FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Mobilní aplikace pro zobrazení výsledků hlasování Poslanecké sněmovny
Student:	Bc. Lukáš Dang
Vedoucí:	Ing. Ondřej John
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je návrh, implementace a otestování Android aplikace sloužící k zobrazení výsledků hlasování poslanců Poslanecké sněmovny Parlamentu ČR. Aplikace bude obsahovat následující funkce:

- Souhrnný výsledek hlasování (přijat/nepřijat, počty hlasů pro/proti/zdržených).
- Výsledek hlasování s rozpisem hlasování jednotlivých poslanců.
- Profil poslance obsahující seznam jeho hlasů.
- Odkazy na web PS.

Součástí práce bude implementace backend služby poskytující data pro mobilní aplikaci prostřednictvím REST API.

- Analyzujte a popište strukturu zdrojových dat poskytovaných PSP.
- Specifikujte funkční a nefunkční požadavky na mobilní aplikaci a back-end API.
- Po dohodě s vedoucím práce navrhněte uživatelské rozhraní mobilní aplikace.
- Navrhněte rozhraní REST API, které bude poskytovat data pro mobilní aplikaci.
- Navrhněte datovou strukturu backend služby.
- Implementujte a otestujte API.
- Implementujte a otestujte mobilní aplikaci.
- Shrňte výsledek práce, popište její přínos.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 7. října 2021 v Praze.

Diplomová práce

MOBILNÍ APLIKACE PRO ZOBRAZENÍ VÝSLEDKŮ HLASOVÁNÍ POSLANECKÉ SNĚMOVNY

Bc. Lukáš Dang

Fakulta informačních technologií
Katedra webového inženýrství
Vedoucí: Ing. Ondřej John
14. února 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych tímto poděkoval svému vedoucímu, Ing. Ondřej John, za jeho vstřícnost, trpělivost a čas, který mi věnoval při vedení mé diplomové práce. Dále bych chtěl poděkovat své rodině, která mě při psaní podporovala.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 14. února 2023 .....

Abstrakt

Diplomová práce popisuje návrh a implementaci mobilní aplikace, která slouží k zobrazení výsledků hlasování poslanců Poslanecké sněmovny Parlementu ČR. Součástí práce je i návrh a implementace backendu, který bude zpracovávat data z webu poslanecké sněmovny parlamentu ČR a poskytovat je mobilní aplikaci prostřednictvím REST API.

Klíčová slova poslanecká sněmovna, parlament, hlasování, poslanec, poslanecký klub, REST, Android, mobilní aplikace, backend, Kotlin, Java

Abstract

The diploma thesis describes the design and implementation of a mobile application that serves to display the voting results of members of the Czech Parliament's Chamber of Deputies. The work also includes the design and implementation of a backend that will process data from the Czech Parliament's Chamber of Deputies website and provide it to the mobile application through a REST API.

Keywords Chamber of Deputies, parliament, voting, member of parliament, parliamentary party group, REST, Android, mobile application, backend, Kotlin, Java

Shrnutí

Cíle

V této sekci budou uvedeny cíle této práce.

Úvod

Tato sekce slouží jako úvod pro diplomovou práci.

Motivace a požadavky

Tato kapitola slouží jako úvod do tematiky hlasování v poslanecké sněmovně a motivace k vytvoření mobilní aplikace pro zobrazení výsledků hlasování v PSP. Dále zde budou uvedeny funkční a nefunkční požadavky kladené na mobilní aplikaci a backend.

Analýza existujících řešení

Následně budou analyzována a zhodnocena existující řešení. Konkrétně budou analyzovány zahraniční mobilní aplikace politiscope a Congress.

Analýza zdrojových dat

Součástí analýzy budou zdrojová data z webu poslanecké sněmovny, která budou použita pro přípravu dat pro mobilní aplikaci.

Výběr architektur

V této kapitole budou porovnávány a vybírány architektury mobilní a aplikaci a backend.

Návrh

V této kapitole bude na základě funkčních a nefunkčních požadavků popsán návrh uživatelského rozhraní mobilní aplikace, dotazů na REST API a odpovědí z něj, a databázový model.

Implementace mobilní aplikace

Na základě návrhů bude popsána implementace mobilní aplikace. Popis bude zahrnovat popis využitých nástrojů a technologií, a případně budou porovnány s alternativami. Popis implementace bude po jednotlivých vrstvách architektury.

Implementace backendu

Zde bude popsána implementace backendu. Stejně jako u mobilní aplikace budou popsány využité nástroje a technologie, a případně budou porovnány s alternativami. Implementace bude popsána po vrstvách.

Testování

Po popisu implementace budou popsány testy ověřující funkčnost mobilní aplikace a backendu.

Nasazení

V rámci této kapitoly bude popsán způsob nasazení mobilní aplikace a backendu.

Shrnutí

V této kapitole budou shrnuty výsledky práce a popsány její přínosy.

Seznam zkratek

API	Application Programming Interface
CSV	Comma Separated Values
ČR	Česká republika
FIT	Fakulta Informačních Technologií
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
ORM	Objektově Relační Mapování
PSP	Poslanecká sněmovná Parlamentu ČR
REST	Representational state transfer
SDK	Software Development Kit
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator

Cíle

Prvním cílem práce je seznámit čtenáře s problematikou hlasování a uvést motivaci k této práci. Druhým cílem je analyzovat existující řešení. Dalším cílem je specifikace funkčních a nefunkčních požadavků kladené na mobilní aplikaci a backendu. Následujícím cílem je návrh, implementace, otestování a nasazení mobilní aplikace pro zobrazení výsledků hlasování PSP. Zároveň je cílem návrh, implementace, otestování a nasazení backendu, který bude pravidelně stahovat zdrojová data a transformovat je pro vhodné použití mobilní aplikací. Posledním cílem je shrnutí výsledků práce a popis jejího přínosu.

Úvod

Hlasování v Poslanecké sněmovně Parlamentu České republiky je jedním ze základních aspektů demokracie. Je to způsob, kterým mohou občané nepřímo ovlivňovat zákony v ČR. V PSP je hlasování o návrzích zákonů uskutečňováno prostřednictvím politických reprezentantů, kteří jsou voleni občany. Přestože hlasování hraje důležitou roli pro určování politického směru ČR, přístup k informacím o hlasováních v PSP je pro uživatele mobilních zařízení omezený.

Oficiální portál PSP [1] poskytuje kompletní informace o hlasováních, nicméně není reponzivní pro mobilní zařízení. V dnešním světě, který je čím dál více orientovaný na chytré telefony, to může občanovi ztěžit sledování výsledků hlasování.

Účelem této práce je vyřešení daného problému vytvořením mobilní aplikace pro operační systém Android, která bude poskytovat jednoduché a intuitivní uživatelské rozhraní pro sledování informací o hlasováních. Občané mohou díky aplikaci zůstat informovaní, i když jsou na cestě. Poskytnutím snadno přístupného a uživatelsky přívětivého způsobu pro získání informací o hlasováních lze zvýšit transparentnost hlasovacího procesu a informovanost občanů ČR.

Kapitola 1

Motivace a požadavky

Táto kapitola slouží jako úvod do tematiky hlasování PSP a motivace k této práci. Na konci budou popsány funkční a nefunkční požadavky kladené na mobilní aplikaci a backend.

1.1 Poslanecká sněmovna

Základní prvky politického systému ČR představuje prezident, vláda, Parlament a ústavní soud. Ústava ČR dělí moc na zákonodárnou, výkonnou a soudní. Zákonodárnou moc má na starosti Parlament, výkonnou prezident, vláda a zastupitelství, a soudní Ústavní soud a obecné soudy [2].

Parlament České republiky se skládá ze dvou komor – Poslanecké sněmovny a Senátu. Poslanecká sněmovna se skládá z 200 poslanců a je volena na čtyři roky na základě poměrného volebního systému [3].

1.2 Hlasování v poslanecké sněmovně

Komora PSP je usnášeníschopná, pokud je přítomna alespoň jedna třetina jejích členů. K přijetí usnesení (tzn. ke schválení zákona) je nutný souhlas nadpoloviční většiny přítomných poslanců, pokud ústava nestanoví jinak [3]. Proces návrhu a schvalování zákona je komplexní a řídí se podle určitých pravidel. Avšak pro účely této práce stačí pochopit schvalovací proces v PSP. Kompletní proces přijímání zákonů lze najít na webu PSP [4].

1.3 Webový portál psp.cz

Hlavním zdrojem dat je oficiální webový portál PSP [1], kde jsou mimo jiné informace o hlasováních. Zároveň portál poskytuje zdrojová data ve strojově čitelné formě [5].

1.4 Motivace pro tuto práci

Web PSP obsahuje veškeré informace ohledně hlasováních, nicméně není responzivní a přizpůsobený pro mobilní zařízení. Zároveň srovnatelná aplikace na českém trhu práce ještě neexistuje. V neposlední řadě tato práce podporuje to, abychom měli informované voliče, zajímající se o to, jak jimi volení zástupci hlasují.

1.5 Funkční a nefunkční požadavky

V této sekci budou popsány funkční a nefunkční požadavky na mobilní aplikaci a backendu. Funkční požadavky specifikují funkcionality, které by měl daný software poskytovat. Nefunkční požadavky určují omezení kladená na daný software. Funkční a nefunkční požadavky budou uvedeny ve formě tabulek. Každá tabulka bude mít dva sloupce. Jeden pro identifikátor a druhý pro popis požadavku.

Funkční požadavky

Seznam funkčních požadavků pro mobilní aplikaci jsou v tabulce 1.1 a pro backend v tabulce 1.2.

Funkční požadavky pro mobilní aplikaci	
ID požadavku	**Popis požadavku**
FP_01	Aplikace bude umět zobrazit seznam návrhů zákona. Návrh bude obsahovat popis a výsledek jeho hlasování, a datum a čas hlasování.
FP_02	Aplikace bude umět zobrazit detail návrhu zákona. Detail bude zahrnovat název, datum a čas, odkaz na stenoprotokol, odkaz na oficiální zdroj, a celkovou statistiku hlasování. Celkovou statistikou hlasování rozumíme počet hlasování pro a proti, a počet nepřihlášených, omluvených a zdržených. Dále bude obsahovat výsledky hlasování jednotlivých poslaneckých klubů a jejich členů pro daný návrh zákona.
FP_03	Aplikace bude umět zobrazit seznam poslanců. V seznamu budou následující informace o poslancích: jméno a příjmení, volební kraj, název klubu a profilová fotku.
FP_04	Aplikace bude umět zobrazit detail poslance. Detail poslance bude obsahovat jméno a příjemní, datum narození, profilovou fotku, odkaz na oficiální zdroj, datum začátku mandátu, poslanecký klub a volební kraj. Dále bude obsahovat výsledky jeho hlasování o jednotlivých návrzích zákona.
FP_05	Aplikace bude poskytovat možnost nastavení staršího volebního období. Aplikace bude zobrazovat návrhy zákonů a poslance vždy z aktuálně nastaveného volebního období.
FP_06	Aplikace bude umožňovat filtrovat seznam návrhů zákona podle popisu.
FP_07	Aplikace bude umožňovat filtrovat seznam poslanců podle jména.
FP_08	Aplikace bude poskytovat způsob, jak aktualizovat seznamy, nějakým způsobem prostřednictvím UI.

Tabulka 1.1 Funkční požadavky pro mobilní aplikaci.

Funkční požadavky pro backend	
ID požadavku	**Popis požadavku**
FP_01	Backend bude poskytovat endpoint pro seznam všech volebních období.
FP_02	Backend bude poskytovat endpoint pro seznam návrhů a výsledků jejich hlasování.
FP_03	Backend bude poskytovat endpoint pro detaily návrhu zákona a výsledku jeho hlasování.
FP_04	Backend bude poskytovat endpoint pro výsledky hlasování poslaneckých klubů a jejich členů o daném návrhu zákona.
FP_05	Backend bude poskytovat endpoint pro seznam poslanců.
FP_06	Backend bude poskytovat endpoint pro detaily poslance.
FP_07	Backend bude poskytovat endpoint pro výsledky hlasování daného poslance o návrzích zákonů.

Tabulka 1.2 Funkční požadavky pro backend.

Nefunkční požadavky

Seznam nefunkčních požadavků pro mobilní aplikaci jsou v tabulce 1.3 a pro backend v tabulce 1.4.

Nefunkční požadavky pro mobilní aplikaci	
ID požadavku	**Popis požadavku**
NP_01	Aplikace nebude provádět výpočetně náročná zpracování dat pro šetření aplikace. Ty budou delegována na backend.
NP_02	Aplikace bude mít jednoduché a intuitivní uživatelské rozhraní.
NP_03	Aplikace bude fungovat na zařízeních s OS Android 5.1 a výš pro cílení většího množství uživatelů.
NP_04	Aplikace nebude sbírat uživatelská data.

Tabulka 1.3 Nefunkční požadavky pro mobilní aplikaci.

Nefunkční požadavky pro backend	
ID požadavku	**Popis požadavku**
NP_01	Backend bude data stahovat z webu PSP.
NP_02	Backend bude data stažená z portálu PSP transformovat do databázového modelu 5.3. Cílem je, aby data byla předzpracovaná a připravená pro rychlý přistup z mobilní aplikace.
NP_03	Backend bude ztransformovaná data perzistentně ukládat do databáze.
NP_04	Backend bude data z databáze vystavovat prostřednictvím REST API [6].
NP_05	Backend bude každý den stahovat nová data z portálu PSP a aktualizovat databázi.
NP_06	Backend bude data vystavovat ve formatu JSON [7].
NP_07	Část dat, u kterých má vývojář pocit, že jejich zpracování probíhá příliš dlouho, lze zpracovat až za běhu.

Tabulka 1.4 Nefunkční požadavky pro backend.

Kapitola 2
Analýza existujících řešení

Tato kapitola se zabývá rešerší existujících řešení. Pro rešerši byly zvoleny aplikace, které poskytují informace o výsledcích hlasování o návrzích zákonů. Aplikace byly nalezeny v rámci různých článků na internetu o aplikacích poskytujících informace o aktuálním politickém dění, včetně informace o návrzích zákonů.

2.1 politiscope

Autor: Android Politiscope Developer

Počet stažení: více než 10 000

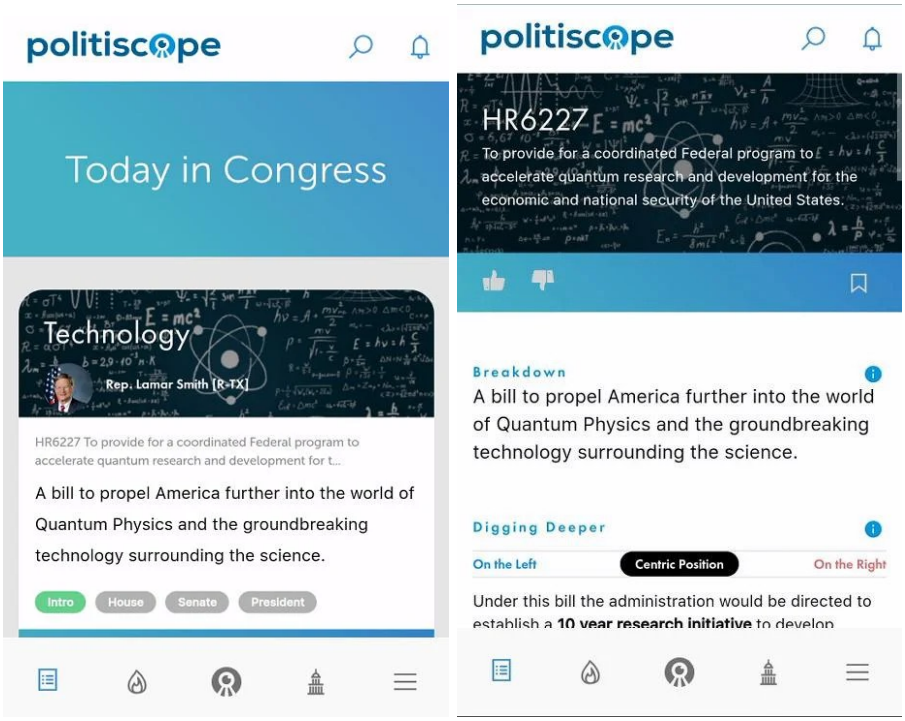
Analyzovaná verze: 2.4 (26. 1., 2023)

Aplikace politiscope [8] dle popisu na Google Play poskytuje informace ohledně politickém dění ve Spojených Státech. Informace jsou objektivní a jsou podány v jednoduché formě. Aplikace poskytuje informace o politických reprezentantech a jejich hlasováních. Aktuální témata jsou barevně označena. Uživatelé mají možnost ukládat návrhy zákonů a sledovat průběh jejich hlasování. Uživatelé mají také možnost sledovat konkrétní politické reprezentanty. Návrhy zákonů jsou označeny tagy pro snazší vyhledání. U témat jsou oficiální sumarizace a odkazy na oficiální zdroje. Lze také sledovat průběh voleb a kampaní.

Aplikace čerpá data z API poskytunýchz následujících zdrojů

- prorepublica [9] – ProPublica je nezávislá, nezisková redakce.
- theunitedstates [10] – @unitedstates je projekt poskytující data ohledně Spojených Států veřejností a pro veřejnost.
- congress [11] – Congress.gov je oficiální portál pro informace z Kongresu a orgánů státní správy Spojených Států.
- openfec [12] – OpenFEC je oficiální portál vlády Spojených Států.

Výše uvedené informace jsou čerpány čistě z popisu a screenshotů aplikace na Google Playi. Do aplikace se mi nepodařilo dostat. Pro přístup je potřeba zaregistrovat se a přihlásit se. Po registraci mě to však automaticky přesměruje na přihlášovací obrazovku, kde po zadání přihlašovacích údaju to pak píše, že účet se zadanými přihlašovacími údaji neexistuje. Aplikaci jsem testoval na dvou různých zařízeních a na obou se vyskytl stejný problém. Aplikace má přesto přes 10 000 stažení, a tudíž ve většině případech funguje. Tipuji, že problém souvisí s geografickou lokací mobilního zařízení. Na obrázcích 2.1 lze vidět, jak aplikace vypadá.



Obrázek 2.1 Android aplikace politiscope [8]

2.1.1 Zhodnocení

Přestože se mi aplikaci nepodařilo zprovoznit, stálo za to zahrnout ji do analýzy kvůli jejím funkčnostem. Další výhodou této aplikace je také přívětivé uživatelské rozhraní a fakt, že data získává z již existujících API.

2.2 Congress

Autor: Eric Mill

Počet stažení: více než 500 000

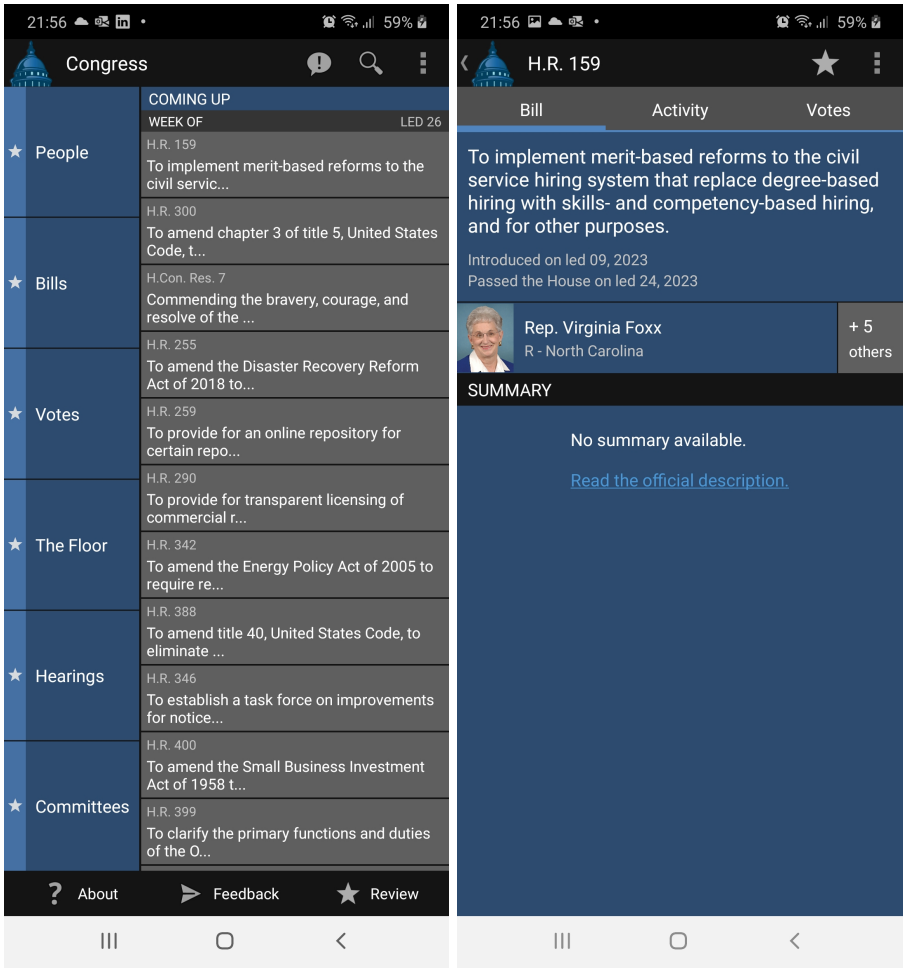
Analyzovaná verze: 4.9.2 (27. 1., 2023)

Aplikace Congress [11] dle popisu na Google Play poskytuje informace o politických reprezentantech a výsledcích jejich hlasování o návrzích zákonů ve Spojených Státech. Návrhy zákonů lze vyhledávat podle názvu.

Při spuštění aplikace uvidíme domovskou obrazovku, která obsahuje menu pro seznam politických reprezentantů, návrhů zákonů, výsledků hlasování, aktivit v rámci Kongresu, schůzek komisí a seznam komisí. Na domovské stránce uvidíme také seznam nejnovějších návrhů zákonů.

Obrazovka pro seznam politických reprezentantů obsahuje seznam aktuálně sledovaných politických reprezentatntů. Reprezentatny lze filtrovat podle státu, sněmovny a senátu. Na obrazovce konkrétního politického reprezentanta lze vidět jeho jméno, politickou stranu, příslušný stát, telefonní číslo, jak hlasoval, které zákony navrhoval, ke kterým komisím patří a odkaz na oficiální stránku s informacemi o něm.

Obrazovka pro návrhy zákonů obsahuje seznam návrhů sledovaných uživatelem, seznam aktivních návrhů a seznam nových návrhů.



Obrázek 2.2 Android aplikace Congress [11]

Na obrazovce pro výsledky hlasování lze vidět popis návrhu, výsledek hlasování o návrhu, datum a čas hlasování, a údaj o tom, zda se hlasovalo ve Sněmovně nebo Senátu. Obrazovka s detailem návrhu zákona obsahuje výsledek jeho hlasování, počet hlasování pro a proti, a počet lidí, kteří nehlasovali. Dále obsahuje informaci o tom, kolik lidí je potřeba být ve fyzické přítomnosti, aby hlasování bylo platné. Dále obrazovka obsahuje informaci o tom, jak hlasoval který politik o daném návrhu.

Obrazovka pro události v kongresu obsahuje seznam událostí. Události jsou rozdělené podle toho, zda nastaly ve Sněmovně nebo v Senátu.

Obrazovka pro schůzky komisí a obrazovka pro seznam komisí byly v době analýzy aplikace prázdné.

Na obrázcích 2.2 lze vidět, jak aplikace vypadá.

2.2.1 Zhodnocení

První dojem z této aplikace je to, že je velmi propracovaná z hlediska různorodosti poskytovaných informací. Přitom díky dobře navrženému uživatelskému rozhraní nepůsobí nepřehledně. Naopak působí velmi intuitivně. Z menu se lze dostat na jednotlivé hlavní obrazovky, které jsou dále rozděleny na taby. U návrhů zákonů lze snadno vidět výsledek jejich hlasování, jak hlasoval který

politik, proces schvalování návrhu a aktuální stav schvalování. Politiky lze vyhledávat podle jména, státu a příslušnosti ve Sněmovně nebo Senátu. Politiky a návrhy zákonů lze sledovat. Je také možné nastavit si notifikaci o změnách.

Kapitola 3
Analýza zdrojových dat

V této kapitole budou analyzována zdrojová data PSP, která budou použita pro backend.

3.1 Zdrojové soubory

Zdrojové soubory se nachází v souborech ve formátu zip, které lze stáhnout na webu PSP [5]. Pro tuto práci budou použity následující zdrojové soubory:

- **poslanci.zip** - Obsahuje datové soubory pro poslance a osoby, jejich zařazení do orgánů a orgány jako takové.
- **hl-XXXXps.zip** – Obsahuje datové soubory pro návrhy zákonů a výsledky jejich hlasování, a výsledky hlasování poslanců.

3.2 Formát dat

Data v datových souborech jsou poskytována ve formátu UNL [5], tj.:

- Každý řádek v souboru odpovídá jednom řádku v databázi.
- Oddělovačem je znak roury (|).
- Pokud je sloupec prázdný, je jeho hodnota typu `null`.
- V sloupcích jsou používány tzv. escape sekvence k zápisu speciálních znaků s úvodním znakem (backslash) následovaný znakem.

Další informace o datech:

- Kódování dat je `windows-1250`.
- Pokud bude strunktura dat doplňována, budou nové sloupce přidávány na konec.

Kódování `windows-1250` obsahuje mimo jiné všechny znaky z české abecedy. Je tedy třeba dbát na správně nastavené kódování při parsování nebo ukládání do databáze, aby datům vráceným mobilní aplikaci nechyběly např. háčky a čárky. Zároveň při parsování UNL souborů je potřeba brát ohled na přidávání nových sloupců nakonec.

3.3 Aktualizace

Web PSP [5] uvádí, že data obsahují úplný stav a že rozdílové aktualizace nejsou poskytovány. To je pro použití neideální, jelikož při aktualizaci databáze je pak potřeba z webu stáhnout znovu všechna zdrojová data a poté buď sestavit znovu celou databázi nebo zjistit rozdílový stav a na základě něho aktualizovat stávající databázi.

3.4 Datové typy

Na webu PSP [5] jsou poskytovány informace o datových typech sloupců. V tabulce 3.1 jsou uvedeny ty datové typy, které jsou použity v rámci této práce.

Typy dat sloupců v tabulkách	
Typ	**Popis**
int	integer
char(X)	textový řetězec, s blíže neuvedenou délkou
date	datum, ve formátu DD.MM.YYYY
datetime(year to hour)	datum a čas, do úrovně hodin, ve formátu YYYY-MM-DD HH
datetime(hour to minute)	čas, ve formátu HH:MM

Tabulka 3.1 Datové typy sloupců v datových souborech [5]

3.5 Licence

Data jsou poskytována bezplatně, využití dat je podmíněno uvedením zdroje dat a případně datem zpracování dat [5]. V mobilní aplikaci bude uveden zdroj.

3.6 Datové soubory

Zde budou vypsány použité datové soubory ve zdrojových souborech uvedených v podkapitole 3.1 a jejich stručný popis. Soubory budou popsány detailněji v následující sekci 3.7. Ze zdrojového souboru `poslanci.zip` budou použity následující datové soubory:

- **organy** – Reprezentuje orgány ve státní správě. Zahrnuje parlamenty všech volebních období a poslanecké kluby. Data budou použita pro nalezení poslaneckých klub.
- **poslanec** – Reprezentuje poslance, včetně těch z předchozích volebních období. Data budou použita pro získání všech poslanců v určitém volebním období.
- **osoby** – Reprezentuje osobu. Obsahuje osobní údaje jako jméno, příjmení a datum narození. Data budou použita pro zjištění osobních údajů o poslanci.
- **zarazeni** – Obsahuje zařazení osoby do nějakého orgánu. Data budou použita pro získání členů poslaneckého klubu.

Ze zdrojových souborů `hl-XXXX.zip`, kde XXXX je volební rok, budou používány následující datové soubory:

- **hl_hlasovani** – Reprezentuje výsledky hlasování o návrhů zákonů. Data budou použita pro získání informací o návrhu zákona a výsledcích jeho hlasování.
- **hl_poslanec** – Reprezentuje výsledek hlasování poslance o návrhu zákona. Data budou použita pro získání výsledků hlasování poslance o návrhu zákona.
- **omluvy** – Reprezentuje časově ohraničené omluvy z jednání a hlasování. Data budou použita pro získání počtu omluvených.

3.7 Tabulky

Na diagramu zdrojových dat C.1 jsou ilustrovány tabulky reprezentující datové soubory popsané v předchozí sekci 3.6. Diagram byl vytvořen pomocí nástroje [13]. Atribut identifikující danou tabulku je zvýrazněn tučným písmem. Některé tabulky jsou identifikovány více atributy. Ve zbytku této sekce budou detailněji popsány jednotlivé tabulky. V tabulkách budou uvedeny pouze ty atributy, kterou jsou použity v rámci této práce. Kompletní výčet atributů lze najít na webu PSP [5]. Většina poznatků ve zbytku této sekce pochází z analýzy obsahu datových souborů a nejsou na stránkách PSP nijak zdokumentována.

organy

Tabulka **organy** 3.2 bude použita pro nalezení poslaneckých klubů v určitém volebním období. Všechny poslanecké kluby mají hodnotu atributu **id_typ_organu** rovnou 1. Poslanecké kluby v daném volebním období lze získat dotazem nad všemi kluby, jejichž hodnoty atributů **od_organ** a **do_organ** spadají mezi hodnotami atributů **od_organ** a **do_organ** PSP. PSP je totiž také orgánem, kterého lze získat přes atribut **id_organ**. První PSP má identifikátor 165. PSP v každém následujícím volebním období má hodnotu identifikátoru o jednu větší než PSP v předchozím volebním období. Tedy např. PSP z volebního období 2017 až 2021 má identifikátor 172. Dále pokud hodnota atributu **do_organ** je rovna `null`, pak jde o aktuální orgán.

Tabulka 3.2 Tabulka organy

Tabulka organy		
Sloupec	**Typ**	**Použití a vazby**
`id_organ`	`int`	Identifikátor orgánu
`id_typ_organu`	`int`	Identifikátor typu orgánu
`zkratka`	`char(X)`	Zkratka orgánu
`nazev_organu_cz`	`char(X)`	Název orgánu v češtině
`od_organ`	`date`	Datum ustavení orgánu
`do_organ`	`date`	Datum ukončení orgánu

osoby

Tabulka **osoby** 3.3 bude použita pro získání osobních údajů o poslancích. Zde je zajímavé datum narození, které je 1. 1. 1900, pokud je neznámo. Zobrazovat toto datum v mobilní aplikaci při

absenci data narození poslance by nebylo ideální. V kapitole 5.2 je uvedena ztransformovaná verze tohoto data.

Tabulka 3.3 Tabulka osoby

Tabulka osoby		
Sloupec	**Typ**	**Použití a vazby**
`id_osoba`	`int`	Identifikátor osoby
`jmeno`	`char(X)`	Jméno
`prijmeni`	`char(X)`	Příjmení
`narozeni`	`date`	Datum narození, pokud neznámo, pak 1.1.1900.
`pohlavi`	`char(X)`	Pohlaví, M jako muž, ostatní hodnoty žena

zarazeni

Tabulka zařazení bude použita ke zjištění příslušnosti poslance do poslaneckého klubu. Atribut **id_of** může reprezentovat buď tabulku **organ** (tu používáme) nebo tabulku **funkce** (tu nepoužíváme, a tudíž tu není uvedena) podle toho, zda je hodnota atributu **cl_funkce** rovna 0 nebo 1. Tabulka **funkce** reprezentuje konkrétní funkci v daném orgánu. Pro účely této práce tato informace není potřeba, a proto zde tabulka **funkce** není uvedena. Je však potřeba zjistit příslušnost poslance do poslaneckého klubu, kterou lze zjistit kombinací tabulek **poslanec**, **osoby**, **zarazeni** a **organy**. Zajímat nás tedy budou pouze taková zařazení, která mají hodnotu atributu **cl_funkce** rovnou 0. Dále pokud hodnota atributu **do_o** je rovna `null`, pak jde o aktuální zařazení.

Tabulka 3.4 Tabulka zarazeni

Tabulka zarazeni		
Sloupec	**Typ**	**Použití a vazby**
`id_osoba`	`int`	Identifikátor osoby, viz osoba:id_osoba
`id_of`	`int`	Identifikátor orgánu či funkce: pokud je zároveň nastaveno zarazeni:cl_funkce == 0, pak id_o odpovídá organy:id_organ, pokud cl_funkce == 1, pak odpovídá funkce:id_funkce.
`cl_funkce`	`int`	Status členství nebo funkce: pokud je rovno 0, pak jde o členství, pokud 1, pak jde o funkci.
`od_o`	`datetime(year to hour)`	Zařazení od
`do_o`	`datetime(year to hour)`	Zařazení do

poslanec

Tabulka **poslanec** 3.5 slouží pro získání seznamu poslanců v daném volebním období. Pokud je někdo poslancem ve více volebních obdobích, objeví se v této tabulce vícekrát, pokaždé s jiným

identifikátorem. Dále pomocí atributu **id_kraj** lze skrz tabulku **organy** získat název volebního kraje. A pomocí atributu **id_obdobi** lze zjistit volební období, v kterém byl poslancem. Hodnota atributu **id_obdobi** totiž odpovídá identifikátoru PSP v určitém volebním období. Např. poslanec s hodnotou atributu **id_obdobi** rovnou 165 je poslancem PSP s identifikátorem 165. Dále někteří poslanci mají profilové foto. Po analýze fotek poslanců na webu PSP byl zjištěn následující vzor pro URL fotky poslance: `https://www.psp.cz/eknih/cdrom/XXXXps/eknih/XXXXps/poslanci/iYYY.jpg`, kde `XXXX` je identifikátor osoby a `YYY` je identifikátor PSP.

Tabulka 3.5 Tabulka poslanec

Tabulka poslanec		
Sloupec	**Typ**	**Použití a vazby**
`id_poslanec`	`int`	Identifikátor poslance
`id_osoba`	`int`	Identifikátor osoby, viz osoba:id_osoba
`id_kraj`	`int`	Volební kraj, viz organy:id_organu
`id_obdobi`	`int`	Volební období, viz organy:id_organu
`foto`	`int`	Pokud je rovno 1, pak existuje fotografie poslance

hl_hlasovani

Tabulka **hl_hlasovani** obsahuje informace o návrzích zákonů a výsledcích jejich hlasování. Číslo schůze bude použito pro sestavení URL adresu ke stenoprotokolu jednání o daném návrhu zákona. Po analýze stenoprotokolů na webu PSP bylo zjištěno, že lze sestavit URL adresu ke stenoprotokolu ale pouze na první jeho stránky. Nelze sestavit URL adresu na stránku, kde se jedná o daném návrhu zákona. Vzor pro URL adresu stenoprotokolu je `https://www.psp.cz/eknih/XXXXps/stenprot/YYYschuz`, kde `XXXX` je volební rok a `YYY` je číslo schůze. Dále podle webu se od účinnosti novely jednacího řádu 90/1995 Sb. nerozlišuje zdržel se a nehlasoval, tj. příslušné počty se sčítají. Z toho plyne, že by mobilní aplikaci měla u hlasováních uskutečněných před účinností této novely mít kolonku pro počet poslanců, kteří nehlasovali. A u hlasováních uskutečněných po účinnosti této novely by tam daná kolonka již být neměla.

Tabulka 3.6 Tabulka hl_hlasovani

Tabulka hl_hlasovani		
Sloupec	**Typ**	**Použití a vazby**
`id_hlasovani`	`int`	Identifikátor hlasování
`schuze`	`int`	Číslo schůze
`cislo`	`int`	Číslo hlasování
`datum`	`date`	Datum hlasování
`cas`	`datetime(hour to minute)`	Čas hlasování
`pro`	`int`	Počet hlasujících pro
`proti`	`int`	Počet hlasujících proti
`zdrzel`	`int`	Počet hlasujících zdržel se
`nehlasoval`	`int`	Počet přihlášených, kteří nestiskli žádné tlačítko
`prihlaseno`	`int`	Počet přihlášených poslanců
`vysledek`	`char(X)`	Výsledek: A – přijato, R – zamítnuto, jinak zmatečné hlasování
`nazev_dlouhy`	char(X)	Dlouhý název bodu hlasování

hl_poslanec

Tabulka **hl_poslanec** obsahuje výsledky hlasování poslanců o návrzích zákonů, tj. informace o tom, jak hlasoval který poslanec o kterém návrhu zákona. Výsledky 'B' a 'N' jsou interpretovány stejně, oba znamenají hlasování proti. Výsledek 'F' znamená, že poslanec nehlasoval. Používal se pouze před rokem 1995, jak bylo popsáno v předchozí sekci 3.7. Po roce 1995 má vždy hodnotu 0. Výsledek 'W' se v datech vyskytuje velmi málo a v případě, že nastane, se jeho počet přičte k počtu nepřihlášených. U výsledku 'K' se mi nepodařilo zjistit, co přesně znamená zdržel se/nehlasoval. V datech se nevyskytuje moc často. Na webu PSP je tento výsledek hlasování u jednoho návrhu zákona v roce 2013 interpretován jako nehlasoval. Z toho lze usoudit pouze to, že 'K' může znamenat, že poslanec nehlasoval. Není však z toho jasné, kdy může znamenat to, že se poslanec zdržel. Na výsledek hlasování to nemá vliv. Na výsledek hlasování má pouze vliv počet pro a proti, a tudíž bylo rozhodnuto, že výsledek 'K' bude ignorován.

Tabulka 3.7 Tabulka hl_poslanec

Tabulka hl_poslanec		
Sloupec	**Typ**	**Použití a vazby**
`id_poslanec`	`int`	Identifikátor poslance, viz poslanec:id_poslanec
`id_hlasovani`	`int`	Identifikátor hlasování, viz hl_hlasovani:id_hlasovani
`vysledek`	`char(X)`	Hlasování jednotlivého poslance. 'A' - ano, 'B' nebo 'N' – ne, 'C' – zdržel se, 'F' – nehlasoval, '@' – nepřihlášen, 'M' – omluven, 'W' – hlasování před složením slibu poslance, 'K' – zdržel se/nehlasoval.

omluvy

Tabulka **omluvy** 3.8 zaznamenává časově ohraničené omluvy poslanců z jednání PSP. Slouží pouze po spočtení počtu omluvených poslanců během hlasování. Používá se pro spočítání počtu omluvených poslanců, který v tabulce 3.6 nejsou. Podle webu PSP slouží tato tabulka pro nahrazení výsledku hlasování typu '@', tj. pokud výsledek hlasování jednotlivého poslance je nepřihlášen. Odůvodnění je, že počet omluvených je přidáván do dat se zpožděním. Dále pokud čas hlasování spadá do časového intervalu omluvy, pak se za výsledek považuje 'M', tj. omluven.

Tabulka 3.8 Tabulka omluvy

Tabulka omluvy		
Sloupec	**Typ**	**Použití a vazby**
`id_organ`	`int`	Identifikátor volebního období, viz organy:id_organ
`id_poslanec`	`int`	Identifikátor poslance, viz poslanec:id_poslanec
`den`	`date`	Datum omluvy
`od`	`datetime(hour to minute)`	Čas začátku omluvy, pokud je `null`, pak i omluvy:do je `null` a jedná se o omluvu na celý jednací den.
`do`	`datetime(hour to minute)`	Čas konce omluvy, pokud je `null`, pak i omluvy:od je `null` a jedná se o omluvu na celý jednací den.

Kapitola 4

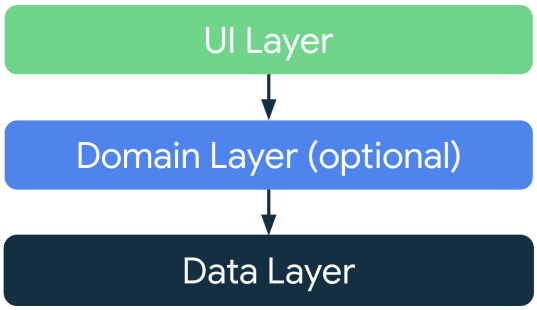
Výběr architektury

V této kapitole budou popsány možné architektury pro mobilní aplikaci a backend. Cílem je rozdělit aplikaci do různých vrstev, kde každá vrstva bude mít na starosti jednu část zodpovědnosti aplikace. komunikují mezi sebou prostřednictvím pevně definovaného rozhraní. Díky tomu změna implementace jedné vrstvy neovlivní ostatní vrstvy, pokud rozhraní mezi vrstvami zůstane stejné.

4.1 Mobilní aplikace

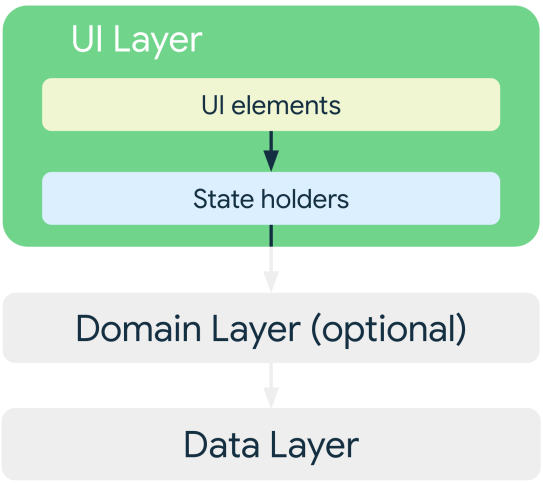
4.1.1 Architektura doporučená Googlem

Softwarovou architekturu doporučenou Googlem [14] ilustruje následující obrázek:



Obrázek 4.1 Architektura Android aplikaci [14]

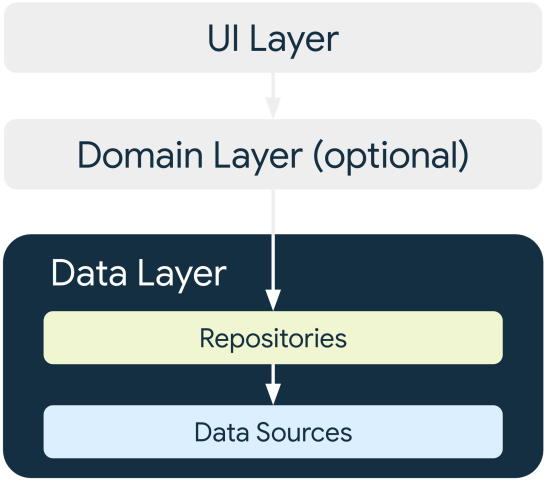
Architektura je rozdělená na UI vrstvu, doménovou vrstvu a datovou vrstvu. UI (také prezentační) vrstva má na starosti zobrazování dat v UI. UI je aktualizováno na základě změn ve stavu UI. Stav UI je měněno prostřednictvím událostí (např. kliknutí na tlačítko) nebo externích vstupů (odpověď ze síťového volání). UI vrstva je složena z UI elementů a držitelů stavu, jak je ilustrováno na následujícím obrázku:



Obrázek 4.2 Složení UI vrstvy [14]

UI elementy reprezentují elementy, které jsou vykreslovány na obrazovku. Držitelé stavů mají na starosti správu stavu UI a obsluhu událostí vyvolaných z UI. Změna stavu v držiteli stavu způsobí změnu UI elementů a znovu vykreslení obrazovky.

Datová vrstva, jak ilustruje následující obrázek, je složena z repozitářů a datových zdrojů. Repozitář vystavuje data pro zbytek aplikace, čímž ho abstrahuje od konkrétního způsobu získávání dat (např. zda jsou data získávána lokálně nebo vzdáleně). Dále repozitář slouží pro centralizaci dat, což je způsob správy dat, kdy je vytvořen jednotný zdroj pravdy, tj. všechna data v aplikaci pochází vždy z tohoto zdroje, ať už přímo nebo nepřímo. Díky tomu jsou všechna data v aplikaci konzistentní. Datový zdroj pak reprezentuje konkrétní zdroj dat, z kterého jsou získávána data. Může to být např. lokální soubor, backend nebo lokální databáze.



Obrázek 4.3 Složení datové vrstvy [14]

Doménová vrstva má na starosti zapouzdření business logiky. Díky tomu je business logika přepoužitelná ve více držitelích stavů, pokud jich používáme několik. Třídám v této vrstvě říkáme *use cases* nebo *interactors*.

4.1.2 Zhodnocení

Architektura doporučená Googlem má následující výhody:

- **Oddělení zodpovědnosti** – Rozdělení aplikace do vrstev zlepšuje udržitelnost aplikace. Vrstvy mají mezi sebou pevně definované rozhraní. Změna implementace jedné vrstvy bude mít minimální vliv na ostatní vrstvy.
- **Řízení UI podle stavu dat** – Uživatelské rozhraní automaticky reaguje na změnu stavu dat UI.
- **Jednotný zdroj pravdy** – Zajišťuje konzistenci dat.
- **Jednosměrný tok dat** – Data putují pouze zeshora dolu (např. z datového zdroje k repozitáři, z repozitáře k držiteli stavu). Události putují pouze zezdola nahoru (např. z UI elementů k držiteli stavu, z držitele stavu k repozitáři). Díky tomu je fungování aplikace predikovatelné a snadno debugovatelné.
- **Je doporučený Googlem** – Architektura je díky tomu obecně známá. Když by na této práce začal pracovat jiný Android vývojář, vyznal by se díky této architektuře v kódu lépe.

Z těchto důvodů byla pro implementaci mobilní aplikace vybrána tato architektura.

4.2 Backend

4.2.1 5-vrstvá architektura

5-vrstvá architektura [15] rozděluje aplikaci do následujících vrstev:

- **Prezentační** – Má na starosti prezentaci dat uživatelovi a obsluhu událostí. Data získává z aplikační vrstvy.
- **Aplikační** – Funguje jako prostředník mezi prezentační a businessovou vrstvou. Má na starosti aplikační logiku a řídí tok dat mezi různými vrstvami. Implementuje mimo jiné autentizaci, autorizaci a validaci dat.
- **Businessová** – Obsahuje businessovou logiku aplikace. Definuje operace, které mohou být prováděny na datech, a implementuje businessové procesy. Komunikuje s perzistentní vrstvou pro ukládání a získání dat.
- **Perzistentní** – Tato vrstva má na starosti ukládání a získávání dat z databáze. Funguje jako abstraktní rozhraní pro bussinessovou vrstvu pro přístup k datům. Data ukládá a získává prostřednictvím komunikace s API databáze v databázové vrstvě.
- **Databázová** – Má na starosti ukládání dat. Může být implementována např. pomocí relační databáze. Poskytuje data perzistentní vrstvě.

U větších projektů lze projekt také rozdělit podle funkcionalit (např. funkcionalita pro poskytování informací o hlasování a funkcionalita pro poskytování informací o poslancích) a v rámci těchto funkcionalit použít 5-vrstvou architekturu.

4.2.2 Architektura mikroslužeb

Architektura mikroslužeb je architektonický styl, který rozděluje aplikaci do služeb, což jsou komponenty s následujícími vlastnostmi:

- **Samostatně nasaditelné** – Každou službu lze nasadit zvlášť od ostatních služeb.
- **Volně propojené** – Služby komunikují mezi sebou prostřednictvím API, a nejsou tedy závislé na implementaci druhého.
- **Rozdělené podle domény** – Každá služba má na starosti určitou doménu v byznysu. Např. jedna služba má na starosti účetnictví a jiná má na starosti objednávky.
- **Vývoj v několika týmech** – Jednotlivé služby lze vyvíjet nezávisle na ostatních službách, dokud je zachováno API mezi nima.
- **Dobře testovatelné a udržitelné** – Každou službu lze testovat a udržovat zvlášť od ostatních služeb.

4.2.3 Zhodnocení

5-vrstvá architektura je pro účely této práce dostačující. Prezentační vrstva bude použita pro vytvoření endpointů REST API. Aplikační vrstva nebude pro jednoduchost implementována, a místo toho budou její zodpovědnosti implementovány v rámci businessové vrstvy. Businessová vrstva bude sloužit pro abstrahování prezentační vrstvy od perzistentní vrstvy. Zároveň bude sloužit pro validaci dat. Backend nebude mít složitou business logiku, bude pouze zpracovávat data z webu PSP a vystavovat je prostřednictvím REST API. Perzistentní vrstva bude použita pro abstrakci businessové vrstvy od databázové. To usnadní práce s databází. V rámci databázové vrstvy budou ukládáná data. Chybí tu však vrstva pro pravidelnou aktualizaci databáze. Backend bude totiž pravidelně stahovat data z webu PSP a aktualizovat databázi. K této architektuře tedy bude navíc přidána vrstva pro synchronizaci databáze se zdrojovými daty na web PSP.

Architektura mikroslužeb se hodí pro velké projekty, kde je předpokládáno, že budou často přicházet nové požadavky o nové funkcionality v různých částech backendu. V takovém případě je dobré backend rozdělit do nezávislých částí a ty vyjíjet, testovat, nasazovat a udržovat odděleně ve více týmech. Nevýhodou je větší komplexita projektu. Pro účely této práce toto rozdělení není potřeba, jelikož požadavky jsou předem dané a nepředpokládá se, že se budou v budoucnu měnit. Všechny části backendu tedy budou mezi sebou komunikovat prostřednictvím programového rozhraní, nikoliv API.

Kapitola 5
Návrh

V této kapitole bude popsán návrh uživatelského rozhraní, REST API a databázového modelu.

5.1 Uživatelské rozhraní

V této podkapitole bude popsán návrh uživatelského rozhraní. Každá sekce této podkapitoly bude popisovat návrh jedné obrazovky mobilní aplikace.

Seznam hlasování

Na obrázku A.1 je návrh obrazovky pro seznam hlasování. Ta je složena z hlavičky, seznamu hlasování a dolní navigace. Hlavička obsahuje titul identifikující danou obrazovku, aktuálně nastavené volební období a tlačítko pro filtrování seznamu. Každé hlasování v seznamu obsahuje popis návrhu zákona, datum a čas hlasování, výsledek hlasování ve formě ikonky a textu, a indikátor pro kliknutí. Dolní navigace slouží pro navigaci mezi hlavními obrazovkami, tj. mezi obrazovkou pro seznam hlasování, seznam poslanců a nastavení.

Kliknutím na tlačítko pro filtrování seznamu se zobrazí vyhledávací pole A.2, pomocí kterého lze seznam filtrovat podle popisu návrhů zákonů. Pole obsahuje placeholder text, tlačítko pro smazání textu a tlačítko pro schování vyhledávacího pole.

Detail hlasování

Obrazovka pro detail hlasování A.3 je složena z hlavičky a obsahu. Hlavička obsahuje tlačítko pro navigaci zpět. Obsah je rozdělen do dvou tabů. V prvním tabu se nachází obecné informace o daném návrhu zákona a výsledcích jeho hlasování. Tyto informace zahrnují popis návrhu zákona, datum a čas hlasování, odkaz na stránku se stenoprotokolem, odkaz na oficiální zdroj a tabulku se statistikou toho, jak se hlasovalo. Typy výsledků hlasování jsou popsány textově a pomocí ikonky.

Druhý tab A.4 obsahuje seznam poslaneckých klubů rozdělených do boxů. V každém boxu je název klubu, jeho logo, pokud je k dispozici, a indikátor pro expandování boxu. Po expandování boxu klubu lze vidět informace o tom, jak o daném návrhu zákona hlasoval daný klub a jeho členové. Výsledek hlasování členů je znázorněno ikonkou.

Seznam poslanců

Obrazovka pro seznam poslanců A.6 vypadá obdobně jako obrazovka pro seznam hlasování. Obsahuje hlavičku s titulem, aktuálně nastaveným volebním obdobím a tlačítkém pro filtrování seznamu. Pod hlavičkou je seznam poslanců a dolní navigace. Každý poslanec má profilovou fotku, jméno a příjmení, volební kraj, poslanecký klub a indikátor pro kliknutí. Kliknutím se dostaneme na obrazovku s detailem daného poslance. Dále pomocí vyhledávacího pole lze seznam poslanců filtrovat podle jména A.7.

Detail poslance

Na obrazovce s detailem poslance A.8 lze vidět údaje o daném poslanci a informace o tom, jak hlasoval o jednotlivých návrzích zákonů. Obrazovka je rozdělena na hlavičku a obsah. Hlavička obsahuje tlačítko pro navigaci zpět. Obsah je rozdělen do dvou tabů. První tab obsahuje údaje o daném poslanci, tj. profilovou fotku, jméno a příjmení, datum a narození, datum začátku mandátu poslance, poslanecký klub, a volební kraj. Druhý tab A.9 obsahuje seznam návrhů zákonů s výsledky jeho hlasování o daných návrzích.

Nastavení

Na obrazovce pro nastavení A.11 je seznam nastavení dané aplikace. Obsahuje nastavení pro volební období. Nastavení obsahuje ikonku znázorňující typ nastavení, název nastavení a text nastaveného volebního obdobi. Kliknutím na toto nastavení vyskočí do popředí okno A.11 se seznamem volebních období. Po zvolení volebního období uživatel klikne na tlačítko Uložit pro uložení nebo na tlačítko Zrušit pro zrušení výběru. Pod nastavením se nachází text, kde je uveden zdroj dat, které jsou mobilní aplikací poskytovány a zdroj ikonky aplikace.

5.2 REST API

Mobilní aplikace komunikuje s backendem pomocí REST API. Tato kapitola popisuje endpointy tohoto API, jeho vstupy a výstupy.

HTTP hlavička

- **prev** – Odkaz na předchozí stránku. `null`, pokud aktuální stránka je první.
- **next** – Odkaz na následující stránku. `null`, pokud aktuální stránka je poslední.
- **last** – Odkaz na poslední stránku.
- **self** – Odkaz na aktuální stránku.

Tyto HTTP hlavičky jsou poskytovány pouze u endpointů, které vrací stránkovaný obsah.

Query parametry

U některých endpointů lze specifikovat tyto query parametry:

- **page** – Číslo stránky stránkovaného obsahu.
- **size** – Velikost stránky stránkovaného obsahu.
- **description** – Popis pro filtrování seznamu hlasování.

- **name** – Jméno pro filtrování seznamu poslanců.
- **electionYear** – Volební rok jednoznačně určující volební období.

Endpointy

Výpis kódu 5.1 HTTP dotaz pro stav aplikace

```
1 GET /api/app
```

Vrací informace o stavu aplikace. V době psaní obsahuje pouze seznam volebních období B.1. Použije se pro obrazovku nastavení A.11.

Výpis kódu 5.2 HTTP dotaz pro seznam hlasování

```
1 GET /api/vote
```

Vrací seznam hlasování s následujícími informacemi B.2:

- identifikátor hlasování
- datum a čas hlasování
- popis hlasování
- výsledek hlasování (A – přijato, N – zamítnuto, jinak zmatečné hlasování)

Obsah je stránkovaný. Lze tedy použít následující query parametry: page, size, description a electionYear. Použije se pro obrazovku pro seznam hlasování A.1.

Výpis kódu 5.3 HTTP dotaz pro detail hlasování

```
1 GET /api/vote/{id}
```

Vrací následující informace o detailu hlasování B.3:

- identifikátor hlasování
- datum a čas hlasování
- popis hlasování
- výsledek hlasování (A – přijato, N – zamítnuto, jinak zmatečné hlasování)
- URL odkaz na příslušný stenoprotokol
- URL odkaz na oficiální zdroj
- počet hlasování pro
- počet hlasování proti
- počet nepřihlášených
- počet omluvených
- počet zdržených
- volební rok

Použije se pro obrazovku pro detail hlasování A.3.

Výpis kódu 5.4 HTTP dotaz pro výsledky hlasování klubů

```
1 GET /api/party/vote/{id}
```

Vrací následující informace o výsledcích hlasování poslaneckých klubů o daném návrhu zákona B.4:

- název klubu
- URL odkaz na logo klubu, pokud známo, jinak `null`
- identifikátor hlasování
- výsledky hlasování klubu
- výsledky hlasování členů klubu

Použije se pro obrazovku pro detail hlasování A.5.

Výpis kódu 5.5 HTTP dotaz pro seznam poslanců

```
1 GET /api/member
```

Vrací seznam poslanců s následujícími informacemi B.5:

- identifikátor
- jméno a příjmení
- poslanecký klub
- URL odkaz na profilovou fotku
- volební kraj
- volební rok

Obsah je stránkovaný. Lze tedy specifikovat query parametry: page, size, name a electionYear. Použije se pro obrazovku pro seznam poslanců A.6.

Výpis kódu 5.6 HTTP dotaz pro detail poslance

```
1 GET /api/member/{id}
```

Vrací následující informace o detailu poslance B.6:

- identifikátor
- jméno a příjmení
- pohlaví (M – muž, jinak ostatní)
- poslanecký klub
- datum začátku mandátu
- datum konce mandátu

- datum narození, pokud známo, jinak `null`
- volební kraj
- URL odkaz na profilovou fotku, pokud existuje, jinak `null`
- URL odkaz na oficiální zdroj
- volební rok

Použije se pro obrazovku A.8.

Výpis kódu 5.7 HTTP dotaz pro hlasování poslance

```
1 GET /api/member/{id}/vote
```

Vrací následující informace o hlasováních daného poslance B.7:

- obecné informace o hlasování
- výsledek hlasování poslance (A – ano, N – ne, C – zdržel se, @ – nepřihlášen, M – omluven)

Narozdíl od původní reprezentace výsledků hlasování v tabulce 3.6 bude REST API vystavovat jednodušší verzi. Výsledek 'B' bude interpretován jako výsledek 'N'. Výsledek 'W' bude interpretován jako výsledek '@'. Výsledek 'F' je ignorován, protože mobilní aplikace nebude ukazovat počet poslanců, kteří nehlasovali. Výsledek 'K' je ignorován, jelikož z dat a z dokumentace není jasné, co přesně znamená. Na výsledek hlasování to nebude mít vliv. Použije se pro obrazovku pro výsledky hlasování poslance A.9.

5.3 Databázový model

Databázový model bude obsahovat následující databázové struktury:

- **agency** C.1 – Obsahuje informace o jedntlivých orgánech. Používá se pro získání volebního kraje poslance. Volební kraj bude potřeba pro strukturu **member** popsanou níže. Dále se bude používat pro získání orgánu PSP v konkrétním volebním období. Na základě toho lze získat všechny poslanecké kluby v daném volebním období, což bude využito pro endpoint 5.4. Data pro tuto strukturu lze získat z tabulky 3.2.
- **excuse** C.2 – Obsahuje informace o časově ohraničených omluvách z jednání poslanců, které se bude používat pro získání počtu omluvených v daném hlasování. To se bude používat pro endpointy 5.3 a 5.4. Data pro tuto strukturu lze získat z tabulky 3.8.
- **member** C.3 – Obsahuje informace o poslancích. To se bude používat pro endpoint 5.6. Data pro tuto strukturu lze získat z tabulek 3.3, 3.5, 3.4 a 3.2.
- **member_vote** C.4 – Obsahuje informace o tom, kdo jak hlasoval v kterém hlasování. Používá se pro endpointy 5.4 a 5.7. Data pro tuto strukturu lze získat z tabulky 3.7.
- **membership** C.5 – Obsahuje informace časově ohraničeném zařazení osoby do orgánu. Používá se pro endpoint 5.6. Data pro strukturu lze získat z tabulky 3.4.
- **party** C.6 – Obsahuje informace o politických klubech. Používá se pro endpoint 5.6 a 5.4. Data pro tuto strukturu lze získat z tabulky 3.2.
- **vote** C.7 – Obsahuje informace o hlasováních. Používá se pro endpointy 5.2, 5.3, 5.4 a 5.7. Data pro tuto strukturu lze získat z tabulek 3.6 a 3.8.

Kapitola 6

Implementace mobilní aplikace

V rámci této kapitoly bude popsána implementace mobilní aplikace. V první podkapitole budou popsány použité nástroje a technologie. Mobilní aplikace je implementována pomocí architektury popsanou v sekci 4.1.1. V dalších podkapitolách bude popsána implementace jednotlivých vrstev aplikace.

6.1 Použité nástroje a technologie

V této sekci budou popsány hlavní nástroje a technologie použité pro implementaci mobilní aplikace.

Android Studio

Mobilní aplikace byla vytvořena ve vývojovém prostředí Android Studio, což je oficiální IDE pro vývoj mobilních aplikací pro Android [16]. Výhody tohoto IDE jsou:

- **Podpora pro Android** - Android Studio je IDE určené pro vývoj mobilních aplikací pro Android. Po spuštění IDE se hned zobrazí průvodce pro instalaci Android SDK obsahující nástroje potřebné pro vývoj aplikace. Po instalaci je SDK připravené k použití. Není tedy potřeba SDK manuálně instalovat z internetu a naimportovat ho do IDE. IDE Dále poskytuje průvodce pro vytvoření projektu, pomocí kterého lze vybrat jednu z existujících šablon pro různé typy projektů. Po zvolení šablony se vytvoří projektový adresář se zdrojovými kódy, závislostmi a konfiguračními soubory pro danou šablonu. Po vytvoření projektu je aplikace prázdná, avšak ve spustitelném stavu.
- **Emulátor** - Android Studio poskytuje vestavěný emulátor, který emuluje fyzické mobilní zařízení. Díky tomu může uživatel testovat své aplikace na různých zařízeních s různými konfiguracemi (např. různé rozměry zařízení, verze Androidu).
- **Klávesové zkratky** - Jeden z hlavních důvodů pro použití IDE založených na IDE od JetBrains jsou klávesové zkratky, které jsou stejné pro všechna IDE od JetBrains. Tyto zkratky zvyšují produktivitu vývojáře.

Gradle

Gradle je nástroj pro automatizaci sestavování programu, tj. automatizace kompilace zdrojového kódu do binárního kódu, testování a zabalení do balíčku. Toto jsou další výhody Gradlu:

- **Gradle Pluginy** - Poskytují Gradlu další nástroje jako např. možnost kompilovat programovací jazyk Kotlin, parsovat anotace nebo generovat kód na základě konfiguračního souboru.
- **Externí knihovny** - Pomocí Gradlu lze do aplikace přidat knihovny, které se stáhnou ze vzdáleného repozitáře.
- **Konfigurace** - Gradle umožňuje konfigurovat pluginy a aplikaci. U Androidu lze nakonfigurovat např. SDK verzi a verzi Kotlinu.

Alternativou je nástroj Maven, který lze také použít pro vývoj mobilní aplikace pro Android. Oba mají své výhody a nevýhody pro různé situace. Rozhodl jsem se však pro Gradle, jelikož při vytváření projektu v Android Studiu nebyla možnost výběru mezi Gradlem a Mavenem. Projekt byl automaticky nakonfigurován pomocí Gradlu. Předpokládám tedy, že Google preferuje Gradle jako nástroj pro vývoj v Android aplikací. Maven by se musel nakonfigurovat manuálně, což by pro zprovoznění aplikace bylo časově náročné, a nejspíš i zbytečné.

Kotlin

Pro mobilní aplikaci byl použit programovací jayzk Kotlin [17], který je od roku 2017 preferovaným jazykem pro Android [18]. Původním programovacím jazykem pro Android byla Java [19]. Kotlin má však oproti němu několik výhod:

- **Je stručný** - Kotlin umožňuje např. vytvořit singleton pomocí klíčového slova `object` [20].
- **Je bezpečný** - Kotlin rozlišuje `null` a `non-null` datové typy. `non-null` typy lze dereferencovat vždy, `null` typy pouze po kontrole výskytu hodnoty `null`. To je vynuceno typovým systémem Kotlinu. Díky tomu není možné zkompilovat kód, v kterém by se dereferencovala hodnota `null`, kvůli čemuž by aplikace spadla.
- **Je expresivní** - Kotlin byl navržen s důrazem na stručnost a výstižnost kódu.

Kotlin Coroutines

Dalším důvodem pro použití Kotlinu souvisí s dlouze běžícími blokujícími operacemi jako např. síťovými a databázovými operacemi. Android aplikace běží defaultně na hlavním vláknu, které má na starosti vykreslování obrazovky a obsluhu událostí. Pokud je na tomto vlákně provedena dlouze běžící blokující operaci, vlákno se zablokuje na delší dobu a nebude moct obsluhovat události. Uživatelovi se aplikace pak jeví jako zamrznutá. Jedním řešením pro tento problém je vytvořit nové vlákno a provést operaci na něm. Dvě různá vlákna běží paralelně, a tudíž operace běžící na nově vytvořeném vlákně nebude blokovat hlavní vlákno, a tudíž aplikace nezamrzne. Problémem však je, že tvorba vláken a jejich správa jsou drahé operace.

Alternativním řešením jsou Kotlin Coroutines - paměťově nenáročný způsob, jak psát paralelní kód. Přesné fungování coroutinů je nad rámec této práce, pro účely této práce však stačí vědět, že coroutines se chovají jako paměťově nenáročná vlákna. Oproti vláknům lze coroutinů spustit mnohem více (klidně v řádu stovek nebo tisíců). Zde je příklad použití coroutiny:

Výpis kódu 6.1 Příklad použití coroutiny

```
// Soubor ListViewModel.kt
class ListViewModel {

  // zbytek implementace

  init {
    viewModelScope.launch {
      // paralelne bezici operace 1
    }

    viewModelScope.launch {
      // paralelne bezici operace 2
    }
  }

  // zbytek implementace
}
```

V tomto kusu kódu je coroutina pro operaci 1 vytvořena zavoláním funkce `launch` na objektu `viewModelScope`. Funkce akceptuje callback funkci, která se zavolá v rámci vytvořené couroutiny. Díky coroutině bude funkce běžet paralelně k výpočtům spuštěných v jiných coroutinách. Obě coroutiny přitom běží na stejném vlákně. Objekt `viewModelScope` určuje scope, v rámci které běží coroutina. Ten určuje životní cyklus dané coroutiny, díky čemuž lze předejít úniku dat. Pokud je nějakým způsobem ukončen scope, jsou ukončeny všechny coroutiny bežící v rámci tohoto scopu.

Další důležitou vlastností coroutinů je to, že callback funkce, kterou volá, je suspendovatelná. To znamená:

- Deklare funkce je označená klíčovým slovem `suspend`
- Suspendovatelná funkce může volat ostatní suspendovatelné funkce. Naopak nesuspendovatelná funkce může volat **pouze** nesuspendovatelné funkce.
- Suspendovatelná funkce může coroutinu suspendovat, tj. pozastavit ji, a uvolnit tím vlákno pro použití jinou coroutinou.

Důležitý je třetí bod. Díky mechanizmu pro suspendování coroutiny mohou coroutiny bežět navzájeem paralelně. Příklad suspendovatelné funkce, která suspenduje coroutinu, je funkce `delay()`, která coroutinu suspenduje na určitý počet milisekund. Během této doby je vlákno uvolněno pro použití jinou coroutinou. Jakmile uběhne daný časový interval, coroutina získá zpátky vlákno (ne nutně původní) a pokračuje dál ve svém běhu. Častějším případem použití je např. zavolání síťové operace prostřednictvím knihovny ze třetí strany, kde je tato operace poskytována prostřednictvím suspendovatelné funkce. Po zavolání této funkce knihovna coroutinu suspenduje, díky čemuž je vlákno uvolněno pro použití jinde.

Kotlin Flow

Kotlin Flow [21] je datový typ, který reprezentuje asynchronní proud hodnot poskytující data v čase. Funguje tak, že klientský kód se k proudu zaregistruje, a stane se konzumentem dané flow. Kód produkující nové hodnoty a vkládající je do tohoto proudu je producentem dané flow. Jakmile producent vyprodukuje novou hodnotu, konzument hodnotu automaticky zkonzumuje, a na základě ní provede nějakou akci. Produkce hodnot nezačíná tehdy, když je vytvořena flow, ale až když se k ní konzument zaregistruje. Flow je asynchronní proud hodnot, tj. hodnoty proudu jsou produkovány asynchronním způsobem (např. jsou získávány vzdáleně

z backendu). Pro asynchronní produkci dat se používají coroutiny. Flow může být filtrována nebo transformována. Různé flow mohou být mezi sebou kombinovány, aby mohl konzument reagovat na produkci hodnot obout flowů. Všechny zmíněné operace na flowech lze provádět zavoláním funkcí na dané flow. Tyto funkce lze řetězit. Zde je příklad flow, která produkuje seznam hlasování:

Výpis kódu 6.2 Příklad použití flow

```
1
2  // Soubor VoteListViewModel.kt
3  class VoteListViewModel: ViewModel() {
4
5    // zbytek implementace
6
7    private val _currentElectionYear = MutableStateFlow(2021)
8    val currentElectionYear: StateFlow<Int> = _currentElectionYear
9
10   private val _searchText = MutableStateFlow("")
11   val searchText: StateFlow<String> = _searchText
12
13   init {
14     viewModelScope.launch {
15       currentElectionYear.combine(searchText, ::Pair)
16       .collectLatest { pair ->
17         // zde jsou konzumovany hodnoty pair
18         // zde jsou provadeny dalsi operace nad hodnotou pair
19       }
20     }
21   }
22
23   fun onSearchTextChange(newText: String) {
24     _searchText.value = newText
25   }
26
27   // zbytek implementace
28 }
```

Flow produkuje data asynchronně pomocí coroutinů a každá coroutina musí běžet v nějakém scopu, tudíž každá registrace konzumenta k flow musí proběhnout ve scopu. V tomto případě je použit scope `viewModelScope`, což je scope jedné z komponent Androidu, která bude vysvětlena později v implementaci. V rámci coroutiny jsou kombinovány dvě flowy, `currentElectionYear` a `searchText`. Výstupem této kombinace je jediná flow produkující hodnoty v párech. Všimněme si, že datovými typy flowů je `StateFlow`. Existence těchto flowů je závislé na životním cyklu Android komponenty, v rámci které flowy běží. Výsledná flow je zkonzumována pomocí funkce `collectLatest`. V tuto chvíli začne produkce hodnot. Jak jde však vidět, tak první flow produkuje na začátku pouze jednu hodnotu 2021 a druhá flow pouze prázdný řetězec "". Další produkce dat může proběhnout buď v reakci na nějakou událost nebo na načtení dat ze sítě, databáze, lokálního úložiště atd. Příklad kódu produkující data pro flow je funkce `onSearchTextChange`, která do dané flow jednoduše uloží novou hodnotu. Hodnota je poté v rámci `collectLatest` zkonzumována. Použití Kotlin Flow přináší následující výhody:

- Proud hodnot je asynchronní a založený na paměťově nenáročných coroutinech.
- Proudy lze elegantním způsobem filtrovat a transformovat. Tyto operace lze řetězit.
- Produkce nové hodnoty v proudu automaticky spustí kód konzumenta.
- Kód je čitelnější.

Activity a ViewModel

Android aplikace je složena z Android komponent [22], které jsou jejími základními stavebními kameny. V této práce je použita jedna z nich - `Activity`. Tato Android komponenta slouží jako vstupní bod pro vytvoření UI. UI lze v rámci aktivity dynamicky měnit. Díky tomu lze všechny obrazovky aplikace vykreslovat v rámci jedné aktivity a při navigaci na jinou obrazovku se UI v aktivitě pouze překreslí. V rámci aktivity lze použít třídu `ViewModel`, která reprezentuje držitele stavu, který byl popsán v kapitole 4.2. Ukládáme v něm stav UI a metody pro obsluhu událostí. UI se překresluje v reakci na změny ve stavu v držiteli stavu.

Hilt

Hilt je dependency injection (dále jen DI) knihovna pro Android, která umožňuje přidat závislosti (dále jen injektovat) objektům pomocí anotací [23]. Vysvětlíme si na příkladě:

Výpis kódu 6.3 Příklad použití DI pomocí knihovny Hilt

```
// Soubor VoteListViewModel.kt
@HiltViewModel
class VoteListViewModel @Inject constructor(
  private val getVotes: GetVotesUseCase,
  // dalsi zavislosti
) {
  // implementace tridy
}
```

Tento kód reprezentuje view model, který má nějaké závislosti ve svém konstruktoru. Díky anotacím `@HiltViewModel` a `@Inject` poskytovaným knihovnou Hilt jsou tyto závislosti automaticky dodány Hiltem. Předpokladem pro injektování závislostí je existence těchto závislostí v grafu závislostí spravovaném Hiltem. Tento graf je vytvořen na základě konfiguračního souboru, v kterém specifikujeme objekty a závislosti mezi nimi. Příkladem takové konfigurace je:

Výpis kódu 6.4 Ukázka konfigurace DI pro Hilt

```
// Soubor UseCaseModule.kt

@Module
@InstallIn(ViewModelComponent::class)
object UseCaseModule {

  @Provides
  fun provideGetVotesUseCase(
    voteRepository: VoteRepository,
  ): GetVotesUseCase {
    return GetVotesUseCase(voteRepository)
  }

  // dalsi konfigurace zavislosti
}
```

Konfigurace je specifikována v rámci singletonu. Anotace `@Module` říká, že daný objekt obsahuje konfiguraci objektů a jejich závislostí, které má Hilt přidat do svého grafu závislostí. Pomocí anotace `@InstallIn` specifikujeme životní cyklus pro konfigurované objekty. V tomto případě jsou závislosti vázané na životní cyklus view modelu. Pomocí anotace `Provides` specifikujeme konfiguraci objektu. Funkce v tomto případě vrací objekt typu `GetVotesUseCase`, díky čemuž je Hiltem přidán do jeho grafu závislostí, a může být tím pádem injektován. Všimněme si, že tento

objekt má závislost na objektu typu `VoteRepository`. Tato závislost je také Hiltem injektována. Závislost je nakonfigurována podobným způsobem akorát v jiném souboru. Výhody DI jsou:

- **Loose coupling** - Závislostí objektu nemusí být nutně třídá, ale nějaké rozhraní. Díky tomu objekt nezávisí na konkrétní implementaci třídy, ale akceptuje různé implementace daného rozhraní. Tyto implementace jsou dodány Hiltem a nakonfigurovány v konfiguračním souboru. Zdrojový kód aplikace je tedy upuštěn od nutnosti specifikovat konkrétní implementace rozhraní. Komponenty aplikace jsou díky tomu na sobě relativně nezávislé, protože závisí pouze na rozhraních.
- **Znovupoužitelnost** - Komponenty, které závisí na rozhraních, jsou snadno znovupoužitelné.
- **Snazší testování** - Při testování funkčnosti aplikace jsou pro závislosti často používány testovací objekty, které jsou injektovány do rozhraní komponenty, kterou testujeme.

Jetpack Compose

Jetpack Compose je Googlem doporučený způsob pro implementaci uživatelského rozhraní [24]. UI je implementováno pomocí programového kódu voláním *composable* funkcí, které reprezentují nějaký UI element na obrazovce, ať už to je pouze tlačítko nebo celá obrazovka. Jetpack Compose poskytuje nativní composable funkce např. pro tlačítko, text a obrázek, ale také funkce reprezentující kontejnery pro seskupení UI elementů např. do řádku nebo sloupce. Kombinací těchto funkcí lze vytvořit složitější funkce reprezentující složitější UI elementy. Jelikož UI elementy jsou reprezentovány funkcemi, lze je přepoužívat na více místech zavoláním dané funkce. Pomocí parametrů funkcí lze měnit vzhled a chování UI elementu. Lze mu např. změnit barvu, textový obsah, ale také přidat obsluhu pro události jako kliknutí na daný element a scrollování scrollovatelným kontejnerem. Jelikož je UI popisováno v programovým jazyce, lze využít jeho programové konstrukty jako např. cykly a podmínky. UI elementům, které jsou vytvářeny pomocí composable funkcí, se také říká UI komponenty. Zde je příklad composable funkce:

Výpis kódu 6.5 Ukázka composable funkce

```
1  // Soubor VoteListScreen.kt
2
3  @Composable
4  private fun Metadata(
5    modifier: Modifier = Modifier,
6    vote: Vote?,
7  ) {
8    Column(modifier = modifier) {
9      Description(vote?.description ?: "")
10     Spacer(Modifier.height(15.dp))
11     DateAndTime(vote?.date ?: "")
12
13     // dalsi volani composable funkci
14   }
15 }
16
17 @Composable
18 fun Description(description: String) {
19   Text(
20     text = description,
21     // dalsi atributy
22   )
23 }
```

Composable funkce jsou označené anotací `@Composable`, aby je mohl Jetpack Compose detekovat. Funkce `Metadata` volá nativní composable funkci `Column`, která repezentuje sloupec UI elementů. Parametr `modifier` akceptuje objekt typu `Modifier`, který obsahuje informace o vzhledu a chování dané komponenty. Nemusí to však být jediný parametr pro tento účel. V rámci funkce `Column` jsou volány další tři composable funkce, které jsou vykreslovány pod sebou. `Description` je vlastně definovaná composable funkce, která volá nativní composable funkci `Text` repezentující textový obsah. `Spacer` je nativní composable funkce, která vytvoří mezeru o velikosti 15 pixelů. Implementace composable funkce `DateAndtime` zde pro stručnost není uvedena, ale je to také vlastně definovaná composable funkce. Všimněme si, že composable funkce můžeme pojmenovat jak chceme. Většinou chceme funkci pojmenovat podle jejího účelu. Např. funkce `Description` reprezentuje popis a funkce `DateAndTime` reprezentuje datum a čas. Dalšími důležitými koncepty composable funkcí jsou lokální stav a rekompozice composable funkcí, které budou vysvětleny na následujícím příkladu:

Výpis kódu 6.6 Ukázka composable funkce s vnitřním stavem

```
1  // Soubor VoteDetailsScreen.kt
2
3  @Composable
4  fun VoteDetailsScreen(
5    viewModel: VoteDetailsViewModel = hiltViewModel(),
6    // zbytek parametru
7  ) {
8
9    // zbytek implementace
10
11   val voteDetailsUiState by
12     viewModel.voteDetailsUiState.collectAsStateWithLifecycle()
13
14   Info(voteDetailsUiState = voteDetailsUiState)
15 }
```

Tento kód obsahuje composable funkci pro vykreslení obrazovky pro detail hlasování. Funkce obsahuje argument `viewModel` a proměnnou `voteDetailsUiState`. Nezávislě na tom, co tyto hodnoty přesně znamenají, dochází při jejich změně k rekompozici funkce. To znamená, že je funkce knihovnou Jetpack Compose znovu zavolána, ale tentokrát s novými daty. Důležitou vlastností composable funkcí je to, že k rekompozici dochází pouze u funkcí, které se nachází v podstromu funkce, u které došlo ke změně jejího stavu. Další poznámkou je, že návratový typ funkce `collectAsStateWithLifecycle()` je obecně `State<T>`, kde T je nějaký argument datového typu. Aby změna lokální proměnné vyvolala rekompozici funkce, musí být tohoto datového typu. Ke konkrétní hodnotě typu `T` se lze dostat prostřednictvím atributu `value` třídy `State`. Díky klíčovému slovu `by` se hodnota vrácená funkcí `collectAsStateWithLifecycle()` "rozbalí", díky čemuž máme přímý přístup k vnitřní hodnotě typu `T` prostřednictvím proměnné `voteDetailsUiState`. Jinými slovy datovým typem proměnné `voteDetailsUiState` je `T` a ne `State<T>`. Mechanizmus rekompozice je přitom zachován. Výhody použití knihovny Jetpack Compose jsou tedy následující:

- **Deklarativní popis UI** – Popisujeme, jak má vypada UI a jaké jsou jeho stavy. Nepopisujeme, jak se má vytvořit UI. Díky tomu je kód pro popis UI čitelnější a UI se automaticky znovu znovu vykreslí v reakci na změnu stavu UI.
- **Stručnost a výstižnost** – Kód je stručnější a výstižnější než při použití XML layoutů, který je popsán níže.
- **Efektivní vykreslování** – Díky cílené rekompozici je znovu vykreslována pouze ta část UI, u které se změnil stav dat.
- **Jednoduché testování** – Jetpack Compose umožňuje jednoduše testovat UI.

Alternativou ke knihovně Jetpack Compose pro vytváření UI jsou XML layouty, pomocí kterých lze vytvářet UI prostřednictvím XML tagů v XML souborech. Hlavní myšlenkou tohoto přístupu je to, že oddělujeme popis uživatelského rozhraní od programového kódu, díky čemuž je kód čitelnější a udržitelnější. XML soubory obsahují pouze popis elementů. Programový kód pracuje pouze s částmi aplikace mimo UI. Míchání popisu UI a programového kódu se však nikdy nevyhneme, jelikož layoutům se musí minimálně předat aspoň data. Toho lze dosáhnout pouze propojením programového kódu s layoutem. Dále např. implementace dynamického seznamu pomocí XML layoutu vyžaduje hodně boilerplate kódu [25]. Pro Jetpack Composu je vytvoření takového seznamu otázkou pár řádků kódu [26]. Z toho důvodu jsem se rozhodl implementovat UI pomocí knihovny Jetpack Compose.

Proto DataStore

V aplikaci je možnost nastavit si aktuální volební období. Toto nastavení by si měla aplikace pamatovat i po vypnutí a opětovném aplikace. Z toho důvodu je tento údaj ukládán do lokálního perzistentního úložiště v mobilním zařízení. Jsou dvě různé knihovny, které poskytují rozhraní pro ukládání do lokálního úložiště a čtení z něj: `SharedPreferences` [27] a `DataStore` [28]. Knihovna `DataStore` poskytuje asynchronní práci s lokálním úložištěm pomocí couroutinů, knihovna `SharedPreferences` pouze synchronní. Z toho důvodu byl vybrán `DataStore`. Knihovna `DataStore` poskytuje dvě různé implementace: `Preferences DataStore` a `Proto DataStore` . Implementace `Proto DataStore` ukládá data jako instance vlastně definované třídy, která je vytvořena na základě konfiguračního souboru napsaného v jazyce používaného v `Protocol Buffers` [29]. Díky implementace přes třídu je při ukládání a čtení dat poskytována typová kontrola. Implementace `Preferences DataStore` umožňuje přistupovat k datům pomocí klíčů, nevyžaduje předem definované schéma pro data a neposkytuje typovou kontrolu. Kvůli absenci typové kontroly `Preferences DataStore` byla zvolena implementace `Proto DataStore`, která funguje následovně:

Výpis kódu 6.7 Ukázka práce s Proto DataStore

```
// Soubor user_prefs.proto
message UserPreferences {
  int32 election_year = 1;
}

// Soubor UserPreferencesRepository.kt
class UserPreferencesRepository(
  private val userPreferencesStore: DataStore<UserPreferences>
) {

  val userPreferencesFlow: Flow<UserPreferences> =
    userPreferencesStore.data
    // dalsi manipulace s flowem pro osetreni chyb

  suspend fun updateElectionYear(year: Int) {
    userPreferencesStore.updateData { preferences ->
      preferences.toBuilder().setElectionYear(year).build()
    }
  }
}
```

Zpráva (angl. *message*), jak ji nazývá Google, repezentuje strukturu lokálního úložiště. Na základě ní se při sestavení aplikace vygeneruje třídá `UserPreferences`. Třída `DataStore` poskytuje rozhraní pro práci s lokálním úložištěm. Proměnná `userPreferencesDataStore.data` je flow poskytující data z lokálního úložiště. Přes tuto proměnnou lze získat aktuálně nastavené volební období. Pro aktualizaci volebního roku je použita funkce `updateElectionYear`.

6.2 Implementace prezentační vrstvy

Popis implementace prezentační vrstvy bude rozdělena na popis vstupního bodu do aplikace, popis implementace UI elementů a popis držitelů stavu.

Vstupní bod

Vstupním bodem je třída `MainActivity`, která dědí ze třídy `Activity`. Reprezentuje tedy obrazovku. Jelikož používáme Jetpack Compose, je použita pouze jedna aktivita a v rámci ní bude dynamicky měněn obsah. Tato třída je Android komponenta, a proto má životní cyklus. Nás zajímá především fáze životního cyklu, kdy je aktivita vytvořena. Jakmile se aktivita nachází v této fázi, zavolá se její funkce `onCreate()`, kterou lze v podtřídě přepsat. V této funkce je inicializována aplikace a zavolána kořenová composable funkce:

Výpis kódu 6.8 Třída activity

```
1  // Soubor MainActivity.kt
2
3  @AndroidEntryPoint
4  class MainActivity : ComponentActivity() {
5
6    // zbytek implementace
7
8    override fun onCreate(savedInstanceState: Bundle?) {
9      super.onCreate(savedInstanceState)
10
11     init()
12
13     setContent {
14       PspApp()
15     }
16   }
17
18   // zbytek implementace
19 }
```

Anotace `@AndroidEntryPoint` je informace pro Hilt, že může injektovat závislosti do tříd vytvořených v rámci této Android komponenty. Pomocí funkce `setContent` je zavolána kořenová composable funkce `PspApp`. V rámci funkce `init` je z backendu získán seznam volebních roků a uložen do lokálního úložiště:

Výpis kódu 6.9 Třída activity

```
// Soubor MainActivity.kt

@Inject
lateinit var userPreferencesRepository: UserPreferencesRepository

@Inject
lateinit var pspApi: PspApi

private fun init() {
  initUserPreferences()
}

private fun initUserPreferences() {
  lifecycleScope.launch {
    userPreferencesRepository.userPreferencesFlow.collect {
      userPreferences ->
      if (userPreferences.electionYear == 0) {
        try {
          val appState = pspApi.getAppState()
          userPreferencesRepository
            .updateElectionYear(appState.electionYears.first())
        } catch (_: IOException) {
        }
      }
    }
  }
}
```

Operace stahování dat z backendu a uložení do lokálního úložiště jsou blokující operace, a tudíž je voláme v rámci coroutiny prostřednictvím funkce `lifecycleScope.launch`, kde `lifecycleScope` je scope aktivity. Data s volebními roky jsou získávána z backendu a uložena do lokálního úložiště.

UI elementy

UI bylo implementováno pomocí knihovny Jetpack Compose. Zde je výčet některých použitých composable funkcí a jejich popisů:

- **Column** - Sloupcový kontejner pro elementy. Hodí se pro pozicování UI elementů do sloupce. Nehodí se pro dlouhé seznamy, jelikož při rekompozici této komponenty dojde k rekompzici všech jejích podkomponent, včetně těch které nejsou na obrazovce viditelné kvůli velikosti seznamu. Kontejnerem nejde scrollovat. Používám ji pro pozicování všech elementů, které se nachází ve sloupci pod sebou.
- **LazyColumn** - Sloupcový kontejner pro elementy, který je zoptimalizovaný pro dlouhé seznamy. Při rekompozici komponenty dojde k rekompozici pouze viditelných podkomponent. Kontejnerem lze scrollovat. Používám ji pro dlouhé seznamy, tj. seznam hlasování, seznam poslanců, hlasování jednotlivých klubů a hlasování poslance.
- **Row** - Řádkový kontejner pro elementy. Používám ji pro pozicování komponent vedle sebe.
- **Text** - Textový element. Používám ji pro zobrazení textového obsahu.
- **Image** - Obrázkový element. Obrázek lze získat pomocí URL adresy. Používám ho pro zobrazení profilových fotek poslanců a logů klubů.

- **Icon** - Komponenta pro ikonku. Používám ji např. pro ikonky v hlavičce a v dolní liště.
- **IconButton** - Tlačítko s ikonkou uprostřed. Používám ho pro vyhledávací tlačítko.
- **OutlinedButton** - Tlačítko s kontrastem barev mezi pozadím a obvodem. Používám ho pro tlačíka v popupu na obrazovce nastavení.
- **OutlinedTextField** - Textové pole s kontrastem barev mezi pozadím a obvodem. Používám ho pro vyhledávací pole.
- **FloatingActionButton** - Plovoucí tlačítko. Používám ho pro skok na začátek dlouhých seznamu.
- **Scaffold** - Pomocný kontejner pro seskupení komponent na obrazovce. Umožňuje jednoduchým způsobem přidat plovoucí tlačítko a automaticky ho napozicovat. Používám jeden sdílený mezi všemi obrazovkami. Každá obrazovka s dlouhým seznamem má taktéž tuto komponentu, pro přidání plovoucího tlačítka.
- **TopAppBar** - Horní lišta.
- **Spacer** - Vytváří mezeru mezi dvěma UI elementy. zda je mezera horizontální nebo veritkální, specifikujeme skrz parametr.
- **Divider** - Oddělovač. Dá se nastavit na vertikální nebo horizontální.
- **CompositionLocalProvider** - Umožňuje přepsat hodnotu v kontextu, který obsahuje globální hodnoty dostupné v rámci určitého podstromu komponent. Používám ji pro lokální přepsání globální barvy, a tím nastylování textů pro data a časy.
- **Box** - Kontejner umožňující skládat UI elementy na sebe. Používám ho pro vytvoření popupu v nastavení. Na pozadí je seznam nastavení a na popředí je daný popup.
- **SettingsMenuLink** - Composable funkce z knihovny Alorma [30] pro snadné vytvoření nastavení s defaultním vzhledem.
- **ListItemPicker** - Composable funkce z knihovny Alorma reprezentující seznam hodnot, kterým lze táhnout. Knihovna jí definuje defaultní vzhled.
- **Card** - Karta obsahující libovolný obsah a prvek pro akci. Používám ji např. pro elementy v seznamu hlasování.
- **TabRow** - Lišta s taby. Používám ji na obrazovce pro detail hlasování a na obrazovce pro detail poslance.
- **Tab** - Reprezentuje konkrétní obrazovku v rámci tabu.
- **HorizontalPager** - Layout umožňující horizontálně scrollovat obsahem. Používám ho pro scrollování mezi taby.

Zde je seznam souborů, v kterých je vytvářeno UI. Každý z těchto souborů odpovídá jedné obrazovce:

- **VoteListScreen.kt** - Obrazovka pro seznam hlasování.
- **VoteScreen.kt** - Obrazovka pro detail hlasování.
- **MemberListScreen.kt** - Obrazovka pro seznam poslanců.
- **MemberScreen.kt** - Obrazovka pro detail poslance.

- **SettingsScreen.kt** - Obrazovka pro nastavení.

Všechny UI komponenty v jiných souborech jsou používány v rámci těchto souborů. Každá komponenta je implementována podobně. Nejdřív je definován stav funkce a poté následuje popis UI. Např. UI komponenta pro seznam hlasování vypadá následovně:

Výpis kódu 6.10 Komponenta pro seznam hlasování

```
// Soubor VoteListScreen.kt.kt

@Composable
fun VoteListScreen(
  modifier: Modifier = Modifier,
  onVoteClick: (id: Int) -> Unit,
  viewModel: VoteListViewModel = hiltViewModel()
) {
  val fetchState by viewModel.fetchState
  val refreshing by viewModel.isRefreshing
  val pullRefreshState = rememberPullRefreshState(
    refreshing,
  { viewModel.refresh() }
  )

  val votePagingItems = viewModel.votes.collectAsLazyPagingItems()

  val searchText by viewModel.searchText.collectAsStateWithLifecycle()
  val isSearchBarExpanded = viewModel.isSearchBarExpanded.value

  val electionYearRange by
    viewModel.electionYearRange.collectAsStateWithLifecycle("")

  Column(modifier = modifier) {
    // zbytek implementace
  }
```

Na začátku funkce je definován stav:

- **fetchState** – Příznak pro zjištění, zda je přístup k backendu v pořádku nebo zda nastal nějaký problém jako např. backend hází chybu 500 nebo mobilní zařízení není připojeno k internetu.
- **refreshing** – Příznak, zda je seznam obnovován, tj. zda jsou data z backendu opětovně stahována. Toho lze dosáhnout táhnutím obrazovky dolu.
- **pullRefreshState** – Stav pro obnovy. Tento objekt je z externí knihovny material a má na starosti sledování událostí pro táhnutí obrazovkou dolu.
- **votePagingItems** – Stránkovaný seznam hlasování.
- **searchText** – Aktuálně napsaný text ve vyhledávacím poli.
- **isSearchBarExpanded** – Příznak, zda je vyhledávací pole expandováno.
- **electionYearRange** – Textová reprezentace volebního období.

Po definici stavu je vytvořeno UI s využitím daných stavů. Komponenta pro seznam poslanců vypadá téměř stejně. Ostatní komponenty jsou psány stejným stylem. Navigace je implementována pomocí UI komponenty `NavHost`:

Výpis kódu 6.11 Komponenta pro navigaci

```
// Soubor PspNavHost.kt

@Composable
fun PspNavHost(
  modifier: Modifier = Modifier,
  navController: NavHostController,
  onBackClick: () -> Unit
) {
  NavHost(
    modifier = modifier,
    navController = navController,
    startDestination = VotesDestination.route
  ) {
    composable(route = VotesDestination.route) {
      VoteListScreen(
        onVoteClick = { id ->
          navController.navigateToVoteDetails(id)
        }
      )
    }

    composable(
      route = VoteDetailDestination.routeWithArgs,
      arguments = VoteDetailDestination.arguments
    ) { navBackStackEntry ->
      val voteId = navBackStackEntry.arguments!!
        .getInt(VoteDetailDestination.voteIdArg)

      VoteScreen(
        voteId = voteId,
        onBackClick = onBackClick
      )
    }
  }
}
```

Pomocí funkce `NavHost` vytváříme destinace pro navigaci. Každá destinace je vytvořena pomocí funkce `composable`, která v prvním parametru akceptuje název identifikující danou destinaci a v druhém callback funkci, která se zavolá, když dojde k navigaci k této destinaci. Samotnou navigaci má na starosti objekt `NavHostController`. Při navigaci na detail entity specifikujeme navíc argument s identifikátorem dané entity. Tento argument se musí vyskytovat v definici destinace a musí být specifikován během navigaci.

Držitelé stavů

Držitel stavu obsahuje stav UI a metody pro obsluhu událostí z UI. Data získává z doménové vrstvy. Je implementován pomocí view modelů. Všechny stavy v modelech jsou buď definovaný pomocí datového typu `State` nebo `StateFlow`. Datový typ `StateFlow` je flow, jejíž životní cyklus se řídí podle životního cyklu Android komponenty, v rámci které je vytvořena. V tomto případě je Android komponentou view model. Composable funkce reagují pouze na změny proměnných typu `State`. Proměnné typu `StateFlow` se v composable funkcích musí převést na typ `State` pomocí funkce `collectAsStateWithLifecycle`. Pokud je stavem ve view modelu stránkovaný obsah, pak je typu `StateFlow<PagingData<T>>`, a pro použití v composable funkci se musí převést na typ `LazyPagingItems` pomocí funkce `collectAsLazyPagingItems`, jelikož stránkování se používá pro

dlouhé seznamy a ty jsou implementovány pomocí funkce `LazyColumn`, která vyžaduje datový typ `LazyPagingItems` [31]. Zde je příklad view modelu pro seznam hlasování:

Výpis kódu 6.12 Ukázka využití view modelu

```
// Soubor VoteListViewModel.kt
class VoteListViewModel(
...
): ListViewModel(...) {

  private val _votes: MutableStateFlow<PagingData<Vote>> =
    MutableStateFlow(PagingData.empty())
  val votes: StateFlow<PagingData<Vote>> = _votes

  init {
    viewModelScope.launch {
      currentElectionYear.combine(searchText, ::Pair)
      .collectLatest { pair ->
        getVotes(pair.first, pair.second)
        .cachedIn(viewModelScope)
        .collect { votes ->
          _votes.value = votes
        }

      }
    }
  }

}
```

Na začátku je definován stav pomocí flowů. V rámci inicializační funkce je asynchronně pomocí coroutinů získávána data z backendu a uložena do tohoto stavu.

6.3 Implementace doménové vrstvy

Doménová vrstva má na starosti business logiku aplikace. Jedinou business logikou aplikace je získávání dat různých typů. Zde je výčet souborů implementující doménovou vrstvu:

- **GetAppStateInteractor.kt** - Pro získání stavu aplikace.
- **GetMemberDetailInteractor.kt** - Pro získání detailu poslance.
- **GetMembersInteractor.kt** - Pro získání seznamu poslanců.
- **GetMemberVotesInteractor.kt** - Pro získání výsledků hlasování poslance.
- **GetPartyVotesInteractor.kt** - Pro výsledků hlasování klubu a jeho členů.
- **GetVoteDetailInteractor.kt** - Pro získání detail hlasování.
- **GetVotesInteractor.kt** - Pro získání seznamu hlasování.

Doménová vrstva slouží jako abstrakce datové vrstvy od prezentační vrstvy. Tyto třídy mohou vracet přímo data z datové vrstvy (**GetAppStateUseCase.kt**). Mohou také převádět seznam entit na datový typ flow, pokud chceme pracovat ve view modelu s flow (**GetPartyVotesUseCase.kt**). Pokud má třída vracet stránkovaný obsah, pak kód vypadá následovně:

Výpis kódu 6.13 Ukázka využití třídy doménové vrstvy pro získání stránkovaného seznamu hlasování

```
// Soubor GetVotesUseCase.kt
class GetVotesUseCase @Inject constructor(...) {

  operator fun invoke(electionYear: Int, searchText: String) = Pager(
    PagingConfig(pageSize = DEFAULT_PAGE_SIZE)
  ) {
    VotesPagingSource(
      voteRepository = voteRepository,
      electionYear = electionYear,
      searchText = searchText
    )
  }
  .flow

}
```

Je vytvořen objekt typu `Pager`, který na základě konfigurace v `PagingConfig` a zdroje dat v `VotespagingSource` poskytne flow stránkovaných hodnot. `voteRepository` je repozitář z datové vrstvy, odkud budou získávána data. `electionYear` je volební rok, aby se daly stahovat data pro určité volební období. `searchText` je řetězec pro případné filtrování seznamu.

6.4 Implementace datové vrstvy

Datová vrstva slouží pro abstrahování doménové vrstvy od konkrétních datových zdrojů. Datovým zdrojem může být např. backend nebo lokální databáze. V našem případě je použit pouze datový zdroj pro získávání data z backendu. Zde jsou soubory s repozitářemi implementujícími datovou vrstvu:

- **AppStateRepositoryImpl.kt** - Repozitář pro stav aplikace.
- **MemberRepositoryImpl.kt** - Repozitář pro poslance.
- **PartyRepositoryImpl.kt.kt** - Repozitář pro kluby.
- **VoteRepositoryImpl.kt.kt** - Repozitář pro hlasování.

Repozitáře získávají data z konkrétních datových zdrojů a mapují seznamy na flowy, pokud se nejedná o stránkovaný seznam (pak je logika pro převedení na flow ponechána na stránkovací knihovně, která bude popsána níže), a API entity na doménové entity:

Výpis kódu 6.14 Ukázka datové vrstvy pro data o stavu aplikace

```
class AppStateRepositoryImpl @Inject constructor(
private val pspRemoteDataSource: PspRemoteDataSource
) : AppStateRepository {
  override fun getAppState() = flow {
    emit(pspRemoteDataSource.getAppState().toDomain())
  }
}
```

Datové zdroje jsou pouze abstrakcí nad API pro získání dat. Datový zdroj je v souboru `PspRemoteDataSourceImpl.kt`. Zde je ukázka jeho kódu:

Výpis kódu 6.15 Ukázka datového zdroje

```
class PspRemoteDataSourceImpl @Inject constructor(
  private val pspApi: PspApi
) : PspRemoteDataSource {

  override suspend fun getAppState() =
    pspApi.getAppState()

  ...
}
```

Komunikuje se třídou `PspApi`, která již obsahuje implementační detaily pro přístup k endpontům backendu. Speciálním datovým zdrojem je stránkovací datový zdroj, který není volán z repozitáře, ale z doménové vrstvy a repozitář mu je předán v parametru. Zde je seznam souborů se stránkovacími datovými zdroji:

- **VotesPagingSource.kt** - Stránkování pro seznam hlasování.
- **MembersPagingSource.kt** - Stránkování pro seznam poslanců.
- **MemberVotesPagingSource.kt** - Stránkování pro seznam hlasování poslance.

Implementace stránkovacích zdrojů jsou příliš dlouhé, a proto zde nebude uvedena. Poskytuje funkci `load`, kterou lze přepsat. V této funkci máme k dispozici aktuální stav scrollovatelného seznamu, tj. aktuální číslo a velikost stránky. Na základě toho jsou získána data z backendu skrz síťovou vrstvu. V případě, že získání dat proběhlo v pořádku a nedošlo k žádné síťové chybě, je vracen objekt typu `LoadResult.Page`, kterému jsou předána data aktuální stránky, číslo předchozí stránky a číslo následující stránky. Pokud nastane chyba, pak je vracen objekt typu `LoadResult` reprezentující chybnou stránku.

6.5 Implementace síťové vrstvy

Síťová vrstva je implementována pomocí knihovny Retrofit, která umožňuje vytvářet HTTP dotazy pomocí funkcí a anotací:

Výpis kódu 6.16 Ukázka použití knihovny Retrofit pro získání seznamu hlasování z backendu

```
// Soubor PspApi.kt
interface PspApi {

  @GET("/api/vote")
  suspend fun getVotes(
    @Query("page") page: Int,
    @Query("size") size: Int = DEFAULT_PAGE_SIZE,
    @Query("electionYear") electionYear: Int
  ): List<VoteApiEntity>

  @GET("/api/vote/{id}")
    suspend fun getVoteDetails(
    @Path("id") id: Int,
  ): VoteDetailsApiEntity

  // zbytek dotazu
}
```

Pomocí anotace `@GET` specifikujeme URL adresu endpointu. Pomocí anotace `@Query` specifikujeme query parametry. Pomocí anotace `@Path` specifikujeme parametr v URL adrese. Data z backendu jsou ve formátu JSON. Ty se deserializují do objektů, jejichž typ je specifikován v návratovém typu funkcí.

Kapitola 7
Implementace backendu

V této kapitole bude popsána implementace backendu. V první podkapitole budou popsány použité nástroje a technologie. Backend je implementovaný pomocí více vrstvé architektury: prezentační, doménova, databázová a vrstva pro synchronizaci dat. Sekce budou rozděleny tak, aby odpovídaly těmto vrstvám.

7.1 Použité nástroje a technologie

V této sekci budou popsány hlavní nástroje a technologie použité pro implementaci backendu.

Intellij IDEA

Backend byl vyvíjen ve vývojovém prostředí Intellij IDEA [32]. Výhody použití tohoto IDE jsou:

- **Vestavěný inicializátor Spring Boot aplikací** – Pomocí tohoto inicializátor lze nakonfigurovat a nastavit potřebné závislosti ve Spring Boot[1] aplikaci jednoduše naklikáním v průvodci.
- **Klávesové zkratky** – Toto IDE je vyvíjeno společností JetBrains, a tudíž obsahuje stejné klávesové zkratky jako Android Studio.

Maven

Maven je nástroj pro automatizaci sestavování programu. Maven byl zvolen kvůli tomu, že původně bylo v plánu backend nasadit na Cloud Azure [33]. Ten poskytoval plugin pro maven, který umožňoval backend nasadit a zprovoznit pomocí jednoho příkazu. Avšak kvůli omezené možnosti využití výpočetního výkonu byl backend nakonec nasazen na CloudFIT (více v kapitole 9). Gradle má čitelnější syntaxi, nicméně výběr mezi Mavenem a Gradlem neměl na funkčnost backendu a rychlost vývoje zásadní vliv, a proto se ke Gradlu již nepřecházelo a zůstalo se u Mavenu.

Spring Boot

Spring je open-source framework pro vývoj enterprise aplikací. Obsahuje nástroje pro řešení různorodých problémů. Pro účely této práce byly využity nástroje pro vytvoření webových

[1] Technologie Spring Boot bude popsána později.

aplikací [34], které lze používat i pro implementaci REST API. Nástroje Springu jsou založeny na návrhovém vzoru dependency injection.

Spring Boot je framework, který je postavený na Springu a který má za cíl redukci boilerplate kódu a nutnost počáteční konfigurace. Toto realizuje prostřednictcvím autokonfigurace, což je vlastnost, kdy jsou jednotlivé komponenty Spring Bootu (např. rozhraní pro komunikaci s databází, webový server, ORM) automaticky nakonfigurovány pomocí defaultních hodnot. Defaultní hodnoty jsou Spring Bootem zvoleny tak, aby cílily na nejčastější použití. Díky tomu lze nainstalovat závislost a s minimálním zásahem do konfigurace ji lze rovnou použít. Např. po instalaci knihovny JDBC a MySQL konektoru stačí v konfiguračním souboru nastavit URL adresu a název databáze a databázi lze v kódu rovnou použít. Není potřeba nic navíc konfigurovat. Konfigurace lze vždy přenastavit, pokud to bude potřeba. Další výhodou Spring Bootu je to, že objekty, které se mají přidat do jejího kontejneru s grafem závislostí pro injektování, lze specifikovat anotováním tříd. Na základě určitých anotací dokáže Spring Boot tyto třídy detekovat, vytvořit z nich instanci, a uložit instanci do svého kontejneru.

Alternativou ke knihovně Spring Boot je knihovna Ktor [35]. Výhodou této knihovny je, že vývoj v ní je určený pro psaní v Kotlinu. Lze tedy využít všechny výhody tohoto jazyka. Nevýhodou je, že nepodporuje defaultně dependency injection. Musí se tedy dodatečně nainstalovat a nakonfigurovat. Další nevýhodou je chybějící autokonfigurace. Mnoho závislostí se tedy musí manuálně nakonfigurovat.

Java

Java byla od začátku hlavním programovacím jazykem pro vývoj aplikací ve Spring Bootu. Od roku 2017 přišla integrace jazyka Kotlin do Spring Bootu [36] a v dokumentaci jsou ukázkové kódy psány jak v Javě tak i Kotlinu. Podle mého názoru má Java oproti Kotlinu větší podporu v komunitě, co se týče vývoje ve Spring Bootu. Nalezení řešení pro problém ve Spring Bootu s Javou bylo jednodušší než ve Spring Bootu s Kotlinem. Intellij IDEA však poskytuje nástroje pro automatickou trasnformaci souboru v jazyce Java do souboru v Kotlinu. Výsledný kód bylo však potřeba vždy pročistit, jelikož u všech proměnných vždy obsahoval datové typy, které lze v Kotlinu v některých případech pro čitelnost vynechat. Zároveň výstupní kód někdy nebyl kvůli drobnosti kompilovatelný. Z toho důvodu jsem se rozhodl pro použití Javy. Zdá se však, že podpora pro psaní aplikací Spring Boot pomocí Kotlinu je čím dál tím větší. Když bych měl možnost backend napsat znovu, pouvažoval bych znovu o použití Kotlinu, jelikož je to velmi dobrý programovací jazyk.

7.2 Prezentační vrstva

V prezentační vrstvě jsou implementovány endpointy REST API v rámci *controllerů*, což jsou třídy obsahující definice endpointů. Endpointy lze tedy seskupit do různých tříd. Controllery získávají data z doménové vrstvy. Zde je seznam souborů s controllery:

- **VoteController** - Obsahuje endpointy pro hlasování.
- **PartyController** - Obsahuje endpointy pro kluby.
- **MemberController** - Obsahuje endpointy pro poslance.
- **AppStateController** - Obsahuje endpointy pro stav aplikace.

Zde je kus kódu pro implementaci controlleru s endpointy související s hlasováním:

Výpis kódu 7.1 Ukázka kódu pro vytvoření endpointu

```
// Soubor VoteController.java
@RestController
public class VoteController {
    private final VoteService service;
    private final VoteMapper mapper;

    // zbytek implementace

  @GetMapping("/vote/{id}")
  public DetailedVote getVote(@PathVariable Integer id) {
    Vote vote = service.getVote(id);
    return mapper.toDetailedVote(vote);
  }
```

Podle anotace `@RestController` Spring Boot pozná, že se jedná o controller a na pozadí nám vytvoří implementaci endpointů. Anotace `@GetMapping` definuje URL adresu endpointu. V parametru metody lze specifikovat URL parametr pomocí anotace `@PathVariable` a query parametry pomocí `@RequestParam`. Data získaná z doménové vrstvy (proměnná service) jsou případně transformována (proměnná mapper). U stránkovaného obsahu jsou navíc přidány HTTP hlavičky pro informace o stránkování:

Výpis kódu 7.2 Ukázka nastavení hlaviček pro stránkování

```
// Soubor PaginationHeaderGenerator.java
public static HttpHeaders buildHeaders(int totalPages, int page) {
  HttpHeaders responseHeaders = new HttpHeaders();

  // zbytek implementace

  responseHeaders.set(previousPageString, String.valueOf(prevPage));
  responseHeaders.set(nextPageString, String.valueOf(nextPage));
  responseHeaders.set(lastPageString, String.valueOf(lastPage));

  return responseHeaders;
}
```

Spring poskytuje třídu `HttpHeaders`, pomocí které lze sestavit HTTP hlavičky. Instance této třídy pak bude vracen spolu s daty.

7.3 Doménová vrstva

Doménová vrstva má na starosti business logiku aplikace a abstrahování prezentační vrstvy od implementačních detailů databázové vrstvy. V našem případě backend neobsahuje téměř žádnou business logoiku, pouze filtruje data nebo dozpracovává data, která nebyla z časových důvodů zpracována předem. Zde jsou soubory implementující doménovou vrstvu:

- **MemberService** - Obsahuje business logiku pro získání seznamu poslanců a detailu poslance.
- **PartyService** - Obsahuje business logiku pro získání seznamu poslaneckých klub. a výsledků hlasování klubu a jeho členů.
- **VoteService** - Obsahuje business logiku pro získání seznamu hlasování a detail hlasování poslance.

Pro získání seznamů je použito stránkování, které je implementováno pomocí třídy `Pageable`. Ta je předána repozitáři, který bude popsán v následující sekci:

Výpis kódu 7.3 Ukázka doménové vrstvy pro vrácení seznamu poslanců

```
// Soubor MemberService.java
public Page<Member> getMembers(PagingParams pagingParams) {

  Pageable pageable = PageableGenerator.buildPageable(pagingParams);

  if (filterName == null) {
    return memberRepository
      .findByElectionYear(electionYear, pageable);
  } else {
    // kod pro filtrovani poslancu
  }
}
```

Instance třídy `Pageable` je vytvořena následovně:

Výpis kódu 7.4 Ukázka kódu pro sestavení objektu pro stránkování

```
// Soubor PageableGenerator.java
public static Pageable buildPageable(...) {
  Pageable pageable;

  ...

  pageable = PageRequest.of(page, size, sort);
  // napr. page = 2, size = 20, sort = Sort.by("dateTime").descending())

  return pageable;
}
```

U metody pro získání detailu hlasování je dopočítán počet omluvených a nepřihlášených poslanců, který nebyl spočten při zpracování, jelikož zpracování příliš zpomaloval:

Výpis kódu 7.5 Ukázka dopočtu statistik pro detail hlasování za běhu v doménové vrstvě

```
// Soubor VoteService.java
public Vote getVote(int id) throws IOException {
  ...

  int excusedCount = ...
  int loggedOffCount = ...

  ...

  // nastaveni hodnot excusedCount a loggedOffCount
  // vraceni vysledku
}
```

Pro získání dat z repozitářů jsou využity i streamy (proudy dat, podobně jako Kotlin Flows):

Výpis kódu 7.6 Ukázka použití streamu

```
public List<Party> getParties(int electionYear) {
  return partyRepository
    .findByIdElectionYear(electionYear)
    .stream()
    .filter(Util::isRealParty)
    .collect(Collectors.toList());
}
```

Funkce `findByIdElectionYear` vrátí seznam výsledků. Po té dojde k převedení na stream. Stream lze pak filtrovat. Zde byly ukázky a vysvětlení, které by měly stačit k pochopení zbylých implementací doménové vrstvy.

7.4 Databázová vrstva

Databázová vrstva je implementována pomocí repozitářů a databázových entit. Komunikace s databází je abstrahována pomocí objektově-relačního mapování, díky kterému jsme abstrahováni od databázových tabulek a místo toho pracujeme s objekty (entitami). Pro tento účel používám knihovnu Hibernate [37]. Příkladem databázové entity je:

Výpis kódu 7.7 Entita Vote reprezentující hlasování

```
1  // Soubor Vote.java
2  // dalsi anotace
3  @Entity(name = VOTE)
4  @Getter
5  public class Vote {
6
7    @Id
8    private int id;
9
10   private LocalDateTime dateTime;
11
12   // dalsi atributy
13 }
```

Pomocí anotace `Entity` Hibernate pozná, že se jedná databázovou entitu a vytvoří mapování na tabulku v databázi. Anotace `@Getter` je z knihovny Lombok [38] a slouží pro vygenerování getterů. Anotací `@Id` specifikujeme atribut, který má být primárním klíčem v tabulce.

Pro implementaci repozitářů je použita knihovna spring-data-jpa [39], která poskytuje rozhraní `JpaRepository`, který obsahuje základní metody pro manipulaci s danou entitou jako např. `findAll()` pro získání všech záznamů z tabulky nebo `findById()` pro získání záznamu s daným id. Výhodou této knihovny je možnost vytvoření vlastních metod, kterým se říká query metody. Implementace dotazu se vygeneruje na základě pojmenování query metody. Toto pojmenování se řídí podle určitých pravidel, které jsou popsány v dokumentaci [39]. Např. následující metoda vrací seznam hlasování v daném volebním období:

Výpis kódu 7.8 Repozitář pro hlasování

```
1  // Soubor VoteRepository.java
2  @Repository
3  public interface VoteRepository extends JpaRepository<Vote, Integer> {
4
5    ...
6
7    List<Vote> findByElectionYear(int electionYear);
8
9  }
```

Zde je seznam souborů s repozitáři:

- **VoteRepository.java** - Repozitář pro dotazování se nad hlasováními.
- **MemberRepository.java** - Repozitář pro dotazování se nad poslanci.
- **MemberVoteRepository.java** - Repozitář pro dotazování se nad hlasováními poslance.

- **AgencyRepository.java** - Repozitář pro dotazování se nad orgány.
- **ExcuseRepository.java** - Repozitář pro dotazování se nad omluvami.
- **MembershipRepository.java** - Repozitář pro dotazování se nad zařazeními.
- **PartyRepository.java** - Repozitář pro dotazování se nad kluby.

7.5 Zpracování dat

Backend každý den o půl noci aktualizuje databázi podle zdrojových dat na webu PSP. Aktualizace probíhá v následujících krocích:

- **Stažení zdrojových souborů** – Zdrojové soubory jsou ve formátu zip a jsou stažené z webu PSP.
- **Extrakce datových souborů** – Ze zdrojových souborů jsou vyextrahovány datové soubory.
- **Pročištění dat** – Z datových souborů jsou odstraněny duplicitní data.
- **Parsování dat** – Datové soubory jsou zparsovány a načteny do Java objektů.
- **Transformace dat** – Objekty jsou ztransformovány do databázového modelu.
- **Uložení dat do databáze** – Ztransfomovaná data jsou perzistentně uložena do databáze.

Stahování zdrojových souborů

Stahování zdrojových souborů má na starosti třída následující třída:

Výpis kódu 7.9 Třída pro stahování zdrojových souborů

```
1 // Soubor PspFilesDownloader.java
2 public class PspFilesDownloader {
3   public static void downloadFiles() throws IOException {
4     downloadHlasovani();
5     downloadPoslanci();
6   }
7
8   // dalsi metody
9 }
```

Pro stahování souborů je použita funkce `copyURLToFile` z knihovny `commons-io` [40]. Tato funkce stáhne soubor na dané URL do dané složky a s daným názvem:

Výpis kódu 7.10 Ukázka stahování dat pomocí knihovny `commons-io`

```
// Soubor FileDownloader.java
public class FileDownloader {

  public static void download(
    String downloadUrlString,
    String downloadDestination) {

    File file = new File(downloadDestination);
    URL downloadUrl = new URL(downloadUrlString);

    FileUtils.copyURLToFile(downloadUrl, file);

    ...
  }
}
```

Extrakce datových souborů

Pro extrakci souborů ze zipu byla použita funkce `extractFile` třídy `ZipFile` z knihovny `zip4j` [41]. Ta na základě názvu souboru, který se má vyextrahovat, vyextrahuje daný soubor do dané složky a soubor bude mít daný název:

Výpis kódu 7.11 Ukázka extrakce souborů ze zipu

```
// Soubor ZipExtractor.java
public class ZipExtractor {

  public static void extract(
    String pathToZip,
    String fileToExtract,
    String destinationDir
  ) {
    ZipFile zipFile = new ZipFile(pathToZip)
    zipFile.extractFile(fileToExtract, destinationDir, fileToExtract);

  }

}
```

Pročištění dat

Duplicitní záznamy jsou odstraněny pomocí skriptu napsaného v jazyce Bash:

Výpis kódu 7.12 Skript pro odstranění duplicitních řádků

```
// Soubor removeDuplicates.sh.java
for FILE in "$1"/*; do
  sort "$FILE" | uniq > 'tmp.unl'
  mv 'tmp.unl' "$FILE"
  rm 'tmp.unl'
done
```

Skript funguje následovně:

- Skript očekává ve svém prvním argumentu cestu k adresáři s datovými soubory.

- Na začátku se iteruje přes všechny soubory v daném adresáři.
- Každý soubor se pomocí příkazu `sort` seřadí vzestupně podle abecedy.
- Ze seřazeného souboru se odstraní duplicitní řádky jdoucí za sebou pomocí příkazu `uniq`. Vy tuto chvíli jsou ze souboru odstraněny všechny duplicity.
- Zbytek kódu již je pouze přesouvání obsahů souborů tak, aby datové soubory s odstraněnými duplicitami měly jejich originální název.

Skript je volán ze souboru `PspFilesCleaner`. Důvodem pro odstraňování duplicit pomocí jazyka Bash a ne přímo pomocí Javy je rychlost Bashe pro tento účel a jednoduchá manipulace se soubory.

Parsování dat

Pro parsování zdrojových dat používám knihovnu `opencsv`. Ta umožňuje parsovat CSV přidáním anotací k atributům třídy. Tyto anotace specifikují pozici sloupce v CSV souboru, na který se atribut namapuje:

Výpis kódu 7.13 Parsování datového souboru omluvy.unl

```
1  // Soubor Omluva.java
2  public class Omluva {
3    @CsvBindByPosition(position = 0)
4    private int idOrgan;
5
6    @CsvBindByPosition(position = 1)
7    private int idPoslanec;
8
9    // dalsi atributy
10 }
```

Všechny parsery se nachází ve složce `parsers`. Pro parsování je použita knihovna *opencsv* [42]. Ta poskytuje třídu `CsvToBeanBuilder`, pomocí které lze parsovat CSV soubory a namapovat je na objekty vytvořené pomocí anotací, jak je popsáno výše. Dále poskytuje metody pro nastavení oddělovače, detekci nullové hodnoty a možnost ignorovat uvozovky

Výpis kódu 7.14 Parsování datového souboru omluvy.unl

```
1  // Soubor OmluvyParser.java
2  public class OmluvyParser extends UnlParser {
3    public List<Omluva> read() {
4      String filePath = PspPath.Unl.OMLUVY;
5      BufferedReader reader = getReader(filePath);
6
7      return new CsvToBeanBuilder<Omluva>(reader)
8      .withType(Omluva.class)
9      .withSeparator(UNL_SEPARATOR)
10     .withFieldAsNull(UNL_NULL_SEPARATOR)
11     .withIgnoreQuotations(true)
12     .build()
13     .parse();
14
15   }
16 }
```

Transformace a uložení dat

Po zparsování datových souborů a namapování záznamů na objekty lze tyto objekty začít ztransformovat do databázového modelu. Všechny soubory pro transformaci zdrojových dat do databázového modelu se nachází ve složce `loaders`. V rámci transformátoru se vždy zavolá příslušný parser, který vrátí data, ty se ztranformují a následně uloží do databáze:

Výpis kódu 7.15 Transformace objektu Omluva na databázový objekt Excuse

```
1  // Soubor ExcuseLoader.java
2
3  public class ExcuseLoader extends BaseLoader {
4
5      private final OmluvyParser omluvyReader;
6      private final ExcuseRepository excuseRepository;
7
8      public void load() {
9        excuseRepository.deleteAllInBatch();
10       List<Omluva> omluvaList = omluvyReader.read();
11
12       List<Excuse> excuses = omluvaList.parallelStream()
13             .map(omluva -> {
14               // transformace
15             })
16           .collect(Collectors.toList());
17
18       Lists
19         .partition(excuses, BATCH_SIZE)
20         .forEach(excuseRepository::saveAll);
21
22   }
23
24 }
```

Před samotným parsováním jsou smazány všechny záznamy v příslušné tabulce. Poté probíhá parsování datových souborů a seznamu namapovaných objekt. Z tohoto seznamu je vytvořen paralelní stream, , který umožňuje mapovat načtené prvky seznamu do databázových entit paralelně. Databázové entity jsou následně perzistetně uloženy do databáze. Ukládání je optimalizováno tak, že se neukládá po jednom ale po skupinách. Každá skupina obsahuje 1000 objektů. Ukládáme tedy po 1000 objektech.

Kapitola 8
Testování

V této kapitole bude popsáno testování mobilní aplikace a backendu. Při testování mobilní aplikace byl kladen důraz na uživatelské testování, kdy je aplikace otestována reálným uživatelem. Dále byla otestována doménová a datová vrstva aplikace prostřednictvím automatizovaných unit testů. Na závěr byly přidány ještě UI testy které simulovali uživatelskou interakci s aplikací. Při testování backendu byl kladen důraz na testování prezentační vrstvy.

8.1 Mobilní aplikace

8.1.1 Uživatelské testování

Cílem uživatelského testování je sledovat chování uživatelů při používání aplikace, a odhalovat tím případné chyby nebo nedostatky. Testování se zúčastnili dva uživatelé, kteří dostali několik úkolů, přičemž se sledoval jejich postup řešení. Od účastníků byla na konci poskytnuta zpětná vazba k aplikaci. Účastníci používali následující mobilní zařízení:

Tabulka 8.1 Uživatelé a konfigurace zařízení

	Název zařízení	Displej	Verze OS
`Uživatel 1`	`Poco X3`	`2400 x 1080 6.67 "`	13.0
`Uživatel 2`	`Samsung Galaxy M20`	`2340 x 1080 6.3 "`	10.0

Uživatelé dostali postupně následující úkoly:

1. **Nalezněte detail libovolného hlasování.**
   - Uživatel 1 vykonal test bez problémů.
   - Uživatel 2 vykonal test bez problémů.
2. **Nalezněte informace o počtu hlasování pro a proti**
   - Uživatel 1 vykonal test bez problémů.
   - Uživatel 2 vykonal test bez problémů.
3. **Nalezněte odkaz na oficiální zdroj hlasování**

- Uživatel 1 vykonal test bez problémů.
- Uživatel 2 vykonal test bez problémů.

4. **Nalezněte informace o tom, jak hlasovali kluby a jejich členové**
   - Uživatel 1 vykonal test bez problémů.
   - Uživatel 2 vykonal test bez problémů.
5. **Nalezněte detail poslance**
   - Uživatel 1 vykonal test bez problémů.
   - Uživatel 2 vykonal test bez problémů.
6. **Nalezněte informace o tom, jak hlasoval libovolný poslanec**
   - Uživatelovi 1 vykonal test bez problémů, na začátku však navigoval k detailu hlasování, kde si zobrazil výsledky hlasování členů klubu, jehož je daný poslanec členem. Poté si uvědomil, že k výsledku hlasování člena se lze dostat jednodušeji prostřednictvím detailu poslance.
   - Uživatel 2 vykonal test bez problémů.
7. **Vyfiltrujte seznam hlasování**
   - Uživatel 1 vykonal test bez problémů.
   - Uživatel 2 vykonal test bez problémů.
8. **Vyfiltrujte seznam poslanců**
   - Uživatel 1 vykonal test bez problémů.
   - Uživatel 2 vykonal test bez problémů.
9. **Nastavte jiné volební období**
   - Uživatel 1 vykonal test bez problémů.
   - Uživatel 2 vykonal test bez problémů.

Vyhodnocení

Uživatelské testování proběhlo v pořádku. Jediný zádrhel byl v úkolu při nalézání výsledku hlasování konkrétního poslance, jelikož se tato informace nachází na dvou různých místech. Poté co se uživatel však seznámil s aplikací, se mu již aplikace používala pohodlně. Od jednoho uživatele bylo oceněno, že při nalézání informací není potřeba se proklikat aplikací příliš hluboko.

8.1.2 Unit Testy

Unit testy jsou automatizované testy pro ověření funkčnosti dílčích částí zdrojového kódu (např. funkce a třídy). Tyto dílčí části jsou testovány v izolaci, tj. nejsou závislé na jiných částech. V praxi to znamená, že pokud testujeme např. třídu, která obsahuje referenci na objekt typu rozhraní, pak pro toto rozhraní vytvoříme dvě různé implementace. Jedna implementace bude obsahovat skutečnou funkcionalitu daného objektu. Druhá implementace bude testovací a bude funkcionalitu skutečného objektu pouze simulovat. Díky tomu můžeme třídy testovat nezávisle na sobě. Unit testy se nachází ve složce `test`. Byla testována základní funkčnost doménové

(interactory a stránkovací mechanizmus) a datové vrstvy (datové zdroje). Pro interactory byly vytvořeny testovací implementace repozitářů. Pro datové zdroje byla vytvořena testovací implementace Retrofit rozhraní, které simulovalo síťové operace. Závislosti byly vkládány pomocí DI knihovny Hilt. Zde je výčet testovacích souborů s unit testy:

- **PspRemoteDataSourceImplTest** - Test vzdáleného datového zdroje.
- **MembersPagingSourceTest** - Test stránkovacího mechanizmu pro seznam poslanců.
- **MemberVotesPagingSourceTest** - Test stránkovacího mechanizmu pro seznam hlasování poslanců.
- **VotesPagingSourceTest** - Test stránkovacího mechanizmu pro seznam hlasování.
- **GetAppStateInteractorImplTest** - Test business logiky pro získání stavu aplikace.
- **GetMemberDetailsInteractorImplTest** - Test business logiky pro získání detail poslance.
- **GetPartyVotesInteractorImplTest** - Test business logiky pro získání hlasování klubů.
- **GetVoteDetailsInteractorImplTest** - Test business logiky pro získání detailu hlasování.

8.1.3 UI testy

UI testy jsou automatické testy pro testování uživatelského rozhraní. V této aplikaci vyžadují fyzické zařízení nebo emulátor. Pro toto testování byla použita knihovna Jetpack Compose , jelikož UI je implementováno taktéž pomocí knihovny Jetpack Compose [43]. Ta poskytuje objekty a metody pro dotazování se nad komponentami a jejich informace na obrazovce, a události jako kliknutí na tlačítko nebo scrollování. Tyto testy se nachází ve složce `androidTest`. Testovány byly komponenty reprezentující obrazovku. Zde je výčet testovacích souborů s UI testy:

- **VoteListScreenTest** - Testování obrazovky pro seznam hlasování.
- **PartyVotesScreenTest** - Testování obrazovky pro seznam hlasování klubů a jejich členů.
- **VoteDetailsScreenTest** - Testování obrazovky pro seznam detail hlasování.
- **MemberListScreenTest** - Testování obrazovky pro seznam poslanců.
- **MemberDetailsScreenTest** - Testování obrazovky pro detail poslance.
- **SettingsScreenKtTest** - Testování obrazovky pro nastavení.

8.2 Backend

Na backendu byla testována prezentační vrstva, kde se především ověřovalo, zda JSON vracený z REST API je ve správném formátu a zda vrací korektní data. Testování bylo implementováno pomocí knihovny Spring Boot. Testování bylo prováděno podle návodu na stránkách Spring Bootu [44], kde se jednotlivé controllery v prezentační vrstvě testují jako unit testy. Ty jsou závislé na komponentách service z doménové vrstvy a mapper pro mapování objektů. Místo vytvoření rozhraní pro ně a testovací implementace, jako se to dělalo u mobilní aplikace, jsou tyto komponenty mockovány, tj. jsou nahrazeny za jeho testovací imitaci, která neprovádí žádnou funkci a jen se tváří jako původní objekt [45]. Aby prováděla nějakou funkci, musíme ji to explicitně nastavit prostřednictvím mockovací knihovny Mockito [46]. Testy se nachází ve složce `test`:

- **MemberControllerTest** - Testy pro endpointy pro poslance.
- **PartyControllerTest** - Testy pro endpointy pro kluby.
- **VoteControllerTest** - Testy pro endpointy pro hlasování.

Kapitola 9
Nasazení

Tato kapitola popisuje způsob publikování mobilní aplikace a nasazení backendu na prodoukci.

9.1 Mobilní aplikace

V plánu je mobilní aplikaci publikovat na Google Play. Před tím je však potřeba nechat aplikaci projít kontrolní fází, kdy jsou Googlem ověřovány různé podmínky, které aplikace musí splňovat, aby mohla být publikována. To může trvat několik dní a v době psaní práce je aplikace stále v kontrolní fázi od 11. 2. 2023. Není mi známo, kdy bude ověřování dokončeno, aplikaci lze však nainstalovat pomocí APK souboru, který se nachází spolu se zdrojovým kódem v přiloženém CD. Soubor APK je potřeba přesunout z CD to mobilního zařízení. V souborovém systému zařízení pak otevřeme daný soubor a nainstalujeme aplikaci do mobilu.

9.2 Backend

Backend je nasazený na cloudu Fakulty informačních technologií ČVUT a je dostupný na URL adrese `https://psp-server.ksi.fit.cvut.cz/`. Pro distribuci byla použita technologie Docker, která umožňuje jednoduchým způsobem zdrojový kód zabalit distribuovat na server.

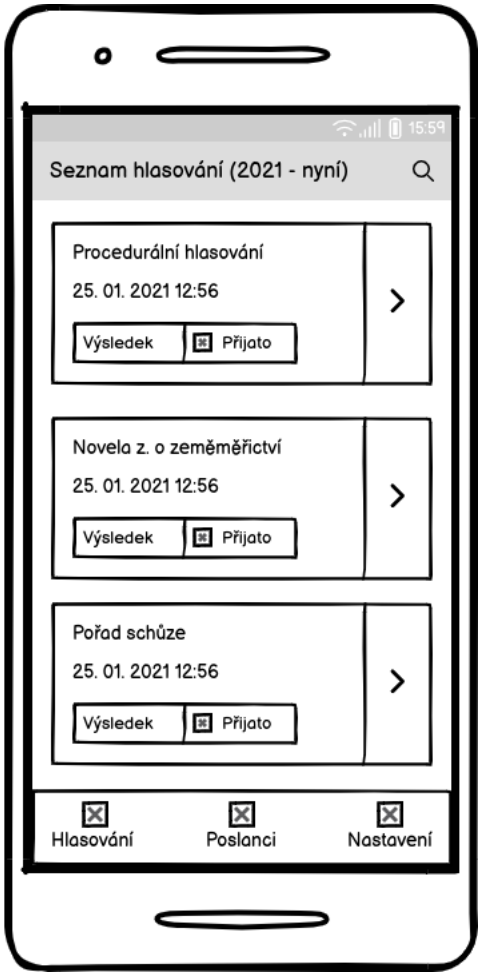
Kapitola 10
Závěr

Podle zadání byly vytvořena mobilní aplikace, která bude zobrazovat souhrnný výsledek hlasování, výsledek hlasování s rozpisem hlasování jednotlivých poslanců, profil poslance obsahující seznam jeho hlasů a odkazy na web PS. Zároveň byla implementována backend služba poskytující data pro mobilní aplikaci prostřednictvím REST API. S tím byla analyzována zdrojová data poskytovaná PSP a popsána jejich struktura. Dále byly specifikovány funkční a nefunkční požadavky na mobilní aplikace a backend API. Po dohodě s vedoucím práce bylo navrženo uživatelské rozhraní mobilní aplikace. Poté byla navrženo REST API, které bude poskytovat data pro mobilní aplikaci. Následně byla navržena datová struktura backend služby. Dále byla na základě návrhů implementována mobilní aplikace a otestováno API.

Tato práce poskytuje mobilní aplikace s jednoduchým a intuitivní uživatelským rozhraním pro sledování návrhů zákonů a výsledků jejich hlasování v PSP. Zároveň poskytuje informace o hlasováních jednotlivých poslaneckých klubů a jejich členů. Uživatel pomocí aplikace může sledovat aktuální dění na cestách. Aplikace pomáhá uživatelům být informovaní ohledně politického dění v ČR.

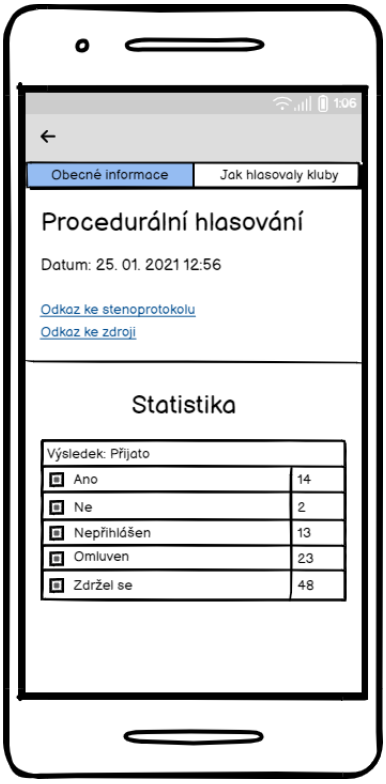
Příloha A

Wireframes rozhraní aplikace

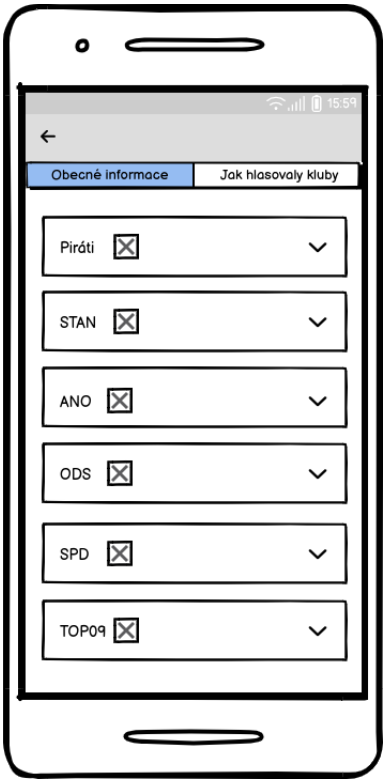


Obrázek A.1 Seznam hlasování

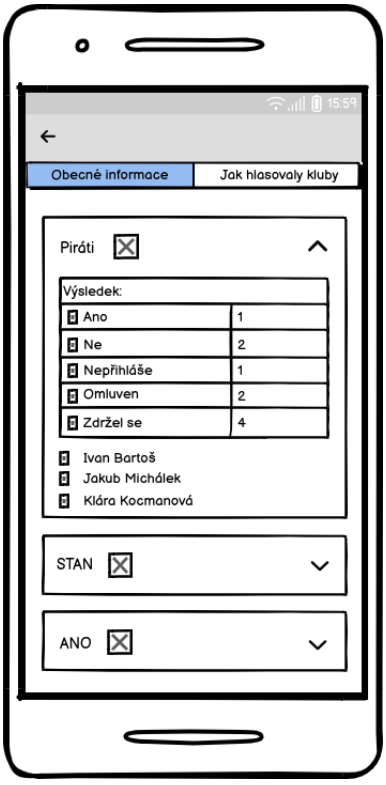
Obrázek A.2 Vyhledávání v seznamu hlasování



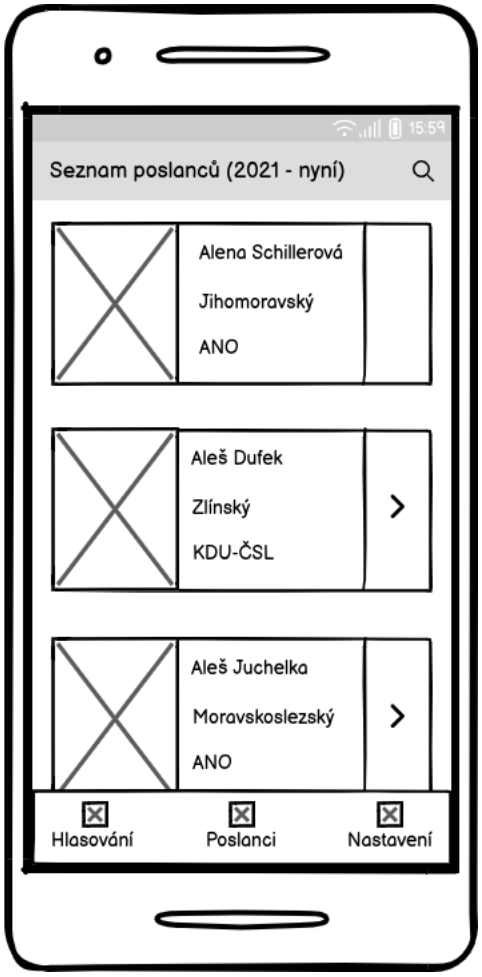
Obrázek A.3 Detail hlasování



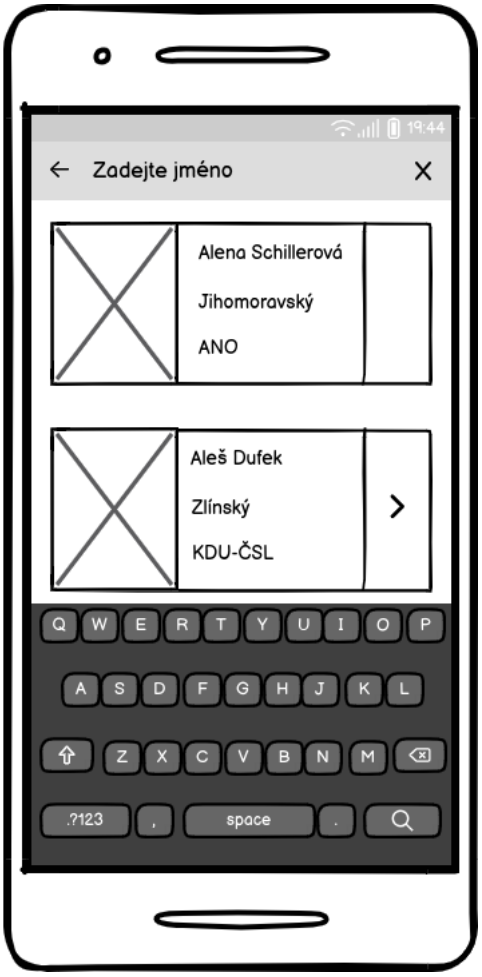
Obrázek A.4 Jak hlasovaly kluby



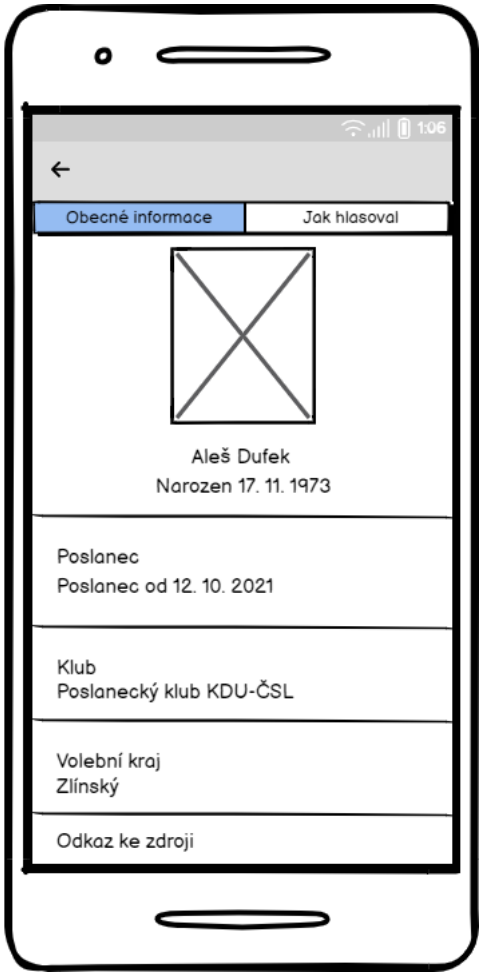
Obrázek A.5 Jak hlasovaly kluby s expandovaným oknem klubu



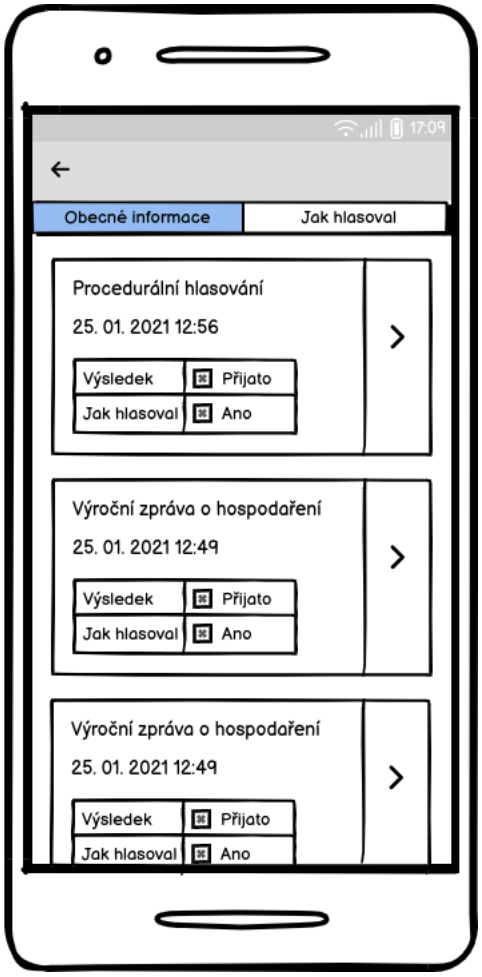
Obrázek A.6 Seznam poslanců



Obrázek A.7 Vyhledávání v seznamu poslanců

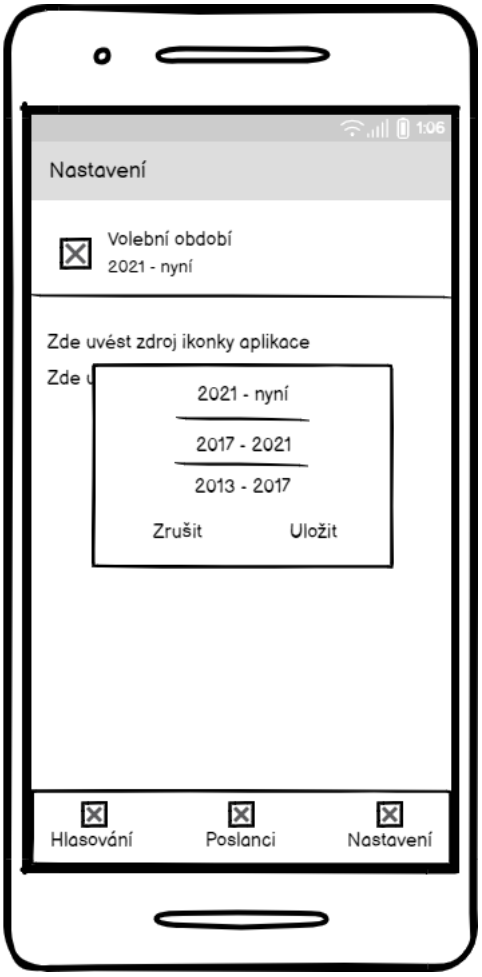


Obrázek A.8 Detail poslance



Obrázek A.9 Jak hlasoval poslanec

Obrázek A.10 Seznam nastavení



Obrázek A.11 Nastavení volebního období

Příloha B

REST API odpovědi

Výpis kódu B.1 Tělo odpovědi pro dotaz `GET /api/app`.

```
{
  "election_years": [
  2021,
  2017,
  2013,
  2010,
  2006,
  2002,
  1998,
  1996,
  1992
  ]
}
```

Výpis kódu B.2 Tělo odpovědi pro dotaz `GET /api/vote`

```
[
{
  "id": 1,
  "date_time": "16. 12. 2022 13:29",
  "description": "Hlasovani 1",
  "result": "A"
},
{
  "id": 2,
  "date_time": "16. 12. 2022 13:26",
  "description": "Hlasovani 2",
  "result": "A"
},
]
```

Výpis kódu B.3 Tělo odpovědi pro dotaz `dGET /api/vote/i`

```
{
  "id": 1,
```

```
    "date_time": "16. 12. 2022 13:29",
    "description": "Hlasovani 1,
    "result": "A",
    "steno_protocol_url": "http://www.psp.cz/eknih/2021ps/
       stenprot/048schuz",
    "source_url": "https://www.psp.cz/sqw/hlasy.sqw?g=77302&l=cz"
       ,
    "yes_count": 100,
    "no_count": 0,
    "logged_off_count": 64,
    "excused_count": 0,
    "refrained_count": 36,
    "election_year": 0
  }
```

Výpis kódu B.4 Tělo odpovědi pro dotaz `GET /api/party/vote/1`

```
[
{
  "party_name": "Nazev klubu",
  "logo_url": "https://www.psp.cz/pics/klub/l-cps.jpg",
  "vote_id": 1,
  "party_results": {
    "yes_count": 2,
    "no_count": 0,
    "logged_off_count": 1,
    "excused_count": 0,
    "refrained_count": 0
  },
  "member_results": [
  {
    "member_name": "Poslanec 1",
    "vote_result": "@"
  },
  {
    "member_name": "Poslanec 2",
    "vote_result": "C"
  },
  {
    "member_name": "Poslanec 3",
    "vote_result": "A"
  },
  {
    "member_name": "Poslanec 4",
    "vote_result": "A"
  }
  ]
}
]
```

Výpis kódu B.5 Tělo odpovědi pro dotaz `GET /api/member`

```
[
{
  "id": 1,
  "name": "Poslanec 1",
  "party": "ANO",
  "photo_url": "https://www.psp.cz/eknih/cdrom/2021ps/eknih/2021ps/poslanci/i6474.jpg",
  "election_region": "Volebni kraj 1",
  "election_year": 2021
},
{
  "id": 2,
  "name": "Poslanec 2",
  "party": "ODS",
  "photo_url": "https://www.psp.cz/eknih/cdrom/2021ps/eknih/2021ps/poslanci/i6804.jpg",
  "election_region": "Volebni kraj 2",
  "election_year": 2021
},
]
```

Výpis kódu B.6 Tělo odpovědi pro dotaz `GET /api/member/1`

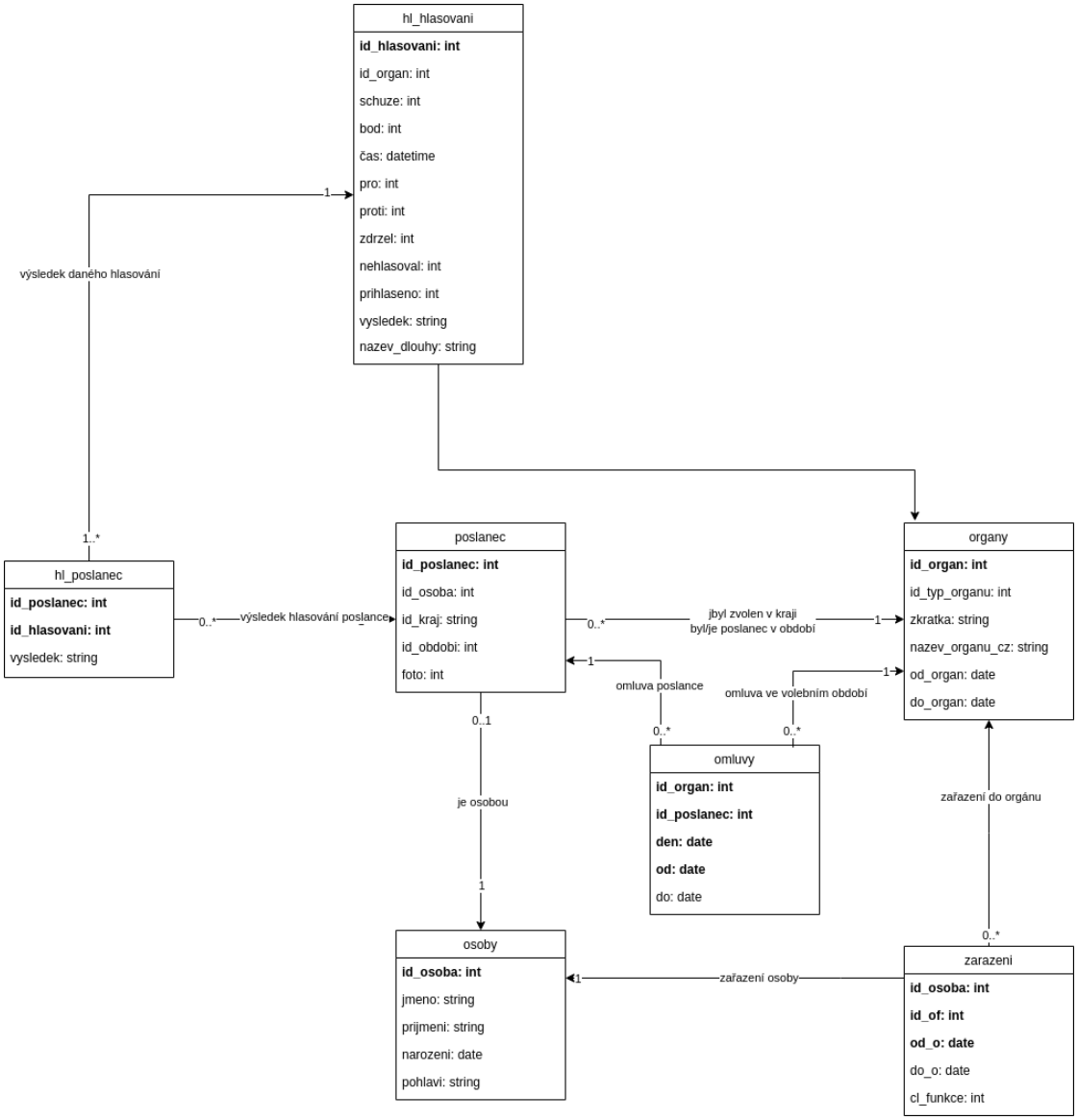
```
{
  "id": 1,
  "name": "Poslanec 1",
  "gender": "M",
  "party": "Poslanecky klub",
  "member_from": "12. 10. 2021",
  "member_to": null,
  "date_of_birth": "25. 09. 1970",
  "election_region": "Volebni kraj 1",
  "photo_url": "https://www.psp.cz/eknih/cdrom/2021ps/eknih/2021ps/poslanci/i6474.jpg",
  "source_url": "https://www.psp.cz/sqw/detail.sqw?id=5700",
  "election_year": 2021
}
```

Výpis kódu B.7 Tělo odpovědi pro dotaz `GET /api/member/1/vote`

```
[
{
  "vote": {
    "id": 1,
    "date_time": "16. 12. 2022 13:29",
    "description": "Hlasovani 1",
    "result": "A"
  },
  "how_member_voted": "@"
},
{
  "vote": {
    "id": 2,
    "date_time": "16. 12. 2022 13:26",
    "description": "Hlasovani 2",
    "result": "A"
  },
  "how_member_voted": "@"
}
]
```

Příloha C

Databázový model



Obrázek C.1 Diagram zdrojových dat PSP

Tabulka C.1 Struktura agency

Název	Typ	Popis
`id`	`int`	identifikátor orgánu
`abbreviation`	`varchar(255)`	zkratka názvu orgánu
`end_date`	`date(255)`	datum zániku orgánu
`name`	`varchar(255)`	název orgánu
`start_date`	`date`	datum založení orgánu
`type_id`	`int`	identifikátor typu orgánu

Tabulka C.2 Struktura excuse

Název	Typ	Popis
`member_id`	`int`	identifikátor poslance, který je omluven
`date`	`date`	datum, kdy je poslanec omluven
`start_time`	`time`	čas, od kterého byl poslanec omluven
`end_time`	`time`	čas, do kterého byl poslanec omluven
`election_year`	`int`	volební rok

Tabulka C.3 Struktura member

Název	Typ	Popis
`id`	`int`	identifikátor poslance, který je omluven
`date_of_birth`	`date`	datum narození
`election_region`	`varchar(255)`	volební kraj
`election_year`	`int`	volební rok
`gender`	`varchar(255)`	pohlaví
`member_from`	`date`	datum začátku členství
`member_to`	`date`	datum konce členství
`name`	`varchar(255)`	jméno
`person_id`	`int`	identifikátor osoby
`photo_url`	`varchar(255)`	URL profilové fotky
`source_url`	`varchar(255)`	URL oficiálního zdroje
`party_id`	`int`	identifikíátor poslaneckého klubu, jehož je členem

Tabulka C.4 Struktura member_vote

Název	Typ	Popis
`result`	`varchar(255)`	jak hlasoval poslanec
`member_id`	`int`	jak identifikátor poslance
`vote_id`	`int`	identifikátor hlasování

Tabulka C.5 Struktura membership

Název	Typ	Popis
`end_date`	`datetime`	datum a čas konce zařazení
`agency_id`	`int`	identifikátor orgánu
`person_id`	`int`	identifikátor osoby
`start_date`	`datetime`	datum a čas začátku zařazení

Tabulka C.6 Struktura party

Název	Typ	Popis
`abbreviation`	`varchar(255)`	zkratka pro název klubu
`name`	`varchar(255)`	název klubu
`election_year`	`int`	volební rok
`party_id`	`int`	identifikátor klubu

Tabulka C.7 Struktura vote

Název	Typ	Popis
`date_time`	`datetime`	datum a čas hlasování
`description`	`varchar(255)`	popis hlasování
`election_year`	`int`	volební rok
`excused_count`	`int`	počet omluvených
`logged_off_count`	`int`	počet nepřihlášených
`meeting_number`	`int`	bod hlasování
`no_count`	`int`	počet hlasování proti
`refrained_count`	`int`	počet zdržených
`result`	`varchar(255)`	výsledek hlasování
`steno_protocol_url`	`varchar(255)`	stenoprotokol
`yes_count`	`int`	počet hlasování pro
`number`	`int`	číslo hlasování
`id`	`int`	identifikátor hlasování

Bibliografie

1. PARLAMENT ČESKÉ REPUBLIKY. *Poslanecká sněmovna Parlamentu České republiky* [online]. [B.r.] [cit. 2023-02-11]. Dostupné z: `https://www.psp.cz/sqw/hp.sqw?akk=7`.
2. HUŠEK, Petr; SMOLÍK, Josef. *POLITICKÝ SYSTÉM A POLITICKÉ STRANY ČESKÉ REPUBLIKY*. Zemědělská 1, 613 00 Brno: Mendelova univerzita v Brně, 2019. ISBN 978-80-7509-665-4.
3. HUŠEK, Petr; SMOLÍK, Josef. *POLITICKÝ SYSTÉM A POLITICKÉ STRANY ČESKÉ REPUBLIKY*. Zemědělská 1, 613 00 Brno: Mendelova univerzita v Brně, 2019. ISBN 978-80-7509-665-4.
4. PARLAMENT ČESKÉ REPUBLIKY. *Přijímání zákonů* [online]. [B.r.] [cit. 2023-02-11]. Dostupné z: `https://www.psp.cz/sqw/hp.sqw?k=173.`.
5. PARLAMENT ČESKÉ REPUBLIKY. *Data Poslanecké sněmovny a Senátu* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: `https://www.psp.cz/sqw/hp.sqw?k=1300`.
6. RED HAT, INC. *What is a REST API?* [Online]. [B.r.] [cit. 2023-02-11]. Dostupné z: `https://www.redhat.com/en/topics/api/what-is-a-rest-api`.
7. CROCKFORD, Douglas. *Introducing JSON* [online]. [B.r.] [cit. 2023-02-11]. Dostupné z: `https://www.json.org/json-en.html`.
8. ANDROID POLITISCOPE DEVELOPER. *politiscope* [online]. [B.r.] [cit. 2023-02-11]. Dostupné z: `https://play.google.com/store/apps/details?id=com.junkie.android.politiscope&gl=US`.
9. PRO PUBLICA INC. [online]. [B.r.] [cit. 2023-01-26]. Dostupné z: `https://www.propublica.org/`.
10. @UNITEDSTATES [online]. [B.r.] [cit. 2023-01-26]. Dostupné z: `https://theunitedstates.io/`.
11. MILL, Eric. *Congress* [online]. Jan 2019 [cit. 2023-01-27]. Dostupné z: `https://play.google.com/store/apps/details?id=com.sunlightlabs.android.congress&hl=en&gl=US`.
12. FEDERAL ELECTION COMMISSION [online]. [B.r.] [cit. 2023-01-26]. Dostupné z: `https://api.open.fec.gov/developers/`.
13. JGRAPH LTD. *https://app.diagrams.net/* [online]. [B.r.] [cit. 2023-02-13]. Dostupné z: `https://app.diagrams.net/`.
14. GOOGLE. *Guide to app architecture* [online]. [B.r.] [cit. 2023-02-09]. Dostupné z: `https://developer.android.com/topic/architecture`.

15. INDEED EDITORIAL TEAM. *What Are the 5 Primary Layers in Software Architecture?* [Online]. [B.r.] [cit. 2023-02-09]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-are-the-layers-in-software-architecture.
16. GOOGLE. *Android Studio* [online]. [B.r.] [cit. 2023-02-09]. Dostupné z: https://developer.android.com/studio.
17. JETBRAINS. *Kotlin* [online]. [B.r.] [cit. 2023-02-09]. Dostupné z: https://kotlinlang.org/.
18. LARDINOIS, Frederic. *Kotlin is now Google's preferred language for Android app development* [online]. [B.r.] [cit. 2023-02-09]. Dostupné z: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/.
19. ORACLE. *Java* [online]. [B.r.] [cit. 2023-02-09]. Dostupné z: https://www.oracle.com/java/.
20. JETBRAINS. *Object expressions and declarations* [online]. [B.r.] [cit. 2023-02-09]. Dostupné z: https://kotlinlang.org/docs/object-declarations.html.
21. JETBRAINS. *Asynchronous Flow* [online]. [B.r.] [cit. 2023-02-09]. Dostupné z: https://kotlinlang.org/docs/flow.html.
22. GOOGLE. *Application* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://developer.android.com/guide/components/fundamentals.
23. GOOGLE. *Dependency injection with Hilt* [online]. [B.r.] [cit. 2023-02-14]. Dostupné z: https://developer.android.com/training/dependency-injection/hilt-android.
24. GOOGLE. *Build better apps faster with Jetpack Compose* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://developer.android.com/jetpack/compose.
25. GOOGLE. *Create dynamic lists with RecyclerView* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://developer.android.com/develop/ui/views/layout/recyclerview.
26. GOOGLE. *Lists and grids* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://developer.android.com/jetpack/compose/lists.
27. GOOGLE. *Save key-value data* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://developer.android.com/training/data-storage/shared-preferences.
28. GOOGLE. *Working with Proto DataStore* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://developer.android.com/topic/libraries/architecture/datastore.
29. GOOGLE. *Protocol Buffers* [online]. [B.r.] [cit. 2023-02-14]. Dostupné z: https://protobuf.dev/.
30. PARONELLA, Bernat Borrás. *alorma/Compose-Settings* [online]. [B.r.] [cit. 2023-02-14]. Dostupné z: https://github.com/alorma/Compose-Settings.
31. GOOGLE. *Paging library overview* [online]. [B.r.] [cit. 2023-02-14]. Dostupné z: https://developer.android.com/topic/libraries/architecture/paging/v3-overview.
32. JETBRAINS. *Create dynamic lists with RecyclerView* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://www.jetbrains.com/idea/.
33. MICROSOFT. *Azure* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://azure.microsoft.com/en-us.
34. VMWARE, INC. *Web applications* [online]. [B.r.] [cit. 2023-02-14]. Dostupné z: https://spring.io/web-applications.
35. JETBRAINS. *Ktor* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://ktor.io/.
36. VMWARE, Inc. *Introducing Kotlin support in Spring Framework 5.0* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0.

37. RED HAT. *Hibernate* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://hibernate.org/.

38. THE PROJECT LOMBOK AUTHORS. *Project Lombok* [online]. [B.r.] [cit. 2023-02-14]. Dostupné z: https://projectlombok.org/.

39. VMWARE, Inc. *Spring Data JPA - Reference Documentation* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.

40. APACHE, Commons. *Apache Commons IO* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://commons.apache.org/proper/commons-io/.

41. LINGALA, Srikanth Reddy. *zip4j* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://github.com/srikanth-lingala/zip4j.

42. SMITH, Glen. *opencsv* [online]. [B.r.] [cit. 2023-02-10]. Dostupné z: https://opencsv.sourceforge.net/.

43. GOOGLE. *Testing your Compose layout* [online]. [B.r.] [cit. 2023-02-13]. Dostupné z: https://developer.android.com/jetpack/compose/testing.

44. VMWARE TANZU. *Testing the Web Layer* [online]. [B.r.] [cit. 2023-02-13]. Dostupné z: https://spring.io/guides/gs/testing-web/.

45. PHPFASHION. *Jak mockovat final třídy?* [Online]. [B.r.] [cit. 2023-02-13]. Dostupné z: https://phpfashion.com/jak-mockovat-final-tridy#:~:text=Mockov%C3%A1n%C3%AD%20znamen%C3%A1%20nahrazen%C3%AD%20p%C5%AFvodn%C3%ADho%20objektu,chov%C3%A1n%C3%AD%2C%20kter%C3%A9%20pot%C5%99ebujeme%20kv%C5%AFli%20testov%C3%A1n%C3%AD..

46. MOCKITO. *Tasty mocking framework for unit tests in Java* [online]. [B.r.] [cit. 2023-02-14]. Dostupné z: https://site.mockito.org/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu CD
- `psp.apk` ..... instalační balíček aplikace
- `src`
  - `impl` ..... zdrojové kódy mobilní aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF