VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÍ SYSTÉMU MODIN PRO STÁTNÍ SPRÁVU O MODUL SPRÁVY ÚKOLŮ ZAMĚSTNANCŮ

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN POKORNÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148613

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Pokorný Martin**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Rozšíření systému MODIN pro státní správu o modul správy úkolů zaměstnanců**
Kategorie:	Informační systémy
Akademický rok:	2022/23

Zadání:

1. Seznamte se s požadavky státní správy na modul pro sledování a přidělování úkolů zaměstnanců.
2. Prostudujte dostupná vývojová prostředí a zvolte nejvhodnější pro tuto práci. Volbu zdůvodněte.
3. Navrhněte modul dle zjištěných požadavků, navrhněte vhodné propojení se systémem MODIN včetně API. Navrhněte vhodnou architekturu řešení.
4. Implementujte navržený modul včetně pokročilých funkcí a otestujte ho na vhodném vzorku dat.
5. Zhodnoťte dosažené výsledky a další možnosti v pokračování tohoto projektu.

Literatura:

- BASL, J., BLAŽÍČEK, R.: Podnikové informační systémy: podnik v informační společnosti. 3., Praha: Grada, 2012, 323 s. ISBN 978-80-247-4307-3.
- MARSH, J.: UX pro začátečníky. Brno : Zoner Press, 2019. 256 s. ISBN:978-80-7413-397-8

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	24.10.2022

Abstrakt

Tato bakalářská práce se zabývá návrhem a vývojem modulu informačního systému, který zajišťuje organizaci práce zaměstnanců na přidělených úkolech. V práci jsou také popsány praktiky, které se využívají ve vývoji serverových aplikací a technologie, kterými je toto dosaženo.

Abstract

This bachelor thesis is about the design and development of an information system module that ensures the organization of employees' work on assigned tasks. The thesis also describes the practices used in the development of server applications and the technologies used to achieve this.

Klíčová slova

Informační systém, Databáze, Uživatelské prostředí, API, Kontejnerové služby

Keywords

Information system, Databases, User Interface, API, Container services

Citace

POKORNÝ, Martin. *Rozšíření systému MODIN pro státní správu o modul správy úkolů zaměstnanců*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Rozšíření systému MODIN pro státní správu o modul správy úkolů zaměstnanců

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka Ph.D. Další informace mi poskytli zaměstnanci státního archivu v Hradci Králové. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Pokorný
9. května 2023

Poděkování

Děkuji Ing. Vladimíru Bartíkovi Ph.D za vedení, pomoc a rady při vedení této práce. Také bych chtěl poděkovat všem, kteří mi pomáhali při řešení práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tato bakalářská práce pojednává o analýze, návrhu a implementace webové aplikace pro zlepšení organizace zaměstnanců Státního oblastního archívu v Hradci Králové (dále jen SOA HK). Zaměstnanci této instituce individuálně nebo ve skupinách pracují na zadaných úkolech. Sdílení podproblémů, pokroků nebo dalších informací je důležité pro udržení efektivity. Cílem této práce je vyvinutí základu pro informační systém správy úkolů, který bude podporovat možnost rozšíření o dodatečné funkcionality. Samostatný základ bude podporovat běžný systém upomínek a sdílení souborů.

Aplikace bude sloužit jako rozšíření již existujícího systému Modin, který funguje jako hlavní interní informační systém pro zaměstnance archivu, který nabízí několik již existujících modulů, například pro vedení administrativy. Aplikace však má sloužit jako oddělený modul, a kromě základních mechanik dát i možnost budoucímu rozšíření nebo přenesení pod novou základnu, a tím pádem zmodernizovat nynější systém.

Tato práce se bude systematicky zabývat přípravou, návrhem a implementací webového informačního systému.

V kapitole Požadavky budou popsány základní požadavky na funkčnost vytvářeného systému a představen systém Modin, spolu s přehledem jeho prvků.

Kapitola Vývoj webových informačních systémů seznámí čtenáře s nejčastěji používanými praktikami při vývoji webových aplikací, jako jsou architektonické vzory a používané jazyky pro vývoj.

Kapitola Vybrané technologie a praktiky poskytne čtenáři informace o vybraných technologiích a praktikách z předešlé kapitoly.

Kapitola Návrh se bude věnovat konkrétnímu návrhu aplikace, popíše, co bude aplikace obsahovat a jakým způsobem bude implementovat požadavky. Cílem této kapitoly je poskytnout čtenáři představu o tom, jak bude aplikace vypadat.

Kapitola Implementace se bude zabývat obecným a konkrétním postupem implementace aplikace.

Poslední kapitola - Testování, se bude věnovat testování aplikace a popíše jeho důležitost a postupy, které při něm byly použity.

Celá práce bude shrnuta a zhodnocena v Závěru.

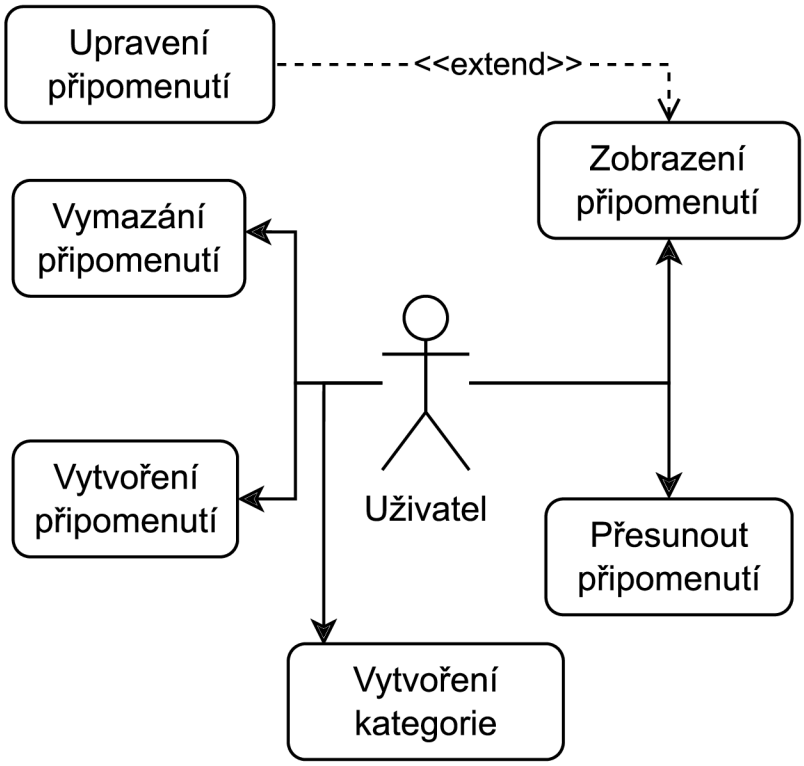
Kapitola 2

Základní požadavky

Při vývoji jakéhokoliv programu je zásadní fáze analýzy. Zákazník nejdříve uvede své požadavky, které se postupem času zkonkrétní v rámci další komunikace. Z této fáze by také mělo vyplynout poznání, zda jsou požadavky splnitelné. Po ujasnění požadavků je možné přejít k návrhu systému. Během vývoje je možné, že se požadavky upraví či upřesní, což může vést k prodloužení vývoje. Proto je nejlepší mít co největší množství konkrétních požadavků ještě před započetím návrhu.
Tato kapitola se bude zabývat pouze základním prvotním návrhem. Dodatečné implementace budou vypsány v kapitole Implementace.

Hlavním cílem bylo vytvoření modulu pro existující informační systém MODIN, který bude pomáhat s organizováním zaměstnanců a zlepšováním efektivity jejich práce. Jeden z hlavních požadavků systému je realizace připomínkového systému, kde si uživatel bude moci vytvořit upomínku na určitou událost nebo úkol, který ho čeká. Pro základní realizaci je nutné vědět, jaký uživatel vytvořil připomínku, její název a popis. Systém by měl dovolovat uživateli přehledně zobrazovat, vytvářet, upravovat a archivovat připomínky. Důvod vytvoření tohoto systému je pomoci uživatelům lépe organizovat svůj čas a zlepšit práci v týmu. Systém bude považován za úspěšný, pokud se podaří eliminovat či drasticky snížit zapomínání, nedorozumění a redundanci práce.

Jako dobrý nástroj posloužilo vytvoření základního diagramu užití 2.1, který dovoluje ukázat všechny možné akce uživatele.



Obrázek 2.1: Diagram užití pro základní implementaci

Modin

Modin je informační systém, který byl vyvinut zaměstnanci SOA HK. Jedná se o centralizovaný systém, který hostuje několik dalších modulů. Jednotlivé moduly slouží pro správu vnitřních především administrativních záležitostí a patří do nich například docházkový systém, nebo systém pro vedení knihy jízd. Celý systém byl vyvinut pomocí jazyku PHP 8.1 za použití MySQL databáze. Systém kvůli právním a bezpečnostním nařízením funguje pouze na lokální síti a není přístupný pro veřejnost. Modin vlastní svoji databázi, kde si ukládá data potřebné pro své moduly a také data o uživatelích.

2.1 Informační systém pro připomínky

Tento modul informačního systému je v základu navržen pro správu osobních připomínek, které jsou dostupné pouze pro jednotlivé uživatele a nejsou sdíleny s ostatními uživateli. Pro přístup k modulu je vyžadováno ověření identity uživatele pomocí přihlašovacích údajů do mateřského systému, v našem případě se tedy jedná o systém MODIN.

Vytváření připomínek

Uživatel může vytvořit neomezené množství připomínek, které mohou být přiřazeny do uživatelem vytvořených kategorií. Každá upomínka má svůj název a popis, který popisuje její obsah. Její název nemusí být unikátní. Dále obsahuje datum splatnosti, podle kterého se určuje závažnost a důležitost dané připomínky.

Úprava připomínek

Připomínky je možno upravovat a měnit jejich základní informace, jako jsou název, popis a datum splatnosti. Také je možné je přemisťovat mezi různými kategoriemi.

Splnění/Archivace připomínek

Pokud uživatel dokončí svou práci na upomínce, může ji označit jako splněnou a přesunout ji do archivu. Archivované připomínky zůstávají v systému a nelze je smazat.

Zobrazení připomínek

Uživatel má možnost kdykoli zobrazit své připomínky tak, aby byly přehledné a uživatelsky přívětivé. Dále může nastavit automatické upozornění na blížící se datum splatnosti dané upomínky.

Nastavení upozornění

Uživatel může nastavit automatické upozornění na určité požadavky, pomocí kterých následně bude upozorněn na blížící se datum vypršení lhůty pro práci na připomínce.

2.2 Rozšířitelnost

Dalším důležitým požadavkem na modul je jeho snadná rozšiřitelnost pro budoucí funkce a schopnost sloužit jako základ pro vývoj nového plnohodnotného informačního systému. To vyžaduje jasnou a čistou architekturu, modularitu a čitelnost kódu, aby bylo možné jednoduše přidat nové funkce a upravit stávající. Kód by měl být psán tak, aby byl co nejvíce oddělený, a závislosti mezi jednotlivými částmi by měly být co nejmenší. To usnadní úpravy a rozšíření modulu v budoucnosti a minimalizuje riziko chyb při úpravách. Důležité je také vytvoření dostatečné dokumentace a popisu rozhraní, aby bylo možné budoucí vývojáře snadno a rychle seznámit s kódem a jeho funkcionalitou.

Tato aplikace by měla být rozšířitelná o další funkce, jako je například možnost sdílení souborů pomocí FTP[1] (Protokol pr přenášení souborů), skupinové připomínky a možnosti opakovatelnosti. Aplikace by měla být samostatná, ale také by měla být použitelná jako asynchronní prvek pro obecnější informační systém.

[1] https://cs.wikipedia.org/wiki/File_Transfer_Protocol

Kapitola 3

Vývoj webových informačních systémů

Výběr správných technologií je stěžejní pro vývoj funkčního informačního systému a je nutné mu věnovat velkou pozornost. V následující části jsou popsány praktiky vývoje webové aplikace a vývoje informačního systému a následně i výběr technologií užitých pro zpracování bakalářské práce.

3.1 Způsoby prezentace

Způsoby prezentace se dělí na několik typů lišících se hlavně podle způsobu načítání obsahu ze serveru. [19]

Statický web

V minulosti byly webové stránky prezentovány uživatelům pomocí statických souborů, které obsahují HTML[1] kód, CSS[2] styly a jednoduchý JavaScript[3]. Tyto soubory jsou vytvořeny na straně serveru a následně odeslány uživatelům. Statické stránky jsou pasivní a nereagují na akce uživatele. Z tohoto důvodu je nutné na každou změnu reagovat znovuvytvořením dané stránky se zobrazenými změnami a znovu ji načíst na straně uživatele. Takový způsob vývoje je přirozeně neefektivní pro stránky, které často mění obsah na základě vstupu od uživatele.

Dynamický web

Dynamický web je modernější verzí prezentace webu, kde obsah může být pozměněn přímo v prohlížeči uživatele. Web pak nevyžaduje při každé změně znovu vytvořit a načíst stránku ze serveru, ale sám ji pomocí již pokročilejšího JavaScriptu upraví bez nutnosti komunikace se serverem. Daný způsob je tak mnohem úspornější a řadí se mezi efektivní způsoby prezentace.

[1] https://developer.mozilla.org/en-US/docs/Web/HTML
[2] https://developer.mozilla.org/en-US/docs/Web/CSS
[3] https://developer.mozilla.org/en-US/docs/Web/JavaScript

Single Page Application [SPA]

SPA (aplikace s jedinou stránkou) [13][4] je plně dynamická a ze serveru přímo načte pouze úvodní stránku. Pokud se obsah stránky mění, je to provedeno buď na straně klienta, nebo se obsah načte asynchronně ze serveru a poté je vložen do úvodní stránky. To umožňuje uživatelům používat web bez viditelného načítání nových stránek. Tato technologie patří mezi nejefektivnější a též uživatelsky nejpřívětivější, ale zároveň patří mezi implementačně složité z důvodu nároků pokročilé implementace na straně uživatele.

3.2 Asynchronní požadavky

Pro správnou funkčnost Dynamického webu 3.1 a SPA 3.1 je důležité, aby se data načítala ze serveru během používání aplikace uživatelem. K tomuto účelu se využívá technologie AJAX (Asynchronous JavaScript And XML), která umožňuje posílat asynchronní HTTP[5] požadavky na server. Po zaslání požadavku čeká, dokud nepřijde odpověď, a poté reaguje na obsah zprávy. [2] Data obsažená v odpovědi jsou poté integrována do již načtené stránky. Díky tomu stránka nemusí načítat celý obsah, ale pouze aktuálně potřebná data, což vede k rychlejšímu a efektivnějšímu načítání.

3.3 Architektonický vzor

Architektonický vzor je způsob rozdělení aplikace do logických celků podle zodpovědností, které obstarávají. Tento přístup pomáhá k organizovanosti projektu a řeší problematiku "špagetového kódu"[24]. Vzájemné použití verzí se však nevylučuje, a je možné použít několik vzorů najednou a doplnit tak jejich funkčnost.

Model-View-Controller (MVC)

MVC (Model-Pohled-Kontrolér) je architektonický vzor, který rozděluje aplikaci do tří hlavních částí:

- **Model** – reprezentuje data aplikace
- **Pohled** – zobrazuje data uživatelům
- **Kontrolér** – řídí interakci mezi uživatelem a aplikací

Mezi výhody MVC patří lepší organizace a údržba kódu, rozdělení zodpovědností a zvýšená modularita aplikace, protože jednotlivé části nejsou na sobě závislé. Mezi nevýhody patří větší složitost kódu a náročnější vývojový proces, čímž se stává nevhodným pro užití při vývoji malých aplikacích.

Model-View-ViewModel (MVVM)

MVVM (Model-Pohled-PohledModel) stejně jako MVC dělí aplikaci na tři hlavní části, ale narozdíl od MVC vynechává potřebu Kontrolerů, ale zaměřuje se na přímé vázání dat na Pohled.

[4] Ve slovníku pod "SPA (Single-page application)"
[5] https://en.wikipedia.org/wiki/HTTP

Repository pattern

Tento vzor se používá na oddělení aplikace od zdroje dat, například od databáze. Repository pattern se používá pro jednotné rozhraní s různými zdroji dat. Použitím jediného rozhraní pro získání a ukládání dat z různých zdrojů umožňuje jednodušší údržbu kódu a škálování aplikace.

3.4 Aplikační rozhraní

Aplikační rozhraní je soubor funkcí a pravidel, které existují uvnitř aplikace. Dovolují komunikaci s aplikací jinak než pomocí uživatelského rozhraní. [13][6]

REST (REpresentational State Transfer)

Aplikační rozhraní REST je v současnosti nejčastější a běžně se používá pro vytváření webových aplikací a služeb. Využívá standardní HTTP protokol pro komunikaci a nabízí jednoduché rozhraní pro operace CRUD (Create, Read, Update, Delete)[7]. K posílání dat využívá několik formátů, jako jsou například JSON[8], HTML, XLT[9], Python nebo PHP [18].

Dané metody jsou:

GET	Jednoduchá funkce, která očekává odpověď, která obsahuje určitá data.
POST	Funguje pro poslání dat do serverové části a používá se pro vytvožení nových dat.
PUT	Podobné jako metoda POST, ale využívá spíše na aktualizovaní existujících dat.
DELETE	Slouží pro značení vymazání dat.

Tabulka 3.1: Metody u HTTP protokolu

Výhodou REST je jednoduchost a snadná použitelnost pro tvorbu a konzumaci API (Application Programming Interface). REST rozhraní je navrženo tak, aby bylo bezstavové, což znamená, že každý požadavek od klienta obsahuje všechny potřebné informace k jeho zpracování. To umožňuje snadnou škálovatelnost aplikace.

SOAP (Simple Object Access Protocol)

Starší typ aplikačního rozhraní, které se ale stále aktivně používá. Využívá protokol HTTP a XML pro kódování zpráv. Rozhraní je popsáno pomocí WSDL (Popisovací jazyk webových služeb). Hlavní výhodou SOAP je podpora pro přenos informací ve velkém rozsahu a jeho robustnost, avšak s tím souvisí také vyšší složitost jeho užívání.

GraphQL

GraphQL je moderní typ aplikačního rozhraní, který nabízí možnost specifikovat uživateli data, která chce získat, a umožňuje tím efektivnější způsob získávání dat a komunikaci mezi klientem a serverem [17]. Definuje vlastní dotazovací jazyk a vrací odpověď ve formátu

[6] Ve slovníku pod "API"
[7] https://cs.wikipedia.org/wiki/CRUD
[8] https://en.wikipedia.org/wiki/JSON
[9] https://fileinfo.com/extension/xlt

JSON. Nevýhodou může být složitost implementace. GraphQL je nezávislý na protokolu a může využívat jak HTTP tak WebSocket[10].

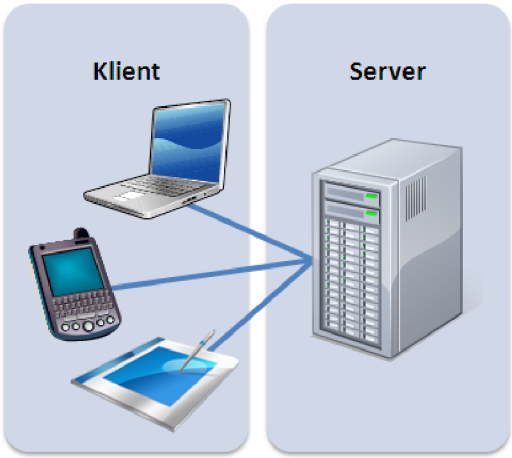
3.5 Architektura

Architektura slouží k rozdělení aplikace na části, které mezi sebou komunikují. I přesto, že rozdělení se může zdát podobné s rozdělením u Architektonických návrhů 3.3, jejich zaměření a problematika, kterou řeší, je spíše globální a obecná.

Dvouvrstvá architektura klient-server

Architektura se skládá ze dvou částí: klient a server, a je také proto označována jako dvouvrstvá. Princip spočívá v tom, že klient je přímý žadatel o službu ze strany serveru, kde server je pouze vlastníkem dat, a tím pádem veškerá aplikační logika musí být obstarána uživatelem. [11] Tento přístup má několik nevýhod. Jedna z nejdůležitějších je snížená bezpečnost, kde klient má přímý přístup k datům na serveru, kde se mohou nalézat citlivá data. Jako další problém je vnímán požadavek na vyšší výpočetní výkon klienta z důvodu nutnosti zpracování dat na jeho straně.

Mezi výhody však patří rychlost při nízkém souběžném použití, jelikož klient komunikuje přímo s datovým serverem a není tedy nutné čekat na prostředníka.



Obrázek 3.1: Architektura klient-server (převzato z https://managementmania.com/cs/architektura-klient-server

Třívrstvá architektura

Třívrstvá architektura je rozšířením architektury klient-server3.5, která přerozděluje povinnosti z původního rozdělení mezi klienta a server. Třívrstvá architektura implementuje tři vrstvy, které jsou na sobě nezávislé, a umožňují jednodušší údržbu a lepší rozšiřitelnost:

- Prezenční vrstva
- Aplikační vrstva

[10] https://en.wikipedia.org/wiki/WebSocket

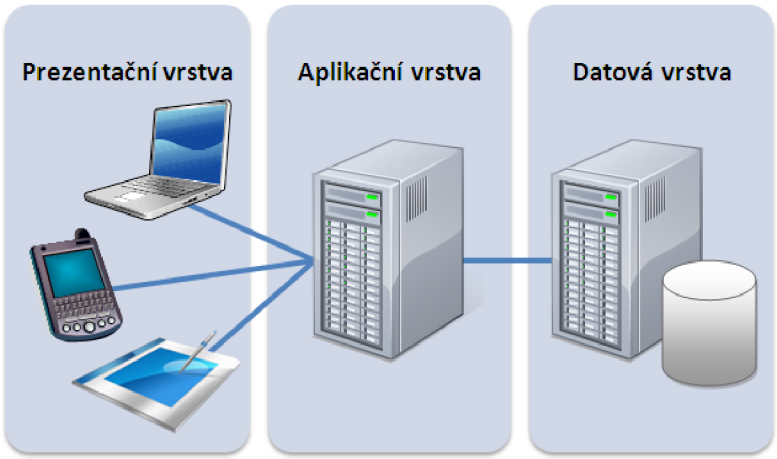
- Datová vrstva

Prezenční vrstva se nachází na straně klienta a má za úkol pouze zobrazovat data. Tato vrstva již nezpracovává a nevypočítává data.

Aplikační vrstva slouží jako zprostředkovatel mezi uživatelem a servem, kde probíhá výpočet a operace prováděné mezi vstupně-výstupními požadavky a daty.

Datová vrstva je nejnižší vrstva a zajišťuje práci s daty. Tedy základní datově-funkční operace.

Hlavní výhodou třívrstvé architektury je rozdělení vrstev tak, aby na sobě nebyly závislé, což umožňuje nezávislou práci na každé vrstvě, jednodušší údržbu a lepší rozšířitelnost.



Obrázek 3.2: Třívrstvá architektura (převzato z https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture

3.6 Jazyky vhodné pro vývoj

V nynějším světě se nabízí pro vývoj webové aplikace v rámci programovacích jazyků hned několik vhodných kandidátů. Každý z jazyků má kromě rozdílné syntaxe i jiné možnosti, rychlost a přístup k OOP (Objektově Orientované Programování). Pojednány jsou pouze jazyky, které se dají přímo využít k vývoji webové aplikaci [21].

JavaScript

JavaScript je vysokoúrovňový programovací jazyk, který je často používán pro vývoj interaktivních webových aplikací. Obvykle se používá pro práci na klientské straně aplikace, kde slouží k interakci s uživatelským rozhraním a dynamickému generování obsahu. Je navržen tak, aby byl přímo využíván při práci s HTML dokumenty, což z něj dělá ideální nástroj pro tvorbu webových aplikací.

JavaScript je široce používán v různých interaktivních aplikacích a umožňuje vývojářům vytvářet funkce, které zlepšují uživatelský zážitek. Mezi takové funkce mohou patřit například formulářové validace, animace, dynamické nabídky a podobně. JavaScriptový kód je často běžně používán na různých platformách a může být integrován do různých webových aplikací. [13][11]

[11] Článek je nazván "What is JavaScript?"

Python

Python je objektově orientovaný programovací jazyk, který má široké využití v oblasti webového vývoje, softwarového vývoje, skriptování a zpracování dat [5]. Mezi jeho velké přednosti patří jednoduchá syntaxe a čitelnost. Python je multiplatformní a často se používá pro rychlé prototypování aplikací.

Python je populární také pro vývoj serverových aplikací, zejména díky své schopnosti integrace s dalšími jazyky, jako jsou například C, C++ a Java. To umožňuje využití stávajících knihoven a nástrojů v těchto jazycích a značně zjednodušuje vývoj aplikací.

PHP

PHP je skriptovací jazyk, který se primárně věnuje webovému vývoji, a díky svým funkcím patří mezi nejvyužívanější jazyky pro tvorbu webových aplikací. PHP se využívá zejména na serverové straně aplikace, avšak umožňuje také přímou integraci skriptů na straně klienta. [16]

PHP je mnohokrát používán jako skriptovací jazyk pro vytváření dynamických webových stránek a aplikací. Mezi výhody jeho použití patří rychlost vývoje a možnost interakce s různými databázovými systémy, jako je například MySQL. PHP také nabízí řadu knihoven a frameworků, které usnadňují a zrychlují vývoj webových aplikací. Mezi ty nejznámější patří například Laravel, Symfony nebo CakePHP.

C#

C# je obecně použitelný programovací jazyk, který je jazykem pro framework *.NET* vyvinutý společností Microsoft. Díky svému původu vývojáři mohou využít bohatou nabídku učebních materiálů a podpory poskytované společností Microsoft [15]. C# také nabízí vynikající interoperabilitu s dalšími programovacími jazyky, jako jsou například C, C++, Java nebo JavaScript.

C# byl vytvořen jako součást platformy .NET a je jedním z hlavních jazyků pro vývoj aplikací pro tuto platformu. C# se vyznačuje statickou typovou kontrolou, což umožňuje vývojářům identifikovat chyby v kódu během kompilace a tím minimalizovat chyby v průběhu běhu aplikace. C# také nabízí vysokou úroveň abstrakce a modularitu, což umožňuje snadnou správu a úpravy velkých projektů.

Ruby

Ruby je dynamický, objektově orientovaný programovací jazyk, který je velmi oblíbený mezi vývojáři, zejména v oblasti webového vývoje, díky své elegantní syntaxi a jednoduchosti použití.

Ruby podporuje více paradigmat programování, jako jsou funkcionální a imperativní styl, což dává vývojářům více svobody a flexibilitu při řešení problémů. Ruby má také velmi rozvinutou knihovnu funkcí a modulů, které zjednodušují vývoj a umožňují vytváření vysoce kvalitního kódu.

Dále nabízí podporu pro různé frameworky, jako je například Ruby on Rails, což je jedním z nejpopulárnějších frameworků pro webový vývoj. Ruby on Rails zjednodušuje tvorbu webových aplikací a poskytuje vývojářům řadu funkcí a nástrojů, které urychlují vývoj a zvyšují kvalitu kódu.

3.7 Frameworky pro PHP

Kapitola se věnuje frameworkům pro PHP, které jsou souborem nástrojů, knihoven, pravidel a konvencí, které usnadňují a urychlují vývoj software. Tyto frameworky poskytují hotovou architekturu a strukturu, na kterou mohou programátoři navazovat svůj vlastní kód. Tím se ušetří čas na vytváření základních komponent aplikace a programátoři se mohou soustředit na programování vlastních funkcí.

Mezi populární frameworky pro PHP patří Laravel, Symfony a CakePHP.

Laravel

Laravel je open-source webový framework pro vývoj webových aplikací napsaný v programovacím jazyce PHP. Je navržen tak, aby usnadnil a urychlil vývoj moderních a robustních webových aplikací. Laravel je postaven na konceptu architektury MVC (Model-View-Controller) a nabízí ucelené a intuitivní API pro práci s relačními databázemi, autentizaci, validaci formulářů, a mnoho dalších funkcí [9].

Díky svým funkcím a vlastnostem se Laravel stal velmi populárním vývojovým frameworkem pro PHP a získal si velkou komunitu vývojářů, kteří přispívají k jeho rozvoji a podpoře. Laravel je také známý pro své rozšíření a rozšiřitelnost, umožňující vývojářům snadno integrovat další funkce a rozšíření do svých aplikací.

Symfony

Symfony je open-source webový framework napsaný v jazyce PHP, který pomáhá vývojářům vytvářet robustní, modulární a škálovatelné webové aplikace. Framework využívá architekturu Model-View-Controller (MVC), ale neyvužívá ji naplno [20]. Přináší mnoho užitečných funkcí pro usnadnění vývoje, jako jsou například automatické generování formulářů, validace dat, podpora různých databázových systémů a integrace s dalšími knihovnami a komponentami.

Symfony je velmi oblíbeným frameworkem v PHP komunitě díky své flexibilitě, modularitě a rozšířitelnosti. Jeho použití je často doporučováno pro větší projekty, kde je potřeba spravovat složitější logiku a komunikaci s databázemi. Symfony také nabízí kvalitní dokumentaci a aktivní komunitu vývojářů, kteří přispívají k jeho dalšímu rozvoji a vylepšování.

Jedná se o největší a nejkomplexnější framework pro PHP.

CakePHP

CakePHP je open-source webový framework pro vývoj aplikací v jazyce PHP. Byl vytvořen pro usnadnění vývoje aplikací pomocí konvencí a zahrnutím běžných funkcí, které jsou potřebné pro webový vývoj, jako jsou například autentizace, autorizace, validace dat a manipulace s databází. CakePHP staví na model-view-controller architektuře. CakePHP také nabízí podporu pro testování aplikací, což usnadňuje odhalování chyb a zajišťuje, že aplikace bude fungovat správně i po úpravách kódu. CakePHP má narozdíl od Symfony a Laravel menší komunitu uživatelů a může tak být těžší při prýáci s ním získat radu či najít pomoc.

3.8 Bezstavový a stavový přístup

Informace vycházejí z [3]. Stavy jsou proměnné, které se mohou měnit během běhu programu. Tyto stavy mohou být libovolného typu, ať už jednoduché příznaky, nebo složitější

objekty. Některé části aplikace vyžadují jeden z následujících přístupů k udržování stavu aplikace:

Bezstavový přístup

Bezstavový přístup je charakterizován tím, že ignoruje jakýkoliv stav, který se v aplikaci vyskytuje, a vždy reaguje stejným způsobem na stejné vstupy. Tento přístup je žádoucí v procesech, kde se očekává vždy stejný výstup. V rámci webového vývoje může být tento přístup uplatněn například při odpovědích na požadavky.

Stavový přístup

Na druhé straně stavový přístup udržuje stavy a reaguje na ně. To vede k různým výstupům v závislosti na stavu aplikace. Jedná se o vhodný přístup pro řešení dynamických požadavků, například při autentizaci, kde se v závislosti na tom, zda je uživatel přihlášen nebo ne, vrátí různé výsledky.

3.9 Předávání událostí

Předávání událostí (events) je nezbytné pro předávání informací o změnách v klientské části, aby byla aplikace dynamická. Události jsou zprávy, které jsou vyslány funkcí a informují o nějaké změně, která nastala. Další funkce poté mohou tyto události poslouchat a v závislosti na nich reagovat.

I když v některých případech stačí jednoduché poslouchání událostí, někdy by taková implementace byla příliš složitá. Proto existuje několik způsobů, jak tuto implementaci zjednodušit.

Sběrnice událostí (EventBus)

Sběrnice událostí je softwarová komponenta sloužící k předávání událostí. Sběrnice je neměnné centrum aplikace, která funguje jako most pro zprávy. Zpráva se obvykle pošle do sběrnice a ta ji zopakuje všem připojeným komponentám, které poté můžou reagovat na danou událost. Slouží jako prostředník mezi všemi softwarovými komponentami.

Úložiště stavů

V základu se jedná o jiný koncept, jenž slouží pro ukládání stavů aplikace. Avšak v dnešní době se úložiště stavů využívá i jako rozšířenější implementace sběrnice událostí 3.9. Úložiště stavů funguje odlišně od sběrnice, jelikož nevyužívá přeposílání zpráv všem komponentům, nýbrž pouze upravuje stav, který si ukládá v paměti. Tyto změny jsou následně odpozorovány komponentami, které aktivně odposlouchávají a čekají na změnu. Hlavním rozdílem oproti sběrnici je možnost ukládat data v úložišti stavů a provádět logiku nad uloženými daty.

Kapitola 4

Vybrané technologie a praktiky

V kapitole je uvedeno jaké technologie a praktiky [sekce 3] byly vybrány pro praktickou část Bakalářské práce.

4.1 Vybrané praktiky

Praktiky slouží k strukturalizaci kódů a definování postupů, které budou využívány.

Způsob prezentace

Pro implementaci aplikace a její prezentování byl zvolen typ SPA (Single Page Application) 3.1. Tento způsob prezentace aplikace přináší výhodu v uživatelské přívětivosti a snazší práci s aplikací. Vzhledem k tomu, že vyvíjená aplikace bude obsahovat interaktivní prvky a bude vyžadovat rychlé odezvy na uživatelské akce, bylo zvoleno SPA jako nejvhodnější řešení. SPA umožňuje uživatelům pracovat s aplikací bez nutnosti přebíhání mezi jednotlivými stránkami, což přispívá k vyšší uživatelské spokojenosti. Nicméně implementace SPA vyžaduje vzhledem ke své složitosti při vývoji vyšší úsilí v porovnání s jinými typy aplikací.

Návrhový vzor

Z návrhových vzorů byl vybrán MVC (Model-Pohled-Kontrolér), který poskytuje vhodné rozdělení aplikace a podporuje modulární návrh. Vzor MVC odděluje prezentaci od logiky aplikace, což zvyšuje flexibilitu a usnadňuje údržbu aplikace. Zároveň je MVC velmi rozšířený a běžně používaný vzor pro webové aplikace. Pro daný projekt je toto řešení vhodné, protože aplikace bude obsahovat několik modulů, které budou spolupracovat.

Aplikační rozhraní

Pro realizaci aplikačního rozhraní byl zvolen REST (Representational State Transfer) 3.4 s využitím formátu JSON pro serializaci dat. Zvolený přístup umožňuje snadnou a efektivní komunikaci mezi klientem a serverem a zvyšuje tak interoperabilitu aplikace. Zároveň je REST velmi rozšířený a běžně používaný přístup v moderních webových aplikacích. REST také poskytuje jednoduchý a konzistentní způsob, jak navrhovat API (Application Programming Interface). JSON je velmi populární a rozšířený formát pro výměnu dat a jeho použití umožňuje snadné zpracování dat v aplikaci. Daný přístup je relevantní pro tento projekt, protože aplikace bude obsahovat rozsáhlou komunikaci s databází.

4.2 Správci knihoven

Správci knihoven slouží k jednoduchému spravování stažených knihoven. Starají se o jejich stažení, aktualizovaní a zajištění všech dalších potřebných balíčků a knihoven pro chod knihovny.

NPM

Node Package Manager (NPM) je open-source softwareový nástroj, který slouží k správě balíčků v prostředí Node.js. To je prostředí pro běh JavaScriptu mimo webový prohlížeč, který umožňuje vývojářům vytvářet výkonné a škálovatelné webové aplikace. NPM poskytuje uživatelům přístup k centrálnímu repozitáři, který obsahuje tisíce balíčků s otevřeným zdrojovým kódem, včetně modulů, knihoven a dalších zdrojů. NPM umožňuje uživatelům snadno stahovat a instalovat tyto balíčky, aktualizovat je a spravovat jejich závislosti. Nástroj je důležitou součástí ekosystému Node.js a je často používán v oblasti webového vývoje.

NPM je také důležitou součástí ekosystému Vue.js, moderního JavaScriptového frameworku pro tvorbu interaktivních webových aplikací. Vue.js využívá NPM k instalaci potřebných balíčků a závislostí, což umožňuje vývojářům rychle a efektivně vytvářet složité aplikace. NPM je tedy klíčovým nástrojem pro vývojáře, kteří pracují s Vue.js, a pomáhá jim urychlit a usnadnit proces vývoje webových aplikací.

Composer

Composer je softwarový nástroj pro správu závislostí v jazyce PHP. Je to open-source projekt, který umožňuje PHP vývojářům snadno instalovat a aktualizovat závislosti svých projektů, jako jsou knihovny, frameworky, balíčky a další.

Composer funguje na principu definování balíčků v souboru composer.json, který obsahuje seznam závislostí, verze a další metadata. Poté, co je soubor composer.json definován, lze spustit příkaz `composer install`, který stáhne a nainstaluje všechny potřebné balíčky a jejich závislosti.

Composer také umožňuje vývojářům spravovat závislosti pro různé verze PHP a instalovat různé verze knihoven a balíčků pro různé projekty. To znamená, že Composer umožňuje vývojářům flexibilitu v tom, jaké balíčky a závislosti použijí v každém projektu.

V dnešní době je Composer běžně používán v komunitě PHP vývojářů a je klíčovým nástrojem pro správu závislostí v PHP projektech.

4.3 PHP – Laravel

Laravel je open-source webový aplikační framework, který je napsán v PHP. Jeho hlavním cílem je usnadnit a zrychlit vývoj webových aplikací pomocí elegantní syntaxe, intuitivního API a mnoha zabudovaných funkcí. Laravel poskytuje mnoho nástrojů pro rychlý vývoj webových aplikací, jako například routování, řízení stavu sezení, práci s databázemi, uživatelskou autentizaci, třídy pro práci s e-maily, front-endové nástroje a mnoho dalšího.

Laravel je navržen tak, aby byl snadno rozšiřitelný a přizpůsobitelný potřebám vývojářů. Obsahuje mnoho předpřipravených balíčků, které usnadňují vývoj, ale také umožňuje vývojářům vytvářet a sdílet vlastní balíčky. Laravel je také dobře dokumentován a má aktivní komunitu, což znamená, že je k dispozici mnoho zdrojů a návodů pro vývojáře, kteří se s ním chtějí naučit pracovat.

4.4 Vue.js 3

Vue.js je moderní a dynamický JavaScriptový framework určený pro vytváření webových uživatelských rozhraní (UI). Jeho hlavním cílem je umožnit vývojářům vytvářet interaktivní webové aplikace s lehkostí a efektivitou.

Vue.js využívá deklarativní přístup k vytváření UI, což znamená, že programátor popisuje, jak by měl UI vypadat a chovat se, a framework se postará o to, jak toho dosáhnout. Vue.js také nabízí širokou škálu funkcí a knihoven, které usnadňují práci s různými technologiemi, jako jsou například animace, komunikace se serverem, správa stavu a mnoho dalšího.

Vue.js je velmi flexibilní a může být použit samostatně nebo v kombinaci s dalšími technologiemi a frameworky. Je navržen tak, aby byl snadno použitelný pro jakýkoli typ projektu, od malých až po velké webové aplikace.

Inertia

Inertia.js je open-source nástroj, který umožňuje spojit moderní frontendový framework (např. Vue.js nebo React) s backendovým frameworkem (např. Laravel) tak, aby se daly využít výhody obou světů bez nutnosti psát zbytečně mnoho kódu.

Konkrétně Inertia.js poskytuje jednoduché API pro vytváření stránek a komponent s použitím frontendového frameworku, které se poté mohou zobrazovat v backendové aplikaci bez nutnosti opakovaného načítání celé stránky. To umožňuje vytvořit rychlejší a plynulejší uživatelské rozhraní, protože uživatelé nebudou muset čekat na načítání celé stránky při každém přechodu.

Inertia.js také umožňuje využít všechny funkce backendového frameworku, jako je například routing, middleware nebo autentizace, přičemž frontendový framework může být integrován přímo do HTML šablon backendu.

Inertia byla vybrána právě pro nejlepší kombinaci s frameworkem Laravel a pro jeho přímou podporu.

Vite

Vite je moderní nástroj pro vývoj webových aplikací, který umožňuje v reálném čase zobrazovat změny provedené na kódu. Vite je postaven na technologii ESM (ECMAScript Modules), což umožňuje efektivnější zpracování a optimalizaci kódu.

Pinia

Pinia je oficiální stavové řízení pro Vue.js a novou alternativou pro bývalé oficiální řízení Vuex [22]. Pinia staví na základech Vuex, ale je stavěna tak, aby byla intuitivnější k použití. [7] Její Intuitivní API umožňuje vývojářům s spravováním stavu pomocí reaktivního způsobu.

VueUse

VueUse je knihovna pro Vue.js, která obsahuje řešení nejčastějších problémů při vývoji webových aplikací. Tato knihovna poskytuje užitečné utilitní funkce a komponenty, které mohou zlepšit výkonnost a efektivitu vývoje. VueUse tak umožňuje vývojářům se zaměřit na tvorbu funkcionalit samotné aplikace, místo aby řešili opakující se problémy. Výsledkem je rychlejší vývoj a snížení chyb v kódu. [8]

Draggable

Vue3-draggable[1] je komunitní knihovna pro Vue3, která slouží pro jednoduché zpracování vizuální uživatelské možnosti přenosu jednotlivých položek mezi kontejnery a zároveň k aktualizací dat i v rámci uložení.

4.5 TailwindCSS

TailwindCSS je moderní CSS framework, který umožňuje snadné a rychlé psaní CSS stylů pro webové stránky a aplikace. Je navržen tak, aby urychlil vývoj webových aplikací tím, že poskytuje sadu předdefinovaných CSS tříd, které se dají použít pro rychlé vytváření rozvržení, typografie, barev, efektů a dalších designových prvků. Tento framework je často používán v kombinaci s frameworky jako Laravel a Vue a výrazně usnadňuje proces designu a vývoje webových stránek a aplikací.

HeadlessUI

HeadlessUI[2] je open-source knihovna komponent pro React a Vue, která poskytuje předem navržené a optimalizované komponenty pro vývojáře webových aplikací. Tyto komponenty jsou navrženy tak, aby byly plně přizpůsobitelné a snadno použitelné pro tvorbu vlastních uživatelských rozhraní.

Co je zajímavé na této knihovně, je to, že jsou komponenty navrženy tak, aby byly "headless", což znamená, že jsou odděleny od vizuálního stylu a je na vývojáři, aby je přizpůsobil svému konkrétnímu stylu. To poskytuje vývojářům maximální flexibilitu při tvorbě uživatelských rozhraní bez nutnosti přepisování nebo rozšíření existujících komponent.

4.6 MySQL

MySQL je open-source relační databázový systém, který umožňuje ukládání a zpracování dat [14]. Je založen na SQL a přidává funkce pro manipulaci s daty. Jeho vlastnosti zahrnují podporu pro transakce, bezpečnost, škálovatelnost a vysokou výkonost. MySQL je vyvíjen společností Oracle Corporation a je k dispozici zdarma pro většinu účelů použití.

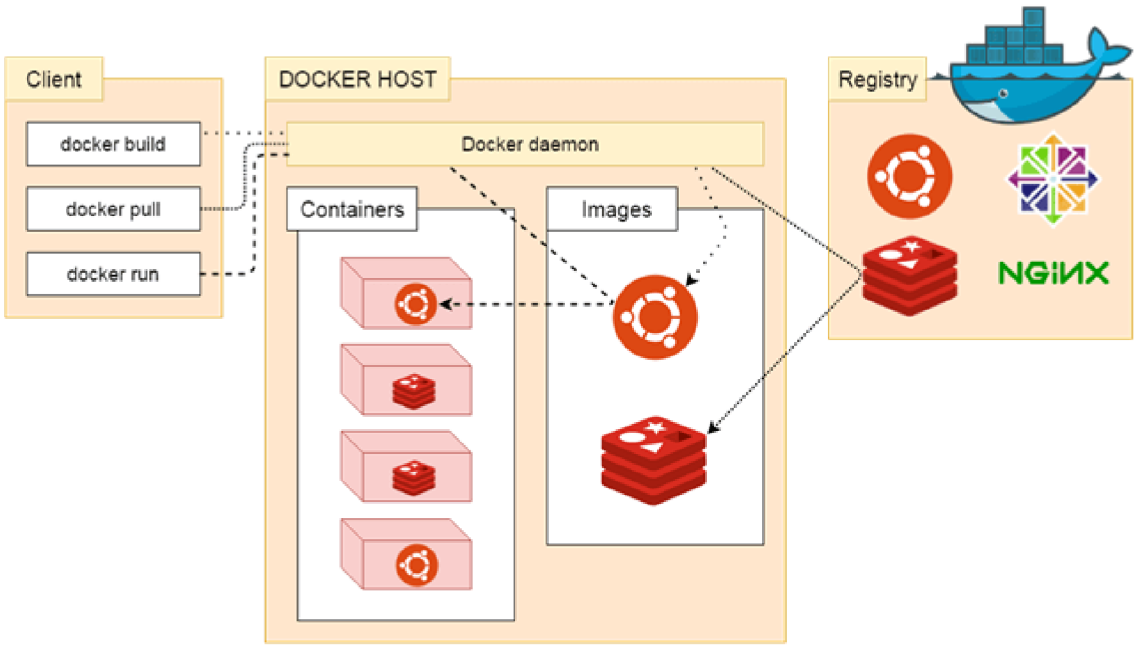
4.7 Kontejnerové služby

Momentálně existují dvě kontejnerové služby a to jsou `Docker` a `Podman`[3]. Obě pro uživatele fungují stejně a mění se pouze ve vnitřní implementaci. Zvolen byl `Docker`.

Docker kontejnery umožňují oddělení aplikace od prostředí, na kterém běží. To znamená, že lze spouštět aplikace na různých počítačích a serverech bez nutnosti konfigurace prostředí na každé z těchto platforem. [6]

Docker funguje na základě Dockerfile, což je soubor instrukcí, který popisuje, jak vytvořit kontejner s danou aplikací. Obsahuje příkazy pro instalaci potřebných závislostí a aplikací, nastavení konfigurace a další specifické kroky potřebné pro přípravu běhového prostředí aplikace. Po vytvoření Dockerfile se aplikace sestaví do tzv. obrazů systému 4.1. [6]

[1] https://www.npmjs.com/package/vue3-draggable
[2] https://github.com/tailwindlabs/headlessui
[3] https://podman.io/



Obrázek 4.1: Architektura Docker (převzato z [6])

Docker umožňuje rychlé a snadné nasazení aplikací na cloudové služby nebo virtuální servery, což snižuje náklady a zvyšuje flexibilitu při vývoji a nasazování aplikací.

Docker Compose

Docker Compose je nástroj pro správu více Docker kontejnerů jako jednoho celku. Umožňuje definovat a spustit aplikace, které se skládají z více služeb, každá běžící v samostatném kontejneru. Docker Compose také umožňuje nastavit různá prostředí, propojení kontejnerů a předání konfigurace služeb pomocí proměnných prostředí. Tento nástroj je velmi užitečný pro vývojáře, kteří potřebují spouštět a testovat složité aplikace v izolovaném prostředí. [1]

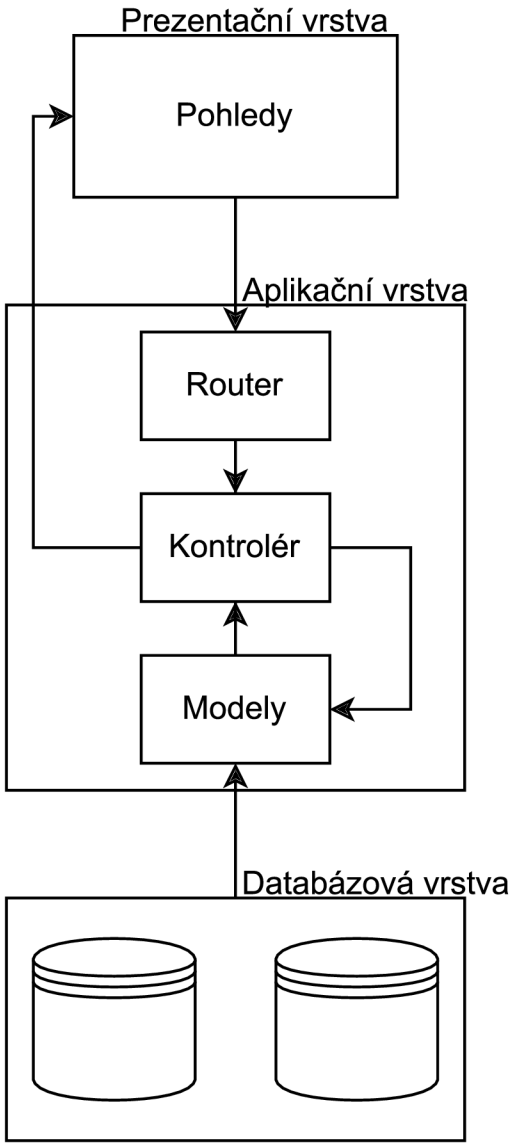
Kapitola 5

Návrh

Správný návrh architektury je klíčový pro správný vývoj, rozšířitelnost a udržitelnost systému. Následující kapitola představí zvolený návrh aplikace, která obsahuje architekturu, datový návrh v podobě Databázového diagramu a návrh uživatelského rozhraní.

5.1 Architektura

Pro informační systém byla zvolena architektura třívrstvá 3.5 5.1, a to kvůli možnostem jednoduché rozšířitelnosti, větší bezpečnosti a jednoduší údržbě. V rámci architektury budou vytvořeny dvě datové vrstvy. První bude sloužit jako hlavní datový zdroj pro chod aplikace, druhá bude sloužit jako zdroj dat o uživatelích, kteří budou ukládáni v rámci jiného systému. Prezentační vrstva bude vyvinuta pomocí frameworku Vue.js 4.4 pro webové rozhraní. Aplikační vrstva bude postavena na frameworku Laravel 4.3, který bude provádět veškeré výpočty a úpravy v datech. Pro webovou prezentační vrstvu bude využit návrhový vzor MVC 4.1, který ještě více rozdělí odpovědnosti jednotlivých částí.



Obrázek 5.1: Návrh architektury pro informační systém

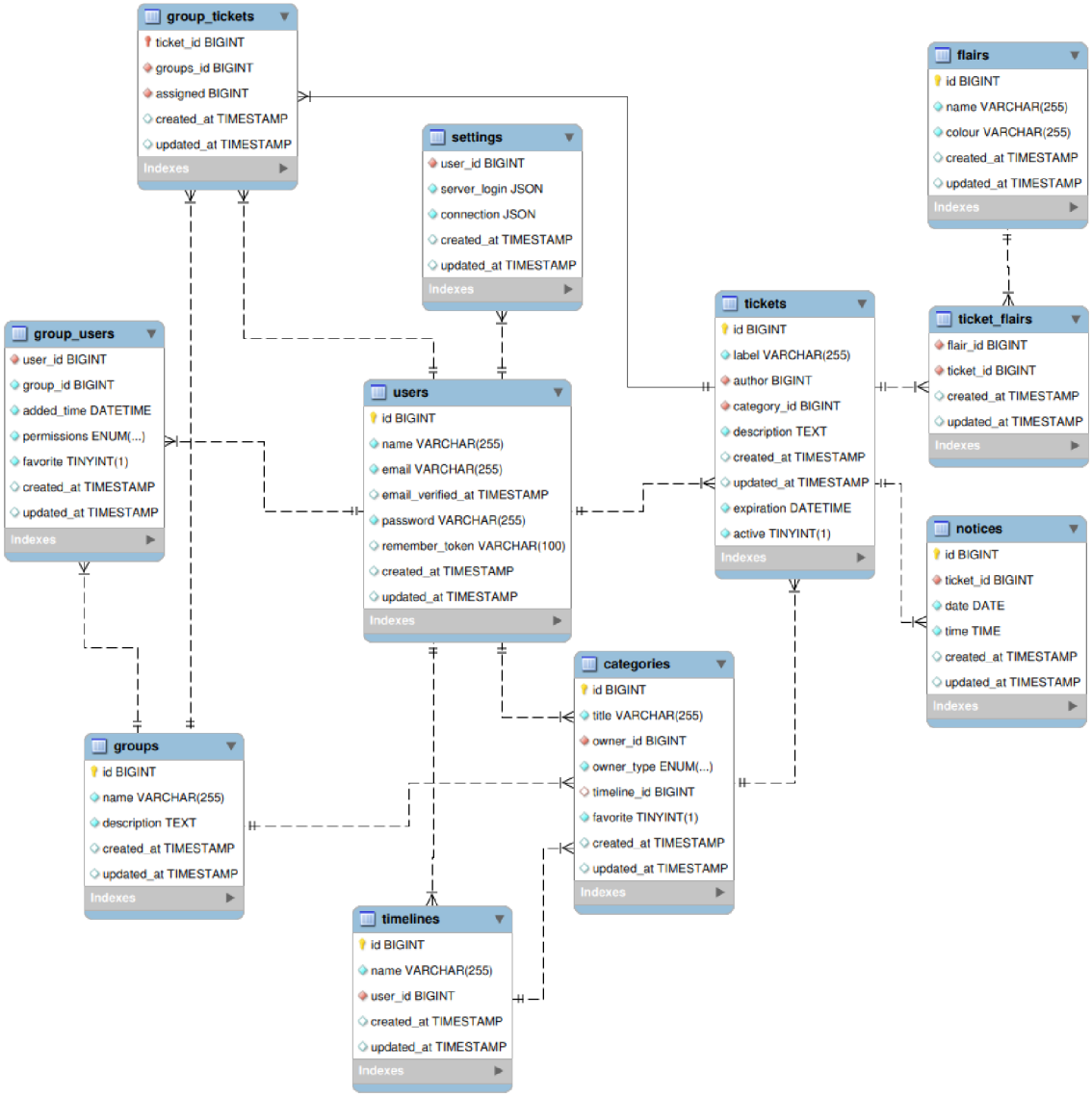
5.2 Databázový návrh

Návrh databáze, jejich entit a vztahů je důležitá a udává konečnou účinnost zpracování dat. Pokud by databáze byla navrhnuta a implementována nevhodně, celá implementace aplikační vrstvy by mohla být těžší a efektivita aplikace by se snížila.

ER Diagram

ER (Entity Relationship) Diagram je grafická reprezentace, která slouží k zobrazení vztahů mezi jednotlivými entitami [23] návrhu databáze. ER obsahuje několik důležitých prvků, které se používají k modelování databáze. Patří mezi ně **Entita**, což je reprezentace objektu nebo události. Každá Entita má určené **Atributy**, které popisují danou entitu a data, která

o ní známe. Mezi nejdůležitější prvky patří **Vztahy** mezi entitami, které určují, jak dané entity spolu logicky souvisí.



Obrázek 5.2: ER diagram

Entity použité v systému.

Uživatel (User)

Uživatel zde bude využit, pouze pokud systém nebude mít vlastní tabulku uživatelů. Ukládá si pouze základní informace, které se můžou lišit od nasazení.

Připomínka (Ticket)

Připomínka má vlastní ID, Název a Popis. Také si ukládá autora, což je přímý vztah s tabulkou Uživatele. Každá připomínka má pouze jednoho autora. Další atributy, které

vlastní, je příznak aktivity, který detekuje, zda daná připomínka již byla smazána, datum expirace, které určuje její datum vypršení, a poslední datum uvádějící, do kdy má být připomínka zapsána.

Upozornění (Notice)

Tabluka Upozornění slouží k uložení všech časových okamžiků, kdy chce být uživatel upomenut. Ukládá si pouze připomínku, ke které patří datum a čas provedení.

Skupina (Group)

Skupina uživatelů má název a svůj popis. Dále je používána pouze jako identifikátor pro rozdělení uživatelů. Skupina je navázána na uživatele pomocí spojovací tabulky, která si ukládá datum přidání každého uživatele a jeho pravomoc v rámci skupiny.

Skupinová připomínka (Group Ticket)

Skupinová připomínka je *rozšířením* Připomínky a přidává relaci skupiny a momentálně přiřazeného uživatele k této připomínce.

Označení (Flair)

Označení slouží k označení připomínky pomocí určitých klíčových slov a má za atribut pouze jméno a barvu. S Připomínkami je spojena pomocí spojovací tabulky.

Kategorie (Category)

Kategorie má několik připomínek a také náleží k určité nástěnce, nebo skupině. Podle toho komu náleží, buďto odkazuje k majiteli jako k uživateli, nebo ke skupině.

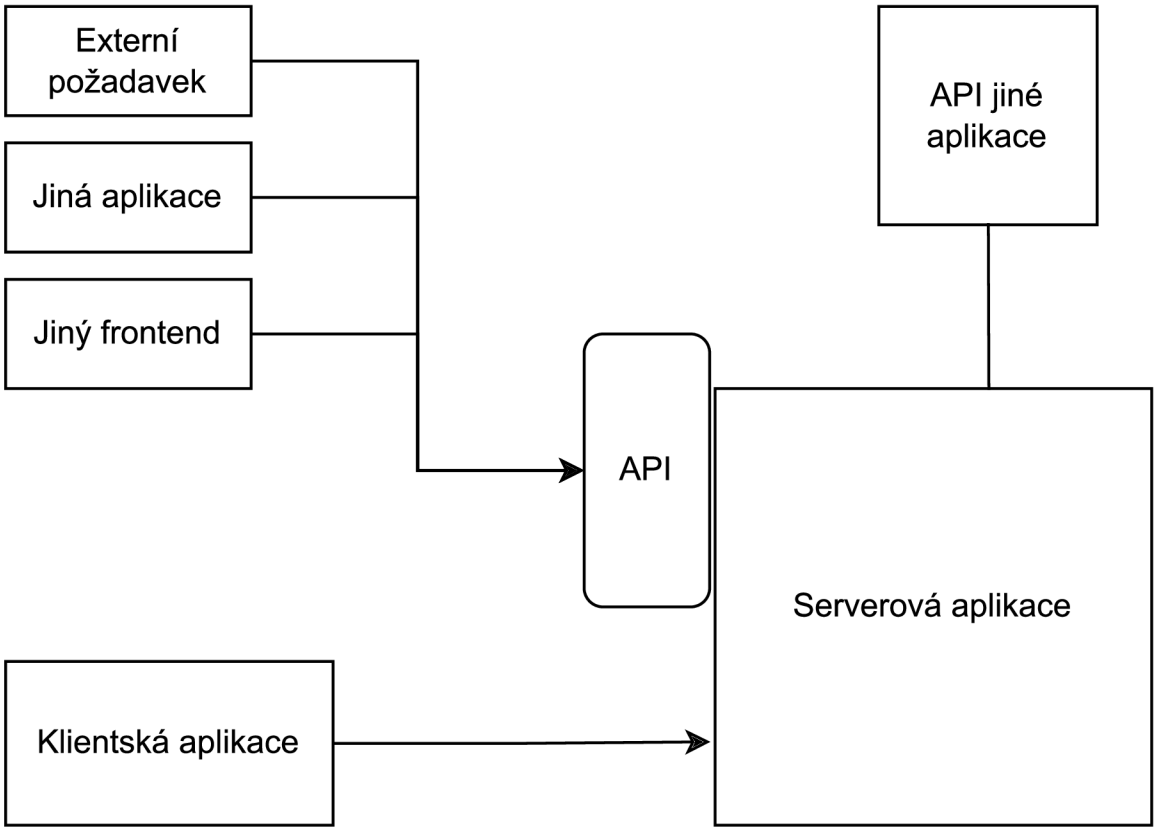
Nástěnka (Timeline)

Nástěnka vlastní několik kategorií a má jednoho vlastníka. Každá nese vlastní jméno.

5.3 Návrh API

Součástí aplikace by mělo být i API (Application Programming Interface), které by mělo fungovat pro jinou externí komunikaci s aplikací, než pomocí webového rozhraní. Pro návrh je využito technologie REST API 3.4.

Návrh počítá s takovým API, které by dovolilo vyvinout nezávislou klientskou část, a proto z velké části bude kopírovat již existující cesty, které slouží pro momentální implementaci webového pohledu.



Obrázek 5.3: Komunikace s aplikací

5.4 Způsob nasazení

Projekt byl navržen s ohledem na kompatibilitu se stávajícím systémem Modin a jeho uživatelskou databází, což bylo důležité kritérium pro splnění zadání. To umožní snadnou integraci existujících uživatelů do nové služby.

Z důvodu bezpečnostních a právních požadavků bude aplikace fungovat na lokální síti. Tento faktor by však neměl ovlivnit celkovou implementaci a aplikace by stále měla implementovat nejbezpečnější a nejnovější praktiky.

Z důvodu jednoduché údržby a přívětivější správy bude aplikace využívat služby `Docker`, který vytvoří pro každou službu systému vlastní prostředí. Tímto se předejde problémům s kompatibilitou, aktualizacemi a vznikne další bezpečnostní vrstva. Zároveň se tímto předejde i některým možným problémům s nasazením aplikace.

Pro zajištění snadné přenosnosti budou do projektu zahrnuty proměnné prostředí, které umožňují definovat důležitá data pro chod aplikace. Tyto proměnné umožňují uživatelům jednoduše konfigurovat a přizpůsobit službu dle svých potřeb, zároveň opět snižují potenciální problémy s nasazením.

5.5 Návrh uživatelského rozhraní

Záměrem je uživatelské rozhraní, které bude maximálně přívětivé pro uživatele. Počáteční návrh může být upraven v závislosti na změně požadavků uživatele. Pro návrh uživatelského

rozhraní se využívají drátové modely, které umožňují představit uživatelský pohled bez funkčnosti.

Pro vytvoření drátových modelů se využívají specializované nástroje, jako například Figma nebo Sketch. Tyto nástroje umožňují vytvoření detailních návrhů uživatelského rozhraní, včetně interaktivních prototypů, které umožňují uživatelům simulovat používání aplikace.

V návrhu uživatelského rozhraní je důležité zohlednit také principy uživatelského designu, které pomáhají zlepšit uživatelskou přívětivost a usnadnit uživatelům interakci s aplikací. Dané principy zahrnují například jednoduchost, konzistenci, zřetelnost a přizpůsobivost. [12]

Dalším důležitým prvkem návrhu uživatelského rozhraní je zajištění jeho responzivního designu, tedy schopnosti přizpůsobit se různým velikostem obrazovek a zařízení. To je důležité zejména v době, kdy se čím dál více lidí připojuje k internetu přes mobilní zařízení.

Celkově je návrh uživatelského rozhraní důležitou součástí vývoje aplikace, která pomáhá zlepšit uživatelskou zkušenost a zvyšuje pravděpodobnost úspěchu aplikace na trhu.

Hlavní pohled

Hlavní pohled zobrazuje navigační menu a přehled. Uživatel si může vybrat, zda chce vytvořit osobní nebo skupinovou připomínku. Po vyplnění údajů následně připomínku vytvoří. Dále má možnost zobrazit svoji osobní nebo skupinovou nástěnku či si vybrat oblíbenou kategorii.

Při vybrání nástěnky se uživateli zobrazí přehled všech kategorií, které pod ní patří. Po otevření připomínky se uživateli zobrazí detailní pohled s možností připomínku upravit. Záměrem je celkové uživatelské rozhraní, které je intuitivní a jednoduché na použití.

Obrázek 5.4: Návrh uživatelského rozhraní [Wireframe]

Kapitola 6

Implementace

Kapitola se zabývá konečnou implementací, nejdříve je popsána obecná implementace a následně konkretní implementační části.

6.1 Implementované funkce

Sekce přibližuje funkce, které jsou implementovány ve finální verzi praktické části Bakalářské práce.

Připomínky

Připomínky jsou klíčovým prvkem celého projektu. Každá připomínka je charakterizována svým názvem, který slouží jako stručné shrnutí a její popis. Připomínky se dále zařazují do kategorií a tabulí, což umožňuje uživatelům snadno organizovat své úkoly a sledovat jejich stav.

Připomenutí

Kromě základních vlastností, jako je jméno a kategorie, mohou připomínky také obsahovat další informace. Například může mít připomínka nastavenou upomínku, které upozorní uživatele na nadcházející termín splatnosti. Připomenutí jsou zatím implementována jako upozornění, které se spustí den před datem splatnosti, kdy bude zaslána uživatelům prostřednictvím e-mailu upomínka.

Osobní připomínky

V projektu jsou osobní připomínky, které jsou určeny pouze pro konkrétního uživatele a nelze je sdílet s ostatními. Ty slouží k organizaci osobních úkolů a úkolů, které nejsou relevantní pro ostatní uživatele.

Skupinové připomínky

Dále jsou implementované Skupinové připomínky, které mohou být sdíleny mezi uživateli, pokud patří do stejného týmu. Každý tým má svou vlastní nástěnku, která může obsahovat neomezený počet kategorií. Lze k nim též připojovat soubory pomocí FTP serveru. Řešení umožňuje týmům spolupracovat na úkolech a snadno sdílet informace.

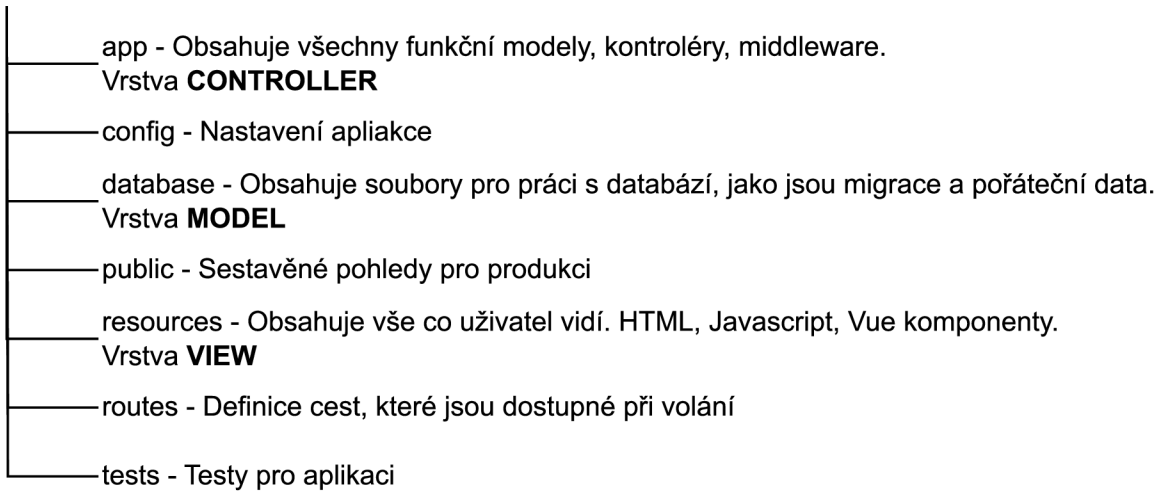
6.2 Implementace serverové části

V dílčí kapitole následuje obecný přehled implementace, který přináší náhled nad celkovou implementací. Popisovány jsou komponenty, workflow a vybrané součásti.

Rozložení souborů

Rozložení souborů odpovídá stavu na následujícím obrázku, jednotlivé položky jsou následně přiblíženy.

.



Obrázek 6.1: Rozložení souborů

Konfigurace

Vše začíná konfigurací, kde se nachází veškeré nastavení pro aplikaci. Mezi takové nastavení patří například připojení databáze, e-mailového klienta, adresy serveru a další. Tyto konfigurace se nachází ve složce `config` a napomáhají nastavitelnosti a přizpůsobitelnosti aplikace. V konfiguraci se často využívají proměnné prostředí, které jsou definované v každém systému samostatně, nebo můžou být definovány v souboru `.env`. Definice v souboru může být ovšem nebezpečná z důvodu možného úniku dat.

Databáze

Databáze je generována pomocí migrací. Ty jsou rozděleny do několika souborů, které jsou obvykle uloženy ve složce `database/migrations`. V migracích jsou definované akce generování, úprav a mazání tabulek a jejich atributů. Zároveň jsou zde i definované vztahy mezi tabulkami. Toto umožňuje mít jednu implementaci pro různé databázové systémy, jako jsou například MySQL, Oracle nebo PostgresSQL. Využívaný databázový systém je definovaný pomocí konfigurace 6.2.

Eloquent

Laravel je vybaven objektově orientovaným mapovacím nástrojem Eloquent (ORM), který umožňuje snadnou interakci s databází a poskytuje vývojářům jednoduchý přístup ke všem databázovým operacím bez nutnosti složitých dotazů. ORM Eloquent se stará o převod

objektů aplikace na řádky v databázi a zpět, což umožňuje snadnou manipulaci s daty v aplikaci. Díky tomu mohou vývojáři rychle a efektivně pracovat s databází bez nutnosti ručního psaní SQL dotazů. [4]

Modely

V rámci Laravel frameworku jsou modely definovány v podadresáři `app/models`, kde každý model má svůj vlastní soubor s definicí. Modely slouží k reprezentaci jedné tabulky v databázi, nebo logické jednotky více tabulek. Díky své přizpůsobivosti umožňují definovat řadu atributů, které usnadňují vývoj aplikace. Tyto mohou zahrnovat samotné atributy modelu, atributy skrývající určité informace během vracení modelu a další typy. Také lze nastavit, jakým připojením se má s modelem komunikovat. V rámci projektu je tato možnost využita pro propojení s jiným připojením v případě tabulky uživatelů v systému MODIN. Hlavním cílem definice modelu jsou však metody 6.1, které jsou opětovně použitelné pro daný model. Ty umožňují definovat, jaké údaje se mají vrátit a v jaké podobě. Díky Eloquent ORM je také možné definovat vztahy mezi modely, které reprezentují relace mezi jednotlivými tabulkami v databázi.

```
    # Definice vsupnich parametru a jmena
    public static function get($user_id, int $category_id)
    {
        # Definovani SQL dotazu pomoci Eloquent
        $tickets = Ticket::where('author', $user_id)->with(['extended' =>
        function($q) use ($category_id) {
            $q->where('category_id', $category_id);
        }])->has('extended')->get();
        return $tickets;
    }
```

Výpis 6.1: Ukázka metody modelu

Poté, co jsou nad modely definovány potřebné metody, jsou tyto metody následně využity v **kontroléru** příslušícímu danému modelu. Důležité je však také dbát na optimalizaci dotazů, aby nedocházelo k zbytečnému opakování dotazů nebo aby nebyl navrácen příliš velký počet výsledků, který by následně musel být oříznut až v aplikaci.

Kontroléry

Kontroléry jsou klíčovým prvkem architektury MVC, které mají za úkol správně reagovat na dotazy od uživatele. V rámci kontroléru jsou definovány funkce, které slouží jako delegátory pro metody modelů a přidávají další logiku, pokud je to potřebné.

Každý logický prvek nebo model má svůj vlastní kontrolér, který zajišťuje správnou obsluhu příslušných dotazů. Je velmi důležité zajistit, aby kontrolér byl bezstavový, tj. aby na každý dotaz odpovídal nezávisle na předchozích dotazech, a aby nedocházelo k nechtěným vedlejším efektům při opakovaném volání stejného dotazu.

Na straně Kontroléru je také možné hlídat vstupní data pomocí validace vstupních dat a naopak kontrolovat výstupní data. Vstupní data je nutno hlídat například při vytvoření nového tiketu, aby byla všechna potřebná pole vyplněná správně, protože při nesprávném či nedostatečném vyplnění požadavku by došlo k chybě programu nebo nekonzistenci dat.

Také je potřeba vstupní data kontrolovat z důvodů pokusů o nabourání systému například pomocí `SQL Injections`[1] nebo `XSS`[2] útoků.

Router

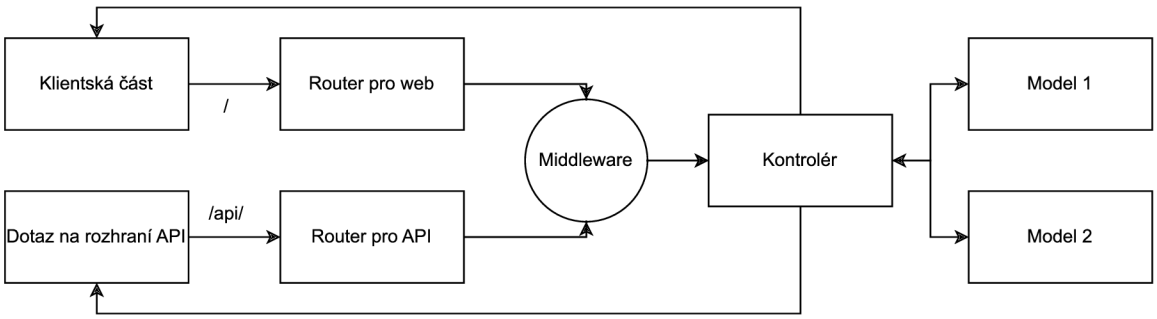
Router je v aplikaci Laravel zodpovědný za správné směrování dotazů na příslušný kontrolér. Definuje koncové body (Endpoints), které jsou dostupné v aplikaci, a určuje, které HTTP metody jsou povoleny pro daný koncový bod. Tím umožňuje využívat stejné URL pro různé akce.

Kromě toho umožňuje definovat middleware pro každý koncový bod, což jsou funkce, které jsou spuštěny před odesláním požadavku na kontrolér. Middleware lze použít k ověření autentizace uživatele nebo ke zpracování požadavku před jeho předáním kontroléru 6.2. Výhodou použití middleware je, že se snižuje duplikace kódu, a zvyšuje se znovupoužitelnost kódu v aplikaci.

```
1 Route::controller(\App\Http\Controllers\GroupController::class)->middleware(['auth'])->
      group(function () {
2     Route::get('/groups', 'get_list');
3     Route::get('/group/{id}', 'show');
4     Route::post('/group_add', 'create');
5     Route::put('/group_update', 'update');
6     Route::delete('/group_delete', 'destroy');
7 });
```

Výpis 6.2: Definice cest za pomocí skupin a volání middleware

Laravel rovněž umožňuje seskupování cest s cílem zlepšit čitelnost kódu. Tento proces je prováděn definováním kontroléru a middleware pro skupinu následovaným definováním jednotlivých cest pro jednotlivé metody 6.2. Tímto způsobem může být kód organizován do logických skupin a lze tak zlepšit jeho srozumitelnost a údržbu.



Obrázek 6.2: Cesta dotazu skrz aplikaci, při použití MVC

Aplikační rozhraní

API 4.1 bylo implementováno tak, aby kopírovalo cesty, které se využívají v klientské části a bylo přidáno několik koncových bodů, které by pro budoucí vývoj mohly být důležité. Cílem je napomoci rozšiřitelnosti a uživatelské přívětivosti aplikace, protože si tak uživatel může dovolit implementovat vlastní nástroje nad aplikací.

Všechny koncové body začínají ve stylu `{hostname}.{domain}/api/`

[1] https://en.wikipedia.org/wiki/SQL_injection
[2] https://owasp.org/www-community/attacks/xss/

Příklady koncových bodů

- GET: `ticket/{id}`
- GET: `ticket_short/{id}`
- POST: `ticket_add`
- PUT: `ticket_update/{id?}`
- GET: `group/{id}`
- POST: `group_add`
- PUT: `group_update`
- DELETE: `group_delete`
- POST: `category_add`
- GET: `category/{id}`

Mezi další aspekty aplikace, které stojí za pozornost, patří například využití nepovinných parametrů v metodě `ticket_update`, kdy je v těle zprávy dostatečný obsah identifikující tiket a není nutné jej zahrnovat v hlavičce. Dále lze použít metodu `PUT` pro archivaci místo mazání připomínek. Samotná připomínka tak zůstává v systému a pouze se aktualizuje její stav, což lépe odpovídá charakteru požadované operace.

6.3 Implementace klientské části

Klientská část se nachází v složce `resources` a obsahuje vše pro potřebu vykreslení klientských pohledů od stylů, přes **HTML** po jednotlivé JavaScripty. Jelikož je používáno **Vue3.js** 4.4 objevují se zde i komponenty, které jsou hlavním stavebním kamenem pro stavbu pohledů pomocí Vue3.

Komponenty

Komponenty ve Vue.js jsou samostatné, znovupoužitelné části uživatelského rozhraní, které se skládají z HTML šablony, JavaScriptové logiky a CSS stylů. Tyto komponenty lze poté opakovaně použít v různých částech aplikace a umožňují tak snadnou údržbu a škálovatelnost kódu. Komponenty mohou mít své vlastní stavy a propojení, což umožňuje vývojářům psát modulární kód, který lze snadno rozšířit a upravit.

Jedna z významných komponent aplikace je komponenta kategorie, což je dynamický kontejner umožňující zobrazování jednotlivých připomínek. Pro správné fungování dané komponenty bylo nezbytné implementovat možnost přesouvání jednotlivých připomínek a aktualizaci dat udržovaných na pozadí v závislosti na těchto změnách. Další informace o implementaci funkcionality lze najít v konkrétních implementačních případech v kapitole 6.4.

Šablony

Vue využívá pro vytváření komponent tzv. Šablony, které umožňují definovat vzhled uživatelského rozhraní v HTML syntaxi a propojovat je s funkcionalitou JavaScriptu. Šablony jsou poté kompilovány Vue.js do virtuálního DOM, což je abstraktní reprezentace struktury HTML stránky, kterou Vue používá k manipulaci s obsahem stránky. [10]

V šablonách můžeme využívat direktivy Vue.js, které umožňují vkládat dynamické obsahy, manipulovat s atributy HTML elementů, provádět iterace, vkládat podmínky a mnoho dalšího. Tím umožňují vytvářet bohatá a dynamická uživatelská rozhraní.

Díky oddělení HTML kódu od JavaScriptu, který zajišťuje logiku a funkcionalitu, se šablony stávají snadno udržovatelné a rozšiřitelné.

Sloty (angl. slots) jsou speciálními místy v šablonách, které umožňují dynamické doplnění generovaného obsahu. Sloty mohou být dále specifikovány a propojovány s dalšími komponentami, čímž se dosahuje maximální flexibility a znovupoužitelnosti komponent. Tento mechanismus umožňuje vývojářům jednoduše upravovat a rozšiřovat šablony bez nutnosti vytvářet nové komponenty a tím zvyšovat efektivitu vývoje.

Rozložení

Komponenty v rámci Vue umožňují vytvářet nejen jednotlivé části uživatelského rozhraní, ale také podporují vytváření celých rozložení pro dané komponenty. Tento přístup je pro vývoj stránek ve stylu SPA (Single Page Application) velmi důležitý, neboť umožňuje udržovat jedinou stránku a za běhu změnit pouze doplnitelné části rozložení pomocí slotů.

Správné rozložení nám umožňuje implementovat aplikaci ve stylu SPA. Hlavním přínosem je využití slotů ve frameworku Vue, které poskytují prostor pro dynamické vkládání dalších komponent.

V rámci dané implementace je hlavní rozložení stránky načteno vždy jako první a obsahuje navigační panel, který je stále přístupný. To umožňuje rychle a efektivně měnit pouze obsah hlavního panelu bez nutnosti opakovaného načítání celé stránky. Dále je v hlavním rozložení vyhrazeno místo pro další dynamicky vkládané komponenty, které poskytují přehledný a interaktivní způsob prezentace dat a interakce s uživatelem.

Ukládání stavu

Pro udržení dynamičnosti stránky je nezbytné zajistit reakce na dané akce, zejména při vytváření nových připomínek, a informovat ostatní komponenty, aby se aktualizovaly. Pro tento účel nabízí Vue možnost vysílání událostí rodičovskému komponentu, který může danou událost odchytit a spustit pro ni požadovaný kód. Avšak tato metoda se stává nepraktickou v případě, kdy je potřeba poslat událost vyšším rodičům nebo sourozencům. V případě posílání zprávy sourozeneckým komponentám je nutné předat zprávu nejdřív rodičovské komponentě a poté až dané komponentě, což není velmi praktické.

Aby bylo možné upozornit všechny komponenty, které by mohly mít zájem o danou akci, je nutné mít schopnost komunikace mezi všemi komponentami zároveň. Tento způsob komunikace zjednodušuje implementaci a nabízí nové možnosti.

Pro zajištění zamýšlené možnosti bylo implementováno úložiště pro ukládání stavů, které umožňuje ukládání příznaků, objektů a jiných dat, a následně poskytuje přístup k těmto datům ostatním komponentám. Úložiště bylo implementováno pomocí knihovny **Pinia** 4.4,

která umožňuje definovat obrovské množství úložišť, které se automaticky načtou do pozadí hlavní aplikace, a tak mají všechny načtené komponenty přístup k datům.

Poté mohou komponenty sledovat změny určitého stavu a provádět akce jako reakce na tyto změny. Více informací o implementaci lze nalézt v kapitole *Konkrétní implementace* 6.4.

6.4 Konkrétní implementace

Konkrétní implementace některých úseků v následující kapitole bude předvedena s cílem zlepšit srozumitelnost a zajištění správného chápání daného procesu. Některé úseky mohou být složité a vyžadovat podrobnější vysvětlení, proto bude v této kapitole poskytnuto více informací o konkrétních částech implementace. Informace by měly usnadnit pochopení návrhu aplikace a pomoci čtenáři při vytváření vlastní aplikace nebo pomoci při budoucí úpravě vyvíjené aplikace.

Přesouvání připomínek mezi kategoriemi

Funkcionalita přesouvání připomínek mezi kategoriemi již byla obecně popsána v kapitole s komponentami 6.3. Jedná se o funkci, která poskytuje uživateli pohodlnou možnost přesouvat jednotlivé prvky mezi různými kontejnery, které slouží jako úložiště pro dané prvky.

K implementaci funkcionality jsme využili knihovnu Draggable 4.4, která umožňuje implementovat přesouvání jednotlivých komponent mezi komponentami, které slouží zároveň jako úložiště pro dané komponenty. V našem konkrétním případě se jedná o přesun připomínek mezi kategoriemi. Každá kategorie využívá komponentu, kterou poskytuje použitá knihovna, a tím se stává kontejnerem pro prvky stejného typu, které již vlastní.

Při implementaci funkcionality přesouvání byly vstupními parametry určeny atributy, podle kterých se sledují změny v seznamu položek, samotný seznam položek, a to, do jaké skupiny patří. Tím se zajišťuje přijímání změn z dané skupiny. Nepovinné parametry jsou určeny pro nastavení stylů a efektů pro vizuální stránku přesouvání. Funkce byla zavedena jako nadstandardní v rámci zadání, protože se během vývoje ukázala být potencionálně užitečnou a příjemnou pro uživatele.

```
1                <draggable :id="category.id" :list="sorted_tickets" itemKey="id"  group="
    tasks" ghost-class="ghost-card" @change="log">
2                    <template #item="{ element }">
3                        <Card :ticket="element" :category="category.id"></Card>
4                    </template>
5                </draggable>
```

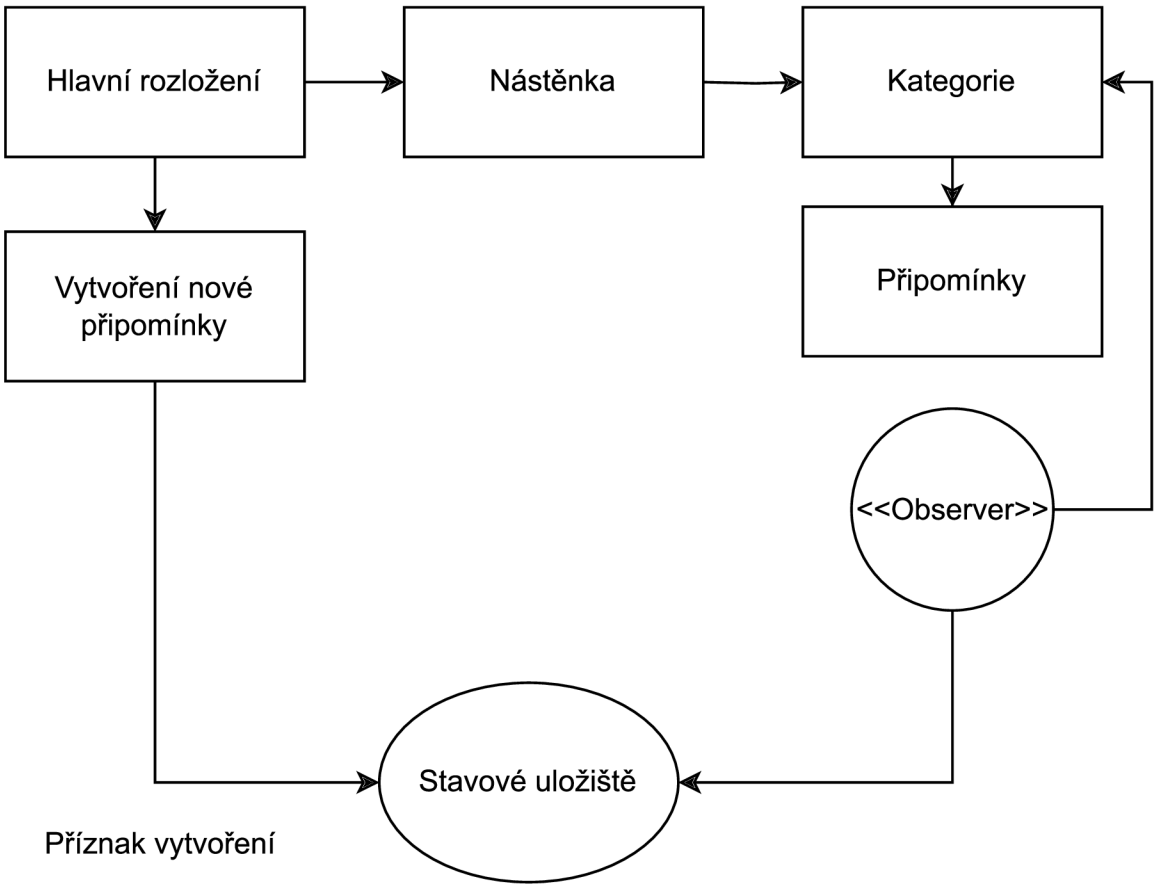
Výpis 6.3: Implementace přesouvání připomínek

Operace s událostmi

V kapitole o implementaci 6.3 byla vysvětlena důležitost událostí a nutnost spolehlivé a jednoduché komunikace mezi komponentami. Tato část se bude zabývat podrobnou implementací předávání událostí mezi komponentami.

V diagramu 6.3 je znázorněn hrubý postup předávání událostí a reakce na ně. Komponenta `Hlavní rozložení` má dva potomky: `Vytvoření nové připomínky` a `Nástěnka`. `Nástěnka` obsahuje několik `Kategorií`, z nichž každá vlastní jednotlivé `Připomínky` (viz obrázek 6.3). V takové situaci by komponenty, které nejsou přímí potomci `Hlavního rozložení`, měly být informovány o změnách v aplikaci, například při vytvoření nebo úpravě připo-

mínky. Proto musí komponenta `Kategorie` reagovat na tyto změny aktualizováním dat a musí odebírat události od `Stavového uložiště`. Při každé změně příznaku se komponenta `Kategorie` aktualizuje na základě nových dat. Příznaky spouštějí úpravy připomínek a vytvoření nových připomínek, které jsou následně zapsány do úložiště. Tím se vyvolá změna, která vynuluje příznak, připravující komponenty na novou změnu.



Obrázek 6.3: Postup upozornění na nově vytvořené upozornění

Stavové úložiště se vytváří jako vlastní komponenta se speciálními funkcemi nabízenými knihovnou Pinia 4.4. Nejdůležitější částí je deklarování stavů. Každý deklarovaný stav 6.4 umožňuje komponentě vytvořit pozorovatele 6.5 pro daný stav. Pokud se stav změní, událost je poslána všem pozorovatelům. Další funkce definované při implementaci úložiště slouží podobně jako v jiných objektech v klasickém OOP přístupu a umožňují kontrolovanou a předem definovanou modifikaci dat.

```
defineStore('dataFlagsStore', {
# Definovani upravitelnych stavu
state: () => ({
    new_data: Boolean,
    new_timeline: Boolean,
}),
# Definice akci, ktere vraci dane stavy
getters: {
    getNewData: (state) => state.new_data,
    getNewTimeline: (state) => state.new_timeline,
},
# Definovani akci, ktere nad ulozistem muzou byt uskutecneny
actions: {
    setNewData(new_data) {
        this.new_data = new_data;
        this.new_data = false;
    },
    setNewTimeline(new_timeline) {
        this.new_timeline = new_timeline;
        this.new_timeline = false;
    },
}

```

Výpis 6.4: Vytvoření stavového uložiště

Komponenta může využívat úložiště a vytvořit si tak pozorovatele pro konkrétní stav. Pokud se stav změní, pozorovatel vyvolá přiřazenou funkci v komponentě, která může na změnu reagovat a provést potřebné akce. Tímto způsobem komponenty mezi sebou efektivně spolupracují a komunikují.

```
flags.$subscribe((mutation, state) => {
    # Pri zmene je do console vypsan typ zmeny a zavolana funkce 'load_data'
    console.log("Mutation: " + mutation.type)
    load_data()
})

```

Výpis 6.5: Vytvoření pozorovatele

Díky implementaci stavového úložiště je možné velmi snadno zajistit včasnou reaktivitu a aktualizaci dat. Přijaté řešení umožňuje komponentám sledovat změny stavů a při jejich modifikaci automaticky reagovat a aktualizovat zobrazená data. To vede k vysoké efektivitě a uživatelské přívětivosti aplikace.

Autentizace

Implementování autentizace bylo složitější z důvodu umístění uživatelské tabulky v jiné aplikaci. I když aplikace nabízí možnost využít vlastní systém databáze uživatelů, musela být autentizace implementována na straně aplikačního rozhraní a nikoliv z pozice vlastníka uživatelů.

Autentizace tedy probíhá na straně vyvíjené aplikace, která zodpovídá za správu cookies, což jsou data, která jsou ukládána v prohlížeči uživatele, a za validaci tokenů, které aplikace rozděluje, a které se užívají místo přihlašovacích údajů. Autentizovat se dá dvěma způsoby. Pomocí Cookies, které se ukládají ve webovém prohlížeči klienta, nebo pomocí autentizačního **Bearer Tokenu**, který uživatel může získat po zavolání koncového bodu `/api/login` za pomocí metody `POST` obsahující přihlašovací jméno a heslo. Tento auten-

tizační token je poté poslán uživateli jako text. Pro zajištění funkčnosti musí být token obsažen v autentizační hlavičce.

Pomocí tokenu je také možné využívat webovou aplikaci bez nutnosti přihlášení, jelikož při každém pokusu o autentizaci je kontrolována nejdříve přítomnost Cookies a poté přítomnost Tokenu.

6.5 Možnosti nasazení

Ačkoli aplikace může být spuštěna přímo na zařízení nezávisle na jiných službách, doporučuje se použít kontejnerové služby, jako jsou například *Docker* nebo *Podman* spolu s orchestračním programem *Docker Compose*. Tento přístup umožňuje spouštět aplikaci v odděleném prostředí a zajistit tak plnou kompatibilitu a funkčnost bez nutnosti složité instalace. Součástí aplikace jsou také soubory, které usnadňují správné nasazení, včetně souborů `Dockerfile` a `docker-compose.yml`. `Dockerfile` obsahuje kroky nutné k vytvoření prostředí pro spuštění aplikace, zatímco `docker-compose.yml` popisuje jednotlivé služby potřebné pro správný chod aplikace, včetně databáze, která může být nezávislá na databázi nainstalované na hostitelském zařízení. Celkem se vytvoří 2 služby pro vývojový režim a 3 služby pro produkční režim. Tyto služby budou:

- Aplikace
- Databáze
- Webový server

Uživatelské zapojení

Díky využití kontejnerů se zapojení uživatele do procesu instalace minimalizovalo, neboť uživatel potřebuje pouze nainstalovat kontejnerové služby jako `Docker` nebo `Podman` a `Docker Compose`. Následně stačí definovat soubor `.env` s proměnnými prostředí přímo v adresáři s aplikací, nebo je doplnit přímo do souboru `Docker Compose`.

Tato práce, za pomoci souboru Makefile[3], je velmi přímočará a vyžaduje pro nainstalování a spuštění pouze několik příkazů, které jsou předdefinovány v souboru.

Návod na přesný postup při instalaci je popsán v souboru `README.md`.

[3] https://www.gnu.org/software/make/

Kapitola 7

Testování

V rámci aplikace byly provedeny testy na několika úrovních, včetně automatizovaných testů v prostředí aplikace. Hlavním účelem provedených testů bylo zajistit požadované chování aplikace a jejích částí. Pro vývojáře je testování v dané fázi výhodné například pro odhalení chyb již v průběhu vývoje, což snižuje čas potřebný na jejich opravu po nasazení do produkčního prostředí. Testování také zajišťuje potřebnou konzistenci, všechny části aplikace se chovají stejně i při úpravách programu a nedojde k nechtěné změně chování. Dále jsou testy skvělým způsobem popisu chování aplikace. Existují dva hlavní způsoby testování: **automatické** a **manuální**.

7.1 Automatické průběžné testování

Během vývoje je často nutné provádět rychlé a efektivní testování, a proto se používá automatické testování. Toto testování má určitá omezení a vyžaduje počáteční větší investice času, nicméně náročnost testování se postupně snižuje na minimum.

Automatické testování se dělí na několik kategorií, z nichž každá testuje jinou funkcionalitu. Laravel nabízí přímou podporu pro dva typy testování: jednotkové testy a testování funkcionality (Func), které nahrazují integrační, regresní a koncové testy.

Jednotkové testy

Jednotkové testování představuje kontrolu nejmenších jednotek aplikace, které často spočívá v ověřování vstupů a výstupů jednotlivých funkcí. V rámci aplikace byly provedeny testy všech modelů a jejich funkcí pro získávání dat, s důrazem na tzv. *"Happy Path"* scénáře, kdy je testováno správné chování systému. Výsledkem použití jednotkových testů bylo zlepšení zpracování jednotlivých funkcí a jejich výstupů, zjednodušení používaných procesů a zrychlení celého systému.

Funkční testy

V rámci funkčního testování byly ověřovány všechny koncové body aplikace, včetně kontrolérů a jejich vstupů a výstupů, což umožnilo získat detailnější a komplexnější přehled o správnosti fungování systému.

Díky propojení s Inertia 4.4 bylo možné testovat i vrácené stránky a jejich atributy. Testování jednotlivých koncových bodů umožnilo ověřit, že jednotlivé části aplikace spolupracují korektně.

Funkční testování vedlo ke zlepšení volání koncových bodů, využití validátorů dat a zjednodušení předávání informací. Při požadavcích typu `POST` a `PUT` byla navíc přidána možnost specifikovat *id* položky v adrese jako nepovinnou.

7.2 Manuální testování

Manuální testování je postup, který nevyžaduje mnoho počátečního úsilí, avšak je časově náročný a celkové testování se provádí pouze občasně. Bohužel toto testování zároveň přináší i větší faktor nepřesnosti, jelikož se vše musí testovat ručně, takže může dojít z důvodu prosté lidské chyby k opomenutí testování některé funkce nebo přehlédnutí chybového stavu.

Některé části aplikace nelze testovat pomocí automatických testů, testování klientské části, která vyžaduje uživatelský vstup, je velmi náročné na automatizaci. Proto byla většina klientské části testována při vývoji ručně.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo navrhnout a vyvinout modul informačního systému, který pomůže řešit organizování práce zaměstnanců v instituci SOA HK. Najít způsob, jak efektivně pomáhat uživatelům se správou pracovních úkolů a výsledné řešení implementovat. Práce se zabývá obecným popisem řešení problémů, které se váží s danou problematikou, a následně přechází k přijetí konkrétních postupů a vypracování konkrétního návrhu. Součástí práce byl vývoj samostatné aplikace, který je v ní popsán s důrazem na vybrané části vývoje aplikace.

V rámci zadání byly určeny základní i dodatečné cíle práce, které by mohly být implementované. Záměr práce především mířil na dobré praktiky, rozšiřitelnost a modulovatelnost výsledku práce.

Následně jsem pokračoval nastudováním používaných praktik a technologií, které se používají k vývoji webové aplikace. Především byly popsány typy architektury a a typy návrhů, které jsou důležité pro udržení čistoty kódu. Také byly popsány technologie, se kterými se můžeme setkat při vývoji webové aplikace.

Poté následovala implementace aplikace a její průběh, kde jsem se popsal obecnou problematiku implementace a následně i konkrétnější případy, které by mohly být pro čitatele složité nebo zajímavé.

Celkově si myslím, že praktiky byly vhodně vybrány pro zpracování práce a odpovídají momentálním požadavkům pro moderní webové aplikace. To zaručuje jednoduchou údržbu systému a dovoluje další úpravy.

Použité technologie byly vybrány tak, aby splňovaly požadavky zadavatele a zároveň byly ještě dlouho relevantní. To snižuje šanci na rozbití systému i případnou nutnost znovu vytvářet podobný systém.

Návrh byl vytvořen i pro další součásti systému, o které může být v budoucnu rozšířen, ale již během implementace byly určité prvky návrhu změněny z důvodu větší efektivity zpracování. Celkově si myslím, že návrh systému byl dobrý a postupem vývoje se zlepšil a zefektivnil.

Implementace pokrývá jen část z celého návrhu, ale nabízí všechny hlavní funkce, které budou využívány a nabízí i pár dalších pokročilých funkcí, jako je například upomínka přes e-mail, pokud má daná připomínka blízké datum vypršení. Jako další možnost budoucího rozšíření je k dispozici například implementace automatického opakování připomínek. Důvod pro neimplementaci opakování spočívá v tom, že byla kladena větší důraz na implementaci skupinových připomínek. Dalším potenciálním rozšířením mohou být komentáře pod skupinovými připomínkami, což by mohlo pomoci s organizací a přehledností. Mož-

nost přikládání souborů by byla další užitečnou funkcí, která by mohla usnadnit organizaci a sdílení informací. Pro mnoho z těchto funkcí je již v aplikaci připraven základ.

Konečná aplikace splňuje všechny základní požadavky a nabízí všechny pilíře, na které jsem mířil, jenž jsou udržovatelnost, rozšiřitelnost a přehlednost implementace.

Literatura

[1] Adermann, N. *Composer*. 2023. Dostupné z: https://getcomposer.org/.

[2] Agarwal, V. *Ajax — Async, Callback & Promise*. Listopad 2018. Dostupné z: https://medium.com/front-end-weekly/ajax-async-callback-promise-e98f8074ebd7.

[3] BasuMallick, C. *Stateful vs. Stateless: Understanding the Key Differences*. 2022. Dostupné z: https://www.spiceworks.com/tech/cloud/articles/stateful-vs-stateless/.

[4] Correa, D. a Vallejo, P. *Practical Laravel: Develop clean MVC web applications*. Kindle, 2022. ISBN 979-8416784195.

[5] Coursera. *What Is Python Used For? A Beginner's Guide*. Duben 2023. Dostupné z: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python.

[6] Docker. *Container Docker*. Listopad 2021. Dostupné z: https://www.docker.com/resources/what-container/.

[7] Eduardo San Martin, M. *Pinia | The intuitive store for Vue.js*. 2023. Dostupné z: https://pinia.vuejs.org.

[8] Fu, A. *VueUse*. 2023. Dostupné z: https://vueuse.org/guide/.

[9] Laravel. *Laravel - The PHP Framework For Web Artisans*. 2023. Dostupné z: https://laravel.com/.

[10] Lim, G. *Beginning Vue 3 Development: Learn Vue.js 3 web development*. Kindle, 2017.

[11] ManagementMania. *Architektura klient-server (Client–server model)*. 2016. Dostupné z: https://managementmania.com/cs/architektura-klient-server.

[12] Marsh, J. *UX pro začátečníky: (rychlokurz - 100 lekcí)*. Brno: Zoner Press, 2019. ISBN 978-80-7413-397-8.

[13] MDN. *MDN - Learn web development | MDN*. Březen 2023. Dostupné z: https://developer.mozilla.org/.

[14] Oracle. *MySQL :: Getting Started with MySQL*. 2023. Dostupné z: https://dev.mysql.com/doc/mysql-getting-started/en/.

[15] Peck, A. *C# for Web Development: Best Practices*. 2022. Dostupné z: https://visualobjects.com/web-development/blog/c-for-web-development.

[16] PHP. *PHP: What can PHP do? - Manual*. 2023. Dostupné z: https://www.php.net/manual/en/intro-whatcando.php.

[17] Red Hat. *GraphQL*. 2019. Dostupné z: https://www.redhat.com/en/topics/api/what-is-graphql.

[18] Red Hat. *REST*. 2020. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[19] Schwarzmüller, M. *Dynamic vs SPA vs Static Websites*. 2019. Dostupné z: https://academind.com/tutorials/dynamic-vs-static-vs-spa.

[20] Shah, J. *Laravel vs Symfony in 2023 : A Detailed Comparison*. Leden 2022. Section: Technology. Dostupné z: https://wpwebinfotech.com/blog/laravel-vs-symfony/.

[21] Wambua, D. W. *Most Popular Programming Languages for Web Development*. Březen 2022. Dostupné z: https://careerkarma.com/blog/best-programming-languages-for-web-development/.

[22] You, E. *Vue.js - The Progressive JavaScript Framework | Vue.js*. 2023. Dostupné z: https://vuejs.org/.

[23] Zendulka, J. a Rudolfová, I. *Databázové systémy - studijní opora*. 2006. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIDS-IT%2Ftexts%2FDatabazove_systemy.pdf.

[24] Čápka, D. *MVC architektura*. Dostupné z: https://www.itnetwork.cz/mvc-architektura-navrhovy-vzor.