TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Aplikace pro podporu studentské mobility TUL

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Matěj Beran**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Aplikace pro podporu studentské mobility TUL

Jméno a příjmení: **Bc. Matěj Beran**
Osobní číslo: M19000149
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie
Zadávající katedra: Ústav nových technologií a aplikované informatiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Seznamte se s problematikou progresivních webových aplikací (PWA), včetně problematiky automatizovaného testování, zveřejnění aplikace v produkčním režimu a poskytování uživatelské podpory.
2. Navrhněte PWA pro podporu studentských mobilit v rámci programu Erasmus a podobných. Při návrhu využijte předchozí vytvořený prototyp i nově získané teoretické poznatky. Navrhněte také systém pro práci se zpětnou vazbou uživatelů a poskytování uživatelské podpory.
3. Návrh prakticky implementujte a zveřejněte, včetně poskytování podpory uživatelům aplikace.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	40 – 50 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] ATER, Tal. *Building progressive web apps: bringing the power of native to the browser*. Beijing: O'Reilly, 2017. ISBN 978-1491961650.
[2] *Vue.js Guide* [online]. [cit. 2020-04-29]. Dostupné z: https://vuejs.org/v2/guide/.

Vedoucí práce:	Mgr. Jiří Vraný, Ph.D. Ústav nových technologií a aplikované informatiky
Datum zadání práce:	19. října 2020
Předpokládaný termín odevzdání:	17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2020

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

17. května 2021 Bc. Matěj Beran

Aplikace pro podporu studentské mobility TUL

Abstrakt

Cílem této práce bylo navrhnout a implementovat aplikaci pro podporu studentské mobility v rámci programu Erasmus+. Projekt se zabývá problematikou progresivních webových aplikací a jejich využitím pro vývoj univerzální multiplatformní responzivní aplikace. V byl kladen důraz na to, aby výsledná aplikace podporovala dobré praktiky pro vývoj takového typu aplikace. To zahranuje automatizované testování a nástroje pro udržování kvality a jednotného stylu kódu za účelem zajištění kontinuity budoucího vývoje aplikace.

Aplikace je navržena podle architektury klient-server, kde serverová část byla vytvořena v jazyce PHP za použití frameworku Laravel. Jejím cílem je poskytování dat klientské části aplikace. Ta byla vyvinuta pomocí frameworku VueJS s nadstavbou NuxtJS. Aplikace využívá této architektury pro nezávislý a paralelní vývoj obou částí skrze pevně dané REST rozhraní.

Hlavním přínosem této práce je usnadnit a zvýšit bezpečnost procesu výjezdu studentů skrze program Erasmus+ a zároveň tím usnadnit práci zahraničnímu oddělení TUL. Aplikace poskytuje studentům důležité informace, které jsou jim pak k dispozici v případě, že se dostanou do nějaké z krizových situací. Těmi mohou být například ztráta dokladu, dopravní nehoda nebo jakýkoliv trestný čin a další. Aplikace obsahuje jednoduchý systém pro práci se zpětnou vazbou, který studentům umožňuje odeslat názor nebo připomínku. Naopak uživatelům s privilegovaným přístupem umožňuje tuto zpětnou vazbu revidovat a dělat na základě toho další rozhodnutí.

Klíčová slova: PHP, Laravel, PWA, Erasmus, VueJS, NuxtJS, JavaScript, REST

Application for TUL students mobility assistance

Abstract

The aim of this work was to design and implement an application to support student mobility within the Erasmus+ program. The project deals with the issue of progressive web applications and their use for the development of universal multiplatform responsive applications. Emphasis was placed on the resulting application to support good practices for the development of this type of application. It includes automated testing and tools to maintain quality and consistent code style to ensure the continuity of future application development.

The application is designed according to the client-server architecture, where the server part was created in PHP using the Laravel framework. Its goal is to provide data to the client part of the application. It was developed using the VueJS framework with the NuxtJS extension. The application uses this architecture for independent and parallel development of both parts through a fixed REST interface.

The main benefit of this work is to facilitate and increase the security of the student departure process through the Erasmus+ program and at the same time to facilitate the work of the TUL foreign department. The application provides students with important information, which is then available to them in case they get into any of the crisis situations. These can be, for example, the loss of a document, a traffic accident or any criminal offense and others. The application contains a simple system for working with feedback, which allows students to send an opinion or comment. On the contrary, it allows privileged users to review this feedback and make further decisions based on it.

Keywords: PHP, Laravel, PWA, Erasmus, VueJS, NuxtJS, JavaScript, REST

Poděkování

Děkuji vedoucímu diplomové práce Mgr. Jiřímu Vranému, Ph.D za cenné rady, připomínky a metodické vedení práce.

Obsah

Seznam obrázků 10

Seznam zkratek 12

Úvod **13**

1 Progresivní Webové Aplikace **15**

1.1 Service worker 17

1.2 Manifest 18

1.3 Výhody 19

1.4 Nevýhody 21

2 Návrh aplikace **22**

2.1 Architektura 23

2.2 Technologie 24

2.3 Serverová část 24

2.3.1 Návrh REST API 25

2.3.2 Návrh databáze 26

2.3.3 Autentifikace a JWT 30

2.4 Klientská část 30

3 Tvorba aplikace **32**

3.1 Server . . . 32

3.1.1 Struktura projektu . . . 33

3.1.2 Proces autentifikace . . . 34

3.1.3 Generování JWT tokenu . . . 35

3.2 Klient . . . 36

3.2.1 Struktura projektu . . . 36

3.2.2 Komunikace s API . . . 38

3.2.3 Generování otisku zařízení . . . 39

3.2.4 Registrace aplikace jako PWA . . . 40

3.2.5 Aktualizace verzí . . . 41

3.2.6 Grafická stránka aplikace . . . 42

3.3 Testování . . . 43

3.3.1 Statická analýza kódu a stylizace kódu . . . 44

4 Uživatelská podpora **45**

4.1 Systém pro práci se zpětnou vazbou . . . 45

4.2 Logování klientské aplikace . . . 46

5 Nasazení aplikace **47**

5.1 Přepisování URL apachem . . . 48

5.2 Verzování . . . 49

6 Závěr **50**

6.1 Dosažené výsledky . . . 51

6.2 Další možnosti vývoje 52

Příloha I - zdrojové kódy **56**

Seznam obrázků

1.1 Schopnosti vs dosah u PWA, zdroj: https://web.dev/what-are-pwas/ 15

1.2 Android instalace PWA 16

1.3 iOS instalace PWA 16

1.4 Průzkum nejvíce oblíbených programovacích jazyků od společnosti Stack Overflow 20

2.1 API architektura 23

2.2 API specifikace v Apiary 26

2.3 Původní návrh databáze 27

2.4 Porovnání rychlosti databáze 28

2.5 Nové schéma databáze 29

2.6 Průzkum nejoblíbenějších frameworků od Stack Overflow za rok 2019 31

3.1 Struktura projektu serverové části aplikace 33

3.2 API autentifikace 35

3.3 Struktura klientské části aplikace 37

3.4 Komunikace klientské aplikace s API 39

3.5 Logika aktualizace aplikace 41

3.6 Ukázka vzhledu aplikace na desktopu 42

Seznam zkratek

PHP	PHP: Hypertext Preprocessor
REST	Representational state transfer
PWA	Progressive Web Application
PNG	Portable Network Graphics
API	Application programming interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
DOM	Document Object Model
JSON	Javascript Object Notation
SPA	Single Page Application
JWT	JSON Web Token
URL	Unirform Resource Locator
NFC	Near Field Communication
ORM	Object-relational Mapping
UUID	Universally Unique Identifier

Úvod

Progresivní webové aplikace (PWA) se za posledních 5 let od své specifikace dostali do velmi silné pozice mezi webové a nativní aplikace. Za toto období je adoptovalo mnoho technických i komerčních společností díky své versatilitě a nižším nákladům na vývoj. Jsou velmi rychlé, paměťově nenáročné a jejich distrubuce ke koncovým uživatelům je také jednodušší, protože je není nutné distribuovat skrze internetové obchody, jako například Google Play nebo AppStore.

Tato práce navazuje na předchozí prototyp aplikace, který byl zpracován jako ověření konceptu během diplomového projektu. Zadání vzešlo z iniciativy zahraničního oddělení TUL, které tento program organizuje a zaštiťuje. To se potýkalo se studenty, kteří často neznali všechna úskalí výjezdu za hranice naší země. Cílem je to, aby studenti vyjíždějící přes program Erasmus+ za hranice našeho státu měli k dispozici aplikaci, která jim tento výjezd zpříjemní. Studentům TUL bude ve fázi před výjezdem sloužit jako itinerář a během výjezdu jako místo, kde najdou všechny důležité informace pro případ, že by se dostali do nějaké krizové situace.

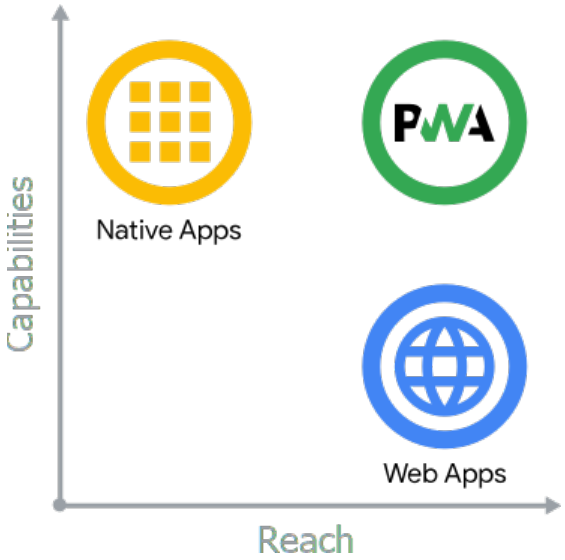
Práce si klade za cíl, aby student měl k dispozici aplikaci, která bude poskytovat jasný a srozumitelný přehled všech náležitostí, které je nutné si před výjezdem za hranice zařídit nebo obstarat. Aplikace studenta provede jednoduchým formulářem, který ho v logicky navazujících krocích seznámí se všemi nezbytnostmi výjezdu. Jedná se například o nahlášení výjezdu osobě blízké, pojištění nebo zjištění si kontaktů na integrované záchranné složky dané země. Také studenty vyzývá k tomu, aby se zaregistrovali do systému DROZD [1]. Tento systém zaštiťuje Ministerstvo zahraničních věcí České republiky a každý občan se zde může dobrovolně registrovat, pokud cestuje za hranice. Jeho benefit se nachází v tom, že pak dokáže účinně organizovat pomoc českým občanům v případě přírodních katastrof nebo sociálních nepokojů. Následně si pak student aplikaci nainstaluje do svého mobilního zařízení, kde bude mít tyto informace k dispozici, a to i v případech, kdy nemá přístup k internetu.

První kapitola se zaměřuje na PWA jako technologii. Co jí předcházelo a co definuje její základní charakteristiky. Také jsou zde uvedeny příklady, kdy je dobré tuto technologii využít a kdy je dobré raději zvolit nativní aplikaci pro danou platformu. Návrhu aplikace se věnuje navazující kapitola, kde se věnuji návrhu REST API a technologiím použitých pro obě části aplikace. Jsou zde popsané i nedostatky předchozího řešení a způsob, jakým je nové řešení eliminuje. Další kapitoly tento návrh překládají do praktické implementace a nasazení na server, který mi poskytl vedoucí práce.

1 Progresivní Webové Aplikace

Termínem „progresivní webové aplikace“ (PWA) popsali v roce 2015 inženýři z firmy Google takové aplikace, které jsou schopné využít nové funkce podporované v moderních prohlížečích [2]. Mezi tyto funkce se řadí push notifikace, online platby, geolokace nebo detekce toho, zda je zařízení připojené k internetu a mnoho dalších.

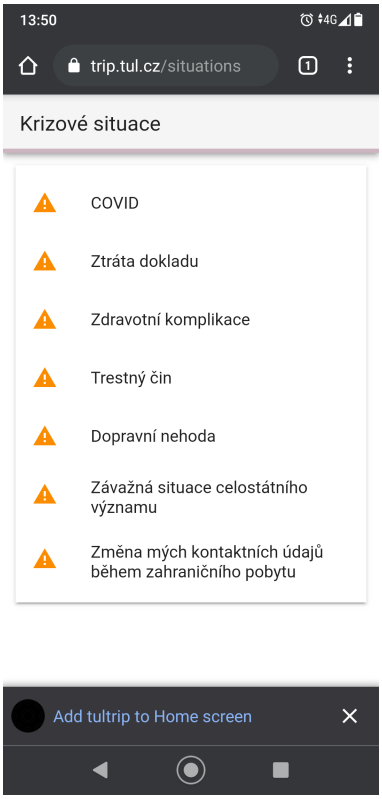
Jedná se tedy o rozšíření webových aplikací, čímž se nabízejí jako velmi silná alternativa nativních aplikací. Prakticky se jedná o aplikaci, která v sobě kombinuje výhody webových a nativních aplikací (viz Obrázek 1.1). Tu je pak možné nainstalovat a využít mnoho funkcí, ke kterým mají přístup nativní aplikace. Aplikace jako taková se pak nespouští uvnitř prohlížeče, jak je tomu zvykem u webových aplikací, ale naopak je spuštěna ve svém vlastním okně. To jí dodává dojmu nativní aplikace. Uživatel, který si aplikaci nainstaluje tedy vůbec nepozná rozdíl mezi tím, zda si nainstaloval PWA nebo nativní aplikaci pro svůj operační systém.



Obrázek 1.1: Schopnosti vs dosah u PWA, zdroj: https://web.dev/what-are-pwas/

Tyto aplikace je pak možné realizovat tak, že se do veřejného adresáře webu nebo

webové aplikace umístí service worker, manifest a ikona (logo) aplikace. Pokud je vše správně nakonfigurováno, tak už se jedná o PWA a při další návštěvě si může uživatel nainstalovat aplikace na své zařízení. Pro Android to funguje tak, že pokud uživatel na web přistoupí pomocí nativního prohlížeče, tak dostane vizuální notifikaci, zda si přeje aplikaci nainstalovat (viz Obrázek 1.2). Naopak společnost Apple není tak rychlá v implementaci této technologie a uživatel si na platformě iOS musí aplikaci připnout na plochu zařízení ručně (viz Obrázek 1.3).



Obrázek 1.2: Android instalace PWA

Obrázek 1.3: iOS instalace PWA

Konfigurace PWA se nemusí vždy provádět manuálně v tom smyslu, že by vývojář musel implementovat vlastní service worker a při každém sestavení aplikace měnil manifest. Frontendové frameworky jako jsou VueJS [3] nebo ReactJS [4] mají utility, které tento proces značně zjednodušují a automatizují. To hlavně z toho hlediska, že většina aplikací má na tento proces velmi podobné požadavky.

Základní kritéria pro to, aby se o webové stránce nebo aplikaci začalo mluvit jako o progresivní, by se dalo shrnout do těchto několika bodů:

- Jsou přístupné a instalovatelné z bezpečeného zdroje (HTTPS).
- Fungují i bez připojení k internetu (zajišťuje service worker).

- Obsahují Manifest s meta informacemi o aplikaci (název, popis, ikony, atd.).
- Mají logo ve formátu PNG o velikosti alespoň 144×144 px.

1.1 Service worker

Service worker je skript, který běží v pozadí prohlížeče odděleně od webové stránky [5]. Tím přináší nové funkce, které nevyžadují interakci webové stránky nebo uživatele. Service worker je součástí API prohlížeče a podporuje funkce jako jsou push notifikace nebo synchronizace dat na pozadí. Díky tomu je možné zajistit kvalitní offline funkcionalitu a mít nad ní i jako vývojář značnou kontrolu. Service worker nemůže přímo ovlivňovat DOM strukturu webové aplikace. Ta může se service workerem komunikovat pomocí rozhraní `postMessage` [6]. Například je díky tomu možné implementovat tlačítko, které po stisku vyvolá aktualizaci service workeru. Následně se může uživatel sám rozhodnout, zda aplikaci chce nebo nechce aktualizovat.

Před tím, než byla specifikace service workerů, se offline funkcionalita webových aplikací řešila pomocí technologie AppCache [7]. Ta byla také součástí API prohlížeče, ale už je nyní zastaralá. Bohužel tato technologie měla řadu nedostatků, kterým service workery předchází už tím, jakým způsobem byly navrženy. Jedním z těchto nedostatků bylo samotné cachování stránek, kde AppCache byla velice proaktivní a poskytovala cachovanou verzi stránek navzdory tomu, že aplikace již měla přístup k internetu.

V kontextu PWA je service worker primárně skriptovatelná síťová proxy, která umožňuje programově vyřizovat HTTP požadavky za pomoci vyrovnávací paměti. Ve své zásadě dokáže zpracovat tyto požadavky aplikace v situacích, kdy není k dispozici připojení k internetu nebo naopak je připojení velmi pomalé. Ale jak již bylo zmíněno výše, tak jeho jedinou funkcí není zajistit síťový provoz v nepříznivých podmínkách.

Tyto service workery jsou dále obohaceny o knihovnu workbox [8], která vznikla za účelem poskytnout vývojářům ucelené formy toho, jak může aplikace reagovat na jednotlivé HTTP požadavky. Workbox, stejně jako service worker, je součástí API prohlížeče, takže není potřeba do aplikace instalovat další balíčky nebo ji nijak samostatně konfigurovat. Níže je uvedeno několik strategií, které je možné využít:

- **Stale While Revalidate** – odpoví se rychle z cache, ale na pozadí se aplikace pokusí data aktualizovat.

- **Network First** – nejdříve se zkouší HTTP požadavek, a v případě selhání začne aplikace hledat záznam v cache.
- **Cache First** – nejdříve zkouší odpovědět pomocí cache, a pokud záznam neexistuje, tak pošle HTTP požadavek.
- **Network Only** – pouze posílání HTTP požadavků.
- **Cache Only** – pouze přístup do cache zařízení.

1.2 Manifest

Manifestem pro PWA se rozumí soubor, který obsahuje všechny důležité meta informace o aplikaci [9]. V případě PWA je to soubor ve formátu JSON, žádný jiný není podporovaný. Tento soubor hraje zásadní roli v tom, aby bylo možné aplikaci nainstalovat, potažmo umístit na domovskou obrazovku zařízení. Výsledný manifest může například vypadat takto:

```
{
  "name": "TULtrip",
  "short_name": "TULtrip",
  "description": "Erasmus+ aplikace pro podporu studentské mobility na TUL.",
  "icons": [
    {
      "src": "/_nuxt/icons/icon_64x64.fec11e.png",
      "sizes": "64x64",
      "type": "image/png",
      "purpose": "any maskable"
    },
    ...
  ],
  "start_url": "/?standalone=true",
  "display": "standalone",
  "background_color": "#f5f5f5",
  "theme_color": "#6C1D45",
  "lang": "cs"
}
```

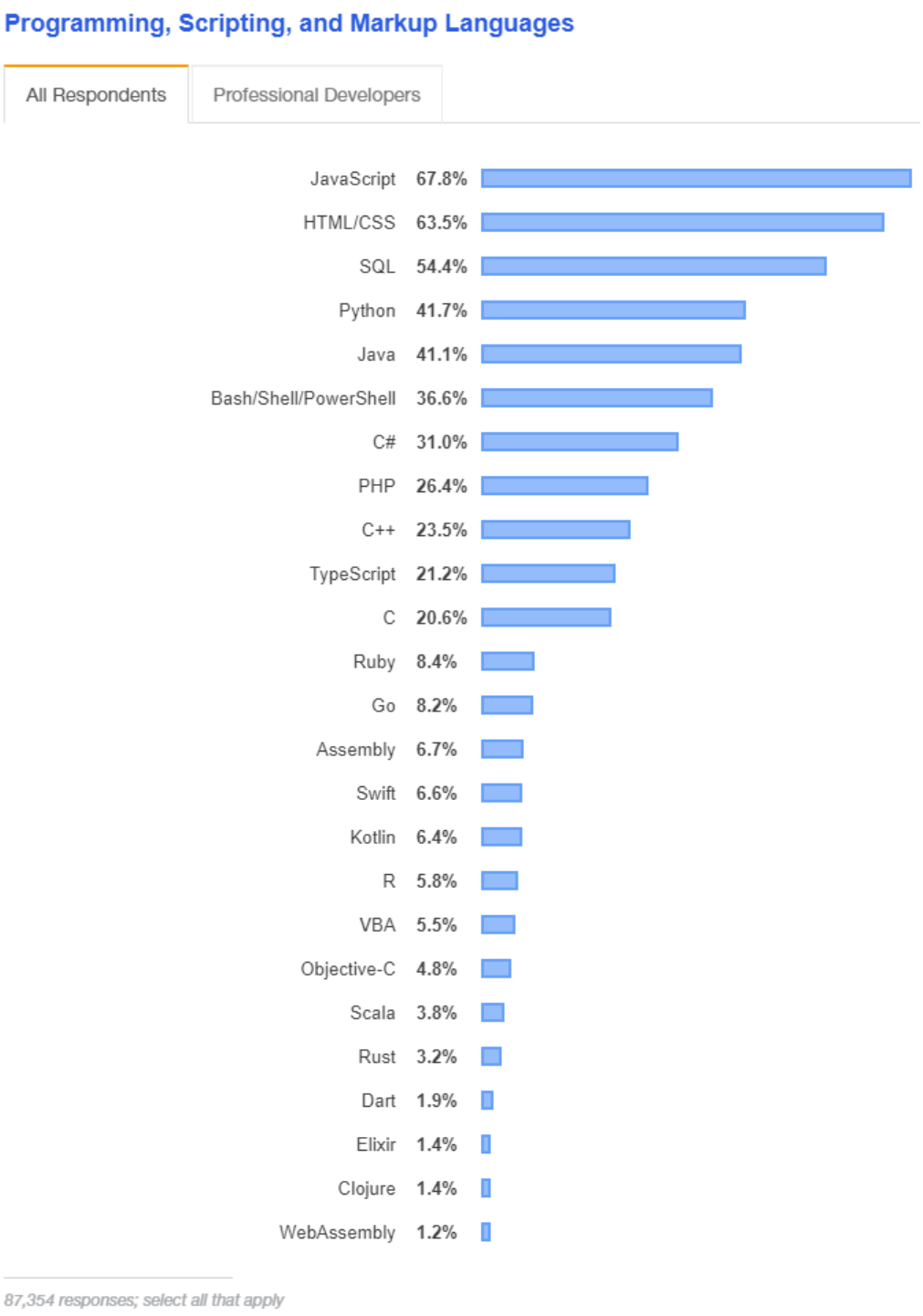
Tento manifest definuje plný název aplikace (`name`) a krátký název (`short_name`), kde krátký název se zobrazuje v seznamu aplikací. Dále je zde krátký popis toho, co aplikace dělá (`description`) a seznam ikon ve všech dostupných velikostech (`icons`). Poslední body jsou pak více specifické pro PWA, kde se definuje URL, na které se aplikace otevře (`start_url`) a položka `display`, která určuje, jestli se má aplikace otevřít uvnitř prohlížeče, nebo v samostatném okně. Poslední informace pak náleží lokalizaci a barevnému schématu aplikace.

1.3 Výhody

Hlavní výhodou PWA je, že fungují ve webovém prohlížeči, který má každý uživatel s přístupem k internetu. Aplikace jsou rychlé, paměťově nenáročné (řádově jednotky až desítky MB) a není nutné jejich distribuce skrze internetové obchody jako je Google Play nebo AppStore. PWA je nezávislá na platformě, protože webový prohlížeč funguje, až na výjimky, totožně na všech platformách.

V dubnu 2021 má 93 % uživatelů přístup k prohlížeči, který plně podporuje PWA (zahrnuto se service workery). Následně 86 % z těchto uživatelů si může PWA nainstalovat jako samostatnou aplikaci. Mezi operační systémy, které PWA podporují, patří Android, iOS, iPadOS, Windows 7 a vyšší, Xbox One, macOS, Linux 64 bit, Chrome OS a kaiOS [10].

Mezi velkou výhodu PWA také patří široká komunita kolem jazyka JavaScript, ve kterém se tyto aplikace implementují. Podle průzkumu z portálu Stack Overflow z roku 2019 [11] (viz Obrázek 1.4) byl Javascript 7 let v řadě nejpopulárnější programovací jazyk, kde ho jako oblíbený označilo 67 % z 87 354 respondentů.



Obrázek 1.4: Průzkum nejvíce oblíbených programovacích jazyků od společnosti Stack Overflow

1.4 Nevýhody

Hlavní nevýhodou PWA je to, že nedokáže plně využít všech vlastností softwaru a hardwaru tak, jako nativní aplikace. Proto jejich použití nebude tak dobré pro aplikace, které využívají periferie systému, jako například NFC, Proximity sensor nebo přístup ke kontaktům. Webová stránka What can web do today [12] poskytuje srovnání funkcí, ke kterým má prohlížeč přístup. Ta může být velice užitečná při rozhodování, zda aplikaci implementovat formou nativní aplikace nebo formou PWA.

2 Návrh aplikace

Hlavním cílem této práce bylo vyvinout multiplatformní responzivní aplikaci, která bude sloužit studentům TUL při výjezdu za hranice v rámci programu Erasmus+. Návrh vychází z požadavků zahraničního oddělení TUL a předchozího prototypního řešení. Aplikace má za úkol studentovi poskytnout ucelené informace o tom, co potřebuje pro výjezd za hranice. Potřebné věci si zaznamená do aplikace, které následně mu je poskytnout kdykoliv je bude potřebovat. Princip toho, jak se uživatel bude s aplikací seznámen je následující:

- Student vyjíždějící přes program Erasmus+ včas dostane odkaz na aplikaci.
- Aplikaci otevře na svém zařízení, to může být jakékoliv koncové zařízení.
- Při prvotním otevření aplikace na daném zařízení se ověří skrze centrální systém TUL.
- Po úspěšném přihlášení projde jednotlivé položky formuláře, kde se dozví o jednotlivých bodech, které je potřeba před odjezdem zvážit nebo zařídit.
- Není nutné vše vyplnit hned v daný moment, ale může informace vyplňovat průběžně.
- Jakmile má student vše zjištěné, tak se přihlásí do aplikace i na svém mobilním zařízení, a aplikaci si do svého zařízení nainstaluje.
- Pokud se přihlašuje poprvé, tak se opět autorizuje skrze centralní systém TUL a díky tomu se mu do zařízení synchronizují informace, které předtím vyplnil.

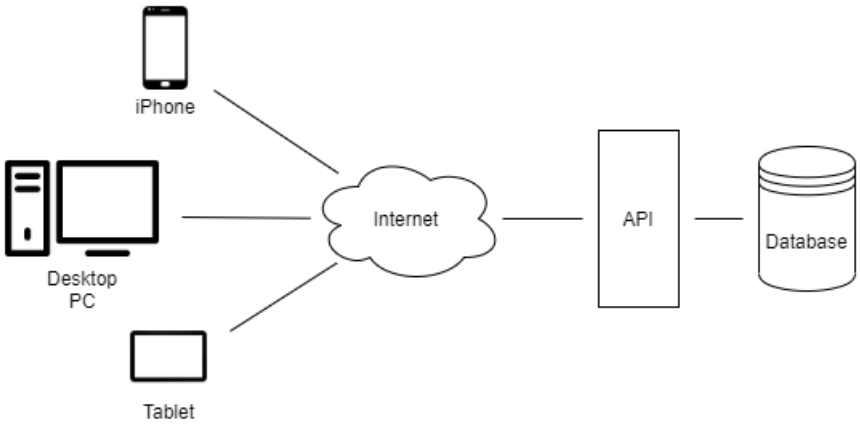
Takto nainstalovaná aplikace studentům slouží během výjezdu za hranice. Její hlavní přínos je především v tzv. krizových situacích. To znamená, že když například dojde k odcizení dokladů, zdravotním komplikacím, dopravní nehodě, trestnému činu atd., tak má k dispozici kontaktní informace na záchranné složky daného státu nebo kontakt na svého koordinátora v rámci TUL.

V aplikaci je možné provádět změny i poté, co si ji student nainstaluje. A to i navzdory tomu, že nemá přístup k internetu. Příklad může být vyplnění koordinátora v zahraničí, kterého dopředu nemusí znát. V případě, že nemá přístup k internetu, tak se data ukládají do lokálního uložiště začízení. Jakmile se zařízení k internetu připojí, dojde k synchronizaci dat, které zadal offline. Za „správná“ data se vždy považuje to, co student zadá na svém koncovém zařízení.

2.1 Architektura

V tomto směru nedošlo od prototypního řešení ke změně a zůstává architektura klient-server (viz Obrázek 2.1). Server prostřednictvím REST API poskytuje data ostatním klientům ve formátu JSON. Zabezpečení API je zajištěno pomocí JWT tokenu (více v kapitole 2.3.3) a unikátního otisku zařízení. To se generuje každému klientovi po přihlášení. Všechny požadavky na server se posílají asynchronně, a pokud bude uživatel připojený k internetu, tak při každé změně v datech si aplikace znovu data načte ze serveru, aby se udržovala v konzistentním stavu.

Toto řešení má tu výhodu, že odděluje od sebe data a uživatelské rozhraní. Je tedy možné kompletně vyvíjet klientskou a serverovou část aplikace nezávisle na sobě. Zároveň je možné vyvíjet novou verzi aplikace bez nutnosti měnit serverovou část a celkově klesají nároky na údržbu. V nespolední řadě je možné připojit více než jednu klientskou aplikaci prostřednictvím API.



Obrázek 2.1: API architektura

2.2 Technologie

U serverové části této aplikace je na výběr z mnoha jazyků a frameworků, kterými by bylo možné implementovat REST API. Již v prototypní etapě aplikace byl zvolen jazyk PHP, který toto zvládne stejně dobře jako jakákoliv jiná alternativa. Serverový kód nebude náročný, protože pouze čte data z databáze a serializuje je do formátu JSON, který je příjemný pro klientskou část aplikace. V předchozí iteraci aplikace byl zvolen framework Symfony, ale byl nahrazen za framework Laravel z důvodů, které uvádím v další kapitole. Součástí technologie je tedy PHP v7.4, MySQL v5.6 a Laravel v8.0.

Z hlediska klientské aplikace není důvod technologii měnit, ale došlo k refaktorování vnitřní struktury kódu. A to z důvodu toho, že během psaní prototypní aplikace byla tato technologie pro mě nová a ne ve všech případech jsem dodržoval dobré návrhové vzory a praktiky.

2.3 Serverová část

Serverová část aplikace již ve fázi prototypování byla implementována za pomoci frameworku Symfony, ale z několika důvodů došlo ke změně frameworku na Laravel. Oba z těchto frameworků jsou si velmi vyrovnané v tom, co zvládnou a ve světě PHP jsou sobě největšími konkurenty. Co je od sebe odděluje nejvíce je jejich fylozofie. Symfony je více podobné Javě a frameworku Spring, který je podobně jako Java určený spíše pro enterprise aplikace. Naopak Laravel se snaží do svého frameworku dodat co nejvíce jednoduchosti a volnosti směrem k vývojářům.

Symfony má velmi strmou učící křivku a psaní kódu zde vyžaduje mnoho práce a úsilí. To pro nováčka nebo někoho, kdo v projektu pokračuje, může být velmi skličující zkušenost. Zároveň využívá objektově-relačního mapování (ORM), což může být dvojsečná zbraň. ORM se snaží databázi abstrahovat do objektů, což může být problém, když programátor potřebuje hodně specifický dotaz. V tom přestává stačit pouhá znalost SQL, ale je potřeba se dobře orientovat v jazyce, kterým dotazy píše ORM. Nehledě na to, že ORM nemusí podporovat všechny příkazy, co podporuje instalace konkrétní databáze. Konkrétně v mém případě, pokud bych chtěl napsat například `ORDER BY FIELD`, tak na klíčovém slově `FIELD` ORM bez vlastní implementace selže, ačkoliv moje instalace databáze tento příkaz podporuje.

Naopak Laravel má v tomto případě navrch, ačkoliv se tam dá napsat mnoho „ošklivého“ kódu. Laravel nenutí tolik uživatele do svého ekosystému, jako to dělá Symfony, ale je to na úkor globálního namespace a velké míry statických volání. Laravel využívá jako abstrakční model databáze takový hybrid mezi nativním dotazem a ORM. Definuje něco, čemu se říká „Model“, který reprezentuje danou tabulku vně databáze. Ale narozdíl od Symfony uživateli nevnucuje Repository Pattern (ale jde velmi jednoduše implementovat) a dotaz se dá díky statickému voláni realizovat prakticky kdekoliv v aplikaci bez více obtíží. A pokud si člověk neví rady, tak poskytuje velmi kvalitní dokumentaci s velkým množstvím příkladů, od kterých se lze rychle odpíchnout a oproti Symfony bude mnohem příjemnější pro někoho, kdo bude na aplikaci pokračovat.

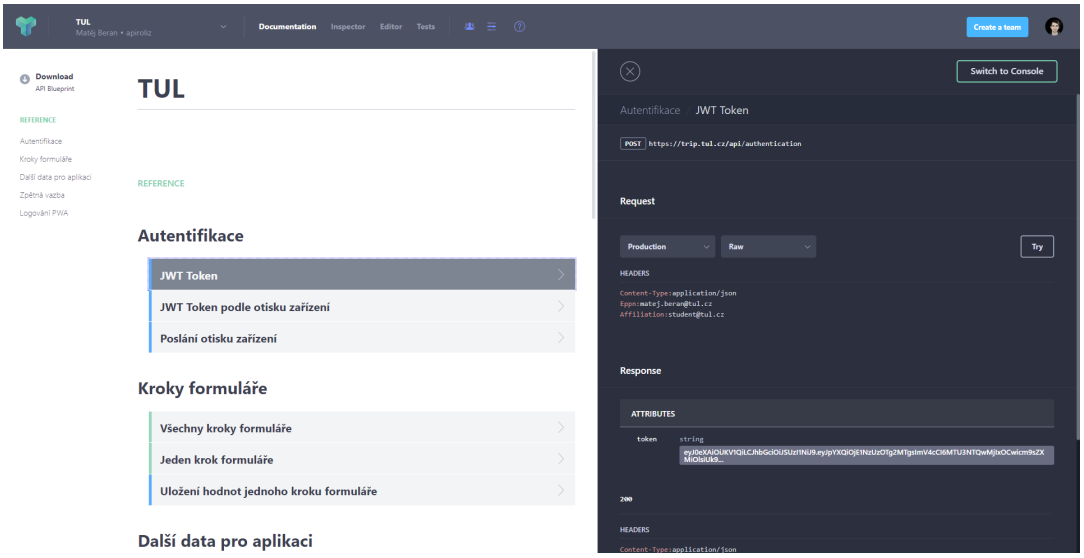
2.3.1 Návrh REST API

Architektura RESTových rozhraní není novinkou posledních let, ačkoliv jsou to tedy poslední roky, kdy jsem zaznamenal její velký nárůst. Velmi jednoduše definuje jednotlivé zdroje, které vždy posktyuje v textové podobě. Data se mohou vracet v různých formátech, ale v kontextu tohoto typu aplikace je nejrozšířenější formát JSON.

Již při návrhu prototypu aplikace byla pro definici API použita služba Apiary, která je vlastněná společností Oracle. Poskytuje jednoduchý editor, kde se v jazyce Markdown specifikuje tzv. *API Blueprint*. Tuto definici jsem upravil vzhledem ke změnám v databázi a rozšířil o nové endpointy pro logování a práci se zpětnout vazbou. Dalo by se to nazvat jako API specifikace. Výsledkem je pak dokumentace, kde jsou popsané jednotlivé API endpointy (viz Obrázek 2.2).

Každý typ enpointů je rozdělený do kolekcí, kde například `[GET] /steps` vrátí celou kolekci kroků formuláře. Pokud bych pak chtěl nějaký konkrétní prvek z kolekce, tak existuje endpoint `[GET] /steps/{stepId}`, který vrátí jeden konktrétní krok formuláře podle `id`. Touto typologií se řídí všechny ostatní endpointy API. Všechny tyto endpointy jsou idempotentní, takže pokud bych chtěl mazat zpětnou vazbu, která neexistuje, tak server odpoví stavovým kódem 204.

Takto definované API na Apiary má tu výhodu, že dovoluje vytvořit tzv. *Mock server*. Jedná se prakticky o testovací API server, díky kterému je možné vyvíjet klientskou část i serverovou část aplikace zároveň.



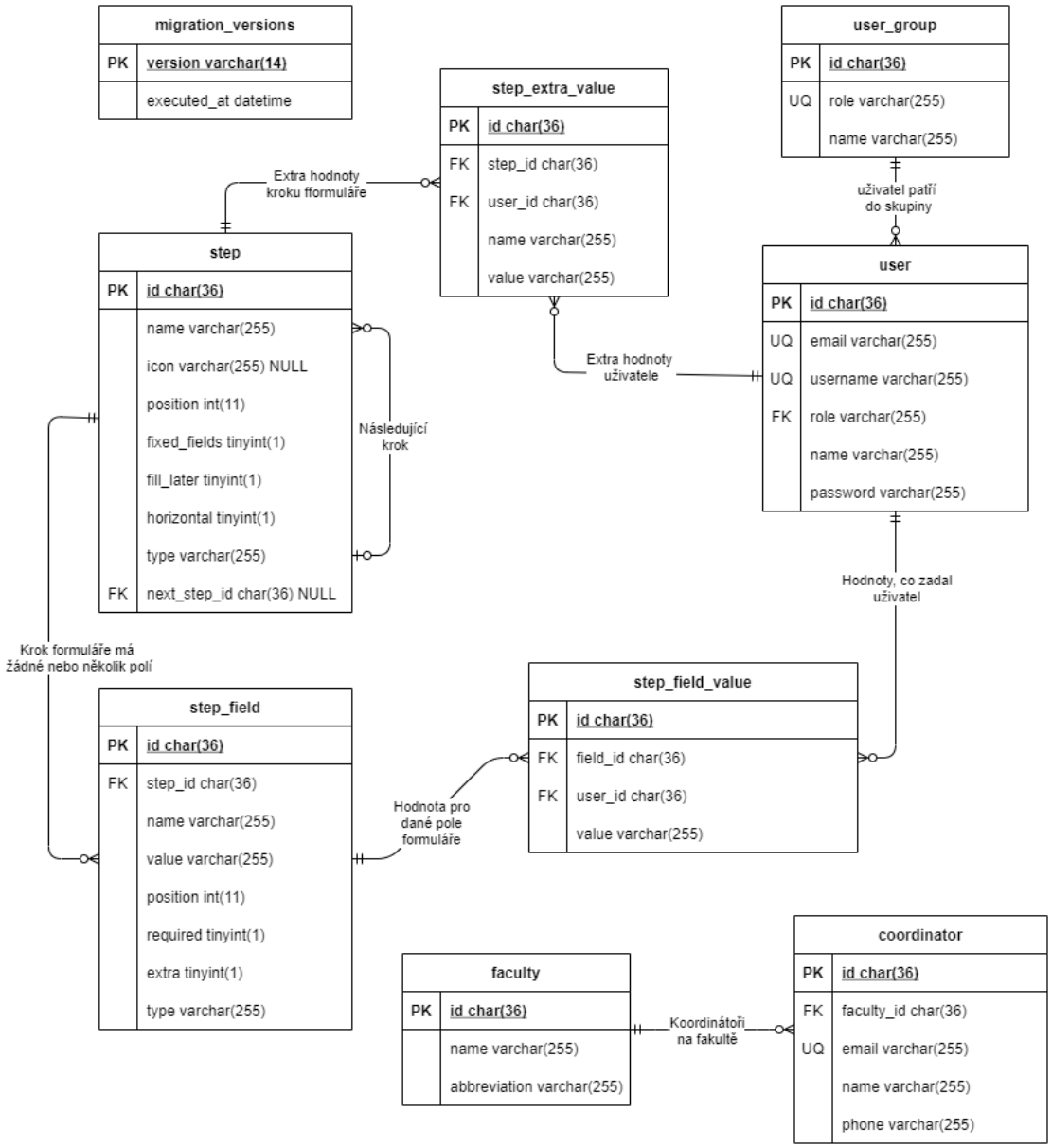
Obrázek 2.2: API specifikace v Apiary

2.3.2 Návrh databáze

Jak jsem psal v předchozích kapitolách, databáze již není generována automaticky. To mi umožnilo udělat klasický návrh databáze, kde jsem se zabýval návrhem jednotlivých tabulek namísto objektů.

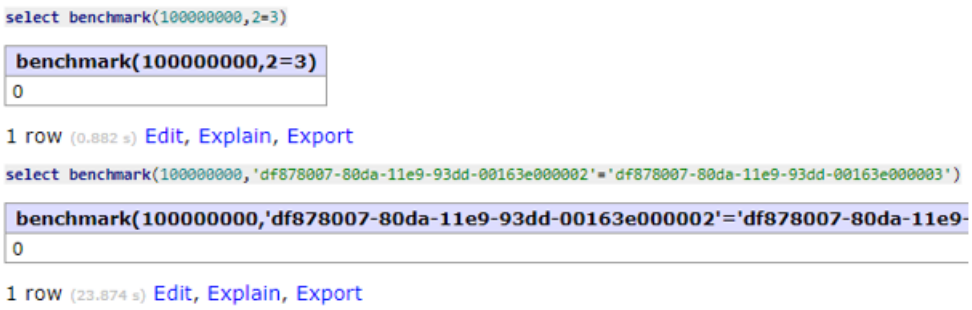
Původní schéma databáze mělo několik nedostatků (viz Obrázek 2.3). První z nich bylo použití UUID jako primárního klíče všech tabulek, kromě tabulky s migracemi. Z hlediska Symfony to bylo dobré v tom smyslu, že když jsem vytvořil nový objekt, který se později měl uložit do databáze, tak se s ním dalo plně pracovat hned po instanciaci. Standardně, když se v Symfony vytvoří objekt, který má primární klíč typu auto increment, tak nelze hned pracovat s jeho položkou unikátního identifikátoru. Musí se hned uložit do databáze a Symfony na pozadí aktualizuje primární klíč podle toho, jak ho přiřadila databáze. Nevýhoda je v tom, že pak určité tabulky mají více atritubutů, které je unikátně identifikují. V případě uživatelské skupiny to byl atribut `id` a atribut `role`, kde atribut `id` je zde zbytečný.

Další nedostatek obsahovala tabulka `step_field_value`. Zde se nacházel atribut `value`, kam se ukládala hodnota konkrétního pole pro konktétního uživatele. Problém byl v tom, že hodnotou mohla být i reference do jiné tabulky. Takže se de facto jednalo o cizí klíč, který reálně cizím klíčem nebyl.



Obrázek 2.3: Původní návrh databáze

Posledním nedostatkem, ačkoliv v tomto případě méně důležitým, je i rychlost dotazů, pokud v podmínce `WHERE` budeme hledat podle primárních klíčů. Pokud použijeme int místo uuid, tak dotaz může být až 24 krát rychlejší (viz Obrázek 2.4). Samozřejmě na obrázku níže je sice nejhorší scénář, ale přinejmenším poskytuje rozsah problému.

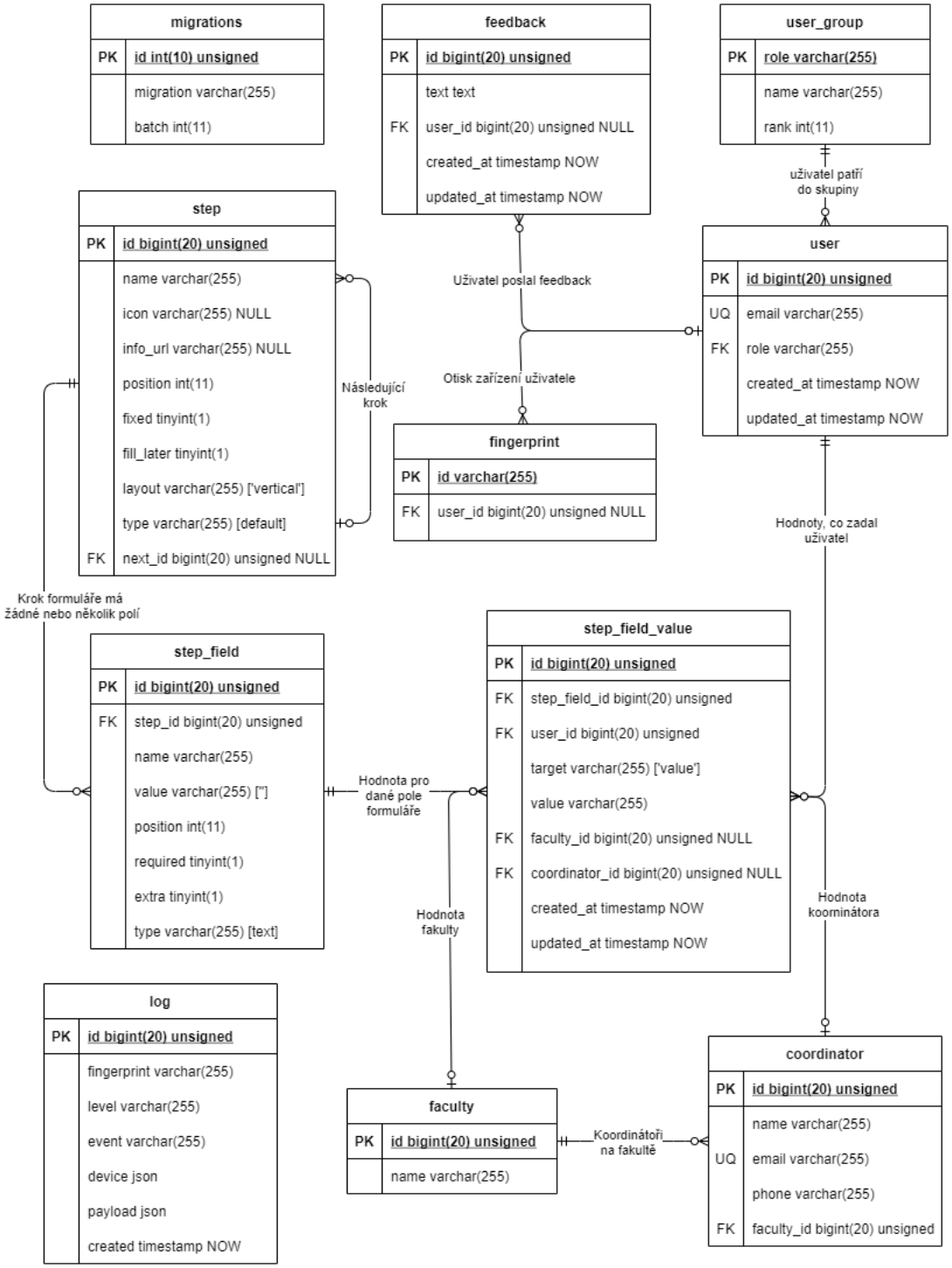


Obrázek 2.4: Porovnání rychlosti databáze

Všechny uvedené nedostatky a nekonzistence se snaží řešit nové schéma databáze (viz Obrázek 2.5), které je zároveň rozšířeno o zpětnou vazbu uživatelů, otisky zařízení a logování. Každá z tabulek databáze prošla renovací hlavně v oblasti primárních a cizích klíčů.

V uživatelské části jsem přidal k uživatelské skupině rank, což je číselná hodnota, díky které je možné omezovat přístup k určitým částem aplikace jako je například seznam zpětné vazby.

Problém s cizími klíči jako hodnotou konkrétního pole tabulky `step_field_value` jsem vyřešil tak, že jsem přidal sloupec `target`, který může nabývat hodnot `value`, `faculty_id` nebo `coordinator_id` podle toho, o jaké pole se jedná. Označuje se tím, který sloupec reprezentuje hodnotu. Také došlo ke sloučení `step_extra_value` do tabulky `step_field`, kde se přidal příznak `extra`. To je tedy hlavně pro položky, které se dají přidávat dynamicky uživatelem.



Obrázek 2.5: Nové schéma databáze

2.3.3 Autentifikace a JWT

Autentifikace uživatele v aplikaci probíhá ve dvou krocích. Uživatel se přihlásí přes centrální přihlašovací systém TUL (Shibboleth) a na základě toho, jestli se přihlášení podaří, se uživateli vygeneruje JWT token. Tímto tokenem se bude po zbytek komunikace identifikovat.

Výsledný token je řetězec, ve kterém jsou pomocí algoritmu base64 zakódovány informace, které uživatele identifikují. Skládá se z hlavičky, vlastních dat a podpisu. V hlavičce je uložen typ tokenu a algoritmus, kterým se vygeneroval podpis. Ve vlastních datech tokenu jsou informace identifikujícího uživatele. Poslední část, podpis, je generován tak, že hlavička a data se zakódují algoritmem base64, následně se spojí znakem `"."` a tento řetězec se zašifruje pomocí privátního a veřejného klíče za použití algoritmu, který je definován v hlavičce tokenu (např. HS256). Všechny tyto informace v řetězci jsou odděleny znakem `"."`. Výsledek může tedy vypadat například takto:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwi
bmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2
QT4fwpMeJf36POk6yJV_adQssw5c
```

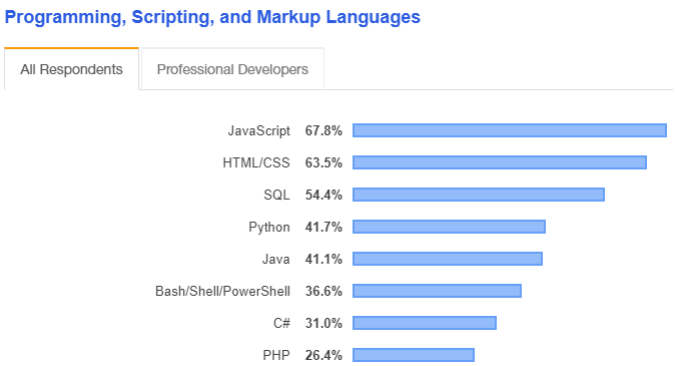
Tento token se po zbytek komunikace posílá v hlavičce požadavku jako pole `Authorization`. Hodnota tohoto pole je pak ve formátu `Bearer <token>`. V konkrétním případě pak může obsah hlavičky u tohoto pole vypadat následovně:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI...
```

2.4 Klientská část

Klientské PWA aplikace se dají realizovat pomocí řady frameworků – například VueJS, ReactJS, AngularJS nebo Ionic. Podle průzkumu Stack Overflow [11] (viz Obrázek 2.6) nejvyšší popularity dosahují frameworky VueJS a ReactJS (v průměru je označila 74 % respondentů jako oblíbený). Rozdíl je mezi nimi primárně v zápisu, kvalitě dokumentace a své křivce učení, kterou VueJS nemá tak strmou jako ReactJS a bude jednodušší pro toho, kdo na aplikaci potenciálně naváže.

Již ve fázi prototypování byla tato aplikace vyvíjena za pomoci frameworku VueJS. To není nutné měnit ani nyní. Stejně jako ReactJS, tak i VueJS poskytuje velmi kvalitní dokumentaci, ale jeho komponenty mají mnohem kompaktnější zápis bez používání objektů a dedičnosti.



Obrázek 2.6: Průzkum nejoblíbenějších frameworků od Stack Overflow za rok 2019

Aplikace disponuje nadstavbou pro VueJS – framework NuxtJS. Tento framework není tak významný z hlediska implementace, ale má velkou výhodu v tom, že zjednodušuje proces vývoje aplikace. Jeho velkou výhodou je způsob, jakým aplikaci standardizuje. Jedná se o:

- Strukturu projektu
- Konfiguraci
- Router a přesměrování v rámci aplikace
- Registrování dalších knihoven v projektu
- Integrace s knihovnami třetích stran (například i18n)
- Jednodušší implementace middleware a layoutu

3 Tvorba aplikace

V této části práce se budu zabývat tím, jak jsem prototypní řešení předělal do aplikace, která bude připravena pro to, aby ji mohli studenti začít používat. Cílem bylo, aby se aplikace testovala během školního roku, ale vzhledem ke globální situaci s pandemií to nebylo možné.

Při vytváření jsem se snažil dodržovat dobré programátorské praktiky tak, aby aplikace byla co nejvíce jednotná. Zároveň jsem se snažil aplikaci držet v takovém rámci, aby případní lidé, kteří budou na této aplikaci pokračovat, měli co nejjednodušší práci.

Během vývoje jsem také velmi úzce spolupracoval se zahraničním oddělením TUL. Takto úzká spolupráce velmi zrychluje vývoj zejména na začátku životního cyklu aplikace vzhledem k tomu, že každá nová funkce se dala velmi rychle implementovat a následně nasadit. Protože nebylo možné aplikaci poskytnout vyjíždějícím studentů, tak alespoň díky tomu jsem měl nějaké podněty ke zlepšení aplikace.

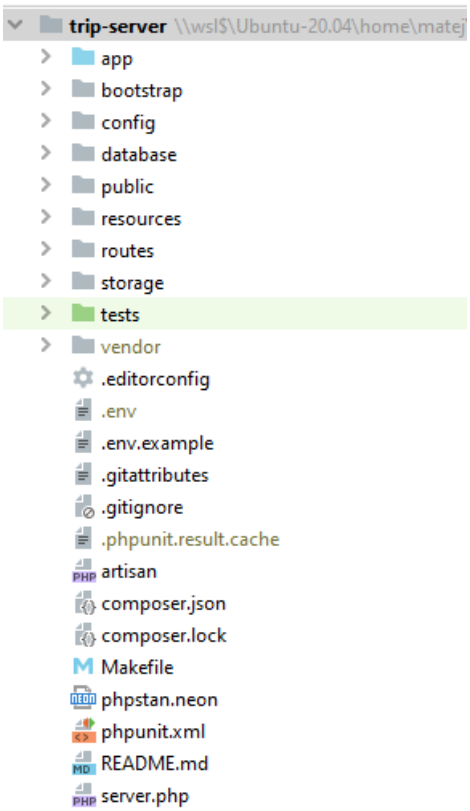
3.1 Server

Jak jsem již předesílal v předchozí kapitole návrhu aplikace (viz kapitola 2.3), tak pro serverovou část aplikace jsem využil framework Laravel ve verzi 8 spolu s databází MySQL ve verzi 5.7. Protože mám s PHP větší zkušenosti, a zároveň se jednalo o tu méně náročnější část aplikace, tak jsem tuto část implementoval jako první.

Úkolem serverové části aplikace je autentifikovat uživatele vůči centrálnímu systému TUL a persistentně ukládat jeho data do databáze. Tyto data pak poskytuje klientské aplikaci skrze REST rozhraní ve formátu JSON.

3.1.1 Struktura projektu

Struktura projektu odpovídá klasické struktuře, kterou má výchozí instalace Laravelu (viz Obrázek 3.1). V kořenovém adresáři projektu se pak nacházejí utilitní soubory, jako je třeba Makefile pro zjednodušení některých příkazů, konfigurace nástroje pro statickou analýzu kódu nebo seznam závislostí projektu. Jedním z hlavních souborů v kořenovém adresáři je soubor *artisan*. Jedná se o skript, díky kterému se dají třeba snadno vytvářet a spouštět databázové migrace nebo čistit mezipaměť projektu.



Obrázek 3.1: Struktura projektu serverové části aplikace

- **app** – zde se nachází hlavní logika aplikace, kde mezi důležité části patří registrace jednotlivých komponent, databázové modely, controllery a middleware.
- **bootstrap** – sestavuje aplikaci a cachují se zde komponenty containeru.
- **config** – zde se nacházejí konfigurační soubory jednotlivých logických celků aplikace. Je zde například konfigurace připojení databáze, cache nebo konfigurace logování.

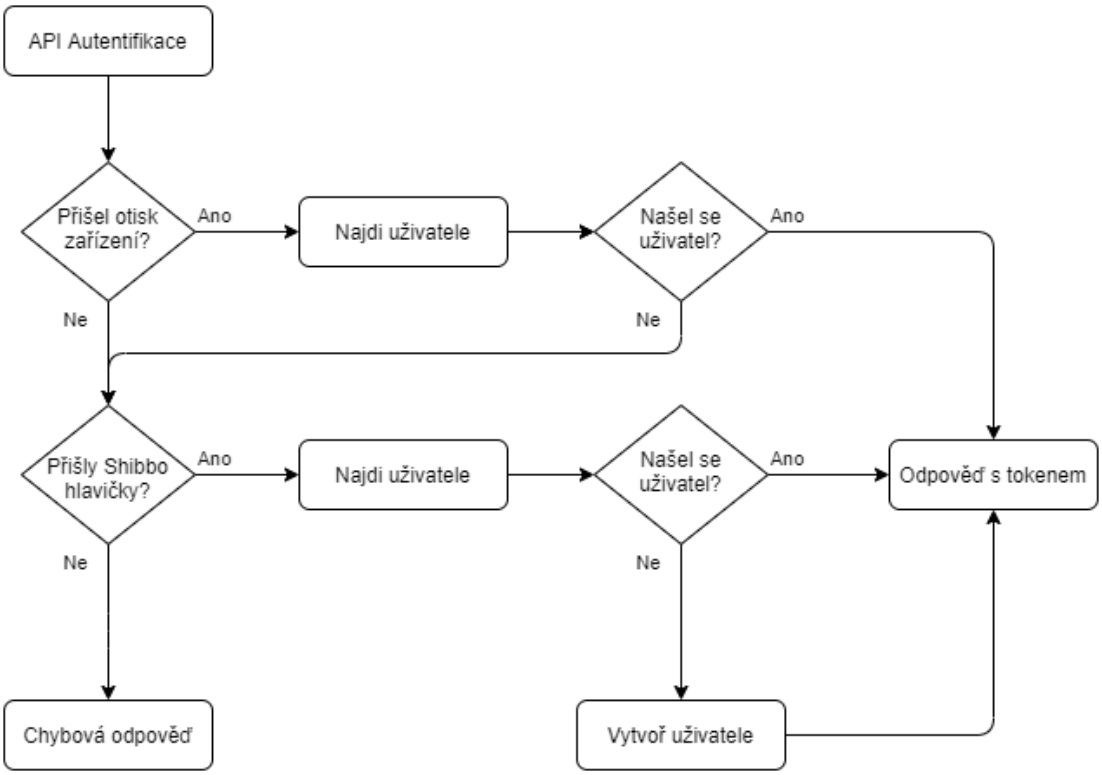
- **database** – obsahuje jednotlivé migrace databáze spolu s inicializační sadou dat aplikace. Nachází se zde také testovací SQLite databáze.
- **public** – veřejný adresář projektu. Nachází se zde `index.php` jakožto vstupní bod do aplikace.
- **resources** – z hlediska této aplikace nedůležitý adresář, ale mohou se zde nacházet šablony nebo překlady aplikace.
- **routes** – zde je popis jednotlivých API endpointů. Vlastně říká, které požadavky jsou zpracovány kterou částí aplikace.
- **tests** – zde se nacházejí všechny testy aplikace.
- **storage** – tento adresář je určený pro ukladání logů aplikací, session aplikace a dalších aplikačních souborů definovaných danou aplikací.
- **vendor** – obsahuje jednotlivé balíčky třetích stran, které jsou nainstalované skrze composer (balíčkovací systém pro PHP).

3.1.2 Proces autentifikace

Způsob, jakým se uživatel v aplikaci ověřuje, by se dal rozdělit do dvou částí. První částí je Shibboleth jako centrální systém, kterým se ověřují všichni v rámci univerzity. Druhou částí je pak autentifikace proti API pomocí JWT tokenu a otisku zařízení (viz Obrázek 3.2).

Student se tedy přihlásí skrze Shibboleth a je přesměrován na klientskou část aplikace. Při načtení aplikace pošle požadavek na API pro vygenerování tokenu, kterým se bude dále ověřovat. Pokud se student už někdy do aplikace přihlásil, tak se na základě otisku zařízení najde jeho záznam v databázi a jako odpověď se pošle token. Naopak když se jedná o první přihlášení, tak se kontrolují hlavičky `Eppn` a `Affiliation`. Hlavička `Eppn` obsahuje e-mail a `Affiliation` říká, zda se přihlásil student nebo zaměstnanec univerzity. Povinná je v tomto případě pouze hlavička `Eppn`, kde naopak hlavička `Affiliation` se nastaví na výchozí hodnotu `student@tul.cz` v případě, že není vyplněna.

Když si API vyřeší tuto vnitřní logiku a odpoví tokenem, tak se na klientské straně aplikace spočítá otisk zařízení a to se pošle na API, kde se uloží do databáze a přiřadí se k přihlášenému uživateli. Pokud uživatel již daný otisk má přiřazený, tak se dotaz ignoruje. Tohle umožňuje při příštím otevření aplikace přeskočit proces



Obrázek 3.2: API autentifikace

ověřování přes Shibboleth a hned začít komunikovat s API pomocí otisku zařízení, který jednoznačně identifikuje uživatele.

3.1.3 Generování JWT tokenu

Zabezpečení API jako takového je realizováno pomoctí JWT tokenu (viz kapitola 2.3.3). Prakticky je implementováno pomoctí balíčku jwt-auth [14], který je navržen specificky pro Laravel. Tento balíček vyžaduje pro svoji funkčnost registraci komponent v Laravelu a vygenerování klíče o délce 64 znaků. Vygenerování klíče je velmi jednoduché díky příkazu `php artisan jwt:secret`. Tento příkaz vygeneruje potřebný klíč a zapíše ho jako proměnnou `JWT_SECRET` do souboru `.env`, který se nachází v kořenovém adresáři projektu a rozšiřuje proměnné prostředí.

Druhou částí konfigurace balíčku je vybrat si entitu, kterou bude potřeba tokenem ověřit. V tomto konkrétním případě se jedná o databázový model `User.php`. Tato třída musí implementovat rozhraní `JWTSubjet`. Následně je nutné implementovat dvě metody, které vrací unikátní identifikátor enity, tedy unikátní identi-

fikátor uživatele a další tvrzení o uživateli. Jsou to metody `getJWTIdentifier` a `getJWTCustomClaims`.

V momentě, kdy je tímto způsobem provedená konfigurace, tak stačí implementovat a zaregistrovat middleware, který rozšiřuje ten základní poskytovaný knihovnou. Uvnitř se z HTTP požadavku vyparsuje token a následně se zkontroluje, zda je validní. Pokud validní není, tak aplikace hned vrací HTTP odpověď se statusem 401 a požadavek není propagován do vnitřní logiky aplikace.

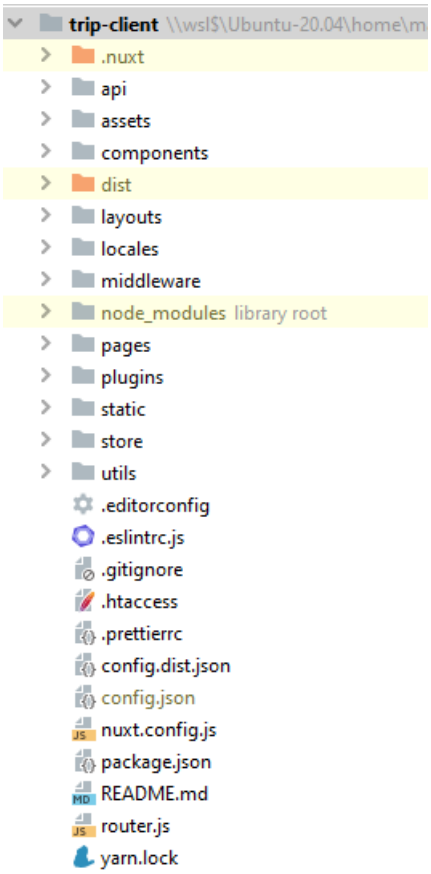
3.2 Klient

Klientská část aplikace je napsaná v jazyce JavaScript a frameworku VueJS s nadstavbou NuxtJS. VueJS je framework pro implementaci uživatelského rozhraní a sám o sobě poskytuje pouze systém jednoduchých komponent, které lze vykreslit na webové stránce. Je možné ho zakomponovat prakticky do jakéhokoliv projektu. Může se jednat například o velmi malou komponentu košíku, administrační panely, nebo celé weby a aplikace (SPA). Jeho struktura není nijak daná a celý proces je delegován na vývojáře. Zde přichází výhody nadstavby NuxtJS, která má definovanou adresářovou strukturou, jednoduchý konfigurační soubor a předdefinované komponenty. Zároveň je velmi rozšiřitelná sama o sobě, ale i s podporou komunitních balíčku, které například zjednodušují právě vývoj a konfiguraci PWA.

3.2.1 Struktura projektu

Adresářová struktura opět velmi odpovídá výchozí instalaci NuxtJS, ale je zde několik rozšíření (viz Obrázek 3.3). Mezi tyto adresáře patří adresář api, kde je logika pro získávání dat ze serveru a adresář utils s obecnými nástroji.

- **.nuxt a dist** – tyto adresáře obsluhuje NuxtJS a používá je při vývoji a sestavování aplikace, kde v adresáři dist se pak nachází kompletně sestavená aplikace.
- **api** – obsahuje moduly pro komunikaci s API serverem, kde tyto moduly jsou rozdělny podle toho, kterou sadu kolekcí obsluhují.
- **assets** – zde se nachazejí soubory, které nějakým způsobem ovlivňují vzhled aplikace. Pro tuto aplikaci je zde pouze ukázkový soubor, pokud se do budoucna bude více zasahovat do vzhledu aplikace.



Obrázek 3.3: Struktura klientské části aplikace

- **components** – tento adresář obsahuje všechny znovupoužitelné komponenty aplikace. Tyto komponenty se pak používají uvnitř komponent z adresáře pages.
- **layouts** – zde se nacházejí komponenty, které se používají jako základ pro komponenty z adresáře pages. Například pokud je nebo není uživatel přihlášený, nebo pokud došlo v aplikaci k chybě.
- **locales** – zde se nacházejí veškeré překlady a celkově lokalizace. Jedná se o soubory ve formátu JSON, a zatím obsahuje pouze lokalikzaci do českého a anglického jazyka.
- **middleware** – obsahuje jeden middleware, který kontroluje přítomnost autentifikačního JWT tokenu.
- **node_modules** – balíčky, které aplikace vyžaduje pro svůj běh. Jsou nainstalovány skrze balíčkovací systém nodejs za pomoci správce balíčků Yarn.
- **pages** – zde se nacházejí komponenty aplikace, které představují jednotlivé

stránky. Respektive je možné se na ně dostat skrze URL (obsluhovány přes `router.js`).

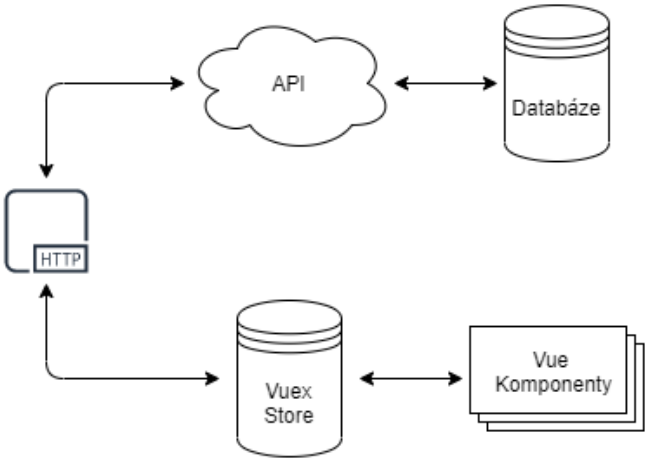
- **plugins** – zde jsou rozšíření, které jsem do aplikace dopsal. Nejedná se se o rozšíření třetích stran. Je zde například lokalizace, rozšíření pro service worker nebo registrace globálních proměnných aplikace.
- **static** – tento adresář slouží pro statické položky aplikace. Zpravidla se jedná o obrázky. V případě PWA se zde nachází i výsledný service worker.
- **store** – zde jsou všechny soubory, které souvisí se správou interního stavu dat. Opět jsou rozděleny podle jednotlivých logických celků aplikace.
- **utils** – zde se pak mohou nacházet jakékoliv utility, které jsou často využívány napříč aplikací.

3.2.2 Komunikace s API

Klientská aplikace komunikuje s API v několika krocích. Ale nejdříve bych zmínil, jak si aplikace jako taková data ukládá. Ve VueJS si standardně data drží sama komponenta, které je potřebuje. Pak ale vzniká problém s tím, že se data duplikují a je náročné do všech komponent aktualizovat změny jednotlivých stavů. Proto jsem pro správu interního stavu aplikace doinstaloval knihovnu Vuex, kterou VueJS oficiálně podporuje a doporučuje. Vuex data aplikace centralizuje do vlastního kontejneru. Tomuto kontejneru se v kontextu VueJS aplikací říká *store*. Díky tomu existuje v aplikaci jedno místo, kam se data ukládají, a všechny komponenty z něho čtou a nebo ho atomicky mění. Všechny změny se hned propagují zpět do všech komponent, které data používají.

Pro samotné HTTP požadavky na API jsem do aplikace doinstaloval knihovnu axios [15], která je obstarává. Druhou variantou bylo fetch API, které javascript obsahuje nativně pro posílání a zpracování HTTP požadavků. Ačkoliv obě varianty jsou velmi dobré, tak jsem nepotřeboval takovou granularitu, kterou poskytuje fetch API. Proto jsem nakonec zvolil axios, který je velmi jednoduchý na použití a dostatečně rozšiřitelný pro toto řešení. Tato knihovna je pak v aplikaci obalena do jednotlivých modulů, které zpracovávají jednotlivé API endpointy. Například pro zpracování zpětné vazby je zde modul `api.feedback.service.js`, který obsahuje všechny dostupné metody API. Jako zajímavou a dobrou věc u axiosu bych ještě zmínil jeho systém interceptorů, kdy každý požadavek je možné před jeho odesláním zachytit a ještě něco s ním udělat. To je v tomto konkrétním případě dobré pro

nastavení hlavičky Authorization a příslušného tokenu z API.



Obrázek 3.4: Komunikace klientské aplikace s API

Když se vrátím k samotnému procesu, jak aplikace získává data, tak je vyobrazen níže (viz Obrázek 3.4). Jako první nějaká komponenta aplikace zjistí, že potřebuje data ze storu aplikace. V tu chvíli store data buď poskytne a nebo pošle požadavek na API. Hned, jak data z API dostane, tak si aktualizuje svůj vnitřní stav a následně se data propagují i do všech komponent, které je potřebují.

3.2.3 Generování otisku zařízení

Kromě toho, že se uživatel do aplikace přihlašuje skrze Shibboleth, tak je autentifikace ještě doplněna o otisk daného koncového zařízení. Důvod je hlavně ten, aby se uživatel nemusel neustále přihlašovat skrze Shibboleth. Funguje to tak, že uživatel se přihlásí na svém zařázení skrze Shibboleth a po úspěšném přesměrování do aplikace se hned spočte otisk zařízení, který se pošle na API. Při příští návštěvě se spočte znovu a token se vygeneruje na základě toho, jestli API zařízení zná nebo ne (viz kapitola 3.1.2).

Otisk je možné generovat například pomocí knihovny FingerprintJS [16] nebo ClientJS [17]. V rozhodování mezi těmito knihovnami hrálo největší roli to, že knihovna FingerprintJS poskytuje kvalitnější rozhraní i dokumentaci. Zároveň se podle aktivity obou repozitářů na GitHubu dá usoudit, že FingerprintJS je mnohem aktivnější projekt, takže je do budoucna perspektivnější. Také poskytuje i placenou verzi jejich projektu, která zvyšuje přesnost identifikace zařízení na 99,5 %. Kvůli všem těmto důvodům jsem nakonec zvolil knihovnu FingerprintJS, která při gene-

rování otisku bere v potaz parametry systému a prohlížeče. Mezi tyto parametry patří například:

- Rozlišení obrazovky
- Typ a verze operačního systému
- Typ a verze prohlížeče
- Jazyk jak v prohlížeči, tak i v systému
- Hloubka barev zařízení
- Seznam nainstalovaných fontů
- Seznam nainstalovaných rozšíření do prohlížeče

3.2.4 Registrace aplikace jako PWA

Pro to, abych z aplikace udělal PWA, jsem použil balíček `@nuxtjs/pwa`, který obsahuje předpřipravenou šablonu service workeru, kterou je možné implementovat základní konfiguraci aplikace. Toto má výhodu v tom, že je možné rychle implementovat PWA bez vynaložení většího úsilí. Pro náročnější funkce to má i své nedostatky v tom, že daný service worker není až tak jednoduché rozšířit o vlastní kód a chybí příklady toho, jak použít určité parametry konfigurace. To ale v tomto projektu nebylo potřeba.

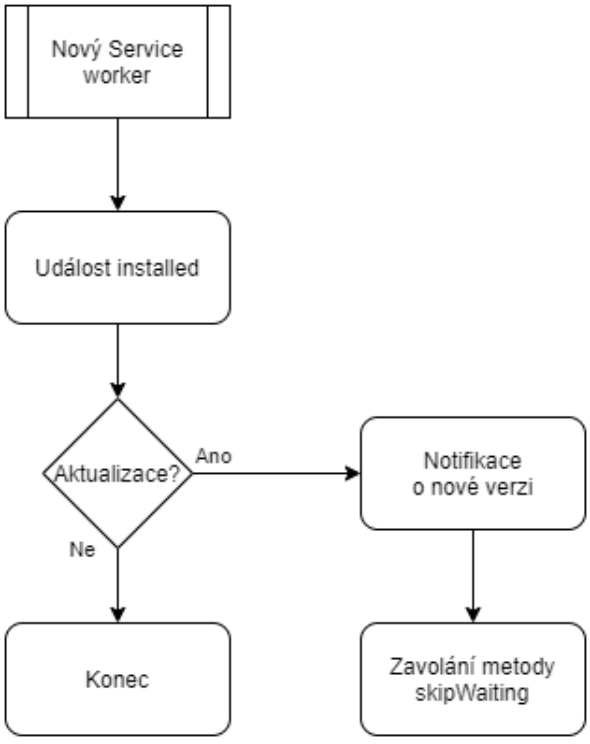
Service worker se vytváří tak, že knihovna načte svou šablonu `sw.js`, kterou během sestavení zkopíruje do cílového adresáře v závislosti na konfiguraci. Konfigurace má položku `skipWaiting`, která by měla vždy přeskočit čekání na registraci nového service workera. Tato možnost při testování nefungovala, takže jsem musel dopsat vlastní kód, který se nachází v souboru `/plugins/wb-ext.js`. Obsah tohoto souboru během sestavení zakomponuje do výsledného service workera, tím je možné zajistit vlastní funkcionalitu.

Tato knihovna zároveň zpracovává logo a manifest aplikace. Knihovně stačí poskytnou jedno logo ve formátu PNG o velikost 512 px na výšku i na šířku. Během sestavení knihovna automaticky ikonu vezme a změnší ji do všech velikostí, které potřebuje. Všechny tyto informace včetně ostatní konfigurace pak použije k tomu, aby se nakonec vygeneroval i výsledný manifest soubor.

3.2.5 Aktualizace verzí

Při použití PWA prohlížeč vnitřně registruje service worker. Aktualizací se tedy v tomto kontextu myslí to, že předchozí service worker se nahradí novým. API prohlížeče pak posyktuje jednotlivé události, na které se dá poslouchat a na jejich základě implementovat logiku pro to, aby se service worker nahradil novým. Bez této logiky by jinak service worker zůstal velmi dlouho ve stavu čekání.

V tomto případě je nejzajímavější událost `installed`, která jako data posílá i informace o tom, jestli je aktualizace nebo není. V případě, že není je to velmi jednoduché a není třeba dělat žádnou akci. V opačném případě se uživateli ukáže notifikace o tom, že je k dispozici nová verze aplikace spolu s tlačítkem na obnovení stránky, na které se nachází. V momentě, kdy na tlačíko klikne, se pošle do nového service workeru událost `message`, která obsahuje data `"skipWaiting"`. Tento service worker na tuto událost naslouchá, a pokud zjistí, že poslaná data odpovídají řetězci `"skipWaiting"`, tak zavolá metodu `self.skipWaiting()`, čímž se původní service worker nahradí novým (viz Obrázek 3.5). To má za následek to, že se invaliduje cache původního service workeru a vše by se mělo aktualizovat.

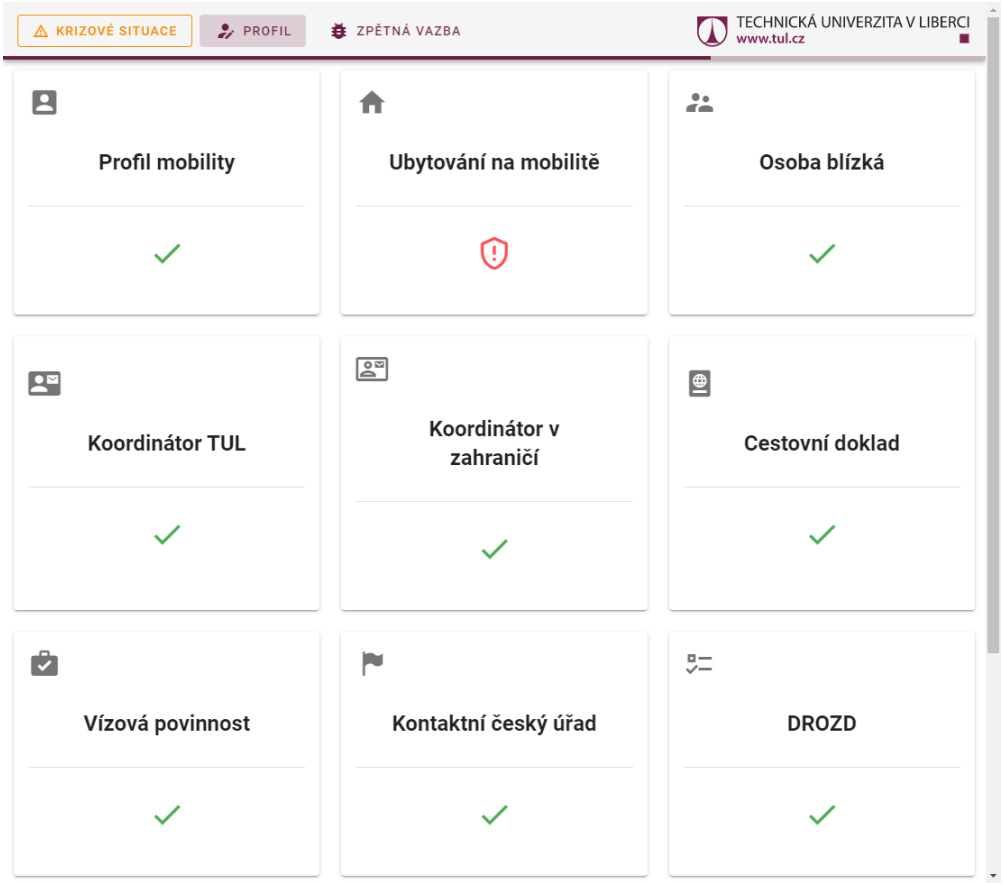


Obrázek 3.5: Logika aktualizace aplikace

3.2.6 Grafická stránka aplikace

Vzhled aplikace jsem vyřešil použitím grafické knihovny Vuetify [18], která obsahuje předpřipravené komponenty, které implementují Google Material Design. Vuetify má velmi dobrou dokumentaci s předpřipravenými situacemi pro použití jednotlivých komponent. Je možné si je nakonfigurovat online a poté jen překopírovat do vlastního projektu. Ukázka vzhledu aplikace je níže (viz Obrázek 3.6).

Knihovna je rozšiřitelná a konfigurovatelné z hlediska barev i témat. Poslední dobou jsou velmi populární tmavá témata aplikací. To lze velmi jednoduše implementovat i zde.



Obrázek 3.6: Ukázka vzhledu aplikace na desktopu

3.3 Testování

Serverovou část aplikace jsem testoval za pomoci testovací knihovny PHPUnit ve verzi 9. Tato knihovna je zakomponována do jádra Laravelu, takže bylo velmi jednoduché ji použít.

Testy aplikace jsou umístěny v adresáři **tests**, kde se ještě dělí na **Feature** a **Unit** testy. V kontextu této aplikace nebylo nutné implementovat Unit testy, protože drtivá většina práce serverové části aplikace je pouze poskytnout data části klientské. Naopak jsem ve velké míře využil Feature testy pro jednotlivé API endpointy.

Každý z testů pracuje s databází a přihlášením uživatele přes JWT. Každý API endpoint pak testuju na tyto základní příznaky:

- **200** – vše proběhlo v pořádku, zpravidla GET požadavky.
- **201** – vše proběhlo v pořádku, zpravidla POST požadavky.
- **400** – došlo k chybě v parametrech požadavku.
- **401** – došlo k chybě při ověřování platnosti tokenu.
- **405** – API endpoint byl zavolán se špatnou metodou.

Pro požadavky typu GET, které vracejí strukturovaná data se pak ještě kontroluje, zda daná struktura odpovídá specifikaci, která je uvedena v Apiary.

Každý z testů vyžaduje přístup k databázi ve stejném stavu. Aplikace poskytuje základní sadu inicializačních dat, která se při každém spuštěném testu obnovuje. Testovací databáze je typu SQLite a je umístěna v adresáři `/database/database.sqlite`. Jak testovací, tak i produkční databáze je plně konfigurovatelná v adresáři config nebo v kořenovém adresáři v souboru `phpunit.xml`.

Testy se spouští příkazem `php artisan test`, který spustí všechny testy v adresáři tests. Tento příkaz má mnoho přepínačů, kterými je možné testy zastavit na první chybě, spustit je paralelně a nebo spustit jeden konkrétní test. Před prvním spuštěním testů je důležité, aby existoval soubor `/database/database.sqlite`, jinak dojde k chybě, protože aplikace si jej nedokáže vytvořit sama.

V klientské části aplikace je možné testy provádět například skrze framework Jest, který je specificky navržen pro JavaScript a VueJS ho doporučuje ve své oficiální dokumentaci. Zde je možné využít tzv. *End-to-end* testy, které ve virtuálním prohlížeči simulují chování uživatelů tak, jak jsou definované v testech. Vzhledem k tomu, že během vývoje nebylo uživatelské rozhraní pevně definováno, tak po konzultacích s vedoucím práce tyto testy nebyly realizovány.

3.3.1 Statická analýza kódu a stylizace kódu

Za součást testování považuji i statickou analýzu kódu a stylizaci kódu, aby se jednoduché chyby daly velmi rychle podchytit a odladit. To zvyšuje i jednotnost a kvalitu kódu jako celku. Pro to, abych toho docílil jsem využil několik nástrojů. Začnu nástroji pro serverovou část aplikace a pomalu přejdu ke klientské.

Prvním nástrojem je nástroj PHPStan [19] od českých vývojářů, který dokáže staticky analyzovat kód bez jeho spuštění, čímž dokáže odhalit například podmínky, které by se například vždy vyhodnotily jako nepravda. PHPStan funguje nejlépe, když je kód dobře anotovaný a snaží se být co nejvíce typově striktní. Funguje v několika úrovních analýzy, kdy nejnižší je úroveň 1 a nejvyšší je úroveň 7. Aplikace v momentálním stavu nehlásí žádné chyby, když spustím analýzu pro úroveň 7. PHPStan má poněkud složitější zápis volání, proto je v kořenovém adresáři Makefile, který ho schová do jednoduchého příkazu `make stan`.

Další nástroj je využíván jak u serverové, tak u klietské části aplikace. Jedná se o Editorconfig [20], který standardizuje věci typu maximální délka řádku, velikost odsazení, typ odsazení (mezery, tabulátory), ukončení řádku (LF, CRLF) a mnoho dalších. Ve vývojovém prostředí jako je PHPStorm jde jednoduše nainstalovat doplněk, který bude tyto pravidla aktivně vynucovat a zobrazovat vývojáři jako chyby.

Posledním nástrojem je ESLint [21], který dodržuje code style pro JavaScript a VueJS. Tento nástroj se doplňuje s nástrojem Husky, který se napojuje na události gitu (commit, push, pull, atd.) a umí spustit nějaký z definovaných příkazů. V tomto projektu to je udělané tak, že když chce vývojář udělat commit, Husky spustí ESLint. Pokud kód není konzistentní a ESLint selže, tak Husky nedovolí commit udělat. Díky tomu je možné zaručit určitou kvalitu kódu.

4 Uživatelská podpora

Jedním z bodů zadání bylo do aplikace zahrnout systém, který by od uživatelů získal zpětnou vazbu. Tento systém byl navržen tak, aby byl co nejjednodušší jak pro uživatele, tak i pro obsluhu, která zpětnou vazbu bude vyhodnocovat.

4.1 Systém pro práci se zpětnou vazbou

Klientská část aplikace implementuje jednoduché rozhraní, které se skládá z textového pole a tlačítka. Textové pole je omezeno na 500 znaků a po jeho vyplnění se stiskem tlačítka zpětná vazbá skrze API odešle na server. V případě, že je přihlášený uživatel s vyšším oprávněním (např. některý ze zaměstnanců školy), tak místo textového pole a tlačítka je zde tabulka. Ta obsahuje přehled nevyřešené zpětné vazby, kterou studenti v rámci aplikace odeslali. Tabulka umožňuje stránkování a vyhledávání podle e-mailu studenta. Každý ze záznamů tabulky také obsahuje tlačítko, kterým je možné zpětnou vazbu označit jako vyřešenou. Po zmáčknutí tlačítka se objeví hláška s dalším akčním tlačítkem, které tuto akci zvládne vrátit zpět. To je pro případ, že by uživatel zmáčkl tačítko omylem.

Serverová část aplikace pak zpětnou vazbu persistentně ukládá do databáze a posyktuje ji klientské části aplikace. Implementuje tyto čtyři endpointy pro získávání a manipulaci se zpětnou vazbou:

- `[GET] /api/v1/feedback` - seznam všech nevyřešených záznamů zpětné vazby, které jsou chronologicky seřazeny od nejnovější.
- `[POST] /api/v1/feedback` - uložení nové zpětné vazby.
- `[PUT] /api/v1/feedback/{feedbackId}/close` - uzavření zpětné vazby.
- `[PUT] /api/v1/feedback/{feedbackId}/open` - otevření zpětné vazby.

4.2 Logování klientské aplikace

Systém zpětné vazby je doplněn o logování klientské aplikace. Pokud je aplikace připojena k internetu a chyba nenastala v kódu, který obsluhuje HTTP požadavky, tak aplikace pošle logovací záznam na server. Tyto logy se dají skrze otisk spárovat s uživatelem a mohou být použity pro diagnostiku potenciálních problémů. Pokud je aplikace offline, tak si do lokálního uložiště uloží posledních 20 logů, které se při připojení k internetu pošlou na server. Jeden logovací záznam se skládá z:

- `fingerprint` - otisk zařízení, aby bylo možné logy přiřadit ke konkrétnímu uživateli.
- `level` - úroveň akutnosti logu, může nabývat hodnot info, warn a error, kde info je nejmenší závažnost a error nejvyšší.
- `event` - doplňující informace o tom, kdy k logu došlo (například přihlášení).
- `device` - informace o zařízení uložené ve formátu JSON.
- `payload` - vlastní informace logu ve formátu JSON.
- `created` - datum, kdy byl záznam vytvořen.

5 Nasazení aplikace

Aplikace je hostována ze školního virtuálního serveru z domény https://trip.tul.cz/. Na tomto serveru je nainstalováno Ubuntu 20.04 společně s MySQL 5.7, PHP 7.4 a webserverem apache ve verzi 2.4.

Obě části aplikace jsou poskytované ze stejné domény, ale API je schováno za URL https://trip.tul.cz/api. To umožňuje apache direktiva `Alias`, která na straně serveru tento provoz směřuje do adresáře `/var/www/server`. Klientská část aplikace je pak poskytována z adresáře `/var/www/client`. Oba tyto adresáře je možné velmi jednoduše aktualizovat skrze technologii git, která je použita jako verzovací nástroj projektu.

Za pomocí direktivy `<LocationMatch>` je Shibboleth nakonfigurovaný tak, že vyžaduje přihlášení na `/` a `/api/authentication`. Díky tomu je možné na API číst hlavičky `Eppn` a `Affiliation`, které Shibboleth poskytuje. Cestu `/api` pak ignoruje a ta je zabezpečena JWT tokenem v rámci samotné aplikace. Důležitá část konfigurace pak vypadá následovně:

```
<VirtualHost *:443>
    <Directory /var/www/client>
      Options All
      AllowOverride All
      order allow,deny
      allow from All
    </Directory>

    Alias /api /var/www/server/public
    <Directory /var/www/server/public>
      Options All
      AllowOverride All
      order allow,deny
      allow from All
```

```
    </Directory>
    ...
</VirtualHost>
...
<Location />
  AuthType shibboleth
  ShibRequestSetting requireSession 1
  ShibUseHeaders On
  Require shib-session
</Location>

<LocationMatch /api/(?!authentication)>
  Satisfy Any
  allow from All
</LocationMatch>

<LocationMatch /_nuxt/>
  Satisfy Any
  allow from All
</LocationMatch>
```

5.1 Přepisování URL apachem

Při konfiguraci apache jsem narazil na problém s direktivou `<LocationMatch>`, protože jsem se snažil zajistit to, aby některé URL zabezpečoval Shibboleth a některé samotná aplikace. Direktivy Apache se provádějí v následujícím pořadí:

1. `<Directory>` (obsahuje zpracování souborů `.htaccess`)
2. `<DirectoryMatch>`
3. `<Files>` a `<FilesMatch>`
4. `<Location>` a `<LocationMatch>`
5. `<If>`

Problém nastává v tom, že direktiva `<Directory>` najde soubor `.htaccess` v aplikaci. Ten přepisuje URL na `index.php` a globální proměnná `$_SERVER` pak

poskytuje informace o tom, kterou URL má zpracovat. Takto to má nastavené framework Laravel a nebylo možné do toho zasáhnou bez významných zásahů nebo dokonce změny technologie. To znamená, že například `/api/steps` se přepíše na `/api/index.php`. To má pak za následek, že kdybych chtěl direktivou `<LocationMatch>` odchytávat cestu `/api/steps`, tak to nebude fungovat, protože URL se mezitím změnila. Toto se stane pouze v případě, pokud požadovaná složka nebo soubor neexistuje. Vyřešil jsem to tedy tak, že jsem do public složky API umístil další index.php, který se nachází v cestě `public/api/authentication/index.php`. Logika v tomto souboru je velmi jednoduchá. Nastartuje se Laravel a místo toho, aby zpracoval požadavek a sám se rozhodl jak odpovědět, tak se mu přesně řekne, aby odpověděl autentifikací uživatele.

5.2 Verzování

Celá aplikace je verzována technologií git a je hostována na univerzitním gitlabu. Aplikace je rozdělena do 3 repozitářů – **trip-server**, **trip-client** a **trip-client-build**. Je to vymyšlené tak, aby se na server, kde aplikace hostovaná, nahrávala až finální aplikace a nemusela se tam sestavovat. Proto je zde 3. repozitář trip-client-build. Proces u klientské části aplikace je takový, že po vývoji nové verze se aplikace sestaví na počítači vývojáře. Pak se následujícím příkazem vyčistí repozitář pro sestavení:

```
cd $client_build_repository
git rm -rf .
git clean -fxd
```

Potom se provede commit, který jsem se snažil držet ve formátu `Build [<trip-client commit hash>]`. Následně se stačí skrze SSH přihlásit na server a skrze git aplikaci aktualizovat.

6 Závěr

Hlavním cílem této práce bylo navrhnout a implementovat aplikaci pro podporu studentské mobility na Technické univerzitě v Liberci. Tato aplikace vycházela z předchozího prototypu, který byl navržený během diplomového projektu jako ověření konceptu, že je něco takového možné realizovat. Během vývoje jsem velmi blízce spolupracoval se zahraničním oddělením TUL, aby aplikace obsahovala vše potřebné a splňovala jejich požadavky.

Aplikace jako taková pak studentům slouží ve dvou fázích výjezdu za hranice. V první, kdy si před výjezdem aplikaci otevřou a projdou ideálně všechny kroky formuláře, který je informuje o všem, co pro výjezd za hranice potřebují. Druhá fáze je pak samotný výjezd, kdy aplikaci mají instalovanou na svém mobilním zařízení a informace, které předtím vyplnili mají nyní k dispozici. A to zejména pro případ, že by se dostali do krizové situace.

Práce se zaměřila na technologii progresivních webových aplikací (PWA), která rozšiřuje webové aplikace a dělá z nich rovnocenné konkurenty nativních aplikací. Díky tomu je možné vytvořit aplikace, které fungují na všech platformách s podporou webového prohlížeče a mají tedy i čistě pragmatický benefit v tom, že není nutné implementovat aplikaci zvlášť pro každý operační systém. Aplikace také využívá moderní frameworky a nástroje pro dodržení kvality kódu tak, aby byla zajištěna kontinuita vývoje. Ukazuje se zde i síla této technologie a proč je tolik populární. Hlavní výhodou je nízká náročnost na vývoj a jednoduchá distribuce ke koncovým uživatelům formou odkazu na webovou stránku. Každopádně je vždy dobré mít na paměti, že pro velmi specifické aplikace bude zpravidla lepší použít nativní aplikaci pro danou platformu.

Během vývoje byla vyslovena i myšlenka, že by existovalo více aplikací, které by zajišťovaly funkcionalitu této aplikace. Tedy, že jedna aplikace by byla webový formulář před výjezdem a druhá by byla mobilní aplikace, která by během výjezdu data zobrazovala. Po konzultacích s vedoucím této diplomové práce a zahraničním

oddělením TUL jsme ale došli k závěru, že více aplikací nemá mnoho benefitů. V případě změn by bylo nutné zasahovat do mnoha míst kódu v různých aplikacích. Zároveň se eliminuje potenciální nejasnosti u studentů v tom, co která aplikace vlastně dělá, a co která poskytuje.

Při zadávání této práce bylo v plánu aplikaci na začátku roku 2021 dát mezi uživatele, a tím aplikaci začít testovat, a uvést do provozu. Bohužel vzhledem ke globální situaci spojenou s pandemií to nebylo možné, ale v době psaní této práce (duben 2021) se aplikace již dostala mezi 2 studenty, kteří aplikaci začali testovat.

6.1 Dosažené výsledky

V rámci této diplomové práce vznikla aplikace s názvem TULtrip, která byla realizována jako progresivní webová aplikace. Aplikace je určena pro studenty Technické univerzity v Liberci, kteří vyjíždějí za hranice přes program Erasmus+.

Vzhled aplikace je implementován v souladu s specifikací Material Desing od společnosti Google a barevně vychází z logo manuálu univerzity. Součástí výsledné aplikace jsou 3 repozitáře hostované na gitlabu TUL a specifikace API, která byla vytvořena ve službě Apiary jako tzv. *API Blueprint*.

Aplikace obsahuje jednoduchý systém pro práci se zpětnou vazbou, který studentům umožňuje odeslat názor nebo připomínku. Naopak uživatelům s privilegovaným přístupem umožňuje tuto zpětnou vazbu revidovat a dělat na základě toho další rozhodnutí.

Architekturou aplikace odpovídá architektuře klient-server. Server je v tomto případě REST API, které prakticky slouží jako persistentní uložiště pro klientskou stranu aplikace. Komunikace je zabezpečena JWT tokenem, který je vystaven na základě toho, zda se podaří ověřit uživatelovu identitu vůči TUL. V dalších případech se pak využívá unikátního otisku zařízení, který se sestaví na základě parametrů prohlížeče a systému zařízení. Výsledná aplikace byla posléze nasazená na virtuální univerzitní server, kde aplikaci poskytuje webový server apache2. Aplikace je dostupná na adrese https://trip.tul.cz/.

6.2 Další možnosti vývoje

V době psaní této diplomové práce aplikace ještě prochází grafickými návrhy, které řeší zahraniční oddělení TUL. Do budoucna se tedy ještě počítá s jejich implementací a nasazením.

Je také dobré zmínit, že aplikace byla testována na jejím ústředním bodě, za co považuji API rozhraní mezi serverem a klientem. To znemaná, že je otevřená možnost dodělat ještě integrační a funční testy na klientské straně aplikace. Tato část není tak rozsáhlá, a proto zde testy neměly takovou prioritu.

Během vývoje mi bylo také sděleno, že o aplikaci slyšeli i jiné univerzity a nápad se jim velmi líbil. To by mohlo určit směr dalšího vývoje aplikace tak, aby šla nasadit například i na jiné univerzitě kromě TUL.

Literatura

[1] DROZD | Projekt Dobrovolné registrace občanů České republiky při cestách do zahraničí. DROZD | Projekt Dobrovolné registrace občanů České republiky při cestách do zahraničí [online]. Copyright © Ministerstvo zahraničních věcí České republiky, 2021 [cit. 26.04.2021]. Dostupné z: https://drozd.mzv.cz/

[2] Getting Started with Progressive Web Apps | Google Developers. Google Developers [online].[cit. 26.04.2021]. Dostupné z: https://developers.google.com/web/updates/2015/12/getting-started-pwa

[3] Vue.js Guide [online]. [cit. 26.04.2021]. Dostupné z: https://vuejs.org/v2/guide/

[4] React – A JavaScript library for building user interfaces. React – A JavaScript library for building user interfaces [online]. Copyright © 2020 Facebook Inc. [cit. 26.04.2021]. Dostupné z: https://reactjs.org/

[5] Service Workers: an Introduction | Web Fundamentals. Google Developers [online]. [cit. 26.04.2021]. Dostupné z: https://developers.google.com/web/fundamentals/primers/service-workers

[6] HTML Standard. HTML Standard [online]. [cit. 26.04.2021]. Dostupné z: https://html.spec.whatwg.org/multipage/workers.html#dom-worker-postmessage

[7] Using the application cache - HTML: HyperText Markup Language | MDN. [online]. Copyright © 2005 [cit. 11.05.2021]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache

[8] Workbox Strategies | Google Developers. Google Developers [online]. [cit. 26.04.2021]. Dostupné z: https://developers.google.com/web/tools/workbox/modules/workbox-strategies

[9] W3C on GitHub [online]. [cit. 26.04.2021]. Dostupné z: https://w3c.github.io/manifest/

[10] Progressive Web Apps in 2020. It's been more than 12 years since… | by Maximiliano Firtman (firt.dev) | Medium. Medium – Where good ideas find you. [online]. [cit. 26.04.2021]. Dostupné z: https://medium.com/@firt/progressive-web-apps-in-2020-c15018c9931c

[11] Stack Overflow Developer Survey 2019. Object moved [online]. [cit. 26.04.2021]. Dostupné z: https://insights.stackoverflow.com/survey/2019

[12] What Web Can Do Today. What Web Can Do Today [online]. [cit. 26.04.2021]. Dostupné z: https://whatwebcando.today/

[13] Representational state transfer - Wikipedia. [online]. [cit. 22.04.2021]. Dostupné z: https://en.wikipedia.org/wiki/Representational_state_transfer

[14] GitHub - tymondesigns/jwt-auth: JSON Web Token Authentication for Laravel & Lumen. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 22.04.2021]. Dostupné z: https://github.com/tymondesigns/jwt-auth

[15] GitHub - axios/axios: Promise based HTTP client for the browser and node.js. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 12.05.2021]. Dostupné z: https://github.com/axios/axios

[16] GitHub - fingerprintjs/fingerprintjs: Browser fingerprinting library with the highest accuracy and stability.. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 12.05.2021]. Dostupné z: https://github.com/fingerprintjs/fingerprintjs

[17] GitHub - jackspirou/clientjs: Device information and digital fingerprinting written in pure JavaScript.. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 12.05.2021]. Dostupné z: https://github.com/jackspirou/clientjs

[18] Vuetify — A Material Design Framework for Vue.js. Vuetify — A Material Design Framework for Vue.js [online]. Copyright © 2016 [cit. 12.05.2021]. Dostupné z: https://vuetifyjs.com/en/

[19] Playground | PHPStan. Playground | PHPStan [online]. Copyright © 2016 [cit. 25.04.2021]. Dostupné z: https://phpstan.org/

[20] EditorConfig. EditorConfig [online]. [cit. 26.04.2021]. Dostupné z: https://editorconfig.org/

[21] ESLint - Pluggable JavaScript linter. ESLint - Pluggable JavaScript linter [online]. [cit. 26.04.2021]. Dostupné z: https://eslint.org/

Příloha I - zdrojové kódy

- Klientská část aplikace
  - tultrip_client.zip
- Serverová část aplikace
  - tultrip_server.zip
  - api_blueprint.apib