ve formuláři ballotu dát rozhodčímu na výběr vyplnit pouze ty týmy, které se účastní aktuálního turnaje.

4.2.1.1.6 Model Institution

Tento model reprezentuje jednotlivé instituce (většinou školy) na kterých probíhají debatní turnaje nebo z kterých pochází jednotlivé debatní týmy. Instituce má jako vlastnosti název a zkratku a poznámku, jedná se tedy o relativně jednoduchý pomocný model. Je spojena s osobami, které na takové instituci mohou studovat, s týmy, které tam mohou býti registrovány, eventy, které se zde mohou konat a s adresou, neboť každá Instituce musí mít svoji adresu.

4.2.1.1.7 Model Motion

Tento model je pro funkci tvorby ballotů velmi důležitý. Jedná se o tezi, na kterou se v dané debatě debatuje. Teze mají také svojí vlastní stránku a controller, kde je možno je přidávat, mazat a upravovat, aby se poté mohly zvolit ve formuláři pro balloty ze seznamu. Teze uchovává parametry o názvu a o tom, kdo danou tezi přidal a editoval. Spojené jsou pak s osobou, aby se mohl rozlišit autor a nakonec s ballotem, neboť každý ballot musí mít informace o tezi, o které daná debata byla.

4.2.1.1.8 Model Role

Jedná se o pomocný model, který existuje primárně kvůli případnému začlenění do stránek debatovani.cz protože tam se role používá pro rozlišení práv. V této aplikaci se rozlišuje mezi debatérem a rozhodčím (případně pozorovatel nebo trenér, ti mají ale stejná práva jako debatér). Role určuje, zda uživatel může přidávat nebo upravovat a mazat teze a balloty. Uchovává údaje o názvu role a ikoně. A je spojena pouze s uživatelem, ke kterému náleží.

4.2.1.1.9 Model Team

Team je model, který sám o sobě nemá mnoho údajů, je ale propojen s mnoha dalšími modely. Jedná se o reprezentaci jednotlivých týmu v debatách. Parametry tohoto modelu jsou pouze název a osoba, která tým registrovala. Team je pak propojen s osobami, kteří jsou členi týmu a existuje zde metoda, která tyto členy vrací. Dále je zde metoda na propojení s institucí, která určuje, na jaké instituci byl tým registrován a s Eventem,

kterého se tým účastní. Další metody jsou zde pak proto, aby se mohly nalézt balloty daného týmu, ale nakonec nejsou ve výsledné implementaci využity.

4.2.1.2 Implementace migrací

Další etapou programování backendu ve frameworku Laravel byla tvorba migrací, které jsou kód pro objektově relační mapovací vrstvu Eloquent. Ta vezme příkazy napsané v migraci v PHP a převede je na relační tabulky a atributy v připojené databázi. V zásadě se zde definují všechny potřebné vlastnosti, které by se definovaly i přímo v SQL při manuální tvorbě tabulek v SŘBD, ale nemusí se psát žádný SQL kód. V následujícím obrázku bude uveden příklad tvorby jednodušší migrace pro model Person. Je možno si zde všimnout funkcí up a down, přičemž funkce up definuje všechny kroky, které se mají provézt při tvorbě tabulky a funkce down naopak uvádí všechny kroky, které se mají provézt při jejím mazání. Je to zde pro to, že když se nějakou migrací upravují již existující tabulky, je důležité uvést databázi do stavu ve kterém byla před provedením migrace, pokud bude nutné efekt migrace smazat. Dělá se to proto, že migrace se vždy provádějí všechny a to tak, jak jdou za sebou.

Dále je možné si ve funkcích migrace povšimnout užití třídy Schema, která reprezentuje danou databáze a volají se její funkce create, table nebo drop. Funkce create vytváří novou tabulku, která se tam poté definuje jako vstupní parametr. Definují se zde jednotlivé atributy, jejich datový typ a jiné omezení a vlastnosti. Funkce table poté upravuje již existující tabulky, buď se v této funkci přidávají nové atributy nebo definují cizí klíče pomocí vlastnosti foreign nebo se tyto klíče nebo atributy také mažou pomocí vlastnosti dopForeign a dropColumn. Funkce dropIfExists pak maže celou tabulku, pokud již existuje. Tento příkaz zde je pro případ, kdyby se migrace spouštěla pro již existující databázi.

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('persons', function (Blueprint $table) {
            $table->id();
            $table->string('name', 31);
            $table->string('surname', 31);
            $table->date('birthdate')->nullable();
            $table->string('id_number', 31)->nullable();
            $table->string('street', 255)->nullable();
            $table->string('city', 127)->nullable();
            $table->string('zip', 7)->nullable();
            $table->string('school', 255)->nullable();
            $table->text('note')->nullable();
            $table->timestamps();
        });
        Schema::table('users', function($table) {
            $table->unsignedBigInteger('person')->after('password')->nullable();
            $table->foreign('person')->references('id')->on('persons');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::table('users', function($table) {
            $table->dropForeign(['person']);
            $table->dropColumn('person');
        });
        Schema::dropIfExists('persons');
    }
};
```

Obrázek 18 - Příklad implementace Migrace pro model User

4.2.1.3 Implementace controllerů

Poslední etapou tvorby backendu potom jest tvorba controllerů, které pak obsahují veškerou logiku, která nakládá s jednotlivými modely a plní je daty nebo nad nimi provádí další operace.

4.2.1.3.1 AuthController

Jedná se o controller, který zodpovědný za veškerou autorizaci, implementuje funkce login, logout, register, delete a returnUser. Jména funkcí hovoří sami za sebe, jedná se obecně controller, který obstarává přihlašování a registraci uživatelů, stejně jako jejich

mazání, odhlašování a mazání účtu. Na tomto controlleru bude demonstrováno, jak vypadá deklarace funkce ve frameworku Laravel. Jedná se o funkci login, která bere jako vstupní parametr request, což je request s daty pro přihlášení, které odeslal klient. Data z requestu se vezmou a zavolá se na ně funkce validate, která ověří správný formát těchto dat podle zadaných parametrů, tedy to, že přihlašovací jméno i heslo jsou povinné údaje a že přihlašovací jméno má mít podobu emailu. Dále se do proměnné user nahraje instance uživatele, která se hledá podle normalizovaného uživatelského jméno z requestu. Dále se ověřuje, jestli tento uživatel existuje a zdali je heslo korektní. Pokud tomu tak není a uživatele se podle zadaného uživatelského jména nepodařilo najít nebo pokud se zadané heslo neshoduje s heslem uloženým v dané instanci modelu User, aplikace vrátí response se stavovým kódem 401, tedy „Unauthorized“ s chybovou hláškou „Invalid Credentials“. Pokud je zadané uživatelské jméno i heslo správné, v bloku try se zavolají uživatelovy funkce pro přiřazení korektní role a přidělení přihlašovacího tokenu. Nakonec se vrátí response s kódem 200, tedy „OK“ a daty onoho uživatele. Do hlavičky response se pak vloží přiřazený přihlašovací token v klíčí authorization. Pokud by v průběhu přihlašování došlo k chybě, aplikace vrátí response s kódem 500, tedy „Internal server error“ a s konkrétním zněním dané chyby.

```php
public function login(Request $request)
{
    $this->validate($request, [
        'username' => 'required|email',
        'password' => 'required'
    ]);
    $username = User::normalizeUserName($request->input('username'));
    $user = User::where('username', $username)->first();
    if (empty($user))
    {
        return response()->json(['message' => 'invalidCredentials'], 401);
    }
    if (!$user->isPasswordCorrect($request->input('password')))
    {
        return response()->json(['message' => 'invalidCredentials'], 401);
    }
    try {
        $user->setApiToken();
        $user->setRole(); // TODO: vyřešit elegantněji
        $user->id_token = $user->api_token;
        // Needs key in header called Authorization with the value of the token
        return response()->json($user, 200)
            ->header('Authorization', 'Bearer '.$user->api_token)
            ->header('Access-Control-Expose-Headers', 'Authorization');
    } catch (\Illuminate\Database\QueryException $e) {
        return response()->json(['message' => $e->getMessage()], 500);
    }
}
```

Obrázek 19 - Příklad deklarování funkce v AuthControlleru

4.2.1.3.2 BallotController

Jedná se o jeden z nejdůležitějších controllerů aplikace. Obstarává tvorbu, editaci a mazání ballotů. Také může vracet jeden konkrétní ballot anebo seznam všech. Má funkce createPDF pro tvorbu PDF z ballotu, poté funkce create, update, delete pro tvorbu, přidání a mazání ballotů, pak funkce showOne a showAll, které vrací jeden ballot nebo všechny spolu s funkcí showMine, která vrací všechny balloty, s kterými je propojený aktuálně přihlášený uživatel. Další funkcí je potom ballotDataFill, která pomáhá nahrát všechny potřebná data z databáze a vyplnit jimi ballot, který se vrací klientovi. Dalšími pomocnými funkcemi pak jsou returnTeams a returnTournaments, které vrací všechny týmy a turnaje (eventy) pro potřeby dat k filtrování ballotů a tezí a také pro data, z kterých může rozhodčí vybírat při vyplňování ballotu.

4.2.1.3.3 MotionController

Tento controller je vcelku jednoduchým controllerem, který obstarává stránku pro přidávání, editování a mazání tezí. Jedná se tedy o jednoduchý controller s CRUD funkcemi. Obsahuje funkce create, update, delete a showOne a showAll. Princip je stejný jako u controlleru pro balloty. Create tezi přidává, update jí edituje, a delete maže. ShowOne potom vrací jednu konkrétní tezi a showAll vrací všechny teze.

4.2.1.3.4 PersonController

Poslední controller obsluhuje data osoby. Je zde hlavně kvůli možnosti editace uživatelského profilu a jedná se o další vcelku jednoduchý controller s CRUD funkcemi. Tentokrát jsou zde funkce create, update a delete pro přidání, editování a mazání dat osoby, a navíc ještě funkce returnInstitutions, která vrací všechny instituce, aby si z nich mohl uživatel vybrat, když specifikuje, z jaké pochází školy ve formuláři úpravy profilu.

4.2.1.4 Definování URL cest

Důležitou součástí zprovoznění funkčního API je definovat URL adresu, kterou je nutné zadat, aby se provedla kýžená operace. Framework Laravel má proto interní funkcionalitu, které se říká router, která umožňuje definovat různé URL adresy a přiřadit je k příslušným controllerům a jejich funkcím. Zadání patřičné adresy URL má pak za výsledek zavolání

přiřazené funkce. V následujícím obrázku jsou uvedeny všechny definované URL cesty, které aplikace používá. Je zde vždy uvedena html metoda get, post, put nebo delete, kterou cesta očekává a poté samotné znění adresy spolu s controllerem a jeho funkcí, oddělenou zavináčem (@). Tyto cesty jsou také zahrnuty ve skupině, která jim všem nařizuje mít prefix „api“, tedy každá URL adresa musí začínat slovem api.

```
$router->group(['prefix' => 'api'], function () use ($router) {
    $router->post('login', ['uses' => 'AuthController@login']);
    $router->get('logout', ['uses' => 'AuthController@logout']);
    $router->post('register', ['uses' => 'AuthController@register']);
    #$router->get('user/{id}', ['uses' => 'AuthController@showOne']);
    $router->get('user', ['uses' => 'AuthController@returnUser']);
    $router->delete('user/{id}', ['uses' => 'AuthController@delete']);

    $router->post('ballot', ['uses' => 'BallotController@create']);
    $router->get('ballot/pdf/{id}', ['uses' => 'BallotController@createPDF']);
    $router->get('ballots', ['uses' => 'BallotController@showAll']);
    $router->get('ballot', ['uses' => 'BallotController@showMine']);
    $router->get('ballot/{id}', ['uses' => 'BallotController@showOne']);
    $router->delete('ballot/{id}', ['uses' => 'BallotController@delete']);
    $router->put('ballot/{id}', ['uses' => 'BallotController@update']);

    $router->get('teams', ['uses' => 'BallotController@returnTeams']);
    $router->get('tournaments', ['uses' => 'BallotController@returnTournaments']);

    $router->post('motion', ['uses' => 'MotionController@create']);
    $router->get('motion', ['uses' => 'MotionController@showAll']);
    $router->get('motion/{id}', ['uses' => 'MotionController@showOne']);
    $router->delete('motion/{id}', ['uses' => 'MotionController@delete']);
    $router->put('motion/{id}', ['uses' => 'MotionController@update']);

    $router->post('person', ['uses' => 'PersonController@create']);
    $router->delete('person/{id}', ['uses' => 'PersonController@delete']);
    $router->put('person/{id}', ['uses' => 'PersonController@update']);
    $router->get('institutions', ['uses' => 'PersonController@returnInstitutions']);
});
```

Obrázek 20 - Definice všech URL cest pro backend

4.2.1.5 Implementace exportu PDF

Poslední věcí, kterou je důležité na backendu zmínit je samotná implementace tisku PDF souborů z dat daného ballotu. Je toho docíleno pomocí pluginu DomPDF, který slouží k tvorbě PDF souborů z html kódu. Tento plugin je použit ve funkci createPDF v BallotControlleru. Tato funkce si připraví konkrétní ballot a naplní ho daty, jako by ho vracela klientovi ve funkci showOne, poté tyto data ale předá pluginu DomPDF ve formě pole. Tento plugin poté těmito daty zaplní připravenou html šablonu a poté z ní vytvoří PDF, které pak pošle klientovi jako stáhnutelný soubor. Tato šablona je vytvořena pomocí templatovacího enginu Laravelu, nazvaným Blade, který umožňuje vytvářet html šablony a poté je plnit PHP kódem, který se při překladu mění na html kód, čímž dělá stránku

responzivní. Většinou jsou tyto šablony použity, pokud se pomocí frameworku vytváří monolitické aplikace bez odděleného frontendu. Tato aplikace však slouží pouze jako API a má pouze jednu šablonu, která se používá právě k účelu generování PDF. V této šabloně je poté definovaná struktura ballot dokumentu v html a pomocí PHP kódu jsou na správná místa dosazeny všechny potřebné hodnoty konkrétního ballotu. Pomocí CSS je definovaný potřebný font a jazyk.

4.2.2 Frontend

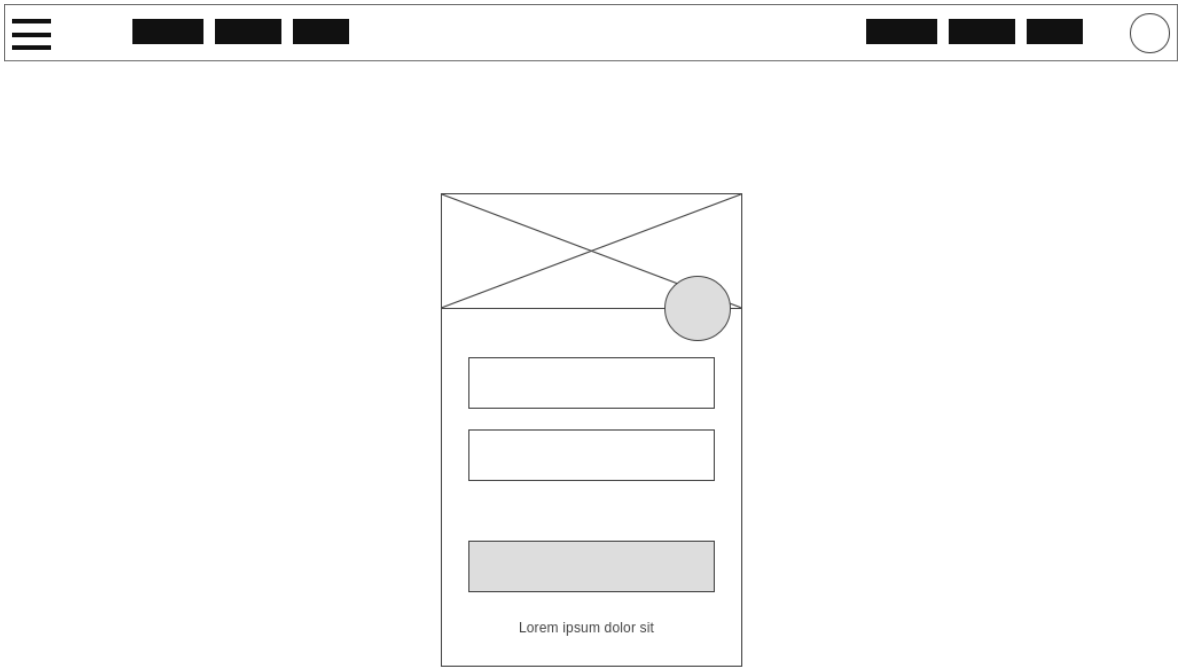
Poté co byl backend jako prerekvizita pro funkci frontendu hotov, bylo možné započít jeho implementaci. K tvorbě frontendu byl použit framework Quasar ve standartním módu. Jeho součástí byly moduly Router pro definování URL cest na frontendu, Pinia pro definování storů (jakýchsi globálních objektů s vlastnostmi a funkcemi, které se vkládají kdekoliv v kódu jsou potřeba) a i18n pro vícejazyčné překlady stránek.

4.2.2.1 Analýza a tvorba wireframů

Samotná Implementace pak začíná opět fází analýzy. Ta byla v případě tvorby frontendu ovšem celkem krátká, neboť se aplikace měla inspirovat existujícími stránkami debatovani.cz. Základní rozložení ovládacích prvků bylo tedy užito podobně. Byly však modernizovány jiné elementy uživatelského rozhraní. Jinak bylo rozhodnuto, že aplikace bude obsahovat stránky pro Domovskou obrazovku, Přihlašování, Uživatelský profil, Teze a Balloty s tím, že Uživatelský profil, a hlavně Balloty budou mít také svůj formulář.

4.2.2.1.1 Wireframe Přihlašovací obrazovky

V následujícím obrázku je možné vidět návrh přihlašovací obrazovky spolu se základním hlavním navigačním panelem. Jedná se o vcelku standartní přihlašovací formulář s políčky pro zadání přihlašovacího jména a hesla a s jedním tlačítkem.



Obrázek 21 - Wireframe přihlašovací obrazovky

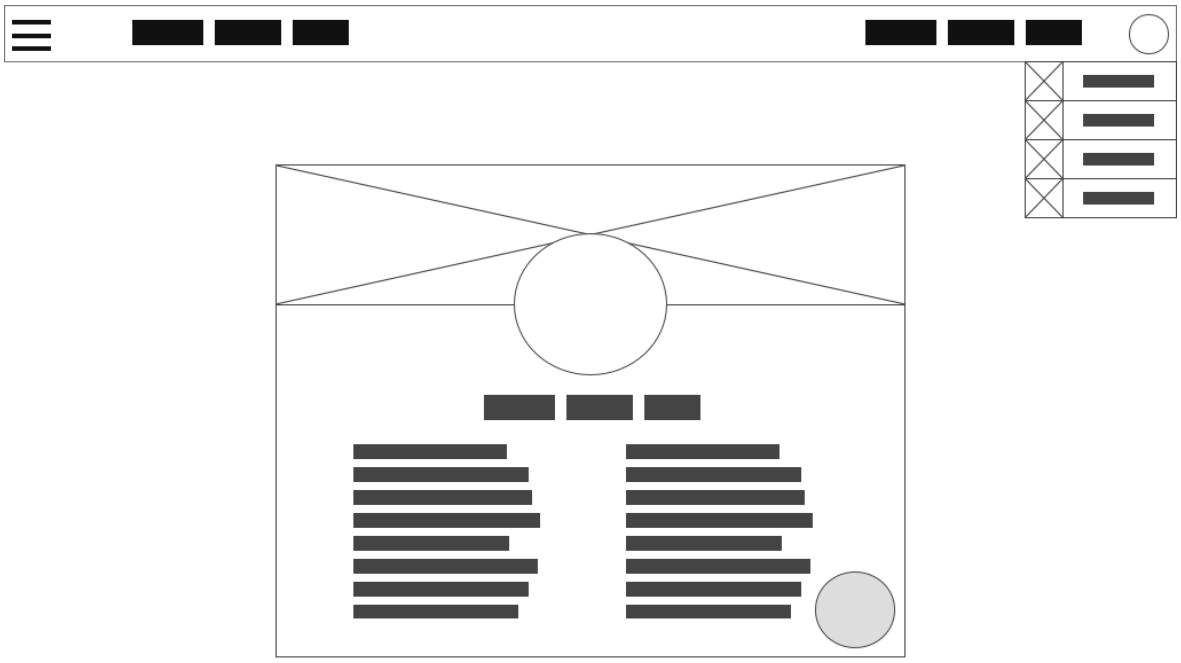
4.2.2.1.2 Wireframe domácí obrazovky

Následuje obrázek s návrhem domovské obrazovky spolu s otevřeným postranním panelem. V této fázi vývoje ještě nebylo přesně určeno, co má na domovské stránce být. Předběžně se předpokládala nějaká informace nebo rychlá navigace na další části stránky.

Obrázek 22 - Wireframe domovské obrazovky

4.2.2.1.3 Wireframe uživatelského profilu

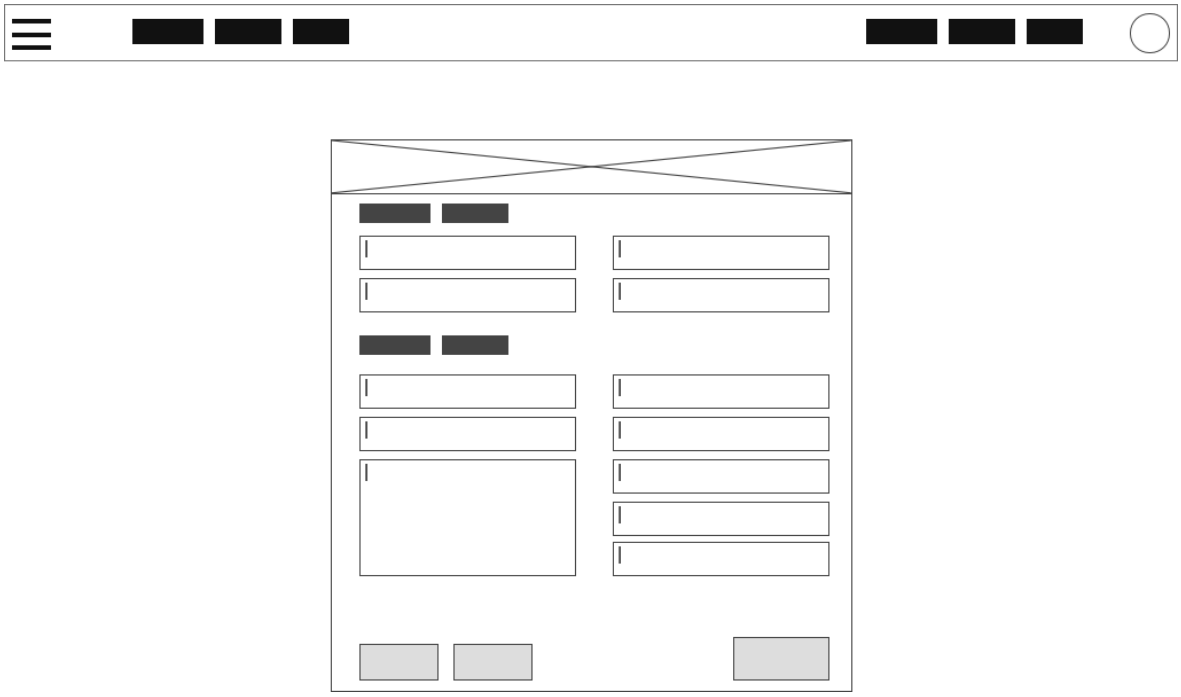
Dále byl v rámci analýzy vytvořen wireframe pro uživatelský profil. Jedná se obrazovku, která má jednoduše prezentovat informace o profilu a umožnit přecházení na formulář pro jeho editování. Na wireframu je také zobrazeno menší menu pro rychlou navigaci pro přihlášeného uživatele.



Obrázek 23 - Wireframe uživatelského profilu

4.2.2.1.4 Wireframe formuláře uživatelského profilu

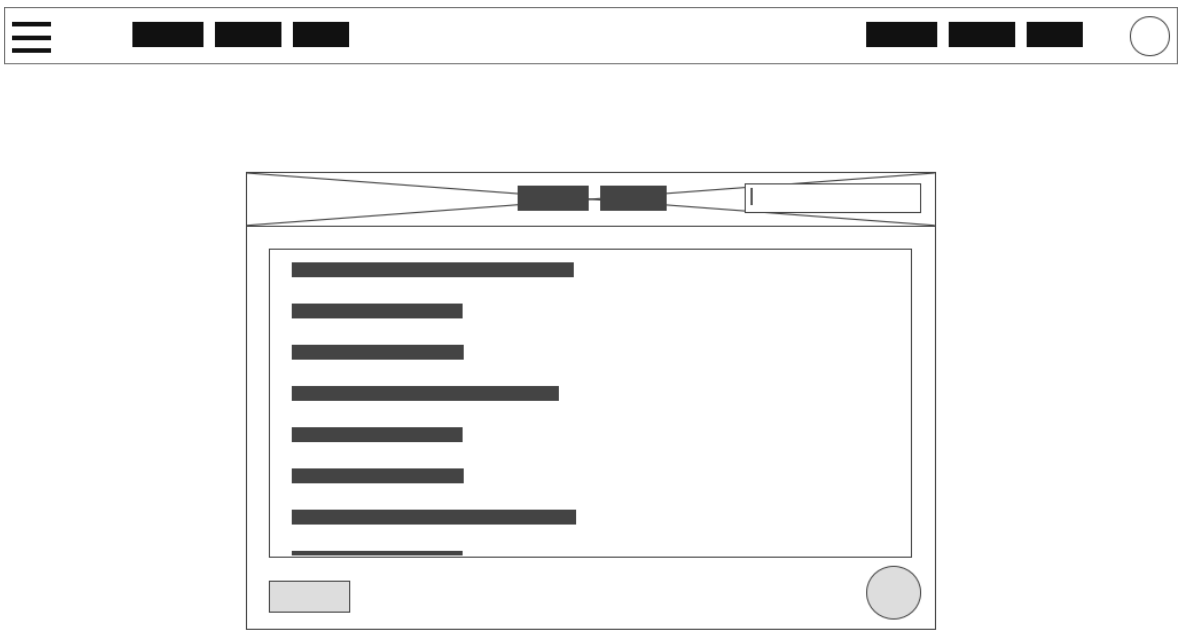
Ke stránce uživatelského profilu patří i formulář, v kterém může být tento profil editován. Jedná se o jednoduchý formulář s přehledně rozdělenými elementy.



Obrázek 24 - Wireframe formuláře uživatelského profilu

4.2.2.1.5 Wireframe stránky pro teze

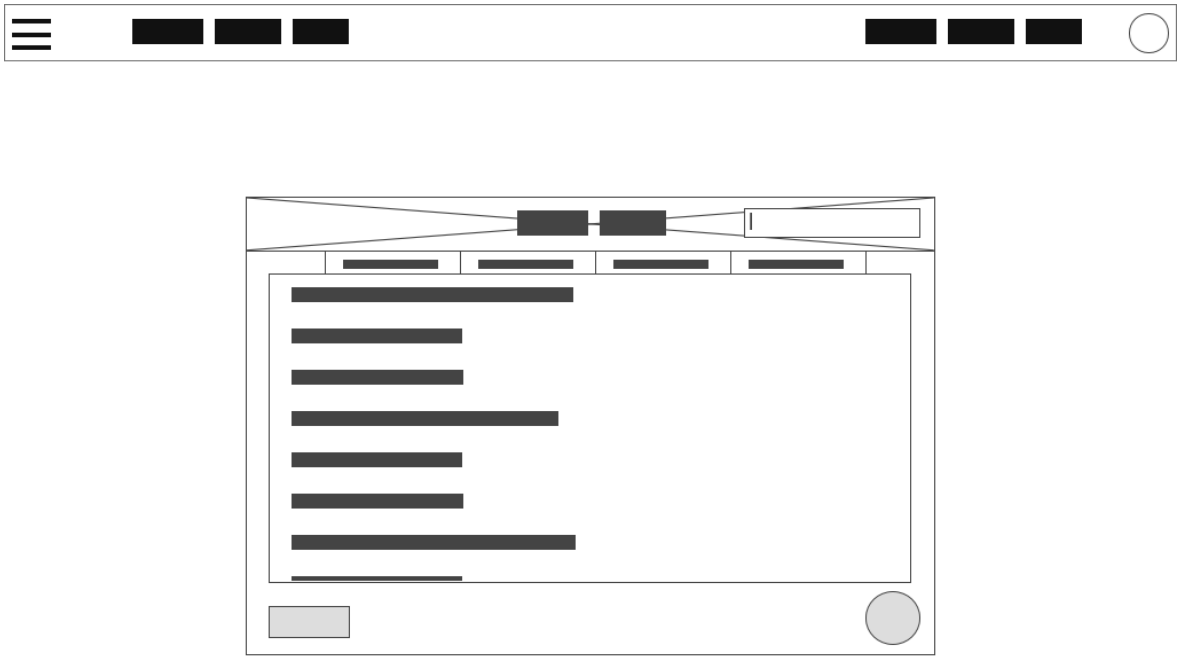
Další důležitou stránkou je stránka, která zobrazuje jednotlivé teze. Tato stránka nemá formulář pro editaci, neboť teze obsahuje pouze jednu vlastnost, a to znění samotné teze. Proto se přidávání nových tezí a upravování stávajících řeší pomocí vyskakovacích oken. Stránka by měla umožňovat vyhledávání tezí a poskytovat jejich přehledný seznam.



Obrázek 25 - Wireframe stránky pro teze

4.2.2.1.6 Wireframe stránky pro balloty

Asi jednou z nejpodstatnějších stránek této webové aplikace je potom stránka na zobrazování jednotlivých ballotů. Tato stránka je svojí funcionalitou totožná stránce na zobrazování tezí, až na to, že poskytuje více možností filtrování ballotů.



Obrázek 26 - Wireframe stránky pro balloty

4.2.2.1.7 Wireframe formuláře pro balloty

Nakonec je vyobrazen wireframe, který zobrazuje samotný formulář na tvorbu ballotů. Jedná se o nejdůležitější stránku aplikace a je to formulář, který poskytuje přehledné rozložení všech elementů a seskupení jednotlivých řečníků do organizovaných skupin.

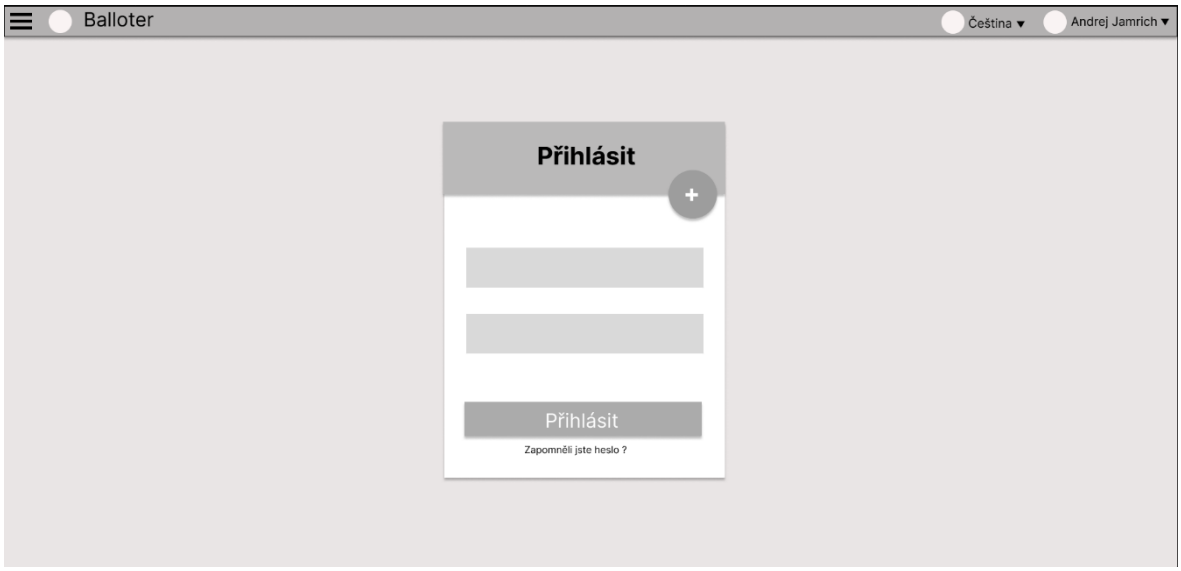
Obrázek 27 – Wireframe formuláře pro balloty

4.2.2.2 Tvorba prototypů

Po ucelené představě o tom, jaké stránky se budou muset udělat byl započat konkrétní návrh designu, který se poté bude programovat. V této podkapitole budou představeny prototypy, vytvořené pro jednotlivé stránky webové aplikace.

4.2.2.2.1 Prototyp přihlašovací obrazovky

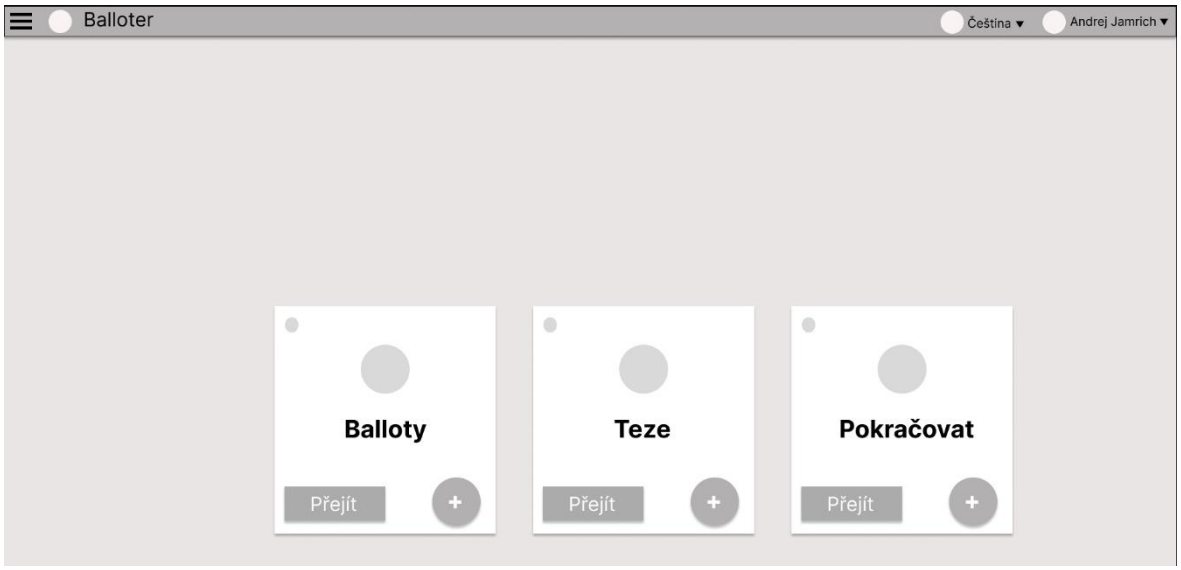
Přihlašování nabylo konkrétnější podoby s prvkem kulatého tlačítka, který se nese napříč celou aplikací.



Obrázek 28 - Prototyp přihlašovací obrazovky

4.2.2.2.2 Prototyp domovské obrazovky

V této fázi vývoje se již rozhodlo, že na domovské obrazovce se budou nacházet karty pro rychlou navigaci na ostatní stránky aplikace.



Obrázek 29 - Prototyp domovské obrazovky

4.2.2.2.3 Prototyp uživatelského profilu

Uživatelský profil nedošel výraznější evoluce, bylo však rozhodnuto, že se nebude používat obrázek pro jednotlivé uživatele, neboť to není důležité pro fungování aplikace.

Obrázek 30 - Prototyp uživatelského profilu

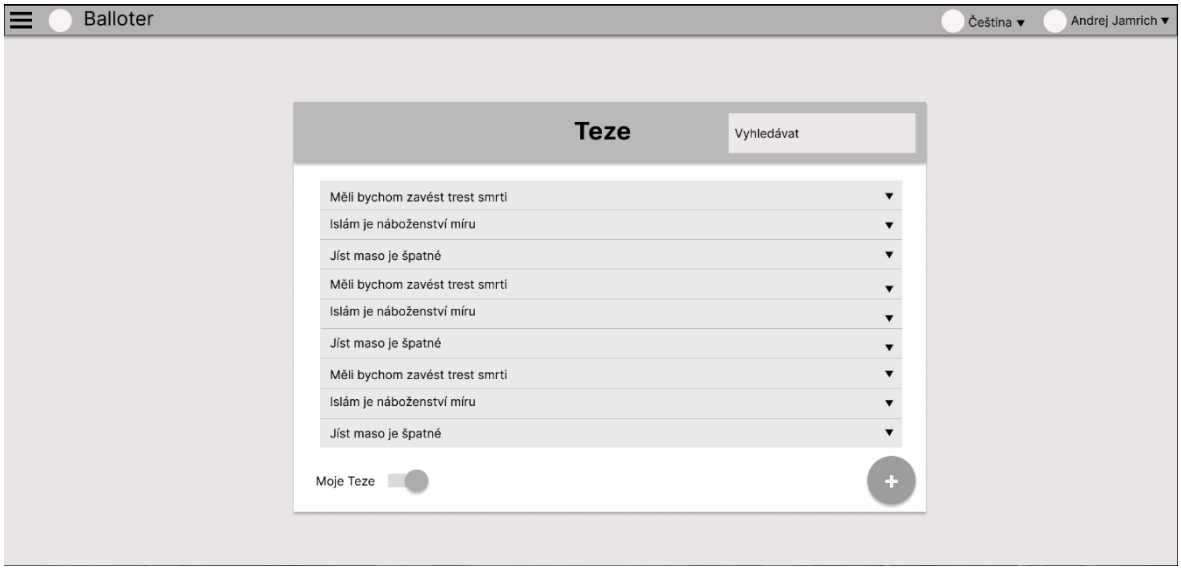
4.2.2.2.4 Prototyp formuláře uživatelského profilu

U formuláře se vybral estetický vzhled polí pro vyplňování s malým textem nápovědy navrchu, místo klasického placeholderu.

Obrázek 31 - Prototyp formuláře uživatelského profilu

4.2.2.2.5 Prototyp stránky pro teze

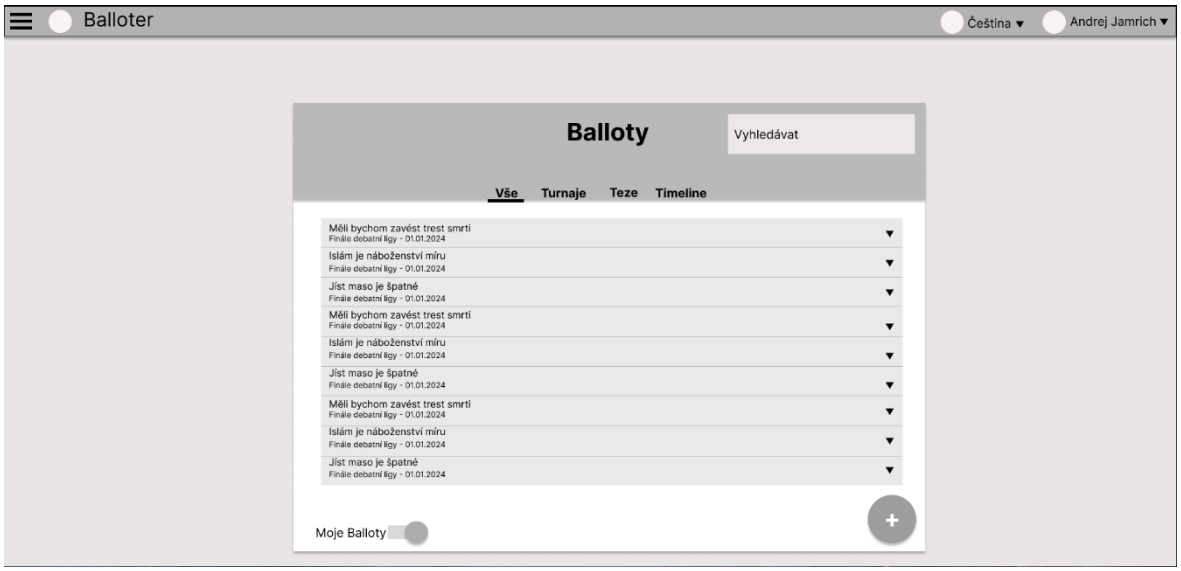
Bylo rozhodnuto, že se použije seznam položek, které jdou rozbalit pro zobrazení více informací.



Obrázek 32 - Prototyp stránky pro teze

4.2.2.2.6 Prototyp stránky pro balloty

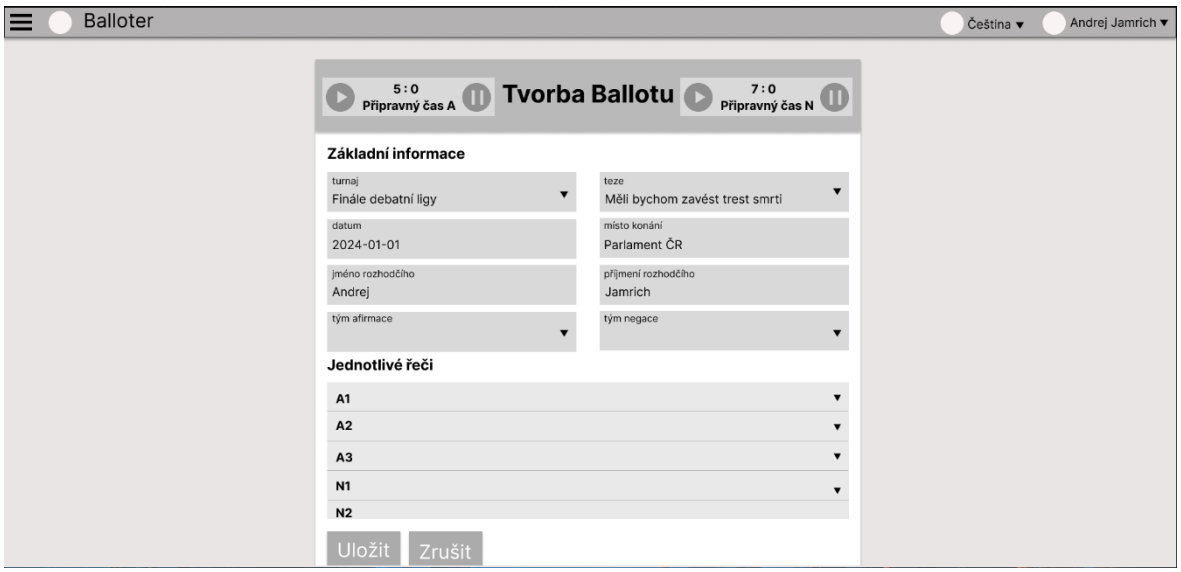
K filtrování ballotů se autor rozhodl použít záložky a seznam položek obohatit o více informací.



Obrázek 33 - Prototyp stránky pro balloty

4.2.2.2.7 Prototyp formuláře pro balloty

Stejný seznam položek byl využit i pro skrytí elementů pro jednotlivé řečníky, aby byl formulář přehlednější. Také se určila konkrétní podoba stopek.



Obrázek 34 - Prototyp formuláře pro balloty

4.2.2.3 Programování stránek

V následujícím kroku již bylo možné přistoupit k tvorbě jednotlivých stránek v rámci frameworku Quasar. Ten využívá funkce frameworku Vue.js, takže jednotlivé stránky jsou definované jako soubory s příponou „.vue“ . Bylo takto vytvořeno osm stránek, přesně tak, jak určily prototypy, plus jedna stránka pro chybu 404, když se nepodaří nalézt vyhledávanou stránku. K tomu byla také vytvořena stránka s kódem pro hlavní navigační panel a menu, kterou dědí každá další stránka, aby se tento panel nemusel neustále přidávat do každé stránky.

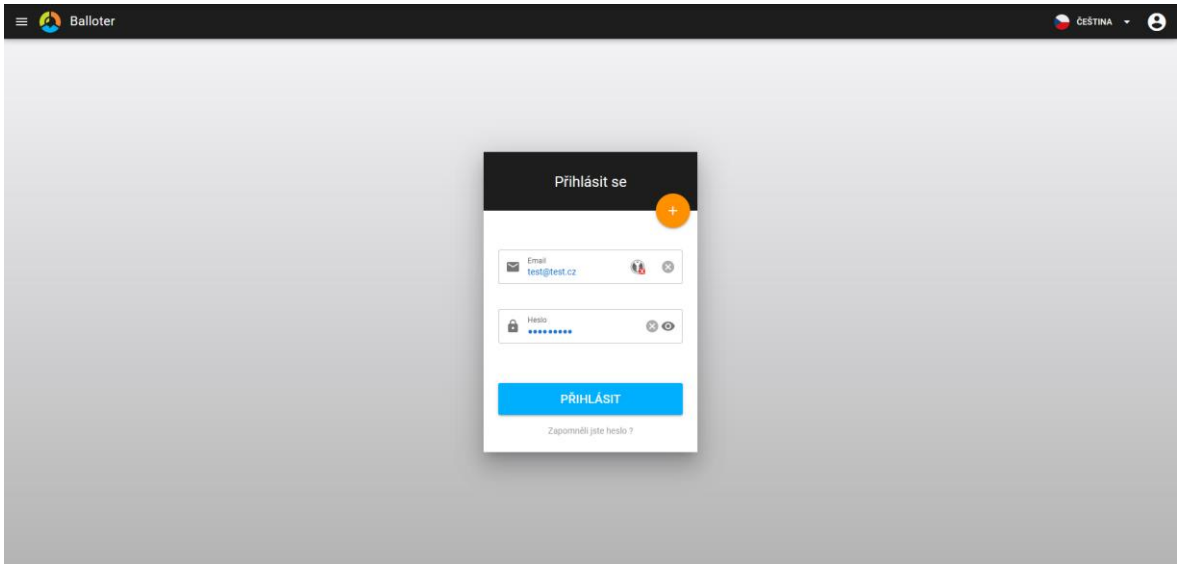
K tomu bylo vytvořeno také 12 komponent, jedná se o jednotlivé elementy, které se napříč aplikací opakují anebo se jedná o tak rozsáhlý kód, že je dobré je dát do svého vlastního souboru a tedy objektu, aby jedna stránka neměla příliš mnoho řádků kódu a nebyla nepřehledná. Místo toho může importovat a používat tyto komponenty, které onen kód obsahují.

Jako hlavní barvy pro aplikaci byly pak zvoleny barvy podobné logu ADK, tedy světle modrá, světle zelená a oranžová. Jako další barva pro zvýraznění prvků v UI byla zvolena černá a tmavě šedá, po vzoru webu debatovani.cz.

Dále je nutno zmínit využití pluginu axios, který se používá ke komunikaci s API. Je nastaven, aby komunikoval s vytvořeným API backendem a veškerý kód, který s ním komunikuje je vložen to Pinia Storu. Celkově jsou zde dva Pinia Story, jeden obsahuje veškerou logiku pro komunikaci s API ohledně autorizace a je používán vlastně na každé stránce pro kontrolu, že je uživatel přihlášen. Další použitý Pinia Store je zde pro balloty a teze, tento store obsluhuje nahrávání všech potřebných informací pro zobrazování a vyplňování ballotů a tezí a zároveň hotové balloty ukládá do backendu. Rozpracované balloty a také přihlašovací token jsou uloženy do paměti prohlížeče, aby v případě obnovení stránky, zavření prohlížeče nebo ztráty připojení nedošlo ke ztrátě dat.

4.2.2.3.1 Stránka pro přihlašování

Jedná se o standartní přihlašovací stránku a na následujícím obrázku bude zobrazen její konečný design. Tlačítko s plusem celý formulář přetransformuje na formulář pro registraci, který se vzhledově liší jenom polem pro potvrzení hesla navíc a jinými nadpisy.



Obrázek 35 - Konečný design přihlašovací obrazovky

Na této stránce bude také demonstrována struktura kódu ve stránkách frameworku Vue.js. Stránka je rozdělena na dvě části. Hlavní část je klasická šablona s HTML kódem, obsahuje v sobě však kód, který se překládá za chodu z JavaScriptu. Následující úryvek kódu bude tuto skutečnost demonstrovat.

```
<template>
  <AuthPageSkeleton v-if="authStore.isLoading" />
  <div id="auth">
    <q-page
      class="row justify-center items-center"
      style="background: linear-gradient(#f3f3f5, #b4b4b4)"
      :class="{ 'bg-grey-9': $q.dark.isActive }"
    >
      <div :class="{ 'q-pa-lg': !$q.platform.is.mobile }">
        <div class="row">
          <q-card
            square
            :class="{ 'shadow-24': !$q.dark.isActive }"
            :style="{
              minWidth: mobileW + 'px',
              minHeight: mobileH + 'px',
            }"
          >
            <!--Name of the form - either Login or Register -->
            <q-card-section
              class="bg-dark"
              :class="{ 'bg-black': $q.dark.isActive }"
            >
              <h4 class="text-h5 text-white text-center q-my-md">
                {{ title }}
              </h4>
            </q-card-section>

            <q-card-section>
              <!--Button for switching between register and login form-->
              <q-btn
                fab
                ref="switchButton"
                color="orange-13"
                @click="switchTypeForm"
                icon="add"
                class="absolute"
                style="top: 0; right: 12px; transform: translateY(-50%)"
```

Obrázek 36 - Ukázka HTML kódu stránky pro přihlašování

Je možné si zde povšimnout například podmínky v-if, která je součástí podmíněného renderování ve Vue.js. Pokud podmínka neplatí, celý HTML element, v kterém se podmínka nachází se nebude vůbec renderovat.

Dále je možné si povšimnout dvou atributů class v několika elementech. Class zapsaný bez dvojtečky přijímá text a aplikuje CSS třídy, které jsou v tomto textu definované (Tyto CSS třídy jsou součástí frameworku Quasar). Class zapsaný s dvojtečkou jako „:class“ ale přijímá JavaScriptový kód, kdy se textovému řetězci definovaného v uvozovkách ve složených závorkách přiřadí hodnota proměnné definované za symbolem dolaru ($). Tato

proměnná je v těchto případech datového typu boolean, takže text dodaný do vlastnosti class bude mít hodnotu buď true nebo false a podle toho se buď přidá k existujícímu atributu class anebo ne. Dá se takto dynamicky určit chování některých elementů pomocí proměnných, které jsou definovány jinde.

Další pozoruhodná záležitost je vkládání hodnoty proměnných do html kódu uvnitř dvojitých složených závorek. Tento kód jednoduše vloží na místo těchto závorek hodnotu, která je přiřazena do těchto proměnných jako text. Lze tak dynamicky zobrazovat výsledky výpočtů a operací pomocí proměnných, ke kterým lze přistoupit i v HTML šabloně.

Poslední využití vlastností frameworku Vue.js v tomto snímku je pak použití události click, definované zavináčem (@). Jedná se o tlačítko, které má tuto událost definovanou a v tomto kódu se k ní přiřadí metoda, která je definovaná dále v kódu. Jinými slovy se tímto docílí toho, že kdykoliv uživatel klikne na element tohoto tlačítka, zavolá se zde definovaná metoda.

Druhou částí kódu je poté čistý JavaScript, kam se zapisuje veškerá logika, kterou je poté možno použít v HTML šabloně. Na následujícím obrázku je demonstrováno, jak taková logika, psaná v JavaScriptu (konkrétně v TypeScriptu) vypadá.

```vue
<script lang="ts">
import { useAuthStore } from 'src/stores/AuthStore';
import { mapStores } from 'pinia';
import AuthPageSkeleton from 'src/components/skeletons/AuthPageSkeleton.vue';
import { QInput } from 'quasar';

export default {
  name: 'AuthPage',

  components: {
    AuthPageSkeleton,
  },

  props: ['register'],

  data: function () {
    return {
      title: this.$t('auth.title.login'), //Přihlásit se
      email: '',
      password: '',
      repassword: '',
      registerForm: false,
      passwordFieldType: 'password' as 'password' | 'text',
      btnLabel: this.$t('auth.label.button.login'), //Přihlásit
      visibility: false,
      visibilityIcon: 'visibility',
    };
  },

  created() {
    //switching the form type based on the link user came from passed through props
    if (this.register == 1) this.switchTypeForm();
  },

  methods: {
    //rule for checking if the password and password confirm match (idk how to get the value of password if i put this rule to the rules file)
    diffPassword(val: string) {
      const val2 = this.password;
```

Obrázek 37 - Ukázka JavaScriptového kódu ve stránce pro přihlašování

V tomto kódu si můžeme povšimnout, že se zde importují jiné komponenty, které jsou nutné pro běh stránky. Importuje se zde například onen Store pro přihlašování, který je zmíněný výše. Dále se zde definuje název této stránky a jaké komponenty používá. Tato komponenta je pak použita v HTML kódu (V obrázku výše je to druhý řádek a jedná se načítací skeleton, který se zobrazí, jenom pokud Store zrovna komunikuje s API a je ve stavu načítání.)

Další vlastností je poté pole props, kam se definují jednotlivé vlastnosti této stránky. To je zejména důležité pro komponenty. Jedná se vlastně o vstupní parametry. V případě této stránky je to parametr, který je stránce předán při přesměrování z URL adresy a určuje, zda se má stránka zobrazit jako formulář pro přihlašování anebo registraci.

Velmi důležitou funkcí jsou pak data, kde se definují všechny proměnné, které je pak možné využít v HTML kódu anebo kdekoliv v JavaScriptovém kódu. Nastavuje se zde pro ně i počáteční hodnota.

Dále je možné si všimnout metody created(), která je součástí základních funkcí frameworku Vue.js. Jedná se o funkci, která se zavolá pokaždé, když se tato stránka začne načítat. Zde je uvedena podmínka, která změní typ formuláře na registrační, pokud je hodnota vstupního parametru „register“ jedna.

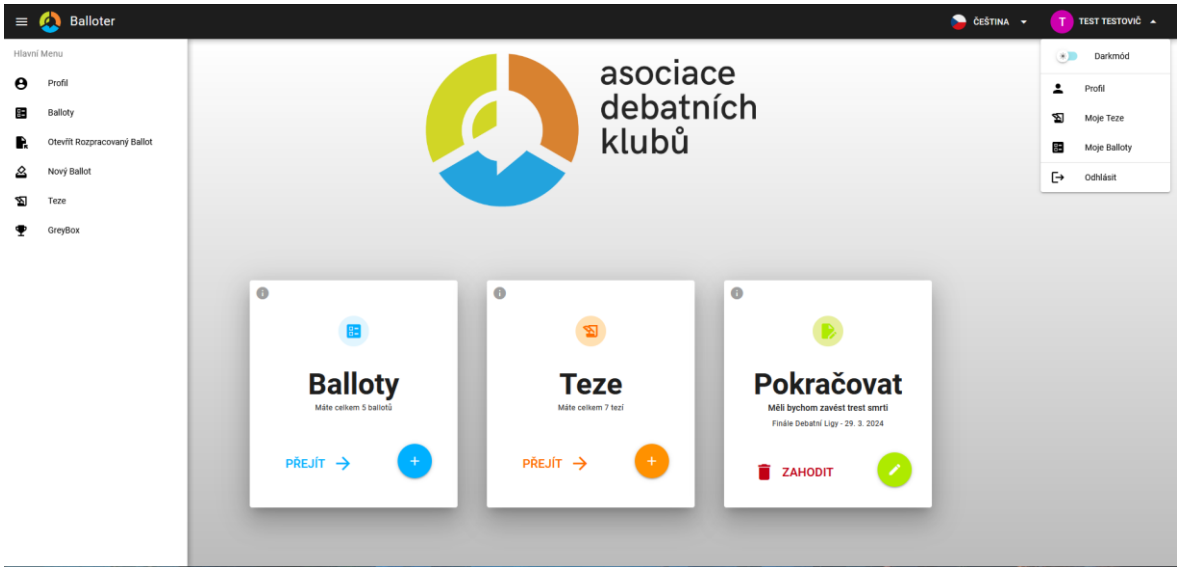
Dalším objektem je pak methods, kam se zapisují jednotlivé funkce volané akcemi v HTML kódu nebo jinde v JavaScriptovém kódu. U této stránky se jedná například o metody pro přihlašování nebo registraci, která z formuláře získají potřebná data a předají je Storu, který komunikuje s API.

Poslední pozoruhodnou věcí na této ukázce kódu je využití globální metody t. (zapsáno jako „this.$t(‘….’);) Jedná se o globální funkci modulu pro překlady. Překlady a všechny texty jsou zde uloženy v komplexních souborech se stromovou strukturou a tato metoda vyhledává v těchto souborech správný textový řetězec a zobrazuje ho. Kvůli tomu, aby byla aplikace vícejazyčná, nesmí být nikde žádné texty, které by byly napevno zapsané v kódu, ale musí být uloženy v souborech s datovými strukturami, kde se potom vybírá správný textový řetězec podle zvoleného jazyka.

4.2.2.3.2 Stránka domovské obrazovky

Domovská obrazovka je velmi prostá stránka. Nabízí pouze možnost rychlého přesměrování do ostatních částí aplikace a v případě, že je uživatel nepřihlášený, ho bude vybízet k tomu, aby se přihlásil nebo registroval.

Následující obrázek demonstruje hotový design stránky pro domovskou obrazovku, včetně nabídek a hlavního panelu. V panelu na pravé straně obrazovky je možné si všimnou možnosti odhlášení nebo přepínače pro temný režim. Karta „Pokračovat“ a položka v menu „Otevřít rozpracovaný ballot“ se objevuje pouze pokud má uživatel v paměti prohlížeče automaticky uložený nedokončený ballot. Položka v menu Moje balloty nebo teze pak uživatele přesměruje na stránky s balloty nebo tezemi s předenem nastaveným filtrem pro vyfiltrování jenom těch tezí nebo ballotů které uživatel vlastní nebo se jich účastnil.

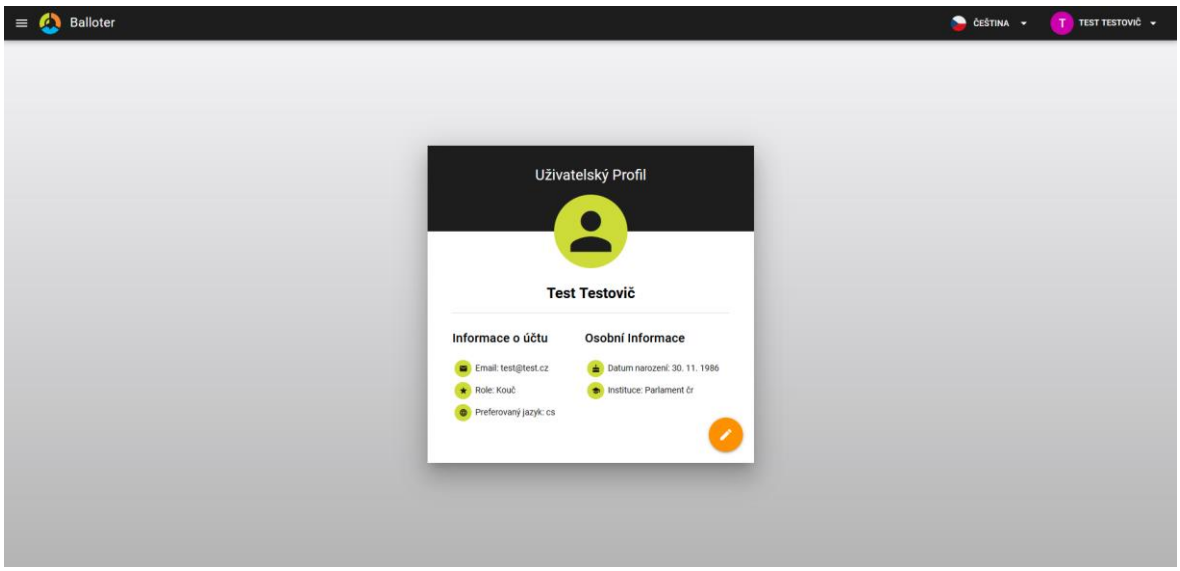


Obrázek 38 - Konečný design domovské obrazovky

4.2.2.3.3 Stránka uživatelského profilu

Uživatelský profil je stránka, která uživateli umožňuje si prohlédnout svoje data a po kliknutí na tlačítko přejít na formulář, kde tyto data může editovat nebo smazat. Má tedy pouze informační charakter a není příliš složitá, pouze načítá data o uživateli z backendu prostřednictvím Storu pro přihlašování.

Na následujícím obrázku je možné vidět konečný design této stránky.

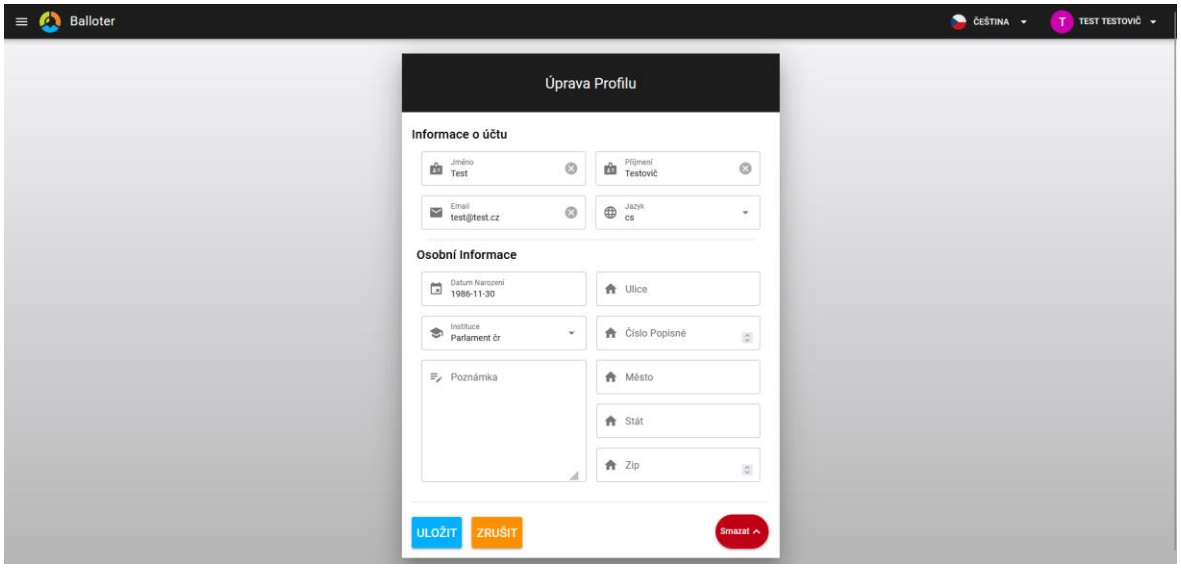


Obrázek 39 - Konečný design uživatelského profilu

4.2.2.3.4 Formulář uživatelského profilu.

Jedná se o formulář, kde uživatel může vyplnit všechny své údaje nebo jenom část anebo je smazat. Je zde možné smazat jenom vyplněné údaje anebo rovnou celý uživatelský účet.

Na následujícím obrázku je možné vidět konečný design tohoto formuláře. Informace jsou rozděleny do dvou kategorií a tlačítka jsou barevně rozlišena pro lepší orientaci.



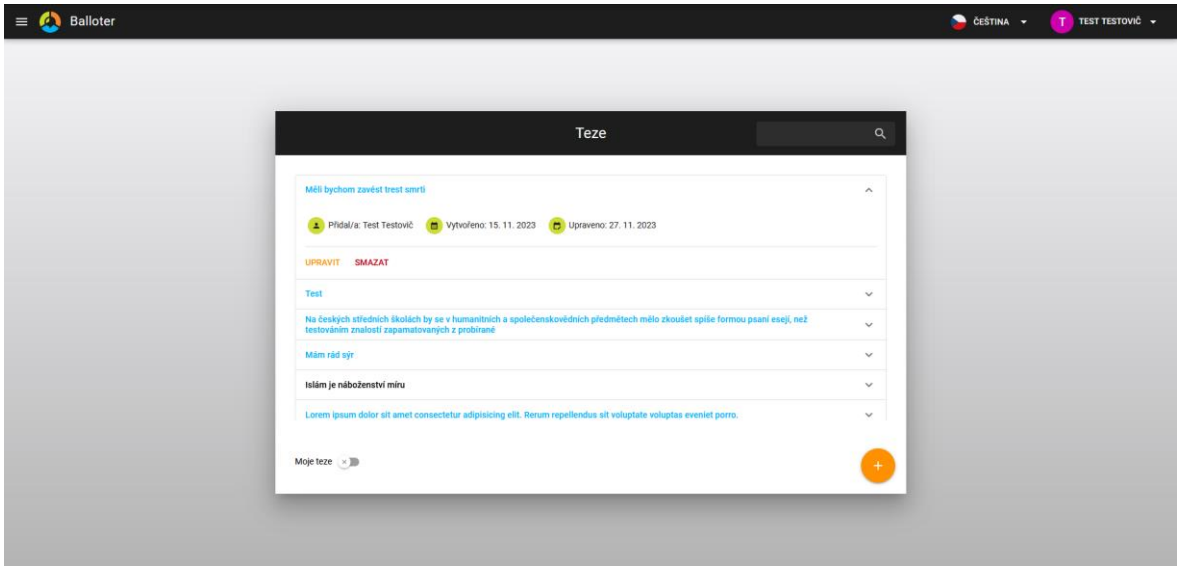
Obrázek 40 - Konečný design formuláře uživatelského profilu

Tento formulář kromě odesílaní dat do backendu pomocí Storu, zároveň i kontroluje validitu vstupu a faktu, jestli uživatel vyplnil povinné údaje. V tomto případě lze například vyplnit pouze celou adresu anebo jí nevyplňovat vůbec. Jméno a příjmení je vždy povinný údaj, bez kterého se nedají používat pokročilejší funkce aplikace, pokud je uživatel rozhodčí.

4.2.2.3.5 Stránka pro zobrazení tezí

Jedná se o jednoduché okno, které zobrazuje seznam všech tezí. Umožňuje uživateli v těchto tezích vyhledávat psaním do textového pole anebo zobrazit pouze teze, které přidal uživatel. Takové teze jsou také vždy odlišeny modrou barvou. Kliknutí na tlačítko plus pak zobrazují vyskakovací okno s výzvou pro vytvoření nové teze a kliknutí na samotnou tezi zobrazuje další informace o tezi a možnost jí smazat nebo upravit.

Následující obrázek demonstruje výsledný vzhled této stránky.

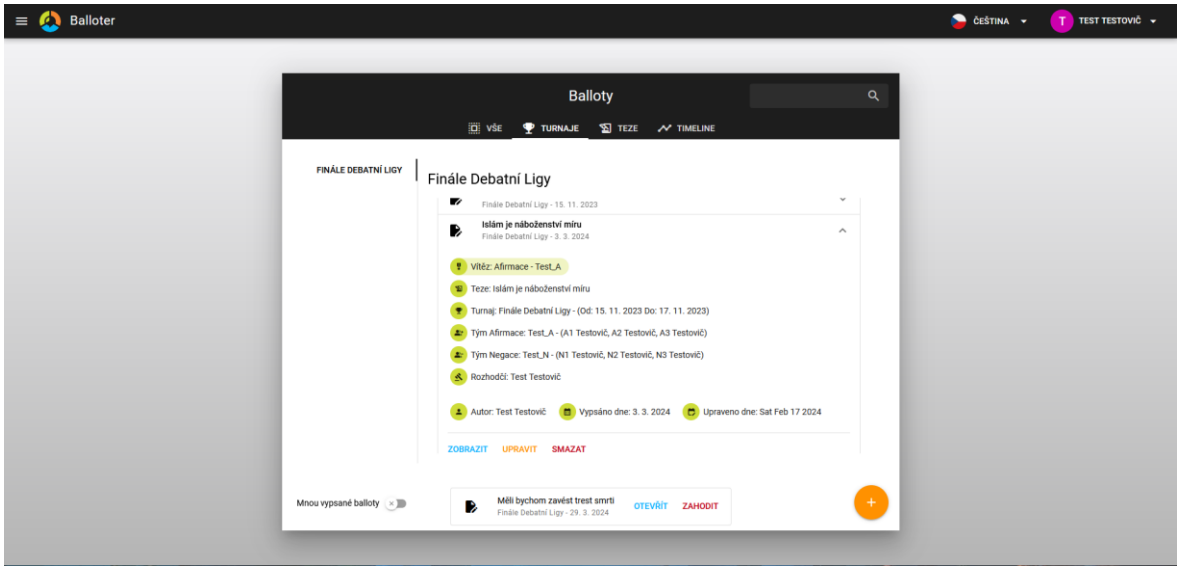


Obrázek 41 - Konečný design stránky pro teze

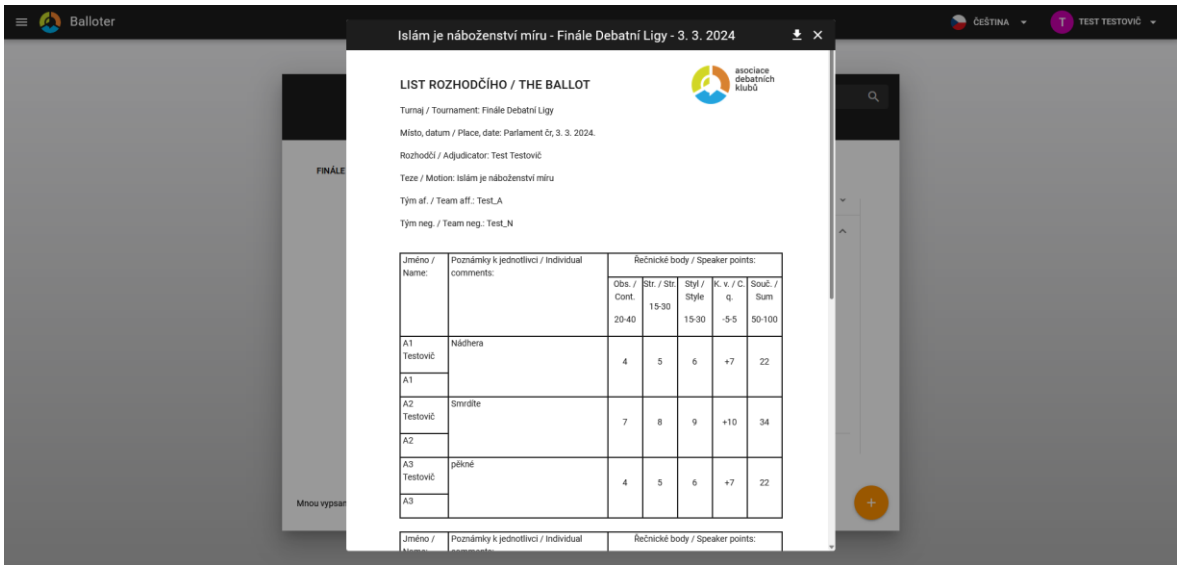
4.2.2.3.6 Stránka pro zobrazení ballotů

Jedná se o stránku, která principiálně funguje totožně jako stránka pro balloty, jenom je komplikovanější. Zobrazuje více informací a umožňuje balloty filtrovat podle turnajů anebo podle tezí. Kliknutí na tlačítko plus pak uživatele přesměruje na formulář pro tvorbu nových ballotů, stejně jako kliknutí na tlačítko upravit, v tom případě bude formulář předem vyplněný. Tlačítko zobrazit poté formulář zobrazuje ve vyskakovacím okně tak, jak by ballot vypadal vytištěný, toto okno zároveň umožňuje ballot exportovat v PDF a stáhnout. Pokud má uživatel uložený rozpracovaný ballot, bude se také ve spodní části okna zobrazovat možnost tento ballot otevřít nebo zahodit. Záložka timeline pak umožňuje zobrazit balloty v přehledné časové řadě místo zobrazení v seznamu.

Tak jak to vypadá v praxi budou demonstrovat následující obrázky.



Obrázek 42 - Konečný design stránky pro balloty



Obrázek 43 - Ukázka zobrazení detailu ballotu

4.2.2.3.7 Formulář pro balloty

Formulář pro tvorbu ballotů je alfou a omegou této aplikace, je to nejkomplikovanější část celé aplikace a obsahuje asi 1500 řádků kódu.

Aplikace zde načte seznam turnajů a tezí, z kterých poté rozhodčí může zvolit ty správné. Podle zvoleného turnaje pak předvyplní pole pro místo konání turnaje a načte jen ty týmy, které jsou na tomto turnaji registrované a umožní uživateli z nich vybrat. Datum se vkládá automaticky podle dnešního data. K tomu má aplikace stopky pro přípravné časy obou

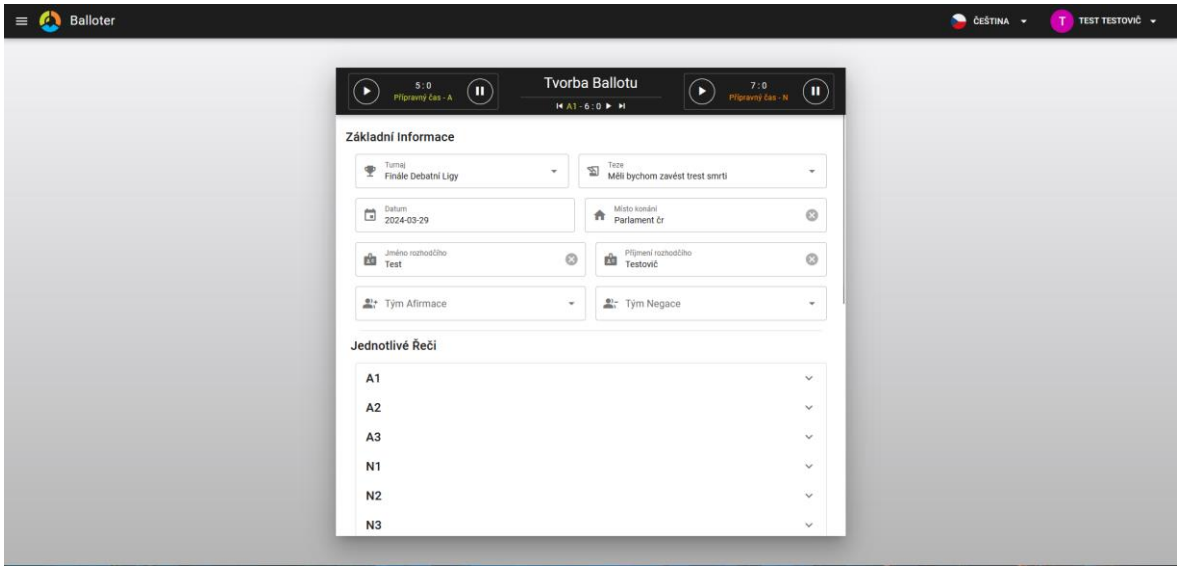
týmů přehledně v hlavičce spolu s displejem a ovládacími tlačítky pro všechny ostatní časy řečníků.

Řečníci jsou pak situovaní do seznamu, kde jdou jednotlivé položky minimalizovat a maximalizovat podle potřeby. U každého řečníka má poté uživatel možnost výběru jména. Možnosti pro výběr řečníků se načtou podle vybraných týmů. Dále jsou zde stopky pro příslušnou řeč a osobní komentáře, kde aplikace kontroluje počet zadaných znaků. Další pole jsou jednotlivé body, u kterých aplikace hlídá, aby uživatel nezadal neplatnou hodnotu.

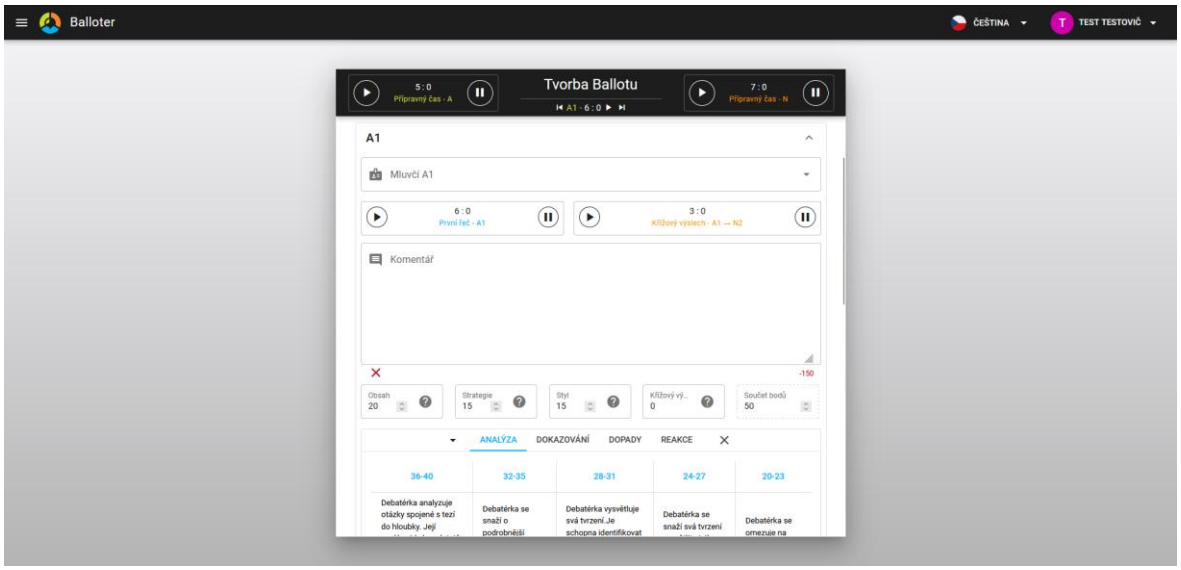
Každé pole pro body také obsahuje tlačítko s otazníkem, které zobrazí tabulku s přehledně organizovanými radami z bodovací tabulky spolu s tlačítky pro výběr bodového rozsahu pro každou bodovou kategorii. V případě, že uživatel tyto tlačítka v tabulce využije, celkové počty bodů se pak dopočítávají automaticky a pakliže ještě není nic napsáno v osobním komentáři, aplikace se podívá na ty zvolené podkategorie bodů, které se nejvíce vymykají průměru a doporučí uživateli napsat osobní komentář právě o nich. Aplikace dále umí převádět starý systém zápisu bodů za křížový výslech na nový a součet bodů dopočítává automaticky.

Aplikace navíc ukládá ballot do paměti, vždy, když se změní nějaká hodnota.

Následující obrázky budou demonstrovat, jak vypadá formulář ve své konečné podobě.



Obrázek 44 - Konečný design formuláře pro balloty



Obrázek 45 - Ukázka polí pro jednoho řečníka ve formuláři pro balloty

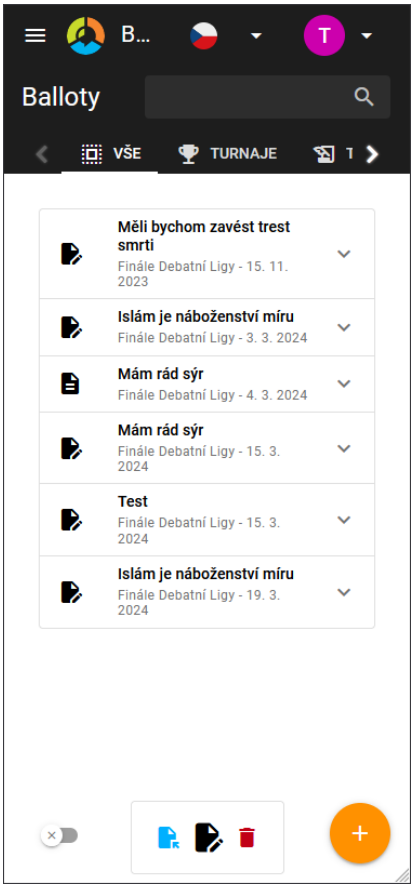
4.2.2.4 Finalizace aplikace

Poté co byly hotovy všechny podstatné stránky a jejich funkcionalita byla uspokojivá bylo nutné doladil další aspekty aplikace. Prováděly se kosmetické úpravy stejně jako praktická vylepšení. V následující podkapitole budou shrnuty tyto poslední kroky před testováním aplikace.

4.2.2.4.1 Responzivní design

Aplikace byla již od začátku tvořena tak, aby podporovala rozličné typy obrazovek. Bylo ale potřeba učinit několik úprav, aby vypadala dobře i na mobilních zařízeních. Bylo tedy nutné projít celou aplikaci a zajistit, aby se žádné elementy nepřekrývaly a zobrazovaly se správně i na všech typech zařízení. Nyní je aplikace plně responzivní a podporuje i mobilní telefony.

Následující obrázek bude ilustrovat, jak aplikace vypadá na mobilním telefonu.

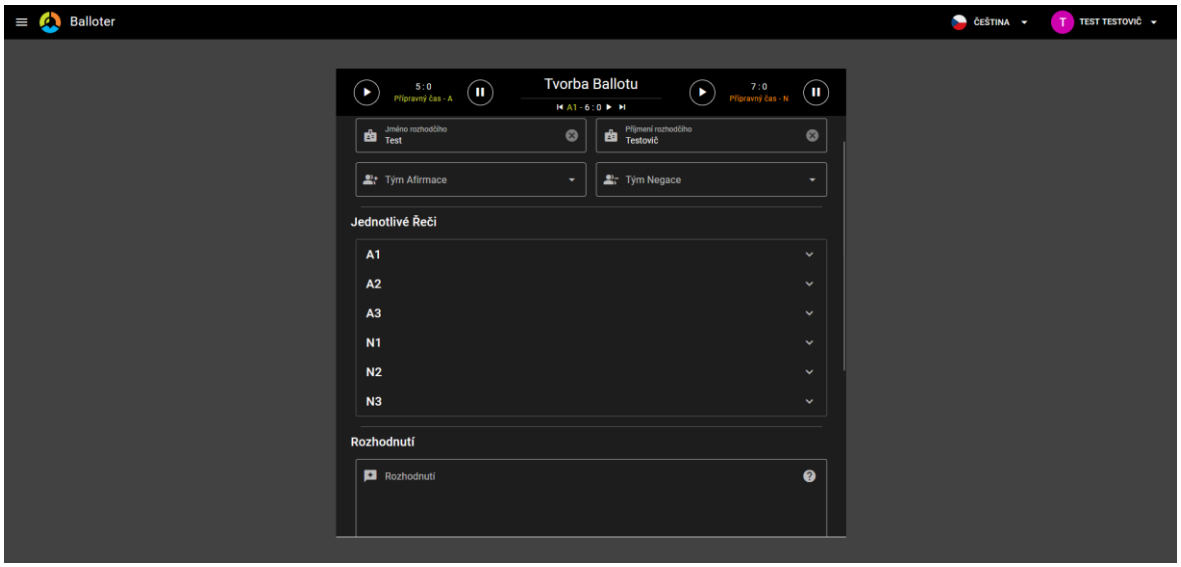


Obrázek 46 - Ukázka responzivního designu na mobilním zařízení

4.2.2.4.2 Tmavý režim

V dnešní době je velmi moderní dávat uživateli možnost přepnout aplikaci do tmavého nebo světlého režimu. Tato funkce usnadňuje používání a dělá aplikaci příjemnější na čtení. Autor tedy využil předpřipravené funkcionality frameworku Quasar a zajistil, aby se aplikace zobrazovala správně i ve tmavém režimu.

Následující obrázek demonstruje, jak aplikace vypadá s tmavým režimem.



Obrázek 47 - Ukázka tmavého řežimu

4.2.2.4.3 Skeletony

Další drobnou úpravou bylo přidání skeletonů do každé stránky jako komponenty. Jedná se o hrubý obrys stránky, která se má zobrazit a zobrazuje se během toho, co se aplikace načítá, aby měl uživatel představu o tom, jakou podobu bude mít obsah, který se právě načítá.

Následující obrázek ukáže, jak vypadá takový skeleton pro formulář pro tvorbu ballotů.

Obrázek 48 - Ukázka skeletonu pro formulář pro balloty

4.2.2.4.4 Překlad aplikace

Aby mohla být aplikace vícejazyčná, musel autor využít funkce modulu i18n, který slouží k internacializování webových aplikací. Funguje tak, že se pro každou stránku vytvoří soubory formátu json, které mají strukturu podobnou klasickým objektům v JavaScriptu. Do tohoto souboru se zadají všechny texty v aplikaci jako proměnné a tam, kde se má tento text zobrazovat se zavolá globální funkce t. Tato funkce bere jako parametr textový řetězec, který specifikuje kde v hierarchii hodnot ve správném souboru se má tento textový řetězec načíst. Každý tento soubor potom má svojí identickou kopii pro každý jazyk. V každé kopii je pak samozřejmě text zadaný v jiném jazyce. V tomto případě pouze v angličtině. Jazyky se pak přepínají elementem v hlavním ovládacím panelu.

V následujícím obrázku bude uveden příklad souboru s těmito texty pro formulář pro tvorbu ballotů. V aplikaci pak existuje stejný soubor, kde jsou tyto texty v angličtině, ale názvy proměnných jsou totožné.

```
export default {
  //BallotForm
  title: 'Tvorba Ballotu',
  tooltip:
    'Kliknutím zobrazte/skryjte časovače na přípravný čas na malých obrazovkách',
  basicInfo: {
    title: 'Základní Informace',
    tournament: 'Turnaj',
    motion: 'Teze',
    date: 'Datum',
    date_close: 'Zavřít',
    place: 'Místo konání',
    name: 'Jméno rozhodčího',
    surname: 'Příjmení rozhodčího',
    teamAFF: 'Tým Afirmace',
    teamNEG: 'Tým Negace',
  },
  speakers: {
    title: 'Jednotlivé Řeči',
    A1: 'A1',
    A2: 'A2',
    A3: 'A3',
    N1: 'N1',
    N2: 'N2',
    N3: 'N3',
    speaker: 'Mluvčí',
    comment: 'Komentář',
    content: 'Obsah',
    strategy: 'Strategie',
    style: 'Styl',
    cq: 'Křížový výslech',
    sum: 'Součet bodů',
    tooltip: {
      content: 'Prohlédněte si BOTA o Obsahu a naklikejte body (20 - 40)',
      strategy: 'Prohlédněte si BOTA o Strategii a naklikejte body (15 - 30)',
      style: 'Prohlédněte si BOTA o Stylu a naklikejte body (15 - 30)',
      cq: 'Křížový výslech - (-5 - +5) nebo (1 - 10) - to znaménko tam prosím pište explicitně',
    },
```

Obrázek 49 - Příklad řešení textu s překlady

4.2.2.4.5 Směrování

Proto, aby bylo možné se v aplikaci pohybovat je nutné mít router, který zpracovává URL adresu zadanou do prohlížeče a zobrazuje patřičnou stránku. Je k tomu použit router pro Vue.js kde se definuje každá cesta URL a její vstupní parametry a přiřazuje se k ní, jaká stránka se má zobrazit. V nastavení tohoto routeru je také funkce, které se říká router guard. Tato funkce umožňuje spouštět kód při každém přesměrování a zobrazuje z jaké cesty na kterou cestu se uživatel snaží přesměrovat. V této aplikaci je funkce router guard použita k ověření toho, zda je uživatel přihlášený před každým přesměrováním. Pokud přihlášený není, aplikace ho vždy přesměruje na přihlašovací obrazovku.

Následující obrázek bude demonstrovat, jak vypadá takový kód nastavení cest pro router. Je možné si povšimnout, že úplně nahoře se definuje cesta pro „Main Layout“ což je hlavní panel spolu s nabídkami. Všechny ostatní cesty jsou pak jeho potomky, což znamená, že

dědí jeho vlastnosti a tento panel a nabídky se poté zobrazuje na každé stránce. Pro každou cestu se zde definuje komponenta neboli stránka, která se má načíst, dále jméno této cesty a samotná cesta. Parametry s dvojtečkou napsány do cesty jsou proměnné parametry, které se pak vkládají do stejnojmenné hodnoty props v dané stránce.

```
import { RouteRecordRaw } from 'vue-router';

const routes: RouteRecordRaw[] = [
  {
    path: '',
    component: () => import('layouts/MainLayout.vue'),
    children: [
      {
        path: '',
        name: 'home',
        component: () => import('pages/IndexPage.vue'),
      },
      {
        path: '/auth/:register',
        name: 'auth',
        props: true,
        component: () => import('pages/AuthPage.vue'),
      },
      {
        path: '/profile',
        name: 'profile',
        component: () => import('pages/ProfilePage.vue'),
      },
      {
        path: '/profile/edit',
        name: 'profileEdit',
        component: () => import('pages/ProfileForm.vue'),
      },
```

Obrázek 50 - Ukázka definicí URL adres pro router na frontendu

4.3 Testování aplikace

Po dokončení vývoje všech funkcí aplikace bylo nutné tyto funkce otestovat. Autor této práce prováděl průběžné menší testy dílčích funkcí aplikace a podle jejich výsledku aplikaci upravil. Tyto testy byly zejména důležité při návrhu rozložení UI aplikace a ověřování funkčnosti klíčových funkcí aplikace.

Toto testování nebylo příliš dlouhé a probíhalo bez testovacích scénářů. Jednalo se zejména o ověřování funkčnosti napsaného kódu. Když testování dílčích funkcí aplikace proběhlo dle očekávání, nový kód byl připojen ke zbytku a bylo možné se posunout k další etapě vývoje aplikace. Takto se vyvíjela celá aplikace, dokud se nepodařilo splnit všechny požadavky definované při zadání a funkcionalita aplikace nebyla kompletní.

Po dokončení implementace aplikace se konalo testování pomocí scénářů, které popisuje tato podkapitola.

4.3.1 Testovací scénáře

Název testu: Testování uživatelského rozhraní a funkčnosti aplikace

Cíl testu: Zhodnotit uživatelskou přívětivost, funkcionalitu a efektivitu aplikace

Metody testování:

1. **Testování uživatelského rozhraní:** Uživatelé budou požádáni, aby provedli sérii úkolů v aplikaci a vyhodnotili snadnost použití, jasnost navigace a celkovou přívětivost uživatelského rozhraní.
2. **Testování funkčnosti:** Budou provedeny testy na základní funkce aplikace, jako je přidávání tezí, přidávání ballotů, úprava uživatelského profilu a vyhledávání ballotů
3. **Testování kompatibility:** Aplikace bude testována na různých zařízeních (počítače, chytré telefony) a různých webových prohlížečích, aby se zajistila správná funkčnost na různých platformách.

Testovací scénáře:

1. *Registrace a přihlášení:* Uživatel provede registraci a přihlášení do aplikace a vyhodnotí proces.
2. *Vytvoření nového ballotu:* Uživatel vytvoří nový ballot, vyplní všechny hodnoty a ballot zkusí uložit.
3. *Úprava profilu a smazání účtu* Uživatel se pokusí změnit informace ve svém uživatelském profilu a smazat svůj uživatelský účet.
4. *Prohlížení ballotů:* Uživatel bude hledat konkrétní ballot v seznamu a zhodnotí náročnost procesu, dále se uživatel pokusí ballot zobrazit a stáhnout v PDF.

Očekávané výstupy:

1. Zpětná vazba uživatelů ohledně přívětivosti uživatelského rozhraní.

2. Identifikace případných chyb a nedostatků ve funkčnosti aplikace.
3. Zjištění kompatibility aplikace s různými zařízeními a prohlížeči.

Závěrečné shrnutí:

Na základě získaných výstupů budou identifikovány klíčové oblasti pro vylepšení aplikace a budou navrženy konkrétní opatření pro optimalizaci uživatelského zážitku a celkového výkonu aplikace.

4.3.2 Výsledky testování

Aplikace byla testována 5 uživateli, všichni z nich měli zkušenosti s akademickou debatou. Jednalo se o tři muže a dvě ženy.

4.3.2.1 1. Testování uživatelského rozhraní:

- *Přívětivost uživatelského rozhraní:* Většina uživatelů označilo uživatelské rozhraní za snadno použitelné a intuitivní.
- *Jasnost navigace:* Většina uživatelů hodnotilo navigaci v aplikaci jako jasnou a snadno srozumitelnou. Objevily se pouze připomínky u jednoho uživatele ohledně užívání ikony pro tlačítka přidávání a úpravy místo textu.
- *Celková spokojenost:* Všichni uživatelé byli celkově spokojeni s uživatelským rozhraním aplikace i přes zmíněný nedostatek. Někteří uživatelé si všimli, že aplikace zobrazuje v některých případech špatné skeletony při načítání stránek.

4.3.2.2 2. Testování funkčnosti:

- *Správnost funkcí:* Většina testovaných funkcí aplikace pracovala správně, nicméně byly identifikovány drobné chyby v některých částech aplikace. Konkrétně aplikace neodhlašovala uživatele, pokud byl přihlášen někde jinde. Aplikace také vyhazovala chybu při ukládání uživatelského profilu, který původně neměl žádnou adresu a adresa nebyla ve formuláři vyplněna ani nyní. Aplikace občas zobrazí špatné stopky, pokud se používá panel pro zobrazení času v hlavičce formuláře pro balloty. Pokud měl ballot tezi s příliš dlouhým názvem, mohlo se rozbít rozložení elementů na domovské obrazovce.

- *Kompletnost funkčností:* Většina testovaných funkcí aplikace byla kompletní a plně funkčních, byly však identifikovány drobné nedostatky.

4.3.2.3 4. Testování kompatibility:

- *Kompatibilita s prohlížeči:* Aplikace byla z většiny kompatibilní s většinou moderních webových prohlížečů, včetně Chrome, Firefox a Edge. Na prohlížečích na bázi chromia se nezobrazuje vlajka u výběru jazyka jako vlajka, ale jako text.
- *Kompatibilita se zařízeními:* Aplikace byla úspěšně testována na různých zařízeních, včetně počítačů a chytrých telefonů.

Závěrečné zhodnocení:

Celkově bylo testování aplikace úspěšné a přineslo cenné poznatky o uživatelském zážitku, funkcionalitě a výkonu aplikace. Identifikované nedostatky byly opraveny v následných verzích aplikace s cílem zlepšit uživatelskou spokojenost a zvýšit efektivitu používání aplikace. Do aplikace bylo přidáno chování, kdy se zavolá funkce pro odhlášení při každém přesměrování, kdy není jasné, zda je uživatel přihlášen, dále byla opravena chyba na backendu, kdy se volala funkce na hodnotu adresy, která byla v případě prázdné hodnoty null. Bylo nutné přidat podmínku, která s touto možností počítá a případně uloží místo adresy null. Byla implementována funkce, která automaticky zkracuje dlouhé názvy ballotů a tezí, aby zůstal vzhled a rozložení grafického rozhraní netknutý. Zbytek nedostatků byl považován za ne tak podstatný a bylo rozhodnuto, že bude vyřešen v dalších verzích programu.

5 Výsledky a diskuse

V rámci této diplomové práce se autor zaměřil na analýzu, návrh a implementaci aplikace pro usnadnění rozhodování akademických debat formátu Karl Popper. Na základě provedené analýzy, vstupních požadavků a potřeb uživatelů autor navrhnul a vyvinul komplexní řešení, které kombinuje uživatelskou přívětivost s bohatou funkcionalitou. Při návrhu a implementaci autor aplikoval nově nabyté znalosti, popsané v teoretické části této práce.

Jedním z klíčových prvků aplikace je intuitivní uživatelské rozhraní, které umožňuje uživatelům snadnou navigaci a přístup ke všem důležitým funkcím. Zároveň byla implementována široká škála funkcí pro správu a tvorbu listů rozhodčího neboli ballotů, včetně zobrazování a vyhledávání ballotů a tezí, možnosti zobrazování detailů jednotlivých ballotů a jejich exportování do PDF, možnosti upravování a přidávání nových tezí, správy uživatelského profilu, a hlavně možnosti tvorby nových ballotů pomocí robustního formuláře s velkým množstvím pomocných funkcí.

Během testování aplikace získal autor užitečnou zpětnou vazbu od uživatelů, která mu umožnila vylepšit uživatelský zážitek z aplikace a odstranit odhalené nedostatky. Výsledkem je robustní a funkční webová aplikace, která plní potřeby uživatelů stejně jako splňuje vstupní požadavky, vytyčené na začátku této práce, a která přináší skutečnou přidanou hodnotu při rozhodování debat.

Do budoucna má aplikace potenciál být integrována do dalších webových aplikací Asociace debatních klubů, zejména se systémem statistik a výsledků debatních turnajů greybox. Díky využití frameworku Quasar je také možné vytvořit verze aplikace, které by fungovaly jako nativní desktopové nebo mobilní programy, do kterých by se dala přidat offline funkcionalita.

6 Závěr

Cílem této diplomové práce byla analýza, návrh a implementace webové aplikace k usnadnění rozhodování soutěžních debat. V rámci návrhu a analýzy byla nejprve provedena konzultace se zadavatelem, následně došlo k vytyčení funkčních požadavků a tvorba person spolu s vytvořením patřičných diagramů v jazyce UML.

V dalším kroku, tedy v rámci samotné implementace, byl vytvořen backend aplikace jako REST API pomocí frameworku Lumen v PHP. Backend obsahuje 9 modelů a 4 controllery a spolupracuje s databází MySQL. V další fázi implementace byl proveden návrh UX pro frontendovou část aplikace a následně byl frontent aplikace naprogramován ve frameworku Quasar v JavaScriptu, obsahující 8 stránek a 12 komponent. Frontend byl vyvinut včetně podpory přepínání jazyků a propracovaného responzivního designu. Jako poslední krok bylo provedeno testování aplikace. Při implementaci byly užity patřičné metodiky softwarového inženýrství. Aplikace byla vyvíjena metodou prototyp za využití řádných postupů objektově orientovaného programování a SOLID principů.

V první části práce jsou představeny všechna nutná teoretická východiska a poznatky, které byly použity jako podklady pro samotnou implementaci. Jedná se zejména o principy softwarového inženýrství jako metodiky vývoje software nebo filozofie objektově orientovaného programování. V neposlední řádě jsou popsány také nutné teoretické poznatky pro tvorbu databází, backendu a frontendu webové aplikace, včetně všech užitých technologií, zejména framework Laravel a Lumen pro backend a framework Quasar a Vue.js pro frontend spolu s potřebnými podklady pro porozumění akademickému debatování.

V druhé části práce se autor zabývá praktickou stránkou věci, tedy samotným implementováním webové aplikace. Je představen celý postup vývoje, počínaje prvotní analýzou a stanovením funkčních požadavků spolu s tvorbou diagramů, přes programování backendu v jazyce PHP a následné prototypování a programování frontendu v JavaScriptu a testováním aplikace konče.

Na základě provedené implementace webové aplikace pro rozhodování debat lze konstatovat, že tato diplomová práce přinesla očekávané výsledky a splnila stanovené cíle. Autor navrhl a vyvinul uživatelsky přívětivou aplikaci, která umožňuje jejím uživatelům efektivně a pohodlně vytvářet nové balloty během debaty a zároveň umožňuje jejich komfortní správu a prohlížení.

Díky provedené analýze a testování získal autor užitečnou zpětnou vazbu, která mu pomohla vylepšit aplikaci a zajistit její kvalitu a spolehlivost. Výsledná aplikace má potenciál poskytnout uživatelům užitečný nástroj v rámci debatování jak pro rozhodčí, tak pro debatéry.

7 Seznam použitých zdrojů

ASOCIACE DEBATNÍCH KLUBŮ, 2021. *Pravidla debaty 2021/2022*. 1. Dostupné z: https://debatovani.cz/wp-content/uploads/2023/08/pravidla-debaty_21-22.pdf. [cit. 2024-03-28].

ASOCIACE DEBATNÍCH KLUBŮ, 2023. *O NÁS*. online. In: Asociace debatních klubů. Dostupné z: https://debatovani.cz/o-nas/. [cit. 2024-03-28].

ASOCIACE DEBATNÍCH KLUBŮ, 2023. *O DEBATOVÁNÍ*. online. In: Asociace debatních klubů. Dostupné z: https://debatovani.cz/debata/. [cit. 2024-03-28].

BOS, Bert, 2024. *A brief history of CSS until 2016*. online. In: W3C. Dostupné z: https://www.w3.org/Style/CSS20/history.html. [cit. 2024-03-27].

CODEACADEMY, 2024. *What is a Web App?*. online. In: Code Academy. Dostupné z: https://www.codecademy.com/article/what-is-a-web-app. [cit. 2024-03-26].

COMPOSER COMMUNITY, 2024. *Introduction*. online. In: Composer. Dostupné z: https://getcomposer.org/doc/00-intro.md. [cit. 2024-03-27].

CONGER, Steve, 2012. *Hands-on database: An introduction to database design and development*. 1. Pearson Education. ISBN 978-0-13-610827-6.

FIGMA, 2024. *FreeCodeCamp Website Ui*. online. Dostupné z: https://www.figma.com/file/mh52sQHBF8Bq2pIZhLKVuh/freeCodeCamp-Website-Ui?type=design&node-id=0-1&mode=design. [cit. 2024-03-26].

FOWLER, Martin, 2004. *UML distilled: a briefguide to the standard object modeling language*. 3rd ed. Addisson-Wesley object technology series. Boston: Addison-Wesley. ISBN 03-211-9368-7.

HANNAH, Jaye, 2024. *A Complete Guide to the UI Design Process*. online. In: UX Design Institute. Dostupné z: https://www.uxdesigninstitute.com/blog/guide-to-the-ui-design-process/. [cit. 2024-03-25].

HARTINGER, David, 2024. *Lekce 1 - Úvod do UML*. online. In: ITnetwork.cz. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-uvod-historie-vyznam-a-diagramy. [cit. 2024-03-24].

HARTINGER, David, 2024. *Lekce 5 - UML - Class diagram*. online. In: ITnetwork.cz. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-class-diagram-tridni-model. [cit. 2024-03-24].

HARTINGER, David, 2024. *Lekce 2 - UML - Use Case Diagram*. online. In: ITnetwork.cz. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram. [cit. 2024-03-24].

HARTINGER, David, 2024. *Lekce 1 - Úvod do objektově orientovaného programování v C#*. online. In: ITnetwork.cz. Dostupné z: csharp/oop/c-sharp-tutorial-uvod-do-objektove-orientovaneho-programovani. [cit. 2024-03-25].

HOW-TO GEEK, 2024. *What Is GitHub, and What Is It Used For?*. online. In: How-To Geek. Dostupné z: https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/. [cit. 2024-03-28].

HUSAR, Alex, 2024. *How to Use REST APIs – A Complete Beginner's Guide*. online. In: FreeCodeCamp. Dostupné z: https://www.freecodecamp.org/news/how-to-use-rest-api/. [cit. 2024-03-26].

CHACON, Scott a STRAUB, Ben, 2014. *Pro Git*. online. 2. Apress. ISBN 1484200772. Dostupné z: https://git-scm.com/book/en/v2. [cit. 2024-03-28].

INTERACTION DESIGN FOUNDATION, 2016. *User Experience (UX) Design*. online. In: INTERACTION DESIGN FOUNDATION. Interaction Design Foundation. Dostupné z: https://www.interaction-design.org/literature/topics/ux-design. [cit. 2024-03-25].

INTERNATIONAL JAVASCRIPT INSTITUTE, c2015-2022. *History of JavaScript*. online. In: International JavaScript Institute. Dostupné z: https://www.javascriptinstitute.org/javascript-tutorial/history-of-javascript/. [cit. 2024-03-27].

KOĎOUSKOVÁ, Barbora, 2024. *Co je to API a jaké jsou možnosti jeho využití?*. online. In: Rascasone. Dostupné z: https://www.rascasone.com/cs/blog/co-je-api/. [cit. 2024-03-26].

LARAVEL HOLDINGS, c2011-2024. *Installation*. online. In: Laravel. Dostupné z: https://laravel.com/docs/11.x. [cit. 2024-03-27].

LARAVEL HOLDINGS, c2011-2024. *Eloquent: Getting Started*. online. In: Laravel. Dostupné z: https://laravel.com/docs/11.x/eloquent. [cit. 2024-03-27].

LARAVEL HOLDINGS, c2011-2024. *Artisan Console*. online. In: Laravel. Dostupné z: https://laravel.com/docs/11.x/artisan. [cit. 2024-03-27].

LEMONAKI, Dionysia, 2024. *Frontend VS Backend – What's the Difference?*. online. In: FreeCodeCamp. Dostupné z: https://www.freecodecamp.org/news/frontend-vs-backend-whats-the-difference/. [cit. 2024-03-27].

Looka, 2024. online. Dostupné z: https://looka.com/blog/wireframe-examples/. [cit. 2024-03-26].

LUCID SOFTWARE INC., 2024. *What is an Entity Relationship Diagram (ERD)?*. online. In: LUCID SOFTWARE INC. Lucidchart. Dostupné z: https://www.lucidchart.com/pages/er-diagrams. [cit. 2024-03-24].

LUCID SOFTWARE INC., 2024. *What is a Workflow Diagram*. online. In: LUCID SOFTWARE INC. Lucidchart. Dostupné z: https://www.lucidchart.com/pages/tutorial/workflow-diagram. [cit. 2024-03-24].

MOZILLA, c1998–2024. *What is the difference between webpage, website, web server, and search engine?*. online. In: MOZILLA. MDN Web Docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/Pages_sites_servers_and_search_engines. [cit. 2024-03-26].

MOZILLA, c1998–2024. *MVC*. online. In: MOZILLA. MDN Web Docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/MVC. [cit. 2024-03-26].

MOZILLA, c1998–2024. *What is a web server?*. online. In: MDN Web Docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server. [cit. 2024-03-26].

MOZILLA, c1998–2024. *HTML: HyperText Markup Language*. online. In: MDN Web Docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML. [cit. 2024-03-27].

MOZILLA, c1998–2024. *CSS: Cascading Style Sheets*. online. In: MDN Web Docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS. [cit. 2024-03-27].

MOZILLA, c1998–2024. *What is JavaScript?*. online. In: MDN Web Docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [cit. 2024-03-27].

MOZILLA, c1998–2024. *JavaScript*. online. In: MDN Web Docs. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/javascript. [cit. 2024-03-27].

MŮČKA, Jan, 2024. *Relační databáze vs. nerelační databáze: jaké jsou mezi nimi rozdíly?*. online. In: MasterDC. Dostupné z: https://www.master.cz/blog/relacni-databaze-nerelacni-databaze-jake-jsou-rozdily/. [cit. 2024-03-27].

NAEEM, Tehreem, 2024. *What is Data Integrity in a Database? Why Do You Need It?*. online. In: Astera. Dostupné z: https://www.astera.com/type/blog/data-integrity-in-a-database/. [cit. 2024-03-27].

NAVONE, Estefania Cassingena, 2022. *What is Programming? A Handbook for Beginners*. online. In: FREECODECAMP. FreeCodeCamp. Dostupné z: https://www.freecodecamp.org/news/what-is-programming-tutorial-for-beginners/. [cit. 2024-03-24].

NECKÁŘ, Jan, 2016. *Algoritmy.net*. online. Dostupné z: https://www.algoritmy.net. [cit. 2024-03-25].

NOVIKOVA, Darya, c2016-2024. *Website VS Web Application: the Difference Explained*. online. In: SolveIt. Dostupné z: https://solveit.dev/blog/web-app-vs-website. [cit. 2024-03-26].

O'REILLY MEDIA, 2024. *Chapter 1. MySQL History and Architecturel*. online. In: O'Reilly. Dostupné z: https://www.oreilly.com/library/view/understanding-mysql-internals/0596009577/ch01.html. [cit. 2024-03-28].

OLORUNTOBA, Samuel, 2024. *SOLID: The First 5 Principles of Object Oriented Design*. online. In: DigitalOcean. Dostupné z: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. [cit. 2024-03-25].

ORACLE, 2024. *Co je to databáze?*. online. In: OCI. Dostupné z: https://www.oracle.com/cz/database/what-is-database/#link7. [cit. 2024-03-27].

ORACLE, 2024. *What is MySQL?*. online. In: OCI. Dostupné z: https://www.oracle.com/mysql/what-is-mysql/. [cit. 2024-03-28].

PECINOVSKÝ, Rudolf, 2014. *Objektově orientované programování? Co to je?*. 1. Praha: Vysoká škola ekonomická. [cit. 2024-03-25].

PRESSMAN, Roger S. a MAXIM, Bruce R., 2015. *Software Engineering, a practitioner's approach*. online. 8. New York: McGraw-Hill Education. ISBN 978-0-07-802212-8. [cit. 2024-03-22].

PROCHÁZKA, Jaroslav a KLIMEŠ, Cyril, 2009. *SOFTWAROVÉ INŽENÝRSTVÍ*. 2. Ostrava: Ostravská univerzita v Ostravě. Dostupné z: https://web.osu.cz/~Zacek/6swen/skriptaSWENG.pdf. [cit. 2024-03-21].

PULSARDEV SRL, c2015-2024. *Why Quasar?*. online. In: Quasar. Dostupné z: https://quasar.dev/introduction-to-quasar#a-wide-range-of-platform-support. [cit. 2024-03-27].

RŮŽIČKOVÁ, Natálie, 2024. *Lekce 9 - Co to jsou persony a jak si je vytvořit v UX*. online. In: ITnetwork.cz. Dostupné z: https://www.itnetwork.cz/html-css/user-experience/co-to-jsou-persony-a-jak-si-je-vytvorit-v-ux-. [cit. 2024-03-25].

SASS TEAM, c2006-2024. *Sass Basics*. online. In: Sass. Dostupné z: https://sass-lang.com/guide/. [cit. 2024-03-27].

SKLENÁŘ, Vladimír, 2007. *SOFTWAROVÉ INŽENÝRSTVÍ*. 1. Olomouc: Univerzita Palackého v Olomouci. Dostupné z: https://phoenix.inf.upol.cz/esf/ucebni/syspro.pdf. [cit. 2024-03-21].

SKŘIVAN, Jaromír, 2008. *Datové modely a návrhy relačních schémat*. 1. Filozofická fakulta Univerzity Karlovy v Praze.

SPITTEL, Ali, c2016-2024. *What is a Web Framework, and Why Should You use one?*. online. In: DEV. Dostupné z: https://dev.to/aspittel/what-is-a-web-framework-and-why-should-i-use-one-38c0. [cit. 2024-03-27].

STAUFFER, Matt, 2024. *Introducing Lumen from Laravel*. online. In: Matt Stauffer. Dostupné z: https://mattstauffer.com/blog/introducing-lumen-from-laravel/. [cit. 2024-03-27].

THE PHP GROUP, 2024. *What is PHP?*. online. In: PHP. Dostupné z: https://www.php.net/manual/en/intro-whatis.php. [cit. 2024-03-27].

THE PHP GROUP, c2001-2024. *What can PHP do?*. online. In: PHP. Dostupné z: https://www.php.net/manual/en/intro-whatcando.php. [cit. 2024-03-27].

THE PHP GROUP, c2001-2024. *History of PHP*. online. In: PHP. Dostupné z: https://www.php.net/manual/en/history.php.php#history.php. [cit. 2024-03-27].

VUE.JS, c2014-2024. *Introduction*. online. In: Vue.js. Dostupné z: https://vuejs.org/guide/introduction.html. [cit. 2024-03-27].

W3SCHOOLS, c2009-2024. *Laravel-History*. online. In: W3Schools. Dostupné z: https://www.w3schools.in/laravel/history. [cit. 2024-03-27].

W3SCHOOLS, c2009-2024. *HTML History*. online. In: W3Schools. Dostupné z: https://www.w3schools.in/html/history/. [cit. 2024-03-27].

WINTEMUTE, Doug, 2024. *What Is Back-End Development?*. online. In: Computer Science.org. Dostupné z: https://www.computerscience.org/bootcamps/guides/what-is-back-end-development/. [cit. 2024-03-26].

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam použitých zkratek

ADK – Asociace Debatních Klubů
BOTA – Bodovací Tabulka,
CLI – Command Line Interface,
CRUD – Create, Read, Update, Delete,
CSS – Cascadinh Style Sheets,
DDL – Data Definition Language,
DML – Data Manipulation Language,
HTML – HyperText Markup Language,
KPDP – Karl Popper Debate Program,
MVC – Model View Controller,
ORM – Objektově Relační Mapování,
PHP – Hypertext Preprocessor,
PWA – Progressive Web Application,
UI – User Interface,
UX – User Experience,
RTL – Right To Left,
Sass – Syntactically Awesome Style Sheets,
SCM – Source Code Management,
SCSS – Sassy CSS,
SFC – Single File Component,
SPA – Single Page Application,
SQL – Structured Query Language,
SŘBD – Systém Řízení Báze Dat,
SSG – Static Site Generating,
SSR – Server Side Rendering
XML – Extensible Markup Language,