FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Webová aplikace pro půjčování zařízení a hardware
Student:	Ilona Andriyashyn
Vedoucí:	Ing. Jan Blizničenko
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je vytvořit webovou aplikaci umožňující studentům FIT ČVUT snadné vzájemné vypůjčení různých zajímavých zařízení (např. Orange Pi). Studenti se budou přihlašovat pomocí svého školního účtu (auth.fit.cvut.cz).

Aplikace by měla obsahovat dashboard s přehledem zapůjčených zařízení, nastavení profilu, zobrazení a možnost rezervace zařízení vč. kalendáře s výběrem termínu. Uživatelé rovněž budou moct vložit své nové zařízení, nabídnout ho k zapůjčení a rezervaci a zápůjčky spravovat.

Aplikace by měla být plně responzivní, dostupná v češtině a angličtině a připravena pro další možné lokalizace.

Instrukce:
- Proveďte rešerši podobných webových i mobilních aplikací.
- Proveďte rešerši relevantních technologií, nástrojů, frameworků a knihoven.
- Vytvořte návrh aplikace a jejích součástí.
- Proveďte implementaci aplikace.
- Otestujte a zdokumentujte své řešení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 26. ledna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webová aplikace pro půjčování zařízení a hardware

Ilona Andriyashyn

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jan Blizničenko

13. května 2021

Poděkování

Děkuji vedoucímu práce Ing. Janu Blizničenkovi za rady a veškerou pomoc při tvorbě práce. Děkuji Dominiku Dosoudilovi za technické rady, uživatelské testování a podporu.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Andriyashyn, Ilona. *Webová aplikace pro půjčování zařízení a hardware*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato bakalářská práce se zabývá tvorbou webové aplikace pro Fakultu informačních technologií ČVUT v Praze, kde si studenti či zaměstnanci mohou vzájemně půjčovat zařízení a hardware. V práci je provedena rešerše podobných aplikací. Dále jsou popsány vybrané technologie a také vysvětlen způsob implementace. Aplikace je rozdělená na serverovou a klientskou část, které spolu komunikují pomocí REST API. Vytvořená webová aplikace je responzivní a dostupná v češtině a angličtině. Přihlásit se do ní může kdokoliv, kdo vlastní FIT účet. V závěru práce je k nalezení popis možných rozšíření, kterými lze na tuto práci navázat.

Klíčová slova webová aplikace, rezervace zařízení, FIT účet, responzivní web design, Node.js, TypeScript, React

Abstract

This bachelor thesis focuses on creating a web application for the Faculty of Information Technology, CTU in Prague, which allows students or employees to borrow devices and hardware from each other. Research of similar applications is done at the start of this thesis, then chosen technologies and implementation of application are described. Server-side and client-side applications communicate together via a REST API. A created web application is responsive and is available in both English and Czech. Anyone who has a FIT account can log in to the application. At the end of the thesis, a description of possible features that can extend the application can be found.

Keywords web application, devices reservation, FIT account, responsive web design, Node.js, TypeScript, React

Obsah

Seznam obrázků

Seznam ukázek kódů

Úvod

Studenti a zaměstnanci FIT ČVUT v Praze tráví velkou část svého dne ve škole. Každý den se potkávají s lidmi ze stejného zaměření jako oni sami. Pravděpodobně mají stejný zájem o novinky na trhu technologií. Mnozí z nich vlastní různá zařízení určená pro zábavu (brýle pro virtuální realitu, mikropočítač, programovatelného robota...), pro zjednodušení průběhu práce (chytrý prsten s NFC čipem, čtečku otisků prstů, čtečku čipových karet...), nebo dokonce zařízení zdravotní (EEG přístroj podporující metodu biofeedback...). Rádi by je svým spolužákům půjčili, ale chybí jim vhodná platforma, kde by udržovali přehled o zařízeních a rezervacích. Proto vzniká aplikace Lend And Learn, určená pro studenty či zaměstnance fakulty.

Ovládání aplikace si klade za cíl přehlednost a jednoduchost. Uživatel nahraje do systému zařízení, která nabídne k vypůjčení. Student či zaměstnanec, který má zájem si nějaké zařízení vyzkoušet, ale není si jistý, zda si ho chce kupovat, může využít této aplikace. Podívá se, jestli je konkrétní zařízení v nabídce. Pokud ano, zarezervuje si ho na určitý volný termín. Dále už je na majiteli zařízení, aby rezervaci schválil a studenta či zaměstnance kontaktoval.

Aplikace Lend And Learn bude využívat přihlášení pomocí školního FIT účtu. Díky tomu bude Lend And Learn bezpečnější, což může být pro uživatele motivující k jejímu používání. Použití školního účtu výrazně omezí riziko například odcizení zařízení, jelikož nebude možné, aby se registroval člověk, který není studentem či zaměstnancem FIT.

Tato bakalářská práce se zaměřuje na návrh a implementaci aplikace Lend And Learn. V první části práce jsou popsána již existující řešení nebo podobné systémy, které se pro tento účel dají použít. Dále je provedena rešerše vhodných technologií a je popsán návrh databázového schématu. Nakonec je popsána samotná implementace a testování aplikace.

Kapitola 1

Cíl práce

Hlavním cílem této bakalářské práce je vytvořit přehlednou webovou aplikaci, která studentům či zaměstnancům Fakulty informačních technologií umožní vzájemné vypůjčení hardware, počítačových zařízení a příslušenství.

Cílem teoretické části práce je provést rešerši již existujících řešení. Zaměřit se na analýzu těchto řešení z pohledu běžného uživatele a jejich srovnání mezi sebou. Cílem je také zjistit jejich výhody a nevýhody. Dále prozkoumat vhodné technologie a nástroje, které zajistí správnou funkčnost aplikace a také snadné propojení backendu a frontendu.

Cílem praktické části je návrh a implementace aplikace pomocí vybraných technologií. Aplikace bude responzivní. Bude také dostupná v angličtině a češtině. Zároveň bude připravená pro další případné lokalizace. Dalším cílem praktické části je poskytnout studentům a zaměstnancům FIT snadné spravování svých rezervací, přidání a přehled zařízení. Jako poslední stanovený cíl této části je otestování a detailní popis řešení.

Aplikace je zaměřená na podporu sebevzdělání a výsledek této bakalářské práce bude přínosný hlavně studentům a zaměstnancům Fakulty informačních technologií.

Kapitola 2

Analýza

V této kapitole se provádí analýza aplikace. Nejprve jsou specifikovány požadavky na aplikaci. Dále je provedena rešerše existující konkurence, ze které plyne jednoduché shrnutí. Nakonec jsou popsány typy uživatelů aplikace Lend And Learn.

2.1 Požadavky na aplikaci

V této sekci jsou specifikovány požadavky na aplikaci Lend And Learn. Jejich přesné stanovení je velmi důležité pro správný vývoj projektu. Požadavky vznikly na základě konzultací s vedoucím práce a jsou rozděleny do dvou kategorií, a to funkční a nefunkční. Splnění těchto požadavků je ověřeno pomocí akceptačních testů.

2.1.1 Funkční požadavky

Funkční požadavky říkají, které funkcionality bude aplikace implementovat. Pro snadné odkazování na určitý funkční požadavek se používá zkrácené označení F[číslo].

- **Přihlášení (F1)**
  Přihlášení do aplikace bude provedeno přes formulář, který poskytne server Fakulty informačních technologií ČVUT. Uživatel se přihlásí pomocí svého unikátního uživatelského jména, které mu bylo přiděleno na začátku studia fakultou, a příslušného hesla. Jedná se o tzv. FIT účet, který se využívá k přihlášení se do řady dalších školních systémů.

- **Prohlížení všech zařízení (F2)**
  Uživateli bude umožněno prohlížení všech existujících zařízení, nehledě na jejich počet.

- **Prohlížení vlastních zařízení (F3)**
  Uživateli bude umožněno prohlížení pouze svých vlastních přidaných zařízení, nehledě na jejich počet.

- **Přidání zařízení (F4)**
  Uživatel bude moci přidat zařízení. Název zařízení bude evidovaný jako povinný údaj. Dále bude evidovaný nepovinný údaj o popisu zařízení.

- **Smazání zařízení (F5)**
  Uživateli bude umožněno smazání vlastního zařízení.

- **Prohlížení konkrétního zařízení (F6)**
  Uživatel bude moci zobrazit informaci o konkrétním zařízení.

- **Editace konkrétního zařízení (F7)**
  Uživatel bude moci editovat data o svém konkrétním zařízení.

- **Žádost o rezervaci zařízení (F8)**
  Uživatel bude moci rezervovat pouze cizí zařízení. Rezervace bude provedena pomocí kalendáře, kde si bude možné vybrat volný termín. Termíny, které jsou již obsazené, nebudou uživateli přístupné. Bude zapotřebí zadat datum počátku a datum ukončení rezervace.

- **Schválení žádosti o rezervaci zařízení (F9)**
  Jednotlivou žádost o rezervaci bude moci schválit pouze majitel zařízení, ke kterému se žádost vztahuje.

- **Odmítnutí žádosti o rezervaci zařízení (F10)**
  Neschválenou (čekající) rezervaci bude možné odmítnout. Učiní to buď majitel zařízení, nebo uživatel, který žádost o rezervaci vytvořil.

- **Prohlížení aktuálních příchozích rezervací (F11)**
  Uživateli bude umožněno prohlížení aktuálně probíhajících příchozích rezervací. Jedná se o rezervace vztahující se na zařízení, která on vlastní.

- **Prohlížení aktuálních odchozích rezervací (F12)**
  Uživateli bude umožněno prohlížení aktuálně probíhajících odchozích rezervací. Jedná se o ty, které uživatel vytvořil a vztahují se na půjčené od jiného uživatele zařízení.

- **Ukončení příchozí rezervace zařízení (F13)**
  Uživatel bude moci ukončit aktuálně probíhající příchozí rezervaci, která se vztahuje na některé z jeho zařízení.

- **Prohlížení profilu uživatele (F14)**
  Aplikace umožní prohlížení profilu jakéhokoliv uživatele.

- **Poslání e-mailu (F15)**
  Aplikace umožní poslat e-mail každému uživateli, kromě sama sobě. Tato akce bude provedena pomocí e-mailového klienta.
- **Aktualizace osobních údajů (F16)**
  Přihlášený uživatel bude moci aktualizovat své osobní údaje. Toto bude provedeno pomocí získání dat ze školní databáze namísto databáze aplikace Lend And Learn.
- **Přepínání mezi jazyky (F17)**
  Uživatel bude moci aplikaci zobrazit v anglickém, nebo českém překladu.
- **Prohlížení dat o vlastních zařízeních (F18)**
  Uživatel bude moci zobrazit počet všech zařízení, která vlastní. Dále počet aktuálně vypůjčených zařízení a také počet zařízení, která jsou aktuálně k dispozici.
- **Odhlášení (F19)**
  Uživatel bude moci se z aplikace odhlásit.

Všechny požadavky, kromě F1, se vztahují pouze na přihlášené uživatele.

2.1.2 Nefunkční požadavky

Tato sekce se zabývá nefunkčními požadavky. Wiegers ve své knize píše, že nefunkční požadavek je vlastnost nebo omezení systému, které má dopad například na architekturu aplikace, jelikož tuto vlastnost nebo omezení musí implementovat, případně respektovat. [3] Nefunkční požadavek je označený pomocí zkratky N[číslo].

- **Webová aplikace (N1)**
  Aplikace bude navržena jako webová aplikace.
- **Responzivita (N2)**
  Aplikace bude určená primárně pro zobrazení v internetových prohlížečích jako webová aplikace, nicméně bude plně responzivní, čímž umožní snadný pohyb v aplikaci i přes mobilní zařízení.
- **Lokalizace (N3)**
  Aplikace bude připravená na veškeré možné budoucí překlady.
- **Bezpečnost (N4)**
  Do aplikace se bude možné bezpečně přihlásit pomocí účtu, který spravuje fakultní server. Aplikace Lend And Learn nebude ukládat heslo uživatele do vlastní databáze.
- **Snadné nasazení (N5)**
  Aplikace bude snadno nasaditelná ve virtuálním prostředí pomocí nástroje Docker.

2.2 Analýza konkurence

Aplikace Lend And Learn je určená pouze pro studenty nebo zaměstnance FIT. Podobná aplikace, která by se zaměřila na půjčování technického zboží, v tuto chvíli na fakultě neexistuje. V této části práce je rešerše zaměřená na aplikace pro širokou veřejnost.

2.2.1 Půjčím.to

Půjčím.to je česká webová platforma, kterou je možné využít pro vzájemné půjčování různého druhu zboží mezi sebou. Svou cílovou skupinu rozděluje na běžné uživatele a profesionální půjčovny. Vypůjčení ve většině případů probíhá na základě finanční odměny, ale je možné si věc půjčit i zdarma. [1]

Na svých webových stránkách [1] autoři uvádějí několik motivačních důvodů, proč jejich platformu používat. Některé z nich jsou:

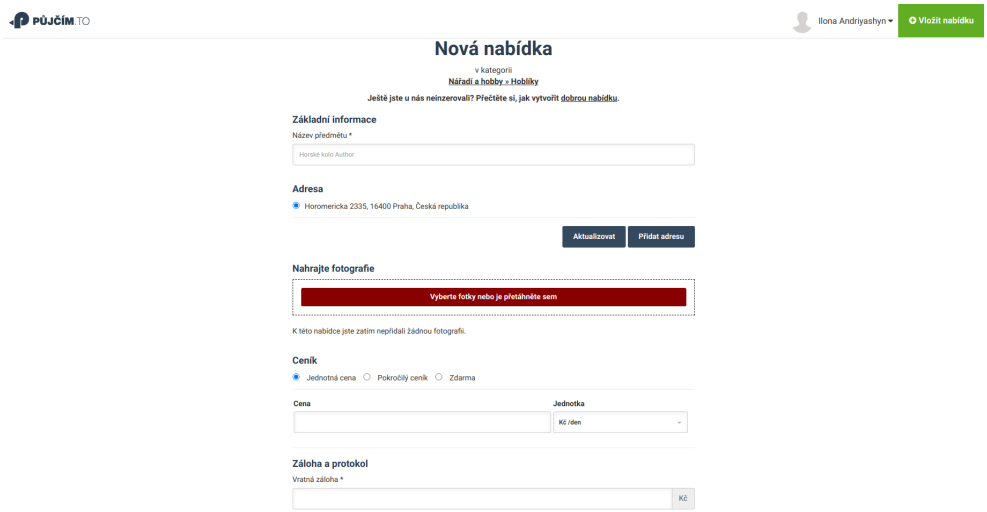
- výhodné získání zboží určeného k nepravidelnému využití,
- možnost výdělku,
- možnost srovnání nabídek.

Pro využití aplikace je zapotřebí se registrovat. Je možné se registrovat pomocí sociální sítě Facebook. V tomto případě některé údaje, jako jméno nebo e-mail, se předvyplní. Účet se také propojí s Facebook profilem. Je potřeba uvést kontaktní adresu a telefonní číslo.

Na svém profilu uživatel má k vidění svá hodnocení. Uživatel může hodnotit pouze toho, komu zaslal nějakou poptávku. Je mu k dispozici formulář, přes který může posílat zprávy. Nicméně tento formulář působí velmi nepřehledně. E-mail a telefonní číslo jsou již předvyplněnými údaji od přihlášeného uživatele. E-mail navíc nejde změnit. Není tak jasné, jaký typ zprávy se posílá a kdo je příjemce.

Proces vytvoření nabídky je jednoduchý. Při každé nabídce se zvolí kategorie a podkategorie zboží. Cena se uvádí v Kč a je možné ji specifikovat za hodinu, den, týden, měsíc, rok. Dále je možné určit minimální dobu, po kterou je nutné si zboží půjčit. Nabídka je zabezpečená formou zálohy. Výši této zálohy určí nabízející. Také je možné přidat seznam závad, které zboží obsahuje. Nechybí ani pokročilá funkce, pomocí které se nabídka nezveřejní okamžitě. Část procesu přidání nabídky je zobrazena na obrázku 2.1.

Vyhledání nabídky je možné upřesnit pomocí filtrů, například podle lokality, kategorie, ceny, doby zapůjčení. Při výběru termínu rezervace konkrétního zboží se okamžitě spočítá cena, kterou uživatel odvede, což je velmi nápomocné. Dále se zobrazuje seznam podobného zboží. Velkou výhodou je také možnost oslovit majitele zboží s dotazem přímo přes aplikaci. Registrované půjčovny mohou při nabídce zboží přidat odkaz na svou platformu s pří-



Obrázek 2.1: Proces přidání nabídky [1]

padným rezervačním systémem. Potom rezervace nebude probíhat na webové stránce projektu Půjčím.to.

Aplikace je díky své bohaté funkcionalitě velmi vhodná pro využití ve svém oboru. Mezi nevýhody patří výše zmíněný nepřehledný formulář a zobrazení reklam. Je responzivní, nicméně některé funkcionality, jako například zobrazení podobných nabídek, nejsou v malém rozlišení dostupné.

2.2.2 Reservio

Rezervační systém Reservio je využívaný řadou podniků a služeb. Je to velice obsáhlý a pokročilý systém určený pro firmy. Nenabízí sdílení věcí mezi lidmi, ale spíše rezervaci termínu u nějakého konkrétního podniku, například holičství. Kromě webové aplikace je také nově dostupný i ve formě mobilní aplikace pro platformy Android a iOS. [4]

Při registraci se uvádí název firmy. Dále se vybírá obor podnikání a následně již stačí přidat poskytovanou službu. Jde nastavit délka trvání a také, kdo službu poskytuje. Díky tomu klient při rezervaci termínu například do nějakého holičství si vybere konkrétního holiče. Při vytvoření rezervace si uživatel zvolí službu, den a konkrétní čas. Má k dispozici recenze podniku.

Aplikace nabízí velmi pokročilou správu rezervací. Rezervace můžou být skupinové i individuální. Rezervační kalendář je možné synchronizovat s jinými aplikacemi, jako Outlook nebo Google. [5] Aplikace je responzivní a je dostupná ve 12 jazycích. [4] Nicméně oproti výše zmíněné platformě Půjčím.to má jiné zaměření, proto se obě aplikace nedají dobře srovnávat. Totéž platí o Reservio a Lend And Learn.

2.2.3 Facebook

Aplikace Facebook je aktuálně nejvíce využívaná platforma pro vyřešení sdílení věcí mezi členy komunity FIT. Studenti či zaměstnanci se můžou přidat do příslušné skupiny na této platformě. V takových skupinách se uživatelé baví o různých tématech včetně nabídky nebo poptávky technických zařízení. Nicméně takové příspěvky se rychle ztrácejí mezi těmi, které přibývají. Následně je možné je vyhledat pomocí tagů. Je to velmi nepřehledné. Je možné vytvořit skupinu, která bude určená pouze pro tento problém, avšak neexistuje možnost ověření, že člověk na fakultě opravdu studuje nebo pracuje. Na druhou stranu výhoda Facebooku spočívá ve velmi snadném kontaktování uživatele.

2.2.4 Discord

Další, velmi populární mezi studenty a zaměstnanci FIT, je aplikace Discord. Půjčování věcí přes tuto aplikaci je bezpečnější než přes Facebook, protože přihlášení probíhá přes fakultní server, tudíž stejně jako Lend And Learn. Díky tomu si uživatelé mohou být jistí, že opravdu komunikují s člověkem z fakulty. Nicméně přihlášený uživatel si může zvolit přezdívku, kterou ostatní uvidí. Při komunikaci to potom často vede k poměrně nepříjemnému pocitu. Na druhou stranu je dobře zachována anonymita člověka.

V této aplikaci existují tématicky zaměřené kanály, jako například bazar, ubytování a koleje, poradna. Tyto kanály fungují podobně jako skupiny v aplikaci Facebook. Příspěvek s nabídkou nebo poptávkou je možné přidat do kanálu. Výhoda aplikace Discord je opět snadné kontaktování člověka. Nicméně chybí přehled a správa rezervací.

2.2.5 Závěr

Požadavkům na Lend And Learn nejvíce odpovídá aplikace Půjčím.to. Má podobné zaměření a nabízí kvalitní a snadné řešení. Nenabízí ale přihlášení pouze pro studenty či zaměstnance FIT. Reservio je spíše zprostředkovatel služeb od firmy pro klienta. Facebook je velice široce zaměřená sociální síť. Opět zde chybí možnost ověření identity studenta či zaměstnance. Přehled a správa zápůjček by v této aplikaci byla složitá. Aplikace Discord jako jediná řeší problém využití pouze studenty či zaměstnanci FIT. Nicméně nabízí možnost zvolit si přezdívku, která se zobrazí místo jména. Jednoduchá správa zápůjček či zařízení v této aplikaci není.

Kladem Lend And Learn je absence potřeby registrace. Využívá totiž fakultního účtu k přihlášení, což je pro cílovou skupinu této aplikace velmi pohodlné. Další výhodou je přehled a správa rezervací na jednom místě. Není potřeba si jednotlivé žádosti o rezervaci zařízení zpětně dohledávat mezi zprávami, což může být velmi časově náročné. Learn And Learn oproti Discordu

neobsahuje možnost zvolení si přezdívky. Díky tomu uživatel přesně ví, s kým aktuálně komunikuje.

Na závěr se dá říci, že Lend And Learn kombinuje některé funkcionality těchto aplikací, aby umožnila vytvoření nejvhodnější platformy pro daný účel.

2.3 Analýza uživatelů

V této sekci práce je provedena analýza uživatelů. Správná specifikace jednotlivých skupin uživatelů a vyhodnocení jejich požadavků je klíčové pro budování aplikace, která je maximálně uživatelsky přívětivá.

Aplikace Lend And Learn má jasně daného koncového uživatele. Tím je student či zaměstnanec FIT. Pohybují se v aplikaci stejně a mají stejná práva. Je zapotřebí brát v úvahu uživatelská rozhraní již existujících aplikací na fakultě, které jsou studenti či zaměstnanci zvyklí používat.

Na druhou stranu sebemenší část aplikace bude přístupná veřejnosti, a to hlavní stránka. Proto je uživatel níže rozdělen na přihlášeného a nepřihlášeného.

- **Nepřihlášený uživatel**
  Takový uživatel uvidí hlavní stránku, která ho stručně informuje o aplikaci. Uvidí tlačítko na přihlášení a bude moci ho použít. Po kliknutí na přihlášení uvidí formulář serveru fakulty, kde bude moci zadat své údaje.

- **Přihlášený uživatel**
  Po úspěšném zadání svých údajů se uživatel stane přihlášeným. Může využívat všechny funkcionality, které aplikace nabízí. Nicméně nemůže si zobrazit hlavní stránku s přihlášením.

Kapitola 3

Návrh

Tato kapitola se věnuje návrhu prototypu aplikace Lend And Learn. Představuje se zde volba návrhových vzorů. Dále je v příslušné sekci provedena rešerše vhodných technologií, programovacích jazyků, frameworků a knihoven. Popisuje se jejich účel, výhody a případné nevýhody. V další sekci je k nalezení návrh databáze včetně diagramu. Konkrétně jsou popsány jednotlivé tabulky, jejich sloupce a vazby mezi sebou.

3.1 Architektura

V této sekci je popsán způsob navržení aplikace a způsob komunikace jednotlivých částí.

3.1.1 API

API je způsob komunikace samostatných aplikací nebo částí jedné aplikace. Pro komunikaci mezi frontendem a backendem aplikace Lend And Learn se využívá návrhového vzoru REST. Klient se dotáže serveru pomocí konkrétní URL adresy a ten mu pošle potřebnou reprezentaci dat, se kterými klient následně manipuluje. Tato data jsou zpravidla ve formátu JSON. Každý dotaz musí být na sobě nezávislý. [6]

Liew ve svém článku [6] popisuje čtyři komponenty, ze kterých se skládá API dotaz. Jde o endpoint neboli URL adresu, metody, hlavičky a samotná data.

První část endpointu tvoří URL adresa konkrétního API. Druhá část je specifikace samotného dotazu, ze které je jasné, o jaká data klient žádá server. Říká se tomu cesta. Veškeré možné cesty jsou popsány v dokumentaci konkrétního API, pokud tedy existuje. Pokud ne, tak je potřeba mít k implementaci serveru přístup a seznámit se s kódem a s tím, co všechno nabízí a umožňuje. Endpoint je možné rozšířit o parametry, které ještě přesněji specifikují do-

taz. Takové parametry sestávají z klíče a jeho hodnoty a v dotazu se oddělují pomocí `&`.

Metoda určuje typ dotazu. Přesněji řečeno říká serveru, jakým způsobem dotaz zpracovat. Následující metody jsou nejvíce používané.

GET metoda se používá k získání konkrétních dat. Server požadovaná data získá a následně vrátí klientovi.

POST je metoda pro vytváření. Typicky se používá k vytvoření nového záznamu v databázi a vrací informaci, zda bylo vytvoření úspěšné.

PUT dotaz je určený k aktualizaci databázového záznamu. Také vrací informaci o úspěchu nebo neúspěchu zpracování dotazu.

DELETE metoda říká serveru, že má z databáze odstranit určitý záznam. Opět vrací informaci, zda se vykonání dotazu povedlo.

V hlavičce dotazu je možné uvést formát, ve kterém se data posílají. Dále lze přidat například access token, pomocí kterého server provede autentizaci dotazu.

Zbývají už data, která se posílají serveru. Ten je podle typu dotazu zpracovává.

Jak se výše uvádí, server na některé dotazy odpoví informací o tom, zda bylo zpracování dotazu úspěšné. Tato informace je zpravidla ve formě HTTP kódů. Kódy z rozsahu [200, 300) informují klienta o úspěchu. Naopak kódy počínaje 400 značí neúspěch, a to pokud je kód v rozsahu [400, 500), tak se jedná o chybu na straně dotazujícího se klienta. Při vrácení kódu od 500 a výš se jedná o chybu na straně serveru, který dotaz zpracovává. [7]

3.1.2 MVC

Při procesu vývoje softwaru je potřeba myslet na jeho udržitelnost, přehlednost a rozšiřitelnost. To všechno usnadní budoucí vývoj aplikace. Nezkušený vývojář, který pracuje na projektu sám, má tendenci ukládat logiku kódu, přístup do databáze a zobrazení výstupu do jednoho souboru, jelikož má myšlenku celého projektu v hlavě a nemyslí si, že by mu dělalo potíže cokoliv v kódu najít. Nicméně projekty se velice rychle rozrůstají a takové soubory jsou nepřehledné. Často se k vývoji přidá další člověk, pro kterého orientace v kódu už tak snadná nebude. Právě architektonické vzory mají za účel vývojáři pomoct svůj kód dobře organizovat, aby se s těmito problémy nesetkal.

Jedním z nejvíce používaných vzorů při tvorbě webové aplikace je MVC. Tento vzor rozděluje kód programu do tří komponent, kterými jsou *Model*, *View* a *Controller*. [8]

Celý proces začíná, když uživatel pošle API dotaz. Router na základě URL adresy rozhodne, který Controller zavolat. Controller přijme parametry a předá je příslušnému Modelu. Model odpovídá za logiku aplikace. Přijme parametry od komponenty Controller, provede potřebný výpočet a následně výsledek vrátí. V Controlleru je možné volat i více Modelů, aby se povedlo získat celkový výsledek. Na konci Controller předá data View vrstvě, která

Obrázek 3.1: Návrhový vzor MVC

tato data zpracuje a zobrazí uživateli. Z tohoto postupu je vidět, že Controller spojuje vrstvy Model a View. [8] Tok dat je znázorněn na obrázku 3.1, který je inspirovaný článkem [8] od vývojářů Google.

3.2 Vybrané technologie

V této části jsou popsané technologie, které se v projektu používají.

3.2.1 TypeScript

TypeScript je open-source programovací jazyk, jehož autorem je společnost Microsoft. Je implementován jako nadstavba nad jazykem JavaScript, kde autoři přidali statické typování. [9]

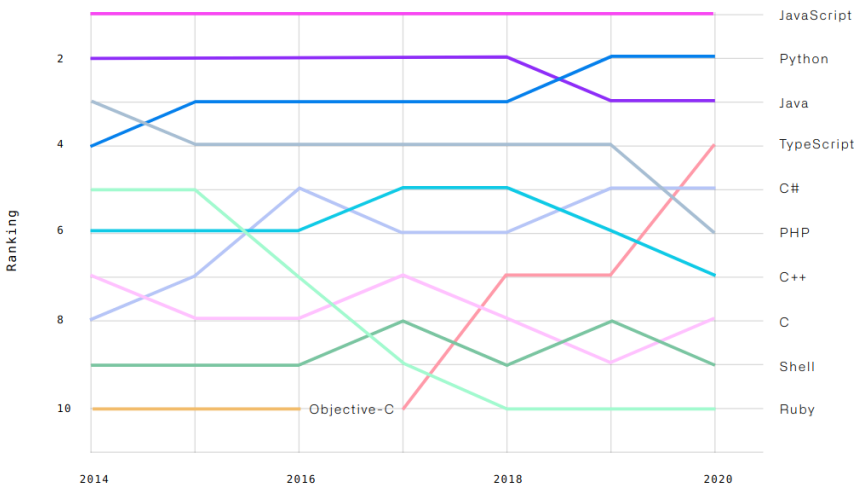
Statické typování požaduje přiřazení typu proměnné již při její deklaraci. Programátor si tak musí typy nadefinovat předtím než se jednotlivé proměnné použijí. Kontrola použití datových typů probíhá při kompilaci. [10] Je to velmi vhodné pro zamezení a včasné odhalení chyb. Dokáže to program i výrazně zrychlit, jelikož se značně eliminují kontroly za běhu aplikace. Příklady staticky typovaných jazyků jsou podle článku [11] C, C++, Java.

U dynamického typování se nepožaduje specifikace datového typu u proměnné, proto může proměnná odkazovat na jakoukoliv hodnotu. Typová kontrola se provádí při běhu programu. [10] Kód se tak v pořádku zkompiluje i přesto, že může obsahovat chyby v typování a v aplikaci se to nemusí projevit okamžitě. Ve velkém projektu může trvat podstatně déle takovou chybu zpětně dohledat. Nicméně dynamicky typované jazyky obsahují i řadu výhod. Jsou například flexibilní a umožňují kratší kód a rychlejší vývoj. Opět ve článku [11] autor uvádí některé dynamicky typované jazyky jako například velmi populární JavaScript, PHP, Python.

Oproti JavaScriptu je TypeScript silně typovaný. [12] Díky tomu nedochází k přetypování a chybám jako například přiřazení čísla do proměnné, která očekává text nebo opačně.

Psaní typů v TypeScriptu není povinné. Metody nebo proměnné si totiž typ umějí odvodit od kontextu použití. Tomuto procesu se říká *type inference*. Specifikace typů se nicméně vřele doporučuje, jelikož snižuje pravděpodobnost výskytu chyby způsobené lidským faktorem. [9]

TypeScript je v dnešní době velmi populární volbou pro implementaci backendové logiky. Z ročních GitHub statistik 3.2 je vidět, že v roce 2020 patřil mezi 4 nejpoužívanější jazyky na této platformě.



Obrázek 3.2: Nejpoužívanější jazyky na platformě GitHub za několik posledních let [2]

3.2.2 Node.js

Node.js je asynchronní běhové prostředí JavaScriptu řízené událostmi. Tato technologie je navržena tak, aby umožňovala programátorům psaní škálovatelných aplikací. [13]

Jedná se o open-source, což samo o sobě podporuje vytváření velké komunity a tím i značný prostor pro vylepšení. Další velkou výhodou použití tohoto prostředí je snadný přechod mezi serverovou a klientskou částí. Není potřeba osvojovat si nový jazyk, jelikož se na obou stranách používá JavaScript.

Node.js přináší NPM neboli Node package manager. Je to nástroj pro instalaci balíčků, který je v praxi velice používaný. Společnost vlastní veřejně přístupný registr balíčků, který je největším softwarovým registrem na světě. Napočítá již přes milion balíčků. Nahrání nebo stažení a použití balíčků je bez jakýchkoliv poplatků. [14]

3.2.3 NestJS

NestJS je progresivní Node.js framework. Je ideální možností pro psaní v jazyce TypeScript, protože ho plně podporuje, nicméně umožňuje také použití čistého JavaScriptu. Jeho cílem je vytváření dobře testovatelných, snadno udržovatelných aplikací, kde nevznikají úzké vazby. [15]

Využívá návrhového vzoru Dependency injection. [16] Ten funguje na principu, že třída získá veškeré další objekty, na kterých závisí, například v kon-

struktoru. Tzn., že není potřeba vytvářet globální proměnné nebo se nějakým způsobem snažit získat závislosti uvnitř třídy. [17] TypeScript správu závislostí velmi usnadňuje, jelikož nabízí typování.

3.2.4 Docker

Při spuštění aplikace na jiném zařízení se programátor musí postarat o stažení veškerých potřebných závislostí a zkontrolovat verze potřebných balíčků. Při současném spuštění více aplikací se může vyskytnout problém, kdy se obě aplikace usilují o stejný port. Tyto problémy řeší Docker.

Docker je platforma, kde lze aplikaci spustit pomocí kontejnerů. [18] Kontejner je izolované prostředí, které obsahuje samotný kód a také všechny potřebné závislosti aplikace včetně běhových prostředí. Kontejner je spuštěn v běhovém prostředí Docker Engine. [19]

Docker se velice často využívá pro *Continuous integration* a *Continuous delivery*. [18]

Průběžná integrace neboli *continuous integration* je technika vývoje softwaru, při které vývojáři integrují kód do sdíleného repozitáře a následně při každé integraci proběhne spuštění testů, díky čemuž se rychle odhalí případné chyby. [20]

V případě *continuous delivery* se jedná o techniku průběžného nasazování projektu, kdy se kód vyvíjí, testuje a nasazuje ve velice krátkých časových úsecích. [21]

Mimo podporu výše zmíněných technik, Docker je rychlý. Jeho kontejnery potřebují mnohem méně místa než například virtuální stroje, proto je spuštění více kontejnerů současně pohodlné a efektivní. [19]

3.2.5 PostgreSQL

Kvůli struktuře prototypu aplikace Lend And Learn a vazbám tabulek je zapotřebí zvolit relační databázi. Při výběru konkrétního databázového systému se kladl důraz na jistotu. Proto byl PostgreSQL logickou volbou.

Jedná se o open-source formu a díky tomu se na vývoji a neustálém vylepšování může aktivně podílet kdokoliv. [22] Nelze vynechat, že PostgreSQL je velice populární a využívají ho giganti jako Instagram nebo Netflix. [23] Popularita technologií je dost klíčová ke hledání vývojáře pro budoucí vývoj nebo podporu. PostgreSQL používá dotazovací jazyk SQL.

3.2.6 TypeORM

ORM neboli objektově relační mapování je technika, která řeší mapování objektů do relační databáze. [24] Pro tuto konverzi dat bylo zvoleno TypeORM, jelikož je psaný v jazyce TypeScript. [25]

Jako svou výhodu oproti konkurenci autoři TypeORM uvádí možnost použití obou návrhových vzorů, konkrétně Active Record a Data Mapper. [26]

Při použití Active Record se k databázi přistupuje přímo v modelech entit. Naopak v případě návrhového vzoru Data Mapper se metody, které přistupují k databázi, definují v samostatných souborech, kterým se říká repozitáře. [27]

Pro aplikaci Leand And Learn byl zvolený vzor Data Mapper, jelikož díky tomuto postupu se snadněji udržuje přehlednost, zvlášť v aplikacích, které plánují růst.

3.2.7 Class-validator

Je vhodné v této kapitole zmínit také balíček *class-validator*, který je zdarma ke stažení v registru NPM. Jedná se o knihovnu, která se v projektu Lend And Learn využívá k validaci atributů tříd pomocí dekorátorů. Dle [28] je psaná v jazyce TypeScript a vhodná k použití spolu s Node.js.

3.2.8 Pg-mem

Pg-mem je knihovna, pomocí které lze napodobit PostgreSQL databázi tzv. *in-memory*. To znamená, že se ukládá do počítačové paměti místo disku. Autor uvádí, že se jedná o experimentální knihovnu, která je poměrně nová. Nicméně začátkem března 2021 tato knihovna měla necelých 4000 stažení za týden. [29]

Tato knihovna je v projektu použita pro testování backendu. Jejím velkým konkurentem na trhu je SQLite, ta avšak nepodporuje strukturu Enum, která se v aplikaci Lend And Learn využívá.

3.2.9 Passport

„*Passport je autentizační middleware pro Node.js.*“ [30] (překlad autora) Obsahuje tzv. strategie, které lze ve vlastní aplikaci použít k přihlášení pomocí účtů třetích stran, jako například Facebook, Twitter apod. [30] V prototypu této bakalářské práce se využívá strategie *passport-oauth2*. Tato strategie funguje na základě protokolu OAuth 2.0.

3.2.10 OAuth 2.0

OAuth 2.0 je autorizační protokol, pomocí kterého aplikace může přistupovat k API na základě uživatelských oprávnění, aniž by měla k dispozici jeho přístupové údaje. [31] Aplikace tak nemusí ukládat ani podporovat autentizaci uživatelských údajů, a to je velký bezpečnostní přínos.

V dokumentu [31] Hardt definuje 4 role, které jsou zásadní pro pochopení jak tento protokol funguje.

- **Majitel zdrojů**
  Za majitele zdrojů se zpravidla považuje osoba, která je schopná poskytnout přístup k některým chráněným zdrojům. Příkladem je člověk, který se do aplikace přihlásí a schválí přístup.

- **Zdrojový server**
  Jedná se o server, kde jsou chráněné zdroje uloženy. Tento server je schopný tyto zdroje poskytnout třetí aplikaci, která o ně žádá.

- **Klient**
  Klientem je každá aplikace, která se snaží získat přístup ke chráněným zdrojům na základě žádosti od majitele těchto zdrojů.

- **Autorizační server**
  Tento server ověřuje identitu majitele zdrojů a posílá klientovi tzv. access token. Pojem access token je vysvětlený níže.

Access token jsou data, která se používají k identifikaci uživatele nebo aplikace a svým obsahem uživatele nebo aplikaci reprezentují. [31]

Samotný proces probíhá následně. Klient požádá majitele zdrojů o autorizaci. Po úspěšné autorizaci klient požádá o access token. Autorizační server provede autentizaci žádosti a pokud bude úspěšná, tak klientovi poskytne access token. Dále klientovi stačí dotázat se zdrojového serveru pomocí tokenu. Zdrojový server token ověří a v případě úspěchu poskytne data. [31]

S tímto protokolem můžou souviset počítačové útoky podvodníků, zejména phishing. Phishing je taková podoba počítačových útoků, kde podvodníci zjistí citlivá data uživatele, jako například uživatelské jméno či heslo. Hardt v dokumentu [31] uvádí, že potenciální problém je přesměrování kvůli přihlášení, kde může dojít k záměně adresy stránky, aniž by si toho uživatel všiml. V takovém případě se uživatel přihlásí na webové stránce podvodníků, která dost často vypadá naprosto shodně s tou pravou, a prozradí tím své údaje.

3.2.11 React

React je knihovna, která je založená na komponentách, pro implementaci uživatelského rozhraní. Je vytvořená dobře známou společností Facebook původně pro vlastní potřeby. [32]

Pokud by se o této knihovně mluvilo v kontextu návrhového vzoru MVC, tak React je view vrstva. To znamená, že odpovídá za zobrazování. [33]

React předpokládá, že komponenty jsou základem klientské vrstvy aplikace. Ty se potom dají poskládat do větších celků. Příkladem může být přihlašovací formulář. Ten se zpravidla skládá z tlačítek a políček pro zadání vstupních informací. Tato tlačítka a políčka jsou typicky komponenty, které na jednom místě shromažďuje nějaká další komponenta a tím specifikuje strukturu přihlašovacího formuláře.

Pro následující odstavec je potřeba vysvětlit pojem DOM neboli Document Object Model. DOM je objektová podoba HTML dokumentu a často se o ní mluví jako o stromu. Ačkoliv DOM vychází z HTML, ne vždy má stejnou strukturu. HTML strom například neobsahuje elementy, které jsou skryté pomocí stylů. DOM naopak ano. [34]

React efektivně vykreslí komponenty, ve kterých se mění data, a to díky technologii virtuální DOM. [32] Díky tomuto konceptu není potřeba při každé změně nahrazovat celý DOM, stačí pouze aplikovat změny.

3.2.12 Axios

Axios je HTTP klient, díky kterému je možná snadná komunikace s API. [35] Axios podporuje Promise, které jsou často obtížné k pochopení, a proto jsou níže stručně vysvětleny.

Promise je objekt, jehož hodnota nemusí být ze začátku známá, ale reprezentuje očekávanou hodnotu, která se v průběhu volání asynchronní funkce vrátí. Uchovává tedy informaci o úspěšnosti funkce a případné vrácené hodnotě. [36]

Při inicializaci se Promise nachází v čekajícím stavu, jemuž se říká *pending*. Promise je splněný neboli *fulfilled*, pokud se asynchronní funkce úspěšně vykoná. V případě neúspěchu je Promise odmítnutá či *rejected*. [36]

Jednou z hlavních funkcí Promise je možnost přidat callback neboli funkci, která se zavolá až asynchronní funkce dokončí svůj běh. Zpropagování chyby je také snadné. V případě prvního selhání se Promise přesune do bloku, který lze nadefinovat pomocí `catch()`. V tomto bloku se specifikuje, jak s chybou naložit. [37]

Jednou z dalších funkcí, kterými axios disponuje, je tzv. *interceptor*. Interceptor se definuje jak u dotazů, tak u odpovědí. U odpovědí jde o možnost zachytit objekt s odpovědí předtím než se dostane do funkce, která ho očekává, a provést potřebnou operaci. Například je možné si upravit podobu chybové hlášky. V případě dotazů se interceptor využívá například k zpropagování tokenu.

3.2.13 React Redux

Tato knihovna umožňuje vytvoření tzv. store neboli úložiště potřebných informací a jeho propojení s React komponentou. [38] Je napsaná v jazyce TypeScript [39], a proto není potřeba si do projektu Lend And Learn stahovat externí balíček s typy. Komponenta si získá potřebná data z Redux store a díky optimalizacím, které React Redux implementuje, se komponenta znovu vykreslí pouze v případě, že se data v Redux store změní. [38]

Reducer je funkce, která přijímá aktuální stav Redux store a akci a na základě akce, zpravidla pomocí *switch*, vrací nový stav. Akce je objekt s povinným parametrem *type*, na základě kterého se reducer rozhoduje o změně. Dalšími parametry jsou změny, které akce propaguje.

3.2.14 Redux-Saga

S React Redux souvisí potřeba zvládat asynchronní operace, jako například volání API. Takovým operacím se říká *side effects* a pro jejich snadnou imple-

mentaci a testování vznikla knihovna Redux-Saga. Dle [40] je to middleware, kde se používají speciální funkce generátory.

Generátor je funkce, jejíž běh je možné pozastavit a později opětovně spustit, aniž by se přišlo o její kontext. [41] V kódu se definice této funkce značí `function*`.

Autoři oficiální dokumentace této knihovny [42] popisují dva typy funkcí Saga.

- **Watcher**
  Taková Saga naslouchá na akce a v případě, že se některá zpropagovala, zavolá příslušnou Saga.

- **Worker**
  Worker je Saga, kterou zavolá Watcher. Vykoná hlavní práci.

Worker Saga typicky obsahuje instrukce pro middleware, kterým předchází `yield`. Příkladem instrukce může být propagace akce do Redux store, jelikož do něho má plný přístup. [43] Nejčastější praxí je si tyto instrukce nadefinovat postupně za sebou.

Použitím yield a operací z knihovny není potřeba v testech vytvářet mock těchto operací, ale pouze dat, která vrací, což bývá mnohem jednodušší. [44]

3.2.15 React Router

React Router je knihovna, díky které lze jednotlivé části aplikace zpřístupnit pod určitou URL adresou. Přesměrování funguje například po kliknutí na nějaký prvek na stránce. API této knihovny poskytuje opravdu velký výběr komponent a funkcionalit, které správu URL adres v projektu usnadní. To zahrnuje například zacházení s parametry v adresách nebo bezproblémové přesměrování z Redux Saga.

3.2.16 Material Design

Jednoduchý design je základ úspěšné aplikace, jelikož má za cíl poskytnout uživateli snadný pohyb v aplikaci. Existuje mnoho aplikací, které za sebou nesou zajímavou myšlenku, ale nemají přehlednou navigaci a uživatel určitou funkcionalitu hledá příliš dlouho. Takové aplikace nedokážou uživatele na své webové stránce udržet. Material Design obsahuje sadu doporučení jak dodržet konzistentní vzhled své webové aplikace. Popisuje, jaké komponenty může aplikace obsahovat a jakým způsobem se mají chovat.

3.2.17 Material-UI

Material-UI vychází z Material Design a jako React UI framework přináší široký výběr komponent, které lze použít pro stylování webové aplikace. Díky

jejich API lze komponenty přizpůsobit svému designu. Tento framework shromažďuje barvy, styly písma a velikosti obrazovek pro responzivitu neboli tzv. *breakpoints* v jednom souboru a říká tomu *theme*. Tu je možné libovolně předefinovat tak, aby odpovídala požadavkům na design.

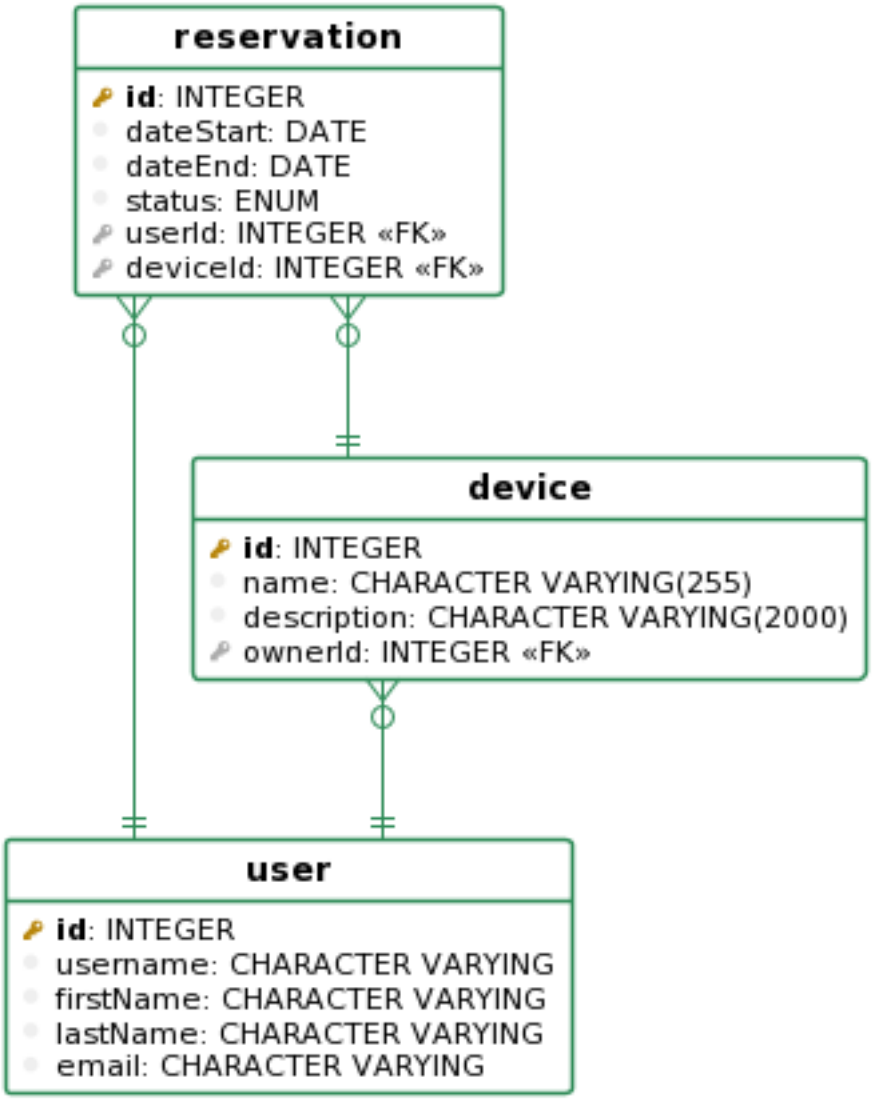
3.3 Návrh databáze

Návrh databáze je zobrazen na diagramu 3.3.

Tabulka **user** znázorňuje uživatele. U této tabulky se eviduje uživatelské jméno, jméno, příjmení a e-mail. Tyto sloupce jsou typu *varchar*, který značí řetězec, jehož délka není pevná. Dále obsahuje primární klíč, který má typ *int*. Navržení této tabulky úzce souvisí s předpokladem, že se uživatelé získávají z fakultní databáze Usermap. Z tohoto důvodu uživatelské jméno, jméno, příjmení a e-mail nejsou omezená vlastní délkou. Předpokládá se totiž, že Usermap taková omezení implementuje a dodržuje a v této aplikaci nelze vyloučit jakéhokoliv uživatele, který je v databázi Usermap.

Dále je v projektu tabulka **device**, která eviduje údaje o zařízení. Jméno zařízení je typu *varchar*, který je tentokrát omezený délkou 255 znaků. Dále se nepovinně eviduje popis zařízení, jehož délka je maximálně 2000 znaků. Tato délka je zkušební a v budoucnu se může upravit podle využití v reálném provozu. Dále obsahuje tabulka primární klíč typu *int*, ale také cizí klíč, který identifikuje majitele zařízení. Mezi těmito tabulkami existuje vazba, kde jeden uživatel může vlastnit více zařízení, ale zařízení má pouze jednoho vlastníka.

Poslední tabulka je **reservation**. Eviduje se datum začátku a datum konce rezervace, která jsou typu *date*. Tento typ slouží k uchování dne, měsíce a roku, ale neumožňuje uchování času. Pro prototyp této bakalářské práce není potřeba evidovat konkrétní čas. Dále se eviduje status rezervace, který je typu *enum*. Může nabývat čtyř hodnot, které odpovídají stavům vytvoření, schválení, aktuálnímu průběhu a ukončení rezervace. Tato tabulka obsahuje primární klíč a dva cizí klíče. První cizí klíč je uživatelův identifikátor. Jde o uživatele, který rezervaci vytvořil. Druhý cizí klíč odpovídá zařízení, pro které byla rezervace vytvořena. Tato tabulka má vazbu na další dvě tabulky, a to na uživatele a na zařízení. Uživatel může vytvořit více rezervací, ale jedna rezervace má pouze jednoho uživatele, který ji vytvořil. Jedna konkrétní rezervace je vytvořená právě pro jedno zařízení, ale pro zařízení může být vytvořeno více rezervací.



Obrázek 3.3: UML diagram tabulek v projektu Lend And Learn

Kapitola 4

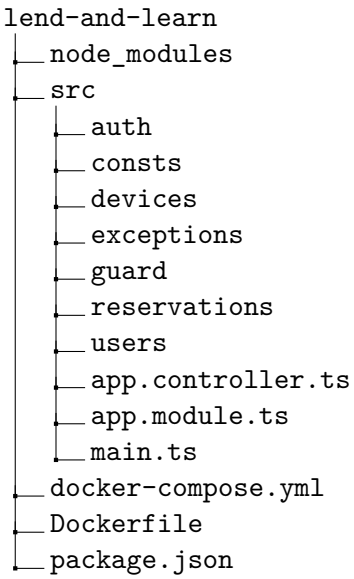
Implementace

V této kapitole je popsána implementace aplikace Lend And Learn s ohledem na zvolené technologie. Jsou zde zmíněny nastroje, které se pro vývoj použily. Dále je popis implementace rozdělený do dvou částí, a to backend a frontend. Některé sekce jsou doplněny ukázkami kódu nebo obrázky se vzhledem aplikace. Také je zde zmíněn způsob dokumentace projektu a jak tuto dokumentaci zobrazit.

4.1 Vývojové nástroje

Projekt je rozdělený do dvou částí, a to backend a frontend. Každá část byla verzována a spravována pomocí systému *Git*. Git repozitář byl uložen a průběžně aktualizován na platformě GitLab, kde má FIT vlastní server. Vzhledem k tomu, že se na projektu nepodílelo více vývojářů, veškerý vývoj probíhal na větvi *master*. Byl kladen důraz na maximální využití *commitů*, kde jeden commit obsahoval nějaký celek práce. V případě výskytu chyby je právě díky commitům možné pohodlně se pohybovat v předchozích stavech projektu.

Obě části byly vyvíjeny ve vývojovém prostředí WebStorm. WebStorm obsahuje velice přehlednou základní správu verzování, což dokáže efektivně ušetřit čas, jelikož není potřeba ručně zadávat Git příkazy. Navíc je to oproti klasickému terminálu pro člověka přirozenější. WebStorm má širokou nabídku pluginů, které vývoj usnadňují. Příkladem je plugin Database Tools and SQL. Díky němu lze k databázi přistupovat nebo ji upravovat přímo z prostředí WebStorm. Další plugin, který byl v projektu využit, je GNU GetText files support (*.po). Tento plugin podporuje soubory s koncovkou *.po*, v nichž se nachází překlady.



Obrázek 4.1: Struktura backendu

4.2 Implementace backendu

Implementace backendu popisuje strukturu projektu a základní komponenty, ze kterých se backend skládá. U těchto komponent se vysvětluje jejich fungování včetně ukázek kódu pro lepší představu. Popsána je rovněž důležitá část projektu, a to proces přihlášení pomocí FIT účtu. Nakonec je stručně popsán proces autentizace dotazů.

4.2.1 Struktura projektu

Diagram 4.1 ukazuje zjednodušený adresářový strom backendu.

Soubor **package.json** uchovává potřebná metadata projektu, například jméno autora, název či verzi projektu. Dále se v tomto souboru definují skripty, které lze spouštět pomocí npm (viz sekce 3.2.2). Jsou to například skripty pro spuštění aplikace nebo spuštění automatických testů. Dále se zde ukládají veškeré závislosti. Je to užitečné, jelikož člověk, který bude projekt spouštět pomocí npm, nemusí stahovat závislosti zvlášť. Příkaz `npm install` tyto závislosti, které jsou definované v souboru **package.json**, stáhne automaticky a uloží je do složky **node_modules**.

Docker kontejner (viz sekce 3.2.4) je instance Docker image. **Dockerfile** je soubor, který shromažďuje instrukce pro vytvoření Docker image.

Příkazem 4.1 se specifikuje konkrétní Docker image. Zde se používá image pro Node.js, který se zakládá na linuxové distribuci *alpine*. Jeho výhoda je, že je podstatně menší, než většina ostatních obrazů, jež vychází z jiných linuxových distribucí.

```
FROM node:15.10.0-alpine3.10
```

Kód 4.1: Specifikace Docker image

```
[název]
├── [název].entity.ts
├── [název].repository.ts
├── [název].controller.ts
├── [název].service.ts
└── [název].module.ts
```

Obrázek 4.2: Struktura složek backendu

V souboru **docker-compose.yml** se kombinují kontejnery a jejich závislosti na sobě. Například spuštění backendu závisí na databázi. Dále se definuje například vnitřní port, který je využívaný spuštěnou aplikací v kontejneru, a vnější port, přes který lze k takové aplikaci přistoupit.

Main.ts je soubor, ve kterém se vytváří instance Nest aplikace. Specifikuje se zde i port, na kterém aplikace naslouchá příchozím HTTP dotazům.

Ve složce **consts** se nacházejí konstanty nebo výčty, které se mohou v projektu vyskytnout na více místech.

Složka **exceptions** slouží k uchovávání vlastních výjimek, které se rovněž používají na více místech v kódu.

Soubory ve složkách **auth** a **guard** jsou určeny k přihlášení a ověření identity uživatele. Vzhledem ke komplexnosti těchto procesů jejich princip fungování je popsán v sekcích zvlášť, a to 4.2.7 a 4.2.8.

Zbylé složky tvoří API aplikace. Jejich typická struktura je znázorněna na obrázku 4.2.

4.2.2 Entity

Pro každou databázovou tabulku je vytvořený konceptuální datový model, čímž je Entity. Jedná se o třídu, která je označená dekorátorem `@Entity()`. Taková třída se skládá ze sloupců a musí mít primární sloupec, který slouží k identifikaci databázového záznamu. Výpis kódu 4.2 ukazuje třídu Entity pro uživatele. Identifikátor je číslo, které se automaticky navyšuje při vkládání záznamu, a to díky dekorátoru `@PrimaryGeneratedColumn()`. Pomocí dekorátorů se také jednoduše definují vazby. Například je zde patrné, že uživatel může vlastnit více zařízení nebo vytvořit více rezervací.

```
@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ unique: true })
  username: string;

  @Column()
  firstName: string;

  @Column()
  lastName: string;

  @Column()
  email: string;

  @OneToMany(() => Device, (device) => device.owner)
  ownedDevices: Device[];

  @OneToMany(
    () => Reservation,
    (reservation) => reservation.user
  )
  reservations: Reservation[];
}
```

Kód 4.2: Entity pro uživatele

4.2.3 Repository

Repository je třída, která umí pracovat s konkrétní tabulkou. Tyto třídy jsou implementovány v knihovně TypeORM (viz sekce 3.2.6) a nabízí vývojáři řadu užitečných metod, jako například metody pro vytvoření záznamu do tabulky, vkládání záznamu, jeho nalezení, smazání či editace.

Často ale tyto základní metody pro komplexnější projekt nestačí, proto existuje možnost vytvořit si vlastní Repository. Zpravidla taková třída vychází z Repository, které poskytuje knihovna, ale obsahuje vlastní metody navíc.

Pro složitější SQL dotazy se používá metoda `createQueryBuilder()`, která je z knihovny TypeORM. V kódu 4.3 je vyobrazena vlastní třída Repository, jež obsahuje metody s přístupem do databázové tabulky rezervací. Metoda `countCollisions()` počítá počet kolizí rezervací. Výpočet spočívá

```
@EntityRepository(Reservation)
class ReservationsRepository extends Repository<Reservation> {
   countCollisions(
    dateStart: Date,
    dateEnd: Date,
    deviceId: number
  ): Promise<number> {
    return this.createQueryBuilder()
     .where(
       `
       (
         (
           :start BETWEEN "dateStart" AND "dateEnd"
           OR :end BETWEEN "dateStart" AND "dateEnd"
         )
         OR
         (:start <= "dateStart" AND :end >= "dateEnd")
       )`,
      )
       .andWhere(
       '"status" IN (:optionCreated, :optionInProgress)'
      )
      .andWhere(':id = "deviceId"')
      .setParameters({
        start: dateStart,
        end: dateEnd,
        id: deviceId,
        optionCreated: ReservationsStatus.Created,
        optionInProgress: ReservationsStatus.InProgress,
      })
      .getCount();
    }
}
```

Kód 4.3: Repository pro rezervace

ve vyhledání rezervací pro konkrétní zařízení, které jsou vytvořené, nebo aktuálně probíhají, a u kterých kolidují data začátku nebo ukončení rezervace. Implementace této metody využívá právě Query Builder.

4.2.4 Controller

Soubory s koncovkou controller.ts definují URL adresy API dotazů. Pomocí dekorátorů se určí HTTP metoda, případně adresa. Dále je možné přidat dekorátor, který konkrétní dotaz autentizuje. Controller přijímá dotaz s případnými parametry od klienta, deleguje dotaz dál, shromáždí potřebné výpočty a pošle klientovi odpověď.

V ukázce kódu 4.4 se vytváří Controller, který je přístupný na adrese *devices*. Tento Controller spravuje dotazy ohledně zařízení. V konstruktoru přijímá potřebné závislosti, v tomto případě DevicesService. Následuje asynchronní funkce, která si z adresy získá parametr id. Tělo této funkce je obaleno do bloků try a catch. V části try se zavolá metoda, která se nachází v DevicesService. Ta má za úkol smazat zařízení. Část catch hlídá, zda při vykonání funkce v části try nenastane chyba. Pokud ano, vyhodí výjimku, což zabrání spadnutí programu. Dekorátor `@Delete()` určuje HTTP metodu a také koncovku URL adresy. Takto dotaz na adrese *devices/1* s metodou DELETE se váže na tuto funkci, kde 1 je identifikátor zařízení. Dekorátor `@UseGuards()` přijímá v parametrech třídu, která implementuje autentizaci dotazu. Před provedením se tak dotaz autentizuje.

4.2.5 Service

Service odpovídá za logiku dotazu a v NestJS se označuje jako Provider. Jde o třídu, která je označená dekorátorem `@Injectable()`. Díky této anotaci NestJS dovoluje různé provázání takových tříd.

V ukázce 4.5 je implementována třída DevicesService, která v projektu zodpovídá za veškerou logiku manipulace se zařízeními. Konkrétně je zde vidět asynchronní metoda, která do databáze přidá nové zařízení. Zařízení musí mít validního autora, proto tato metoda posílá požadavek na nalezení uživatele. K tomu použije UsersService, která provádí výpočty ohledně uživatelů. To je možné právě díky tomu, že třída UsersService je také označená dekorátorem `@Injectable()`. Následně využívá metod z použité knihovny TypeORM, která přes Repository pracuje s konkrétní tabulkou v databázi. Pomocí těchto metod vytvoří a uloží do databáze nové zařízení. DevicesService ve svém konstruktoru přijímá DevicesRepository, a to opět pomocí dekorátoru `@InjectRepository()`, který knihovna TypeORM poskytuje.

4.2.6 Module

NestJS sází na modulární strukturu kódu. Modul shromažďuje všechny výše zmíněné soubory. Napomáhá to lepší organizaci kódu. Jednotlivý modul se potom může importovat vícekrát v projektu.

V jednom modulu se definují i ostatní moduly, které se importují. Dále se uvádí třídy, které patří do *controllers* a *providers*. Všechny soubory, které jsou uvedeny v *providers*, je možné sdílet v rámci celého modulu. Do *exports*

```
@Controller('devices')
export class DevicesController {
  constructor(private readonly deviceService: DevicesService) {}

  @Delete(':id')
  @UseGuards(ValidationGuard)
  async delete(@Param('id') id: number, @Req() req) {
    try {
      await this.deviceService.deleteDevice(id, req.userName);
    } catch (e) {
      if (e instanceof DeviceNotFound) {
        throw new BadRequestException('Device not found');
      }
      if (e instanceof DeviceWithActiveReservations) {
        throw new BadRequestException({
          code: CustomCodes.DeviceWithActiveReservations,
          message: 'Device has some active reservations',
        });
      }
      throw e;
    }
  }
}
```

Kód 4.4: Controller pro zařízení

se zapisují všechny *providers*, které mohou ostatní moduly importovat. Entity a Repository potřebují zvláštní registraci v modulu, a to pomocí metody `forFeature()`.

Výpis kódu 4.6 je ukázkou modulu pro jeden celek aplikace, konkrétně rezervace.

Kromě těchto menších modulů v projektu existuje jeden hlavní modul, a to v souboru **app.module.ts**. Ten importuje všechny moduly, které tvoří aplikaci. Dále importuje TypeOrmModule, který se zde rovněž konfiguruje. Entity se zde načítají automaticky, a to eliminuje častý problém, kdy je vývojář zapomene přidat do hlavního modulu.

4.2.7 Proces přihlášení

Přihlášení probíhá pomocí FIT účtu. V procesu přihlášení spolu komunikují následující strany:

- **LAL server** (Lend And Learn server),

```
@Injectable()
export class DevicesService {
  constructor(
    @InjectRepository(DevicesRepository)
    private devicesRepository: DevicesRepository,
    private usersService: UsersService,
  ) {}

  async addNewDevice(
    name: string,
    description: string,
    username: string
  ): Promise<Device> {
    const owner = await this.usersService.findOne(username);
    if (!owner) {
      throw new UserNotFound();
    }
    const device = this.devicesRepository.create({
      name,
      description,
      owner,
    });
    return this.devicesRepository.save(device);
  }
}
```

Kód 4.5: Service pro zařízení

- **LAL klient** (Lend And Learn klient),
- **LAL databáze** (Lend And Learn databáze),
- **FIT server** (auth.fit.cvut.cz server),
- **FIT klient** (auth.fit.cvut.cz klient).

Ze začátku je potřeba svou aplikaci registrovat u poskytovatele ČVUT API. V tomto případě to lze udělat v aplikaci AppsManager, která se nachází na adrese `https://auth.fit.cvut.cz/manager`. Po přihlášení se do AppsManager je potřeba svůj projekt vytvořit a pojmenovat. Rovněž je potřeba specifikovat Redirect URI, jelikož je tento údaj nezbytný pro fungování tohoto procesu. Jedná se o adresu, kam FIT server přesměruje a přidá do parametru autorizační kód. AppsManager nabízí vývojáři možnost vybrat si konkrétní

```
@Module({
  imports: [
    TypeOrmModule.forFeature([
        Reservation,
        ReservationsRepository
    ]),
    UsersModule,
    DevicesModule,
    GuardModule,
    HttpModule,
  ],
  controllers: [ReservationsController],
  providers: [ReservationsService],
  exports: [ReservationsService],
})
```

Kód 4.6: Modul pro rezervace

API, o které má zájem. Bez zvláštního povolení jsou vývojáři přidělena zpravidla pouze čtecí práva. Aplikace Lend And Learn využívá Usermap API, které poskytuje identity uživatelů nebo jejich role. Autorizační server pro Lend And Learn generuje údaje, které jsou důležité pro proces přihlášení. Tyto údaje jsou:

- Cliend ID,
- Client Secret.

Client ID je unikátní řetězec, který je vytvořený na základě informací o Lend And Learn. Tento řetězec slouží k identifikaci aplikace. Client Secret je tajný údaj mezi autorizačním serverem (FIT server) a klientskou aplikací (LAL server).

Samotný proces začíná, když LAL klient přesměruje uživatele na FIT klienta, konkrétně na adresu s přihlašovacím formulářem. Adresa přesměrování je `https://auth.fit.cvut.cz/oauth/oauth/authorize` a navíc obsahuje tyto parametry:

- Cliend ID (již výše zmíněné),
- Redirect URI (již výše zmíněné),
- Response Type (požadovaný typ odpovědi),
- Scope (pravomoci).

Response Type je v tomto případě *code*. Poté, co uživatel zadá do formuláře své údaje, autorizační server tyto údaje ověří a zároveň identifikuje aplikaci pomocí výše zmíněných parametrů. Následně je LAL klient přesměrován na adresu, která je specifikována v parametru Redirect URI, a k této adrese přidá autorizační kód. LAL klient tento kód pošle LAL serveru.

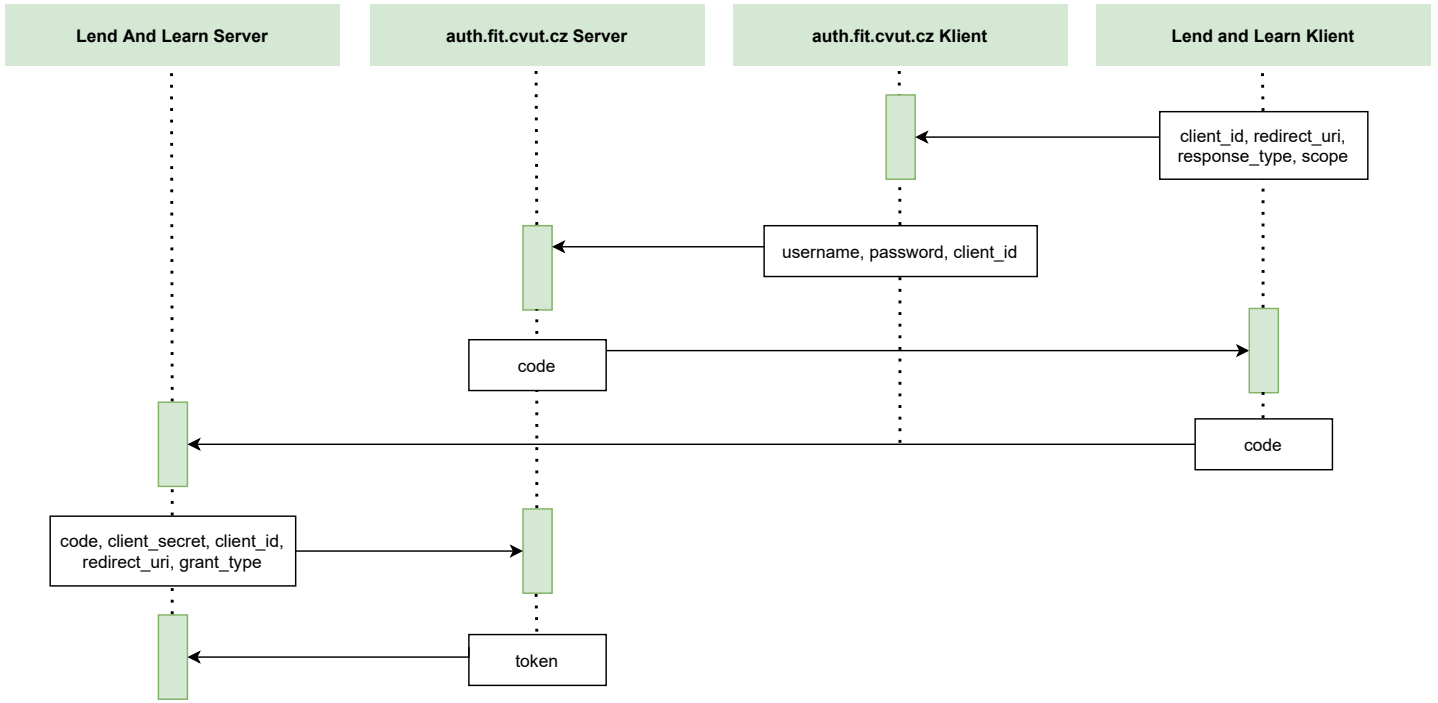
LAL server přijme autorizační kód a požádá FIT server o jeho výměnu za access token. Toto proběhne na základě poslání HTTP POST dotazu na adresu `https://auth.fit.cvut.cz/oauth/oauth/token` s parametry Client ID, Client Secret, Redirect URI, Code, Grant Type. Parametr Grant Type musí být *authorization_code*, protože se v dotazu posílá získaný autorizační kód. Pokud FIT server dotaz úspěšně validuje, vrátí v odpovědi access token. Tento proces je znázorněn na diagramu 4.3.

Dále je potřeba získat konkrétního uživatele a případně ho uložit do LAL databáze. Údaje o uživateli FIT server zpřístupní v odpovědi na dotaz na adrese `https://auth.fit.cvut.cz/oauth/oauth/check_token`, kam se pošle access token. Potom se zavolá funkce, která zkontroluje, zda záznam o uživateli již v LAL databázi existuje. Pokud takový záznam nenajde, tak ho vytvoří.

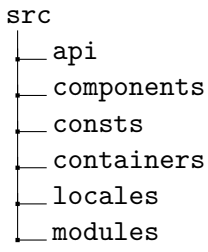
O správný průběh tohoto procesu se stará knihovna Passport.js, a to pomocí vytvořené tzv. strategie. Ukázka implementace této strategie je zobrazena v kódu 4.7. Konkrétně je zde vidět specifikace potřebných parametrů v konstruktoru strategie. O samotné volání API a posílání parametrů se stará knihovna, tudíž pro získání tokenu stačí tato implementace konstruktoru.

```
class FitStrategy extends PassportStrategy(Strategy, 'fit') {
  constructor(private http: HttpService) {
    super({
      authorizationURL:
        `https://auth.fit.cvut.cz/oauth/oauth/authorize?
        ${stringify({
          client_id: ENV.CLIENT_ID,
          redirect_uri: ENV.REDIRECT_URL,
          response_type: 'code',
      })}`,
      clientID: ENV.CLIENT_ID,
      clientSecret: ENV.CLIENT_SECRET,
      callbackURL: ENV.REDIRECT_URL,
      tokenURL: 'https://auth.fit.cvut.cz/oauth/oauth/token',
      scope: ENV.SCOPE,
    });
  }
}
```

Kód 4.7: PassportStrategy pro přihlašování pomocí FIT účtu



Obrázek 4.3: Proces získání tokenu



Obrázek 4.4: Struktura složky src na frontendu

4.2.8 Proces autentizace

Po úspěšném přihlášení backend očekává dotazy od uživatele. Každý dotaz je potřeba před provedením autentizovat. To znamená, že se musí ověřit, zda ten, kdo dotaz posílá, má práva na jeho provedení.

Lend And Learn implementuje tzv. token-based autentizaci. Access token, získaný v předchozím kroku přihlášení, je uchovávaný na Lend And Learn frontendu. Při každém dotazování se backendu se tento token přidá do hlavičky dotazu.

Toto je uskutečněno pomocí tzv. Guards, které NestJS poskytuje. Takový Guard obsahuje asynchronní funkci `canActivate()`, která se o ověření tokenu stará. Tato funkce vrací true, pokud uživatel má právo dotaz provést. V případě, že toto právo nemá, vrací false. V ukázce funkce 4.8 je vidět volání fakultního API, které podle tokenu vrací data o uživateli. Data se přidají do proměnné, do které má přístup i Controller, jenž tento Guard používá. Pokud je token nevalidní, protože například vypršel, vyhodí se výjimka, kterou funkce zachytává.

4.3 Implementace frontendu

Některé části frontendu, jako například stažené balíčky nebo Dockerfile, fungují na stejném principu jako na backendu. Proto jsou zde popsány pouze ty části projektu, které jsou od backendu odlišné. Je zde vysvětlen stav klientské části a jak probíhá jeho změna. Dále jsou popsány komponenty, které tvoří uživatelské rozhraní. Konkrétně je zmíněno, co mají tyto komponenty za úkol a jak vypadají. Jsou zde zahrnuty ukázky projektu.

4.3.1 Struktura projektu

Struktura frontendu se velice podobá struktuře backendu. Odlišný je obsah složky src, který je zobrazen na obrázku 4.4.

```
async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const accessToken = request.headers['x-api-key'];
    if (!accessToken) {
      throw new UnauthorizedException();
    }
    try {
      const { data } = await this.http
        .post(
          `https://auth.fit.cvut.cz/oauth/oauth/check_token
          ?token=${accessToken}`
          ,
          {
            headers: { Authorization: `Bearer ${accessToken}` },
          }
          ).toPromise();
      request.userName = data.user_name;
      request.accessToken = accessToken;
      return true;
    } catch (e) {
      throw new UnauthorizedException();
    }
  }
```

Kód 4.8: Funkce, která autentizuje dotazy

4.3.2 Modules

Ve složce **modules** se nachází logika klientské části aplikace, která je opět pro lepší organizaci rozdělená do více jednotlivých celků.

Modul **app** odpovídá za všeobecné výpočty napříč aplikací, například změnu jazyka. V modulu **auth** je uložena logika přihlašování a odhlašování do aplikace. Veškeré výpočty ohledně zařízení jsou obsaženy v modulu **devices**. Manipulace s rezervacemi je zahrnuta v modulu **reservations**. Výpočet statistik je v odpovídajícím modulu **statistics**. Poslední je modul **users**, kde se nachází výpočetní logika ohledně uživatelů. Typická struktura modulu je k nalezení na obrázku 4.5.

V souboru **actionsTypes.ts** se nachází konstanty, které specifikují typ vyvolávané akce. Tyto konstanty jsou v podobě řetězce a obecně se dělí na tři typy. První typ je ten, který značí žádost o provedení změny stavu aplikace. Další typ slouží k označení, že akce proběhla úspěšně. Poslední typ značí naopak neúspěch akce. V kódu 4.9 jsou vidět typy, které jsou použity pro

```
[název]
├── [název].actions.ts
├── [název].actionsTypes.ts
├── [název].reducer.ts
├── [název].saga.ts
├── [název].selectors.ts
└── [název].types.ts
```

Obrázek 4.5: Struktura modulu frontendu

```
const CREATE_RESERVATION_REQUEST = 'CREATE_RESERVATION_REQUEST';
const CREATE_RESERVATION_SUCCESS = 'CREATE_RESERVATION_SUCCESS';
const CREATE_RESERVATION_FAILURE = 'CREATE_RESERVATION_FAILURE';
```

Kód 4.9: Typy akcí

označení různých stavu vytvoření rezervace.

Soubor **actions.ts** obsahuje tzv. akce, které jsou vyvolány uživatelem. Tyto akce značí určitou změnu ve stavu aplikace. Přijímají parametry nutné ke změně. Vrácená hodnota je objekt, který tyto parametry zahrnuje, a to buď ve stejné podobě, ve které byly přijaty, nebo v upravené podobě na základě nějakých výpočtů. Tento objekt navíc nutně obsahuje klíč *type*, jehož hodnota je konstanta, popsaná v odstavci výše.

Veškeré akce jsou otypovány a tyto typy jsou definovány v souboru **types.ts**.

Reducer.ts je soubor reprezentující stav aplikace. Vytváří se v něm počáteční stav, který je neměnný. Počáteční stav je takový stav, ve kterém se aplikace nachází bezprostředně po jejím spuštění a není ovlivněn nastavením uživatele. Samotná aktualizace stavu aplikace probíhá ve funkci, která přijímá aktuální stav a také akci. Na základě typu akce se provede potřebná změna. Nicméně se neupraví původní stav, ale vznikne nový, a to tak, že se zkopíruje část původního stavu, která se nezměnila, a přidají se pozměněné části. Ve funkci Reducer se neprovádí žádné komplexní výpočty, volání API či jakékoliv jiné asynchronní funkce. Správné umístění těchto vyjmenovaných věcí je soubor **saga.ts**.

V souboru **saga.ts** se nachází Watcher Saga a Worker Saga (viz sekce 3.2.14). Watcher Saga naslouchá na typy jednotlivých akcí a podle toho zavolá konkrétní Worker Saga. Na obrázku 4.10 je příklad funkce Watcher Saga. Naslouchá na 2 typy, a to vytvoření zařízení a odstranění zařízení. Balíček *redux-saga/effects* poskytuje příkaz `takeEvery`, který v parametrech přijímá typ akce a konkrétní Worker Saga. Tento příkaz namapuje každou vyvolanou akci s tímto typem na příslušnou Worker Saga. Často se používá také

```
export default function* () {
  yield all([
    takeEvery(CREATE_DEVICE_REQUEST, createDeviceSaga),
    takeEvery(DELETE_DEVICE_REQUEST, deleteDeviceSaga),
  ]);
}
```

Kód 4.10: Watcher Saga

`takeLatest`. Ten v případě, že se akce se stejným typem zavolá vícekrát po sobě, namapuje pouze poslední akci na příslušnou Saga.

Tělo Worker Saga je obaleno do bloku try a catch. Je zvykem v této funkci volat API. Pokud se z backendu vrátí potřebná odpověď, zpropaguje se akce, která značí úspěch. Na jejím základě se aktualizuje stav aplikace. Pokud se naopak vyhodí výjimka, Saga ji odchytí a zavolá akci značící neúspěch.

Komponenty svůj výpis velice často řídí podle stavu aplikace. Proto k tomuto stavu potřebují mít přístup. Soubor **selectors.ts** obsahuje funkce, které ze stavu aplikace získají požadovanou položku. Tím se zároveň minimalizuje potřeba opakovaně implementovat stejnou funkci, která získá data ze stavu aplikace, v různých komponentách, právě když více komponent potřebují získat stejná data ze stavu aplikace. Stačí použít funkci, která je definována v tomto souboru.

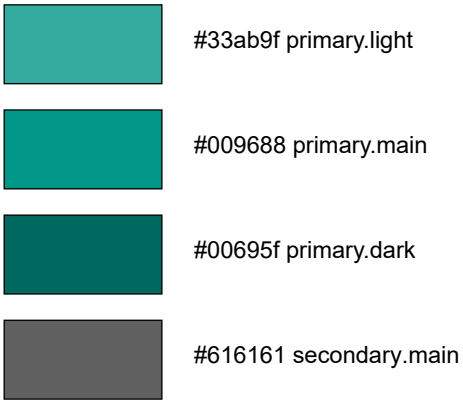
Celý proces změny stavu aplikace začíná, když uživatel vyvolá určitou akci, například kliknutím na tlačítko. Saga zachytí typ akce, provede případné volání API či jiné výpočty. Dále zpropaguje akci, kterou zachytí Reducer. Ten provede potřebnou změnu, která se projeví v komponentě. A takto se nový výsledek dostane k uživateli.

Při vývoji aplikace bylo použito rozšíření do webového prohlížeče Chrome React Developer Tools. To umožňuje sledovat promítnuté akce a změnu stavu aplikace, což usnadňuje její ladění.

4.3.3 Api

Ve složce **api** jsou uchovávány funkce, které posílají HTTP dotazy na backend. Soubory v této složce jsou rozděleny podobně jako moduly, konkrétně jeden soubor vždy obsahuje funkce, které se týkají stejného předmětu.

Pro každý HTTP dotaz je nastavený interceptor (viz sekce 3.2.12), který ze začátku získá ze stavu aplikace token a potom ho přidá do hlavičky dotazu. Pro každou odpověď, kterou dotaz vrátí, je rovněž nastavený interceptor. Ten kontroluje, zda nebyla vyhozena výjimka. Pokud je kód výjimky 401 (neúspěšná autentizace uživatele), aplikace uživatele odhlásí a zobrazí mu hlášku s důvodem odhlášení.



Obrázek 4.6: Paleta, kterou využívá frontend

4.3.4 Theme

Ve složce **consts** se kromě konstant nachází také theme. Theme shromažďuje styly, které jsou stejné pro celou aplikaci. Mít jasně definovaný Theme je základem dobrého uživatelského rozhraní, jelikož častý problém je například nekonzistence barev. Jedno tlačítko se může od druhého lišit barevně pouze o tón, a to je špatně. Dalším příkladem může být nekonzistence fontu či velikosti písma.

Material-UI poskytuje výchozí Theme, který se na frontendu Lend And Learn používá. Nicméně primární a sekundární barvy jsou přepsány. Tyto barvy jsou zobrazeny na obrázku 4.6, kde je vidět náhled barvy, její hexadecimální hodnota a název v projektu.

4.3.5 Components

Komponenta je obecně funkce, která uživateli zobrazuje výstup. V projektu jsou komponenty pouze nějaký malý celek, u kterých se předpokládá, že se použijí vícekrát. Pokud je to potřebné, jsou ve stejném souboru definované styly pro komponentu.

Komponenta **Alert** notifikuje uživatele o konkrétní události. Tato komponenta se může zobrazit ve třech variantách, a to success, warning, error, podle stavu akce, o které právě notifikuje. Zobrazí také doplňující text, aby uživateli bylo jasné, která akce byla provedena. Tato komponenta našla své časté využití v Redux Saga, kde se obvykle po úspěšné či neúspěšné akci zobrazí notifikace.

Komponenta **DataTable** zobrazuje data, která přijímá v parametrech, v tabulce. Díky této komponentě všechny tabulky v projektu jsou nastylované stejně. Liší se pouze jejich obsah. DataTable se stará pouze o výpis dat a jejich vzhled. Nestará se například o stránkování tabulky.

Další komponentou je **LangSwitch**. Jedná se o dvě tlačítka oddělená svislou viditelnou čárou, která přepínají mezi jazyky. První tlačítko je pro angličtinu, druhé pro češtinu. Kliknutím na jedno z nich se změna projeví okamžitě. Toto řešení by nebylo vhodné, pokud by možných překladů bylo více. Nebylo by to přehledné a mohlo by to zabrat značnou část obrazovky. Jedno z alternativních řešení je zobrazení celého seznamu možných jazyků pouze po nějaké akci. Ideální volbou je komponenta Select, kterou nabízí Material-UI.

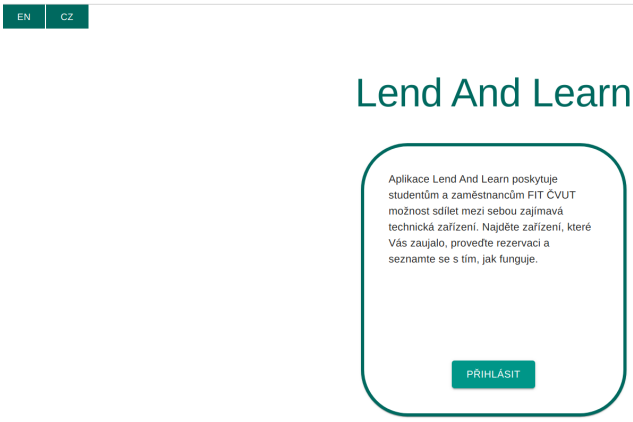
V projektu, kde jsou asynchronní operace, se musí čekat na jejich dokončení, a tím pádem i na zobrazení výstupu, který je na těchto operacích závislý. Velice častou situací je, když komponenta potřebuje více různých informací z backendu. Ať jsou na sobě závislé, či ne, jen málokdy odpověď dorazí ve stejný čas. V tom případě komponenta zobrazuje pouze část výstupu a postupně výstup aktualizuje na základě přijatých dat. Vhodnější pro uživatele je zobrazení dat najednou s podmínkou, že je mu naznačeno, že se některá data stále načítají. K indikaci čekání na asynchronní data je použita komponenta Suspense, kterou poskytuje React. Místo vykreslení komponenty, jež na data čeká, React vykreslí komponentu, která je specifikována v parametru *fallback* komponenty Suspense. V tomto parametru je vlastní komponenta **Loading**, která vrací komponentu CircularProgress z Material-UI s upravenými styly. CircularProgress je točící se kolečko, které je velice častým řešením pro indikaci načítání.

Komponenta **Sidebar** je určena pro navigaci. Zobrazuje se na levé straně obrazovky a jejím obsahem jsou odkazy na různá místa v projektu. Chování této komponenty se liší podle velikosti obrazovky. U větších velikostí má Sidebar pevné umístění. Pro obrazovky s velikostí méně než 600 px je tato komponenta ve výchozím stavu skrytá. Pro její zobrazení či skrytí je potřeba kliknout na příslušné tlačítko umístěné na horní liště. Při zobrazení Sidebar překrývá obsah stránky. Navíc je obsah stránky ztmavený, a to za účelem soustředit pozornost uživatele na Sidebar.

Projekt obsahuje horní lištu, která se zobrazuje napříč celým projektem, kromě úvodní obrazovky nepřihlášeného uživatele. Implementace této lišty je v komponentě **TopBar**. Vlevo je umístěná ikona značící Sidebar. Právě na tuto ikonu lze kliknout pro zobrazení či skrytí komponenty Sidebar v menších velikostech obrazovek. Uprostřed se nachází název projektu. Vpravo je umístěné uživatelské jméno přihlášeného uživatele. Má podobu tlačítka a kliknutím na něj se zobrazí menu s výběrem akcí. Akce jsou dvě, a to navigace na uživatelův profil a odhlášení se.

4.3.6 Containers

Existuje mnoho způsobů rozdělení komponent a kontejnerů. V projektu Lend And Learn se za kontejner považuje komponenta, která se skládá z více komponent. Zpravidla jeden kontejner je jeden pohled na projekt.



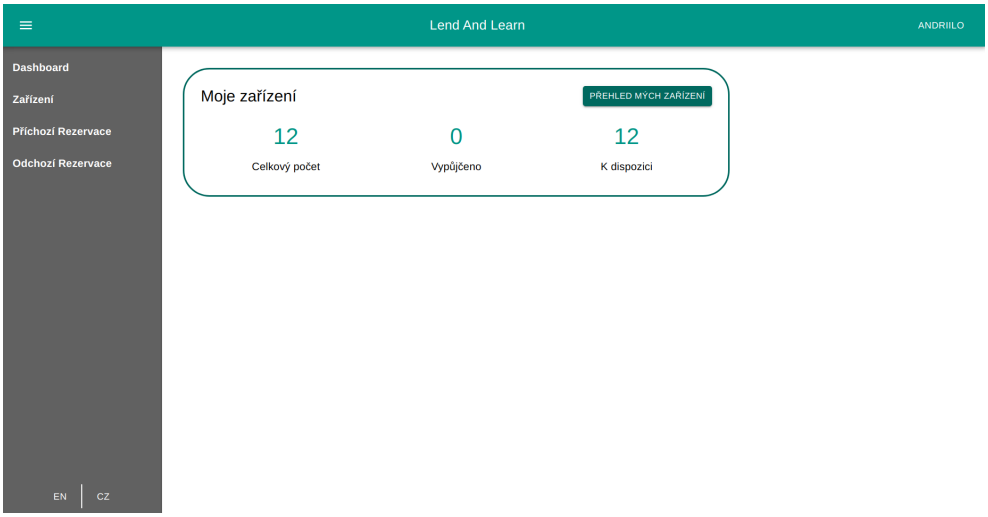
Obrázek 4.7: Úvodní obrazovka

K většině těchto pohledů se uživatel dostane nejen pomocí navigace, ale také pomocí URL adres. Takový pohled musí být předán v parametru komponentě Route. V projektu se rozlišují veřejné a privátní pohledy. Přístup k privátnímu pohledu má pouze přihlášený uživatel. Z tohoto důvodu byla vytvořená komponenta PrivateRoute, která zhodnotí, zda má uživatel právo přistoupit k požadované stránce. Pokud je tedy uživatel nepřihlášený, bude přesměrován na úvodní stránku.

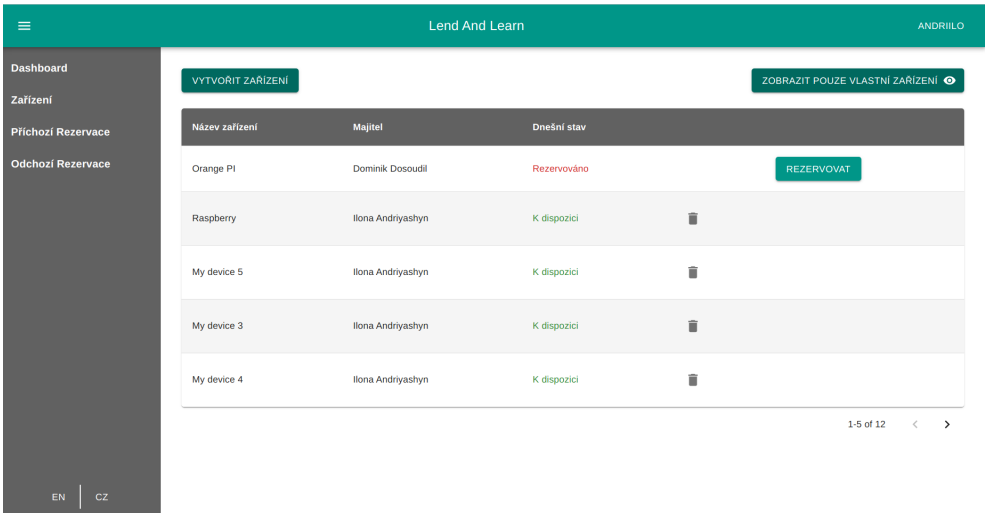
Náhled úvodní stránky je k nalezení na obrázku 4.7. Tento náhled je v češtině. Možnost přepínání mezi jazyky je zde umístěna na horní liště vlevo. Úvodní obrazovka obsahuje název projektu, jeho krátký popis a tlačítko k přihlášení se. Návrh této obrazovky byl inspirován fakultní aplikací Klasifikace, což může být pro uživatele přívětivé, jelikož Klasifikaci již znají. Může je to také více motivovat, aby Lend And Learn používali. Aplikace Klasifikace je k nalezení na adrese `https://grades.fit.cvut.cz/`.

Po přihlášení je uživatel přesměrován na stránku Dashboard. Jedná se o stránku, kde jsou umístěny statistiky. Statistiky, které mezi sebou souvisí, jsou obaleny do jednoho vizuálního bloku. Takový blok je zatím pouze jeden a přináší uživateli přehled o jeho zařízeních. Konkrétně uvádí počet zařízení, a to aktuálně vypůjčených, aktuálně k dispozici a celkový počet. Obsahuje také tlačítko, jež uživatele odkazuje na přehled zařízení. Ukázka obrazovky Dashboard je vidět na obrázku 4.8.

Dalším pohledem je stránka se zařízeními. Zařízení jsou vypsaná v tabulce. Uvádí se u nich základní údaje, jako název a majitel. Dále se uvádí jejich aktuální stav. Ten značí, zda je zařízení v tento den k dispozici, nebo je rezervované. Vlastní zařízení je možné přímo v tabulce odstranit. Cizí zařízení vedle sebe obsahuje tlačítko, přes které je možné provést rezervaci. Data v tabulce lze



Obrázek 4.8: Obrazovka Dashboard



Obrázek 4.9: Obrazovka se zařízeními

filtrovat tak, aby se zobrazila zařízení, která vlastní aktuálně přihlášený uživatel. Toto lze učinit pomocí příslušného tlačítka. Tabulka obsahuje stránkování, díky kterému se vypisuje pouze 5 zařízení. Při každé změně stránky se pošle dotaz na backend, kde jsou specifikovány potřebný počet zařízení a offset. Právě podle těchto parametrů backend vrátí v odpovědi potřebná zařízení. Toto řešení je efektivnější, jelikož není potřeba načítat a ukládat celý seznam zařízení, kde může být mnoho záznamů. Náhled této obrazovky je na obrázku 4.9.

Pro řadu akcí jsou v aplikaci implementovány modály. Modál se zobrazí po kliknutí na určitou akci a vždy obsahuje tlačítko na jeho zavření. Dále obsahuje

také tlačítko, které provede příslušný dotaz se vstupními daty od uživatele. Toto tlačítko může být podle nastavených podmínek deaktivované. Dále modál obsahuje políčka pro vstup od uživatele vždy s nějakým popiskem, aby bylo jasné, co a v jaké podobě se očekává. Přes modální okno probíhá například vytváření zařízení, jeho editace či rezervace.

V aplikaci je možné zobrazit si profil uživatele či profil zařízení. Účelem bylo, aby tyto profily vypadaly co nejvíce podobně. U obou je na stejném místě k nalezení ikona, která symbolizuje konkrétní profil. U obou profilů je stejně veliká. Tyto ikony poskytuje knihovna Material-UI. Oba profily jsou vidět na obrázku 4.10.

Profil zařízení z obrázku 4.10 je z pohledu uživatele, který si toto cizí zařízení prohlíží. Je mu k dispozici tlačítko s akcí rezervace zařízení. Dále na této stránce najde název zařízení, jeho popis a jméno majitele. Jméno majitele je rovněž tlačítko, jež odkazuje na jeho profil.

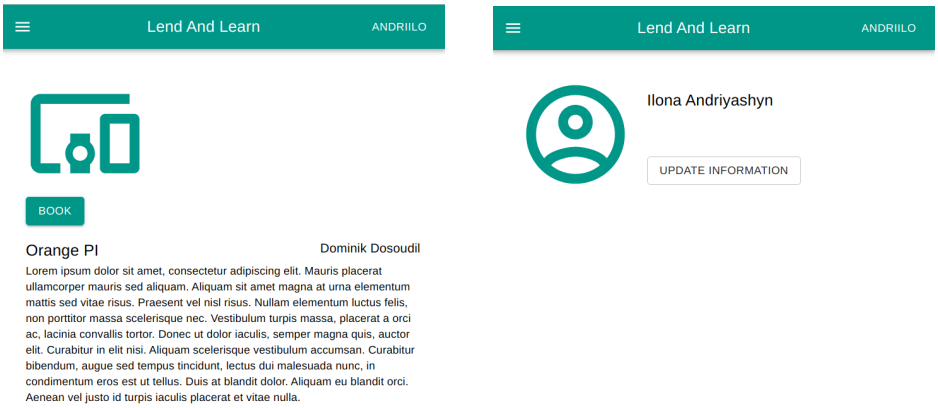
Profil uživatele, který je k nalezení na obrázku 4.10, je z pohledu přihlášeného uživatele, jenž se dívá na svůj vlastní profil. Vidí pouze své jméno a toto jméno může aktualizovat pomocí příslušného tlačítka. Tato akce získá jméno uživatele z fakultní databáze. Uživatel tedy nemá možnost změnit si jméno přímo přes aplikaci a nahradit ho například za nějakou přezdívku. Nicméně aktualizace dat řeší problém právní změny jména, například po sňatku. Při prohlížení jiného než vlastního profilu tato akce samozřejmě není dostupná. Nahradí ji akce poslání uživateli e-mailu. Kliknutím na takové tlačítko se otevře počítačový e-mailový klient. Tímto způsobem mohou uživatelé mezi sebou komunikovat.

Další dva pohledy se týkají rezervací. Uživatel si může prohlédnout své příchozí a odchozí rezervace.

Příchozí rezervace jsou z pohledu přihlášeného uživatele ty, které se týkají jeho zařízení. Tzn., že jiný uživatel má zájem si určité zařízení půjčit. Přihlášený uživatel je tedy správcem příchozích rezervací. Může je prohlížet ve dvou tabulkách podle jejich stavu. V první tabulce jsou zatím neschválené rezervace. U takových rezervací jsou dostupné dvě akce, a to schválení a odmítnutí. Po schválení se rezervace ihned přesune do druhé tabulky, kde jsou právě probíhající rezervace. U právě probíhajících rezervací je pouze jedna akce, a to ukončení rezervace.

Odchozí rezervace jsou ty, které vytvořil přihlášený uživatel. Tzn., že má zájem si půjčit něčí zařízení. Tyto rezervace jsou také rozdělené do dvou částí. Pokud je odchozí rezervace zatím neschválená, může ji přihlášený uživatel zrušit. Tabulka s výpisem právě probíhajících odchozích rezervací má pouze informativní účel, proto u ní nejsou dostupné žádné akce.

Výpis rezervací je v tabulkách se stránkováním. U všech rezervací se vypisuje jejich termín. Dále se vypisuje název zařízení, který odkazuje na jeho profil. V tabulce s příchozími rezervacemi se zobrazuje jméno uživatele, který o rezervaci žádá. U odchozích rezervací se vypisuje jméno uživatele, který vlastní zařízení. Jméno také odkazuje na profil uživatele.



(a) Profil zařízení (b) Profil uživatele

Obrázek 4.10: Obrazovky profilů

4.3.7 Locales

Překlady se nachází v PO souborech ve složce **locales**. Tento typ souborů doporučuje framework pro lokalizaci LinguiJS, který se v projektu používá. Tyto soubory jsou přehledné jak pro vývojáře, tak pro překladatele.

4.4 Dokumentace

Kvůli budoucímu vývoji projektu je potřeba mít řádně zdokumentované API. Pro generování API dokumentace existuje sada pravidel, která říkají, jak má taková dokumentace vypadat. Jedná se o OpenAPI Specification. V projektu se používá Swagger modul od NestJS, který poskytuje nástroje pro tvorbu dokumentace dodržující OpenApi specifikaci.

Dokumentace je přístupná na adrese `http://localhost:3000/api` a pro její zobrazení je potřeba mít spuštěnou aplikaci. Dokumentace se zobrazí pomocí uživatelského rozhraní Swagger UI a obsahuje název projektu, verzi a samotné API dotazy. U každého dotazu je zdokumentována HTTP metoda a URL adresa. Dále jsou popsány parametry, jejich typ a zda jsou povinné. Následuje odpověď, která obsahuje HTTP kód a případnou ukázku dat. Kromě definic API dotazů je k dispozici také popis schémat, jako například tabulek či DTO.

Kapitola 5

Testování

Tato kapitola popisuje způsob testování aplikace. Nejdříve je popsáno testování backendu pomocí automatických testů. Dále jsou zmíněny uživatelské testy.

5.1 Testování backendu

Backend je otestován pomocí testovacího frameworku Jest. Konkrétně je otestován každý Controller, Repository a Service. Soubory s testy obsahují koncovku spec.ts a jsou umístěny ve stejné složce jako soubory, ke kterým jsou tyto testy napsány.

Při testování jednotlivých komponent je potřeba vytvořit mock všech jejich závislostí, při kterém se nasimuluje jejich chování. U Service je obvykle potřeba vytvořit simulaci odpovídajícího Repository a jeho metod, a to jak vlastních, tak i metod knihovny. U některých metod knihovny je takový mock vytvořen vícekrát kvůli jejich přetěžování. Service odpovídá za výpočty, proto je otestován každý možný průběh metod.

Testování vlastních metod Repository vyžaduje mock databáze. K tomu je využita knihovna pg-mem (viz sekce 3.2.8). Pomocí ní jsou před každým testem vloženy záznamy do tabulek. Toto vložení probíhá buď před testy pomocí příkazu `beforeAll()`, nebo v jednotlivých blocích s testy, podle toho, zda stačí mít pro všechny testy stejné záznamy v databázi. Testy jsou zaměřeny na funkčnost metod a jejich schopnost pracovat s různými záznamy v databázi.

Testy pro Controller ověřují správnou funkčnost HTTP dotazů. Je zde ověřená validace vstupních dat, volání potřebných metod s výpočtem či vyhazování výjimek. Je potřeba mít vytvořený mock pro Guard, který ověřuje token, jelikož pro účely testování toto ověření není potřeba.

V ukázce 5.1 je zobrazena konfigurace testovacího modulu pro soubor DevicesController. Do providers je přidán mock DevicesService. Dále je zaměněn

```
const modRef = await Test.createTestingModule({
  imports: [],
  controllers: [DevicesController],
  providers: [
    {
      provide: DevicesService,
      useValue: devicesService,
    },
  ],
})
  .overrideGuard(ValidationGuard)
  .useValue(guard)
  .compile();
app = modRef.createNestApplication();
await app.init();
```

Kód 5.1: Konfigurace testovacího modulu pro DevicesController

Guard s validací na simulovaný Guard.

5.2 Uživatelské testy

Uživatelské testy dokážou vývojáři poskytnout velice dobrou zpětnou vazbu, protože tester nehodnotí pouze funkčnost aplikace, ale i zda je pohyb v aplikaci intuitivní a jednoduchý.

Při testování aplikace Lend And Learn byla zjištěna zcela zásadní chyba, a to nefunkčnost vytvoření rezervace. Po prozkoumání chování bylo zjištěno, že chyba spočívala ve špatné konverzi typů. Před vznikem této chyby byla na backendu povolena implicitní konverze typů. Tzn., že pokud proměnná, která je typů Date, přijme řetězec, který je v ISO formátu, tento řetězec se převede na typ Date. A toto chování způsobilo problém, jelikož u této proměnné byl dekorátor `@IsDateString()`, který ověřoval, že datum je řetězec. Proto dotaz vždy skončil s kódem 400, který značí špatný dotaz. Tato chyba byla odstraněna záměnou původního dekorátoru na `@IsDate()`.

Žádné další chyby nebyly zjištěny. Všechny požadavky byly splněny a celkový dojem testera z aplikace byl dobrý.

Závěr

Cílem této bakalářské práce bylo vytvořit webovou aplikaci pro vzájemné půjčování zařízení mezi studenty či zaměstnanci FIT ČVUT. Cíle byly plněny v rámci teoretické a praktické části této práce. Teoretická část si kladla za cíl rešerši podobných řešení a také vhodných technologií. Hlavním cílem praktické části byla implementace aplikace, jejíž ovládání mělo být intuitivní. Uživatelské rozhraní mělo být jednoduché a aplikace měla být dostupná ve dvou jazycích.

Na základě těchto cílů vznikla webová aplikace Lend And Learn, která je napsaná v programovacím jazyce TypeScript. Přihlásit se může každý, kdo vlastní FIT účet, čímž byl splněn požadavek na cílovou skupinu. K tomu byla využita knihovna Passport.js a bezpečnostní protokol OAuth 2.0. Aplikaci lze snadno nasadit, a to pomocí nástroje Docker.

Klientská část je implementována za pomocí knihovny React. Je plně responzivní a dostupná ve dvou jazycích, konkrétně v češtině a angličtině. Mohou ji tak využívat i studenti či zaměstnanci, kteří mluví pouze anglicky. Navíc jsou možné další lokalizace, stačí přidat soubor s překladem. Jednoduchosti aplikace přispívají validace, které obsahují formuláře, jelikož v ní usnadňují pohyb. Tyto validace jsou samozřejmostí i v serverové části, protože se nejedná jen o přehlednost uživatelského rozhraní, ale i o bezpečnost aplikace.

Práce byla rozdělena do kapitol, které reprezentují dílčí části projektu. Nejdříve byla provedena analýza, kde byly stanoveny funkční a nefunkční požadavky na aplikaci. Dále byla zhodnocena konkurence, kde bylo zjištěno, že aktuálně chybí vhodné řešení této problematiky, což poskytuje velký prostor pro aplikaci Lend And Learn. V další kapitole byl popsán návrh aplikace a vybrané technologie. Při výběru těchto technologií byl kladen důraz na jejich modernost a aktivní komunitu, protože správné zvolení technologií je klíčové pro projekt, který se plánuje do budoucna rozšiřovat. Jejich nesprávné zvolení totiž může v blízké budoucnosti vést k potřebě přechodu k jiným technologiím. Následovaly kapitoly, kde byl popsán způsob implementace a testování.

Možná rozšíření

Na základě této práce vznikl pouze funkční prototyp. Nabízí se rozšíření, která tuto aplikaci vylepší.

Užitečné mohou být například notifikace na uživatelův e-mail. Takové notifikace uživatel obdrží například při změně stavu jeho rezervace. Změnou stavu rezervace je myšleno její schválení či zrušení. Dále je může obdržet jako připomínku, že se blíží konec některé z rezervací.

V tomto stavu aplikace uživatelům mohou chybět recenze k zařízením či pouze nějaké poznámky ostatních uživatelů. Toto může být vyřešeno formou diskuze o konkrétním zařízení. Uživatel, který toto zařízení již měl vypůjčené, bude moci o své zkušenosti veřejně napsat.

Jiné rozšíření může být implementace chatu. Dosavadní komunikace mezi uživateli může být nepohodlná, jelikož se komunikuje přes e-mail. Uživatel si musí konkrétní e-mail v případě potřeby zpětně dohledat. Lepší by bylo mít tyto zprávy přímo v aplikaci.

Literatura

[1] Střálka, J.: Půjčím.to [online]. [cit. 2021-04-02]. Dostupné z: https://www.pujcim.to/

[2] GitHub: The 2020 State of the Octoverse [online]. 2020, [cit. 2021-02-11]. Dostupné z: https://octoverse.github.com/

[3] Wiegers, K.; Beatty, J.: *Software Requirements*. Developer Best Practices, Pearson Education, 2013, ISBN 9780735679627. Dostupné z: https://books.google.cz/books?id=nbpCAwAAQBAJ

[4] Reservio: Reservio [online]. [cit. 2021-04-02]. Dostupné z: https://www.reservio.com/cs/

[5] Reservio: Online rezervační kalendář [online]. [cit. 2021-04-02]. Dostupné z: https://www.reservio.com/cs/funkce/kalendar/

[6] Liew, Z.: Understanding And Using REST APIs [online]. Leden 2018, [cit. 2021-04-09]. Dostupné z: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/

[7] Mozilla and individual contributors: HTTP response status codes [online]. [cit. 2021-04-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[8] Google Developers: MVC Architecture [online]. Září 2012, [cit. 2021-04-17]. Dostupné z: https://developer.chrome.com/docs/apps/app_frameworks/

[9] Microsoft: TypeScript [online]. [cit. 2021-02-11]. Dostupné z: https://www.typescriptlang.org

[10] Oracle: Dynamic typing vs. static typing [online]. Březen 2015, [cit. 2021-02-11]. Dostupné z: https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html

[11] Bhatnagar, M.: Magic lies here - Statically vs Dynamically Typed Languages [online]. Září 2018, [cit. 2021-02-11]. Dostupné z: https://android.jlelse.eu/magic-lies-here-statically-typed-vs-dynamically-typed-languages-d151c7f95e2b

[12] Diep, W.: An Introduction To TypeScript: Static Typing and Interfaces [online]. Leden 2020, [cit. 2021-02-11]. Dostupné z: https://dev.to/wdiep10/an-introduction-to-typescript-static-typing-and-interfaces-241c

[13] OpenJS Foundation: About Node.js [online]. [cit. 2021-02-12]. Dostupné z: https://nodejs.org/en/about/

[14] GitHub, Inc: NPM [online]. [cit. 2021-04-08]. Dostupné z: https://www.npmjs.com/

[15] Mysliwiec, K.: NestJS documentation [online]. [cit. 2021-02-12]. Dostupné z: https://docs.nestjs.com/

[16] Mysliwiec, K.: Dependency injection [online]. [cit. 2021-02-12]. Dostupné z: https://docs.nestjs.com/providers#dependency-injection

[17] Google: Dependency injection in Angular [online]. [cit. 2021-02-12]. Dostupné z: https://angular.io/guide/dependency-injection

[18] Docker Inc.: Docker overview [online]. [cit. 2021-04-15]. Dostupné z: https://docs.docker.com/get-started/overview/

[19] Docker Inc.: What is a Container? [online]. [cit. 2021-04-15]. Dostupné z: https://www.docker.com/resources/what-container

[20] Fowler, M.: Continuous Integration [online]. Květen 2006, [cit. 2021-04-15]. Dostupné z: https://www.martinfowler.com/articles/continuousIntegration.html

[21] Fowler, M.: Software Delivery Guide [online]. Srpen 2019, [cit. 2021-04-15]. Dostupné z: https://www.martinfowler.com/delivery.html

[22] The PostgreSQL Global Development Group: About postgres [online]. [cit. 2021-04-08]. Dostupné z: https://www.postgresql.org/about/

[23] StackShare, Inc.: About postgres [online]. [cit. 2021-04-08]. Dostupné z: https://stackshare.io/postgresql

[24] Torres, A.; Galante, R.; Pimenta, M. S.; aj.: Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology*, ročník 82, 2017: s. 1–18, ISSN 0950-5849, doi:https://doi.org/10.1016/

j.infsof.2016.09.009. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0950584916301859

[25] Mysliwiec, K.: SQL (TypeORM) [online]. [cit. 2021-04-09]. Dostupné z: https://docs.nestjs.com/recipes/sql-typeorm

[26] Guimbal, O.: TypeORM [online]. [cit. 2021-04-08]. Dostupné z: https://typeorm.io

[27] Guimbal, O.: Active Record vs Data Mapper [online]. [cit. 2021-04-09]. Dostupné z: https://typeorm.io/#/active-record-data-mapper/what-is-the-active-record-pattern

[28] class-validator contributors: class-validator [online]. [cit. 2021-04-09]. Dostupné z: https://www.npmjs.com/package/class-validator

[29] Guimbal, O.: Pg-mem [online]. [cit. 2021-04-08]. Dostupné z: https://www.npmjs.com/package/pg-mem

[30] Hanson, J.: Passport documentation [online]. [cit. 2021-04-09]. Dostupné z: http://www.passportjs.org/

[31] Hardt, D.: The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, Říjen 2012. Dostupné z: http://www.rfc-editor.org/rfc/rfc6749.txt

[32] Facebook Inc.: React [online]. [cit. 2021-04-09]. Dostupné z: https://reactjs.org/

[33] Gackenheimer, C.: *Introduction to React*. Apress, 2015, ISBN 9781484212455. Dostupné z: https://books.google.cz/books?id=NZCKCgAAQBAJ

[34] Aderinokun, I.: What, exactly, is the DOM? [online]. Listopad 2018, [cit. 2021-04-09]. Dostupné z: https://bitsofco.de/what-exactly-is-the-dom/

[35] Matt Zabriskie, J. J. J. S.: Axios [online]. [cit. 2021-04-11]. Dostupné z: https://axios-http.com/

[36] Mozilla and individual contributors: Promise [online]. [cit. 2021-04-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[37] Mozilla and individual contributors: Using Promises [online]. [cit. 2021-04-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

[38] Dan Abramov and the Redux documentation authors: React Redux [online]. [cit. 2021-04-11]. Dostupné z: https://react-redux.js.org/

[39] Dan Abramov and the Redux documentation authors: Usage with TypeScript [online]. [cit. 2021-04-11]. Dostupné z: https://react-redux.js.org/using-react-redux/usage-with-typescript

[40] Redux-Saga: About Redux-Saga [online]. [cit. 2021-04-11]. Dostupné z: https://redux-saga.js.org/docs/About

[41] Mozilla and individual contributors: function* [online]. [cit. 2021-04-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function*

[42] Redux-Saga: Declarative Effects [online]. [cit. 2021-04-11]. Dostupné z: https://redux-saga.js.org/docs/Glossary

[43] Redux-Saga: Declarative Effects [online]. [cit. 2021-04-11]. Dostupné z: https://redux-saga.js.org/docs/basics/DeclarativeEffects

[44] Redux-Saga: Dispatching actions to the store [online]. [cit. 2021-04-11]. Dostupné z: https://redux-saga.js.org/docs/basics/DispatchingActions

[45] Tyl, J.: Co je biofeedback? [online]. [cit. 2021-05-11]. Dostupné z: http://www.biofeedback.cz/cs/co_je_biofeedback

[46] Christensson, P.: Framework Definition [online]. Březen 2013, [cit. 2021-05-08]. Dostupné z: https://techterms.com/definition/framework

[47] Christensson, P.: Middleware Definition [online]. Únor 2011, [cit. 2021-05-08]. Dostupné z: https://techterms.com/definition/middleware

Příloha A

Seznam použitých zkratek

API Application Programming Interface

ČVUT České vysoké učení technické

DOM Document Object Model

DTO Data transfer object

EEG Elektroencefalografie

FIT Fakulta informačních technologií

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

JSON JavaScript Object Notation

MVC Model View Controller

NFC Near Field Communication

NPM Node Package Manager

ORM Objektově relační mapování

PO Portable Object

REST Representational State Transfer

SQL Structured Query Language

UI User interface

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

Příloha B

Slovník použitých pojmů

access token Data, která se používají k identifikaci uživatele nebo aplikace a svým obsahem uživatele nebo aplikaci reprezentují. [31]

backend Serverová část aplikace.

EEG biofeedback Metoda, která monitoruje mozkové vlny. Na základě toho se člověk učí tyto vlny ovládat. [45]

framework Platforma, která poskytuje potřebné nástroje pro implementaci aplikace. [46]

frontend Klientská část aplikace.

middleware Vrstva, která propojuje různé softwarové komponenty. [47]

mock Simulace chování objektu či metody.

offset Číslo, které značí vzdálenost od počátečního prvku v poli.

open-source Produkt, jehož kód je volně přístupný. Kdokoliv může tento kód upravit, případně úpravu navrhnout.

tag Štítek, kterým se označí určitá věc. Takový štítek potom usnadňuje vyhledání věcí.

Příloha C

Obsah přiloženého CD

- `README.md` ..... stručný popis obsahu CD
- `exe` ..... soubory připravené pro produkci
  - `lend-and-learn-backend` ..... backend
  - `lend-and-learn-frontend` ..... frontend
- `src` ..... zdrojové kódy implementace
  - `lend-and-learn-backend` ..... backend
  - `lend-and-learn-frontend` ..... frontend
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF