ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Indoor QR navigační systém

David Krejčí
Softwarové inženýrství a technologie

Srpen 2022
Vedoucí práce: Ing. Martin Šipoš, Ph.D.

Poděkování / Prohlášení

Při vytváření práce bych chtěl poděkovat kolegovy Richardem Burkoňovi za spoluprácí na frontendové částí aplikace a vedoucímu práce Ing. Martinu Šipošovi, Ph.D.

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

V Praze dne 14. 8. 2022

.....

Abstrakt / Abstract

Práce za zabývá návrhem a implementací backendové části navigačního systému v rámci budovy využívající QR kódů k lokalizaci uživatele. Práce pokrývá analýzu, výběr technologií a počáteční implementaci aplikace. Výstupem je backend implementovaný v jazyce PHP využívající framework Nette. Komunikace mezi frontendem a backendem je realizována pomocí REST API rozhraní. Systém je postaven pro možnost poskytnutí navigace na více budovách.

Klíčová slova: avigace uvnitř budov, qr kód, textová navigace, webová aplikace, backend, PHP, REST, Nette.

This thesis deals with design and implementation of backend part of the navigation system within building using QR codes to locate the user. The thesis covers analysis, selection of technologies and initial implementation of the application. The main goal of the thesis is backend implementation in PHP using the Nette framework. Communication between frontend and backend is realized using the REST API interface. The system is built to provide navigation for multiple buildings.

Keywords: indoor navigation, qr code, text navigation, web application, backend, PHP, REST, Nette.

Title translation: Indoor QR navigation system

Obsah /

1 Úvod ... 1
2 Teoretická část ... 2
2.1 Plánovaná funkcionalita ... 2
2.2 Analýza požadavků ... 2
2.2.1 Funkční požadavky ... 2
2.2.2 Nefunkční požadavky ... 3
2.3 Programovací jazyk ... 3
2.3.1 Java ... 3
2.3.2 PHP ... 3
2.4 Framework ... 4
2.5 Externí knihovny ... 5
2.6 Systém řízení báze dat ... 5
2.6.1 PostgreSQL ... 5
2.6.2 MySQL ... 6
2.6.3 MariaDB ... 6
2.7 Komunikační protokol ... 6
2.7.1 REST ... 6
2.7.2 SOAP ... 7
2.7.3 GraphQL ... 7
2.8 Zabezpečení ... 8
2.9 Vyhledávací algoritmus ... 8
2.9.1 Prohledávání do šířky (Breadth-first) ... 8
2.9.2 Dijkstra ... 9
2.9.3 A* (A-star) ... 10
2.10 Zpracování audia ... 10
3 Návrh ... 12
3.1 Výběr použitých technologií ... 12
3.2 Použité návrhové vzory ... 12
3.2.1 Dao ... 12
3.2.2 DI ... 12
3.2.3 MVP ... 13
4 Implementace ... 14
4.1 Adresářová struktura ... 14
4.2 Závislosti ... 14
4.3 Databázový model ... 15
4.4 Struktura zdrojových kódů ... 16
4.5 Třídy v jednotlivých jmenných prostorech ... 16
4.5.1 App/AttRoute ... 16
4.5.2 Třídy v App/Data/Dao ... 17
4.5.3 App/Data/Entity ... 17
4.5.4 App/Finder ... 18
4.5.5 App/Finder/Algorithm ... 18
4.5.6 App/Finder/Audio ... 19
4.5.7 App/Persistence ... 19
4.5.8 App/Persistence/Attr ... 20
4.5.9 App/Presenters ... 20
4.5.10 App/Router ... 21
4.5.11 App/Utils ... 21
4.5.12 App/Utils/Json ... 22
4.5.13 App/Utils/Localization ... 22
4.5.14 App/Utils/User ... 22
4.6 REST API ... 22
5 Testování ... 24
5.1 Automatizované testy ... 24
5.1.1 Unit testy ... 24
5.1.2 Integrační testy ... 24
5.2 Zátěžové testy ... 24
5.2.1 Srovnání algoritmů ... 25
5.3 Zpracování hlasových příkazů ... 26
6 Závěr ... 27
Literatura ... 28
A Seznam použitých zkratek ... 31
B Obsah přiloženého média ... 32
C Zadání práce ... 33

Kapitola 1
Úvod

V rámci mých prvních dnů studia na ČVUT jsem měl často problém s nalezením místnosti kde se bude konat výuka. Při výběru tématu pro bakalářskou práci mne pak na první pohled zaujalo téma tvorby indoor navigačního systému.

Cílem práce je kompletní návrh a realizace backendu pro navigační systém v rámci budovy. Backendová část bude v rámci své funkcionality vytvářet plány tras na základě údajů z frontendu. Cesta naplánované trasy půjde ovnivnit uživatelskými preferencemi. K naplánované trase musí umět poskytnout i textový popis celé cesty. Dále bude poskytovat kompletní zabezpečenou funkcionalitu pro správu mapových podkladů.

Navigační systém bude vycházet ze poskytnuté mapy budovy a ze známých pozic `zajímavých bodů` označených QR kódem. Systém bude fungovat tak, že uživatel si vybere cíl, kam se chce v budově dostat a pomocí svého zařízení naskenuje QR kód, z kterého bude moci systém zjistit jeho aktuální pozici. Systém by měl následně najít vhodnou cestu do cíle a tu zobrazit na mapě tak, aby bylo jasné, kudy k cíli dojít. Vizualizace trasy bude doplněna i textovým popisem. V případě, že by se uživatel ztratil, bude mít možnost naskenovat nový QR kód určující aktuální pozici, podle které by systém znovu vytvořil trasu do cílového bodu. Návrh backendového rozhraní byl konzultován s kolegou Richardem Burkoňem, který v rámci své práce vytvářel frontendovou část navigačního systému.

Práce je rozdělena do 5 částí. První část se zabývá požadavky na navrhovaný systém a přehled potencionálních technologií přicházejících v úvahu pro realizaci. Následující část obsahuje návrh architektury a výběr technologií nejlépe splňující požadavky pro realizaci. Třetí část se věnuje výsledné implementaci systému pomocí zvolených technologií. Navazující čtvrtá část popisuje prostředky použité pro testování backendu a výsledky zátěžových testů. Závěr je věnován celkovému zhodnocení a možnostem případného dalšího rozvoje.

Kapitola 2
Teoretická část

Tato kapitola se věnuje soupisu požadavků na funkcionalitu backendu a přehledu potencionálních technologií a softwarových produktů využitelných v rámci práce. Do výběru byly zařazeny jen multiplatformní nástroje pro snadnou přenositelnost. Soupis a odůvodnění nakonec použitých technologií je uveden v sekci návrh 18.

2.1 Plánovaná funkcionalita

Backend bude vytvořen s ohledem na potřebnou funkcionalitu pro frontendovou aplikaci pro navigaci v budově a administraci mapových podkladů vytvořené v rámci práce kolegy Richarda Burkoňe.

Navigační aplikace umožní uživateli vizualizovat mapové podklady vybrané budovy. Aplikace dále poskytne pro vyhledání cesty mimo klasické možnosti výběru výchozího a cílového místa, i možnost načtení QR kódu. Ten může obsahovat informaci jen o cílové místnosti, nebo i výchozího místa. V takovém případě může aplikace rovnou přejít k zobrazení cesty. Po zadání trasy, aplikace poskytne vizualizaci optimální cesty a její textový popis. Pokud se uživatel po cestě ztratí, umožní mu aplikace načíst QR kód z jeho okolí podle kterého se trasa aktualizuje.

Administrační část umožní po přihlášení uživatele spravovat veškeré mapové podklady nutné pro navigační aplikaci.

2.2 Analýza požadavků

Podkapitola obsahuje soupis funkčních a nefunkčních požadavků kladených na navrhovanou implementaci backendu.

2.2.1 Funkční požadavky

Požadavky definující funkcionalitu, kterou navrhovaný systém musí umožňovat.

- F1 – Mapové podklady – backend poskytne všechny potřebné podklady pro vizualizaci mapy v rámci více patrové budovy
- F2 – Seznam místností – pro vyhledávání bude systém poskytovat seznam se zajímavými cíli pro plánování trasy
- F3 – Vyhledání cesty – systém umožní vyhledat trasu mezi 2 body v rámci jednoho objektu, nebo nalézt nejbližší trasu k určitému typu cíle (např. toalety)
- F4 – Textový popis cesty – backend poskytne textový popis naplánované trasy rozdělený do samostatných částí
- F5 – Lokalizace – texty v rámci navigace musí být možné přeložit do více jazyků
- F6 – Načítání QR kódů – v rámci plánování trasy musí být možnost vyhledat místnosti a trasy dle nahraného QR kódu
- F7 – Export QR kódů – backend umožní generovat QR kódy pro tisk

- F8 – Správa mapových podkladů – systém bude poskytovat rozhraní pro vytváření, editaci a mazání budov a pater v jednotlivých budovách, mapových podkladů a externích QR kódů
- F9 – Zadání cesty hlasovým příkazem

2.2.2 Nefunkční požadavky

Požadavky popisující atributy systému

- N1 – Webová aplikace – backend bude volně dostupný přes internet
- N2 – Rychlost odezvy – při průměrném vytížení (100 souběžných požadavků) musí být požadavky vyřízeny do 1 vteřiny

2.3 Programovací jazyk

Výběr programovacího jazyka nejvíce ovlivní vlastnosti výsledné aplikace. Výběr byl zúžen na jazyky, které jsou populární[1] v rámci vývoje webových aplikací a byly vyučovány v rámci studia.

2.3.1 Java

Programovací jazyk Java je multiplatformní objektově orientovaný jazyk pro obecné použití. Syntaxe jazyka je odvozena od C a C++. Jedná se o silně typový statický jazyk, kdy datový typ je povinný uvádět u všech proměnných a jeho kontrola probíhá už při překladu.

Vznik jazyka se datuje již do roku 1990[2] tehdy ještě pod názvem Oak. Z licenčních důvodů byl ale později změněn název na Java. V rámci vydání došlo ke změně ve způsobu číslování a původní verze 1.5 byla přejmenována na 5. Ve verzi 8 byla přidána podpora pro lambda funkce, které zkracují a zpřehledňují zdrojový kód. Poslední verze 17 byla vydaná v roce 2021 a pokračuje v postupném vývoji jazyka.

V rámci podpory Javy jsou udržovány dvě větve[3], poslední stabilní verze a LTS verze s dlouhodobou podporou. Pro standardní verzi jsou zdarma vydávány aktualizace po dobu půl roku od vydání, nebo do vydání další verze. LTS verze má prodlouženou podporu na 5 let. Plán vydání nových verzí je cílen na dvě verze za rok. Java je dostupná pod GPL licencí.

Výhody

- díky striktní syntaxi jsou řešení méně náchylné k chybám
- podpora více vláken

Nevýhody

- vyšší náklady na vývoj a provoz
- při každé změně nutnost rekompilace

2.3.2 PHP

Jedná se o univerzální skriptovací jazyk s hlavním zaměřením na vývoj webových aplikací. Jeho syntaxe je silně inspirována jazykem C a Javou. Ve výchozím stavu se chová jako slabě typový jazyk, datové typy proměnných jsou nepovinné a základní datové typy jsou plně zaměnitelné.

Počátky jazyka sahají do roku 1994[4], o rok později byla vydána verze 1.0, tehdy ještě pod názvem PHP tools, zkratka pro Personal Home Page Tools. V rámci vydání

verze 3.0 došlo k přejmenování na PHP což je rekurzivní zkratka pro PHP Hypertext Preprocessor, které se používá dodnes.

Verze 5.0 přinesla do jazyka rozšířenou podporu pro objektově orientované programování. Za verzi 6 se označuje nikdy nevydaná vývojová větev, která měla za cíl kompletní přechod na utf-8. Z důvodu výrazného dopadu na výkon bylo nakonec od tohoto přechodu upuštěno. Následující verze proto nese označení 7, v rámci které bylo PHP rozšířeno o možnost deklarace silně typového přístupu ke všem proměnným.

Poslední stabilní verze 8.0 byla vydána v roce 2020. Mimo dalšího rozvoje syntaxe jazyka přinesla i JIT kompilaci, která dále zvyšuje rychlost vykonávání programu v některých případech až dvojnásobně.

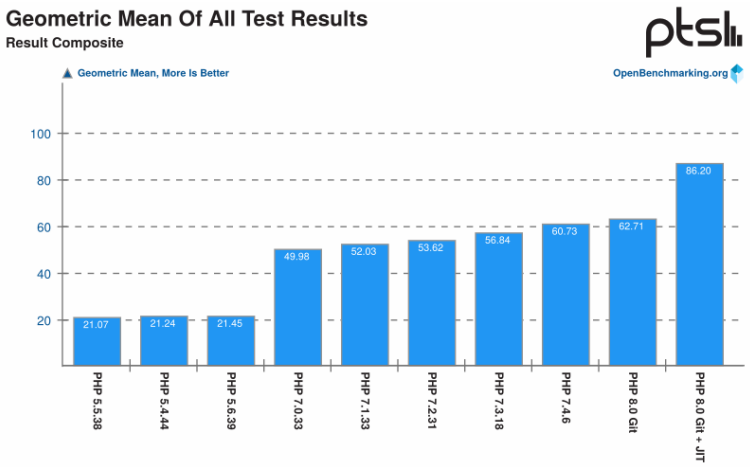
PHP má až na výjimky pravidelný vývojový cyklus, kdy každý rok vychází nová verze. Pro každou verzi jsou pak vydávány volně dostupné aktualizace po dobu 3 let. V praxi je nejvíce používá v takzvaném LAMP stacku, tj. kombinace operačního systému Linux, webového serveru Apache, PHP a MySQL databáze. PHP je distribuováno pod BSD kompatibilní licencí.

Výhody

- používá se na 78% webových stránek[5]
- dostupné na většině webových hostingů
- snadná úprava a aktualizace běžících aplikací
- nízké náklady na provoz, vývoj i údržbu

Nevýhody

- dík vysoké flexibilitě jazyka, je výsledný kód potencionálně náchylnější k chybám oproti jiným jazykům
- pokles popularity jazyka v návaznosti na růstu oblíbenosti řešeních založených na javascriptu



Obrázek 2.1. Srovnání výkonů různých verzí PHP[6]

2.4 Framework

Framework je softwarová struktura doplňující funkcionalitu poskytovanou platformou nad kterou je postaven. Cílem frameworku je nabídnout vývojáři zjednodušení často

se opakujících činností, které jsou příliš specifické nebo komplexní na to, aby byly základní součástí jazyka. V rámci této práce bude hlavně využita část zjednodušující implementaci směrování a zpracování příchozích požadavků.

V případě použití Javy přichází v úvahu použití frameworku Spring MVC[1]. Jedná se o robustní řešení, které již obsahuje komplexní funkcionalitu pro implementaci všech uvažovaných komunikačních protokolů.

Pro PHP existuje několik populárních[7] frameworků pokrývající funkcionalitu požadovanou v rámci této práce. Jedná se například o Codeigniter[2], Laravel[3], Nette[4] či Symfony[5]. Všechny zde uvedené frameworky poskytují požadovanou funkcionalitu.

2.5 Externí knihovny

Pro některou funkcionalitu aplikace, která již není součástí Frameworku, budou využity již existující řešení, např. načtení QR obrázku či generování tiskových podkladů. Způsob jakým se externí knihovny propojí s aplikací, se však v rámci programovacích jazyků značně odlišuje.

V případě programovacího jazyku Java je import závislosti ošetřen již při překladu aplikace. Oproti tomu v PHP neexistuje interní řešení. V rámci komunity PHP proto vznikl nástroj Composer[6], který řeší veškeré závislosti a případné kolize verzí jednotlivých knihoven.

2.6 Systém řízení báze dat

Tento software bude použit pro uchování a práci s informacemi potřebnými pro navigaci. Poskytuje rozhraní zajišťující přístup k uloženým datům.

Vzhledem k tomu, že data budou obsahově konzistentní, byl výběr zúžen jen na klasické relační databáze. Všechny z vybraných variant vychází z deklarativního programovacího jazyka SQL.

Structured Query Language, zkráceně SQL je standardizovaný programovací jazyk, který se používá pro správu relačních databází a provádění různých operací s daty v nich uložených. Počátky jazyka sahají do roku 1970, kdy Dr. E. F. Codd publikoval článek na toto téma v periodiku Association of Computer Machinery[8].

2.6.1 PostgreSQL

Je objektově-relační databázový systém (ORDBMS) s důrazem na co nejlepší podporu standardů SQL jazyka. Oproti klasické relační databázi umožňuje ukládat data i jako objekty.

Počátky PostgreSQL se datují do roku 1986, kdy byl na Kalifornské univerzitě v Berkeley spuštěn projekt POSTGRES financovaný výzkumnou agenturou DARPA a několika dalšími agenturami. Verze 1 byla vydána v roce 1989. Během následujících let došlo k několika změnám a v roce 1994 byl s přidanou podporou SQL vydán pod novým jménem Postgres95. Dva roky na to byl název opět změněn na nynější PostgreSQL[9]. PostgreSQL je licencován pod otevřenou PostgreSQL Licencí, která je podobná BSD či MIT licenci[10].

[1] `https://spring.io/` Spring
[2] `https://codeigniter.com/` CodeIgniter Web Framework
[3] `https://laravel.com/` Laravel - The PHP Framework For Web Artisans
[4] `https://nette.org/` Nette – Comfortable and Safe Web Development in PHP
[5] `https://symfony.com/` Symfony, High Performance PHP Framework for Web Development
[6] `https://getcomposer.org/` A Dependency Manager for PHP

2.6.2 MySQL

Jedná se o nejpopulárnější relační databázový systém (RDBMS), který dle průzkumu používá 53% vývojářů[11]. Velmi často se nasazuje v takzvané LAMP[1] kombinaci tvořící kompletní řešení pro poskytování internetových služeb.

Autorská společnost MySQL AB byla založena v roce 1995 ve Švédsku, nicméně prvopočátky softwaru sahají do roku 1979[12]. Rok po vzniku firmy byl systém vydán ve verzi 1.0. V roce 2000 byl zdrojový kód uvolněn pod GPLv2 licencí. V roce 2008 došlo k odkoupení firmou Sun Microsystems, kterou o rok později koupila firma Oracle Corporation[13]. Následující změny v licenčních podmínkách vedli k odchodu části vývojářů a založení MariaDB.

Na rozdíl PostgreSQL, MySQL poskytuje více typů datových úložišť, který řídící systém používá pro operace nad databázemi. Do verze 5.1 bylo jako výchozí úložiště používáno MyISAM, které je optimalizované na rychlost za cenu omezení funkcionality, jako například podpora transakcí a cizích klíčů pro referenční integritu. Následně se přešlo na InnoDB, které nabízí v mnoha ohledech srovnatelnou funkcionalitu s PostgreSql. Vývoj MySQL je cílen především na stabilitu a rychlost zpracování jednodušších požadavků pro načítání dat uložených v databázi i za cenu omezenější funkcionality oproti konkurenčním řešením.

Díky trvající vysoké popularitě je MySQL dostupné na takřka každém webovém hostingu a tomu odpovída navazující podpora. Dalším benefitem plynoucím z oblíbenosti je existence mnoha aplikací a nástrojů třetích stran.

Databázový systém je dostupný buď jako komerční produkt, nebo je volně distribuovaný ve verzi MySQL Community Edition který je šířen pod GPLv2 licencí.

2.6.3 MariaDB

Po odkoupení tehdejšího majitele MySQL firmou Oracle, byla část vývojářů nespokojena s přístupem nového majitele ke způsobu licencování. Proto v roce 2009 založili na zdrojových kódech vlastní odnož pojmenovanou MariaDB. Díky způsobu vzniku bylo označování verzí převzato z MySQL a tak první verze byla označena jako 5.1.

Z počátku byla v rámci vývoje snaha poskytovat maximální kompatibilitu s aktuálními verzemi MySQL, současně s tím bylo spojené stejné značení verzí hlavních vydání. Změna nastala až s příchodem verze 10, kdy se začala funkcionalita postupně rozcházet. Rozdíly se převážně týkají specifické funkcionalitě mimo SQL standardu a díky tomu jsou ve většině případů stále zaměnitelné.

2.7 Komunikační protokol

Pro výměnu dat se dnes pro webové služby používají převážně tři protokoly: REST, SOAP a GraphQL.

2.7.1 REST

Representational State Transfer je architektonický styl a metodologie pro vývoj internetových služeb. Nejedná se tedy přímo o protokol ale jen o koncept popisující jak by měl výsledný protokol fungovat. Základy pro něj položil v roce 2000 Roy Thomas Fielding v rámci své disertační práce Architectural Styles and the Design of Network-based Software Architectures[14].

[1] kombinace operačního systému Linux, webového server Apache, databáze MySQL a programovacího jazyka PHP

Koncept obsahuje 6 základních omezení, podle kterých se řídí návrh rozhraní. Jde o:

- Klient-server – klientská a serverová část by měli být navzájem izolované a vyvíjené nezávisle pro snazší přenositelnost
- Bezstavovost – jednotlivé požadavky lze vykonávat nezávisle na sobě, každý tak musí obsahovat všechny informace nutné k jeho vykonání.
- Uložitelné do mezipaměti – navržené rozhraní by mělo umožňovat případné označení výsledků požadavků pro uložení do mezipaměti pro opakované použití.
- Jednotné rozhraní – protokol má mít sjednocený formát pro komunikaci
- Vrstvený systém – protokol by měl být postaven na vrstvené architektuře, kdy každá vrstva bude určena pro konkrétní účel.
- Kód na vyžádání – nepovinné – schopnost poskytnout kód ze serveru, který rozšíří funkcionalitu na straně klienta.

Protokol splňující uvedené požadavky může být následně označen jako RESTfull API, či REST API.

V praxi se pro přístup ke zdrojům většinou používají 4 základní metody poskytované HTTP protokolem, které jsou také známé pod označením CRUD:

- POST (Create) – vytvoření zdroje
- GET (Read) – získání zdroje, při použití této metody nemá docházet k žádným změnám
- PUT (Update) – úprava zdroje
- DELETE (Delete) – odstranění zdroje

Formát pro přenos dat není nijak omezen, ale většinouse používají formáty:

- HTML Hypertext Markup Language
- XML Extensible Markup Language
- JSON JavaScript Object Notation

Volba konkrétního formátu se následně odvíjí od plánovaného použití.

2.7.2 SOAP

Simple Object Access Protocol je protokol pro přenos dat založený na XML formátu. Zpráva má na rozdíl od RESTu jasně definovanou strukturu. Protokol byl vyvinut v rámci firmy Microsoft a zveřejněn v roce 1999, jeho první verze ale vznikla již rok předtím[15].

Poskytované rozhraní je definované pomocí WSDL (Web Services Description Language) což je jazyk pro popis webových služeb založený na XML.

2.7.3 GraphQL

GraphQL je dotazovací jazyk pro API a běhové prostředí. Byl vyvíjen od roku 2012 v rámci firmy Facebook jako flexibilnější náhrada REST. Od roku 2019 byl vývoj přesunut pod hlavičku GraphQL Foundation[16].

Pro komunikaci se používá tzv. Schema Definition Language, zkráceně SDL, který popisuje formát pro tvorbu dotazů. Narozdíl od ostatních protokolů je nutné v rámci požadavků určit, která konkrétní data se mají z koncového bodu vrátit. Díky tomuto přístupu se oproti ostatním řešením většinou používá jen jeden koncový bod, který poskytuje veškeré informace. Datový formát pro odpovědi se nejčastěji používá JSON.

2.8 Zabezpečení

Backend má v rámci funkcionality poskytovat možnost editovat mapové podklady. Z tohoto důvodu je nutné zvolit pro tuto část vhodný způsob zabezpečení přístupu. Jako forma přístupu byla nakonec zvolena klasická kombinace unikátního přihlašovacího jména a hesla. Pro zamýšlené použití přichází v úvahu zabezpečení pomocí prosté HTTP autentizace nebo pomocí široce používaného Open Authorization, zkráceně Oauth.

Výhodou použití HTTP autentizace je jednoduchá implementace, nevýhodou je problematická bezpečnost a takřka nulové zabezpečení v případě nešifrované komunikace. Oauth je převážně cílen na delegaci přístupových práv, ale v rámci standardu je i možnost klasického přihlášení jaké poskytuje HTTP autentizace. Nevýhodou použití Oauth je relativně komplikovaná implementace, která je ale vyvažována díky jeho širokému rozšíření, množstvím volně dostupných implementací pro všechny uvažované vývojové platformy.

2.9 Vyhledávací algoritmus

Navigační podklady budou reprezentovány formou grafu. Samotný graf G je definován jako dvojice dvou množin - vrcholů (V) a hran (E).

V rámci navigace budou vrcholy reprezentovat body důležité pro orientaci a hrany vzdálenost mezi dvěma vrcholy. Pro použití grafu k navigaci musí být všechny hrany ohodnoceny. Ohodnocení určuje cenu průchodu danou hranou. Hodnota může vycházet ze vzdálenosti mezi jednotlivými vrcholy, nebo ovlivňovat priorizaci dané cesty před ostatními. Jednotlivé hrany mohou být orientované, jakožto reprezentace např. Eskalátoru.

Pro vyhledání nejkratší trasy je třeba zvolit vhodný algoritmus. V úvahu připadly tři následující.

2.9.1 Prohledávání do šířky (Breadth-first)

Prohledávání do šířky je základní grafový algoritmus pro procházení grafu. Řadí se mezi neinformované vyhledávací algoritmy používající k nalezení cesty dostupné jen informace v rámci samotného grafu.

Algoritmus funguje na postupném procházení všech vrcholů s počátkem v určeném vrcholu. Pro dočasné ukládání nenavštívených vrcholů používá frontu pracující na principu FIFO[1]. Algoritmus je optimální pro použití v grafech obsahující hrany se stejným ohodnocením, protože nemá vliv na pořadí procházení jednotlivými vrcholy[17].

- v grafu se určí počáteční vrchol, který reprezentuje počáteční nebo cílové místo. V daném bodě je nastavena vzdálenost 0, v ostatních bodech je nastavena na nekonečno, či jinou nedosažitelnou hodnotu
- do fronty se vloží počáteční vrchol
- dokud nebude fronta prázdná, vyjme se z fronty první položka a navštíví se všechny sousedící vrcholy
  - pokud vrchol ještě nebyl navštíven, nastaví se předchozí vrchol a vzdálenost jako součet vzdálenosti výchozího vrcholu a ohodnocení hrany. Následně se vrchol zařaď do fronty, pokud tam již není

[1] při zpracování bude první vložený záznam vybrán jako první

- v opačném případě se porovná uloženou vzdálenost s nově vypočtenou. Jestliže je uložená vzdálenost větší, nahradí se novou a aktualizuje se předchozí vrchol. Vrchol se následně zařadí znovu do fronty, pokud tam již není

Jednotlivé vrcholy jsou procházeny postupně podle počtu hran od počátečního vrcholu. Ze zpracovaného grafu lze následně vyčíst nejkratší cestu grafem mezi počátečním a všemi ostatními vrcholy, pokud mezi nimi nějaká cesta existuje.

Asymptotická složitost algoritmu pro grafy s hranami se stejným ohodnocením je

$$O(|V| + |E|)$$

kde V je počet vrcholů a E je počet hran. V opačném případě je složitost

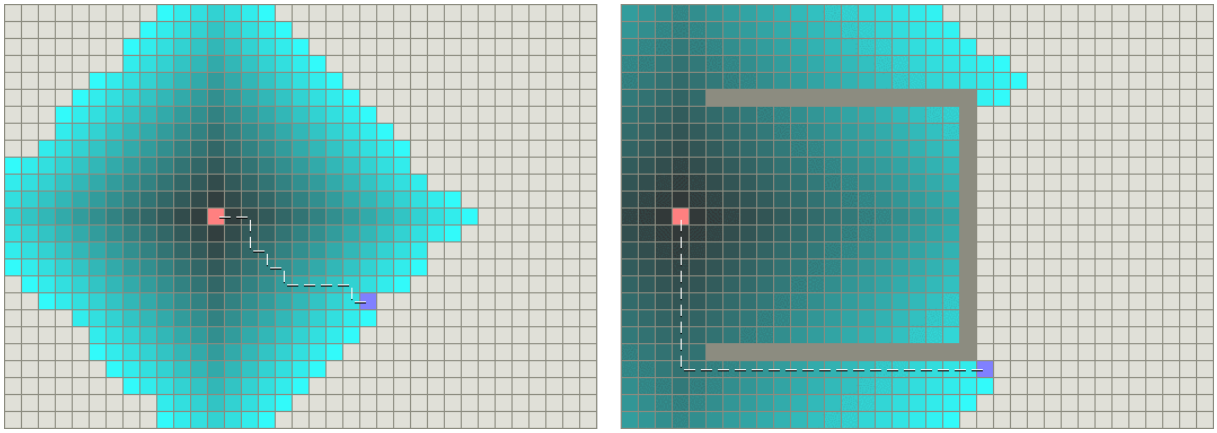
$$O(|V|^2)$$

2.9.2 Dijkstra

Algoritmus publikoval v roce 1959 Dr. Edsger W. Dijkstra[17]. Řadí se mezi takzvané hladové algoritmy, které se snaží při každém kroku při výběru ze všech možností vybrat variantu s nejmenší hodnotou. Dijkstra algoritmus nelze použít na grafy obsahující záporně ohodnocené hrany. V případě použití na graf obsahující jen hrany se stejným ohodnocením prochází algoritmus vrcholy stejně jako v případě prohledávání do šířky.

V základu funguje podobně jako vyhledávání do šířky, po počáteční inicializaci se začne ve výchozím vrcholu. Všechny následující vrcholy se ale vyberou na základě minimální hodnotě vzdálenosti.

Vrcholy jsou v tomto případě procházeny podle postupně rostoucí vzdálenosti od počátečního vrcholu. Díky tomu již není potřeba znovu procházet již navštívené vrcholy a vyhledávání lze ukončit v okamžiku navštívení cílového vrcholu, nebo lze nechat algoritmus projít všechny vrcholy a získat tak možnost vytvoření nejkratší cesty pro každý dostupný vrchol.



Obrázek 2.2. Vizualizace postupného procházení dvourozměrným polem dijkstra algoritmem[18]

Asymptotická složitost je závislá na konkrétní implementaci, v případě nejjednodušší implementace dosahuje složitost

$$O(|V|^2)$$

Při použití optimalizací pro způsob výběru následujícího vrcholu lze dosáhnout složitosti

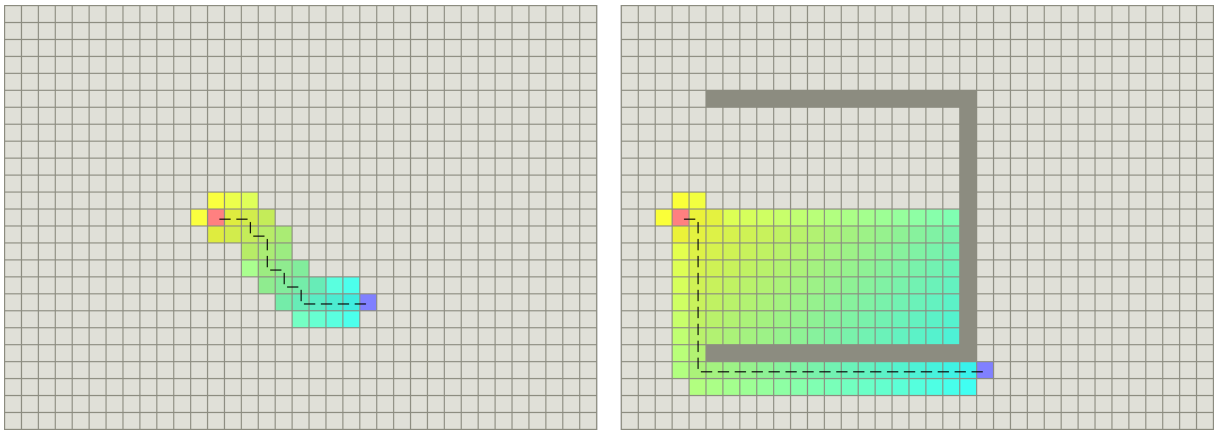
$$O(|V|log|V| + |E|)$$

2.9.3 A* (A-star)

Algoritmus byl publikován v roce 1968 autory: Peter E. Hart, Nils J. Nilsson a Bertram Raphael[19]. A-star se na rozdíl od dijkstry a prohledávání do šířky řadí mezi informované vyhledávací algoritmy. Tyto algoritmy poskytují heuristickou metodu pro odhad ohodnocení vhodnosti při výběru následujícího vrcholu. V optimálním případě by měla metoda poskytnout nejkratší možnou vzdálenost mezi aktuálním a cílovým vrcholem. Pokud nabývá jiné než minimální hodnoty, nemusí být nalezená cesta nejkratší možná.

Při vyhledávání se může postupovat stejně jako v případně dijkstra algoritmu, jen s rozdílným výběrem následujícího vrcholu pro který se použije heuristická metoda.

Oproti předchozím algoritmům lze po skončení zpracování grafu nalézt jen nejkratší cestu mezi výchozím a navštívenými vrcholy.



Obrázek 2.3. Vizualizace postupného procházení dvourozměrným polem A-star algoritmem[18]

Asymptotická složitost algoritmu je silně závislá na použité heuristické funkci, která ovlivňuje množství cest uvažovaných během výpočtu. Složitost by ale nikdy neměla být větší než v případě prohledávání do šířky

$$O(|V| + |E|)$$

2.10 Zpracování audia

Převod hlasu do textové podoby bude v rámci aplikace zajištěn takzvaným STT – Speech To Text, který převede mluvený audio vstup do textové podoby. Textový výstup se následně v aplikaci dále zpracuje. STT řešení jsou dostupná buď ve formě většinou uzavřených zpoplatněných online rozhraních, nebo pomocí lokálních modulů, které většinou neposkytují rozhraní pro jazyky používané pro webové aplikace.

Mezi zástupce online řešení se řadí cloudový Google voice[1]. Pro použití je nutné zadání kreditní karty a častější využívání je zpoplatněno.

Mezi moduly, které umožňují lokální běh, patří DeepSpeech [2] vyvíjený společností Mozzila. Je založen na výzkumném projektu Deep Speech od společnosti Baidu. Pro implementaci bylo využito strojové učení TensorFlow od firmy Google. Modul dle testů [20] patří mezi nejlepší dostupné řešení s otevřeným zdrojovým kódem. DeepSpeech

[1] `https://cloud.google.com/speech-to-text` Speech-to-Text: Automatic Speech Recognition — Google Cloud

[2] `https://deepspeech.readthedocs.io/` Mozilla DeepSpeech

je distribuován pod volnou licencí MPL-2.0. Pro rozeznání mluveného slova vyžaduje DeepSpeach vytrénovaný akustický model. V rámci hlavního vývoje jsou poskytovány vytrénované modely pro anglický a čínský jazyk, ostatní jsou poskytovány v rámci uživatelské komunity.

Kapitola 3
Návrh

Kapitola se věnuje zdůvodnění výběru použitých technologií a návrhových vzorů, které se použijí při vývoji aplikace.

3.1 Výběr použitých technologií

Oba programovací jazyky poskytují veškerou funkcionalitu potřebnou pro implementaci řešení. Výběr nakonec padl na PHP především z důvodu osobních preferencí, snadnosti nasazení a dostupného prostoru pro vystavení výsledné aplikace.

V případě databázového serveru byl vybrán MySQL, drtivá většina požadavků jdoucí na databázi budou příkazy pro výběr záznamů. Potencionální výhody které přináší použití PostgeSql se v tomto případě nijak neprojeví.

Vzhledem k očekávané struktuře budov a z nich vzniklých grafů lze předpokládat, že do jednotlivých vrcholů povede minimum možných cest. Díky omezenějším možnostem optimalizace pro jednotlivé algoritmy vycházející z použitého jazyka, může dojít k tomu, že výhody komplexnějších algoritmů mohou být potlačeny nutností vyšší režie pro hledání následujícího vrcholu. V rámci vývoje tak došlo k implementaci všech uvažovaných algoritmů a finální algoritmus byl vybrán až na základě výkonnostních testů.

Při komunikaci backendu s frontendem bude téměř výhradně docházet jen k přenosu přesně určených dat a není důvod proč nezvolit pro komunikaci REST. Po domluvě s kolegou Richardem Burkoňem byl pro výměnu dat vybrán formát JSON.

Pro zpracování hlasových příkazů byl nakonec zvolen modul DeepSpeech. Hlavní roli v tomto případě hrála je dostupnost bez nutnosti platby či registrace pro tuto volbu a částečně také osobní zvědavost jak takové řešení bude fungovat. Pro testování funkcionality byly použity anglický[21] a český[22] akustický model.

3.2 Použité návrhové vzory

Vývojářské návrhové vzory jsou ustálené, časem ověřené postupy, používané při vývoji aplikací pro řešení problémů, které při práci mohou nastat, případně umožňují se jich vyvarovat.

3.2.1 Dao

Návrhový vzor Data Access Object (DAO) slouží k separaci aplikační a persistentní vrstvy. Tento přístup umožňuje provádět změny v kódu jedné vrstvě bez dopadu na druhou. Nevýhodou tohoto řešení může být potencionální negativní dopad na výkon.

3.2.2 DI

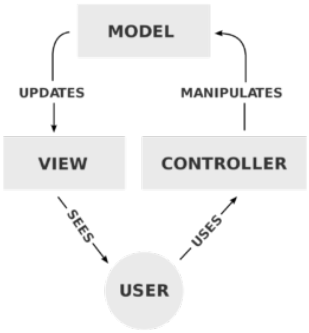
Dependency Injection je technika předávání závislostí v rámci hierarchie. Třída si v tomto případě o své závislosti nežádá ani je nezakládá, ale jsou ji předány. Na tomto návrhovém vzoru staví Nette framework.

3.2.3 MVP

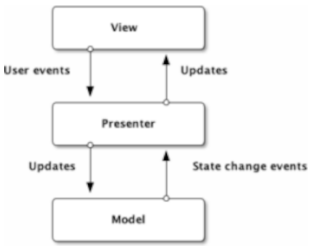
Model-View-Presenter je softwarová architektura na které je založen Nette framework. Jedná se o odvozený vzor od Model-View-Controler MVC. Oba přístupy se snaží o rozdělení struktury do třech vrstev:

- Model - reprezentuje informace s kterou aplikace pracuje
- View – pohled zajišťuje reprezentaci dat z modelu
- Presenter — Controler – řídí reakce na příchozí události

Rozcházejí se ale ve způsobu jakým mezi sebou jednotlivé vrstvy komunikují. V případě MVC Controller řídí Model, který aktualizuje View. Naproti tomu v MVP dochází k obousměrné komunickaci mezi Presenterem a View či Modelem.



Obrázek 3.1. Vizualizace komunikace v MVC[23]



Obrázek 3.2. Vizualizace komunikace v MVP[23]

Kapitola 4
Implementace

Následující kapitola obsahuje popis vnitřní struktury a funkcí jednotlivých částí aplikace.

4.1 Adresářová struktura

- /data – obsahuje uživatelem nahrané obrázkové podklady pro mapy
- /core – adresář obsahuje všechny soubory a nastavení nutné pro provoz aplikace
  - /App – zdrojové kódy, obsah je detailněji popsán v kapitole Struktura zdrojových kódů 4.4
    - /tempates – adresář obsahující soubory pro šablonovací systém Latte
  - /audio – modul a navazující soubory použité pro STT
  - /config – konfigurační soubory
  - /log – úložiště pro chybové hlášky vytvořené nástrojem Tracy
  - /temp – adresář pro dočasné soubory
  - /vendor – adresář pro Composer obsahující komponenty třetích stran
  - .htaccess – konfigurační soubor Apache blokující přístupy k obsahu adresáře
  - bootstrap.php – soubor obsahující PHP skript inicializující aplikaci
  - composer.json – nastavení pro Composer
- .htaccess – konfigurační soubor Apache pro aplikaci
- index.php – spouštěcí PHP skript

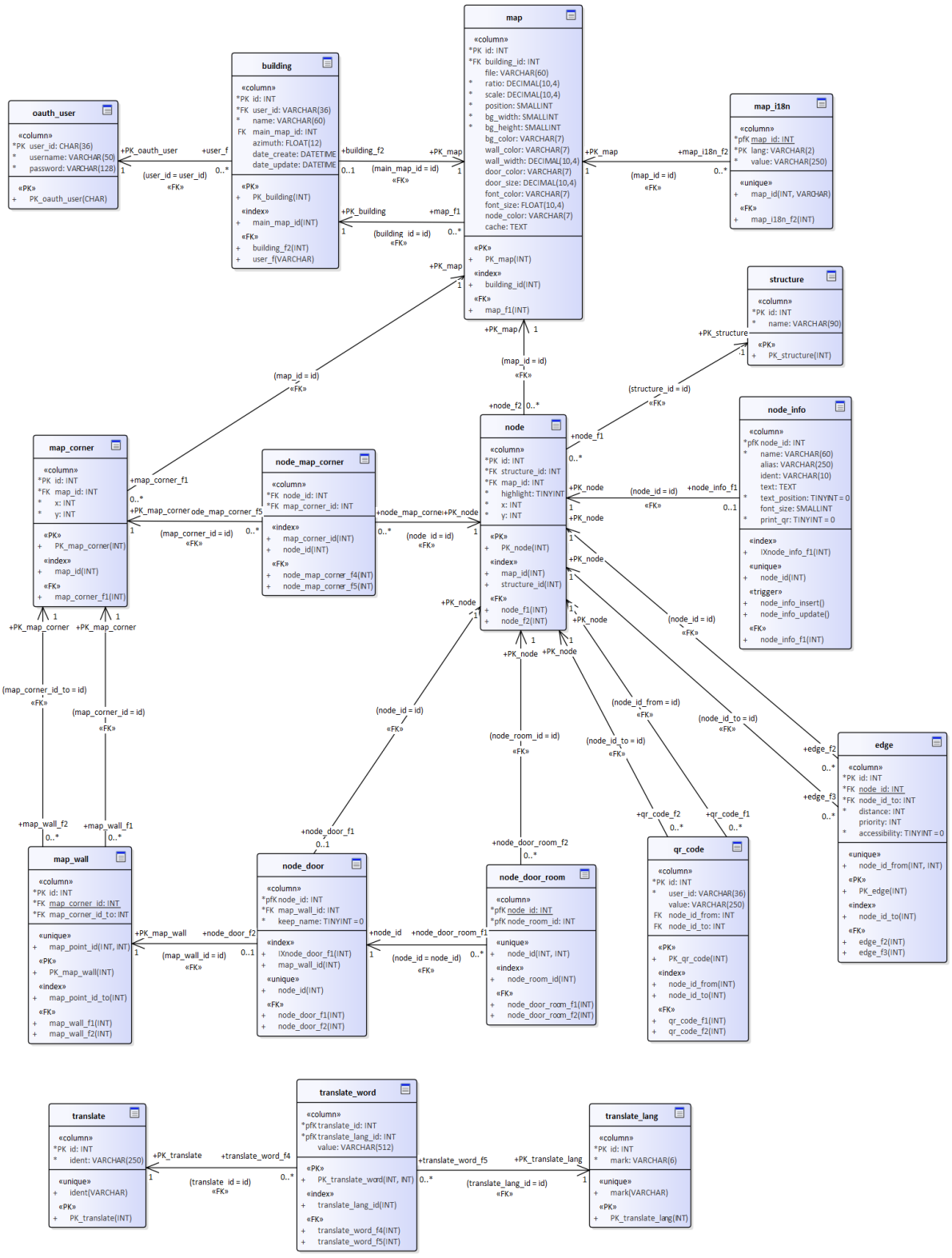
4.2 Závislosti

Aplikace používá mimo Nette frameworku i několik dalších knihoven poskytujících dodatečnou funkcionalitu, zejména pro práci QR kódy. Všechny tyto závislosti zajišťuje nástroj Composer, který importuje následující knihovny:

- khanamiryan/qrcode-detector-decoder – knihovna pro načtení QR kódu z obrázků
- latte/latte – šablonovací nástroj použitý pro stránku s popisem API aplikace, součástí Nette frameworku
- nette/application – základní knihovny Nette Frameworku
- nette/forms – knihovna pro práci s webovými formuláři, součástí Nette frameworku
- tracy/tracy – vývojářský nástroj pro ladění aplikací, vyvíjeno v rámci Nette frameworku
- nette/bootstrap – inicializační funkcionalita pro Nette framework
- nette/database – databázová vrstva pro Nette frameworku
- bacon/bacon-qr-code – zajišťuje export QR kódů do obrázků
- drahak/oauth2 – rozšíření pro Nette přidávající podporu OAuth2
- mpdf/mpdf – knihovna pro export html kódu do pdf

4.3 Databázový model

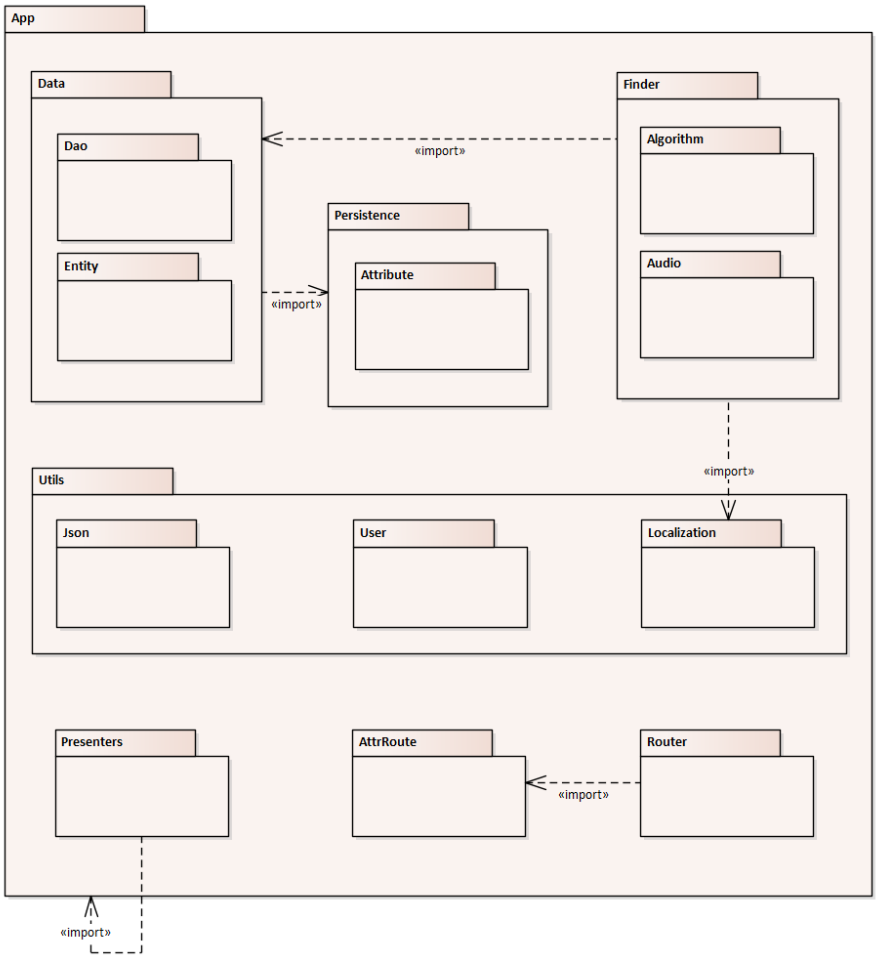
Databázový model popisuje



Obrázek 4.1. Databázový model (neobsahuje tabulky používané rozšířením OAuth2)

4.4 Struktura zdrojových kódů

Zdrojové soubory jednotlivých tříd jsou pro lepší přehlednost rozděleny do skupin obsahující navzájem relevantní funkcionalitu. V PHP se pro toto rozdělení používají jmenné prostory, což je obdoba balíčku v javě. Struktura jmenných prostorů následně odpovídá adresářové struktuře, ve kterých se jednotlivé sobory nacházejí.



Obrázek 4.2. Struktura jmenných prostorů

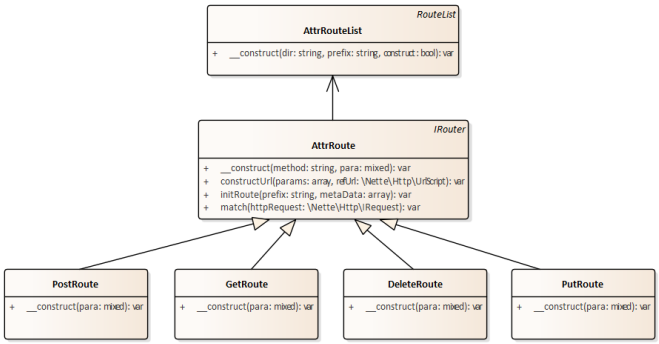
4.5 Třídy v jednotlivých jmenných prostorech

Kapitola obsahuje detailní popis funkcionality tříd rozdělených do jednotlivých jmenných prostorů.

4.5.1 App/AttRoute

Třídy v tomto jmenném prostoru poskytují podporu pro takzvané routování - mapování příchozích URI požadavků na označené metody v presenterech. Pro tuto funkcionalitu používají atributy uvedené u jednotlivých metod, je tak možné adresy měnit v rámci presenteru samotného a ne až v routeru4.5.10, či jiném místě.

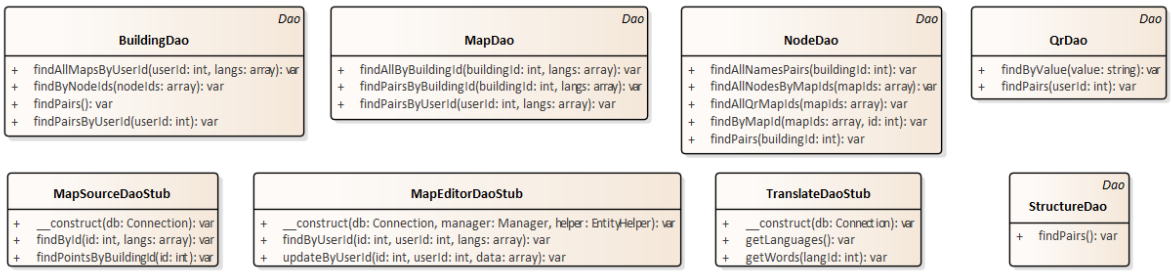
Třída AttrRoute navržena jako dekorátor třídy pro routování z Nette frameworku. K její základní funkcionalitě přidává podporu pro definování i http metody pro routování.



Obrázek 4.3. Diagram tříd ve jmenném podprostoru App/AttRoute

4.5.2 Třídy v App/Data/Dao

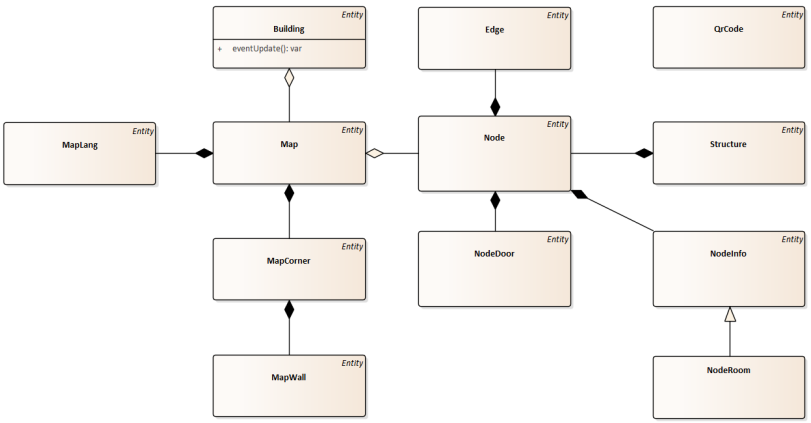
Ve jmenném prostoru se nachází třídy komunikující s datovým úložištěm umožňující práci s entitami4.5.3. Případně poskytují výsledky specificky formátovaných dotazů a data jsou v takovém případě převážně vracena jako pole. Třídy s označením Stub (MapEditorDaoStub, MapSourceDaoStub, TranslateDaoStub) obsahují jen požadavky na úložiště a neposkytují žádnou funkcionalitu pro entity. Ostatní třídy jsou potomkem abstraktní třídy App/Persistence/Dao4.5.7 umožňující načtení enit a CRUD operace pro ně odpovídající.



Obrázek 4.4. Diagram tříd ve jmenném podprostoru App/Data/Dao

4.5.3 App/Data/Entity

Prostor obsahuje veškeré entity reprezentující data uložená na úložišti. Všechny třídy jsou potomkem abstraktní třídy App/Persistence/Entity4.5.7, která definuje základní funkcionalitu pro persistenci4.5.7.

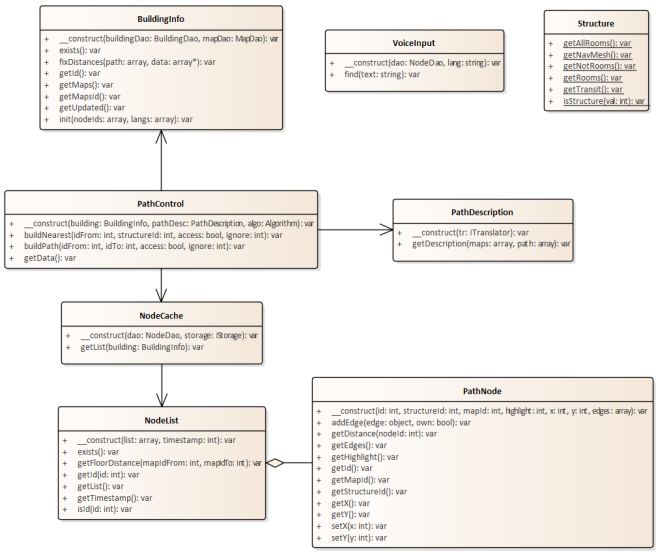


Obrázek 4.5. Diagram tříd ve jmenném podprostoru App/Data/Entity

4.5.4 App/Finder

Třídy obsažené v tomto jmenném prostoru poskytují základní funkcionalitu pro vyhledávání.

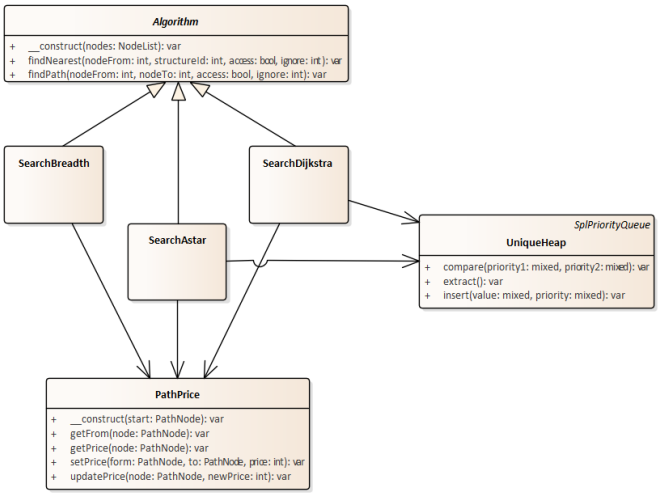
- BuildingInfo – načítá základní informace o budově a mapách, v rámci toho dojde k ověření existence hledaných navigačních vrcholů. Po nalezení trasy provede přepočet vzáleností do metrického systému.
- NodeCache – mezipamět pro třídu NodeList.
- NodeList – načte seznam všech vrcholů ze všech map v budově.
- PathControl – zajišťuje celkové zpracování vyhledávání na základě parametrů předaných z presenteru.
- PathDescription – třída zajišťuje textový popis pro nalezenou cestu. Pro překlad textů používá třídu App/Utils/Localization/Translator4.5.13
- PathNode – reprezentace mapových vrcholů uzpůsobená pro použití ve vyhledávacím algoritmu.
- Structure – podpůrná třída pro vyhledávání definující označení pro jednotlivé typy PathNode – určení zda se jedná např. o místnost, dveře, výtah, navigační bod atp.
- VoiceInput – třída slouží pro vyhledání odpovídajících mapových vrcholů v případě zadání hlasem



Obrázek 4.6. Diagram tříd ve jmenném podprostoru App/Finder

4.5.5 App/Finder/Algorithm

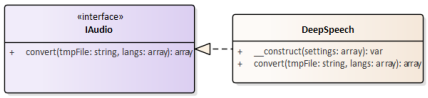
Ve jmenném prostoru jsou umístěny implementace jednotlivých vyhledávacích algoritmů. Třída PathPrice uchovává ohodnocení navštívených vrcholů vyhledávacím algoritmem. Třída UniqueHeap pro algoritmus uchovává a řadí vrcholy pro následující krok při hledání.



Obrázek 4.7. Diagram tříd ve jmenném podprostoru App/Finder/Algorithm

4.5.6 App/Finder/Audio

Tento jmenný prostor obsahuje třídy pro převod zvukových souborů na text.

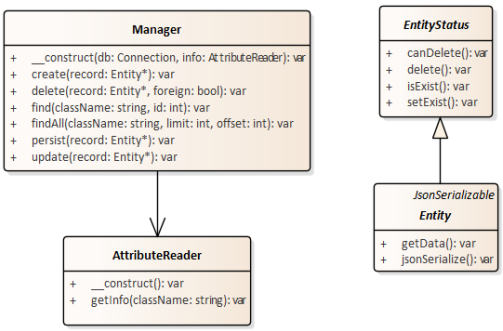


Obrázek 4.8. Diagram tříd ve jmenném podprostoru App/Finder/Audio

4.5.7 App/Persistence

Jmenný prostor obsahuje základní třídy pro mapování dat mezi úložištěm a objekty v aplikaci a jejich datovou persistenci.

- AttributeReader – podpůrná třída sloužící k načtení atributů z entit
- Dao - abstraktní třída omezující funkcionalitu třídy Manager jen pro vybranou entitu
- EntityStatus – abstraktní třída mapující aktuální stav entity ve vztahu k databázi.
- Entity – abstraktní potomek třídy EntityStatus rozšiřující o funkcionalitu převodu obsahu entity do JSON formy.
- Manager – třída poskytuje spojení mezi daty v úložišti a entitami postupnými v aplikaci. Pro entity zajišťuje základní CRUD operace – vytvoření, načtení, editaci a smazání.

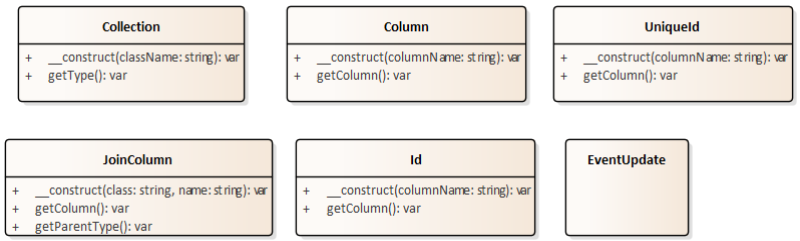


Obrázek 4.9. Diagram tříd ve jmenném podprostoru App/Persistence

4.5.8 App/Persistence/Attr

Třídy z tohoto prostoru umožňují v entitách definovat sloupce se specifickými vlastnostmi v rámci databáze, které jsou nutné pro zajištění datové konzistence při interakci s databází.

- Collection – realizace vztahu 1:n mezi entitami v databázi – entita obsahuje pole určených entit definovaných v rámci atributu
- Column – ve výchozím stavu jsou názvy pro úložiště stejné jako v entitě, tento atribut umožňuje použít jiné jméno proměnné pro databázi
- EventUpdate – atribut umožňuje označit funkci v entitě, která se má zavolat před každou změnou pří zápisu do databázi
- Id – označuje primární klíč
- JoinColumn – označuje sloupec z druhé strany vztahu 1:n
- UniqueId – označuje unikátní klíče



Obrázek 4.10. Diagram tříd ve jmenném podprostoru App/Persistence/Attr

4.5.9 App/Presenters

Tento jmenný prostor obsahuje třídy zajišťující vyřízení příchozích požadavků, které jim předal router4.5.10. Vyjma UserPresenter jsou všechny třídy v prostoru potomkem abstraktní třídy AbstractPresenter, který zajišťuje načtení proměnných zajišťujících základní funkcionalitu potřebnou ve většině presenterů.

Dále lze presentery rozdělit na veřejně přístupné a zabezpečené. K obsahu zabezpečených presenterů se lze dostat jen s platným Oauth tokenem, který lze získat po přihlášení. Implementace zabezpečení je řešena v rámci abstraktní třídy SecurePresenter. Poskytovaná funkcionalita je v tomto případě omezena jen na budovy, které vytvořil přihlášený uživatel.

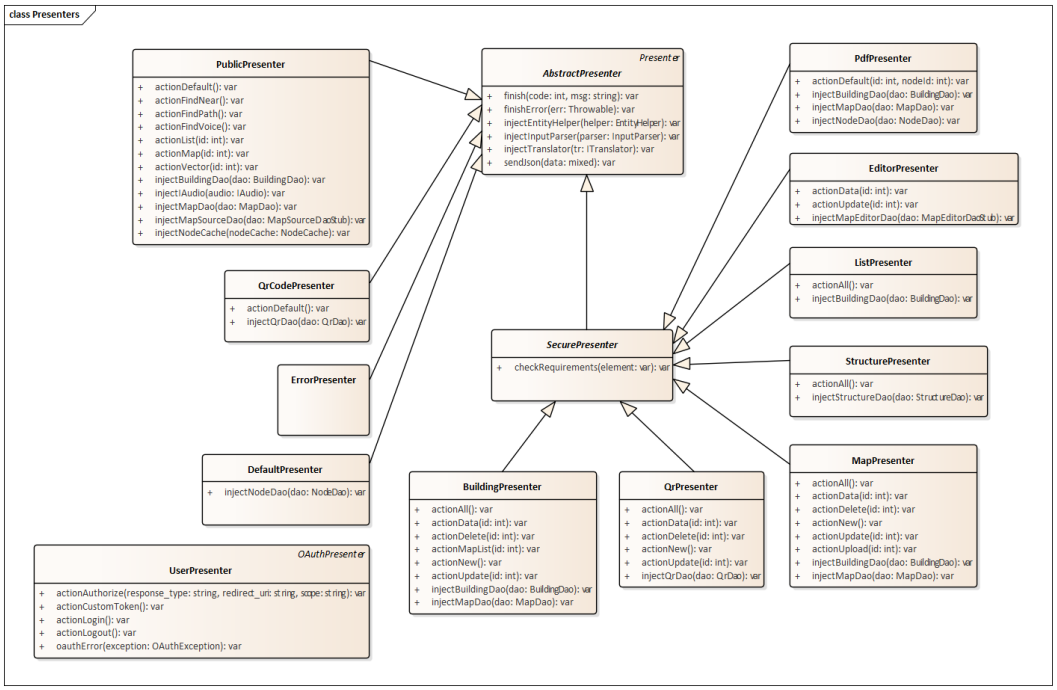
Veřejně přístupné

- DefaultPresenter – zobrazí základní popis rozhraní aplikace a několik testovacích formulářů pro demonstraci funkcionality. Pro vizualizaci používá latte šablonu.
- ErrorPresenter – ke zobrazení chybového presenteru dojde v případě, nastane-li v aplikaci kritická chyba, nebo při požadavku na neexistující URI. Pro vizualizaci používá latte šablonu.
- PublicPresenter – poskytuje veškerá podkladová data pro vizualizaci navigace ve formátu JSON.
- QrCodePresenter – umožňuje načtení obsahu QR kódu z obrázkového zdroje.
- UserPresenter – poskytuje rozhraní pro přihlášení, vydávání a ověřování tokenů OAuth2.

Zabezpečené

- BuildingPresenter – rozhraní pro editaci budov.

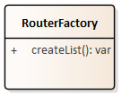
- EditorPresenter – poskytuje specifické rozhraní pro editaci mapových podkladů.
- ListPresenter – poskytuje seznam všech budov a k nim odpovídající mapy.
- MapPresenter – rozhraní pro editaci mapových podkladů.
- PdfPresenter – umožňuje export QR kódů do pdf pro případný tisk.
- QrPresenter – rozhraní pro správu specifických QR kódů.
- StructurePresenter – poskytuje jen seznam všech typů vrcholů.



Obrázek 4.11. Diagram tříd ve jmenném podprostoru App/Presenters

4.5.10 App/Router

V tomto jmenném prostoru je jen třída zastřešující routování požadavků v rámci aplikace.

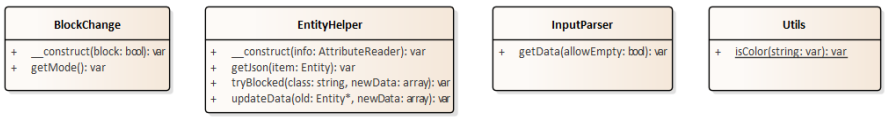


Obrázek 4.12. Diagram tříd ve jmenném podprostoru App/Router

4.5.11 App/Utils

Jmenný postor obsahuje univerzální podpůrné třídy bez specifického zařazení.

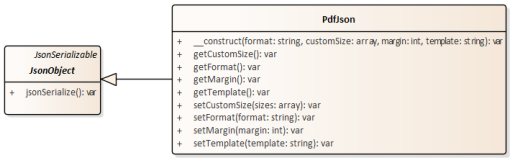
- BlockChange – atribut pro označení proměnné entity, která se nesmí při aktualizaci přes metodu ve třídě EntityHelper změnit
- EntityHelper – obsahuje podpůrnou funkcionalitu pro aktualizaci entity z dat zaslaných uživatelem a jejich validaci.
- InputParser – podpůrná třída pro načtení vstupních JSON dat.
- Utils – obsahuje jen funkci pro kontrolu hexadecimálního formátu barvy.



Obrázek 4.13. Diagram tříd ve jmenném podprostoru App/Utils/Json

4.5.12 App/Utils/Json

V tomto jmenném prostoru se nachází třída poskytující možnost importu nastavení pro export pdf.



Obrázek 4.14. Diagram tříd ve jmenném podprostoru App/Utils/Json

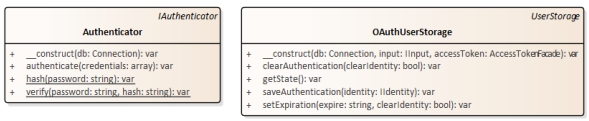
4.5.13 App/Utils/Localization

Jmenný prostor obsahuje pouze jednu třídu Translator zajišťující překlady textů uvedených v rámci zdrojových kódu aplikace.

Obrázek 4.15. Diagram tříd ve jmenném podprostoru App/Utils/Localization

4.5.14 App/Utils/User

Třídy autentizace



Obrázek 4.16. Diagram tříd ve jmenném podprostoru App/Utils/User

4.6 REST API

Koncové body poskytované backendem lze rozdělit do dvou skupin.

Veřejná část - obsah je přístupný bez omezení

- /api/v1/login – přihlášení uživatele do administrační části
- /api/v1/public – poskytuje data pro vizualizaci mapy a vyhledávání cest
- /api/v1/qr-code – funkcionalita pro načtení a zobrazení QR kódu

Administrační část - obsah bude poskytnut jen přihlášeným uživatelům

- /api/v1/building – editace dat budovy
- /api/v1/editor – poskytuje specifické rozhraní pro editor map
- /api/v1/list – seznam všech budov
- /api/v1/logout – odhlášení uživatele

- /api/v1/map – editace dat mapy
- /api/v1/pdf – poskytuje pdf s QR kódy vybraných vrcholů
- /api/v1/qr – editace QR kódů
- /api/v1/structure – seznam podporovaných typů vrcholů

Metoda	Koncový bod	popis
GET	/api/v1/map	seznam `klíč - hodnota`
POST	/api/v1/map	nová mapa - Json
POST	/api/v1/map/[id mapy]	nahrání podkladového obrázku obrázku `file`
GET	/api/v1/map/[id mapy]	data mapy
PUT	/api/v1/map/[id mapy]	update (i částečný) mapy
DELETE	/api/v1/map/[id mapy]	odstranit mapu

Tabulka 4.1. Ukázkový popis koncových bodů pro editaci mapy.

Kapitola 5
Testování

Kolega Richard Burkoň tvořící frontend se zaměřil hlavně na uživatelské testování. Tento přístup ale není příliš vhodný pro ověření funkcionality backendu samotného. Proto se tato kapitola věnuje automatickému testování, testování výkonosti výsledného řešení a testování převodu mluveného slova.

5.1 Automatizované testy

Pro tvorbu testů bude použit testovací framework PHPUnit[1]. Svou funkcionalitou je velmi podobný testovacímu nástroji pro Javu - JUnit[2].

5.1.1 Unit testy

Unit testy cílí na otestování co nejmenší části zdrojového kódu. V případě objektově orientovaných jazyků je to třída. Všechny případné závislosti by měly být nahrazeny takzvanými mock objekty, které požadované závislosti jen simulují. Tento přístup umožňuje testovat třídu bez toho, aniž by již existovali ty, na kterých je třída závislá. Nevýhodou tohoto typu testů je, že se nehodí pro třídy s mnoha závislostmi, nebo třídy ve kterých dochází k interakci mezi závislými třídami. V takovém případě dochází v rámci mockování k jejich reimplementaci. Toto je případ většiny tříd v rámci tohoto projektu, které by bylo vhodné testovat. Z tohoto důvodu nemá značná část tříd vlastní Unit testy.

5.1.2 Integrační testy

Integrační testy se naopak zaměřují na testování interakcí mezi třídami. V rámci testů lze použít princip mockování pro omezení rozsahu který má být testován. Tento typ testů se jeví jako nejvhodnější způsob jak otestovat zdrojový kód aplikace. Testy jsou rozděleny do větších skupin odpovídající interní struktuře aplikace.

- Algoritmy – testování správné funkčnosti vyhledávání
- Presentery – simulace příchozích požadavků
- Dao – ověření správné komunikace s databází

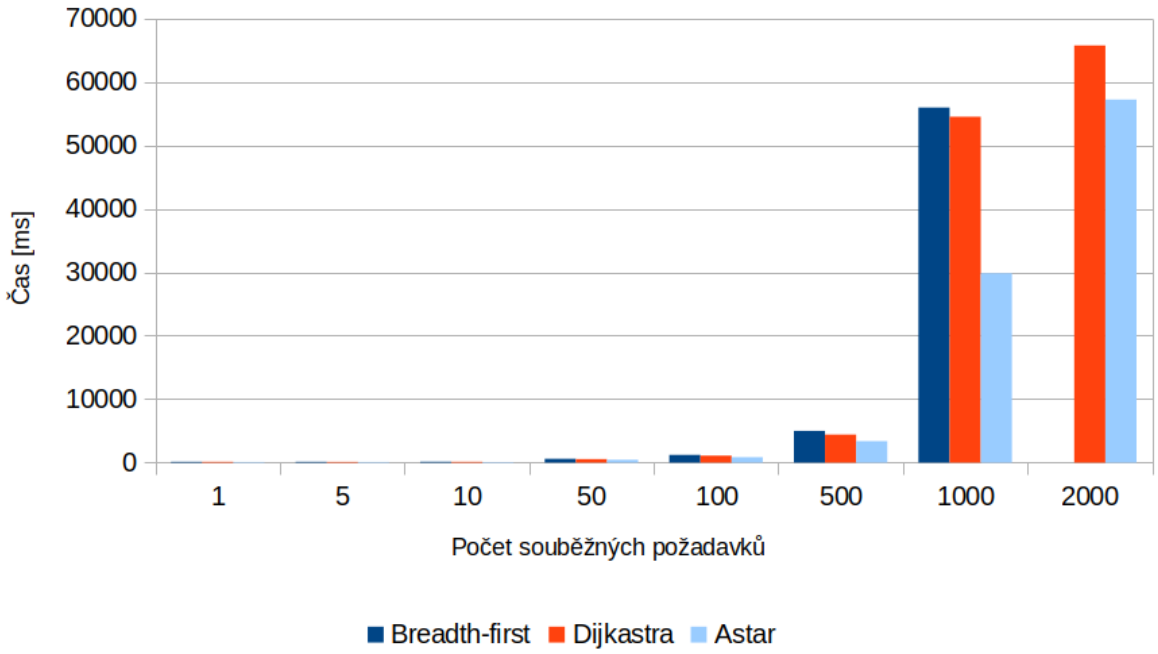
5.2 Zátěžové testy

Pro testovaní byl použit nástroj ApacheBench, který měří dobu trvání vyřízení požadavku, případně jeho zahození v případě přetížení serveru. Nástroj umožňuje nastavit celkový počet zaslaných požadavku a počet zaslaných požadavků v jeden okamžik. Naměřené údaje jsou silně ovlivněny použitým hardwarem a konfigurací použitého softwaru.

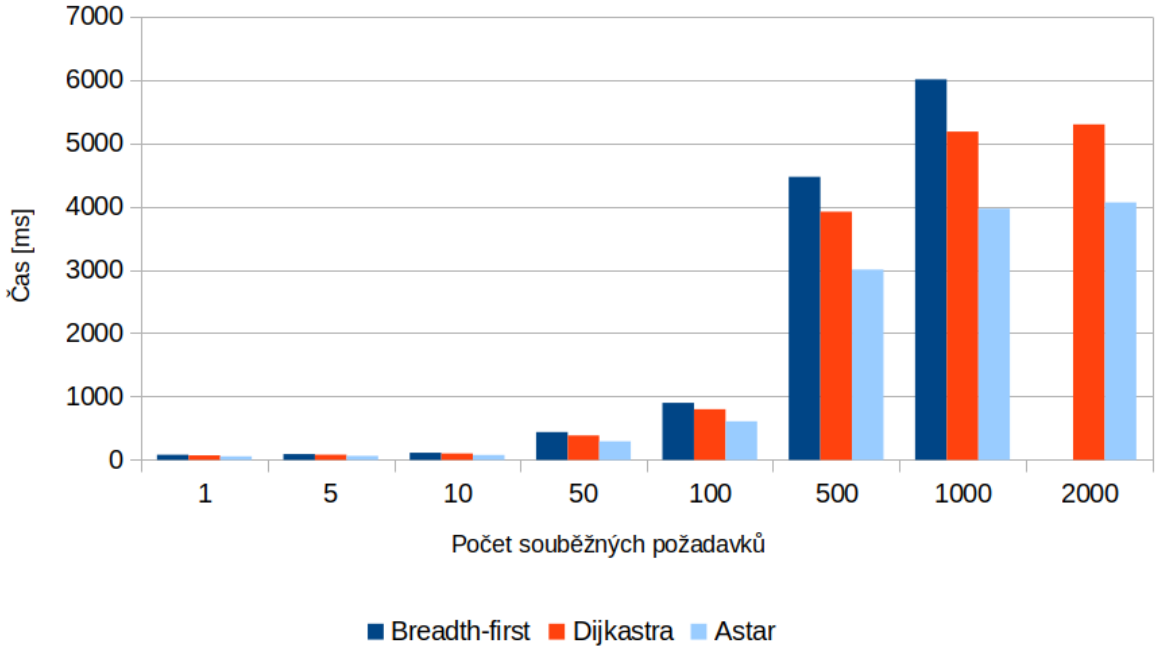
[1] `https://phpunit.de/` PHPUnit - The PHP Testing Framework
[2] `https://junit.org/junit5/` JUnit 5

5.2.1 Srovnání algoritmů

Z výsledků měření vyšel nejlépe algoritmus Astar. Byl proto vybrán jako výchozí vyhledávací algoritmus. Z důvodu zahazování požadavků při velké zátěži nejsou u algoritmu Breadth-fist uvedeny veškeré naměřené údaje.



Obrázek 5.1. Srovnání algoritmů - nejdelší doba trvání vyřízení jednoho požadavku



Obrázek 5.2. Srovnání algoritmů - průměrná hodnota trvání vyřízení jednoho požadavku

5.3 Zpracování hlasových příkazů

Hned na začátku testování se objevil první problém. DeepSpeech nedokázal zpracovat audio záznam s nízkou kvalitou. Toto se vyřešilo změnou použitého zdroje pro záznam. Hned na to se objevil další, jak anglický tak český akustický model nedokáže převést názvy místností používající kombinaci písmena a navazujících číslic, což je v rámci budov ČVUT značný problém. Převod bez problémů funguje jen u specifických názvů místností, optimálně těch, které mají v systému uvedený celý název. Poslední problém vyvstal s celkovou dobou zpracování požadavku. Na testovacím nezatíženém serveru trvalo zpracování v poměru 2 : 1, tj. 2 sekundy audio záznamu se zpracovávali 1 sekundu. Předchozí výčet problémů vedl k přerušení praktického testování a omezení jen na automatické testy pro zpracování výstupního textu.

Kapitola 6
Závěr

V průběhu vývoje došlo k několika podstatným designovým změnám, které ale měli ve výsledku větší dopad na frontedovou část realizovanou kolegou Richardem Burkoňem. V rámci implementace došlo k vypracování veškeré plánované funkcionality. Požadavky pro nasazení do ostrého provozu ale nesplnila doplňková funkcionalita pro podporu hlasového zadávání. Problém s nerozeznáváním názvů místností je potencionálně řešitelný doplněním akustického modelu o podporu pro místnosti začínající písmenem a navazujícím číslem. Dlouhá prodleva a z toho vznikající vysoká zátěž při zpracování je možná zkrátit použitím grafického akcelerátoru na serveru. Takové řešení ale značně navýší provozní náklady. Alternativně by bylo možné pro převod hlasu využít služby třetí strany, což by ale opět vedlo k navýšení provozních nákladů. To v tuto chvíli staví tento problém do neřešitelné pozice a proto byla z ostrého provozu podpora pro hlasové zadávání odstraněna.

V rámci backendové části je mnoho prostoru pro případná rozšíření. Za pravděpodobně nejpřinosnější navazující funkcionalitu lze označit možnost převodu rastrové, či případně vektorové mapy do formátu půdorysu používané systémem.

Další směr pro rozšíření se nabízí v oblasti integrace s dalšími systémy v rámci ČVUT. Například propojení rozvrhu studenta, kdy by aplikace dokázala automaticky v daný čas zobrazit trasu do učebny kde má mít student právě výuku.

Osobně považuji práci za přínosnou, protože se jedná o řešení, které bylo nasazeno do reálného provozu a nebylo v den odevzdání uloženo do šuplíku. Při práci samotné jsem si vyzkoušel novou funkcionalitu v rámci PHP 8 a prohloubil některé znalosti získané v rámci studia. Za sebe doufám, že aplikace usnadní nejen novým studentům jejich pobyt na škole, ale i příchozím návštěvníkům.

Literatura

[1] *TIOBE Index for December 2021 [online.*
https://www.tiobe.com/tiobe-index/.

[2] *The complete History of Java Programming Language.*
https://www.geeksforgeeks.org/the-complete-history-of-java-programming-language/.

[3] *Oracle Java SE Support Roadmap.*
https://www.oracle.com/java/technologies/java-se-support-roadmap.html.

[4] *History of PHP .*
https://www.php.net/manual/en/history.php.php.

[5] *Usage statistics of PHP for websites.*
https://w3techs.com/technologies/details/pl-php.

[6] *PHP 8.0 JIT Is Offering Very Compelling Performance Ahead Of Its Alpha.*
https://www.phoronix.com/scan.php?page=article&item=php8-jit-june&num=3.

[7] *The Best PHP Framework for 2015: SitePoint Survey Results.*
https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results.

[8] *A History of MySQL Database: UNIREG to Open Source to Profitability.*
https://docs.oracle.com/cd/B19306_01/server.102/b14200/intro001.htm.

[9] *PostgreSQL: Documentation: 14: 2. A Brief History of PostgreSQL.*
https://www.postgresql.org/docs/current/history.html.

[10] *The PostgreSQL Licence (PostgreSQL).*
https://opensource.org/licenses/postgresql.

[11] *Most popular databases in 2020 and the new trends.*
https://www.eversql.com/most-popular-databases-in-2020/.

[12] *History of SQL.*
https://exadel.com/news/old-reliable-mysql-history/.

[13] *Oracle Buys Sun.*
https://www.oracle.com/corporate/pressrelease/oracle-buys-sun-042009.html.

[14] *Architectural Styles and the Design of Network-based Software Architectures.*
https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[15] *A Brief History of SOAP.*
https://www.xml.com/pub/a/ws/2001/04/04/soap.html.

[16] *What is the GraphQL Foundation? — GraphQL.*
https://graphql.org/foundation/.

[17] Stuart Russell a Peter Norvig. *Artificial Intelligence - A Modern Approach.* Edition 3 vydání. Pearson: Pearson, 2010. ISBN 0-13-604259-7.

[18] *Introduction to A-Star.*
http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.

[19] *Algoritmus A-Star*.
http://voho.eu/wiki/algoritmus-a-star.

[20] *10 Best Free Linux Speech Recognition Tools - Open Source Software - LinuxLinks*.
https://www.linuxlinks.com/best-free-linux-speech-recognition-tools-open-source-software/.

[21] *GitHub - mozilla/DeepSpeech: DeepSpeech is an open source embedded (offline, on-device) speech-to-text engine which can run in real time on devices ranging from a Raspberry Pi 4 to high power GPU servers*.
https://github.com/mozilla/DeepSpeech.

[22] *GitHub - comodoro/deepspeech-cs: Czech deepspeech STT model*.
https://github.com/comodoro/deepspeech-cs.

[23] *Difference Between MVC and MVP Patterns — Baeldung*.
https://www.baeldung.com/mvc-vs-mvp-pattern.

Příloha A
Seznam použitých zkratek

BSD	Berkeley Software Distribution
CRUD	Create, Read, Update, Delete
ČVUT	České vysoké učení technické v Praze
DAO	Data access object
DARPA	Defense Advanced Research Projects Agency
DI	Dependency injection
FIFO	First In, First Out
GPL	General Public License
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JIT	Just In Type
JSON	JavaScript Object Notation
LAMP	Linux, Apache, MySQL, PHP
MPL	Mozilla Public License
MySQL	My Structured Query Language
Oauth	Open Authorization
ORDBMS	Object Relational Database Management System
PHP	PHP Hypertext Preprocessor
RDBMS	Relational Database Management System
REST	Representational state transfer
SDL	Schema Definition Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
STT	Speech to text
TTS	Text To Speech
URI	Uniform Resource Identifier
WSDL	Web Services Description Language
XML	Extensible Markup Language

Příloha B
Obsah přiloženého média

- /database - zakládací skript pro databázi
- /docs – dokumentace PHP API
- /restapi - detailnější popis REST API
- /src - zdrojové kódy backendu
- /thesis - kopie tohoto dokumentu
- readme.txt - informace k obsahu

Příloha C
Zadání práce

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Krejčí** Jméno: **David** Osobní číslo: **322953**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Backend iQR navigačního systému

Název bakalářské práce anglicky:

Backend of iQR navigation system

Pokyny pro vypracování:

Cílem bakalářské práce je kompletní tvorba backendu indoor QR Navigačního systému (iQNavs). Návrh backendu bude zohledňovat a především rozšiřovat prvotní koncept ověřený v rámci semestrálního projektu. Při návrhu bude student využívat frontend navržený kolegou v rámci jeho DP. V rámci BP zrealizujte kompletní systém, jehož klíčové součásti a vlastnosti budou:
a. Návrh a realizace kompletního backendu iQNavs (návrh plánovacích algoritmů, databázového modelu, zahrnutí preferencí v plánování, tj. bezbariérově, atd.).
b. Návrh a implementace backendového rozhraní pro správu mapových podkladů (serverová validace vkládaných dat, komunikace s databází, přihlašování, atd.).
c. Návrh uživatelsky přívětivého způsobu komunikace s uživatelem (hlasové zadávání, našeptávání, atd.).
d. Otestování v reálných podmínkách (vícepatrové budovy, atd.).

Seznam doporučené literatury:

[1] Hassan A. Karimi: Indoor Wayfinding and Navigation, CRC Press Taylor & Francis Group, ISBN, 978-1-4822-3085-7 (eBook - PDF), 2015.
[2] Steven M. LaValle: Planning Algorithms, 2006, Cambridge University Press
[3] Learning PHP, MySQL "Java Script – with jQuery, CSS" HTML5, O'Reilly Media, 2018, ISBN: 9781491978917

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Šipoš, Ph.D. katedra měření FEL (13138)

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2021** Termín odevzdání bakalářské práce: **15.08.2022**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Martin Šipoš, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta