VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM NA SPRÁVU ZAKÁZEK PRO TECHNIKY PLYNOVÝCH KOTLŮ

ORDER MANAGEMENT SYSTEM FOR GAS BOILER TECHNICIANS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM HOŠEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. PETR JOHN**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

147788

Ústav: Ústav informačních systémů (UIFS)
Student: **Hošek Adam**
Program: Informační technologie
Specializace: Informační technologie
Název: **Informační systém na správu zakázek pro techniky plynových kotlů**
Kategorie: Informační systémy
Akademický rok: 2022/23

Zadání:

1. Popište problematiku vytváření zakázek a jejich správu pomocí moderních technologií a způsoby jejich provozu. Konkrétně se zaměřte na vytváření zakázek pro servis plynových kotlů.
2. Prostudujte zásady návrhu moderních systémů, zaměřte se hlavně na způsoby návrhu pro mobilní zařízení.
3. Prostudujte technologie pro tvorbu webových uživatelských rozhraní a aplikačních rozhraní (např. nástroje React, Vue, Django, FastAPI apod.), způsoby jejich komunikace a vhodné databázové systémy.
4. Analyzujte požadavky na aplikaci, existující řešení, jejich nedostatky a dostupné aplikační rozhraní prodejců náhradních dílů pro plynové kotle (např. firma Viessmann).
5. Navrhněte řešení pomocí jedné nebo více aplikací a propojení s aplikačními rozhraní prodejců náhradních dílů.
6. Navržené řešení implementujte.
7. Finální řešení otestujte a vyhodnoťte výsledky.

Literatura:

- Wroblewski, L. Mobile First. A Book Apart, 2011. Book Apart. ISBN 978193755702.
- Marcotte, E. a Keith, J. Responsive Web Design. A Book Apart, 2011. Book Apart. ISBN 9780984442577.
- *React: Getting Started* [online]. [cit. 2022-09-14]. Dostupné z: https://reactjs.org/docs/getting-started.html
- *Angular: Introduction to the Angular Docs* [online]. [cit. 2022-09-14]. Dostupné z: https://angular.io/docs

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 5.

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

Vedoucí práce: **John Petr, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 17.5.2023
Datum schválení: 25.10.2022

Abstrakt

Tato práce řeší problematiku správy a organizaci zakázek pro servisní techniky plynových kotlů. Tento problém je řešen jako webová aplikace rozdělena na dvě části - aplikační logiku a uživatelské rozhraní komunikující pomocí REST API. Celá aplikace je následně zveřejněna k použití pomocí služeb Amazon AWS. Získáváme tak aplikaci ovladatelnou díky responzivnímu rozložení na osobním počítači i na mobilním telefonu, která umožňuje záznam dat odkudkoli. Servisní technik tím má ulehčenou práci při správě záznamů a následně usnadněnou tvorbu faktur i kvůli napojení aplikace na výrobce plynových kotlů a možnost načítání informací o náhradních dílech.

Abstract

This thesis deals with the management and organisation of contracts for gas boiler service technicians. This problem is solved as a web application divided into two parts - application logic and user interface communicating using REST API. The entire application is then published for use using Amazon AWS services. Thus, we get an application that is controllable thanks to its responsive layout on a personal computer and on a mobile phone, allowing data recording from anywhere. This makes the service technician's job of managing records easier and subsequently making invoicing easier, also because of the app's connection to the gas boiler manufacturer and the ability to retrieve spare parts information.

Klíčová slova

Informační systém, mobilní rozhraní, plynový kotel, správa zakázek

Keywords

Information system, mobile interface, gas boiler, order management

Citace

HOŠEK, Adam. *Informační systém na správu zakázek pro techniky plynových kotlů*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Petr John,

Informační systém na správu zakázek pro techniky plynových kotlů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Johna. Další informace mi poskytli autoři publikací a můj otec. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Adam Hošek
14. května 2023

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Petru Johnovi za odborné rady. Dále také své rodině a přítelkyni za podporu.

Obsah

Kapitola 1

Úvod

Nedílnou součástí servisu plynových kotlů je mimo vlastní údržbu, opravy a seřizování také správa informací o provedených úkonech. První věcí, po které nejspíše sáhneme, bude tužka a papír. Ovšem zde poměrně rychle narazíme na limity tohoto řešení. Papírový poznámkový blok může být snadno ztracen či poškozen při vykonávání servisní činnosti. Nabízí se tedy využít elektronický poznámkový blok či některý ze široké nabídky sofistikovaných systémů na správu zakázek. Při pohledu do praxe ovšem zjistíme, že ani jedna z těchto možností nebývá příliš oblíbena a využívána.

Aplikace k zaznamenávání poznámek sice nabízí synchronizaci dat napříč zařízeními a z toho plynoucí velkou dostupnost dat. Ovšem zároveň kladou vysoké nároky na technika tím, že jsou zcela neomezené ve způsobu zaznamenávání dat. Technik se tak musí starat o jistou strukturu dat, aby se v nich později byl schopen snadno orientovat. Výhoda v podobě synchronizace dat tímto bohužel často mizí. Obzvlášť v případě, kdy zvážíme nevýhody způsobené používáním těchto aplikací na mobilním telefonu, který je obvykle jediným dostupným zařízením. Výsledkem je potom u mnohých návrat zpět k papírovému vedení poznámek, jelikož elektronický poznámkový blok pro ně nemá tolik benefitů.

Další možností jsou již zmíněné systémy na správu zakázek, které jsou k dispozici. Máme na výběr z mnoha dostupných řešení. Ovšem většina z těchto řešení sdílí podobné vlastnosti: hůře optimalizované na mobilní zařízení, pomalejší a v bezplatném variantě poměrně omezené, jelikož mnohé mají limitován počet zakázek. Tato limitace ovšem většinou mizí v placených variantách těchto služeb. Ovšem fakt, že je třeba platit za takto základní věc, demotivuje technika používat jakoukoli z těchto služeb, jelikož mu nenabízí tak vysokou přidanou hodnotu, aby se mu do tohoto typu služeb vyplatilo investovat. Přidaná hodnota může spočívat například v automatickém načítání cen použitých náhradních dílů či jiných praktických funkcích.

Odtud plyne motivace k tvorbě informačního systému zaměřeného na práci servisních techniků plynových kotlů. Cílem této bakalářské práce je tedy vytvořit uživatelsky příjemný systém, snadno použitelný na mobilním telefonu, kde je cílem zadat co nejrychleji nutné informace o provedeném servisu a pokračovat k dalšímu zákazníkovi. Plánovaným řešením pro toto zadání jsou dvě aplikace: jedna zajišťující uživatelské prostředí a druhá spravující data a nabízející propojení k datům 3. stran - výrobců plynových kotlů.

Kapitola 2

Zakázkový informační systém

Většina servisních techniků dříve nebo později narazí na potřebu záznamu dat o svých zakázkách. Není možné si pamatovat veškeré detaily servisních prací, detaily o zákaznících, jako jsou například jejich kontaktní údaje a jejich specifické požadavky, což jsou informace nutné pro následnou tvorbu faktur. Tvorba zakázkových listů je tedy součástí práce každého z nich, ovšem přístupy k této problematice a použité technologie se často liší. Někteří volí poznámky v papírové formě, jiní elektronické poznámkové bloky či některou z dostupných aplikací ke správě zakázek.

Cíl je ale vždy stejný, a to zaznamenat spotřebovaný materiál, podrobnosti o servise, náklady spojené s cestováním na místo výkonu práce, odpracovaný čas a zákazníkovy kontaktní údaje. Část informací se může ovšem hodit i v budoucnu, kdy například potřebujeme zjistit historii servisu u zákazníka. V případě plynových kotlů může jít například o kontrolu složení spalin při jednotlivých prohlídkách, sledování výměn a čistění nejrůznějších částí kotlů a četnost kontrol například pro záruční opravy. Vyhledávání ve fyzických záznamech je sice možné, ovšem časově náročné a omezené na místo, kde máme archiv umístěn; elektronické záznamy toto řeší, ovšem bývají vykoupeny nepohodlím při zadávání, organizaci dat či vysokou cenou.

Jak by tedy měl vypadat ideální informační systém na správu zakázek? Systém by měl kombinovat vlastnosti obou možných přístupů: rychlost, uživatelskou přívětivost jako při psaní poznámek na papír a zároveň dostupnost kdekoli, dobrou organizaci dat a případně i propojení s dodavateli náhradních dílů pro usnadnění práce při jejich oceňování. V ideálním případě také dostupný pro rozumné množství dat zdarma - například fotografii, aby pro drobného živnostníka tento systém netvořil spíše potíže a finanční zátěž než reálné usnadnění práce.

2.1 Zakázka

Prvním slovem k objasnění z nadpisu této kapitoly je pojem *zakázka*, což je synonymum ke slovu objednávka [5]. Tato slova značí vztah mezi zadavatelem a dodavatelem, náplň práce a očekávaný výsledek. Takto totiž definuje pojem *veřejná zakázka* ustanovení § 7 odstavec 1 ZVZ [34], která je oproti klasické zakázce rozšířena o další povinné části, které pro nás nejsou zajímavé.

Zakázka tedy značí vztah dodavatele a zadavatele. Dodavatel může mít ovšem zakázek více, a tím pádem i více zadavatelů. Vzniká zde tedy potřeba zaznamenávat a třídit informace patřícím k různým zakázkám. Tyto informace je poté potřeba k následné tvorbě

faktur či sledování, zda plníme cenovou nabídku smluvenou se zadavatelem. Mezi typické informace, které potřebujeme sledovat, jsou identifikátory zadavatele či odpracovaný čas. Ostatní položky jako jízdné a materiál se poté liší vzhledem k místu a náplni práce.

V našem případě servisu plynových kotlů je potřeba u zakázek sledovat již zmíněné tradiční údaje: identifikaci objednavatele, čas strávený prací na zakázce, jízdné a materiál. Co se specifických věcí pro plynové kotle týče, jde o termíny absolvovaných revizí plynového spotřebiče, kdy se nejedná pouze o konkrétní spotřebič, ale i přívodní infrastrukturu; informace o měření spalin spotřebiče v případech, kdy je tento údaj podstatný. Problémem může být také mnoho specifických náhradních dílů od výrobců kotlů. Zde může být poté potíž v momentě tvorby podkladů pro fakturu, kdy každý díl má jinou koncovou cenu a je třeba je vyhledávat v databázi výrobce.

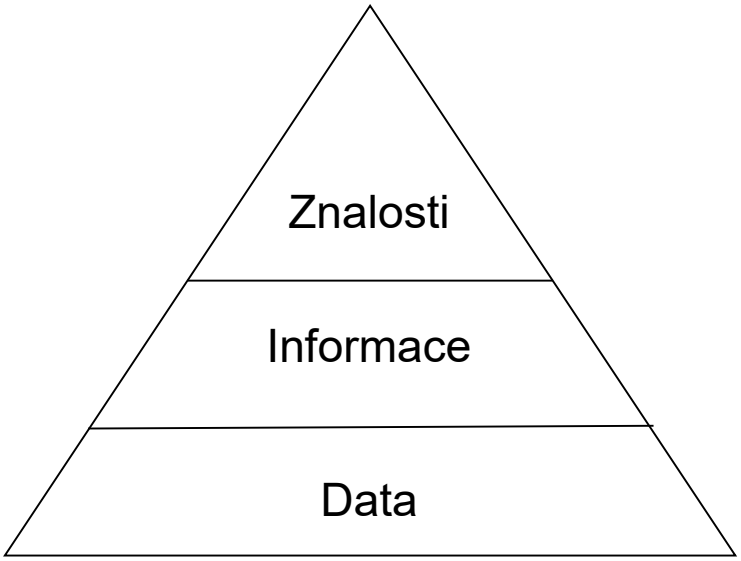
2.2 Informační systém

Abychom pochopili, co slovní spojení informační systémem na správu zakázek znamená, je třeba si nejdříve objasnit, co je informační systém. Vysvětlení tohoto spojení můžeme rozdělit na dvě části stejné jako slova, ze kterých je spojení utvořeno: informace a systém. Vysvětlení informace [20] je složitější, vzhledem k jeho použití napříč mnoha obory a odlišným způsobům jeho pojetí. Existuje spojení mezi slovy data, informace a znalosti [29], kdy nám pochopení tohoto vztahu pomůže s vysvětlením pojmu informace. Data zachycují objektivní fakta, jejich atributy a ostatní záležitosti z reálného světa. Data můžeme chápat jako základní stavební kameny pro informace. Data v informatice bývají typicky zaznamenána v databázích, které umožňují data strukturovat a zachytit vazby mezi nimi. Pokud tedy použijeme předchozí analogii, z dat vystavíme jednotlivé informace - dáme je do kontextu, propojíme je do srozumitelné podoby. Checklad a Scholes [21] vztah mezi daty, informací a znalostí popisují následovně: *Technologie pracují s daty, lidé je interpretují jako informace nesoucí význam, které se stávají podnětem pro další jednání. Proces informace je kognitivní záležitost, ve kterém stěžejní roli hrají znalosti* Ovšem tedy konkrétní znalost neboli subjektivní pochopení informací do celků závisí na konkrétním příjemci a také na reprezentaci jednotlivých informací, kdy neúplně podané informace mohou vést k nesprávným znalostem. K pochopení vztahu mezi těmito termíny nám může pomoci obrázek 2.1

Systém [28] značí spojení určitých věcí (ať hmotných, či nehmotných) do jistých celků, které můžeme následně zkoumat. Je vhodné, když námi zkoumanému systému nastavíme hranice, kde je pro nás ještě propojení daných záležitostí zajímavé.

Kombinací těchto dvou slov tedy získáváme informační systém [26], který používáme ke správě, zpracování a analýze dat určitého druhu. Díky informačnímu systému jsme poté schopni uchovávat dostupná data, efektivně s nimi pracovat (vyhledávat, zobrazovat námi chtěné informace) a vyvozovat závěry důležité pro naši práci, a ve výsledku nám ji usnadňuje [24].

Informační systém jako nástroj bývá realizován pomocí tradiční počítačové aplikace či webové stránky (případně jejich kombinací) spolupracující s informacemi uloženými k tomu určené aplikaci - databázi. Obě aplikace mohou být pouze lokální, tedy běžící pouze na jednom systému pro jednoho uživatele, což se ale v praxi moc nepoužívá, jelikož jsme limitování pouze jedním zařízením, a tedy může s informačním systémem pracovat pouze jeden uživatel. Typicky se spíše používá víceuživatelský přístup z více pracovních stanic, kdy více uživatelů pracuje nad stejnými daty. V dnešní době požadavku přístupu k datům odkudkoli na světě bývá tato záležitost realizována pomocí webové aplikace, případně kombinovanou s aplikací desktopovou, přistupující přes jisté rozhraní vzdáleně. Toto rozhraní i vlastní



Obrázek 2.1: Schéma vztahu data, informace, znalosti

databázi je ovšem nutné někde provozovat a zajistit ideálně maximální možnou dostupnost, aby jednotliví uživatelé zažívali minimum výpadků a mohli se nerušeně věnovat své práci.

Máme tedy v podstatě dvě možnosti: využít vlastní infrastrukturu, nebo si řešení objednat od třetí strany. Vlastní infrastruktura poskytuje výhodu v absolutní kontrole nad tím, co se s našimi daty děje, pokud tedy pracujeme s citlivými daty, pro která je nežádoucí, aby opustila naši společnost (ovšem tyto případy řeší například SLA[1], konkrétně třeba NDA[2]). Toto však klade vysoké požadavky na správce tohoto systému, který musí hlídat veškeré aspekty naší aplikace. Pokud správce adekvátně nezabezpečí celou infrastrukturu, stává se z této výhody spíše nevýhoda. Velká výhoda ve vlastním řešení ale přichází v momentě, kdy pracujeme s velkými balíky dat a jsme je tak schopni ukládat fyzicky uvnitř společnosti. Rychlost k jejich přístupu je poté mnohonásobně vyšší, než přistup přes síť internet, kdy jsme limitováni rychlostí připojení, které bude vždy pomalejší, než nejrychlejší dostupná technologie pro lokální síť. Ovšem v případě, kdy využijeme možnost provozování našich serverů v externích datacentrech poskytovatelů této služby, o benefit rychlého přenosu dat přicházíme. Provozování u externích dodavatelů eliminuje potíže spojené s výpadky elektrické energie či absencí připojení k internetu, tito poskytovatelé mívají přistup k těmto zdrojům zálohovaný.

2.3 Cloudová řešení

Řešení nazvané *cloud computing* neboli cloudová výpočetní síla ovšem řeší většinu potíží, které má využívání vlastní infrastruktury. Mezi její potíže patří již zmíněná vyšší cena, náročné zabezpečení, potřeba řešit zálohování dat, škálovatelnost, její dostupnost a mnohé další. *Cloud computing* většinu těchto nešvarů řeší [27], abychom byli schopni pochopit, jak ty které potíže tento přístup řeší, je potřeba vysvětlit, co *cloud computing* vlastně znamená a z tohoto konceptu plynoucí konkrétní výhody.

[1] Smlouva mezi poskytovatelem a dodavatelem služby
[2] Dohoda o mlčenlivosti mezi poskytovatelem a dodavatelem služby

Cloud computing nebo také *serverless computing* je vzdálené spouštění provozované aplikace zcela zbavené návaznosti na hardware, jelikož je zde použita vysoká míra abstrakce, kdy jednotlivé aplikace běží ve virtualizovaných strojích, nebo pouze ve formě jistých požadavků napojených pomocí rozhraní na infrastrukturu poskytovatele služby. Nejsou tím pádem vázány na výkon jediného stroje, jejich výkon lze libovolně škálovat a využívat tak výkon mnoha strojů či naopak pouze zlomku jediného stroje [19]. Na jednom serveru mohou provozovatelé těchto služeb provozovat více klientských aplikací, kdy v případě zvýšené zátěže můžeme ve zlomcích sekund využít výkon celého serveru či připojit další s tím, že ostatní aplikace mohou být přesunuty na jiné servery. Tímto sdílením je možno snížit náklady na běh, jelikož zdroje jsou využívány efektivněji a tyto služby tak mohou být levnější v porovnání s vlastní infrastrukturou, kde platíme i okamžiky, kdy server není zatížen. Platby zde pak většinou probíhají za využitý procesorový čas nebo požadavek na infrastrukturu. Sdílením zdrojů je také zvýšená eliminace hardwarových závad, kdy výpadek jednoho serveru nás jako zákazníka nijak neohrozí a jsme schopni dále pracovat na jiném serveru. Tato změna je pro nás ale zcela neznatelná a naši práci nijak nenaruší.

Poskytovatelů těchto služeb existuje hned několik. Liší se často cenou, podporou programovacích jazyků a například také bezplatnou nabídkou, kterou může využít zdarma kdokoli za daných podmínek. Dalším rozdílem je také implementace rozhraní, kdy každý poskytovatel používá svoje vlastní. Rozhraní slouží k vyvolání nejrůznějších akcí podporovaných poskytovatelem a jím i optimalizovaných pro jeho vlastní infrastrukturu, čímž lze zajistit vyšší výkon. Mnohdy je také použití tohoto rozhraní jedinou cestou, jak realizovat náš cíl. Vzhledem k nekompatibilitě těchto rozhraní je potřeba si také dávat pozor na limitace přenosnosti těchto aplikací, kdy aplikace využívající rozhraní jednoho poskytovatele nebude schopna fungovat u jiného dodavatele této služby.

Mezi největší poskytovatele těchto služeb patří [32] Microsoft[3] se svým cloudem Azure, Google[4] provozující službu nazvanou jednoduše Cloud, Amazon[5] nabízející službu Lambda a jako zástupce menších méně známých poskytovatelů *cloud computingu* příklad CloudFlare[6] a jejich Workers, který je ovšem jinak ve světě internetových poskytovatelů známý svou ochranou proti *DDoS*[7] (Distributed Denial of Service - zahlcení služby falešnými požadavky a znemožnění přístupu legitimním uživatelům). Každý z nich také nabízí množství doplňkových služeb pro databázi, zabezpečení a mnoho dalších funkcí plně propojených s jejich nabízenou infrastrukturou, jejich přehled je obsažen v tabulce 2.1 Nejzajímavější částí k porovnání a následnému výběru jsou nabízené služby zdarma a poté cena za případné rozšíření na vyšší variantu pro případ, kdy naše služba narazí na limity poskytované zdarma.

Pro porovnání cenových nabídek můžeme vytvořit modelový příklad, kdy budeme počítat například s 4 500 000 požadavků měsíčně a pro uložení dat budeme potřebovat 300 GB úložiště. Pro databázi využijeme prostor 20 GB. Tento modelový příklad následně zadáme do kalkulaček odhadovaných měsíčních nákladů u jednotlivých poskytovatelů a zjistíme odhadované náklady. Bohužel firma Cloudflare tuto možnost nenabízí, tudíž v modelovém cenovém porovnání nebude figurovat. Díky tomuto porovnání získáme představu o případných nákladech na tyto služby v případě, kdy narazíme na hranice bezplatných plánů. Tyto informace jsou znázorněny v tabulce 2.2

[3] Nabídka Microsoft https://azure.microsoft.com/en-us/pricing/free-services/
[4] Nabídka Google https://cloud.google.com/free/docs/free-cloud-features#free-tier
[5] Nabídka Amazon https://aws.amazon.com/free/
[6] Nabídka CloudFlare https://developers.cloudflare.com/workers/platform/pricing/
[7] CoudFlare ochrana proti DDoS útokům https://www.cloudflare.com/ddos/

	Microsoft	Google	Amazon	Cloudflare
Název výpočetní síly	Functions	Run	Lambda	Workers
Požadavky měsíčně	1 000 0000	2 000 0000	1 000 000	100 000 / den
Název úložiště	Cloud Shell	Storage	S3	R2
Kapacita úložiště	5 GB	5 GB	5 GB	10 GB
Název databáze	Cosmos DB	Firestore	DynamoDB	D1
Kapacita databáze	25 GB	1 GB / projekt	25 GB	100 MB

Tabulka 2.1: Bezplatná nabídka pro nás klíčových služeb

	Microsoft	Google	Amazon
Cena / měsíc v USD	41,41	18,72	47,51

Tabulka 2.2: Kalkulace modelového případu

Cílem bylo získat co nejobjektivněji porovnatelné cenové údaje. Ovšem vzhledem k množství specifik jednotlivých poskytovatelů, jako jsou například různě interpretované jednotlivé operace, jejich podsložky či slevy spojených s platbou předem, není zcela možné zajistit objektivní porovnání pro tento modelový příklad. Slouží tedy spíše k přibližné orientaci, následné konkrétní nacenění je třeba provést s přesnými daty z provozu.

Kapitola 3

Návrh informačního systému

Při návrhu moderního informačního systému je potřeba zjistit, ze kterých zařízení jej nejpravděpodobněji budeme ovládat. Dle toho následně volit přístupy k vývoji, použité technologie a určit hlavní situace, kterým uživatel bude pravděpodobně čelit. S tímto nám mohou pomoci každoroční statistiky přístupu z jednotlivých typů zařízení u vysoce navštěvovaných webových stránek. Jedním z poskytovatelů těchto statistik je společnost Statcounter[1]. Tato společnost každoročně zveřejňuje statistiky využití podle prohlížečů, typů operačních systémů, rozlišení a několika dalších ukazatelů. Pro náš případ je ale nejzajímavější rozdělení podle použití mobilního telefonu, tabletu či počítače k přístupu na internet.

Na obrázku 3.1 můžeme vidět data využití zařízení k přístupu k internetu v Evropě za roční období od listopadu 2021 do listopadu 2022 v Evropě. Jak můžeme vyčíst z poskytnutého grafu, použití dotykových tabletů se drží lehce nad 2 %. Použití mobilních telefonů a počítačů je dnes ovšem víceméně podobné (49 % oproti 48 %). Můžeme tedy z tohoto globálního pohledu vyvodit, že bude nejspíše nutné se zaměřit na oba typy zařízení. Situace se může lišit případ od případu, kdy například jisté aplikace vyžadují přístup spíše z počítače či mobilního telefonu, a to hlavně z důvodu rozdílných ovládacích zařízení - rozdíl mezi myší s klávesnicí a dotykovým displejem, pomocí kterého v dnešní době ovládáme mobilní telefony. Komunikátory od výrobce Blackberry nebo například zařízení s operačním systémem Windows Mobile, které k ovládání disponovaly, mimo mnohdy ani neosazeného dotykového displeje, hlavně joystickem k procházení ovládacích položek a plnohodnotnou klávesnicí typu *QWERTY* k pohodlnému psaní, dnes již nejsou příliš používané. Při pohledu do statistik používanosti telefonů Blackberry[2] či již mnoho let nepodporovaného Windows Mobile 6[3], který disponoval zmíněnými ovládacími prvky, vidíme, že tyto prvky jsou dnes již velice výjimečné. Jeho následovník Windows Phone 7 a další již tímto přístupem nedisponovaly a zaměřily se primárně na dotykové displeje. I ze statistik využití operačních systémů nejen na telefonech[4] můžeme vidět, že na mobilních telefonech mají absolutní převahu systémy zaměřené na dotykové ovládání.

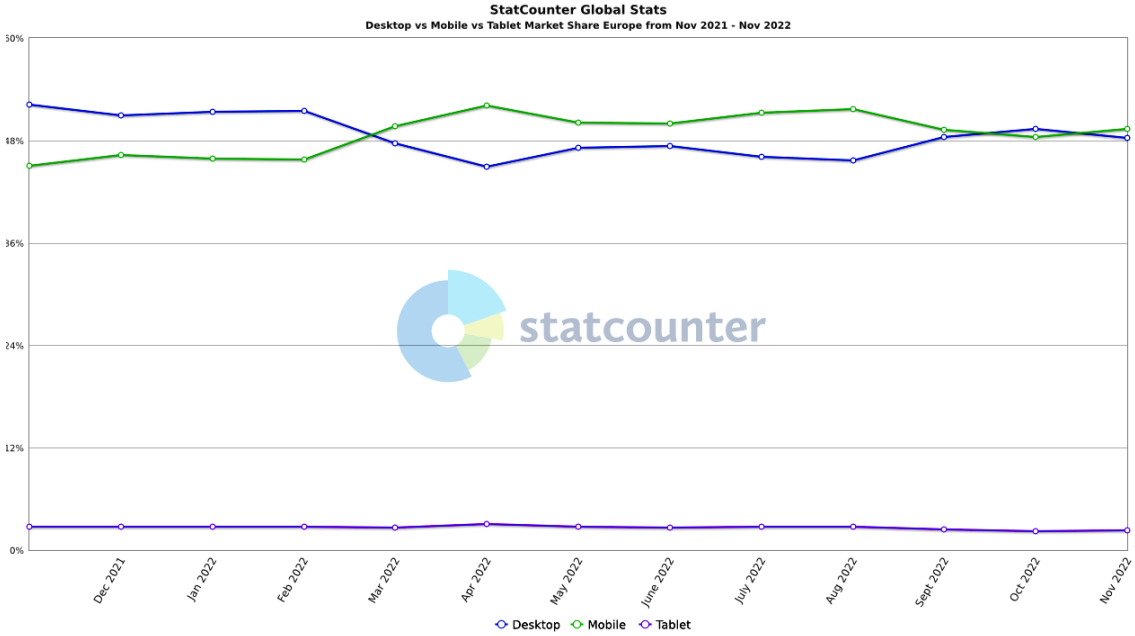
Na základě těchto informací můžeme predikovat pravděpodobné využití našeho systému. Tato očekávání podpoří i zkušenost z praxe, kdy sledováním servisních techniků při práci dospějeme k závěru, že je část práce vykonávána spíše z mobilního telefonu, jelikož je často

[1] Webové stránky společnosti Statcounter https://gs.statcounter.com/

[2] Počet telefonů Blackberry ve spojených státech https://www.statista.com/statistics/232792/forecast-of-rim-users-in-the-us/

[3] Životní cyklus Windows Phone 6 https://learn.microsoft.com/en-us/lifecycle/products/windows-mobile-6

[4] Využití operačních systémů https://gs.statcounter.com/os-market-share



Obrázek 3.1: Zastoupení počítač vs mobilní telefon vs tablet, převzato z [16] .

snadno dosažitelný. Naopak počítač bývá využíván při práci v kanceláři či jinde, kde to situace umožňuje. Je tedy třeba mít aplikaci optimalizovanou na obě platformy a umožnit tak komfortní ovládání aplikace pokud možno z obou typů zařízení.

Potřebujeme tedy dvě rozhraní, jedno zaměřené na dotykové ovládaní a druhé na myš s klávesnicí. Dále potřebujeme vyřešit situaci, kdy každé zařízení může používat jiný operační systém. Varianta, kdy vytvoříme nativní aplikaci pro každý typ zvlášť, zní dobře z pohledu uživatelského zážitku, protože jsme schopni aplikaci plně integrovat do systému. Nejen funkčně, ale i vizuálně, můžeme využít také případných specifických možností daného systému. V tento moment bude uživatelský zážitek na vysoké úrovni. Tento přístup je tedy ideální pro uživatele, ovšem velmi nákladný na vývoj a následnou údržbu, kdy je třeba udržovat a ideálně i modernizovat aplikaci na všech platformách. Platforem, které bývají typicky použity pro přístup k internetu, je několik. Můžeme si opět pomoci daty od společnosti Statcounter[5] a zjistíme, že systémy s podílem nad 5 % jsou čtyři. Pro nás by to tedy znamenalo udržovat minimálně čtyři nativní aplikace.

Další variantou je vyvinout webovou stránku, která bude dobře ovladatelná jak ze zařízení s klávesnici a myší, tak ze zařízení vybaveného dotykovou obrazovkou. Rozhraní, které se přizpůsobí velikosti displeje, a dle toho zvolí vhodné rozložení stránky. Tento přístup k tvorbě webové stránky se nazývá *responzivní vzhled*. Takový způsob tvorby nám poskytne výhodu v podobě nižších nákladů za současného pokrytí všech platforem, které disponují webovým prohlížečem a výkonem pro přístup k moderním webovým stránkám. Aplikaci je potom zároveň možné zabalit do balíčku a distribuovat ji na systémech jako iOS[6] a Android[7], kdy se webová aplikace tváří jako aplikace nativní. Na ostatních systémech, kde tato

[5] Využití operačních systémů https://gs.statcounter.com/os-market-share

[6] Zabalení webu pro iOS https://developer.apple.com/library/archive/documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html

[7] Zabalení webu pro Android https://developer.android.com/develop/ui/views/layout/webapps

možnost není, pro přístup postačí moderní webový prohlížeč. Z těchto důvodů se v dalších částech této práce zaměřených na technické řešení, budeme zabývat vývojem uživatelského rozhraní v podobě webové stránky. Ovšem vhodné přístupy k tvorbě dobře ovladatelného uživatelského rozhraní jsou podobné a rozdíl bude až v podobě použitých technologiích.

3.1 Vývoj rozhraní pro počítač

Byť by naší prioritou měl být takzvaný přístup *Mobile-first design*, což znamená zaměření se na mobilní zařízení, vzhledem ke zjištěním o systémech použitých k přístupu na internet nesmíme opomíjet návrh aplikace pro osobní počítače, které, jak jsme mohli vidět, stále tvoří nemalý podíl. Jak popisuje Ping Zhang a Gisela M. von Dran v publikaci [33], hlavním cílem je vytvořit aplikaci, která bude lidskému oku líbivá a uživatel se s ní naučí rychle a efektivně pracovat. Zároveň aplikace, která se bude uživateli líbit a dobře ovládat, pravděpodobně bude uživatelem doporučována ostatním, a tím naše aplikace získá na popularitě.

V minulosti byl přistup k webovému designu omezený dostupnými technologiemi pro vývoj. Také bylo omezenější množství zařízení, ze kterých se k webu přistupovalo - typicky pouze z osobních počítačů, jiné zařízení v tě době v podstatě neexistovaly. Ovládací prvky tedy byly malé, což dostačovalo pro ovládání myší. Výkon počítačů byl nižší než dnes, a tudíž nebyl prostor na pokročilé vizuální prvky a obsah stránek se víceméně zredukoval na formátovaný text a odkazy případně organizované v tabulce, později rozšířené o nejrůznější grafické prvky. Pro představu nám pomůže dokument [23], ve kterém najdeme souhrn vzhledů webových stránek z počátku tisíciletí a jejich typické prvky. Můžeme si také povšimnout faktu, že weby byly optimalizované na určité rozlišení a v případě většího či menšího rozlišení neposkytovaly žádné přizpůsobení obsahu vzhledem k možnostem obrazovky.

Postupem času ovšem výkon osobních počítačů narůstal a s ním i zobrazované rozlišení, zároveň také narůstal požadavek uživatelů na hezčí a snadněji ovladatelné rozhraní. To se v raných dobách řešilo pomocí různých skriptů [23], později došlo k velkému rozmachu technologie Adobe Flash[8]. Technologie ve své době hodně rozšířená napříč internetem ovšem pro její absolutní nekompatibilitu s nyní rozšířeným systémem iOS[9] a postupem let zhoršující se kompatibilitu s dalším mobilním operačním systémem Android[10]. Tato technologie tudíž byla odsouzena k zániku i vzhledem k vysoké výpočetní náročnosti, bezpečnostním problémům a dalším nevýhodám, které oproti jiným moderním přístupům Adobe Flash má[11]. I přesto tato technologie byla velmi populární mezi lety 2000 a 2010, avšak zmíněné problémy vedly k postupnému úpadku této technologie[12]. Tuto technologii totiž vytlačoval postupný nástup nejrůznějších nadstaveb nad skriptovacím jazykem hojně využívaným na webu JavaScriptem a HTML umožňující mnoho věcí, které dřív byly dosažitelné jen pomocí Adobe Flash či ostatních méně používaných možností.

Použití nadstaveb založených nad jazykem JavaScript[13], což je jazyk, který se používá k tvorbě dynamických prvků webu a je plně interpretovaný na straně uživatele v jeho prohlížeči, nadále usnadňuje vývoj webových aplikací. Později navíc také standardizace[14]

[8] Statistika použití Adobe Flash https://www.statista.com/chart/3796/websites-using-flash/

[9] Adobe Flash a systém iOS https://community.adobe.com/t5/flash-player-discussions/installing-flash-player-on-iphone/td-p/9744809

[10] Adobe Flash a systém Android https://community.adobe.com/t5/flash-player-discussions/flash-player-for-android-phones/td-p/9954925

[11] Porovnání HTML5 a Flash https://www.softwaretestinghelp.com/html5-vs-flash/

[12] Článek o úpadku https://www.kaspersky.com/blog/life-and-death-of-adobe-flash/45906/

[13] Javascript https://developer.mozilla.org/en-US/docs/Web/JavaScript

[14] Standard HTML https://html.spec.whatwg.org/

značkovacího jazyka HTML[15] a aplikačního rozhraní JavaScriptu v podobě HTML5[16]. Tyto skutečnosti napomohly rozšíření využití moderních otevřených přístupů k tvorbě webu a urychlilo eliminaci Adobe Flash. Tento fakt zpříjemnil přístup k webu z ostatních zařízení, nejen z osobních počítačů.

Ovšem dobré zvyky pro tvorbu webových rozhraní jsou známy nehledě na použitou technologii a měli bychom na ně myslet vždy při tvorbě. Této problematice se věnují autoři v knize *The Principles of Beautiful Web Design* [18]. Důraz je kladen především na přehledné rozložení jednotlivých prvků, kombinace barev a nejrůznější grafické prvky pro zpřehlednění obsahu. Významnou roli hraje také typografie a volba obrázků a ilustrací, které napomáhají s konzumací obsahu a upoutávají uživatelovu pozornost. Preference se časem mění, proto je ideální sledovat aktuální trendy webdesignu, jelikož se jedná o velmi progresivní oblast.

Aktuální trendy sleduje například Ellysa Coultas ve svém článku [2] zaměřeném na rok 2023. Populární prvek *paralaxní* posunování, kdy při posunu webové stránky se mění animovaně její obsah (například vyjíždí text ze strany) se ovšem hodí spíše na propagační stránky a ve vlastním informačním systému na něj prostor moc není. Na stránce o představení systému ovšem může jít o zajímavý prvek. Druhou položkou v pořadí je rychlost načítaní[17], kdy se každá ušetřená vteřina počítá a má velký vliv na uživatelský zážitek. Doba načtení stránky by rozhodně měla být jednou z prioritních věcí informačního systému, s čímž zároveň souvisí i chytré načítání obsahu podle požadavků. Načítáme tedy jen části, které jsou potřeba. K organizaci zobrazovaných dat je vhodné použít takzvaný *grid systém*, což si můžeme představit jako sofistikovanou tabulku s funkcemi navíc, ve kterých můžeme vykreslovat data. Trendem, který přetrvává z minulých let, je *Mobile-First* přístup, kdy je kladen důraz na vývoj pro telefony, čemuž se budeme věnovat v následující části.

3.2 Vývoj rozhraní pro mobilní zařízení

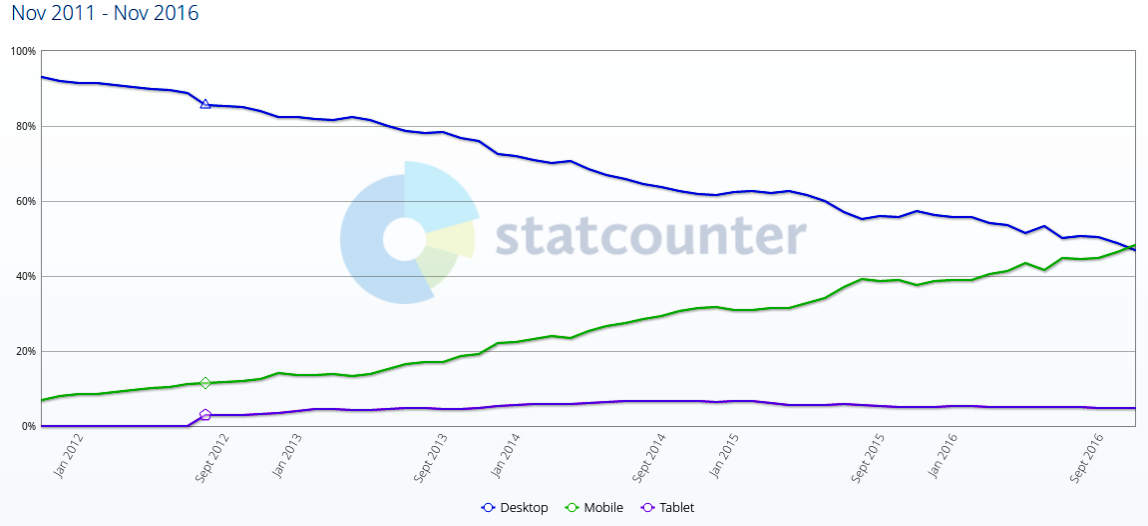
Vytěsnění Adobe Flash a ostatních technologii, které byly pro mobilní zařízení výpočetně náročné, a jejich nahrazení HTML5 zpříjemnilo používání webových stránek na chytrých mobilních telefonech, což vyústilo ve stále zvyšování podílu mobilních zařízení. Na obrázku 3.2 můžeme vidět, jak se od představení HTML5 v roce 2014 podíl mobilních zařízení přehoupl přes 20 % a od té doby dorovnal osobní počítače, což lze vidět na obrázku 3.1. Responzivní vývoj je tedy prioritou, na kterou je potřeba se zaměřit, a to hlavně pro mobilní zařízení, tento přístup se nazývá *Mobile-First*.

Přístupu *Mobile-First* se hojně věnuje publikace od Luke Wroblewskeho [30], ve které popisuje, jak správně navrhovat webové aplikace zaměřené na mobilní telefony. Nejdříve je tedy potřeba si ujasnit, jaké omezení mobilní telefon vlastně má v porovnání s osobním počítačem.

Vykreslované rozlišení na mobilních telefonech je nižší a to i v případech, kdy displej má rozlišení vysoké - v tomto případě se body navíc použijí k ostřejšímu vykreslení obsahu, a ne ke zvětšení plochy. S nižším rozlišením je tedy třeba počítat a je třeba promýšlet, co uživatel na telefonu potřebuje, a co je obsah zbytný. Uživatel určitě raději ocení dobře přístupné ovládací prvky než nicneříkající reklamní prohlášení. Na telefonu se musíme zaměřit na maximální efektivitu ovládání.

Efektivní bychom ale neměli být jen v případě ovládání, ale i v případě výkonu. Moderní mobilní telefony již mívají výkonu mnoho, ovšem i tak se mohou vyskytnout modely

[15] HTML https://developer.mozilla.org/en-US/docs/Web/HTML
[16] HTML5 https://developer.mozilla.org/en-US/docs/Glossary/HTML5
[17] Dokument o rychlosti načítání https://developers.google.com/speed/docs/insights/v5/about



Obrázek 3.2: Vývoj zastoupení mobilních telefonů, převzato z [1] .

s výkonem nižším. Čím ovšem trpí všechny telefony, je rychlost připojení. Ano, v 90 % případů [8] typicky máme k dispozici použitelnou konektivitu, ovšem například při cestování není vždy k dispozici. Musíme se tedy zaměřit na co nejmenší množství přenášených dat na uživatelově zařízení. Tuto optimalizaci jistě ocení i uživatelé osobních počítačů, jelikož zapříčiní zrychlení i jejich verze. Toto má pozitivní dopad i na oblíbenost aplikace - toto dokazuje i článek[18] od společnosti Google zaměřený na problematiku rychlosti webových aplikací a souvislost s jejich oblíbeností.

Je potřeba vzít v potaz také to, že v případě telefonu má uživatel mnohdy sníženou pozornost při práci se zařízením, jak můžeme vidět v průzkumu[19]. Toto použití bychom mohli často přirovnat k sousloví „jedno oko, jeden palec“. Jde o to, že podle průzkumu se telefony používají spíše jako výplň například při čekání, jako výplň prázdného času či dokonce při sledovaní televize. Tato situace je v případě informačního systému méně pravděpodobná, ovšem cíl vytvořit jednoduché a snadno srozumitelné rozhraní i na základě tohoto je užitečné, jelikož při používání aplikace se uživatel cítí lépe.

Rozdíl bývá i v době přístupu v případě soustředění, k lepší ilustraci této situace nám pomůže vysvětlení od Rachel Hinman z firmy Nokia [6], kdy tento rozdíl přirovnala k potápění a šnorchlování. Potápění je případ počítače - přístup na delší dobu. Naopak šnorchlování bývá krátké a rychlé, ovšem do hloubky se dostat také můžeme. Je tedy důležité uživatele nevystavovat překážkám v podobě nepřehledného rozhraní, ovšem je potřeba jej zároveň neochudit o to, proč na stránku přišel a stránku neochuzovat o obsah.

Pokud tedy situaci shrneme, mobilní zařízení má několik výzev, na které musíme myslet. Malá obrazovka, mnohdy pomalejší nebo nestabilní internetové připojení a nižší pozornost uživatelů v některých případech, případně pozornost plná, ovšem na krátkou dobu. Řešení těchto problémů nás ovšem vede k tvorbě lepších webových stránek nejen na telefon, ale i počítač.

Jsou ovšem i věci, kterými mobilní telefon disponuje navíc, jsou rozšířením oproti standardnímu osobnímu počítači, či pokud uchopíme některou z nevýhod a využijeme ji v náš

[18] Článek o rychlosti webových aplikací https://ai.googleblog.com/2009/06/speed-matters.html

[19] Průzkum o požadavcích uživatlů http://web.archive.org/web/20110609040352/http://blog.compete.com/2010/03/12/smartphone-owners-a-ready-and-willing-audience/

i uživatelův prospěch. Jedna z nevýhod, nízká pozornost z důvodu neustále dostupnosti zařízení, a tím pádem fungující jako výplň „mrtvého“ času, lze uchopit jako výhodu, jelikož uživatel vlastně může k naší stránce přistupovat odkudkoli, nehledě na situaci a místo.

Ovšem existují věci, které bývají nejčastěji na mobilních telefonech, řeč je o nejrůznějších senzorech. Geolokace, digitální kompas a akcelerometr případně gyroskop poskytují velkou výhodu v případě, kdy potřebujeme znát aktuální polohu uživatele. U osobního počítače tyto situace musíme řešit dotazem na uživatele, případně odhadujeme polohu pomocí přípojky internetu. Tyto situace typicky nastávají u mapových aplikací či vyhledávačů, kdy nás zajímají například podniky či zastávky městské hromadné dopravy v okolí. Tyto funkce v případě našeho informačního systému nemají velký vliv, ovšem mohou se hodit v momentě, kdy potřebujeme zadat například místo vykonaného servisu. V tento moment můžeme třeba uživateli nabídnout automatické načtení polohy.

Fotoaparát či kamera bývá výbavou i osobních počítačů, ovšem toto zařízení se zde používá spíše pro videokonference než pro fotografování doplňujících informací. Většina moderních telefonů ovšem disponuje kvalitním fotoaparátem, který navíc lze použít velmi dobře k záznamu informací. V minulosti byly potíže s dostupností fotoaparátu v prohlížečích[20], ovšem v nových verzích operačních systémů s tímto potíže nejsou. Potíže mohou nastat se specifickým použitím, kdy chceme přímo do výstupu z kamery přidávat informace o tom, co se v ní aktuálně nachází, může být opravdu problém, ovšem i pro toto existují řešení[21]. Pro pouhý záznam fotografie a její následné uložení do informačního systému ale žádné potíže nejsou a tato funkce je k dispozici[22].

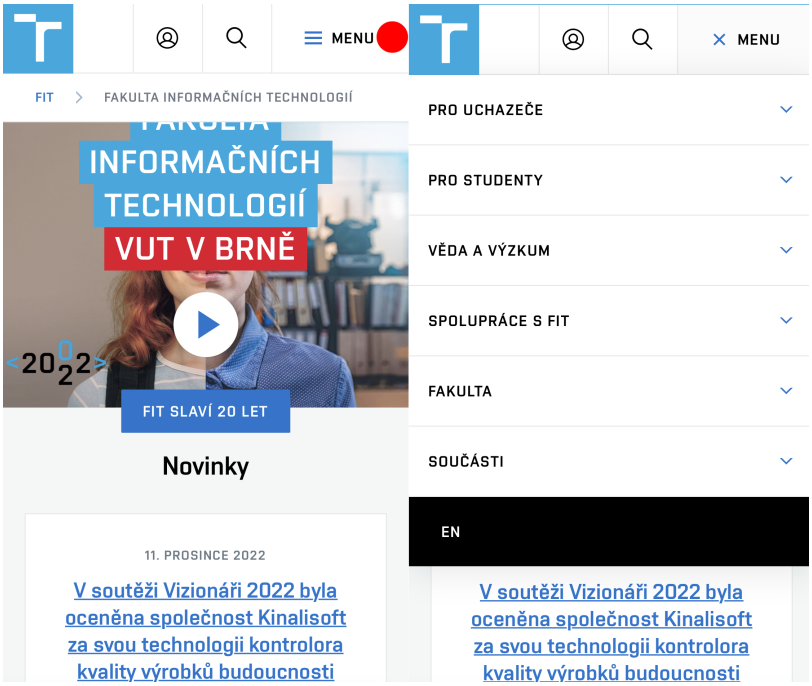
Jak tedy vytvořit kvalitní rozhraní, které bude uživateli vyhovovat na mobilním telefonu a zvrátit zmíněné nevýhody v náš prospěch? I tomuto se věnuje Luke Wroblewski v publikaci Mobile First [30]. Je potřeba se zamyslet, jak bude uživatel aplikaci používat a jaké úlohy bude nejčastěji hledat a uskutečňovat. Často může jít o rychlé vyhledání určité informace či zadání rychlých poznámek v případě informačního systému. Jelikož je na telefonu málo místa k zobrazení obsahu, je třeba mít na paměti, že obsah má přednost před navigačními prvky a zobrazovat hned obsah, pro který uživatel s největší pravděpodobností přišel. Ušetříme tak množství nutných akcí i snížíme množství přenesených dat. K zobrazení navigace je populární rozklikávací menu, což je kompromisem mezi zabranou plochou a dobrou přístupností. Menu můžeme vidět na obrázku 3.3, kliknutí proběhlo na místo označené červenou tečkou. Je také na místě zvážit, kde umístit tyto ovládací prvky. Z hlediska dostupnosti palcem je lepší spodní hrana, ovšem s tímto řešením bývají potíže z hlediska webového vývoje, toto si můžeme dovolit v nativní aplikaci. U webové verze je lepší umístit ovládací prvky v horní řadě. Je také potřeba držet ovládací menu jednoduché a přehledné.

Ovládací prvky a jiné klikatelné odkazy by měly být dostatečně velké. Nemělo by být naším cílem zmenšit položky za účelem maximalizace množství vložených ovládacích prvků. Na malé ovládací prvky se totiž špatně prstem trefuje a mohlo by se uživateli stávat, že omylem klikne na něco, co vlastně nechtěl. Tím se nám zvyšuje množství přenesených dat a snižuje se komfort uživatele. Konkrétně by tlačítka měla být minimálně 7 mm velká, optimálně 9 mm s minimálně 2 mm mezerami mezi tlačítky, lépe však 8 mm [7]. Rozmístění tlačítek také hraje roli. Hlavní ovládací tlačítka by měla být umístěna tam, kde jsou

[20]Přístup ke kameře z prohlížeče https://coderwall.com/p/epwmoa/html5-mobile-device-camera-access

[21]Web webAR https://www.nsocialtr.com/webar-web-augmented-reality-web-ar.html

[22]Povolení přístupu stránky ke kameře https://help.daily.co/en/articles/3388632-unblock-your-camera-microphone-on-mobile-android-and-ios

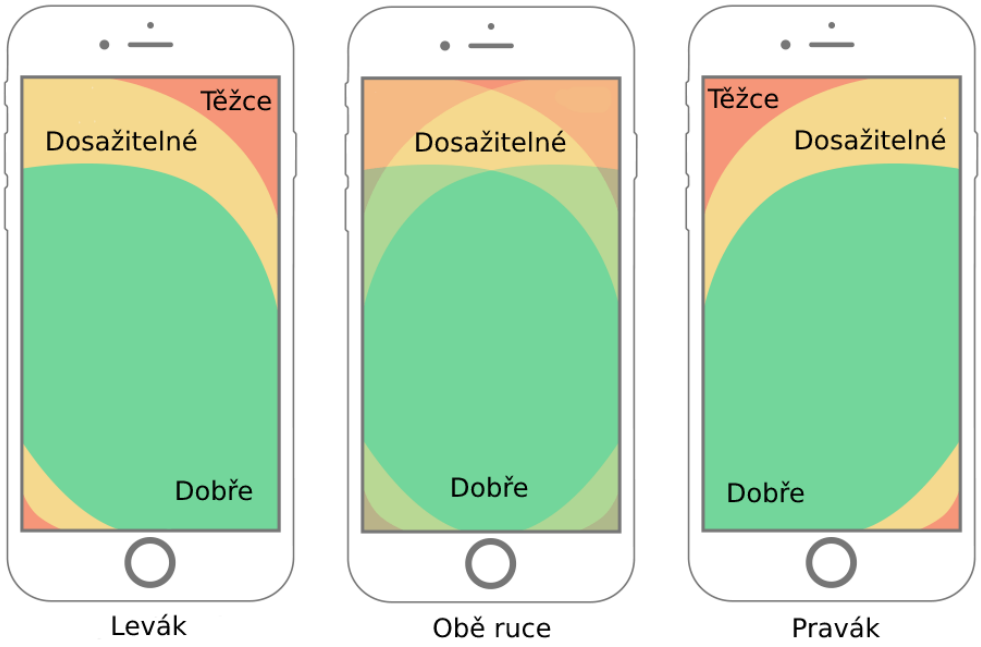


Obrázek 3.3: Rozklikávací menu

pro uživatele snadno dosažitelná. Méně používané ovládací prvky můžeme umístit do hůře dostupných oblastí. Tyto oblasti můžeme vidět na obrázku 3.4.

Z hlediska tradičně používaných gest jsme bohužel omezeni, jelikož některá gesta jsou použita přímo webovým prohlížečem. V tomto mají výhodu nativní aplikace. I tak ale máme určité množství gest k dispozici, toto se ovšem liší napříč různými prohlížeči a jejich verzemi. Je třeba experimentovat, co je v dané chvíli dostupné [30]. Co se týče takzvaných vyjíždějících nabídek známých z osobních počítačů, kdy na určitou položku najedeme kurzorem a změní se například jeho barva či se rozbalí nabídka nebo jiná námi zvolená událost, je třeba si dávat pozor, jelikož tato možnost není na dotykových zařízeních možná - jednoduše proto, že žádný takový kurzor neexistuje. O tyto prvky tedy přicházíme a v případě, kdy jsou pro ovládání aplikace podstatné, je třeba je implementovat jiným způsobem [30].

Formuláře a jiné položky určené pro vkládání obsahu jsou nedílnou součástí webových stránek, obzvláště potom informačních systémů, které jsou tvořeny za cílem zobrazování a vkládání dat. Je tedy nutné i tyto části vytvořit s důrazem na pohodlné používání na mobilních telefonech. Správnou tvorbu těchto položek popisuje také Wroblewski v publikaci [30]. Hlavní myšlenkou je opět uspořádání formuláře, kdy musíme mít na paměti, že pracujeme s malým displejem. Umístění popisků vkládacích polí proto nejspíše umístíme nad něj, nikoli do sloupců vedle sebe, nejlepší řešení je ale vložení krátkého popisu přímo do tohoto pole - je ovšem potřeba tuto vlastnost korektně implementovat, aby se nestalo, že tento popis zůstane ve formuláři a uživatel by jej musel mazat. Také je potřeba zvážit, zda není formulář příliš dlouhý a uživatel by se mohl ztratit v tom, k čemu vložená odpověď náleží. Je potřeba také vhodně vybírat typ vkládacího pole, zda použít zaškrtávací políčko, přepínač, políčko pro heslo, rozbalovací menu či vkládání textu - toto jsou standardní možnosti pro vložení. Je potom na nás zvolit, co bude pro uživatele nejrychlejší v daném případě. Pokud je to možné, je taky vhodné položku naplnit nejpravděpodobnější odpovědí, čímž uživatel nemusí položku vůbec zadávat. V případě textového pole HTML5 umožňuje



Obrázek 3.4: Dosažitelnost prvků palcem, přeloženo, převzato z [3] .

nastavit očekávaný vstup a dle toho nám poté prohlížeč zobrazí vhodnou klávesnici pro zadávání - pro email rozšířenou o znak zavináče, číselné pole zobrazí číselník, pro webovou stránku nabídku domén prvního řádu.

Při volbě rozložení je výhodné vytvořit responzivní, plynule přizpůsobitelný, vzhled webu závislý na aktuální šířce zařízení. Díky tomu jsme schopni efektivně využívat celé zařízení. Při řešení této problematiky můžeme narazit na potíže s různým množstvím pixelů na palec *PPI*, kdy pixely mohou být fyzicky různě velké. Tento problém je řešitelný pomocí parametru `width=device-width`[23], kdy za nás tento problém řeší prohlížeč. Je výhodné dodat obrázky a ostatní grafické prvky v nízkém a vysokém rozlišení, aby případně nedošlo ke snížení kvality obrázků. Nejvhodnější je použít prvky vykreslované přímo prohlížečem či vektorovou grafiku.

To je soubor hlavních tipů, na jaké skutečnosti se zaměřit a jaké jsou odlišnosti od webů vyvíjených pro osobní počítače. Hlavní radu na závěr Wroblewski [30] zmiňuje prototypování a zkoušení na zařízení. Můžeme mít soubor tipů, ovšem v konečném důsledku hraje hlavní roli použitelnost a dobrý uživatelský zážitek s naší aplikací. To je totiž rozhodující faktor.

[23] nastavení parametru https://www.quirksmode.org/Nastavenblog/archives/2010/09/combining_meta.html

Kapitola 4

Technologie pro tvorbu webových rozhraní

V dnešní době existuje několik technologických řešení pro tvorbu webových aplikací, které rozšiřují základní HTML, což je značkovací jazyk, a CSS, jakožto jeho doplněk pro nastavení designu jednotlivých značek. Toto jsou základní stavební kameny pro tvorbu statických webových stránek. Pomocí těchto technologii jsme schopni ovlivnit vizuální stránku webu, ovšem komunikaci s aplikačním rozhraním a jistou interaktivitu pomocí HTML a CSS nejsme schopni realizovat. Jazyk používaný pro programování webových aplikací je JavaScript. Tento jazyk se těší velké oblibě i dnes, byť vznikl v polovině devadesátých let. Pro ulehčení práce ovšem vznikají rozšíření zaměřená na usnadnění práce s tímto jazykem a celkové urychlení vývoje aplikace a tím pádem i snížení nákladů na vývoj.

Použití aplikačního rámce, jak tyto rozšíření nazýváme, nám totiž umožní používání mnoha již hotových neboli knihovních funkcí. Nemusíme se tak věnovat problémům, které se typicky řeší, a můžeme se zaměřit na specifika naší aplikace. Použití již hotových funkcí nás také částečně ochrání před špatnými přístupy při programování vedoucí například k bezpečnostním problémům. Aplikační rámce mívají typicky již předurčenou strukturu aplikace a tudíž nás i zde vedou systémovou cestou a neumožní nám se dopustit chyb, které nám v budoucnu například znemožní další rozvoj aplikace či bude velice nákladný z důvodu nepřehlednosti kódu. Těchto rozšíření existuje obrovské množství, zaměříme se proto jen na některé z nich.

4.1 Aplikační rámce pro uživatelské rozhraní

Nejpopulárnější aplikační rámec [10] nazvaný React, původně vyvinutý společností Facebook, je zaměřený primárně na tvorbu uživatelských prostředí. Projekt React[1] se hned na titulní stránce svého webu označuje jako deklarativní a komponentově založený. Slovo deklarativní [25] v programování znamená, že programátor řeší, jaké operace mají být provedeny. V jakém pořadí je ponecháno na programu, který tyto operace bude zpracovávat. Zjednodušeně tedy popisujeme, co se má provést, nikoli jak se to má provést. V případě rozšíření pro Javascript React si pod tím můžeme představit [13] takové chování, kdy chceme jistý prvek, například tlačítko, na určitém místě s určitými vlastnostmi. React nám zajistí, že zde tlačítko bude a nemusíme přemýšlet nad tím, jaké ostatní změny musíme provést k dosažení tohoto cíle.

[1] Webová stránka projektu React https://reactjs.org/

Další stěžejní vlastností tohoto projektu je cílení na komponenty. Co komponenta v tomto případě znamená [12, 13]? Můžeme si ji představit jako námi vytvořenou funkci v programovacím jazyku. Tvoří určitou část webového rozhraní, která je podle potřeby volána neboli *invokována*, a tím je daná část vykreslena. Komponenty mají vlastní strukturu, mohou obsahovat další funkce a vlastní aplikační rozhraní. Tyto komponenty pak můžeme znovupoužít v různých částech webového rozhraní, což nám ulehčí práci a také zpřehlední aplikační kód.

React nemá žádnou pevně určenou strukturu souborů [9], tudíž je pouze naší volbou, jak soubory nutné k běhu aplikace uspořádáme do adresářové struktury. Existují ovšem doporučení[2], jak lze k tomuto problému přistoupit. Prvním je uspořádání podle společného účelu - komponenty do jednotlivých složek. Druhou je rozdělení podle typu souborů do adresářů. Oba přístupy jsou populární, ovšem vzhledem k volnosti, kterou máme, si můžeme vymyslet vlastní. Doporučením od autorů dokumentace projektu React je nad tímto nestrávit příliš mnoho času na začátku projektu a zvolit nějaký přístup, který můžeme později změnit podle toho, co nám bude v tom kterém projektu vyhovovat.

Progresivnější variantou je projekt Vue.js[3], který v poslední době nabírá na popularitě. Vue.js je totiž zaměřen na výkon, jednoduchost použití, s tím související rychlou učící křivku, kdy při vývoji aplikace nemusíme použít veškeré součásti rozšíření a postavit aplikaci s ohledem na něj. Jednoduše v momentu, kdy narazíme na problém, který pomocí standardních nástrojů nejsme schopni vyřešit, použijeme část Vue.js dle našich aktuálních potřeb. Zaměření na výkon znamená i rychlost prvního načtení a, jak můžeme vidět v benchmarku [15], je Vue.js rychlejší ve všech ohledech než již popsaný základní React.

Další výhodou projektu Vue.js (stejnou výhodou ovšem disponuje i již popsaný React) je, že nám může usnadnit vývoj mobilních aplikací. Jednou z možností pro vývoj na platformy iOS a Android je použití rozšíření pro JavaScript nazvané NativeScript[4]. Existuje rozšíření NativeScript-Vue[5] s velmi podobnou syntaxí jako Vue.js.

Struktura souborů je zde před-generována skriptem pro inicializaci projektu vytvořeného v rozšíření Vue.js. Máme tedy solidní odrazový můstek, ovšem nic nám nebrání vytvořit strukturu vlastní. Bohužel ovšem v aktuální dokumentaci projektu není zmínka o příkladu použití jednotlivých složek, tudíž je na našem výkladu, jak se k tomuto postavíme. A tím pádem, jestli složky použijeme tak, jak autoři zamýšleli, nebo jejich plán ohneme jiným směrem.

4.2 Aplikační logika

Ovšem webová aplikace není jen webové rozhraní ale stojí za ní také aplikační logika, kterou je výhodné oddělit od vlastního rozhraní aplikace [22]. Získáváme výhody v podobě snížení zátěže na servery, jelikož vykreslování probíhá na zařízení u uživatele a na naše servery se obrací pouze se žádostmi o získání či zápis dat, případně jejich zpracování. Další výhodou v případě vhodně zvolené komunikace může být také aplikační rozhraní, ke kterému můžeme později vytvořit několik uživatelských rozhraní. K tvorbě aplikační logiky neboli *backendu* můžeme použít mnoho přístupů i programovacích jazyků. Volbou hned 48 % vývojářů [11] je ovšem programovací jazyk Python, zaměříme se proto na některé rozšíření zaměřené na tvorbu aplikační logiky v tomto jazyce.

[2]Dokumentace projektu React `https://reactjs.org/docs/faq-structure.html`
[3]Webová stránka projektu Vue.js `https://vuejs.org/`
[4]Webová stránka projektu NativeScript `https://nativescript.org/`
[5]Webová stránka projektu NativeScript-Vue `https://nativescript-vue.org/`

Nejdříve ale nastíníme problém komunikace mezi uživatelským rozhraním a aplikační logikou, což dnes bývá řešeno pomocí posílání textových dat, a to například ve formátu JSON či dříve populárnějším formátu XML, který ovšem dnes již začíná zaostávat z hlediska množství rozhraní, která jej využívají[6]. Obě řešení ovšem patří k nejrozšířenějším typům zápisů dat, která můžeme od takzvaného *REST API*, jež je velmi rozšířené, či nově nastupujícího *GraphQL* (zde je preferovaný dat ve formátu JSON), obdržet. REST API je zkratkou slov *Representational State Transfer*. Situaci si můžeme představit tak, že máme určitý požadavek, který doručíme v předem domluveném formátu na správný koncový bod naší aplikace, odkud se nám dostane také odpovědi. U koncových bodů mohou nastat potíže s množstvím dat, která nám zašle zpět. Buď jich může být zbytečně moc a zatěžujeme tak síťovou infrastrukturu, nebo jich může být naopak příliš málo a musíme se pro data obrátit na jiný koncový bod. Tuto situaci řeší specifikace aplikačního rozhraní GraphQL[7], kdy v dotazu specifikujeme požadovaná obdržená data.

Django[8] je rozšíření jazyka Python pro komplexní tvorbu webových stránek. Můžeme jej použít pro generování obsahu dané stránky přímo, či jej použít pro tvorbu aplikačního rozhraní. Django podporuje množství dalších rozšíření pro komunikaci s databází, zjednodušené generování nejrůznějších částí webových stránek a mnoho dalších[9]. Hlavní motivací k využití tohoto rozšíření ovšem je, že za nás Django řeší bezpečnost aplikace jak z hlediska šifrování přihlašovacích údajů a korektní práce s nimi, tak i nejrůznější útoky. Útokům typu podvržení formuláře a zaslání nečekaných dat za účelem získání kontroly či omezení funkcionality služby ostatním díky projektu Django můžeme předcházet nebo je dokonce zcela eliminovat. A to s minimálními nároky na vývoj - tyto věci jsou vestavěny přímo do tohoto rozšíření jazyka Python. Výhodou pro vývoj je také zvýšení udržitelnosti kódu tím, že umožňuje například znovupoužitelnost modulů, kdy jednu věc napíšeme jen jednou a poté ji jen replikujeme. Následná údržba je poté jednodušší a celkově by měl být kód aplikace přehlednější.

Rozšíření zaměřené čistě na aplikační rozhraní a jeho rychlou tvorbu - FastAPI[10], narozdíl od rozšíření Django, které je zaměřené na tvorbu celé webové aplikace, lze pomocí FastAPI realizovat pouze serverovou aplikační část a na tuto část je FastAPI zaměřeno. FastAPI je založen na standardu OpenAPI[11], který specifikuje formát parametrů, zabezpečení a obsah požadavků. Dále nabízí podporu JSON Schema[12], což je deklarativní jazyk pro specifikaci a validaci formátu JSON. Další zajímavostí může být automatické generování dokumentace. Vestavěny jsou dvě možnosti, a to pomocí projektu Swagger UI[13] nebo ReDoc[14], což je možné díky standardu OpenAPI. Obě tyto možnosti generují interaktivní dokumentaci v podobě webové stránky. Celý koncept FastAPI je postaven na moderním Pythonu 3.9 a nabízí tak snadný začátek lidem, kteří se s tímto programovacím jazykem setkali a také možnost využívat všech výhod tohoto populárního jazyka.

[6] Porovnaní JSON a XML https://hackr.io/blog/json-vs-xml
[7] Webová stránka specifikace GraphQL https://graphql.org/
[8] Webová stránka projektu Django https://www.djangoproject.com/
[9] Rozšíření projektu Django https://docs.djangoproject.com/en/4.1/py-modindex/
[10] Webová stránka projektu FastAPI https://fastapi.tiangolo.com/
[11] Webová stránka OpenAPI https://github.com/OAI/OpenAPI-Specification
[12] Webová stránka standardu JSON Schema https://json-schema.org/
[13] Webová stránka projektu Swagger UI https://github.com/swagger-api/swagger-ui
[14] Webová stránka projektu ReDoc https://github.com/Rebilly/ReDoc

4.3 Databázové systémy

Aplikační logika tedy většinu času operuje nad daty, která je ovšem potřeba někde ukládat. K uložení dat slouží nejrůznější databázové systémy. Opět se zaměříme na pár zástupců, jelikož popis všech databázových systémů není v možnostech této práce. Zvolíme jednu databázi relační a druhou nerelační pro zohlednění největšího rozdílu mezi aktuálně nejvíce [14] používanými typy databází pro webové aplikace. Rozdíl mezi relační a nerelační databází spočívá v tom, že relační databázi si můžeme zjednodušeně představit jako množství provázaných tabulek s daty klíč-hodnota. Nerelační databáze, neboli `Not Only SQL`, volně přeloženo *Nejen SQL*, jsou databáze, které se dále dělí dle reprezentace dat například na grafově orientované databáze, dokumentově orientované databáze nebo uložení přístupem klíč-hodnota. Záleží poté na konkrétních datech, která potřebujeme ukládat pro volbu té které databáze vhodné pro náš projekt.

Databáze SQL zajišťují soubor vlastností nazvaný *ACID* [4]. Tento soubor znamená, že transakce do databáze musí být atomické, konzistentní, izolované a trvanlivé [31]. Atomičnost operace znamená, že je operace nedělitelná, je provedena vždy celá a nemůže nastat situace, kdy by byla provedena pouze její část. Konzistence poté označuje, že operace žádným způsobem nenaruší konzistenci databáze. Operace, která je izolovaná, i v případě, kdy by probíhala s jinou operací paralelně, se navenek chová, jako by probíhala pouze samotná. Poslední vlastností je trvalost operace, operace po skončení zanechá v databázi trvalý charakter, a to i v situacích, kdy by došlo k výpadku systému. Databáze NoSQL nemusí vždy dodržovat soubor vlastností ACID.

DynamoDB je zástupce nerelační, takzvané *NoSQL*, databáze poskytované společností Amazon na jejich cloudovém řešení *AWS*. Nerelační databáze DynamoDB[15] automaticky rozkládá zátěž mezi jednotlivé servery v případě více tabulek s daty pro maximální výkonnost, kdy každá tabulka může být zpracovávána na jiném serveru a tím je tak zajištěna maximální rychlost. DynamoDB zároveň také disponuje automatickým zálohováním naší databáze a umožňuje její obnovení po až 35 dnů. Tuto funkci můžeme ocenit v momentech, kdy dojde ke ztrátě dat například kvůli nedostatečnému zabezpečení či chybě uživatele. Další zajímavou funkcí může být automatické mazání zastaralých dat.

Pro přístup do databáze můžeme použít buď aplikační rozhraní *DynamoDB API*, případnou druhou možností je využití *PartiQL* což je jazyk pro dotazování kompatibilní s SQL jazykem pro dotazování.

Podle statistik je nepoužívanější relační databází na světě databáze Oracle [14] pojmenovaná stejně jako společnost, která ji vyvíjí. Na druhé příčce je relační databáze nazvaná *MySQL* taktéž od společnosti Oracle. Databázi Oracle je možné nainstalovat na operační systém Microsoft Windows, Unixové systémy a systémy na bázi Linuxu[16]. Díky tomuto máme pokrytou většinu[17] serverových operačních systémů. Databáze Oracle také nabízí síťovou vrstvu, díky které máme zajištěnou multiplatformnost databáze, kdy se můžeme k databázi běžící na jednom druhu operačního systému připojit z druhého.

Dalšími vlastnostmi, kterými databáze Oracle disponuje je možnost přistupovat k struktuře dat logicky - nemusíme znát skutečné fyzické místo dat, databáze nám přistup k datům automaticky zajistí. Rozdělení dat do menších celků má za důsledek zvýšení výkonnosti da-

[15] Představení databáze DynamoDB https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

[16] Představení databáze OracleDB https://www.oracletutorial.com/getting-started/what-is-oracle-database/

[17] Statistika použití serverových OS https://www.statista.com/statistics/915085/global-server-share-by-os/

tabáze, jelikož velké tabulky dat lze rozdělit na menší a ty následně uložit na různá fyzická úložiště a přistupovat k nim poté paralelně. A vlastně nejdůležitější vlastnost jako je zálohování a následná obnova dat, je v databázi Oracle umožněna hned v několika krocích, a to i například i inkrementální zálohování a následná možnost vrátit se o několik operací zpět.

Kapitola 5

Analýza

Představili jsme si teoretické postupy a technické prostředky k realizaci informačního systému. Ovšem pro jeho tvorbu je základem důležité pochopení toho, co uživatel od systému požaduje a jaké jsou jeho cíle. Až v tento moment jsme schopni využít nabytých znalostí k tvorbě systému, který bude alespoň částečně splňovat požadavky svých uživatelů. S tím nám pomůže rozbor situace, popis uživatelů, výrobce a existujících řešení, díky čemuž lépe pochopíme naše uživatele.

5.1 Firma Viessmann

Mezi výrobce plynových kotlů patří firmy jako Protherm, Weishaupt, Junkers či italská společnost Baxi. Výrobců je ovšem mnohem více, avšak každý z nich požaduje proškolení pro techniky, kteří se plánují zabývat servisem jejich zařízení. Proto není v možnostech jednoho technika obsloužit všechny výrobce a typicky si vybírá jen několik výrobců, na jejichž zařízení bude poskytovat servis.

Firma Viessmann[1], jakožto další z řady výrobců plynových zařízení, vznikla roku 1917 jako rodinný podnik a nyní se zabývá výrobou topné, chladicí techniky a energetických systémů. Patří mezi přední dodavatele těchto systémů. V oblasti vytápění se zabývá tepelnými čerpadly, elektrickými kotli a také výrobou plynových kotlů, což je, mimo aktuální mimořádné situace, stále považováno za ekonomický a relativně ekologický zdroj tepla. Avšak plynové kotle vyžadují pravidelné kontroly a případné opravy, což je stejné jako i u ostatních druhů vytápění, kde dochází k hoření či jiné energetické přeměně - zanedbání pravidelné údržby vede ke zkrácení životnosti výrobku. U plynových kotlů jsou však požadavky nároky na servisní techniky vyšší už jen z hlediska toho, že v plynovém kotli se vyskytují dva druhy „energie“ a to plyn a elektřina.

Kvůli tomu je v kotli také více dílů a řídící techniky, které se časem mohou poškodit nejen vlivem prostředí uvnitř kotle, kde často dochází ke změnám teplot, jelikož plyn zde nehoří neustále a dochází k jeho zapalování v momentech, kdy je potřeba, ale i kvůli prostému opotřebení. Dodavatelé plynových kotlů proto musí být tedy zároveň i dodavateli množství náhradních dílů, jelikož měnit plynový kotel při poruše nedává ekonomický a ani ekologický smysl - na výrobu kotle je potřeba mnoho materiálů a práce k jejich sestavení.

[1] Webová stránka firmy Viessmann https://www.viessmann.cz/

5.2 Uživatel informačního systému

Nejpravděpodobnějším uživatelem informačního systému na správu zakázek pro techniky plynových kotlů bude drobný podnikatel, živnostník, který nabízí servis plynových kotlů, jejich revize či jiné práce spojené s tímto typem zařízení. Vzhledem k tomu, že vytápění v rodinných domech a mnohdy i bytech bývá špatně centralizovatelné, je potřeba mezi kotli cestovat. U bytů je někdy realizované dálkové vytápění, ale i zde většinou ve výsledku narazíme na plynový hořák, který také vyžaduje servis. Jedná se tedy o práci na cestách, v topné sezóně i velmi časově náročnou vzhledem k vyššímu zatížení zařízení a tím pádem více poruchám než přes léto, kdy je plynový kotel použit k ohřevu teplé vody či může být zcela odpojen.

Vzhledem k práci na cestách a pro mnoho zákazníků zároveň vzniká množství evidence o cestovném, stráveném čase prací, použitých dílech či podrobnější informace o stavu zařízení, jako například složení spalin. Zde nastává otázka, kde tato data ukládat, v ideálním případě na internetu, jelikož převážet množství dat s sebou pro snadno dostupnou historii aktuálně servisovaných kotlů není komfortní.

Při následné tvorbě faktur je poté všechny tyto informace agregovat a vytvořit na jejich základě výsledný doklad pro zákazníka. V případě náhradních dílů je poté třeba dohledat jejich výrobcem doporučené ceny pro koncové zákazníky, s čímž by nám mohlo pomoci napojení na data výrobce.

5.3 Existující řešení

Specializovaných online služeb na správu zakázek existuje hned několik, ovšem můžeme využít služeb i z některých poznámkových bloků, ovšem toto klade požadavek na uživatele v podobě uspořádání dat. Lepší volbou tedy bude využití některé z dostupných služeb. Nástroj Bezšanonu[2] je naopak zacílen pouze na správu dokladů a faktur, nikoli na sledování zakázek - pro námi zamýšlené použití se tedy nehodí. Další možností je aplikace Flou[3] či Nicontrol[4] nabízející pěkné uživatelské rozhraní, na mobilním telefonu ovšem hůře ovladatelné. Aplikace také umožňuje vkládání fotografii k zakázkám, ovšem tato funcionalita je limitována i v placeném programu, na který jsme nuceni přejít po 30 dnech v případě Flou nebo 14 dnech v případě Nicontrol, kdy expiruje bezplatná varianta. Placený program je bohužel poměrně drahý a stále neumožňuje vkládání velkého množství fotodokumentace, která může časem vzniknout. Jiným nástrojem je potom MůjZakázkovník[5], který lze za určitých podmínek používat zdarma stále, ovšem je zde limit na maximální počet zakázek. Bohužel tento limit je obsažen i v placené verzi této služby.

5.4 Možné řešení

Ideální řešení by se mělo dát rozumně ovládat jak z mobilního telefonu pro práci na cestách, tak z osobního počítače při tvorbě faktur. Nemělo by mít limit na množství zakázek a v ideálním případě také neomezený objem vložených fotografii - tyto dvě věci se dají jen stěží zajistit, jelikož data je pořád potřeba fyzicky někde uložit. Mělo by být tedy alespoň možné úložiště dat rozumně zvětšovat za adekvátní poplatek. Historie servisních zásahů

[2] Webová stránka aplikace Bezšanonu https://bezsanonu.cz/
[3] Webová stránka aplikace Flou https://www.flou.cz/
[4] Webová stránka aplikace Nicontrol https://www.nicontrol.cz/
[5] Webová stránka aplikace MůjZakázkovník https://www.mujzakazkovnik.cz/

u zákazníka by také měla být dostupná. Velkým a hlavním ulehčením práce by bylo však připojení systému na rozhraní výrobce, odkud by bylo možné například informace o cenách získávat automaticky bez nutnosti jejich vyhledávání v katalogu.

Kapitola 6

Návrh řešení

Z pohledu uživatele je tedy cílem realizovat aplikaci přístupnou z mobilního telefonu i osobního počítače. Do aplikace je potřeba vkládat hlavně textová data a případně i multimediální data, typicky obrázky, která je ale potřeba nějakým způsobem uložit. Tento typ dat je potřeba také zobrazit, a to v různých přehledech, kdy zobrazíme jen základní informace a po otevření položky zobrazovat informace podrobnější. Data je potřeba také upravovat v případě chyby vložení.

Technické řešení je potom mírně komplikovanější než pouhý pohled uživatele. Potřebujeme zapracovat kompletní správu dat, uživatelské rozhraní a aplikační logiku. Mít totiž propojenou aplikační logiku s rozhraním je problematické mj. z hlediska bezpečnosti, kdy do uživatelské části bychom museli přenášet například klíče k databázi, což může otevřít dveře případným útočníkům. Je tedy výhodnější aplikaci rozdělit na dvě části, kdy jedna je pouze uživatelská část a druhá aplikační, kdy tyto dvě části spolu komunikují přes jisté rozhraní, například REST API.

Serverová část poté bude komunikovat s dalšími součástmi aplikace. Například s úložištěm dat - databází. Je však na zvážení, zda data od výrobce načítat v této části či v části uživatelského rozhraní. Obě varianty jsou možné a bude záležet na podrobnostech implementace a konkrétních požadavcích uživatelů.

Uživatelské rozhraní poté bude zasílat požadavky na data potřebná k zobrazení kompletního uživatelského rozhraní včetně naplnění daty. Tato část bude provozována zcela na zařízení uživatele. Bude tedy možné aplikaci stáhnout do zařízení a tím minimalizovat následný datový tok omezený pouze na získávaná data.

6.1 Rozhraní výrobce

Možnost komunikace s naší aplikací nám nabídla společnost Viessmann. K získávání dat máme k dispozici URL adresu katalogu a přihlašovací údaje. Koncový bod sloužící k přihlášení komunikuje pomocí HTTP metody POST a přijímá uživatelský vstup z formuláře. Další koncový bod rozhraní výrobce plynových kotlů Viessmann nabízí souhrnné informace o dílech podle vyhledávacího klíče, kde jsme schopni vyčíst také cenu a to ve formátu HTML. Tento koncový bod komunikuje, na rozdíl od bodu pro přihlášení, pomocí HTTP metody GET.

K přístupu k datům je ovšem potřeba přihlašovacích údajů. Přístup k těmto datům je založen na proškolení technika k servisu zařízení značky Viessmann, což znamená, že přístup mají jen certifikovaní opraváři. Každý uživatel tedy bude muset zadat svoje přihlašovací

údaje, pomocí kterých následně získá přístup k těmto datům. Aplikace bude funkční i bez těchto přístupových práv, ovšem nebude uživateli zobrazovat a nabízet načtení podrobností o výrobku.

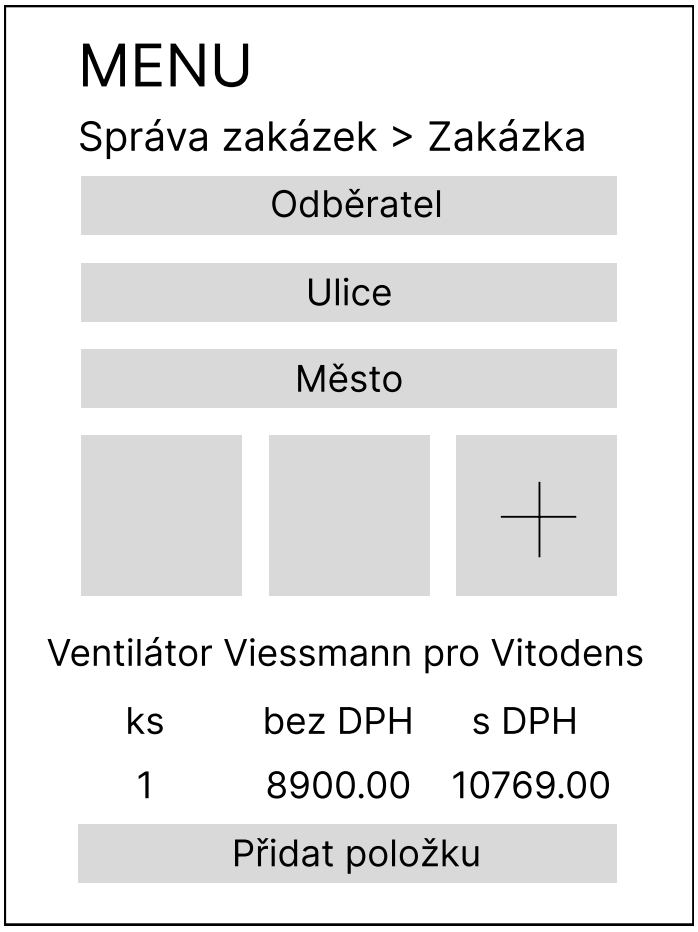
6.2 Návrh uživatelského prostředí

Návrh uživatelského rozhraní je zaměřen jak na osobní počítač, tak na mobilní telefon. Obě rozhraní je potřeba navrhovat rozdílně se zaměřením na jejich specifika přístupu.

Na obrázku 6.1 můžeme vidět uživatelské rozhraní pro osobní počítač v momentu tvorby či úpravy informací o zakázce. Máme k dispozici datová pole obsahující informace o odběrateli, můžeme přidávat fotografie a použité položky. Dále můžeme vidět informace o cenách a jejich součet. Na obrázku 6.2 vidíme stejnou situace jen s rozdílem v zobrazení pro mobilní telefon. To neobsahuje veškeré podrobnosti, které lze zadat později pohodlně z osobního počítače. Záznam potřebných informací pro identifikaci zakázky, uložení fotodokumentace a přidání položek je však stále dostupné k použití.

Obrázek 6.1: Návrh rozhraní pro osobní počítač

Z návrhu vychází fakt, že u každé zakázky je potřeba evidovat množství dat, pomocí kterých jsme pak schopni vyhotovit fakturu pro zákazníka. O zákazníkovi potřebujeme znát jeho fakturační údaje jako jsou jméno, adresa, kontaktní údaje - například telefon či email a v případě, že zákazník disponuje daňovým identifikačním číslem či identifikačním číslem, tak i toto. Dále je pro nás výhodné evidovat například údaje o zařízení jako jeho umístění, uvedení do provozu, typ a výrobní číslo, díky kterým se může technik lépe připravit v případě následujících oprav.



Obrázek 6.2: Návrh rozhraní pro mobilní telefon

Kapitola 7

Implementace a testování

Na základě návrhu tedy budeme implementovat dvě aplikace. Aplikační část neboli *backend* a uživatelskou aplikaci neboli *frontend*. Každou z těchto aplikací budeme implementovat v jiném programovacím jazyce - podle výhod a nevýhod toho kterého jazyka a technologie pro určitý účel. V kapitole 4 byla pro tvorbu aplikační části byla zvolena knihovna FastAPI doplňující programovací jazyk Python o snadnou tvorbu těchto rozhraní. Pro uživatelskou aplikaci volíme aplikační rámec Vue.js.

7.1 Aplikační část

Aplikace postavené pomocí technologie FastAPI využívají modularity jazyka Python, kdy celkovou funkcionalitu můžeme rozdělit do několika menších částí, které pak řídí danou funkcionalitu - v tomto případě koncový bod. Koncových bodů máme hned několik. Postupně si je projdeme a zaměříme se jejich zajímavé části. Kompletní dokumentaci s příklady poskytuje koncový bod `docs`. V dokumentaci lze také vidět požadované položky pro daný úkon a také případnou odpověď.

Práce s databází

Pro aplikační část je potřeba také uchovávat data. Jelikož pro nasazení FastAPI aplikace používáme Lambda funkci od poskytovatele Amazon, využíváme také jejich NoSQL databázi DynamoDB. Práce s touto databází však byla abstrahována pomocí vlastních funkcí, které implementují kýžené operace pomocí operací přímo spojených s databází DynamoDB, konkrétně jde o funkce z knihovny `boto3`. Díky užití abstrakce případný převod aplikace na použití jiné databáze obnáší pouze úpravu několika funkcí, nikoli celé aplikace.

Používáme tedy třídu `Database`, jejíž konstruktor získá přístup k potřebné tabulce s daty, nad kterou následně provádíme snadno operace tvorby, úpravy, získání a odstranění. Pokud potřebujeme volit položku pro práci, předáme konkrétní metodě identifikátor datového pole a jeho hodnotu.

Nad třídou `Database` jsou implementovány další třídy pro práci s konkrétním typem dat, jelikož každý druh dat má svá specifika ať se jedná o názvy položek či specifické úkony.

Aplikace potřebuje ukládat data o uživatelích, zakázkách a také údaje pro získávání dat třetích stran, v našem případě od firmy Viessmann. Každý druh dat je uložen v jedné tabulce, podrobnější strukturu dat si probereme v konkrétních částech koncových bodů, které s daným druhem dat pracují.

Koncový bod `users`

Správu uživatelů zajišťuje koncový bod `users`. Správa uživatelů znamená základní operace tvorby, úpravy, získání uživatele a jeho odstranění s tím, že každý uživatel může tyto operace provádět pouze s účtem, ke kterému je právě přihlášen. Strukturu uživatele ve formátu JSON znázorňuje výpis 7.1. Pro práci s tímto typem dat používáme třídu `DatabaseUsers`, která posouvá míru abstrakce nad databází o další úroveň, kdy pracuje již přímo s tabulkou `database_users`.

```
{
  "username": "string",
  "email": "string",
  "password": "string",
  "name": "string",
  "surname": "string",
  "user_id": "string"
}
```

Výpis 7.1: Datová struktura uživatele

Celá datová struktura 7.1 se ovšem nepoužívá při komunikaci na koncovém bodu, obzvláště jde o položku `password`, která se používá jen při tvorbě či aktualizaci uživatelského hesla. Jeho *hash*[1], který máme uložený v databázi, využíváme pouze interně při ověření přihlášení uživatele.

Tvorba uživatele probíhá zasláním dat ve formátu JSON pomocí HTTP požadavku POST. Ovšem všechny datové položky zmíněné v 7.1 nejsou pro tvorbu uživatele vyžadovány, obzvláště potom položka `user_id`, která je generovaná přímo v aplikační části pro zajištění unikátnosti. Položka `password`, která je povinná, poté obsahuje *hash* obdrženého hesla vytvořeného pomocí algoritmu pro šifrování hesel `bcrypt`. Položka `username` je určená pro uživatelem zvolenou přezdívku, která je stejně jako `user_id`, neměnná, jedinečná a slouží k přihlašování uživatele.

Úprava poté probíhá obdobně zasláním těchto dat pomocí HTTP metody PUT ovšem s tím rozdílem, že se aktualizují pouze data, ve kterých došlo ke změně - tedy není jejich příchozí hodnota nastavena na výchozí. Tohoto je docíleno prvotním získáním původních uživatelských dat, porovnáním a aktualizací s daty příchozími a následným zápisem dat to databáze. Díky tomuto řešení není aktualizováno heslo, pokud uživatel aktualizaci nevyžádal, což bylo hlavní motivací pro toto řešení.

Získání (HTTP metoda GET) i odstranění (HTTP metoda DELETE) uživatele probíhají podobným způsobem a to zasláním uživatelského `user_id` jako parametr url adresy. V případě získání uživatele obdržíme kompletní strukturu 7.1 bez položky `password` obsahující uživatelská data. U mazání poté projdeme také všechny ostatní tabulky s daty, smažeme položky příslušející odstraňovanému uživateli a samozřejmě i jej.

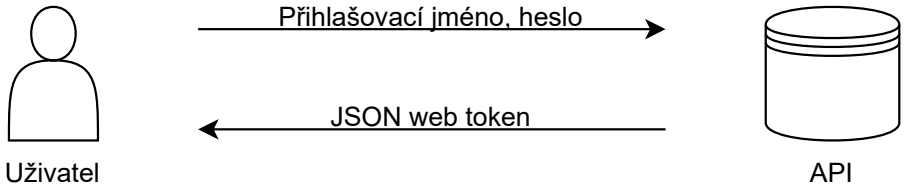
Koncový bod `login`

O přihlašování a autentizaci uživatelů se stará koncový bod `login`. K tomuto používáme také metody třídy `DatabaseUsers` využité již v koncovém bodě `users`. Na obrázku 7.1 můžeme vidět symbolizaci přihlášení k aplikaci. Uživatel zašle ve formátu JSON přihlašovací

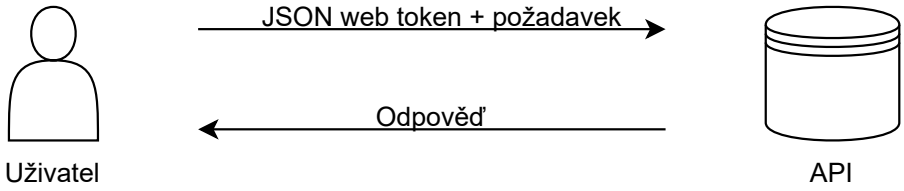
[1]Hash [17] je jednosměrná funkce, která na základě řetězce vygeneruje jiný řetězec konstantní délky, ovšem z tohoto řetězce nelze zpětně vygenerovat původní řetězec.

údaje, kterými jsou položky `username` a `password`. Aplikační rozhraní vyhledá v tabulce s uživateli uživatele s příslušným `username`. Porovná *hash* zadaného hesla a hodnoty z databáze u příslušného uživatele. Pokud se shodují, aplikace vygeneruje přístupový token s nastavenou plastností pomocí algoritmu `HS256`. Výsledkem je řetězec se zašifrovaným uživatelským jménem, který obdrží uživatel, takzvaný *JSON web token*, neboli *JWT*.

Uživatelská část aplikace tento klíč přidá mezi své HTTP hlavičky a každý následující požadavek již odesílá s touto hlavičkou, což značí obrázek 7.2. Aplikační rozhraní poté verifikuje token, tak ověří jeho expiraci a dešifrujeme z něj uživatelské jméno. Následně toto uživatelské jméno zkusíme získat z databáze. Pokud uspějeme, uživatele jsme úspěšně ověřili a můžeme mu poskytnou požadovaná zabezpečená data. Celý tento proces se nazývá *Bearer autorizace*.



Obrázek 7.1: Přihlášení



Obrázek 7.2: Přístup k zabezpečeným datům

Koncový bod `invoices` a `invoices_list`

Práci se zakázkami, jejichž struktura je zachycena ve výpisu 7.2, zajišťují dva koncové body. `Invoices` obstarává veškerou práci se zakázkami - tvorbu, získání, úpravu a odstranění konkrétní zakázky. Položky zakázky `invoice_id, created, updated` a `user_id` jsou nastaveny v aplikační části automaticky bez vlivu uživatele pro ulehčení práce s tímto koncovým bodem. V rámci testování byly upraveny původně navržené položky pro uložení o zakázce, jelikož jejich množství bylo až obtěžující, vzhledem k tomu, že ne každý zákazník disponuje například daňovým identifikačním číslem. Tyto položky byly zahrnuty do položky `notes`, která poskytuje prostor pro všechny poznámky. Také v rámci testování došlo k rozhodnutí, kdy nastavovat pole jako povinná či omezovat jejich vstup na konkrétní typ dat je nepraktické. Aplikace neslouží pro výslednou tvorbu faktur, ovšem pro tvorbu podkladů. Například k ceně je tedy možné si doplnit poznámku přímo do tohoto pole. Tento fakt poskytuje uživatelům volnost a neklade na ně žádný tlak v časové tísni, do které se někdy při servisních zásazích mohou dostat. Můžeme také vidět datové pole `photos`, které slouží jako příprava pro přidávání fotografii k zakázkám, což může být velmi zajímavé rozšíření aplikace.

```
{
  "invoice_id": "string",
  "invoice_name": "string",
  "created": "2023-05-07T23:19:12.932Z",
  "updated": "2023-05-07T23:19:12.932Z",
  "user_id": "string",
  "items": [
    {
      "name": "string",
      "price": "string",
      "partnumber": "string",
      "manufacturer": "string",
      "note": "string"
    }
  ],
  "photos": [
    "string"
  ],
  "customer": "string",
  "note": "string"
}
```

Výpis 7.2: Datová struktura zakázky

Koncový bod `invoices_list` zajišťuje rychlý výpis všech zakázek, který obsahuje pouze položky `invoice_id, invoice_name, user_id, customer, created` a `updated`, což jsou položky potřebné pro zobrazení zakázky a zároveň neobsahující mnoho položek, které zvyšují datový přenos. Jako další optimalizaci je možné implementovat postupné načítání, avšak datový objem je zatím tak malý, že uživatelé mohou profitovat z rychlosti vyhledávání přímo v uživatelské části, pro kterou však potřebujeme kompletní seznam zakázek. Tuto vlastnost uživatelé aplikace při testování ocenili.

Koncový bod `viessmann_login` a `viessmann`

Tento koncový bod vzešel z testování aplikace. Původní myšlenkou bylo totiž pracovat s přihlašovacími údaji v uživatelské části aplikace tak, že se uživatel přihlásí k přístupu k datům firmy Viessmann, získaný přístupový klíč uložíme do lokálního úložiště prohlížeče a následně budeme získávat data odtud. Toto řešení se však ukázalo značně nepraktické, jelikož klíč má krátkou dobu platnosti a bylo vyžadováno pravidelné přihlašování. Navíc uživatelé aplikace ne vždy disponují kvalitním připojením k internetu a načítání dat tak bylo mnohdy zdlouhavé, případně zakončené neúspěchem kvůli expiraci klíče. Data jsou totiž dostupná pouze v podobě HTML stránky generované na serveru, nikoli v standardizovaném formátu typu například JSON, či XML.

Bylo potřeba tedy najít jiné řešení - což je tento koncový bod. Bod `viessmann_login` přijímá přihlašovací údaje k Viessmann účtu, které ukládá do databáze pomocí třídy `DatabaseViessmann`. Uživatelské heslo je šifrováno pomocí symetrické kryptografie `Fernet`, jsme jej tedy schopni rozšifrovat jen na základě tajného klíče v aplikační části.

Získání dat poté probíhá zasláním požadavku na získání přihlašovací stránky s formulářem, tím zároveň dostaneme potřebné *Cookies*[2] pro přihlášení. Přihlášení probíhá zasláním přihlašovacích údajů, které získáme z databáze příslušející aktuálně přihlášenému uživateli do daného formuláře pomocí HTTP metody POST. Následně jsme přesměrováni do editace uživatelského účtu - tedy návratový HTTP kód 302 značí úspěšné přihlášení. V momentě, kdy jsme úspěšně přihlášeni, můžeme přejít k načítání kýžených dat. Data se nachází na koncovém bodu `etapp/parts/cs/9700/{produktové_číslo}`, která vrací pomocí HTTP metody GET HTML stránku s informacemi o produktu. Pod HTML značkou `product-detail-price` najdeme cenu produktu. Ke zpracování HTML stránky je použita Python knihovna BeautifulSoup, která usnadňuje strojové zpracování HTML stránek. Získaná data následně odešleme ve formátu JSON klientovi.

Toto řešení se ukázalo jako nejlepší kompromis mezi zabezpečením a uživatelskou přívětivostí. Stačí se pouze jednou přihlásit a v daném účtu máme již po celou dobu přístupná data od výrobce Viessmann.

7.2 Uživatelská část

Uživatelská část je implementována v JavaScript rozšíření Vue.js ve verzi 3 s veřejně dostupnou knihovnou kaskádových stylů BootStrap[3] ve verzi 5. Vue.js používá několik přístupů k vývoji, v našem případě byl použit takzvaný *Single File Component* a kombinace *Composition API* s *Options API*, které jsou vzájemně kompatibilní a lze použít jejich kombinaci.

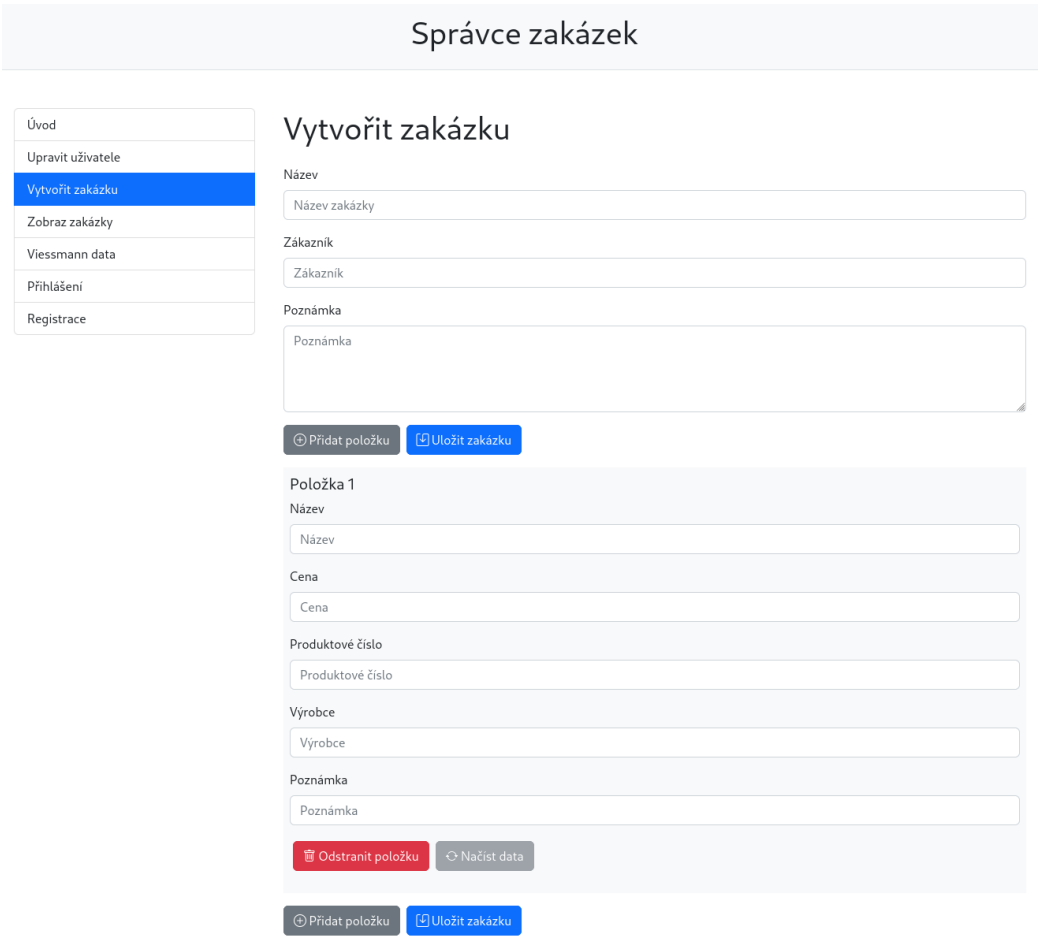
Při načtení stránky se dostáváme na přihlašovací stránku, zároveň máme přístup na stránku s registrací. Ostatní stránky nás při pokusu o přístup v případě, kdy nejsme přihlášeni, či přihlášení vypršelo, přesměrují zpět na přihlašovací stránku - toto zajišťuje rozšíření `vue-router`. Správu přihlášení primárně zabezpečuje soubor `useAuth.js`, který nastavuje hlavičky, ukládá přístupový klíč do lokálního úložiště pro zachování přihlášení, získává informace o aktuálně přihlášeném uživateli a další funkce spojené s přihlášením uživatele.

Následnou komunikaci s aplikačním rozhraním má na starost soubor `api.js`, který je importován přímo do projektu pro usnadnění komunikace s aplikačním rozhraním. Jeho hlavním úkolem je globální nastavení URL adresy aplikačního rozhraní a přesměrování na přihlášení v případě, kdy již nemáme k danému zdroji přístup.

Rozložení stránky můžeme vidět na obrázku 7.3, kdy v levé části vidíme hlavní menu aplikace. Nyní se nacházíme v tvorbě zakázky, kde je každá položka volitelná, což vzešlo jako požadavek z testování aplikace - umožnit technikovi vkládat například poznámku přímo k ceně. V horní i spodní části vidíme několik tlačítek. Horní tlačítka byla přidána po testování, kdy v případě velkého množství položek trvalo příliš dlouho přidání položky nové. Nyní stačí kliknout nahoře na tlačítko `Přidat položku`, vygeneruje se nová položka a zároveň se na ni posuneme. Tlačítko `Načíst data` se zobrazuje pouze v momentě, kdy jsme přihlášeni k datům výrobce Viessmann, a je aktivní v případě, že zadáme odpovídající produktové číslo a název dodavatele produktu.

Stránku ve zobrazení pro mobilní telefon, konkrétně zde na pro Apple iPhone 12/13 mini, můžeme vidět na obrázku 7.4. Na obrázku vidíme stránku se zobrazením zakázek, využíváme aplikační koncový bod `list_invoices` pro jejich rychlé načtení. Díky tomu máme k dispozici funkci okamžitého vyhledávání podle názvu. Tato možnost byla při testování

[2]Malé množství dat uložené v klientském prohlížeči.
[3]Webová stránka knihovny BootStrap https://getbootstrap.com/



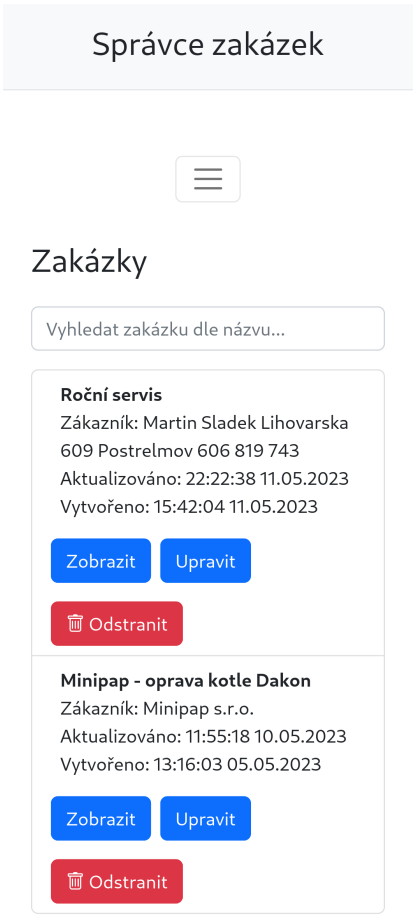
Obrázek 7.3: Tvorba zakázky - počítač

velmi ceněna. V horní části obrázku vidíme tlačítko pro zobrazení menu, které reflektuje dosažitelnost prvků palcem z obrázku 3.4.

Sekce s vlastním seznamem zakázek je seřazena dle poslední aktualizace zakázky. Pro každou zakázku je zde trojice tlačítek. Tlačítko Zobrazit nás přesměruje na stránku s přehledným výpisem položek a informací o zakázce. Tento výpis bývá použit pro tvorbu faktur. Dalším tlačítkem je tlačítko Upravit, které změní stránku na stejný formulář, jako je formulář pro tvorbu zakázky, jen s načtenými daty. Posledním tlačítkem příslušejícím k zakázce je tlačítko k odstranění, kdy po stisknutí a potvrzení proběhne odstranění zakázky ze systému.

Jak už bylo zmíněno v sekci pojednávající o aplikačním rozhraní a položkách ukládaných o zakázkách, vidíme, že množství položek plánovaných v návrhu bylo sníženo a nahrazeno položkou pro poznámku. Tento fakt pochází z testování aplikace na mobilních zařízeních, kdy množství položek, které uživatel mnohdy nevyplnil, zabíralo velké množství prostoru. Což se na menším displeji mobilního telefonu, kde je třeba místem šetřit, ukázalo jako problém. Z testování také vzešlo zjištění, že vyžadovat určitý typ dat v položce není výhodou. Konkrétně jde o položku cena, kdy uživatel potřebuje zadat k ceně poznámku, což v momentě, kdy vyžadujeme striktně číslo, nelze.

Zároveň původně odlišné množství formulářových položek v desktopové a mobilní verzi se ukázalo jako matoucí a uživatelsky nepřívětivé. Uživatel totiž z plného zobrazení na



Obrázek 7.4: Seznam zakázek - mobilní zařízení

počítači věděl o jistých položkách, které v malém množství případů měl zájem vyplnit. Což ovšem v případě, kdy se na mobilním telefonu nezobrazovaly a nebyly dostupné, mělo za vliv pocit nekonzistence prostředí. I v tomto případě došlo k nahrazení položek položkou `Poznámka`. Toto řešení se při testování ukázalo jako skvělý kompromis.

Možným plánovaným rozšířením, které zlepší uživatelský zážitek aplikace, je přidávání fotografii k zakázkám, na což je aplikační rozhraní připraveno. Bohužel kvůli změnám v implementaci získávaní dat od výrobce nebylo toto rozšíření dosud implementováno. Toto rozšíření vyžaduje napojení aplikace na rozhraní poskytovatelů úložiště fotografii, jedním z nich může být například služba Google Fotky[4]

7.3 Testování

Aplikace byla testována se servisním technikem, který se v tomto oboru pohybuje již přes 30 let. Disponuje zkušenostmi se servisem od malých domácích plynových kotlů až po vysokovýkonové hořáky v kotelnách zabezpečujících teplo a teplou vodu pro řadu domácností.

Uživatel velmi ocenil možnost neomezené tvorby zakázek, přidávání neomezeného množství položek k zakázkám a obecně vysokou volnost v aplikaci, kterou tak může snadno za-

[4]Webová stránka služby Google Fotky: https://photos.google.com/

členit do svého pracovního procesu. Synchronizace dat mezi zařízeními plynoucí z použití databáze na serveru (nikoli lokální ukládání dat například v telefonu) patřila také mezi pozitivně hodnocené vlastnosti aplikace.

Uživatel by ocenil možnost napojení více výrobců, jelikož načítání dat od výrobce Viessmann bylo označeno za velké usnadnění práce. Toto je však limitováno jednotlivými výrobci. Možnost kalendáře s notifikacemi pro jednotlivé zakázky by patřila také mezi vítané rozšíření.

Kapitola 8

Závěr

Cílem práce bylo navrhnout a vytvořit informační systém na správu zakázek pro servisní techniky plynových kotlů. Při realizaci tohoto úkolů jsme se potýkali s řadou výzev, jako byl návrh rozhraní pro osobní počítač i pro mobilní telefon s důrazem na ovladatelnost. Dalším úkolem bylo vyřešit problematiku implementace, a to konkrétně výběr vhodných nástrojů jako jsou programovací jazyky, jejich rozšíření a také prostředky k běhu aplikace. Posledním z větších úkolů bylo vyřešit propojení aplikace s rozhraním výrobce pro získávání dat o náhradních dílech.

Výsledné řešení bylo rozděleno na dvě části, část uživatelského rozhraní a část aplikační komunikující s rozhraním výrobce. K implementaci aplikační části bylo použito rozšíření jazyka Python nazvané FastAPI. Toto rozšíření je vytvořeno čistě pro tvorbu těchto rozhraní a jejich snadné nasazení. Aplikační část využívá DynamoDB pro ukládání dat. Pro tvorbu uživatelského rozhraní bylo použito aplikačního rámce pro JavaSript - Vue.js s kaskádovými styly Bootstrap. Obě tyto aplikace spolu komunikují přes rozhraní typu REST. Celé řešení bylo nasazeno u poskytovatele Amazon na jejich službě AWS. Konkrétně aplikační rozhraní na službě Lambda a uživatelské rozhraní je publikováno na úložišti S3. Pro šifrovanou komunikaci HTTPS je využito služby CloudFront od stejného poskytovatele. Aplikace byla otestována a většina podnětů od uživatelů aplikace byla implementována.

Mezi plánovaná rozšíření patří přidání ukládání fotografii, což bohužel nebylo implementováno kvůli změnám v komunikaci s daty výrobce. Aplikační rozhraní je však na toto rozšíření v budoucnosti plně připraveno. Zajímavým rozšířením může být také implementace kalendáře pro správu událostí k jednotlivým zakázkám. Dalším rozšířením aplikace může být připojení více katalogů od jednotlivých výrobců, což ale závisí od poskytnutí těchto dat výrobci. Při plném nasazení aplikace by také bylo vhodné nakonfigurovat a implementovat použití služby Cognito, kterou poskytuje Amazon na svém AWS, pro správu přihlašování uživatelů.

Literatura

[1] *Desktop vs Mobile vs Tablet Market Share Europe* [[online]]. Naposledy navštíveno 12. 12. 2022. Dostupné z: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-201111-201611.

[2] *Top Web Design Trends for 2023* [[online]]. Naposledy navštíveno 13. 12. 2022. Dostupné z: https://www.theedigital.com/blog/web-design-trends.

[3] *The Thumb Zone: Designing For Mobile Users* [[online]]. Naposledy navštíveno 15. 12. 2022. Dostupné z: https://www.smashingmagazine.com/2016/09/the-thumb-zone-designing-for-mobile-users/.

[4] *What Is Oracle Database* [[online]]. Naposledy navštíveno 2. 1. 2023. Dostupné z: https://www.oracletutorial.com/getting-started/what-is-oracle-database/.

[5] *ABZ slovník českých synonym* [[online]]. Naposledy navštíveno 27. 11. 2022. Dostupné z: https://www.slovnik-synonym.cz/web.php/slovo/zakazka.

[6] *Mobile UX Essentials* [[online]]. Naposledy navštíveno 28. 12. 2022. Dostupné z: https://www.lukew.com/ff/entry.asp?1259.

[7] *UI Design and Interaction Guide for Windows Phone 7* [[online]]. Naposledy navštíveno 28. 12. 2022. Dostupné z: http://tableless.github.io/exemplos/pdf/guidelines-interface-mobiles/UI%20Design%20and%20Interaction%20Guide%20for%20Windows%20Phone%207%20v2.0.pdf.

[8] *Veřejné širokopásmové mobilní sítě 4G a 5G* [[online]]. Naposledy navštíveno 28. 12. 2022. Dostupné z: https://digi.ctu.cz/pokryti/.

[9] *File Structure* [[online]]. Naposledy navštíveno 29. 12. 2022. Dostupné z: https://reactjs.org/docs/faq-structure.html.

[10] *The Most Popular JavaScript Frameworks of 2022* [[online]]. Naposledy navštíveno 29. 12. 2022. Dostupné z: https://www.makeuseof.com/most-popular-javascript-frameworks/.

[11] *Top 7 Backend Programming Languages in 2022* [[online]]. Naposledy navštíveno 29. 12. 2022. Dostupné z: https://nexttechnology.io/top-7-backend-programming-languages-in-2022/.

[12] *Understanding Component-Based Architecture* [[online]]. Naposledy navštíveno 29. 12. 2022. Dostupné z: https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238.

[13] *Why is React Declarative? A Story About Function Components* [[online]]. Naposledy navštíveno 29. 12. 2022. Dostupné z: https://medium.com/trabe/why-is-react-declarative-a-story-about-function-components-aaae83198f79.

[14] *Most Popular Databases In The World (2023)* [[online]]. Naposledy navštíveno 30. 12. 2022. Dostupné z: https://www.c-sharpcorner.com/article/what-is-the-most-popular-database-in-the-world/.

[15] *Results for js web frameworks benchmark – round 8* [[online]]. Naposledy navštíveno 30. 12. 2022. Dostupné z: https://stefankrause.net/js-frameworks-benchmark8/table.html.

[16] *Desktop vs Mobile vs Tablet Market Share Europe* [[online]]. Naposledy navštíveno 5. 12. 2022. Dostupné z: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/europe/#monthly-202111-202211.

[17] Araújo, P. M. de. *Django introduction* [[online]]. Naposledy navštíveno 7. 5. 2023. Dostupné z: https://pemtajo.medium.com/an-overview-about-hash-functions-theory-and-security-21e52ddc9993.

[18] Beaird, J., Walker, A. a George, J. *The principles of beautiful web design*. Sitepoint, 2020.

[19] Boss, G., Malladi, P., Quan, D., Legregni, L. a Hall, H. Cloud computing. *IBM white paper*. ACADEMIA. 2007, sv. 321, s. 224–231.

[20] Buckland, M. K. *Information and information systems*. ABC-CLIO, 1991.

[21] Checkland, P. a Scholes, J. *Soft systems methodology in action*. John Wiley & Sons, 1999.

[22] Gong, Y., Gu, F., Chen, K. a Wang, F. The Architecture of Micro-services and the Separation of Frond-end and Back-end Applied in a Campus Information System. In: IEEE. *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*. 2020, s. 321–324.

[23] Ivory, M. Y. a Megraw, R. Evolution of web site design patterns. *ACM Transactions on Information Systems (TOIS)*. ACM New York, NY, USA. 2005, sv. 23, č. 4, s. 463–497.

[24] Kagan, A., Lau, K. a Nusgart, K. R. Information system usage within small business firms. *Entrepreneurship Theory and Practice*. SAGE Publications Sage CA: Los Angeles, CA. 1990, sv. 14, č. 3, s. 25–38.

[25] Kolář, D. *Principy programovacích jazyků objektově orientovaného programování IPP – I*. FIT VUT, 2006.

[26] Rainer, R. K. a Prince, B. *Introduction to information systems*. John Wiley & Sons, 2021.

[27] Saini, H., Upadhyaya, A. a Khandelwal, M. K. Benefits of cloud computing for business enterprises: A review. In: *Proceedings of International Conference on Advancements in Computing & Management (ICACM)*. 2019.

[28] Šarmanová, J. *Databázové a informační systémy.* VŠB, 2008.

[29] Sklenák, V. *Data, informace, znalosti a Internet.* Nakladatelství CH Beck, 2001.

[30] Wroblewski, L. *Mobile first.* A Book Apart, 2011.

[31] Zendulka, J. a Rudolfová, I. *Databázové systémy.* FIT VUT, 2006.

[32] Zhang, M. *Top 10 Cloud Service Providers Globally in 2022* [[online]]. Naposledy navštíveno 27. 11. 2022. Dostupné z: https://dgtlinfra.com/top-10-cloud-service-providers-2022/.

[33] Zhang, P. a Von Dran, G. M. Satisfiers and dissatisfiers: A two-factor model for website design and evaluation. *Journal of the American society for information science.* Wiley Online Library. 2000, sv. 51, č. 14, s. 1253–1268.

[34] ČR, P. *Zákon o veřejných zakázkách* [[online]]. Naposledy navštíveno 27. 11. 2022. Dostupné z: https://www.oziveni.cz/wp-content/uploads/2012/04/zakon_137_2006.pdf.