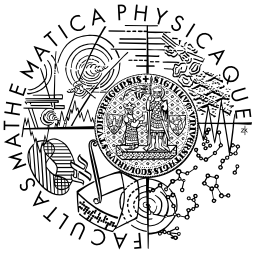


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Milan Abrahám

Internetový obchod se službami státu

Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D.
Studijní program: Informatika
Studijní obor: IPP2

Praha 2022

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Rád bych poděkoval vedoucímu práce doc. Mgr. Martinovi Nečaskému, Ph.D. za návrh zajímavého tématu, konzultace a pomoc při vypracování práce.

Název práce: Internetový obchod se službami státu

Autor: Milan Abrahám

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D., Katedra softwarového inženýrství

Abstrakt: V současné době nejsou v jednotné podobě prezentovány občanům služby poskytované státem. Registr práv a povinností obsahuje data o všech nabízených službách. Cílem této práce je vytvoření funkční návrhu internetového obchodu určeného pro občany. Obchod by kromě vyhledávání a doporučování měl také umožnit vyřizování služeb a komunikaci s úřady. Práce popisuje registr a jeho účel. Dostupná data jsou analyzována. Těch není dostatek pro správné fungování obchodu, je tedy navrženo jejich vhodné doplnění. Komunikace s úřady není dosud sjednocená, práce představuje nový způsob pomocí formulářů. Ty jsou generovány dynamicky, podle navržené datové struktury. Je popsán návrh architektury, vývoj internetového obchodu a jeho následné testování. V závěru jsou shrnuty možnosti případného rozšíření.

Klíčová slova: otevřená data, služby státu, webová aplikace

Title: Internet shop for government services

Author: Milan Abrahám

Department: Department of Software Engineering

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., Department of Software Engineering

Abstract: Services provided by the state are currently not being presented to citizens in a uniform way. The Register of Rights and Obligations (Registr práv a povinností) contains data about all provided services. The goal of this thesis is the creation of a functional internet eshop prototype, targeted to citizens. In addition to searching and recommending, the eshop shall also allow the ordering of services and communication with authorities. This thesis describes the register and its purpose. The available data is analyzed. There are not enough of them for a proper functionality of some parts, so their adjustment is suggested. Communication with authorities isn't unified, this work presents a new way, using forms. They are generated dynamically, according to a designed data structure. The architecture and development of the eshop are described, including a testing process. The possibilities of future expansion are summarized.

Keywords: open data, government services, web application

Obsah

Úvod

Registr práv a povinností (RPP) obsahuje údaje o orgánech veřejné moci a službách, které poskytují. V současné době nejsou tyto informace jednotným a přehledným způsobem prezentovány občanům a firmám. Data jsou dostupná přes Portál otevřených dat ve strojově zpracovatelných formátech. Tyto formáty jsou vhodné pro odbornou veřejnost, nikoliv pro běžné občany či firmy.

Jedním existujícím řešením, které data poskytuje je Portál veřejné správy[1]. V sekci Služby veřejné správy lze najít kategorie služeb. Nelze v nich ovšem snadno vyhledávat a portál neobsahuje žádnou personalizaci. U služeb je přítomen detailní popis, chybí ale informace o jednotlivých úkonech.

Cílem práce je prezentovat nabídku služeb v podobě internetového obchodu. Ten občanům kromě poskytování informací umožní i jejich vyřízení. Nebudou chybět běžné funkce obchodů jako vyhledávání a doporučování. Uživateli bude umožněno přihlášení elektronickou identitou, v této práci bude simulováno.

Komunikace s úřady bývá nejednotná, mnohdy ani není umožněna v elektronické podobě. Momentálně má každá služba formuláře buďto fyzické nebo v podobě velkých aplikací. Práce se snaží o navržení nového konceptu. Aplikace uživatele provede jednotlivými úkony. Umožní mu vyplňovat formuláře, jenž následně odešle úřadu. Formuláře budou generované dynamicky. Je třeba navrhnout jaká data k tomu budou využita a také jejich strukturu.

Práce krátce popisuje fungování eGovernmentu a účel Registru práv a povinností. Poté jsou analyzovány požadavky na aplikaci a jsou přetvořeny do podoby uživatelských příběhů. Následně jsou rozebrána všechna dostupná data. Kromě RPP byl objeven další zdroj dat poskytující detailní popisy služeb. Je znázorněna struktura dat a popsán způsob jejich získávání. Pro některé vyžadované funkce obchodu ale není dat dostatek. Součástí je tedy i návrh jejich doplnění.

Dále jsou navrženy jednotlivé stránky a funkce internetového obchodu. Je rozmyšleno, jakým způsobem bude provedeno doporučování a vyhledávání služeb. Přítomen je i návrh backendu a celkově celé aplikace.

V implementační části je popsán vývoj aplikace. Moduly aplikace jsou detailně rozebrány a je vysvětleno jejich fungování včetně použitých technologií. Na závěr je popsán způsob testování a shrnuty jeho výsledky.

[1] Dostupný na adrese `https://portal.gov.cz/sluzby-verejne-spravy/`

1. Registr práv a povinností

1.1 eGovernment v ČR

Komunikace s úřady může být mnohdy komplikovaná, časově náročná a zastaralá. Občané často musí vystát několik front a vyplnit nespočet dokumentů než docílí splnění jejich požadavků. Potřebné informace jsou rozptýlené v několika zdrojích, někdy nejsou vůbec dostupné elektronicky.

Jako příklad si vezměme vydávání cestovního pasu. Pokud chceme zažádat o cestovní pas, jistě bychom chtěli předem znát, jaké k tomu potřebujeme dokumenty a údaje. Dále by bylo dobré vědět, zda je potřeba vyplnit nějaký formulář a s kterým úřadem musíme komunikovat. Bez využití internetu by se nám tyto informace zjišťovali vcelku složitě. Museli bychom se pravděpodobně zajít dotázat na místní obecní úřad.

Většina z nás už dnes internet využít může a díky vyhledávačům se nám tyto informace budou hledat výrazně snadněji. Jednou ze stránek, kterou můžeme nalézt může být třeba Portál veřejné správy, sekce Služby veřejné správy[1]. Ten obsahuje potřebné informace, ale kompletně chybí jednotlivé úkony. Tento portál, jež je součástí eGovernmentu, provozuje Ministerstvo vnitra České republiky (MVČR).

eGovernment neboli electronic government lze přeložit jako elektronická vláda. Lidinský [1] jej definuje následovně: „eGovernment je využívání informačních technologií veřejnými institucemi pro zajištění výměny informace s občany, soukromými organizacemi a jinými veřejnými institucemi za účelem zvyšování efektivity vnitřního fungování a poskytování rychlých, dostupných a kvalitních informačních služeb.“

1.2 Základní registry České republiky

Jak zmiňuje Doleček [2], již od konce minulého století bylo usilováno o vybudování systému sdílených dat. Data byla dříve uchovávána v různých systémech, mnohdy i duplicitně a v rozdílně kvalitě. Z potřeby tato data sjednotit a využít k tomu současné technologie vznikly základní registry. Registr je informační systém, který je schopný automatizované komunikace přes určité rozhraní a řídí přístup uživatelů pomocí oprávnění [1]. Funkčností v podstatě odpovídá databázi.

Podle MVČR [3] jsou registry čtyři a patří mezi ně:

- Registr obyvatel
- Registr osob
- Registr územní identifikace, adres a nemovitostí
- Registr práv a povinností

Součástí je také Informační systém základních registrů, což je soubor nástrojů pro práci s registry a komunikaci mezi subjekty. Registry jsou řízeny Správou

[1] `https://portal.gov.cz/sluzby-vs/S315`

základních registrů, která data ověřuje a zpracovává. Vzniku registrů předcházel Zákon o základních registrech [4], jenž vymezuje jejich obsah a související práva a povinnosti pro práci s daty.

1.3 Registr práv a povinností

Dle informací Kaliny [5] je celý název registru Základní registr agend, orgánů veřejné moci, soukromoprávních uživatelů údajů a některých práv a povinností. Dále budeme tento registr označovat jako RPP. Pojmy v názvu registru definuje zákon [4] následovně:

Agenda

„Ucelená oblast působení orgánu veřejné moci nebo ucelená oblast působení soukromoprávního uživatele údajů.“

Jako příklady agend můžeme uvést ochranu přírody a krajiny nebo agendu o podmínkách provozu vozidel na pozemních komunikacích.

Orgán veřejné moci (OVM)

„Státní orgán, územní samosprávný celek, fyzická nebo právnická osoba, byla-li jí svěřena působnost v oblasti veřejné správy, notář, soudní exekutor a archiv.“

OVM je například obec s rozšířenou působností, Státní veterinární správa či hlavní město Praha.

Soukromoprávní uživatel údajů

„Podnikající fyzická osoba nebo právnická osoba, která je podle jiného právního předpisu oprávněna využívat údaje ze základního registru nebo z agendového informačního systému.“

Hlavním obsahem jsou data o agendách. Lze tedy zjistit jaké agendy existují, které orgány veřejné moci je vykonávají, z jakých se skládají činností a další relevantní informace. Dále také obsahuje informace o právech přístupu k ostatním registrům. Tedy pokud by někdo chtěl přistoupit k datům v registru nebo je editovat, RPP posoudí, zda má dostatečná oprávnění. Využívá k tomu matici oprávnění, která je z registru každý den generována. Nalezneme v něm také informace o rozhodnutích, která vedla ke změně údajů v jiných registrech. Lze tedy zjistit, kým a proč byla jakákoliv data změněna. Správcem registru je MVČR. (Kalina [5])

1.4 Služba veřejné správy

Cílem této práce je poskytovat informace a možnost vyřízení služeb veřejné správy. Službou veřejné správy se rozumí činnost úřadu poskytující klientovi nějakou hodnotu. Může se jednat o splnění povinnosti nebo uplatnění práva klienta. Klientem se rozumí jakákoliv fyzická nebo právnická osoba. Nejsou evidovány služby, ve kterých dochází pouze ke komunikaci orgánu veřejné moci navzájem. (MVČR [6])

2. Analýza požadavků

2.1 Požadavky

Na počátku práce proběhlo několik schůzek s vedoucím práce a panem Ing. Ondřejem Felixem CSc. Z nichž vzešly následující požadavky na průběh a funkcionalitu projektu:

- Aplikace bude simulovat přihlášení pomocí elektronické identity. Bude obsahovat nultou stránku, kde bude na výběr z několika typů uživatelů.
- Hlavní stránka bude obsahovat stručný přehled aktivních služeb a možnost přejít na stránku s kompletní historií služeb uživatele.
- Hlavní stránka umožní vyhledávání služeb. Také budou zobrazeny služby doporučené pro přihlášeného uživatele.
- Služby budou nabízeny podobně jako v internetových obchodech.
- Hlavní stránka bude obsahovat jen stručný popis služby. Pro více informací uživatel přejde na stránku s detailem služby.
- Na stránce s detaily služby si bude moci uživatel objednat vyřízení.
- Budou využity dva zdroje dat. Prvním jsou data z RPP, ty jsou ale nedostatečná pro objednávání. Druhým zdrojem tedy budou data manuálně doplněná o údaje potřebné k objednávání.
- Uživatel u aktivního úkonu vyplní vygenerovaný formulář, data se doplní do dokumentu. Obojí se odešle úřadu.
- Úkony, které vyžadují akci úřadu, se budou simulovat a provedou se automaticky.

2.2 Uživatelské příběhy

Cílem uživatelských příběhů je naznačit interakci uživatele s aplikací. Příběhy byly nápomocny při následném návrhu jednotlivých stránek a jejich uživatelského rozhraní.

1. Vyhledání informací o službě

Cíl: Uživatel si potřebuje zjistit informace o předem známé službě.

Příklad: Uživatel chce rybářský lístek, neví jaké kroky učinit.

Kroky:

- Otevření hlavní stránky
- Zadání názvu (nebo části názvu) služby do vyhledávacího pole

- Vybrání služby z výsledků vyhledávání
- Zobrazení informací o službě

Alternativní průběh:

- Žádné výsledky nenalezeny - uživatel je informován.

2. Objednání vyřízení služby

Cíl: Uživatel si chce objednat vyřízení předem známé služby.

Příklad: Uživatel si chce požádat o povolení ke kácení dřevin.

Kroky:

- Otevření hlavní stránky
- Zadání názvu služby do vyhledávacího pole
- Otevření stránky služby
- Kliknutí na tlačítko „Vyřídit“

Alternativní průběh:

- Službu nemůže iniciovat klient - uživateli není umožněno objednání a je o tom informován.

3. Zobrazení objednaných služeb

Cíl: Uživatel si chce zobrazit služby, které má objednané.

Příklad: Uživatel si objednal službu, chce se ujistit, že je skutečně objednaná.

Kroky:

- Otevření hlavní stránky
- Kliknutím na tlačítko přechod na objednané služby
- Zobrazení všech aktivních služeb

4. Zobrazení stavu vyřizované služby

Cíl: Uživatel chce zjistit v jakém stavu se nachází objednaná služba.

Příklad: Uživatel se chce podívat, zda už mu úřad schválil žádost.

Kroky:

- Otevření hlavní stránky
- Kliknutím na tlačítko přechod na objednané služby
- Nalezení dané služby v Otevřených službách

Alternativní průběh:

- Byl dosažen koncový krok, v tom případě je služba přesunuta do již dokončených.

5. Vyřizovaná služba vyžaduje akci uživatele

Cíl: Uživatel má již objednanou službu, čeká se na jeho akci.

Příklad: Uživatel si objednal žádost o povolení ke kácení dřevin. Je potřeba, aby vyplnil formulář.

Kroky:

- Služba byla objednána (příběh č. 2)
- Uživatel má otevřenou stránku s objednanými službami (příběh č. 3)
- Zobrazení konkrétní služby
- V části "Další krok„ bude zobrazen formulář
- Některá data mohou být předvyplněná
- Vyplnění formuláře uživatelem
- Zobrazení vyplněného formuláře - strukturovaná data i vyplněný dokument
- Potvrzení odeslání

Alternativní průběh:

- Uživatel vyplnil údaje špatně - je mu umožněno znovu formulář vyplnit.

6. Uživatel dostal rozhodnutí o službě od úřadu

Cíl: Uživatel si zobrazí rozhodnutí úřadu.

Příklad: Uživatel si objednal žádost o povolení ke kácení dřevin. Úřad žádost schválil. Chce si výsledek zobrazit.

Kroky:

- Předcházel příběh č. 2 (objednání) a č. 5 (akce uživatele)
- Zobrazení konkrétní služby
- V posledním kroku se zobrazí rozhodnutí/zpráva od úřadu
- Pravomocné rozhodnutí přijde až do datové schránky/poštou
- Přesunutí služby do dokončených

7. Vyřizovaná služba čeká na akci úřadu

Cíl: Uživatel má již službu objednanou, čeká se na jeho akci.

Příklad: Uživatel si objednal žádost o povolení ke kácení dřevin. Odeslal formulář a čeká na oznámení o rozhodnutí.

Kroky:

- Předcházelo objednání (příběh č. 2)

- Zobrazení konkrétní služby
- Krok se zobrazí jako nevyřízený
- Zobrazí se informace o tom, který úřad službu zajišťuje

V pozdější fázi práce bylo rozhodnuto, že úkony vyžadující akci úřadu se budou provádět automaticky ihned po odeslání akce klienta. Tedy příběh č. 7 byl vyřazen.

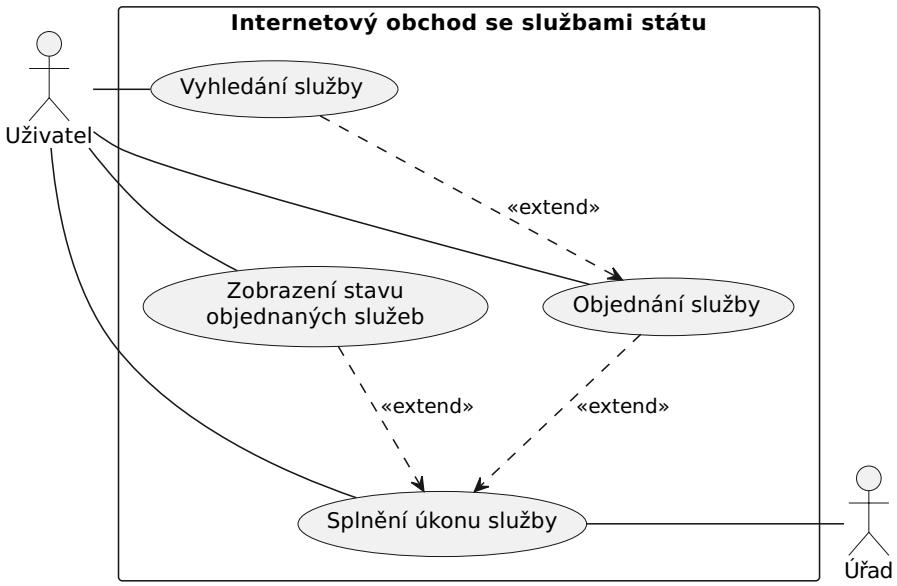
2.3 Use case diagram

Use cases (případy užití) mají reprezentovat akci, která má nějaký výstup pro zúčastněné. V této práci jsou zúčastněnými pouze uživatel a případně úřad vyřizující služby. Jako akce s nějakým výsledkem můžeme definovat následující:

- Vyhledání služby
- Objednání služby
- Zobrazení stavu objednaných služeb
- Splnění úkonu služby

UML use case diagram můžeme vidět na obrázku 2.1. Byl vytvořen pomocí nástroje PlantUML[1]. Zachycuje i vztahy mezi akcemi.

Vztah „extend“ mezi dvěma akcemi znamená, že jedna z nich může být součástí té druhé. Šipka vede z akce, jež může stát samostatně, do té na ní závislé. Například vyhledání služby může být samostatná akce, zatímco objednat službu nelze bez jejího objednání.



Obrázek 2.1: Use case diagram

[1]Nástroj dostupný jako webová aplikace na `https://plantuml.com/`

3. Analýza dat

3.1 Otevřená data z RPP

Data z Registru práv a povinností jsou publikována v Národním katalogu otevřených dat, do kterého lze přistoupit přes Portál otevřených dat[1]. Tato práce bude konkrétně pracovat s datovou sadou Služby veřejné správy. Data jsou poskytovány ve dvou distribucích, přičemž obě by měly poskytovat identická data.

První distribucí je soubor ve strojově zpracovatelném formátu JSON, případně JSON-LD (JSON for Linking Data). JSON-LD je nadstavbou formátu JSON a umožňuje lepší propojení dat z různých zdrojů [7].

Druhou distribucí dat je formát RDF (Resource Description Framework), nad kterým se lze dotazovat jazykem SPARQL [8]. RDF spojuje data do trojice objekt, predikát a subjekt. Tedy data jsou propojena do grafové podoby.

Následuje výběr poskytovaných dat, které jsou pro práci relevantní. Popisy vlastností jsou parafrázované z dokumentace datové sady [9]. Bohužel není garantována dostupnost těchto dat. Všechny vlastnosti jsou označené jako nepovinné, tedy v datech se nemusí vyskytovat.

Služba

identifikátor Číselný identifikátor tvořený písmenem `S` následovaným číslicemi.

název Výstižný, nezaměnitelný název.

popis Krátký ale přesný popis služby. Po přečtení názvu a popisu by klientovi mělo být hned jasné, zda je pro něj služba relevantní.

typ služby Určuje, zda je služby iniciovaná klientem nebo vykonávaná z moci úřední.

klienti Typ oprávněných klientů služby. (např. fyzická osoba)

agenda Služby lze kategorizovat podle agendy, do které patří.

místní příslušnost Určuje orgán veřejné moci, který službu vykonává. Obsahuje typ subjektu (právnická/fyzická osoba) a název vykonavatele.

úkony Seznam úkonů, ze kterých se služba skládá.

Uživatel se při vyhledávání bude rozhodovat podle názvu služby. Ten ale nemusí být vždy výstižný či snadno pochopitelný, proto je potřeba ho doplnit popisem. Tedy při zobrazení výsledků vyhledávání nebo při doporučování budou primárně zobrazené tyto dvě informace.

[1] Portál otevřených dat je dostupný na `https://data.gov.cz/`

Další informace poté budou dostupné až v detailu služby. Typ služby je důležitý, měli bychom klientovi zamezit objednání služby, kterou nemá právo inicializovat.

Místní příslušnost je relevantní hlavně při nutnosti osobní přítomnosti. Uživatel potřebuje vědět, který orgán veřejné moci musí navštívit. Cílem této práce je ukázat možnost přesunutí komunikace do online prostředí, pomocí datové schránky. Tedy přímé lokalizační údaje, například adresu, nepotřebujeme. Ale samozřejmě se stále hodí vědět, jaký orgán je na druhé straně komunikace.

Úkon služby

Každá služba se skládá z alespoň jednoho úkonu. Úkon představuje jednotnou interakci mezi klientem a orgánem veřejné moci (OVM). Postupným plněním úkonů je usilováno o dosažení výstupu služby.

identifikátor Číselný identifikátor tvořeným písmenem *U* následovaným číslicemi.

název, popis Stejně jako u služby slouží klientovi k rychlému získání informací o úkonu.

typ vykonavatele Iniciátor služby. (klient nebo OVM)

fáze Časové zařazení úkonu v sérii úkonů služby. (začáteční, prostřední nebo koncová)

digitální Uvádí zda je úkon možné provést digitálně.

kanály Obslužné kanály, kterými je možné službu vyřídit.

Kromě názvu a popisu úkonu je pro nás podstatný typ vykonavatele. Rozlišuje, zda úkon vykonává klient nebo OVM. Fáze určuje pořadí úkonů. Bohužel jsou fáze jen tři; počáteční, prostřední a koncová.

Zda je úkon digitální je klíčová informace. Pokud není, pak objednávání služby online postrádá smysl. „Offline“ úkony by bylo možné zavést, to by poté ale bylo potřeba navrhnout celý nový systém, jenž by umožnil úřadům takové úkony spravovat.

Obslužný kanál

Popisuje kanál, kterým dochází ke komunikaci klienta a OVM při vyřizování úkonu. Kanálem může být například datová schránka, pošta, ale i osobní vyřízení.

realizován Zda je obslužný kanál realizován. Pokud není realizován může být plánový. Může být dostupný i datum plánovaného spuštění či zrušení.

typ kanálu Typ obslužného kanálu. (např. datová schránka)

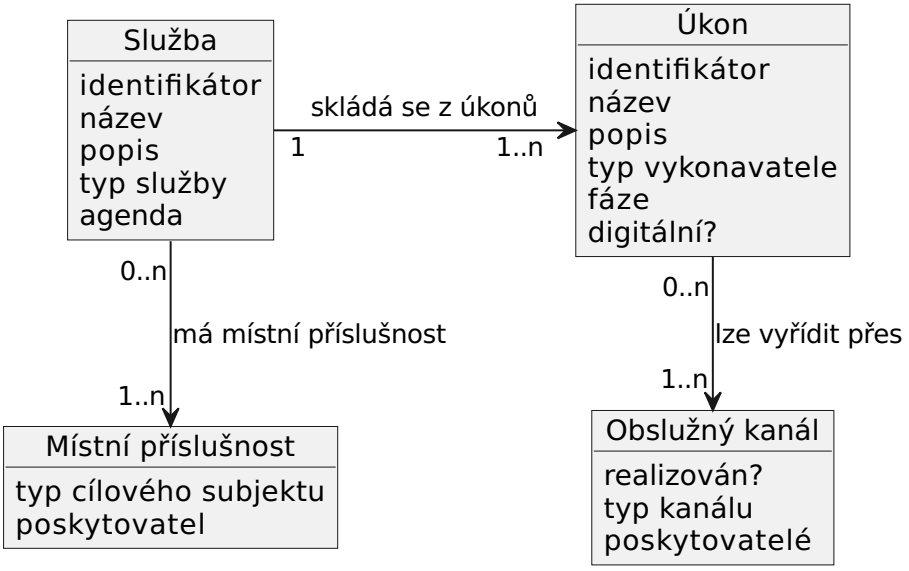
poskytovatelé OVM nebo kategorie poskytující obslužný kanál.

Z komunikačních kanálů nás zajímá pouze datová schránka, komunikaci budeme simulovat přes ní.

3.1.1 Konceptuální diagram

Pro naznačení vztahu a organizace dat byl vytvořen UML class diagram (diagram tříd) v nástroji PlantUML. Jak můžeme vidět na obrázku 3.1, data lze rozdělit do čtyř tříd. Diagram naznačuje i asociace mezi třídami. Služba má jednu a více místních příslušností. Služba se skládá z jednoho či více úkonů. Úkon lze vyřídit alespoň přes jeden obslužný kanál.

Zajímavý je vztah mezi úkonem a službami, ke kterým může patřit. Jeden úkon totiž může patřit jen a pouze k jedné službě. I pokud má více služeb identický úkon, v datové sadě jsou považovány za rozdílné.



Obrázek 3.1: Konceptuální model dat

3.1.2 SPARQL dotazování

Následuje ukázka SPARQL dotazu, který vrátí informace o konkrétní službě podle jejího identifikátoru. Jak již bylo zmíněno, SPARQL spojuje data do trojic. Trojice se skládá z subjektu, predikátu a objektu. Subjekt je propojen s objektem pomocí predikátu/vlastnosti. Tyto vlastnosti jsou pevně definované v RDF slovnících.

Ukázka SPARQL dotazu

```
PREFIX l-pojem:
<https://slovník.gov.cz/legislativní/sbírka/111/2009/pojem/>

PREFIX a-pojem:
<https://slovník.gov.cz/agendový/104/pojem/>

SELECT *
WHERE {
    ?Sluzba a l-pojem:služba-veřejné-správy .

    ?Sluzba a-pojem:má-identifikátor-služby ?ID .
    ?Sluzba a-pojem:má-název-služby ?Nazev .
    ?Sluzba a-pojem:má-popis-služby ?Popis .
    ?Sluzba a-pojem:má-typ-služby ?Typ .
    ?Sluzba a-pojem:je-poskytována-v-rámci-agendy ?Agenda .

    FILTER (STR(?ID) = "S7222")

} LIMIT 1
```

První dva příkazy **PREFIX** zavádějí aliasy pro slovníky používané v dotazu. Příkaz **SELECT** určuje, jaké proměnné dotaz vrátí, v tomto případě úplně všechny. V bloku **WHERE** poté následují podmínky pro zvolení požadovaného objektu a vrácených dat. Dotazy jsou také prováděny pomocí trojic. První trojice specifikuje typ objektu, který bude uložen do proměnné *?Sluzba*. Tedy v tomto případě je to objekt typu *služba-veřejné-správy*. V následujícím **OPTIONAL** bloku jsou poté definované potřebné vlastnosti. Následuje příkaz **FILTER**, jenž nám vybere konkrétní službu podle specifikovaného identifikátoru. Na závěr je specifikován počet vrácených výsledků.

Odpověď je vrácena ve formátu JSON. V ní jsou data spojena právě s proměnnými, které jsou v dotazu zadefinovány a vybrány příkazem **SELECT**.

3.2 Data z Katalogu služeb

Při zjišťování informací o RPP byla na stránkách o architektuře eGovernmentu ČR [6] objevena zmínka o API ke Katalogu služeb. Jejím primárním účelem je poskytování detailních popisů služby. Tato data jsou využita na Portálu veřejné správy [10]. Dotazy jsou psány v jazyce GraphQL a obdržená data jsou ve formátu JSON. K API neexistuje žádná dokumentace a je o něm velmi málo dostupných informací. Navíc je zmíněno, že v budoucnu bude API přesunuto na jinou adresu a přístup bude omezen [6].

Přesto jsou poskytované popisy služeb natolik užitečné, že bylo rozhodnuto jejich zakomponování do aplikace. Nejsou sice dostupné pro všechny služby, ale mají velkou informační hodnotu. Aplikace bude muset být připravená na možnost výpadku API z výše zmíněných důvodů. Na datech by neměli záviset žádné důležité funkce. Pokud data nejsou dostupná, aplikace by neměla vykreslit s nimi související prvky.

Jak bylo zmíněno, neexistuje žádná dokumentace. Jediná dostupná informace je schéma API. To popisuje strukturu a název vlastností ve vracených dat. Aplikace využívá následující vlastnosti:

charakter	Velmi detailní popis služby.
jakyMaSluzbaBenefit	Benefit služby, její možné výsledky.
kdyJednat	V jaké situaci je službu potřeba vyřídit. Popisuje cílové klienty a důvody.
kdyVecResit	Kdy je potřeba službu řešit. Obsahuje časové údaje a limity.

3.2.1 GraphQL dotazování

Následující dotaz je využíván pro získáních potřebných dat pro konkrétní službu. V případě GraphQL se při definici dotazu určuje rovnou struktura odpovědi.

Vstupem je identifikátor služby v proměnné *$kod*. *detailniPopisSluzbyVerejneSpravy* je název struktury. Filtr zaručí nalezení požadované služby podle identifikátoru. A následuje už jen výpis požadovaných vlastností.

Odpovědí jsou data ve formátu JSON ve struktuře popsané v dotazu.

Ukázka GraphQL dotazu

```
query DetailySluzby($kod: String) @lang(lang: CS) {
    detailniPopisSluzbyVerejneSpravy(filter: { kod: $kod }) {
        kod
        nazev
        charakter
        jakyMaSluzbaBenefit

        kdyJednat
        kdyVecResit
    }
}
```

3.3 Využitelnost a doplnění dat

Otevřených dat je poskytováno dostatek na to, aby si uživatel vytvořil představu o účelu služby. Doplňující popisy z katalogu dat pak ještě více specifikují typ cílového uživatele a další podmínky o tom, kdy službu vyřizovat.

Nedostatky se nacházejí u úkonů služeb. Rozlišeny jsou úkony počáteční a koncové. To je ovšem nedostatečné pro určení správného pořadí úkonu při vyřizování služby. Pořadí lze odhadnout z popisů, to však není strojově zpracovatelná informace. Z úkonů je třeba vytvořit graf průběhu. K tomu by stačilo, aby každý úkon měl údaj o svých předcích nebo následnících. Pro správnou funkci simulace vyřizování služby bude potřeba do některých služeb pořadí doplnit.

Další data, jež je potřeba doplnit, jsou údaje pro generování formulářů. Část aplikace s formuláři bude představovat úplně nový koncept. Dosud jsou formuláře pro služby mnohdy i celé aplikace. V této práci chceme ukázat a vyzkoušet možnost sjednocení komunikace s úřady do jedné aplikace.

Účelem formuláře bude získat data od uživatele ve strukturované podobě. Tato data se podle požadavků doplní do dokumentu a oboje bude odesláno úřadu (odeslání bude simulováno). Formulář musí obsahovat nejen pole pro získání dat, ale hlavně popisy polí. Uživateli musí být jasné, která informace je od něj vyžadována.

Bylo by možné, aby formuláře byly v datech již vytvořené v cílovém formátu. Tedy v našem případě formuláře v jazyce HTML. Takový způsob je ale značně nevhodný z mnoha důvodů. Objem dat by byl navýšen, snížila by se strojová zpracovatelnost a editace by nebyla snadná. Navíc by byly smíchány koncepty návrhu uživatelského rozhraní a obsahu formulářů. Rozložení a vzhled formuláře musí být část oddělená od dat.

Z hlediska dat je pro každé pole důležitý jeho popis/název a typ dat, který přijímá. Určitě lze přijímat všechna data v textové podobě, ale pro uživatele bude lepší využít různé typy. Prohlížeče dokáží podle typu pomoci s vyplněním, například v případě datumu. Ale také zvládnou základní ověření správnosti, třeba do pole s číselným typem neumožní vložit jiné znaky. Z hlediska aplikace také bude vhodné mít data v odpovídajících datových typech pro případné zpracování.

Bude tedy potřeba navrhnout způsob obsažení těchto informací v datech o úkonech. Následně je data nutné doplnit do dokumentu, s kterým je formulář spojen. Je tedy nezbytné provázat výslednou hodnotu z pole s místem v textovém dokumentu.

4. Návrh aplikace

4.1 Rozvržení stránek

S využitím požadavků a uživatelských příběhů byla funkcionalita rozdělena do jednotlivých stránek. Poté byly navrhnuty tzv. mockupy. Jedná se o prvotní návrhy stránek v grafickém editoru, jen pomocí textu a jednoduchých tvarů. Hlavním účelem není návrh vzhledu, ale spíše rozmístění funkcí a vytvoření si představy, jak bude uživatel aplikaci používat. Mockupy v této práci byly vytvořeny nástrojem MockFlow[1].

Nultá stránka

Do webové stránky se předpokládá přihlášení elektronickou identitou. K té samozřejmě nemáme přístup, přihlášení bude pouze simulováno. S elektronickou identitou je spojena datová schránka a také údaje o uživateli. Právě tyto údaje lze využít k personalizaci služeb navrhovaných uživateli.

Je třeba představit více uživatelů v různých životních situacích, s rozdílnými zaměstnáními a také rozlišit právnické a fyzické osoby. K tomu bude sloužit nultá stránka. Ta by nebyla součástí reálného projektu. Nám bude sloužit k výběru uživatelského účtu.

Návrh stránky na obrázku 4.1 ukazuje tři panely s popisem uživatelů. Na výsledné stránce jich bude potřeba o trochu více. Také by bylo vhodné přidat informační text o tom, že je stránka pouze demo pro simulaci přihlášení.

Obrázek 4.1: Návrh nulté stránky

[1]Nástroj Mockflow je dostupný jako webová aplikace na `https://mockflow.com`

Hlavní stránka

Hlavní stránka musí sloužit jako rozcestník do všech ostatních funkcionalit, ale musí zůstat přehledná. Uživatel je již přihlášen, tedy tato stránka bude personalizovaná. Uživatel se odtud musí být schopen dostat ke všem službám. Jak k již objednaným, tak i k teprve hledaným. V záhlaví stránky se bude nacházet vyhledávání. Je potřeba doporučit uživateli relevantní služby.

Na obrázku 4.2 je možné vidět jeden z dřívějších návrhů. Prvotním plánem bylo doporučování uskutečnit podle nejčastěji vyhledávaných služeb uživatelů s podobnými vlastnostmi, což ale není vhodné, nemáme žádné uživatele. Z toho důvodu je první sekce pojmenovaná „Nejčastěji hledané“. Později bude přejmenovaná na „Doporučené“. Služby jsou představovány pouze jménem, vhodné by bylo přidat i krátký popis.

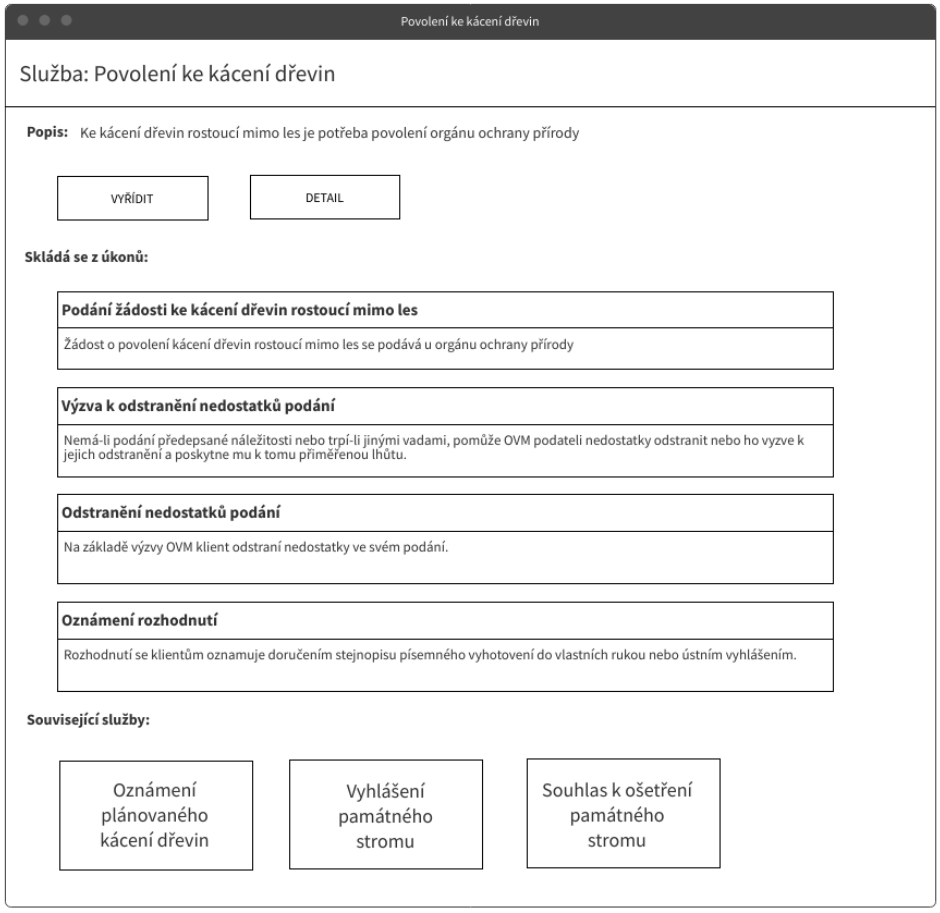
Vyhledávané služby se zobrazí hned pod vyhledávacím polem, ve stejném stylu jako doporučené služby. Je třeba také přidat tlačítka pro odhlášení a přechod do uživatelské stránky. Původně byl přechod zamýšlen kliknutím na uživatelské jméno, to ale nemusí být pro uživatele moc jasné.

Obrázek 4.2: Návrh hlavní stránky

Detail služby

Poté co si uživatel vyhledá vhodnou službu, je potřeba mu o ní zobrazit více informací. Mezi důležité informace patří oprávněný typ klientů, jednotlivé úkony a způsob vyřízení. Zároveň by bylo dobré ukázat související služby, které uživatele také mohou zajímat.

Návrh na obrázku 4.3 je druhou iterací. Do prvního návrhu jsem umístil příliš mnoho informací a stránka by se pro běžného uživatele stala nepřehlednou. Proto je v návrhu umístěno tlačítko „DETAIL“. Jednotlivé úkony jsou nejdůležitější informací o službě. Mělo by být jasné, která strana je inicializuje a jaké je jejich pořadí. Nachází se zde i tlačítko pro objednání služby.



Obrázek 4.3: Návrh stránky s detaily služby

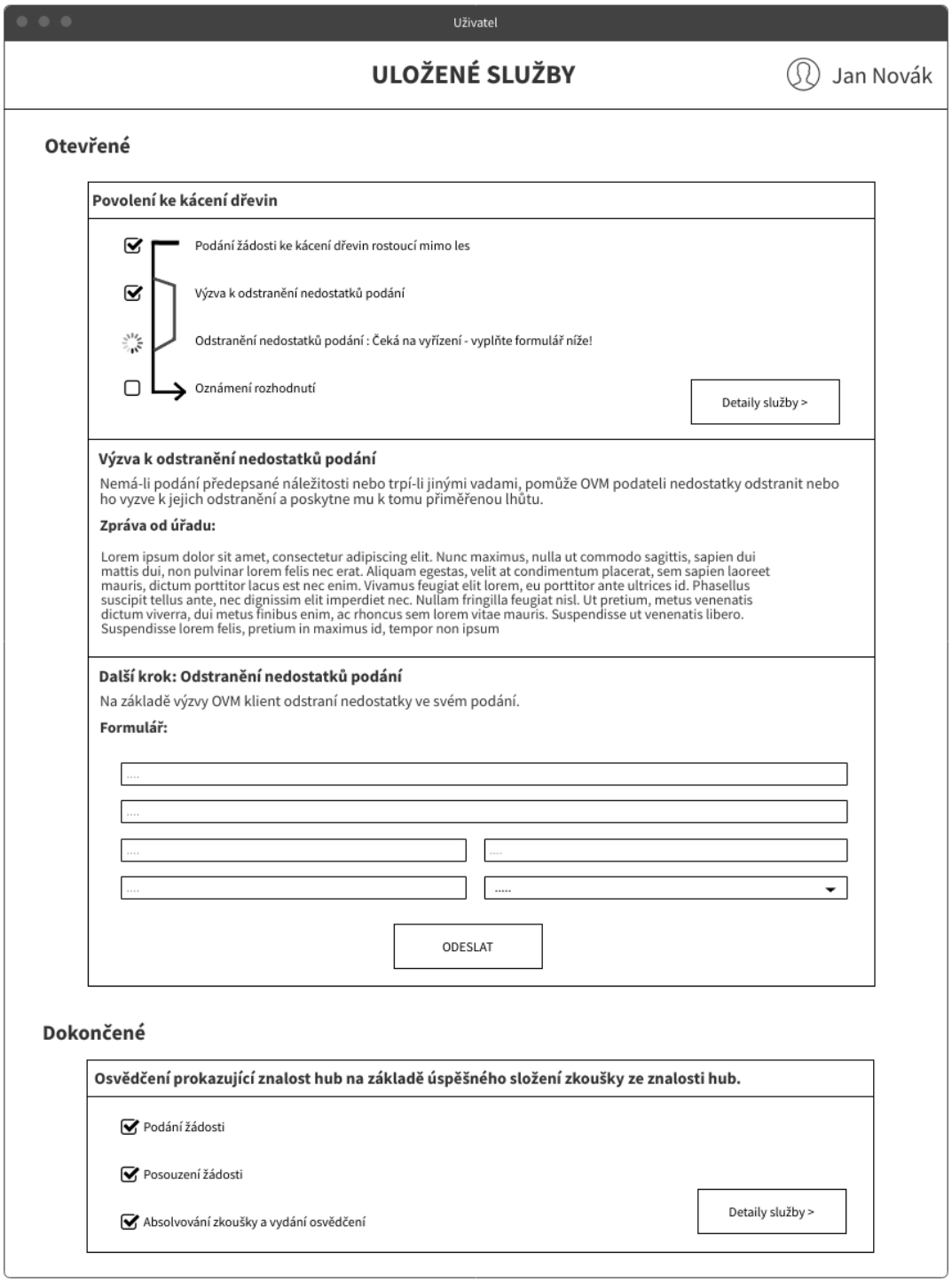
Uživatelská stránka

Uživatelská stránka zobrazí uživateli všechny jeho vyřizované služby. Přenese nás do ní tlačítko pro vyřízení v detailu služby. Stránka je přístupná i z hlavní stránky. Služba se skládá z úkonů a ty mají nějaký průběh. Ten je potřeba naznačit. Každý úkon je iniciován buď klientem nebo úřadem.

Návrh je možné vidět na obrázku 4.4. Na něm můžeme vidět průběh úkonů, zprávu obdrženou od úřadu a také formulář, který je potřeba vyplnit. Průběh je naznačen formou grafu, ovšem ten z dostupných dat nejde sestavit. Data bude nutné doplnit.

Také by bylo vhodné přidat možnost odstranění služby, pokud by se jich nahromadilo větší množství. Případně pokud by si uživatel službu objednal omylem. Komunikace probíhá až po prvním kroku, takže odstranění do té doby dává smysl.

Formulář bude dynamicky generován. Je třeba navrhnout strukturu dat, podle kterých se to učiní. Pole formuláře musejí být dobře popsaná.



Obrázek 4.4: Návrh uživatelské stránky

4.2 Doporučování a vyhledávání

4.2.1 Doporučování služeb

Uživateli by měly být při vstupu do aplikace doporučeny relevantní služby. Elektronická identita poskytuje identifikační a kontaktní údaje jako jméno, adresu, věk a telefon [11]. Ty kromě věku nelze dostatečně využít k doporučování služeb. Dále může poskytovat údaje z registru obyvatel nebo registru osob, s těmi

ale v této práci nebudeme počítat.

Údaje o cílovém klientovi služby jsou v RPP také omezené. Mohou se nacházet v popisu služby, v té podobě nejsou ale strojově zpracovatelné. Bylo by možné využít agendy, do které služba patří, ale to je spíše administrativní klasifikace.

Zdroj detailních popisů služby z Katalogu služeb zmíněný v sekci 3.2 poskytuje k některým službám klíčová slova. Ta by byla pro doporučování i vyhledávání ideální. Narážíme ovšem na problém možného odstavení API a limitované pokrytí služeb. Proto tento způsob také není vhodné využít.

Při analýze dat a procházení některých služeb jsem si všiml, že nabízené služby jde docela vhodně kategorizovat podle skupiny cílových osob. Jako příklad si vezměme služby „Hlášení narození dítěte“ a „Dohoda o užívání příjmení pro dítě“. Takové služby si bude chtít vyřídit pravděpodobně jen nový rodič. Služby jako „Volba způsobu výplaty důchodu“ a „Vdovský a vdovecký důchod“ si nejspíš budou chtít vyřídit senioři.

Aplikace bude tedy doporučovat služby podle role uživatele. Taková data nejsou součástí otevřených dat, je tedy nutné je doplnit. Navíc je potřeba propojit role a služby. Jedním řešením by bylo spojit roli přímo s doporučenými službami. Tedy každá služba by měla údaj o tom, pro koho je vhodná. Služeb jsou ale tisíce a takové doplnění by bylo neefektivní.

Již byla zmíněna klíčová slova z Katalogu služeb. Ty nelze použít. Přece jenom je ale využití klíčových slov vhodné, jen v jiné podobě. Každá role k sobě může mít přiřazená klíčová slova. Ta by se pak využila při vyhledávání. Jedním z možných způsobů je hledat taková klíčová slova v názvech či popisech služeb. Role „Mladý rodič“ by mohla mít klíčová slova jako „dítě“, „mateřská“ či „rodič“. Nebylo by tedy nutné doplňovat data do služeb. Jen je třeba navrhnout role a k nim přiřadit klíčová slova. V aplikaci by pak bylo vhodné, aby si uživatel mohl roli měnit a dynamicky se podle toho měnily doporučené služby.

Pro účely práce byly navrženy role a k nim klíčová slova v tabulce 4.1. Proces byl iterativní, bylo vyzkoušeno že daná klíčová slova naleznou alespoň nějaké relevantní služby. Pro účely tohoto procesu byla klíčová slova hledána v názvech. Některé slova jsou cíleně nekompletní (např. „vozidl“) pro umožnění vyhledání variant skloňování. Role „Obecná osoba“ pak slouží jako výchozí role pro nové uživatele.

Název role	**Klíčová slova**
Mladý rodič	dítě, mateřská, rodič
Farmář	pole, hospodaření, zvěř
Senior	důchod, senior
Profesionální řidič	řidič, mýtné, vozidl
Majitel nemovitosti	nemovitost, daň, nájem
Zástupce právnické osoby - správa lesů	kácení, dřevin, les
Obecná osoba	řidič, pas, občanský průkaz, daň

Tabulka 4.1: Role a jejich klíčová slova pro doporučování

4.2.2 Vyhledávání služeb

Uživatel musí být schopen vyhledat službu, o které chce informace nebo kterou si chce objednat. Nejvhodnější pro vyhledávání je název služby nebo alespoň část názvu. Uživatel si ve většině případů přesný název pamatovat nebude. Je tedy potřeba vyhledat služby s názvem podobným vstupu uživatele. To ale není vůbec jednoduchý úkol. Samozřejmě nejlepší by bylo, kdyby tuto funkci podporoval dotazovací jazyk nad zdrojem dat, tedy SPARQL (viz kapitola 3.1). Ten ale podporu pro hledání podobného textu nemá. SPARQL umí vyhledávat jen podle přesné shody. Tedy nalezne službu, jejíž název obsahuje vstup. To samo o sobě není vhodné, kvůli skloňování v českém jazyce. Například vstup „cestovní pas“ není přímo obsažen v názvu služby „Vydání cestovního pasu“.

Jedna z možností je získat názvy všech služeb a následně využít nějaký nástroj/databázi, který umožňuje vyhledávání podle shody. Ovšem služeb je tisíce, takže nepřipadá v úvahu názvy všech služeb získávat při každém vyhledávání. Možné je mít všechny názvy připravené při startu aplikace a v nich poté vyhledávat. Bylo vyzkoušeno několik různých funkcí/knihoven v jazyce Python, které vrátí nejbližší výsledek nebo výsledky seřadí podle podobnosti. Ovšem takové funkce musejí projít všechny názvy služeb, takže výsledky vrátí v rámci sekund, to je pro webovou aplikaci nepřijatelné. Samozřejmě nejlepší možnost je mít názvy služeb v databázi, která podporuje vyhledávání podle podobnosti. To je ale nad rámec této práce.

Existuje ale alternativa mezi vyhledáváním s přesnou shodou a s podobností. Zadaný vstup je možné rozdělit na jednotlivá slova. Ty potom nalezneme v názvech služeb. Tím získáme výrazně menší podmnožinu služeb. Služby potom můžeme setřídit podle podobnosti názvu a původního vstupu. Tato možnost nám umožní využít data přímo ze zdroje a SPARQL dotaz je rychlý. Zároveň minimalizuje pomalost vyhledávání podle podobnosti, jelikož se bude vyhledávat na malé podmnožině.

4.3 Návrh architektury

Výstupem této práce bude webová aplikace. Ty jsou běžně dělené na dvě části. Část aplikace běžící v klientově prohlížeči se nazývá frontend. Opakem je backend, neboli část aplikace běžící na serveru. Hranice mezi těmito dvěma částmi není pevně stanovená. Rozdělení funkcionality mezi frontendem a backendem většinou závisí na vývojáři a na požadavcích na aplikaci. Následuje návrh architektury jednotlivých funkcionalit v relaci s analýzou požadavků.

Přihlašování a uživatelské údaje

Uživatel se bude do aplikace přihlašovat. Přihlašování v kontextu této práce souvisí spíše s držením údajů o uživateli v průběhu relace. Nejde nám o autentizaci, tu by zajišťovala elektronické identita. Mezi cíle práce nepatří ani zabezpečení údajů, jde skutečně jen o prototyp. Uchovávat potřebujeme uživatelovo jméno, roli a unikátní identifikátor, abychom ho jednoznačně mohli spojit s objednanými službami. Tato data můžeme uložit na straně klienta, v podobě souborů

cookies. S každou žádostí budou odeslána na backend a budou využita k identifikaci uživatele.

Doporučování a vyhledávání

S uživatelem je vázaná jeho role. Chceme umožnit změnu role pro ukázku dynamického doporučování. Je třeba poskytnout výběr všech rolí a následně změnit doporučované služby. S tím souvisí i vyhledávání. Při vyhledávání bude potřeba zobrazit nebo překreslit výsledky. Tedy obě funkce jsou podobné, jen se liší vstup (role nebo hledaný text).

Vhodné je zaregistrovat změnu vstupu na straně uživatele a na server jen vznést požadavek na nová data. Alternativou by bylo při každém vstupu znovu vynutit od serveru načtení celé stránky. To by aplikaci ale zpomalilo. Navíc uživatel by znovunačtení zaregistroval a mohlo by to pro něj být nepříjemné, například tím že by se změnila pozice v okně.

Informace o službách

Data o službách se budou získávat z více zdrojů a některá bude potřeba doplnit. To je vhodnější učinit na straně serveru. Data se doplní do stránky a ta bude odeslaná klientovi. Poté už je stránka v podstatě jen statická. Žádné informace se na ní nemění, nemá žádné funkce kromě několika tlačítek s odkazy.

Vyřizované služby

Každému uživateli by mělo být umožněno vyřizovat více služeb najednou. Oproti uživatelským údajům by nebylo vhodné tato data uchovávat na straně klienta. Potencionálně to může být velké množství dat, které je nejvhodnější uložit do databáze. Musíme propojit uživatele a službu. K tomu by měli stačit jejich unikátní identifikátory.

Vyřizování má nějaký průběh. Je nutné zaznamenat v jaké fázi se vyřizování nachází. Musí být zachována konzistence stavu vyřizování mezi relacemi. Bylo by nepřijatelné, aby se některý úkon zopakoval vícekrát. Nebo naopak, aby se nějaký přeskočil. Dále chceme umožnit doplňování dat do formulářů, a to i data z předchozích formulářů. Musíme tedy uložit i doplňované údaje.

Mezi data, jež je potřeba zaznamenat, tedy patří typ služby, postup vyřizování a data související s formuláři.

4.4 Architektura backendu

Větší část aplikace se bude nacházet na serveru. Hlavními úkoly backendu jsou získávání a zpracování dat o službách, doporučování, práce s databází a vytváření a poskytovaných webových stránek. Všechny tyto části je potřeba rozdělit do vhodných celků pro usnadnění vývoje i s ohledem na možné budoucí rozšíření.

Běžně využívaným architektonickým vzorem při návrhu aplikací je Model-view-controller (MVC). Deacon [12] zmiňuje, že dává smysl oddělit jádro aplikace od jakýchkoliv uživatelských rozhraní. A to z hlediska znovupoužitelnosti i vývoje. Jednotlivé komponenty poté Deacon popisuje následovně.

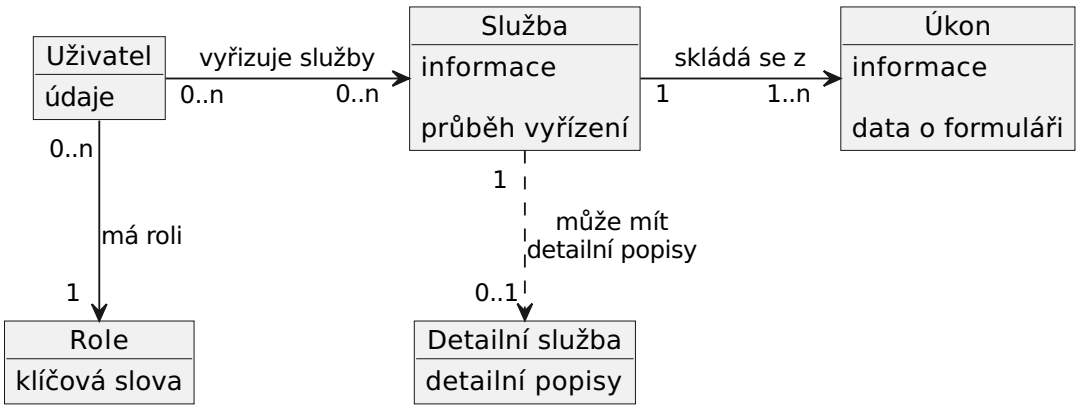
Právě jako neměnné jádro programu označuje **model**. Jsou to třídy modelující a podporující daný problém. Model by neměl mít absolutně žádné spojení se světem mimo aplikaci. Druhou komponentou je **view**. View je jedno nebo více uživatelských rozhraní nebo spíše třídy vytvářející toto rozhraní. View ví o existenci modelu, jelikož z něj zobrazuje data. Případně je může i měnit. **Controller** je pak objekt umožňující manipulaci view. Controller zpracovává vstup, zatímco view zpracovává výstup. Controller ví o view, opačný vztah neplatí.

Tento vzor využijeme i v této práci. Deacon popisuje MVC v kontextu všech aplikací, ne jen webových. Ze vzoru si vezmeme spíše rozčlenění aplikace do komponent než vztahy mezi nimi.

Model

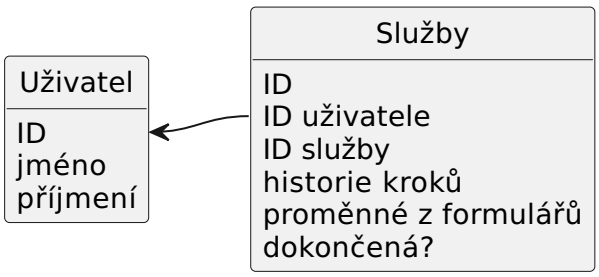
Model bude sloužit k ukládání a práci s daty. Budou v něm třídy reprezentující jednotlivé objekty, se kterými pracujeme. Těmi jsou služba, úkon a uživatel. Ty potřebujeme doplňovat daty, úkolem modelu bude tedy i komunikace s datovými zdroji. Pro ukládání postupu vyřizování budeme potřebovat databázi, s kterou bude model zajišťovat komunikaci.

Na UML diagramu v obrázku 4.5 můžeme vidět objekty, se kterými bude model pracovat, a vztahy mezi nimi. Jde spíše o výšeúrovňový přehled, ne přesné rozvržení tříd. Detailní službou se myslí detailní popisy služeb z datového zdroje Katalogu služeb, jež jsou dostupná jen pro některé služby (viz sekce 3.2).



Obrázek 4.5: Konceptuální model datových tříd

V databázi potřebujeme uložit údaje o uživatelích a postup jednotlivých služeb. Na obrázku 4.6 je navrženo jaká data bude potřeba v databázi uložit. Uživatel může mít objednáno více služeb. V tabulce jsou tři identifikátory, jeden pro uživatele, druhý pro službu a třetí pro rozlišení v případě, že předchozí dva budou stejné. Stejné budou když si uživatel bude vyřizovat dvě identické služby, což také chceme umožnit. Poté potřebujeme uložit průběh, tedy historii již splněných kroků a proměnné z formulářů. Přesné datové typy budou rozmyšleny až po návrhu datové struktury formulářů.



Obrázek 4.6: Návrh tabulek v databázi

Získávání dat bude zajišťováno jednou třídou, ta bude mít funkce na získání různých typů dat z různých zdrojů.

- Získání služeb podle klíčových slov pro doporučování
- Získání služeb podle názvu pro vyhledávání
- Získání služby podle ID z Katalogu otevřených dat
- Získání detailních popisů služby z Katalogu služeb

Controller

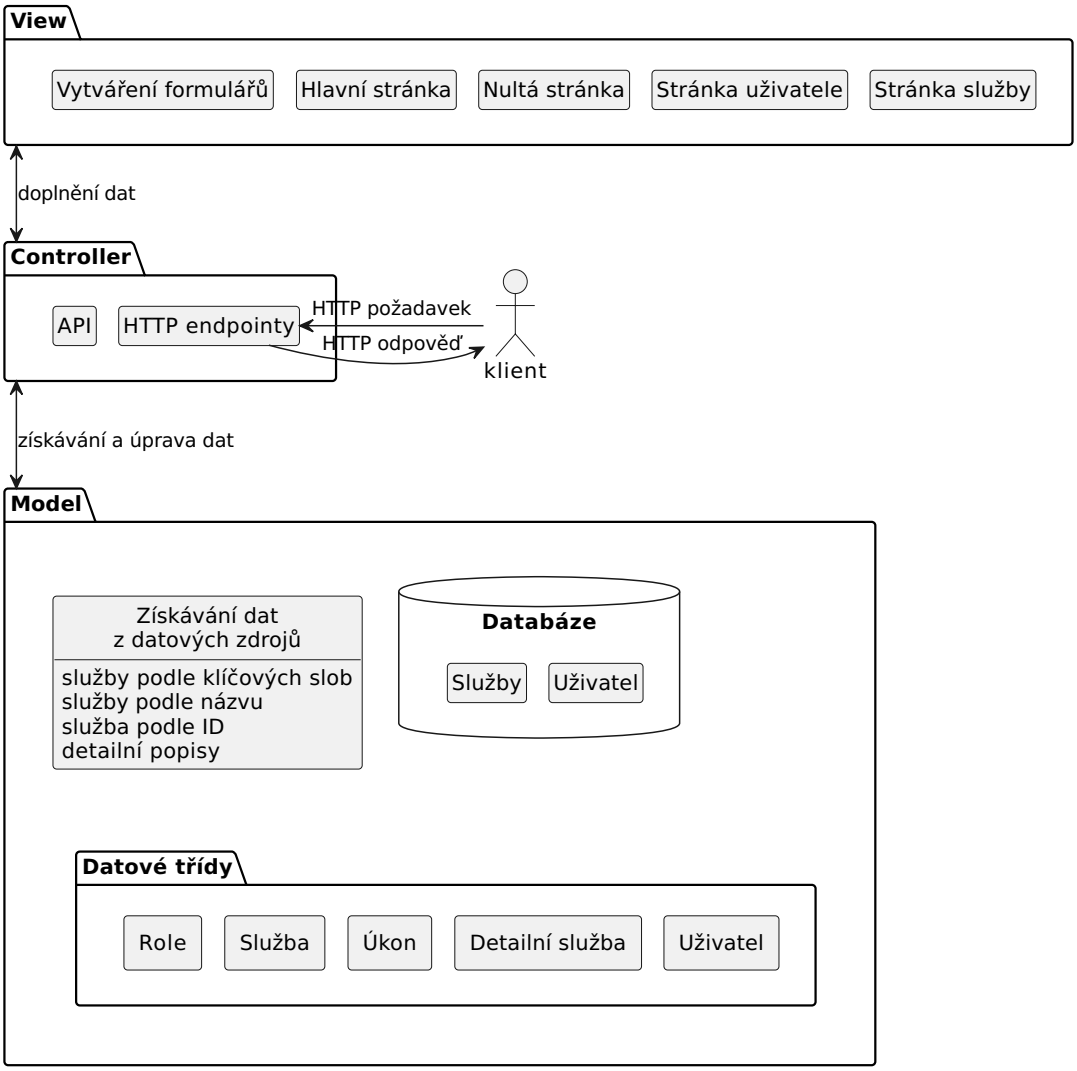
Controller bude přijímat příchozí HTTP požadavky, vybere k nim vhodnou akci, od modelu vyžádá potřebná data (případně je upraví) a předá data view. Vybráním vhodné akce se rozumí detekce použitého endpointu/URL a zavolání k němu namapované funkce. Potřebujeme endpointy jak endpointy pro poskytování služeb, tak API pro komunikaci s frontendem.

View

Za uživatelské rozhraní v této práci lze považovat webové stránky. Tedy view budou jednotlivé stránky a kód do nich doplňující data. Sestavování formulářů je také součástí uživatelského rozhraní. Do view zařadíme i statické soubory, které ke stránkám patří, tedy soubory se skripty v JavaScriptu.

4.5 Shrnutí návrhu

V této kapitole byly navrženy jednotlivé stránky a funkcionality. Poté bylo rozmyšleno, jak budou uchovávána data, jak budou jednotlivé komponenty komunikovat a celkově fungovat. A na závěr byla aplikace rozdělená do modulů a tříd, inspirací byl architektonický vzor MVC. Pomocí nástroje PlantUML byl vytvořen shrnující diagram na obrázku 4.7. Lze na něm vidět 3 hlavní moduly tedy view, controller a model, a jak mezi sebou komunikují. Také je na něm znázorněno, jak jsou moduly rozčleněné a jaké funkce zastávají. View a model jsou navzájem izolované, vůbec o sobě nemusí vědět. Tedy případná změna jedné z těchto komponent je nezávislá na druhé.



Obrázek 4.7: Shrnutí architektury

5. Popis implementace

5.1 Použité technologie

5.1.1 Frontend

Pro popis struktury stránek bylo využito jazyka HTML[1]. Podle dříve navržených mockupů do něj byly jednotlivé stránky převedeny. Ke stylování stránek se využívá jazyka CSS[2]. Ten je možné použít přímo a stránky stylovat s ním. Existuje ale velké množství frameworků usnadňujících stylování pomocí CSS. Pro tuto práci byl využit jeden z nejpopulárnějších, Bootstrap[3].

Bootstrap abstrahuje použití nízkoúrovňových vlastností jazyka CSS do již předpřipravených tříd. Třídy se píší tedy přímo do HTML dokumentu (do atributu „class“ jednotlivým elementům). Ukázka kódu na obrázku 5.1 obsahuje krátký příklad použití. Třídy mohou představovat jen jednoduché vlastnosti. Například třídy `bg-dark` a `text-light` nastaví tmavé pozadí a bílý text a třída `text-center` text vycentruje. Mohou ale také představovat skupinu vlastností či celé komponenty. Z běžného odkazu lze vytvořit tlačítko pro potvrzení přidáním tříd `btn` a `btn-success`.

```
<div class="bg-dark text-light text-center">text</div>

<a href="/main" class="btn btn-success">Potvrdit</a>
```

Obrázek 5.1: Ukázka frameworku Bootstrap

Na pozadí také Bootstrap řeší zobrazování pro různě velké obrazovky pomocí responzivních stylů. Pro tuto práci je vhodný, chceme aby aplikace byla uživatelsky přívětivá a zároveň nechceme komplikované stylování.

Třetí důležitou částí frontendu je umožnění interakce uživatele se stránkou. K tomu je využito skriptovacího jazyka JavaScript. Pro něj existuje také spousta frameworků. Jeho základní verze je ale dostatečně jednoduchá pro přímé využití. Navíc práce vyžaduje na frontendu jen dvě funkcionality, vyhledávání a doporučování. Konkrétně zjištění změny role pro doporučování a vstupu při vyhledávání, následné odeslání požadavku backendu a nakonec zpracování a zobrazení výsledků.

Do stránky je potřeba doplnit data. To je možné udělat na straně klienta například pomocí již zmíněného JavaScriptu. To ale vyžaduje dodatečnou komunikaci frontendu a backendu. Většina dat je neměnná, tedy doplnění dat je prováděno rovnou na serveru.

[1] HyperText Markup Language
[2] Cascading Style Sheets
[3] Bootstrap je dostupný z `https://getbootstrap.com/`

5.1.2 Backend

Jazyk a framework

Backend je dnes možné psát v různých jazycích s mnoha frameworky. Pro tuto práci byl zvolen jazyk Python. Můžeme ho porovnat s jiným populárním jazykem PHP. Python sice není určen přímo pro tvoření backendu. Zato má ale velkou uživatelskou základnu poskytující mnoho knihoven a lépe se v něm pracuje s daty. Také je podle mého názoru lépe čitelný a snáze se v něm programuje. PHP sice může být rychlejší, o to nám ale v této práci nejde.

Je využita nejnovější verze jazyka Python. Tou je v době vývoje práce verze 3.10. Vyžadované knihovny pak mají stanovené přesné verze, které jsou navzájem kompatibilní.

Dva asi nejpopulárnější frameworky pro vývoj webových stránek v Pythonu jsou Django[4] a Flask[5]. Django je plnohodnotný framework držící se přístupu „batteries-included“. Což znamená že obsahuje všechny možné nástroje, jaké by vývojář mohl potřebovat. Je vhodnější pro velké projekty, kde se požadavky výrazně nemění. Není moc flexibilní, nutí vývojáře dodržovat doporučené vzory.

Flask poskytuje jen nutné základy. Umožňuje rozšíření pomocí pluginů. Dává vývojáři větší svobodu pro výběr technologií. Je vhodný pro menší projekty, kde se požadavky mohou měnit. Proto bylo rozhodnuto využití frameworku Flask. Práce sice měla již stanovené požadavky, ale při vývoji bylo potřeba měnit strukturu aplikace a testovat různé knihovny.

Použité knihovny

Python obsahuje rozsáhlou standardní knihovnou s mnoha moduly, které práce používá. Většinu z nich zde popisovat nebudeme, některé ale za zmínku stojí.

Modul `dataclasses` usnadňuje vytváření datových tříd. Běžnou třídu automaticky rozšíří o speciální metody, jako konstruktor nebo metodu umožňující přehledný výpis informací. Aplikace jej využívá k vytváření datových modelů.

Modul `typing` umožňuje přidávat typové anotace funkcím a proměnným. Použité typy sice nejsou nijak vynucované, všechny proměnné zůstanou dynamické. Práce využívá typové anotace jako součást dokumentace. Při vývoji byly výrazně nápomocny s určením obsahu proměnných.

Nyní si projdeme použité externí knihovny. Ty už jsou vytvářeny jinými vývojáři. Dostupné jsou v repozitáři PyPI[6], instalují se pomocí nástroje PIP.

Hlavní použitou knihovnou je již zmiňovaný `Flask`, usnadňuje vytvoření webového serveru a spravuje HTTP komunikaci. Jeho součástí je knihovna `Jinja2`, což je šablonový engine umožňující bezpečné doplňování dat do HTML. K generování formulářů je využito rozšíření `Flask-WTF`. Pro získání dat v podobě JSON souborů a odesílání dotazů GraphQL endpointu je využito knihovny `requests`. Pro vytváření SPARQL dotazů se používá knihovna `SPARQL-Burger`. Jejich následné odeslání a zpracování odpovědi zařizuje `SPARQLWrapper`. Využití těchto knihoven bude podrobněji popsáno v následujících sekcích.

[4] `https://www.djangoproject.com/`

[5] `https://flask.palletsprojects.com/`

[6] Python Package Index `https://pypi.org/`

Jedna z knihoven, konkrétně `SPARQLWrapper`, nefunguje ve verzi Pythonu 3.10 korektně. Chyba byla nahlášena. Vývoj knihovny ale není velmi aktivní a bez aktivity autorů nelze navrhovanou změnu zveřejnit. Proto je opravená verze knihovny distribuovaná s aplikací ve formátu .whl. Je to v podstatě jen ZIP archiv, který umí nástroj PIP správně nainstalovat.

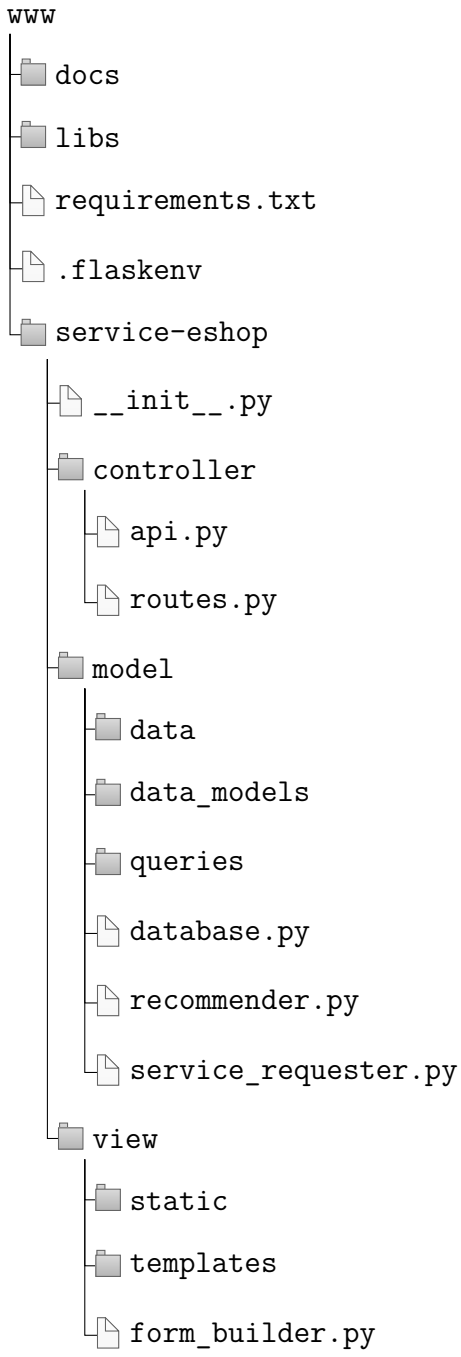
Pro vygenerování dokumentace v podobě HTML stránky je využito knihovny `pdoc`. Dokumentaci generuje z komentářů v kódu a z typových anotací.

Struktura projektu

Na obrázku 5.2 je znázorněná struktura projektu. Adresáře jsou přítomné všechny, ze souborů byly vybrány jen ty nejdůležitější.

V kořenovém adresáři se nacházejí adresáře `docs`, `libs` a `service-eshop`. Do adresáře `docs` je vygenerovaná dokumentace projektu pomocí již zmíněné knihovny pdoc. V `libs` se nacházejí knihovny distribuované přímo s aplikací, v našem případě pouze jedna. A konečně v adresáři `service-eshop` se nacházejí zdrojové kódy projektu. Soubor `requirements.txt` definuje potřebné knihovny a jejich verze pro instalační nástroj PIP. Soubor `.flaskenv` knihovně Flask definuje, kde se aplikace nachází a na kterém portu by měla běžet.

Vstupní bod aplikace se nachází v `service-eshop/__init__.py`. Tam je aplikace inicializovaná a části projektu jsou s ní propojeny. Projekt je rozdělen podle vzoru MVC ze sekce 4.4.



Obrázek 5.2: Struktura projektu

5.2 Controller

Modul controller má za úkol zpracovávat příchozí HTTP žádosti, získat od modelu potředná data, ty pak předá view. Modul se skládá jen ze dvou částí/souborů. V `routes.py` jsou definovány URL cesty pro běžné uživatelské stránky. V `api.py` jsou potom cesty pro poskytování dat pro JavaScript frontend.

Tento modul obsahuje pouze základní logiku, práce jako získávání dat, průchod službou při vyřizování, doplňování dat do HTML apod. jsou přenechány ostatním modulům.

5.2.1 URL cesty

Definice a logika URL cest

Jednotlivé URL endpointy jsou ve Flasku definovány pomocí dekorátorů. Následující ukázka kódu definuje URL `/service/service_id`, kde service_id je identifikátor služby. Identifikátor je předán připojené funkci, jejímž úkolem je vytvořit odpověď. V tomto případě získá funkce službu a její detaily a předá je view přes funkci `render_template`. Takto dekorovaná funkce může vracet string (jako v tomto případě) představující HTML dokument nebo objekt `flask.Response`. Tento objekt vrací třeba funkce pro přesměrování `flask.redirect()` nebo `flask.make_response()`.

Ukázka definice URL cesty pomocí Flasku

```
from flask import render_template, make_response
from .. import app

@app.route("/service/<string:service_id>")
def service(service_id: str):
    service = app.requester.get_service(service_id)
    details = app.requester.get_service_detail(service_id)
    suggested = ...

    return render_template(
        "service.html",
        service=service,
        detail=details,
        suggested=suggested
    )
```

URL cesty a jejich funkce

Následuje výčet všech URL cest, HTTP metod, které přijímají, a jejich funkce. Cesty, které přijímají POST, přijmou i GET, ale hned uživatele přesměrují zpět, pokud by se na ně nějakým způsobem omylem dostal.

U POST endpointů je využito tzv. Post/Redirect/Get návrhu. Tedy po POST metodě je uživatel přesměrován metodou GET na tu samou stránku. To zamezí opětovné odeslání formuláře při znovunačtení stránky.

[GET] `/`
Nultá přihlašovací stránka.

[POST, GET] `/main`
Hlavní stránka. POST může přijít z nulté stránky a bude obsahovat uživatelské údaje.

[GET] `/service/<ID>`
Stránka s detaily služby.

[GET] `/user`
Uživatelská stránka.

[POST] `/user/save`
Uloží novou službu do vyřizovaných.

[POST] `/user/delete`
Odstranění služby z vyřizovaných.

[POST] `/user/form`
Vyplnění nebo odeslání formuláře.

[GET] `/api/roles`
Všechny dostupné role. Vrátí JSON dokument.

[GET] `/api/services/<search_text>`
Hledání služeb podle názvu. Vrátí JSON dokument.

[GET] `/api/recommend/<role_name>`
Doporučení služeb pro roli. Vrátí JSON dokument.

Ukládání uživatelských údajů do cookies

Bylo naplánováno ukládání údajů do cookies. Ty se nastavují tím, že se doplní do hlaviček odpovědi. Ve Flasku se k tomu používá metoda `set_cookie` na objektu `Response`. V controlleru proto byla vytvořena následující metoda. Ta převezme objekt uživatele od modelu a do HTTP odpovědi vloží jméno, příjmení, roli a identifikátor uživatele.

Funkce pro vložení dat do cookies

```
def _user_to_cookies(user: User, resp: Response) -> None:
    one_hour = timedelta(hours=1)
    resp.set_cookie("name", user.name, max_age=one_hour)
    resp.set_cookie("surname", user.surname, max_age=one_hour)
    resp.set_cookie("role", user.role, max_age=one_hour)
    resp.set_cookie("id", str(user.id), max_age=one_hour)
```

5.3 Model

Úkoly modulu model jsou získávání dat, jejich doplnění do datových tříd, správa databáze, vyhledávání a doporučování. V adresáři `data` se nachází data o rolích, uživatelích, doplněná data služeb a také soubor databáze. V `data_models` jsou obsaženy datové třídy a v `queries` funkce pro vytváření SPARQL dotazů. V souboru `database` jsou funkce pro správu databáze, v `recommender` je třída `Recommender` pro doporučování služeb a v `service_requester` je obsažena třída `Requester` pro komunikaci s datovými zdroji.

V souboru `data/roles.json` jsou již zmíněná data o rolích. Jde o JSON soubor, kde jsou zanesené role a jejich klíčová slova přesně podle návrhu v tabulce 4.1. V souboru `users_mock.py` jsou pak tyto role připojeny k uživatelům v podobě datových tříd `User`.

5.3.1 Datové třídy

Datové třídy aplikace využívá čtyři. `Service` obsahuje data služby a její průběh, `ServiceAction` její úkony, `DetailedService` detailní popisy z Katalogu služeb a `User` uchovává uživatelská data. Role nakonec oproti návrhu nedostala vlastní objekt, protože stačila jedna datová položka třídy `User`.

V následující ukázce kódu můžeme vidět datovou třídu `User`. Automaticky je jí vygenerován konstruktor a další funkce. Datovým položkám lze nastavit výchozí hodnotu jako v tomto případě u role.

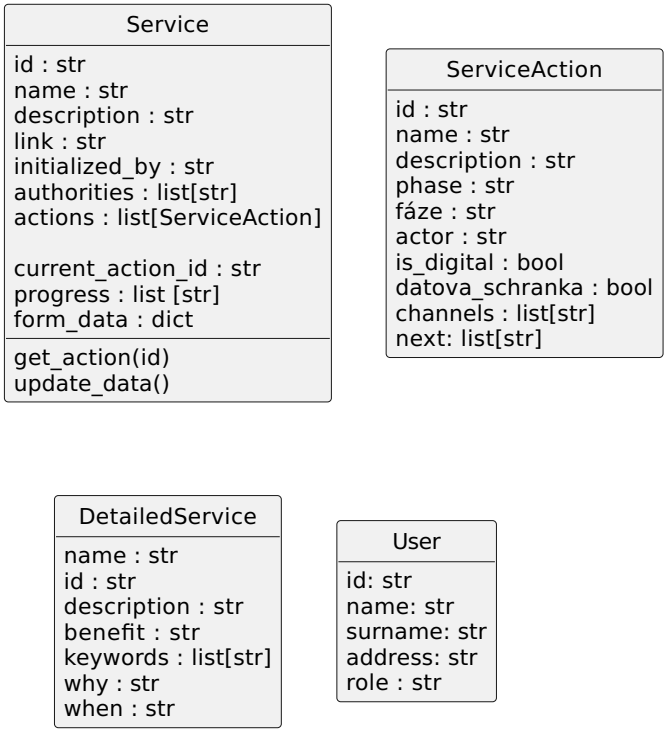
Ukázka datové třídy

```
from dataclasses import dataclass

@dataclass(repr=True, unsafe_hash=True)
class User:
    id: int
    name: str
    surname: str
    role: str = "Obecná osoba"
```

Třída `Service` má navíc dvě metody. Jednou je `get_action` pro získání úkonu (`ServiceAction`). A druhou `update_data` pro aktualizaci formuláře, případně přechodu na další úkon iniciovaný klientem.

Na obrázku 5.3 můžeme vidět přehled datových tříd a jejich položek. Podrobnější popisy jednotlivých datových polí jsou dostupné v dokumentaci, případně v komentářích kódu.

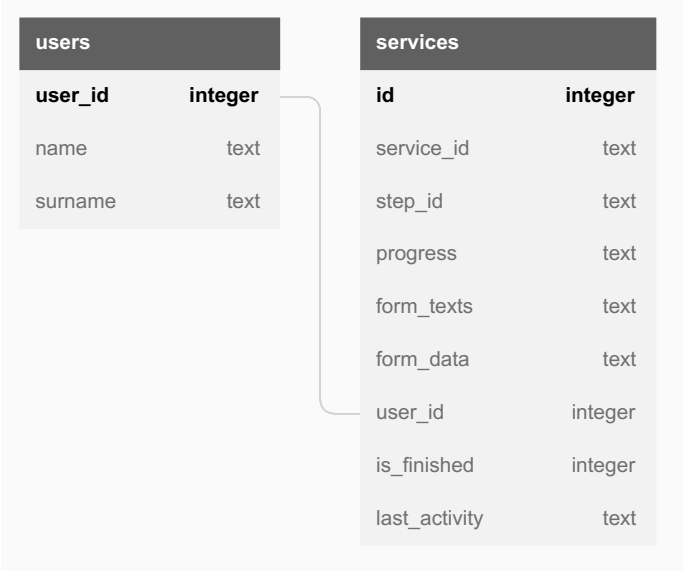


Obrázek 5.3: Datové třídy a jejich položky

5.3.2 Databáze

Python má v sobě zabudovaný databázový systém SQLite[7]. Databáze je uložena na disku a k přístupu k ní se používá jazyk SQL. Funkce pro interakci s databází se nacházejí v souboru `database.py`. SQLite není striktně staticky typovaná databáze, datový typ je asociovaný s hodnotou, ne celým sloupcem.

Na obrázku 5.4 je možné vidět schéma databáze vytvořené v nástroji dbdiagram[8]. Text reprezentuje textový řetězec, integer celé číslo. U uživatele je uložen unikátní identifikátor, jméno a příjmení. U služby už je to trochu komplikovanější. `id`, `service_id`, `step_id` a `user_id` jsou identifikátory instance objednané služby, služby z RPP a aktivního úkonu. `is_finished` je pravdivostní hodnota označující ukončení vyřizování. `last_activity` je časový údaj sloužící k seřazení služeb podle času interakce.



Obrázek 5.4: Schéma databáze

SQLite nepodporuje složitější datové struktury. Lze do ní ovšem vkládat JSON v podobě textu. Obsahuje dokonce i rozšíření pro práci s JSONem. Složitou alternativou by bylo vytvořit více tabulek a využít vztahy M-N. Data by bylo složité dávat dohromady a v Pythonu lze velice snadno převést JSON do slovníku či pole.

Položka `progress` je tedy JSON pole obsahující identifikátory již proběhlých úkonů. `form_texts` je pole dokumentů vyplněných daty z formulářů. Samozřejmě by je šlo znovu generovat, ale to by mohly být změněny, což je z archivačního hlediska nepřijatelné. `form_data` je poté JSON objekt mapující název proměnné z formuláře na její vyplněnou hodnotu.

Ukázka vytvoření tabulky uživatelů

```
CREATE TABLE users (
    user_id integer PRIMARY KEY,
    name text NOT NULL,
    surname text NOT NULL
);
```

[7] https://www.sqlite.org/
[8] https://dbdiagram.io/

Tabulky jsou vytvořeny a uživatelé do nich vloženy při startu webového serveru. Tedy při startu se vymažou existující data. To je jen z důvodu toho, že aplikace je demo a při každém spuštění chceme mít aplikaci ve stejném stavu. Lze to změnit přidáním slov „IF NOT EXISTS“ za „CREATE TABLE“.

Objekt databázového připojení je uložen v globálním kontextu aplikace. Ten je dostupný z jakékoliv žádosti. Byly vytvořeny různé metody abstrahující práci s databází.

query_services() Získá všechny vyřizované služby uživatele.

query_service() Získá jednu vyřizovanou službu podle jejího ID.

insert_service() Vloží novou službu do databáze podle objektu Service.

save_service() Aktualizuje službu v databázi podle objektu Service.

delete_service() Smaže službu z databáze.

init_tables() Vytvoří potřebné tabulky, spuštěno při startu aplikace.

5.3.3 Získávání dat

Data z RPP

Původně bylo k získávání dat plánováno jen využití již zmíněného SPARQL endpointu. Bohužel došlo v průběhu vývoje k několika výpadkům. Výpadek byl nahlášen správci otevřených dat, ministerstvu vnitra. Problém byl po týdnu odstraněn. Výpadek se poté vyskytl ještě jednou. Jelikož SPARQL endpoint není možno považovat za plně spolehlivý, byl přidán alternativní zdroj a to data ve formátu JSON. Poskytuje identická data jen v rozdílném formátu. Pro získání konkrétní služby se využívá primárně JSON zdroj a v případě jeho výpadku se přepne na SPARQL. Pro vyhledávání je použit pouze SPARQL endpoint, díky možnostem dotazovacího jazyka.

Struktura dat a ukázkové dotazy byly již ukázány v kapitole 3. Nyní se zaměříme na to, jak jsou data zpracována. Soubor `service_requester.py` obsahuje třídu Requester, jež má toto na starosti. V jejím konstruktoru se připraví spojení a stáhnout se JSON data.

Metoda `get_service()` slouží k získání dat podle identifikátoru. Využívá k tomu metody `_get_service_json()` nebo `_get_service_sparql()`, podle pravidel zmíněných dříve. Data jsou doplňována do datové třídy `Service`. V případě JSONu jsou data získána z předstaženého souboru. V případě SPARQL jsou vytvořeny dotazy. Další metodou je `get_services_by_keywords()` vracející služby podle klíčových slov.

Pro sestavení dotazů SPARQL dotazů je využito knihovny `SPARQLBurger.SPARQLQueryBuilder` umožňující programatické vytvoření dotazu. Tyto funkce jsou umístěné v samostatných souborech v adresáři `queries`. Tento způsob je lepší než mít dotazy připravené rovnou v textové podobě z důvodu snadnější změny.

GraphQL API Katalogu služeb

Metoda `get_service_detail()` vrací detailní popisy služby z Katalogu služeb. Ten poskytuje GraphQL API. GraphQL dotaz je velmi jednoduchý, není tedy využito žádné knihovny. Dotaz je odeslán knihovnou `requests` a výsledek je obdržen ve formátu JSON.

5.3.4 Doplněné služby

Každý úkon musel být doplněn o následníky. Tedy úkonu byla přidaná datová položka `next`. Je to list řetězců, obsahující identifikátory následujících úkonů. Následujících úkonů tedy může být více. Pokud jsou vyřizovány úřadem, vybere se z nich jeden náhodně. Při vývoji jsem se nesetkal se službou, která by nabízela uživateli více úkonů. Pokud ale taková existuje, při možném budoucím rozšíření je nutné zvážit možnost dát uživateli mezi nimi na výběr. Také jsou doplněna formulářová data.

5.3.5 Datová struktura formulářů

Formulář se skládá z polí a z textu, do kterého budou doplněna, představující dokument. Využívá se JSON formátu.

Klíč `fields` obsahuje pole (array) formulářových polí. Každé pole musí mít tři hodnoty. `field_name` představuje název proměnné, ve které bude hodnota uložena. Není doporučené využívat mezery ani diakritiku. `label` představuje popisek, který bude zobrazen ve formuláři. A do třetice hodnota `field_type` představuje typ formulářového pole a jaké hodnoty bude přijímat. Aplikace momentálně podporuje následující:

text Jednořádkové textové pole

text_area Víceřádkové textové pole

date Datum

int Číselná hodnota

email Emailová adresa

boolean Zaškrtávací pole (hodnota ano/ne)

Tyto datové typy určují nejen výstup, ale také umožňují základní validaci od prohlížeče. Do číselného pole uživatel nebude moci vložit znaky apod.

Pokud se pole jmenuje `name` nebo `surname` bude do něj předvyplněno uživatelovo jméno či příjmení.

Klíč `output` poté obsahuje výsledný dokument. Do něj lze pomocí složených závorek doplnit proměnné z formuláře.

Na obrázku 5.5 je možné vidět ukázku formuláře se třemi poli.

Ukázka struktury formuláře

```
{
    "fields": [
        {
            "field_name": "name",
            "field_type": "text",
            "label": "Jméno"
        },
        {
            "field_name": "date",
            "field_type": "date",
            "label": "Datum"
        },
        {
            "field_name": "request",
            "field_type": "text_area",
            "label": "Obsah žádosti"
        }
    ],
    "output": "Jméno: {name}\nŽádost: {request}\n\nDne: {date}"
}
```

Obrázek 5.5: Ukázka datové struktury formuláře a výsledný formulář

5.4 View

View se skládá ze tří částí. Soubor `form_builder.py` obsahuje logiku pro vykreslování formulářů podle jeho dat. Adresář `static` obsahuje statické soubory, tedy JavaScript kód pro frontend. V adresáři `templates` jsou pak HTML šablony.

5.4.1 Doplňování dat do HTML

Součástí frameworku Flask je také knihovna Jinja[9]. Je to šablonový engine, který umožňuje v HTML psát kód podobný Pythonu. Tedy šabloně jsou předána data a vykonají se části kódu v ní obsažené. Jinja umožňuje vcelku rozsáhlé možnosti skriptování, aplikace ale využívá jen základy.

[9] `https://jinja.palletsprojects.com/`

Controller obsahuje pro každé URL funkce, viz sekce 5.2.1. V sekci se nachází i ukázka kódu funkce. Zaměřme se na její část.

```
return render_template(
    "service.html",
    service=service,
    detail=details,
    suggested=suggested
)
```

V této ukázce lze vidět, jak jsou předána šabloně data. To je zařízeno pomocí funkce `flask.render_template`. Prvním argumentem je jméno šablony. Ty se berou právě z adresáře `templates`. Další argumenty pak už představují předávána data. Konkrétně zde jsou předávány služba, detailní popisy služby a relevantní služby.

Na následující ukázce lze vidět útržky jazyka Jinja. Kód je značně zkrácen a některé atributy byly odstraněny. Obsah v dvojitých složených závorkách je vyčištěn od nevhodných znaků a vložen do HTML. Složené závorky se znakem procenta potom reprezentují kontrolní logiku. Zde můžeme vidět cyklus, který zobrazí všechny úkony. Lze také použít filtry, například `lower`, který vypíše text malými písmeny.

Ukázka doplnění dat pomocí knihovny Jinja

```
<h2>{{ service.name }}</h2>

{% for action in service.actions %}
<div class="card>
    <div class="card-header">
        <h3>{{ action.name }}</h3>
        <h5>{{ action.phase }}</h5>
    </div>
    <div class="card-body">
        {{ action.description }}

        Úkon inicializuje: {{ action.actor|lower }}
        Kanály komunikace: {{ ", ".join(action.channels)|lower }}
    </div>
</div>
{% endfor %}
```

5.5 Dokumentace

Všechny části kódu jsou okomentovány. Pomocí nástroje `pdoc` byla vygenerována HTML dokumentace. Ta je součástí elektronické přílohy práce. Součástí je i uživatelská příručka - návod na použití stránky. Příloha A.1 obsahuje návod k instalaci aplikace a zprovoznění webového serveru.

6. Testování aplikace

6.1 Testování použitelnosti

6.1.1 System usability scale

Pro získání zpětné vazby od uživatelů bylo využito tzv. system usability scale (dále SUS). Do češtiny lze tento název přeložit jako stupnice použitelnosti systému. Stupnici vytvořil v roce 1995 John Brooke [13]. Slouží jako snadný způsob zjištění použitelnosti systému.

Testované osobě je potřeba nejdříve umožnit interakci s aplikací. Poté, bez jakékoliv diskuze, vyplní zadaný dotazník. Osoba by měla odpověď vyplnit co nejdříve, bez dlouhého rozmýšlení.

Jelikož je tato aplikace kompletně nová, je testované osobě potřeba krátce její účel vysvětlit. K tomu byl využit následující text.

„Tato aplikace slouží jako návrh internetového obchodu se službami státu. Lze v ní zjišťovat informace o službách, které stát poskytuje občanům. Služby je v aplikaci možné i vyřizovat, uživatel je poté proveden jejím kroky.“

Poté je testované osobě umožněna interakce s aplikací. Začne se na nulté stránce. Doporučeno bylo projít si následujícími úkoly v uvedeném pořadí.

- Nacházíte se na stránce, která simuluje přihlášení, přihlaste se jako libovolný uživatel.
- Doporučování funguje podle rolí uživatele, změňte svoji roli na libovolnou jinou.
- Vyhledejte službu s názvem „Vydání cestovního pasu“ a přejděte na ní.
- Zkuste zjistit dobu platnosti nového cestovního pasu.
- Podívejte se jaké úkony tato služby vyžaduje.
- Vraťte se na hlavní stránku. Dole v sekci „Možné objednávat“ si zvolte libovolnou službu a zobrazte ji. Můžete si o ní přečíst informace, poté klikněte na tlačítko pro vyřízení.
- Byli jste přesunuti do stránky s Vašimi objednanými službami. Můžete zde vidět i Vámi zvolenou službu. Přečtěte si, co vyžaduje. Zkuste projít všemi úkony a vyplnit přitom formuláře.
- Služba se přesunula do vyřízených, odstraňte ji.

Brooke [13] pomocí výzkumu vybral 10 otázek, které dobře shrnou pocity testované osoby. Testovaná osoba si vždy přečte výrok. Poté ohodnotí, jak moc s ním souhlasí. Hodnotí na stupnici od 1 do 5, kde 1 znamená „rozhodně nesouhlasím“ a 5 „rozhodně souhlasím“. Pokud osoba cítí, že nemůže odpovědět, označí střední hodnotu 3. Výroky pro dotazník jsou dostupné v angličtině. Pro tuhle práci byly přizpůsobeny a přeloženy do českého jazyka.

Výroky pro dotazník

1. Myslím si, že bych tuto aplikaci rád využíval často.
2. Aplikaci jsem shledal zbytečně složitou.
3. Myslím si, že je aplikace snadno použitelná.
4. K užívání aplikace bych potřeboval pomoc odborné osoby.
5. Různé funkce byly do aplikace dobře integrovány.
6. Tato aplikace mi připadala velmi nekonzistentní.
7. Myslím si, že většina lidí by se tuto aplikaci naučilo používat velice rychle.
8. Podle mě je používání aplikace moc těžkopádné.
9. Při používání aplikace jsem se cítil velmi sebejistě.
10. Musel jsem se toho hodně naučit než jsem mohl aplikaci pořádně použít.

6.1.2 Způsob hodnocení

Výsledkem testování SUS je pro každou osobu číslo od 0 do 100. To označuje použitelnost systému. Skóre pro jednotlivé otázky samo o sobě nemá vypovídající hodnotu.

Každá otázka do výsledného skóre přispěje 0 až 4 body. Pro liché otázky je skóre hodnota stupnice mínus 1. Pro sudé otázky je skóre hodnota stupnice odečtena od 5. Skóre pro každou otázku se sečte, výsledek je poté ještě vynásoben 2,5.

Jeff Sauro, který se systémem SUS zabývá, posoudil a analyzoval data od více než 5000 uživatelů z 500 rozdílných testování [14]. Jako medián mu vyšlo skóre 68. Aplikaci s takovým výsledkem lze označit jako dobře použitelnou. Nad devadesátý percentil se lze dostat se skórem nad 80,3. Takovou aplikaci lze označit za výbornou a uživatelé jí s velkou pravděpodobností budou doporučovat ostatním. Naopak skóre pod 51 umístí aplikaci pod patnáctý percentil a naznačuje, že je aplikace těžce použitelná a vyžaduje značné úpravy.

6.1.3 Výsledky testování

Otestované byly čtyři osoby. Jejich výsledná skóre byla 87,5, 95, 92,5 a 80. Průměr těchto výsledků je 88,75. Tedy aplikace lze označit za výborně použitelnou.

První osoba studuje informatiku, tedy více se zaměřila na hledání chyb ve funkcionalitách. Odhalila například, že vyhledávání špatně funguje pokud jsou použita rozdílná malá a velká písmena. Tato chyba byla následně odstraněna. Druhá osoba se zajímá o grafiku a pochválila vzhled stránky a funkčnost na neobvyklém internetovém prohlížeči, který používá.

Zbylé dvě osoby internet umějí využívat dobře, nejsou ovšem pokročilé a s podobnou aplikací se nesetkaly. Bylo tedy zajímavé pozorovat jaké problémy mají při interakci se stránkou. Problém nastával při vyplňování formulářů. Po odeslání

se stránka načetla znovu a okno zůstalo zaměřeno na vrchol stránky. Po testování tohle bylo napraveno, okno se zaměří zpět na formulář nebo náhled dokumentu.

Ze zpětné vazby a hodnocení vyplynulo, že je stránka mírně náročnější na používání. To také naznačuje výrok, jenž získal nejnižší ohodnocení. Tím je „K užívání aplikace bych potřeboval pomoc odborné osoby.“ V průměru ale stejně získal ohodnocení 2, tedy „spíše nesouhlasím“, takže to není vůbec závažný problém. Při případném rozšíření by bylo vhodné přidat přímo do aplikace více vysvětlivek.

6.2 Validace splnění požadavků

Následuje namapování funkcionalit na požadavky a vyhodnocení, zda byly všechny splněny.

Aplikace bude simulovat přihlášení pomocí elektronické identity. Bude obsahovat nultou stránku, kde bude na výběr z několika typů uživatelů. Nultá stránka je přítomna, nabízí výběr ze šesti uživatelů s různými rolemi.

Hlavní stránka bude obsahovat stručný přehled aktivních služeb a možnost přejít na stránku s kompletní historií služeb uživatele. Hlavní stránka zobrazuje aktivní služby a lze přejít na stránku uživatele, kde jsou zobrazeny všechny vyřizované služby včetně dokončených.

Hlavní stránka umožní vyhledávání služeb. Také budou zobrazeny služby doporučené pro přihlášeného uživatele. Aplikace umožňuje vyhledávání služeb podle názvu. Uživateli jsou služby doporučovány pomocí jeho role a k ní přidružených klíčových slov.

Služby budou nabízeny podobně jako v internetových obchodech. Služby je možné vyhledávat a jsou doporučovány relevantní. Je možné zobrazit si detailní informace a případně objednat vyřízení.

Hlavní stránka bude obsahovat jen stručný popis služby. Pro více informací uživatel přejde na stránku s detailem služby. Na hlavní stránce se zobrazuje název a popis služby. Případně jeho část, pokud je moc dlouhý. Stránka služby pak obsahuje mnohem více informací. A pokud jsou dostupné, tak i detailní popisy získané z Katalogu služeb.

Na stránce s detailem služby si bude moci uživatel objednat vyřízení. Všechny služby lze zařadit do objednaných. Pouze u služeb doplněných o potřebná data je možná simulace vyřízení. Takové služby jsou pro rozlišení na hlavní stránce ve speciální sekci.

Budou využity dva zdroje dat. Prvním jsou data z RPP, ty jsou ale nedostatečná pro objednávání. Druhým zdrojem tedy budou data manuálně doplněná o údaje potřebné k objednávání. Data z RPP jsou

v aplikaci využita. Některé služby jsou doplněné o potřebná data k objednávání. Navíc byl nalezen další zdroj dat, API Katalogu služeb MVČR.

Uživatel u aktivního úkonu vyplní vygenerovaný formulář, data se doplní do dokumentu. Obojí se odešle úřadu. U služeb, které mají doplněná data, je umožněno vyplňovat formuláře. Po vyplnění se zobrazí náhled, kde jsou data doplněná do dokumentu. Po odeslání se pokračuje k následujícím úkonům nebo je služba splněna.

Úkony, které vyžadují akci úřadu, se budou simulovat a provedou se automaticky. Úkony inicializované úřadem se provádějí automaticky. Je simulována odpověď úřadu v podobě dokumentu. V práci je naznačeno, jak by se data skutečně posílala úřadům.

Všechny požadavky práce byly splněny.

Závěr

Byl vytvořen internetový obchod představující občanům nabídku služeb státu. Webová aplikace umožňuje vyhledávání služeb a poskytuje o nich detailní údaje. Simuluje se přihlášení za účelem personalizace. Obchod doporučuje uživateli relevantní služby. Doporučování je řešeno podle uživatelských rolí, s kterými jsou spojena klíčová slova.

Práce popisuje fungování RPP a analyzuje dostupná otevřená data. Kromě RPP byl nalezen další datový zdroj poskytující detailní popisy služeb. Pro funkci vyřizování služeb bylo navrženo doplnění dat. Obchod uživatele provádí jednotlivými úkony. Pomocí formulářů je simulovaná komunikace s úřady. Formuláře jsou generovány dynamicky, práce navrhuje strukturu jejich dat.

Součástí práce je i popis návrhu architektury a vývoje. Bylo provedeno testování použitelnosti, jehož výsledky byly pozitivní. Při testování byly objeveny a následně opraveny drobné chyby. Jako přílohy jsou připojeny vygenerovaná dokumentace programu, návod na instalaci a uživatelská příručka.

V průběhu práce se vyskytlo několik problémů. SPARQL endpoint Národního katalogu otevřených dat měl výpadky. Jednou i přes nahlášení problému trval týden. Aplikace je na datech z katalogu stavěna, bez nich nemůže fungovat správně. Katalog poskytuje více zdrojů, později byl pro stabilitu zapojen druhý, data ve formátu JSON. U něj se ale také následně objevily výpadky, naštěstí jen velmi vzácné.

Náročné se také ukázalo seřadit úkony služeb pro potřeby vyřizování. Pro každý úkon je třeba určit následující. U některých to jde odvodit z názvu či popisu úkonu. U většiny by to ale vyžadovalo buď odbornou znalost služby nebo podrobné zjišťování informací z jiných zdrojů.

Bylo zajímavé navrhnout nový způsob komunikace s úřady a vyřizování služeb. Pro každý úkon služby je možné doplnit popis formuláře. Pokud jsou úkony seřazené, obchod uživatele plně provede vyřízením služby a vyplněním formulářů.

Byly splněny všechny požadavky práce. Obchod slouží jako funkční ukázka toho, jak by mohlo pokračovat rozšiřování eGovernmentu.

Možné budoucí rozšíření

Vyhledávání a doporučování služeb, popsané v sekci 4.2, je omezeno funkcemi SPARQL endpointu. Vyhledávání by bylo možné vylepšit zavedením technologie umožňující vyhledávání podle podobnosti textu. Nejvhodnější by bylo fulltextové vyhledávání, nejen v názvech služby. Jednou takovou technologií je například Elasticsearch[1].

Komunikace s úřady je v práci simulovaná, odpovědi jsou generovány automaticky. V kódu je naznačeno, kde by bylo prováděno odeslání formulářů (modul controller, `routes.py`, funkce user_form). Odpověď úřadu by mohla přijít až po několika dnech. Bylo by potřeba navrhnout architekturu umožňující dlouhodobou komunikaci. S tím by také souvisela část aplikace na straně úřadů.

[1] `https://www.elastic.co/`

Formuláře by bylo možné výrazně rozšířit. RPP obsahuje i informace o údajích, které jsou shromažďovány. Nacházejí se v datové sadě Subjekty a objekty údajů a jejich údaje[2]. Tedy každé pole ve formuláři by mohlo být svázáno s metadaty o něm. Poté by šlo ověřovat celkovou validitu formulářů a vynucovat dodržování schématu.

[2]Datová sada dostupná na `https://data.gov.cz/`

Seznam použité literatury

[1] Vít Lidinský. *eGovernment bezpečně*. Grada, Praha, 2008. ISBN 978-80-247-2462-1.

[2] Marek Doleček. Základní registry veřejné správy, 2020. URL `https://www.businessinfo.cz/navody/zakladni-registry-verejne-spravy-ppbi/`.

[3] MVČR. Základní registry, 2022. URL `https://archi.gov.cz/nap:zakladni_registry`.

[4] ČESKO. Zákon č. 111/2009 Sb. o základních registrech. *Sbírka zákonů České republiky*, 2009. URL `https://www.zakonyprolidi.cz/cs/2009-111`.

[5] Martin Kalina. Registr práv a povinností, 2022. URL `https://www.szrcr.cz/cs/registr-prav-a-povinnosti`.

[6] MVČR. Katalog služeb veřejné správy, 2022. URL `https://archi.gov.cz/nap:katalog_sluzeb`.

[7] W3C JSON-LD Community Group. JSON for Linking Data, 2022. URL `https://json-ld.org/`.

[8] L. Feigenbaum, G. T. Williams, K. G Clark, and E. Torres. SPARQL 1.1 Protocol, 2013. URL `https://www.w3.org/TR/sparql11-protocol/`.

[9] Martin Nečaský. Registr práv a povinností - služby veřejné správy, 2021. URL `https://ofn.gov.cz/registr-pr%C3%A1v-a-povinnost%C3%AD/slu%C5%BEby/2021-01-12/`.

[10] MVČR. Služby veřejné správy, 2022. URL `https://portal.gov.cz/sluzby-verejne-spravy/`.

[11] MVČR. Příručka k využití služeb národní identitní autority pro kvalifikované poskytovatele služeb veřejné správy, 2022. URL `https://info.identitaobcana.cz/download/SeP_PriruckaKvalifikovanehoPoskytovatele.pdf`.

[12] John Deacon. Model-View-Controller (MVC) Architecture, 2013. URL `http://www.johndeacon.net/john-deacon/articles/model-view-controller-architecture/`.

[13] John Brooke. SUS: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[14] Jeff Sauro. Measuring Usability with the System Usability Scale (SUS), 2 2011. URL `https://measuringu.com/sus/`.

A. Přílohy

A.1 Instalace a spuštění webového serveru

Ke spuštění aplikace je potřeba mít nainstalovaný Python verze 3.10. Python je možné stáhnout pro většinu OS z `https://www.python.org/downloads/`.

Verzi Pythonu je možné ověřit příkazem:

```
python --version
```

Aplikace je dostupná jako elektronická příloha práce nebo je dostupná na: `https://gitlab.mff.cuni.cz/abraham1/service-eshop`.

Projekt je třeba naklonovat a otevřít adresář `www`:

```
git clone https://gitlab.mff.cuni.cz/abraham1/service-eshop.git
cd service-eshop/www
```

Dále je nutné vytvořit virtuální prostředí, aktivovat ho a nainstalovat potřebné knihovny:

```
python -m venv venv

venv/bin/activate       # Linux

venv\Scripts\activate   # Windows

pip install -r requirements.txt
```

Port serveru je možné změnit v souboru `.flaskenv`. Server lze spustit příkazem:

```
flask run
```

Při výchozím nastavení se aplikace spustí na adrese `http://127.0.0.1:5000`. Tento server je pouze určen pro vývoj. Možnosti nasazení do produkce jsou popsány v dokumentaci Flasku dostupné z `https://flask.palletsprojects.com/en/2.1.x/deploying/`.