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
9. 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 principů 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 předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, 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 způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 9. 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 kapitole budou uvedeny cíle této práce.

Poslanecká sněmovna

Tato kapitola slouží jako úvod do tematiky hlasování v poslanecké sněmovně.

Funkční a nefunkční požadavky

V další kapitole bude popsány funkční a nefunkční požadavky pro mobilní aplikaci i 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 v rámci této práce využita.

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. Dále budou popsány návrhy pro architektury použité pro implementaci mobilní aplikace a backendu.

Implementace

Na základě návrhů budou popsány implementace mobilní aplikace a backendu. Popis bude zahrnovat popis využitých nástrojů a technologií, případně jejich porovnání s možnými alternativami. Dále bude zahrnovat popis implementace po jednotlivých vrstvách architektury mobilní aplikace a backendu.

Testování

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

Nasazení

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

Spuštění

Tato kapitola slouží jako návod pro spuštění lokální zprovoznění backendu a mobilní aplikace.

Shrnutí

V této kapitole bude shrnuta práce, včetně jejího zadání a jejich splnění.

Seznam zkratek

PSP	Poslanecká sněmovná Parlamentu ČR
REST	Representational state transfer
API	Application Programming Interface

Kapitola 1

Cíle

Prvním cílem práce je specifikace funkčních a nefunkčních požadavků, které jsou kladeny na mobilní aplikaci a backendu. Druhým cílem je analýza existujících řešení v zahraničí. Dalším cílem je návrh, implementace, otestování a nasazení mobilní aplikace pro zobrazení výsledků hlasování poslanecké sněmovny pro operační systém Android. Následujícím cílem je návrh, implementace, otestování a nasazení backendu, který bude pravidelně stahovat zdrojová data a transformovat je pro vhodné použití mobilní aplikací. Posledním cílem je shrnutí práce a diskuze ohledně splnění požadavků.

Kapitola 2
Poslanecká sněmovna

Táto kapitola slouží jako úvod do tematiky hlasování v poslanecké sněmovně. V první podkapitole bude popsána poslanecká sněmovna jako taková a jakou má roli v politickém sytému ČR. Dále bude stručně popsán hlasovací proces v poslanecké sněmovně. Následně bude popsán webový portál PSP. Na konci bude motivace pro tuto práci.

2.1 Poslanecká sněmovna

Základní prvky politického systému ČR představuje prezident, vláda, Parlament a ústavní soud. Ústava ČR dělí moc na zákonodárnou – Parlament, který je složen z Poslanecké sněmovny a Senátu, výkonnou – prezident, vláda a státní zastupitelství a soudní – Ústavní soud a obecné soudy. [1]

Parlament České republiky se skládá ze dvou komor – Poslanecké sněmovny (dolní komora) a Senátu (horní komora). Poslanecká sněmovna se skládá z 200 poslanců a je volena na čtyři roky na základě poměrného volebního systému [1].

2.2 Hlasování v poslanecké sněmovně

Komora PS 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 [1].

Proces návrhu a schvalování zákona je komplexní a řídí se podle určitých pravidel. Avšak pro účely této práce stačí pochopit schvalovací proces v PS. Kompletní proces přijímání zákonů lze najít na webu PSP.

2.3 Webový portál psp.cz

Hlavním zdrojem data z PSP je oficiální webový portál psp.cz. Tento portál poskytuje je webová aplikace, ale poskytuje i strojově zpracovatelná data, která budou využita pro tuto práci.

2.4 Motivace pro tuto práci

Web PSP obsahuje veškeré informace ohledně hlasováních, nicméně není responzivní a přizpůsobený pro mobilní zařízení, a tudíž pro uživatele mobilních zařízení je web nepoužitelný. Zároveň srovnatelná aplikace na českém trhu práce ještě neexistuje, a tudíž by se uživatelovi v ČR taková

aplikaci mohla hodit. V neposlední řadě tato práce podporuje to, abychom měli informované voliče, zajímající se o to, jak jimi volení zástupci hlasují.

Kapitola 3

Funkční a nefunkční požadavky

V této kapitole 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. Jedne pro identifikátor daného požadavku pro jednodušší identifikaci v rámci práce, a jeden pro popis daného požadavku.

3.1 Funkční požadavky

Tato podkapitola obsahuje seznam funkčních požadavků pro mobilní aplikaci (3.1) a backend (3.2).

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 a oficiální zdroj, a celkovou statistiku hlasování. Celkovou statistikou hlasování rozumíme počet hlasování pro a proti, a počet nepřihlášených, omluvených a zdržených. Dále bude obsahovat hlasování jednotlivých poslaneckých klubů a jejich členů pro daný návrh.
FP_03	Aplikace bude umět zobrazit seznam členů poslanecké sněmovny. V seznamu budou tyto informace o členech: 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 nabytí statusu poslance, poslanecký klub a volební kraj. Dále bude obsahovat informace o tom, jak daný poslanec hlasovalv jednotlivých návrzích zákona.
FP_05	Aplikace bude poskytovat možnost nastavení staršího volebního období. Uživatel vždy uvidí návrhy zákonů a seznam poslanců z aktuálně nastavenéího volebního období.
FP_06	Aplikace bude poskytovat možnost vyhledávání hlasování podle jeho popisu.
FP_07	Aplikace bude poskytovat možnost vyhledávání poslance podle jeho jména.
FP_07	Aplikace bude pro seznamy návrhů a poslanců umožňovat tyto seznam nějakým způsobem aktualizovat skrz uživatelské rozhraní.

Tabulka 3.1 Funkční požadavky pro mobilní aplikaci.

Funkční požadavky pro back-end	
ID požadavku	**Popis požadavku**
FP_03	Backend bude poskytovat endpoint pro seznam všech volebních období.
FP_01	Backend bude poskytovat endpoint pro seznam návrhů a výsledků hlasování.
FP_02	Backend bude poskytovat endpoint pro detail návrhu a výsledku hlasování.
FP_03	Backend bude poskytovat endpoint pro výsledky hlasování poslaneckých klubů a jejich členů pro daný návrh.
FP_04	Backend bude poskytovat endpoint pro seznam poslanců.
FP_05	Backend bude poskytovat endpoint pro detail poslance.
FP_06	Backend bude poskytovat endpoint pro informace o tom, jak hlasoval konkrétní poslanec v jednotlivých návrzích zákona.

Tabulka 3.2 Funkční požadavky pro back-end.

3.2 Nefunkční požadavky

Tato podkapitola obsahuje seznam nefunkčních požadavky pro mobilní aplikaci (3.3) a backend (3.4).

Nefunkční požadavky pro mobilní aplikaci	
ID požadavku	**Popis požadavku**
NP_01	Aplikace nebude provádět výpočetně náročná zpracování dat pro šetření aplikace. Výpočetně náročna pracování 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 3.3 Nefunkční požadavky pro mobilní aplikaci.

Nefunkční požadavky pro back-end	
ID požadavku	**Popis požadavku**
NP_01	Backend bude data stahovat z oficiální portálu PSP.
NP_02	Backend bude data stažená z portálu PSP transformovat do databázového modelu (7.3). 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.
NP_03	Backend bude každý den stahovat nová data z portálu PSP a aktualizovat databázi.
NP_05	Backend bude data vystavovat ve formatu JSON.
NP_06	Část dat, jejichž zpracování by trvalo příliš dlouho (např. půl dne) lze zpracovat až za běhu.

Tabulka 3.4 Nefunkční požadavky pro back-end.

Kapitola 4

Analýza existujících řešení

Tato kapitola se zabývá rešerší existujících řešení.

4.1 politiscope

Autor: Android Politiscope Developer

Počet stažení: více než 10 000

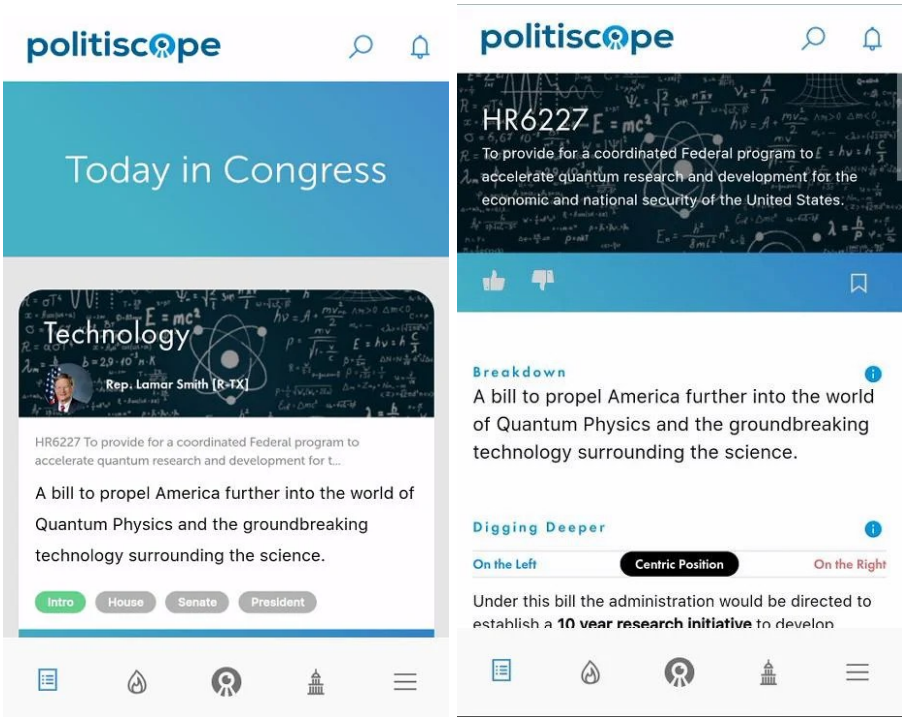
Analyzovaná verze: 2.4 (26. 1., 2023)

Aplikace politiscope [2] dle popisu na Google Play poskytuje informace ohledně politiky 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 návrh zákona a sledovat průběh 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 [3] - ProPublica je nezávislá, nezisková redakce.
- theunitedstates[4] - @unitedstates je projekt poskytující data ohledně Spojených Států veřejností a pro veřejnost.
- congress [5] - Congress.gov je oficiální portál pro informace z Kongresu a orgánů státní správy.
- openfec [6] - 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. Při registraci mě to však automaticky přesměruje na přihlášovací obrazovku. Při zadání přihlašovacích údaju to pak píše, že účet se zadanými přihlašovacími údaji neexistují. Aplikaci jsem testoval na dvou různých zařízeních a na obou je stejný problém. Aplikace má přesto přes 10 000 stažení, a tudíž ve většině případech funguje. Tipuji, že problém souvisí nějakým způsobem s geografickou lokací mobilního zařízení.



Obrázek 4.1 Android aplikace politiscope

4.1.1 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 API. Dat PSP jsou totiž poskytována ve formě CSV souborů, jak bude popsáno v 5.

4.2 Congress

Autor: Eric Mill

Počet stažení: více než 500 000

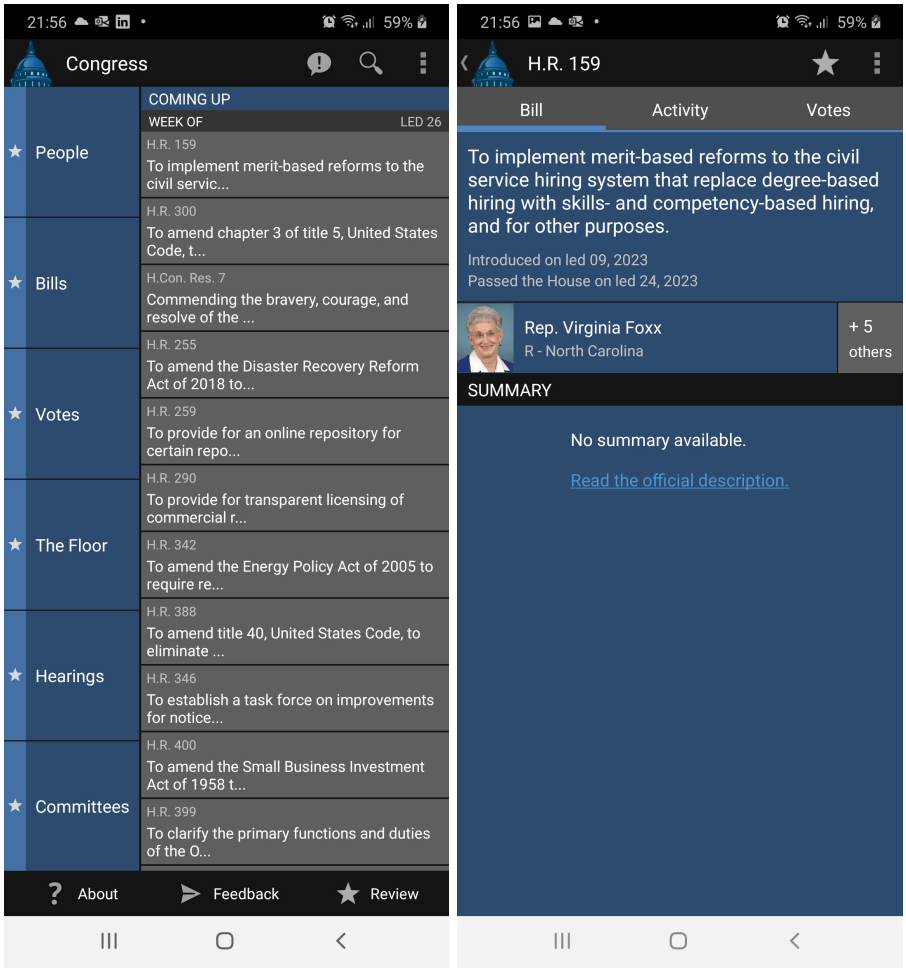
Analyzovaná verze: 4.9.2 (27. 1., 2023)

Aplikace Congress [5] dle popisu na Google Play poskytuje informace ohledně politických reprezentantů a jejich hlasováních, a návrzích zákona ve Spojených Státech. Návrhy a hlasování lze vyhledávat podle klíčových slov.

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

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

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



Obrázek 4.2 Android aplikace politiscope

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

Obrazovka pro události v kongresu obsahuje seznam událostí seřazené sestupně podle času. 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.

4.2.1 Zhodnocení

První dojem z této aplikace je to, že je velmi propracovaná z hlediska různorodosti poskytovaných informací. Přesto 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 lze snadno vidět výsledek hlasování, jak hlasoval který politik, proces schvalování návrhu a aktuální stav. Politiky lze snadno vyhledat podle klíčových slov, státu a příslušnosti ve Sněmovně nebo Senátu. Politiky a návrhy zákonu lze sledovat a nastavit

notifikaci o jejich změnách.

Kapitola 5

Analýza zdrojových dat

V této kapitole budou analyzována zdrojová data, z kterých bude čerpat backend.

5.1 Zdrojové soubory

Zdrojová data se nachází v souborech ve formátu zip, které jsou volně ke stažení na stránkách PSP. Pro tuto práci budou důležité následující soubory, které obsahují datové soubory potřebné pro splnění zadání a funkčních a nefunkčních požadavků:

- **poslanci.zip** - Eviduje osoby, jejich zařazení do orgánů a jejich funkce v orgánech a orgány jako takové. Obsahuje datové soubory potřebné
- **hl-XXXXps.zip** - Eviduje hlasování v PS.

Zbytek práce se bude zabývat daty v datových souborech v těchto dvou zdrojových souborech.

5.2 Formát dat

Data v datových souborech jsou poskytována ve formátu UNL, 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.

5.3 Aktualizace

Web uvádí, že data obsahují úplný stav a že rozdílové aktualizace nejsou poskytovány. To je pro použití neideální, jelikož při aktualizaci dat je potřeba stáhnout všechna data a následně buď zpracovat všechna data znovu nebo naimplementovat vlastní logiku pro zjištění rozdílového stavu a aktualizovat data pouze na základě tohoto rozdílového stavu.

5.4 Datové typy

Na stránce jsou poskytovány informace o datových typech sloupců. Zde je výčet pouze těch datových typů, které jsou nutné pro pochopení dat používaných 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 5.1 Datové typy dat

5.5 Licence

Data jsou poskytována bezplatně, využití dat je podmíněno uvedením zdroje dat a případně datem zpracování dat. V mobilní aplikaci a v repozitářích mobilní aplikace a backendu bude uveden zdroj dat.

5.6 Datové soubory

Zde je výčet použitých datových souborů a stručná analýza. Detailnější analýza bude v následující sekci. Ze zdrojového souboru `poslanci.zip` budou používány následující datové soubory ([7]:)

- **organy** - Reprezentuje orgány ve státní správě. Zahrnuje parlamenty všech volebních období a poslanecké kluby. Tabulka bude využita pro nalezení všech poslaneckých klub;.
- **poslanec** - Reprezentuje poslance v různých volebních obdobích.
- **osoby** - Reprezentuje osobu. Obsahuje osobní údaje jako jméno a příjmení a datum narození. Poslanec je osobou, a tudíž lze o poslanci prostřednictvím téhle tabulky zjistit jeho osobní údaje.
- **zarazeni** - Obsahuje zařazení osoby osoby do nějakého orgánu. Tabulka bude využita pro zjištění členů poslaneckého klubu.

Ze zdrojových souborů `hl-XXXX.zip`, kde XXXX je volební rok, budou používány následující datové soubory ([8]:)

- **hl_hlasovani** - Reprezentuje hlasování o návrhu zákona. Zahrnuje většinu informací o hlasování, které budou potřeba.
- **hl_poslanec** - Reprezentuje výsledek hlasování poslance o návrhu zákona, tedy jak hlasoval který poslanec o kterém návrhu zákona.
- **omluvy** - Repreznetuje časově ohraničené omluvy z jednání a hlasování v poslanecké sněmovně. Bude použito pro zjištění počtu omluvených.

5.7 Tabulky

Na (A.1) lze vidět diagram zdrojových dat, s kterými bude dále v práci pracováno. Obsahuje typy dat, jejich atributy, datové typy atributů a vazby mezi typy. Atributy identifikující daný typ jsou zvýrazněny tučným písmem. Následovat bude výčet použitých tabulek s popisem atributů. Pro stručnost budou u tabulek uvedeny pouze atributy, které jsou v rámci práce používány. Poznatky v textu jsou převážně z **analýzy obsahu datových souborů** a nejsou na stránkách PS nijak zdokumentovány.

organy

Tabulka **organy** (5.2) 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 nalézt kontrolou, zda hodnoty atributů **od_organ** a **do_organ** spadají mezi hodnotami stejných atributů PS. PS je totiž také orgánem a lze ji najít prostřednictvím hodnoty atributu **id_organ**. První PS má identifikátor 165. PS v každém následujícím volebním období má hodnotu identifikátoru o jednu větší než PS v předchozím volebním období. Tedy např. PS ve volebním období 2017 až 2021 má identifikátor 172. Dále pokud hodnota atributu **do_organ** je rovna `null`, pak jde o aktuální orgán.

Tabulka 5.2 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** (5.3) bude použita pro získání osobních údajů o poslancích. Tabulka však nezahrnuje pouze poslance, ale i další osoby senátu. Pro zjištění, kterého poslance reprezentuje daná osoba, je potřeba použít tabulku **zarazeni**. Zde je zajímavé datum narození, které je 1. 1. 1900, pokud je neznámo. Zobrazovat toto datum v mobilní aplikaci by nebylo ideální. V kapitole o návrhu REST API bude popsána transformace tohoto data na jinou hodnotu.

Tabulka 5.3 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. Zde je potřeba dát pozor na to, že 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 tu tabulka **funkce** není uvedena. Je však potřeba zjistit příslušnost poslance do poslaneckého klubu, kterou lze zjistit kombinací tabulky **zarazeni** a **organy**. Důsledkem popsaného je to, že nás budou zajímat pouze zařazení s hodnotou atributu **cl_funkce** rovnou 0. Tím pádem identifikátor **id_of** bude vždy referovat k tabulce **organy**. Tabulka **organy** obsahuje poslanecké kluby, což je přesně to, co potřebujeme. Dále pokud hodnota atributu **do_o** je rovna `null`, pak jde o aktuální zařazení.

Tabulka 5.4 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.

Tabulka 5.4 Tabulka zarazeni

Tabulka zarazeni		
Sloupec	**Typ**	**Použití a vazby**
od_o	datetime(year to hour)	Zařazení od
do_o	datetime(year to hour)	Zařazení do

poslanec

Tabulka **poslanec** (5.5) slouží pro získání informací o všech poslancích. Pokud je někdo poslancem ve více volebních obdobích, pak bude v této tabulce mít více záznamů. Tedy dva poslanci mohou reprezentovat tutéž osobu, pokud jsou z různých volebních období. Dále pomocí atributu **id_osoba** lze tabulku zjistit osobní údaje o poslanci. Dále pomocí atributu **id_kraj** lze získat název volebního kraje. A pomocí atributu **id_obdobi** lze zjisti volební období, do kterého přísluší. Hodnota atributu **id_obdobi** totiž odpovídá identifikátoru některé z PS. Např. pokud poslanec patří do PS v prvním volebním období (orgán mající identifikátor 165), pak je hodnota atributu **id_obdobi** rovna 165. Dále někteří poslanci mají profilové foto. Po analýze fotek poslanců na oficiálním portálu PS 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 (pozor, ne identikátor poslance) a `YYY` je identifikátor PS (např. 165). Příklad: pokud bychom chtěli najít URL s fotkou poslance s identifikátorem 1659, zjistíme hodnoty atributů **id_kraj** a **id_obdobi**, a na základě nich vygenerujeme URL fotky.

Tabulka 5.5 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 většinu potřebných informací o hlasováních o daném návrhu zákona. Čislo schůze a číslo hlasování budou použity pro sestavení URL pro stenoprotokol daného hlasování. 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 5.6 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
čas	datetime(hour to minute)	Čas hlasování
pro	int	Počet hlasujících pro
proti	int	Počet hlasujících proti
zdrzel	int	Počet hlasujících zdržel se, tj. stiskl tlačítko X
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** obsahue informace o tom, jak hlasoval který poslanec v rámci kterého hlasování. Výsledky 'B' a 'N' jsou interpretovány stejně, oba znamenají hlasování proti. Výsledek 'Fb se používalo před rokem 1995. Po roce 1995 má vždy hodnotu 0. Výsledek 'W' je velmi ojedinělý a v případě, že nastane, sečte se do počtu nepřihlášených. U výsledku 'K' se mi nepodařilo zjistit, co přesně znamená zdržel se/nehlasoval. Ve zdrojových datech se vysktuje spíš méně, ale častěji než výsledek 'W'. Na webu PS je tento výsledek u jednoho hlasování v roce 2013 interpretován jako nehlasoval. Z toho lze usoudit pouze to, že 'K' může znamenat nehlasoval. Není však z toho jasné, kdy může znamenat zdržel se. Na výsledek hlasování má vliv pouze počet pro a proti, a tudíž bylo rozhodnuto, že tento výsledek bude mobilní aplikaci ignorován.

Tabulka 5.7 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

Tabulka 5.7 Tabulka hl_poslanec

Tabulka hl_poslanec		
Sloupec	**Typ**	**Použití a vazby**
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** (5.8) zaznamenává časové ohraničení omluv poslanců z jednání Poslanecké sněmovny. Slouží pouze po spočtení počtu omluvených poslanců během hlasování. Podáváme-li se na statistiky o hlasování poskytnuté tabulkou **hl_hlasovani**, uvidíme, že poskytuje počet hlasování pro, proti, a počet zdržených, ale již ne počet nepřihlášených a omluvených. Počet omluvených byl přidán až po 1995, kdy pouze nahradil počet poslanců, kteří nehlasovali. Počet nehlasujících poslanců tedy nepomůže k odvození počtu nepřihlášených a omluvených. Jediná další informace v tabulce **hl_hlasovani** je tedy počet přihlášených. Intuitivně by člověk čekal, že doplněk pro počet přihlášených je počet nepřihlášených, ale není tomu tak. Když jsem od počtu poslanců (200) odečetl počet přihlášených, nikdy to nevycházelo s počty nepřihlášených na webu, ať už jsem to zkoušel s jakýmkoliv hlasování. Z toho důvodu je počet omluvených počítáno z tabulky **omluvy**, kde na základě atributů **den**, **od** a **do** zjistíme, zda datum a čas omluvy poslance spadá do data a času daného hlasování.

Podle popisu tabulky na webu data slouží pro nahrazení výsledku typu '@', tj. pokud výsledek hlasování jednotlivého poslance je nepřihlášen. Pokud čas hlasování spadá do časového intervalu omluvy, pak se za výsledek považuje 'M', tj. omluven.

Tabulka 5.8 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 6

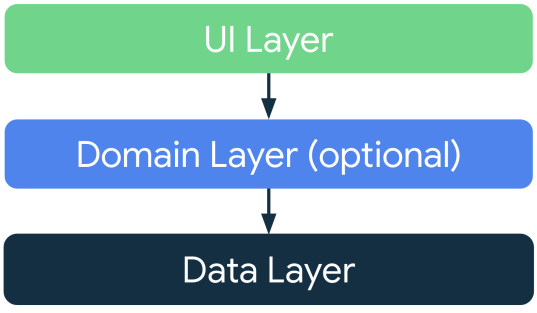
Analýza softwarových architektur

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

6.1 Mobilní aplikace

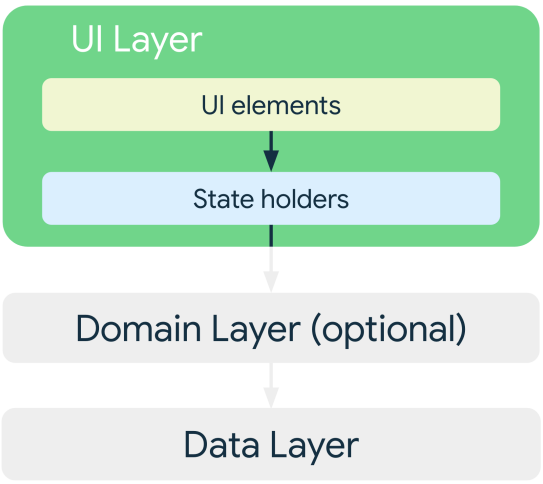
6.1.1 Architektura doporučená Googlem

Softwarovou architekturu doporučenou Googlem [9] ilustruje následující obrázek:



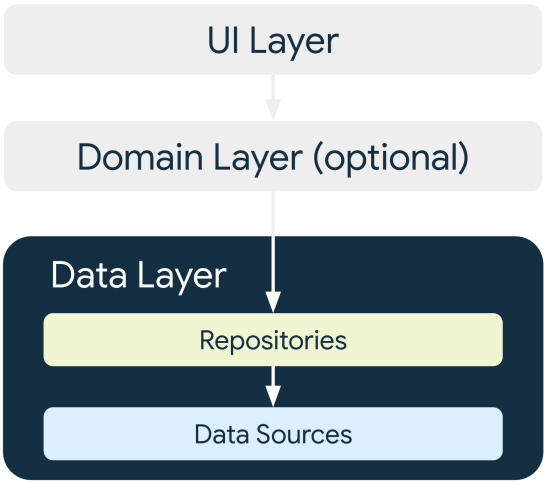
Obrázek 6.1 Architektura pro mobilní aplikaci pro Android [9]

Architektura je rozdělená na UI vrstvu, doménovou vrstvu a datovou vrstvu. UI (také prezentační) vrstva má na starosti zobrazení dat do uživatelského rozhraní. Uživatelské rozhraní je aktualizováno na základě 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 6.2 Složení UI vrstvy [9]

UI elementy reprezentují elementy, které jsou vykreslovány obrazovku. Držitelé stavu mají na starosti správu dat a aplikační logiku. Změna dat v držiteli stavu způsobi změnu UI elementů a znovu vykreslení obrazovky. Datová vrstva, jak ilustruje následující obrázek, je složena z repozitářů a datových zdrojů. Repozitář má na starosti vystavení dat pro zbytek aplikace, abstrahování od konkrétního způsobu získávání dat (např. jestli jsou data získávána z lokální databáze nebo vzdáleně) a centralizaci dat. Centralizace dat slouží pro vytvoření jednotného zdroje pravdy, tj. data jsou zbytkem aplikace vždy čerpána přímo i nepřímo z toho zdroje. Datový zdroj pak reprezentuje konkrétní zdroj dat, z kterého jsou získávána data. Může to být např. lokální soubor, backend nebo lokální databáze.



Obrázek 6.3 Složení datové vrstvy [9]

Doménová vrstva má na starosti zapouzdření složité business logiky nebo business logiky přepoužívané v několika držitelích stavu. Třídy v této vrstvě jsou obvykle pojmenovány jako *use cases* nebo *interactors*.

6.1.2 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.
- **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 repoiztáři). 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.

6.2 Backend

6.2.1 5-vrstvá architektura

5-vrstvá architektura [10] 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 získání dat z databáze. Slouží jako abstraktní rozhraní pro bussinessovou vrstvu pro přístup k datům. Data ukládá a získává komunikací s API databáze poskytované databázovou vrstvou.
- **Databázová** - Má na starosti ukládání dat. Může být implementovaná např. pomocí relační databáze. Poskytuje data perzistentní vrstvě.

6.2.2 Zhodnocení

5-vrstvá architektura je pro účely této práce dostačující. Prezentační vrstvu bude použita pro vystavení 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 a validaci dat. Backend nebude mít téměř žádnou business logiku, bude pouze zpracovávat data z webu PS 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 jedna další vrstva pro pravidelnou aktualizaci databázové vrstvy. Backend bude totiž pravidelně stahovat data z webu PS a aktualizovat databázi. K této architektuře tedy bude navíc přidána **vrstva pro synchronizace databáze**.

Kapitola 7
Návrh

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

7.1 Uživatelské rozhraní

V této podkapitole bude popsán návrh uživatelského rozhraní. Popis návrhu bude rozdělen do sekcí, kde každá sekce bude odpovídat jedné obrazovce na mobilní aplikaci. V každé sekci bude popis návrhu obrazovky a návrh obrazovky pomocí wireframů.

Seznam hlasování

Na obrázku (A.2) 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, tlačítko pro filtrování seznamu a tlačítko pro otevření webu s oficiálním zdrojem. Každé hlasování v seznamu obsahuje popis návrhu zákona, datum a čas hlasování, výsledek hlasování ve formě ikonky a textu, a indikátor pro kliknutí na dané hlasování. Dolní navigace slouží pro navigaci mezi hlavními obrazovkami, tj. mezi obrazovkou pro seznam hlasování, seznam poslanců a nastavení.

Kliknutím na tlačítko pro vyhledávání se zobrazí vyhledávací pole (A.3), které slouží pro vyhledávání seznamu hlasování podle jejich popisu. Do pole uživatel zadává klíčová slova. Pole obsahuje placeholder text, tlačítko pro smazání textu a tlačítko pro schování vyhledávacího pole.

Detail hlasování

Obrazovka pro detail hlasování (A.6) je složena z hlavičky a obsahu. Hlavička obsahuje tlačítko pro navigaci zpět a tlačítko pro otevření webu s oficiálním zdrojem. 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, a tabulku s informacemi ohledně toho, jak se hlasovalo.Typy hlasování (ano, ne, nepřihlášen, omluven, zdržel se) jsou popsány textově a pomocí ikonky.

Druhý tab (A.6) obsahuje seznam poslaneckých klubů v daném volebním období, rozdělených do boxů. V každém boxu je název klubu, jeho logo, pokud je k dispozici, a indikátor pro expandování boxu pro zobrazení informací o tom, jak daný klub a jeho členové hlasovali (A.6). Expandovaný box obsahuje navíc tabulku se statistikou hlasování jako v prvním tabu, ale pro konkrétní poslanecký klub. Pod tabulkou je seznam členů klubu a to, jak pro daný návrh zákona hlasovali. Výsledek hlasování členů je znázorněno ikonkou.

Seznam poslanců

Obrazovka pro seznam poslanců (A.10) vypadá podobně jako obrazovka pro seznam hlasování. Obsahuje hlavičku s titulem, aktuálně nastaveným volebním obdobím, tlačítkem pro vyhledávání a tlačítkem pro otevření webu s oficiálním zdrojem. Obrazovka dále obsahuje seznam poslanců a dolní navigaci. 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 (A.10) lze poslance filtrovat podle jeho jména a příjmení.

Detail poslance

Na obrazovce s detailem poslance (A.12) můžeme 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 a tlačítko pro otevření stránky s oficiálním zdrojem. 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.13) obsahuje seznam návrhů zákona s výsledky jeho hlasování, a k nim dodatečnou informaci o tom, jak o nic hlasoval daný poslanec.

Nastavení

Na obrazovce pro nastavení (A.16) 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 naskočí okno (A.16) se seznamem volebních období. Po zvolení volebního období uživatel může kliknout na tlačítko Uložit, kterým se dané volební období lokálně uloží a nastaví, nebo Zrušit, čímž se zruší aktuální výběr v seznamu. Dále obsahuje tlačítko O aplikaci, které zobrazí okno se stručnými informacemi o dané aplikaci a uvedením zdroje data a ikonku aplikace.

7.2 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 seznam hlasování podle popisu.
- **name** - Jméno pro filtrování seznamu poslanců.
- **electionYear** - Volební rok pro získání dat pro dané volební období.

Endpointy

```
GET /api/app
```

Vrací následující informace o stavu aplikace (A.1). V době psaní práce obsahuje pouze seznam všech volebních období. S dalšími volebními obdobími budou do tohoto seznamu automaticky přidávány. Použije se pro obrazovku nastavení (7.1).

```
GET /api/vote
```

Vrací seznam hlasování s následujícími informace (A.2):

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

Obsah je stránkovaný. Lze specifikovat query parametry: page, size, description a electionYear. Použije se pro obrazovku pro seznam hlasování (7.1).

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

Vrací následující informace o detailu hlasování (A.3):

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

Použije se pro obrazovku pro detail hlasování (7.1).

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

Vrací následující informace o hlasováních poslaneckých klubů v daném hlasování (A.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 7.1.

```
GET /api/member
```

Vrací seznam poslanců s následujícími informacemi:

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

Obsah je stránkovaný. Lze specifikovat query parametry: page, size, name a electionYear. Použije se pro obrazovku pro seznam poslanců (7.1).

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

Vrací následující informace o detailu poslance:

- identifikátor
- jméno a příjmení
- pohlaví (M - muž, jinak ostatní)
- poslanecký klub
- začátek mandátu
- konec mandátu
- datum narození, pokud známo, jinak null
- volební kraj
- URL odkaz na profilovou fotku, pokud existuje, jinak null
- volební rok

Použije se pro obrazovku (7.1).

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

Vrací následující informace o hlasováních daného poslance:

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

Narozdíl od původní reprezentace výsledků hlasování (5.6) bude REST API vystavovat jednodušší verzi. Výsledek 'B' je přičten k 'N'. Výsledek 'W' je přičten k '@'. Výsledkek 'F' je ignorován, protože mobilní aplikace nebude ukazovat počet poslanců, kteří nehlasovali. Jak bylo probíráno v analýze dat, tento údaj je zastaralý. V datech existoval pouze před rokem 1995 a ani před tímto rokem není na webu PS vidět. Výsledek 'K' je taktéž ignorován, jelikož z dat a z dokumentace není jasné, co přesně znamená. Na výsledek hlasování to nebude mít vliv. Použije se pro obrazovku pro detail poslance (7.1).

7.3 Databázový model

Databázový model bude složen z následujících struktur:

- **agency** (A.1) - Obsahuje informace o jedntlivých orgánech. Používá se pro získání volebního kraje poslance. Volební kraj bude potřeba pro strukturu **member** níže. Dále se bude používat pro získání orgánuý PS v konkrétním volebním období. Na základě toho lze získat všechny poslanecké kluby v daném volebním období, což bude využito pro endpoint (7.2). Data pro tuto strukturu lze získat z tabulky **table:organy**.
- **excuse** (A.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 endpoint (7.2) a (7.2). Data pro tuto strukturu lze získat z tabulky (5.8).
- **member** (A.3) - Obsahuje informace o poslancích. To se bude používat pro endpoint (7.2). Data pro tuto strukturu lze získat z tabulek (5.3), (5.5), (5.4) a (5.2).
- **member_vote** (A.4) - Obsahuje informace o tom, kdo jak hlasoval v kterém hlasování. Používá se pro endpointy (7.2) a (7.2). Data pro tuto strukturu lze získat z tabulky (5.7).
- **membership** (A.5) - Obsahuje informace časově ohraničeném zařazení osoby do orgánu. Používá se pro endpoint (7.2). Data pro strukturu lze získat z tabulky (5.4).
- **party** (A.6) - Obsahuje informace o politických klubech. Používá se pro endpoint (7.2) a (7.2).
- **vote** (A.7) - Obsahuje informace o hlasováních. Používá se pro endpointy (7.2), (7.2), (**??**) a (7.2). Data pro tuto strukturu lze získat z tabulek (5.6) a (5.8).

Kapitola 8

Implementace mobilní aplikace

V rámci této podkapitoly bude popsána implementace mobilní aplikace. V první podkapitole budou popsány použité nástroje a technologie. Mobilní aplikace je implementována pomocí Googlem doporučené architektury pro vývoj mobilních aplikací (6.1.1). Následující sekce budou rozděleny tak, aby odpovídaly jednotlivým vrstvám této architektury.

8.1 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 [11]. 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 aplikací. Po instalaci je SDK připravené k použití. Není tedy potřeba SDK manuálně instalovat z internetu a naimportovat do IDE. IDE Dále poskytuje průvodce pro vytvoření projektu, pomocí kterého lze vybrat jednu z existujících šablon pro různé typy projektů. Po zvolení šablony IDE vytvoří projektový adresář se zdrojovými kódy, závislostmi a konfiguračními soubory pro danou šablonu. Po vytvoření projektu je aplikace prázndá, ale je 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 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 z předem definovaného vzdáleného repozitáře.
- **Konfigurace** - Gradle umožňuje konfigurovat pluginy a aplikaci. U Androidu lze nakonfigurovat např. SDK verzi, verzi Kotlinu, aktivaci Jetpack Compose toolkitu (8.2).

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 Androidích 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 [12], který je od roku 2017 preferovaným jazykem pro Android [13]. Původním programovacím jayzkem pro Android byla Java [14]. Kotlin má však oproti němu několik výhod:

- **Je stručný** - Kotlin umožňuje např. vytvořit singleton pomocí klíčového slova `object` [15].
- **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 paměťově nenáročná vlákna. To znamená, že lze vytvořit několik coroutinů, z nichž každý reprezentuje nějaký kus paralelně běžícího výpočtu. Oproti vláknům lze coroutinů spustit mnohem více (na běžném počítači klidne v řádu stovek nebo tisíců). Zde je příklad použití coroutiny:

Výpis kódu 8.1 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 vytvářena zavoláním funkce `launch` na objektu `viewModelScope`. Funkce akceptuje callback funkci, která bude zavolána v rámci vytvořené couroutiny. To znamená, že callback funkce bude běžet paralelně vzhledem k ostatním coroutinám, v tomto případě vzhledem ke coroutině spouštějící druhou operaci. Důležité je, že obě coroutiny běží paralelně, ale na stejném vlákně. Dále objekt `viewModelScope` určuje scope, v kterém běží coroutina. Ten určuje životní cyklus dané coroutiny. Pokud skončí scope a coroutina ještě neboběhla, coroutina automaticky ukončí svůj běh. Jednoduchým příkladem scopu je scope samotné aplikace. Pokud je aplikace spuštěna, pak scope existuje a coroutina může běžet. Pokud je aplikace zavřena, scope přestane existovat a coroutina přestane běžet. Díky tomu lze předejít únikům dat. Když by byla aplikace zavřena a scope by stále existoval, pak by coroutina stále běžela, a až by doběhla, mohla by nějakým způsobem modifikovat data nebo data někám přidávat. při opětovném spuštění aplikace však jsou reference na tato data ztracena.

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

- 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 běžět 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 pokračuje ve svém běhu. Častějším případem použití je např. zavolání síťové operace prostřednictvím knihovny ze třetí strany, která poskytuje suspendovatelné funkce, v rámci kterých je interně implementován suspendovací mechanizmus. Tedy jakmile se provede síťová operace a čeká se na výsledek, knihovna coroutinu sama suspenduje. Uživatel tím pádem nemusí manuálně suspendovat coroutinu po zavolání dlouze běžící operace, ale pouze zavolá funkci knihovny, a ta se postará o zbytek. Tato knihovna bude detailněji popsána v (8.2.4).

Kotlin Flow

Kotlin Flow [16] je datový typ, který reprezentuje asynchronní proud hodnot, který poskytuje 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 a vloží ji do proudu, 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 explicitně (pomocí funkce) zaregistruje a začne ji konzumovat. 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 a transformována, tj. její hodnoty mohou být namapovány na jiné. Různé flow mohou být mezi sebou kombinovány, aby konzument mohl pracovat s hodnotami obou flowů zároveň. Lze nastavit, zda má konzument zareagovat na produkci nových hodnot v obou flowů zároveň nebo jenom jednoho z flowů. Všechny zmíněné operace na flowech lze provést zavoláním funkcí na dané flow a lze je řetězit. Zde je příklad flow, která produkuje seznam hlasování:

Výpis kódu 8.2 Příklad použití flow

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

  // zbytek implementace

  private val _currentElectionYear = MutableStateFlow(2021)
  val currentElectionYear: StateFlow<Int> = _currentElectionYear

  private val _searchText = MutableStateFlow("")
  val searchText: StateFlow<String> = _searchText

  init {
    viewModelScope.launch {
      currentElectionYear.combine(searchText, ::Pair)
      .collectLatest { pair ->
        // zde jsou konzumovany hodnoty pair
        // zde jsou provadeny dalsi operace nad hodnotou pair
      }
    }
  }

  fun onSearchTextChange(newText: String) {
    _searchText.value = newText
  }

  // zbytek implementace
}
```

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 jendoho z komponentu Androidu, view modelu, který bude vysvětlen později. V rámci coroutiny jsou kombinovány dvě flowy, `currentElectionYear` a `searchText`, do jednoho flowu produkující pár hodnot. Toho je dosaženo pomocí funkce `combine` s parametrem `::Pair`, což je reference na konstruktor třídy `Pair`, pomocí kterého se vytváří flow párů. Všimněme si, že datovými typy flowů je `StateFlow`. Existence těchto flowů je závislé na životním cyklu Android komponenty, v kterém běží. V tomto případě běží ve view modelu, který ještě existuje v rámci životního cyklu jiné Android komponenty. Pokud jedna z těchto komponent je po konci svého životního cyklu, všechny `StateFlow` přestanou produkovat hodnoty a jsou odebrány z paměti. Po zkombinování dvou různých flow do jedné je výsledné flow konzumováno pomocí funkce `collectLatest`. V tuto chvíli může začít 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 (např. kliknutí na tlačítko nebo psaní) 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. V tuto chvíli chceme, aby se zavolala callback funkce uvnitř `collectLatest`. Nemusíme ji však zavolat přímo, konzument totiž na změnu hodnoty ve flow zareaguje a spustí daný kód sám. Kotlin Flows mají tedy 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 tohoto proudu.
- Kód je čitelnější.

Android komponenty

Aplikace v Androidu jsou složeny z Android komponentů, které mají na starosti různé odpovědnosti:

- Application - Reprezentuje celou aplikaci. V rámci práce je používaná pouze pro nakonfigurování DI pomocí Hiltu, které bude popsáno v následující sekci.
- Activity - Reprezentuje obrazovku. Je to vstupní bod pro vytvoření UI a pro počáteční inicializace.
- ViewModel - Reprezentuje komponentu, která definuje business logiku a drží si stav UI elementů. Změna stavu komponenty ViewModel způsobí i změnu UI. ViewModel si pamatuje data i po konfiguračních změnách (např. otočení mobilu).

Hilt

Hilt je DI knihovna pro Android. Poskytuje kontejnery pro každou Android komponenty. Tyto kontejnery spravují objekty a jejich závislosti a jsou použity pro injektování závislostí. Příkladem injektování kódu lze vidět na (8.3). Do třídy `GetAppStateUseCase` injektujeme pomocí anotace `@Inject` implementataci rozhraní `AppStateRepository`.

Výpis kódu 8.3 Příklad použití DI pomocí knihovny Hilt

```
class GetAppStateUseCase @Inject constructor(
  val appStateRepository: AppStateRepository,
) {...}

interface AppStateRepository {...}
```

Pro konfiguraci závislostí, na základě kterého Hilt sestaví kontejnery, jsou používány tzv. moduly - objekty pro konfiguraci závislostí. V konfigurace (8.4) říkáme, že do proměnných typu `AppStateRepository` budou injektován objekt `AppStateRepositoryImpl`. Objekt je přiom instancován jako singleton. Objekt zároveň potřebuje závislost na `PspRemoteDataSource`, který musí být opět někde nakonfigurován, aby tu mohl být injektován. Buď v tom samém modulu nebo v jiném, pokud chceme moduly pro čitelnost nějakým způsobem rozdělit.

Výpis kódu 8.4 Příklad konfigurace závislostí pro Hilt

```
@Module
@InstallIn(SingletonComponent::class)
object DataModule {

  @Provides
  fun provideAppStateRepository(
  pspRemoteDataSource: PspRemoteDataSource
```

```
  ): AppStateRepository {
    return AppStateRepositoryImpl(pspRemoteDataSource)
  }

}
```

8.2 Implementace uživatelského rozhraní

V této sekci bude popsána implementace uživatelského rozhraní.

Jetpack Compose

Jetpack Compose je Androidem doporučený způsob pro implementaci uživatelského rozhraní. Rozhraní je implementováno voláním tzv *composable* funkcí, což jsou funkce anotované anotací @composable a reprezentují nějaký UI element na obrazovce. Jetpack Compose nám již poskytuje nativní UI elementy jako text a tlačítko nebo kontejnery pro seskupení element do řádku či sloupců. Tyto nativní UI elementy lze kombinovat, a tím vytvořit složitější komponenty. Tyto složitější komponenty lze pak zapouzdřit do vlastně definované composable funkce. Důsledkem je to, že celé UI popisováno pomocí composable funkcí, které volají další composable funkce. Díky tomu je kód pro vytvoření UI modulární.

Jelikož composable funkce jsou vlastně jenom funkce napsané v programovacím jazyce, lze jim předávat parametry, které určují vzhled či chování daného UI elementu. Které parametry composable funkce přijímá a co určuji závisí na implementaci dané funkce. Jsou parametry, které mění např. barvu a velikost daného elementu. Dále jsou parametry, kterým předáváme funkci. Tato funkce může být volána např. když je na daný element kliknuto. Také jsou parametry, které určují textový obsah nějakého textového elementu. Výhodou použití knihovny Jetpack Compose je možnost definovat UI pomocí programovacího jazyka. To znamená, že lze využít cykly (např. pro vytvoření seznamu elementů) nebo podmínky (např. pro podmíněné vykreslování). Implementace rozhraní je díky tomu jednoduchá a intuitivní.

Alternativou k Jetpack Composu jsou XML layouty, které k popisu UI používají externí XML soubor, kde je rozhraní popisováno pomoxí XML tagů a jejich atributů. Hlavní myšlenkou tohoto přístupu je to, že oddělujeme popis uživatelského rozhraní od programového kódu, díky čemuž je kód čitelnější a udržitelnější. 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. Pokud implementujeme dynamický seznam, situace pro XML layouty je ještě horší, neboť implementace vyžaduje hodně boilerplate kódu. Pro Jetpack Composu je vytvoření takového seznamu otázkou pár řádků kódu.

Z výše uvedených důvodů jsem se rozhodl uživatelské rozhraní implementovat pomocí Jetack Compose.

zdroj: https://developer.android.com/develop/ui/views/layout/recyclerview zdroj: https://developer.android.com
zdroj: https://developer.android.com/jetpack/compose

Composable funkce

Základním stavebním kamenem pro tvorbu uživatelského rozhraní pomocí knihovny Jetpack Compose jsou composable funkce. V (8.5) lze vidět použití nativní composable funkce `Text`, která slouží pro vytvoření textového UI elementu. Vidíme, že funkce přijímá parametr `text`, který určuje textový obsah.

Výpis kódu 8.5 Příklad použití composable funkce Text.

```
Text(text = "Vysledky hlasovani")
```

Composable funkce lze kombinovat pro vytvoření složitějších composable funkcí (8.6). Vytváříme zde ikonku pomocí funkce `Icon` a text. Ikonka přijímá parametr specifikující objekt reprezentující danou ikonku a text používaný pro přístupové služby. Oba elementy jsou pak seskupeny do řádku pomocí kontejnerové composable funkce `Row`.

Výpis kódu 8.6 Příklad skládání composable funkcí. Vykreslí ikonku a text vedle sebe.

```
Row {
  Icon(
    imageVector = Icons.Filled.ArrowBack,
    contentDescription = "Tlacitko zpet",
  )
  Text(text = "Detail hlasovani)
}
```

Pomocí parametrů lze měnit i vzhled a chování composable funkce. (8.7). Vytváříme zde tlačítko s textem a černým okrajem, které při kliknutí vypíše do konzole text.

Výpis kódu 8.7 Příklad parametrů pro změnu vzhledu a chování.

```
Button(
  onClick = { println("Kod pro ulozeni") },
  border = BorderStroke(0.dp, Color.Black)
) {
  Text(text = "Ulozit")
}
```

Kompozice a rekompozice

Kompozice funkce je proces, kdy se zavolá. Rekompozice funkce je pak proces, kdy se zavolá znovu v reakci na změnu jejího lokálního stavu nebo stavu ve ViewModelu, na kterém je závislý, nebo na rekompzici rodiče. Lokální stav composable funkce je popsána v následující sekci. Důsledkem je to, že změna composable funkce (jejího stavu) spustí rekompozici pouze této funkce a všech funkcí, které volá. Ostatní funkce zůstanou nedotčené. Díky tomu dojde k rekompozici pouze u funkcí, kde se něco změnilo nebo potěnciálně změnilo.

Stav composable funkce

Composable funkce mohou v sobě držet lokální stav. Kód (8.8) ukazuje příklad jeho použití. Zde vidíme text a tlačítko. Text se vykresluje podmíněně podle aktuální hodnoty proměnné `expanded`. Defaultní hodnota proměnné `expanded` je `false`, a tudíž při první kompozici funkce se text nevykreslí. Tlačítko v reakci na kliknutí této proměnné nastaví opačnou hodnotu, což spustí rekompozici funkce. Aby změna této proměnné spustila rekompozici, musí být typu `State`. Aby se proměnná však vůbec dala měnit, musí být typu `MutableState`, a proto je defaultní hodnota `false` obalena do funkce `mutableStateOf`. Aby se při rekompozici nenastavila opět defaultní hodnota `false`, ale nová hodnota, musí si composable funkce tento stav pamatovat napříř rekompozicemi, a toho dosáhneme pomocí funkce `remember`. Klíčové slovo `by` je syntactic sugar, který deleguje vrácení hodnoty na funkci `remember`. Pro nás to znamená jenom to, že typ proměnné `expanded` je Boolean a ne State¡Boolean¿. Pracuje se s tím pak lépe.

Výpis kódu 8.8 Příklad composable funkce používající lokální stav.

```
  @Composable
  private fun MyExpandableContent() {
    var expanded by remember {mutableStateOf(false)}

    Row {
      if (expanded) {
        Text(text = "Zbytek obsahu")
      }
      Button(onClick = { expanded = !expanded },
      ) {
        Text(text = "Klikni pro zobrazeni zbytku obsahu")
      }
    }
  }
```

Mnohem obvyklejší je však stav zapouzdřit do ViewModelu, který je určený k ukládání stavu UI, jak je ukázáno na (8.9). ViewModel si pamatuje stav i při navigaci na jinou obrazovku a zpět. Díky může data stažená např. z internetu přepoužít a nemusí je zbytečně stahovat vícekrát, pokud se data málo mění.

Výpis kódu 8.9 Příklad composable funkce používající stav z ViewModelu.

```
@Composable
private fun MyExpandableContent(viewModel: MyViewModel) {
    val expanded = viewModel.expandeed

    Row {
      if (expanded) {
        Text(text = "Zbytek obsahu")
      }
      Button(onClick = { viewModel.toggleExpanded() },
      ) {
        Text(text = "Klikni pro zobrazeni zbytku obsahu")
      }
    }
}

class MyViewModel : ViewModel() {
  val expanded = mutableStateOf(false)

  fun toggleExpanded() {
    expanded = !expanded
  }
}
```

zdroj: https://developer.android.com/jetpack/compose/mental-model

Activity

Aktivita je jedna z hlavních komponent aplikace a reprezentuje obrazovku, do které vkládáme naše uživatelské rozhraní. V rámci aplikace je použita především jako vstupní bod pro vytvoření UI a pro počáteční inicializace stavu aplikace. Aktivita se může nacházet v různých stavech podle toho, zda ji má uživatel v popředí nebo pozadí, nebo zda ji vypíná. Nás bude zajímat především stav, kdy je aktivita poprvé vytvořena, tedy když je aplikace spuštěna. Pro detekci tohoto stavu poskytuje aktivita funkci `onCreate`, která se zavolá, když je aktivita vytvořena. V této funkci

lze pak provádět žádané operace (8.10). Funkce `setcontent` vezme composable funkci `PspApp`, zakomponuje ji do aktuální aktivity a nastaví ji jako kořenovou composable funkci.

Výpis kódu 8.10 Třída MainActivity. (Soubor `psp_fe/app/MainActivity`)

```
class MainActivity : ComponentActivity() {

  ...

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    ...

    setContent {
      PspApp()
    }
  }

  ...
```

8.2.1 Implementace prezentační vrstvy

Popis implementace prezentační vrstvy je rozděleno na popis implementace uživatelského rozhraní a popis implementace držitelů stavů a logiky. Nejdřív bude popsána první implementace a po ní druhá.

Uživatelské rozhraní bylo implementováno pomocí knihovny Jetpack Compose. Pro implementaci byly použity následující composable funkce:

- Column - Sloupcový kontejner pro elementy. Hodí se pro pozicování UI elementů na obrazovce. Nehodí se pro dlouhé seznamy elementů ve sloupci, jelikož nedělá rekompozici pouze viditelných elementů, ale všech. Nejde jím scrollovat. Používám ji ve všech místech, kde je potřeba napozicovat UI elementy pod sebou.
- LazyColumn - Sloupcový kontejner pro element zoptimalizovaný pro dlouhé seznamy. Dělá rekompozici pouze pro viditelné elementy. Lze jím scrollovat. Používám ji pro zobrazení seznamů, tedy na obrazovce pro seznam hlasování, seznam poslanců, seznam hlasování jednotlivých klubů a seznam hlasování poslance.
- Row - Řádkový kontejner pro elementy.
- Text - Textový element.
- Image - Obrázkový element. Obrázek lze získat pomocí URL adresy. Používám ho pro zobrazení profilové fotky poslance a logů klubů.
- Icon - Funkce pro různé ikonky, např. pro ikonky v hlavičce nebo v dolní liště.
- IconButton - Tlačítko, který má místo textu ikonky. Používám ji hlavičce pro vytvoření ikonky pro vyhledávání v seznamu hlasování nebo poslanců.
- OutlinedButton - Tlačítko s kontrastem barev mezi pozadím a obvodem.
- OutlinedTextField - Textové pole s kontrastem barev mezi pozadím a obvodem.
- FloatingActionButton - Plovoucí tlačítko. Používám ji pro skok na začátek seznamu.

- Scaffold - Kontejner umožňující jednoduchým způsobem sdílet UI element mezi více obrazovkami. Umožňuje také jednoduchým způsobem přidat plovoucí tlačítko.
- 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 kontextu. Používám ji pro lokální přepsání globálních barev.
- Box - Kontejner umožňující skládat UI elementy na sebe. Používám ho pro vytvoření popupu s nastavením volebního období v nastavení. Na pozadí je seznam nastavení a na něm se objeví daný popup.
- SettingsMenuLink - Composable funkce z knihovny Alorma pro snadné vytvoření prvku v seznamu nastavení přesně podle návrhu. Knihovna ji definuje defaultní vzhled.
- ListItemPicker - Composable funkce z knihovny Alorma reprezentující seznam hodnot, kterým lze scrollovat. Knihovna ji definuje defaultní vzhled.
- Card - Karta obsahující libovolný obsah a prvek pro akci. Používám ji např. pro elementy v seznam hlasování.
- TabRow - Lišta s taby. Používám ji na obrazovce s detailem hlasování a na obrazovce s detailem poslance.
- Tab - Reprezentuje konkrétní obrazovku v rámci tabu.
- HorizontalPager - Layout umožňující horizontálne scrollovat obsahem. Používám ho pro scrollování mezi taby.

Stav UI a logika pro obsluhu změn UI jsou zapouzdřeny v držitelích stavů a logiky, což jsou view modely. Jsou to objekty, s kterými komunikuje Jetpack Compose pro získání dat nebo pro informování o nějaké události. Na (8.11) lze vidět ukázku využití view modelu. Na začátku view modelu se nachází stav pro seznam hlasování. `PagingData` je datový typ pro stránkovaný obsah. `MutableStateFlow` je datový typ pro flow, který lze měnit. Flow je vyžadován pro implementaci stránkování pomocí `PagingData`. Vně view je viditelná pouze imutabilní verze stavu `StateFlow`. Pro bezpečnost se se stavem dá manipulovat pouze uvnitř view modelu. Při spuštění view modelu se proveden coroutinová operace v rámci scopu view modelu. To znamená, že pokud je view model odstraněn, zruší se i coroutinová operace. Coroutina získává data pomocí funkce `getVotes`. Ta vyžaduje volební rok a klíčová slova pro filtrování seznamu. Jakmile se změní jedna z těchto dvou hodnot, měly by se stáhnout nová data. Obě hodnoty jsou implementeovány jako flow, a proto je nejdřív kombinujeme a zavoláme na nich terminální operátor `collectLatest`. Emitované hodnoty se pomocí `cachedIn(viewModelScope)`zakešují v rámci view modelu. To znamená, že pokud se na daném flow opět zavolá terminální operátor v jiné části kódu, vrátí se zakešovaná hodnota.

Výpis kódu 8.11 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
        }

      }
    }
  }

}
```

Navigace mezi obrazovkami je implementována pomocí composable funkce `NavHost`, která definuje jednotlivé destinace. Destinace reprezentuje obrazovku, ke které lze pomocí této funkce navigovat. Destinace je reprezentována řetězcem. Např. destinace pro seznam hlasování se nazývá "votes". Pro samotnou navigaci se používá objekt `NavHostController`, na které zavoláme metodu `navigate` a předáme ji název destinace, ke které chceme navigovat. Pro navigaci k detailu hlasování nebo poslance používá sepcifičtější název destinace, např. "votes/1234"nebo "member/5678".

8.2.2 Implementace doménové vrstvy

Ukázka implementace doménové vrstvy pro seznam hlasování (8.12). Objekt `Pager` je vstupním bodem pro stránkovací mechanizmus. Nastavujeme mu defaultní velikost stránky, v době psaní práce je to 20. Dále mu předáme `VotesPagingSource` reprezentující zdroj stránkovaných dat. Pager poskytuje proměnnou flow, která bude emitovat stránky. `flowOn` nastaví, aby sbírání flow probíhalo na speciálním vlákně IO, aby neprobíhalo na hlavním vlákně. Není to nutné díky používání coroutinů, ale je to dobrá praxe to používat pro síťové operace.

Výpis kódu 8.12 Ukázka využití use caseu pro získání 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
  .flowOn(Dispatchers.IO)

}
```

Doménové vrstvy pro obrazovky, které nepoužívají seznam, jsou přímočařejší. Ukázku takového use caseu lze vidět na (8.13).

Výpis kódu 8.13 Ukázka využití use caseu pro získání detailu hlasování

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

  operator fun invoke(id: Int): Flow<VoteDetails> =
    voteRepository.getVoteDetail(id)
    .flowOn(Dispatchers.IO)
}
```

8.2.3 Implementace datové vrstvy

Datová vrstva slouží pro abstrahování doménové vrstvy od konkrétních datových zdrojů. V aplikaci je použit jediný datový zdroj, a to vzdálený pro získání data z backendu. V datové vrstvě jsou získávána data z datového zdroje v podobě síťových entit. Ty jsou následně namapována na doménové entity pro použití doménovou vrstvou. Ukázka použití je na (8.14).

Výpis kódu 8.14 Ukázka datové vrstvy pro data o hlasováních

```
class VoteRepositoryImpl @Inject constructor(...) : VoteRepository {

  override suspend fun getVotes(
    page: Int,
    size: Int,
    electionYear: Int,
    searchText: String
  ): List<Vote> =
  pspRemoteDataSource.getVotes(
    page = page,
    size = size,
    electionYear = electionYear,
    searchText = searchText
  ).map { it.toDomain() }

  ...
}
```

Datové zdroje jsou pouze abstrakcí nad konkrétním rozhraním pro získání daného zdroje. Vzdálený datový zdroj tedy abstrahuje datovou vrstvu od přímé komunikace s knihovnou pro získání dat z internetu. Na (8.15) je ukázka datového zdroje. Funkce `getAppState()` pouze zavolá funkci definovanou pomocí knihovny Retrofit, která bude popsána v následující sekci.

Výpis kódu 8.15 Ukázka datového zdroje

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

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

  ...
}
```

8.2.4 Implementace síťové vrstvy

Síťová vrstva je implementována pomocí knihovny Retrofit. Je to HTTP klient pro komunikaci se zdroji na internetu. Aplikaci ho používá pro získání veškerých dat z backendu. Na (8.16) je ukázka použití. URL adresu daného zdroje specifikujeme pomocí anotace `@GET`. Pomocí anotací `Query` specifikujeme query parametry pro jako číslo stránky a počet stránek. Návratovou hodnotou je seznam API entit. Implementaci této funkce má na starosti knihovna. Data z backendu jsou ve formátu JSON a jsou namapována na objekt v návratovém typu.

Výpis kódu 8.16 Ukázka použití knihovny Retrofit pro získání seznamu hlasování z backendu

```
interface PspApi {

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

}
```

zdroj: https://square.github.io/retrofit/

8.3 Backend

8.3.1 Použité nástroje a technologie

Intellij IDEA

Backend byl vyvíjen ve vývojovém prostředí Intellij IDEA. Hlavní důvody pro výběr tohoto IDE:

- 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.

  zdroj: https://www.jetbrains.com/idea/

Maven

Maven je stejně jako Gradle nástroj pro automatizaci sestavování programu. Původně byl použit Gradle kvůli čitelnější syntaxi. Maven byl zvolen z historických důvodů. Původně bylo v plánu backend nasadit na Cloud Azure. Ten poskytoval plugin pro maven, který umožňoval backend nasadit a zprovoznit pomocí jednoho příkazu. Kvůli omezeného free tieru však byl backend nakonec nasazen na jiný cloud (více v kapitole o nasazení). Maven je tedy pozůstatek historického rozhodnutí. Na funkčnosti aplikace to však nemá žádný vliv, a proto se už nepřešlo z Mavenu na Gradle.

zdroj: https://maven.apache.org/
https://azure.microsoft.com/en-us

[1] Technologie Spring Boot bude popsána později.

Spring Boot

Před popisem technologie Spring Boot si popíšeme technologii Spring, na které je Spring Boot založen. Spring je open-source framework pro vývoj enterprise aplikací. Je to seskupení různých nástrojů pro řešení různých problémů. Pro účely této práce byly využity nástroje pro vytvoření webových aplikacím, ale umožňují implementovat i backend s REST API. Hlavními benefity Springu je množství použitelných nástrojů a dependency injection pomocí anotací.

Spring Boot je framework, který je postavený na Springu a který má za cíl redukci boilerplate kódu a nutnost konfigurace, a tím urychlit vývoj aplikace. Toho dosahuje pomocí autokonfigurace, což je vlastnost Spring Bootu, kdy jsou jednotlivé komponenty (např. rozhraní pro komunikaci s databází, webový server, ORM) automaticky nakonfigurovány Spring Bootem pomocí defaultních hodnot. Defaultní hodnoty jsou Spring Bootem nastaveny na nejčastější hodnoty, aby se tím pokrylo co nejvíce situací, na které může vývojář narazit. Důsledkem je to, že pro zprovoznění komponent je potřeba pouze nainstalovat jejich závislost. Pokud bude potřeba použít jinou konfiguraci (např. pro připojení k databázi chceme použít jiný než defaultní port), Spring Boot to umožňuje. Pointa je však, že knihovnu pouze stáhneme bez nutnosti konfigurace nebo s minimální konfigurací a aplikace lze hned spustit.

Alternativou ke knihovně Spring Boot je knihovna Ktor, pomcí které lze taktéž vytvořit backend s REST API. Výhodou této knihovny je, že je určená pro psaní v Kotlinu. Lze tedy využít všechny výhody tohoto jazyka. Nevýhodou je, že nepodporuje defaultně dependency injection. Pro něj se musí zvlášť stáhnout knihovna a a tu nakonfigurovat. Další nevýhodou oproti Spring Bootu je chybějící autokonfigurace. Vše se tedy musí manuálně nakonfigurovat.

zdroj:
https://spring.io/
https://spring.io/why-spring
https://spring.io/web-applications
https://ktor.io/

Java

Java byla od začátku hlavním programovacím jazykem pro vývoj aplikací pomocí technologie Spring Boot. Od roku 2017 přišla integrace jazyka Kotlin do Spring Bootu a v dokumentaci jsou ukázkové kódy psány jak v Javě tak i Kotlinu. Java má oproti Kotlin větší podporu v komunitě, co se týče vývoji ve Spring Bootu. Když jsem při psaní v Javě narazil na nějaký problém, mnohem šlo najít na internetu řešení než při psaní v Kotlinu. Jedním řešením je hledat řešení v Javě a javovský kód pomocí automatického nástroje ztransfomovat do Kotlinu, výsledný kód bylo však potřeba vždy prošistit, jelikož u všech proměnných vždy obsahoval datové typy, které lze v Kotlinu v některých případech vynechat. Zároveň výstupní kód někdy nebyl kvůli nějaké drobnosti kompilovatelný. Z toho důvodu jsem se rozhodl pro použití Java. Zdá se však, že podpora pro psaní aplikací Spring Boot pomocí Kotlinu je čím dál tím větší. Když bych měl možnost backend přepsat, znovu bych zauvažoval o použití Kotlinu, jelikož je to velmi dobrý programovací jazyk.

zdroj: https://spring.io/guides/tutorials/spring-boot-kotlin/
https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0
zdroj:
https://docs.spring.io/spring-boot/docs/current/reference/html/index.html

MySQL

Na databázi backendu nebyly kladeny velké nároky. Pouze bylo potřeba data perzistentní uložit, aby byly připravené pro použití mobilní aplikací. Zvolil jsem tedy MySQL, s kterým jsem měl zkušenosti. Bylo by však možné použít např. i PostreSQL. Na funkčnosti backendu by to však nemělo vliv.

8.3.2 Prezentační vrstva

V prezentační vrstvě byly implementovány jednotlivé endpointy REST API, které přijímají dotazy od mobilní aplikace a vrací ji předzpracovaná data uložena v databázi. Ukázku implementace endpointu pro získání detailu konkrétního hlasování lze vidět na (8.17). Endpoint pro HTTP GET požadavky je ve Spring Bootu implementován tak, že vytvoříme metodu s libovolným názvem a označíme ji anotací `@GetMapping`. V parametru této anotace pak specifikujeme URL adresu daného endpointu. Celá metoda se musí nacházet ve třídě s anotací `@RestController`. Endpointy jsou rozděleny do několika tříd podle typu dat (jedna třída pro entity hlasování, druhá pro entity poslance a třetí pro entity klubů). Pokud URL adresa obsahuje parametr identifikující daný zdroj, musí být obsažen v parametru metody a oanotován pomocí `@PathVariable`

Výpis kódu 8.17 Ukázka kódu pro vytvoření endpointu

```
// Soubor VoteController.java
@RestController
public class VoteController {

  private final VoteService service;
  private final VoteMapper mapper;

  ...

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

Query parametry jsou implementovány pomocí anotace `RequestParam`, kterému je v parametru předán název parametru:

Výpis kódu 8.18 Ukázka endpointu s request parametrem

```
// Soubor VoteController.java
@RestController
public class VoteController {

  private final VoteService service;
  private final VoteMapper mapper;

  ...

  @GetMapping("/vote")
  public ResponseEntity<List<GeneralVote>> getVotes(
    @RequestParam(name = "description") String description
    ...
  ) {
    ...
  }
}
```

Podobným způsobem jsou implementovány query parametry pro stránkování (číslo a velikost stránky). Kromě povinného parametru pro název query parametru lze specifikovat i parametr `required` pro nastavení volitelnosti nebo `defaultValue` pro nastavení defaultní hodnoty query parametru.

Na dvou předchozích ukázkách lze vidět použití hodnot `service` a `mapper`. První reprezentuje

doménovou vrstvu, která získá data z databázové vrstvy a druhá obsahuje metody pro mapování objektů, pokud je to potřeba (pokud je struktura entit uložených v databázi jiná než struktura entit pro REST API). Takto jsou implementovány všechny controllery. Controllery, které obsahují endpoint, který vrací seznam, navíc sestavují hlavičky pro HTTP odpověď. K tomu je použita třída `HttpHeaders`, kterou poskytuje Spring framework. Instanci této třídy se jednoduchým způsobem nastaví potřebné hodnoty a objekt je controllorem vracen:

Výpis kódu 8.19 Ukázka nastavení hlaviček pro stránkování

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

  ...

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

  return responseHeaders;
}

// Soubor VoteController.java
@GetMapping("/vote")
public ResponseEntity<List<GeneralVote>> getVotes(...) {

  ...

  HttpHeaders headers = GenericAndPaginationHeaderGenerator
    .buildHeaders(pagedResult.getTotalPages(), page);

  ...

  return ResponseEntity
    .ok()
    .headers(headers)
    .body(generalVotes);
}
```

8.3.3 Doménová vrstva

Doménová vrstva má na starosti business logiku aplikace a abstrahování prezentační vrstvy od implementačních detailů databázové vrstvy. V našem případě backend neobsahuje business logoiku, pouze filtruje data. Ukázka kódu pro získání detailu poslance:

Výpis kódu 8.20 Ukázka kódu pro získání detailu poslance

```
// Soubor MemberService.java
public Member getMember(int id) {
  return memberRepository
  .findById(id)
  .orElseThrow(() -> new MemberNotFoundException(id));
}
```

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

Výpis kódu 8.21 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 8.22 Ukázka kódu pro sestavení objektu pro stránkování

```
public static Pageable buildPageable(...) {
  Pageable pageable;

  ...

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

  return pageable;
}
```

Doménová vrstva pro ostatní entity jsou implementovány obdobně jako bylo popsáno výše. Výjimkou je metoda pro získání detailu hlasování. Jedním z atributů hlasování jsou statistky hlasování. Výpočet počtu omluvených a nepřihlášených poslanců příliš zpomalovalo zpracování zdrojových dat přes uložením do databáze. Z toho důvodu je tento výpočet prováděn až za běhu, kdy se mobilní aplikace nad daným detailem hlasování dotazuje:

Výpis kódu 8.23 Ukázka dopočtu statistik pro detail hlasování za běhu v doménové vrstvě

```
public Vote getVote(int id) throws IOException {
  ...

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

  ...

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

8.3.4 Databázová vrstva

V databázové vrstvě získávám data z databáze pomocí dotazů. Pro komunikaci s databází je použita knihovna Hibernate, která poskytuje objektově-relační mapování, díky kterému jsme

abstrahování od databázových tabulek a místo toho pracujeme s objekty (entitami), které jsou danou knihovnou namapovány na dané tabulky. Příkladem takového objektu je entita pro hlasování, jejíž ukázka je na (**??**). Pomocí anotace `Entity` říkáme, že daný objekt je databázovou entitou. Hibernate nám automaticky vytvoří tabulku s danými atributy v databázi. Anotace `@Getter` je z knihovny Lombok a slouží pro vygenerování getterů pro všechny atributy. Anotací `@Id` specifikujeme atribut, který má být primárním klíčem v tabulce.

Výpis kódu 8.24 Entita Vote reprezentující hlasování

```
// dalsi anotace
@Entity(name = VOTE)
@Getter
public class Vote {

  @Id
  private int id;

  private LocalDateTime dateTime;

  // zbytek atributu
}
```

Pro dotazování se nad daty používám knihovnu spring-data-jpa, 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. Největší sílou této knihovny je však možnost vytvoření vlastních metod, kterým se říká query metody. Dotaz se sestaví na základě pojmenování query metody podle určitých pravidel. Např. následující metoda vrací seznam hlasování ve volebním období, který začal daným volebním rokem:

Výpis kódu 8.25 Repozitář pro hlasování

```
// Soubor VoteRepository.java
@Repository
public interface VoteRepository extends JpaRepository<Vote, Integer> {

  ...

  List<Vote> findByElectionYear(int electionYear);

}
```

Přitom nebylo potřeba metodu implementovat. Knihovna si na základě názvu metody implementaci vygeneruje. Metodu lze modifikovat, aby vracela stránkovaný obsah:

Výpis kódu 8.26 Ukázka query metody pro dotazování se nad stránkovaným obsahem

```
// Soubor VoteRepository.java
Page<Vote> findByElectionYear(int electionYear, Pageable pageable);
```

Lze i vytvořit komplexnější dotaz, který se dotazuje nad entitou na základě dvou podmínek:

Výpis kódu 8.27 Ukázka query metody s dvěma podmínkami

```
// Soubor MembershipRepository.java
boolean existsByPersonIdAndAgencyId(int personId, int agencyId);
```

zdroj: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

8.4 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 (https://www.psp.cz/sqw/hp.sqw?k=1300).
- **Extrakce datových souborů** - Ze zdrojových souborů jsou vyextrahovány datové soubory, které jsou ve formátu UNL.
- **Pročištění dat** - Některé datové soubory obsahují duplicitní záznamy. Ty jsou pro snadnější parsování odstraněny.
- **Parsování dat** - Datové soubory jsou zparsovány a načteny do Java objektů.
- **Transformace dat** - Objekty jsou ztransformovány do požadované podoby.
- **Uložení dat do databáze** - Ztransfomovaná data jsou perzistentně uložena do databáze.

8.4.1 Stahování zdrojových souborů

Stahování zdrojových souborů má na starosti třída následující třída:

Výpis kódu 8.28 Třída pro stahování zdrojových souborů

```
// Soubor PspFilesDownloader.java
public class PspFilesDownloader {
  public static void downloadFiles() throws IOException {
    downloadHlasovani();
    downloadPoslanci();
  }

  // dalsi metody
}
```

Tato třída pro stahování souborů na základě URL interně používá knihovnu `commons-io` (8.29). Knihovna poskytuje metodu `copyURLToFile`, která akceptuje parametr pro URL zdrojového souboru a parametr pro lokální soubor, do kterého se má zdrojový soubor nakopírovat.

Výpis kódu 8.29 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);

    ...
  }
}
```

zdroj: https://commons.apache.org/proper/commons-io/

8.4.2 Extrakce datových souborů

Pro extrakci datových souborů byla použita knihovna `zip4j`. Ta poskytuje třídu `ZipFile` pro vytvoření objektu reprezentujícího lokální soubor ve formátu zip. Konstruktoru této třídy je předána lokace souboru v souborovém systému. Tato třída poskytuje metodu `extractFile()` pro extrakci souborů ze zipu. Akceptuje tři parametry:

- Název souboru, který se má vyextrahovat.
- Adresář, do které se má soubor vyextrahovaný soubor uložit.
- Výsledný název ukládaného souboru. V našem případě je stejný jako název před vyextrahováním souboru.

Zde je ukázka kódu pro popsanou extrakci souborů:

Výpis kódu 8.30 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);

  }

}
```

zdroj: https://github.com/srikanth-lingala/zip4j

8.4.3 Pročištění dat

Některé datové soubory obsahují duplicitní záznamy. Ty jsou odstraněny pomocí skriptu napsaného v jazyce Bash:

Výpis kódu 8.31 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 předpokládá ve svém prvním argumentu cestu ke adresáři, kde se nachází datové 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 duplicity měly jejich originální název.

Skript je volán ze souboru `PspFilesCleaner`. Důvodem pro odstranění duplicit pomocí jazyka Bash a ne přímo pomocí Javy je to, že v Bashi jsou operace jednodušší a rychlejší než v Javě.

8.4.4 Parsování dat

Pro parsování zdrojových dat používám knihovnu `opencsv`. Ta umožňuje parsovat CSV soubory tak, že každý řádek v souboru namapuje na objekt. Mapování sloupců v souboru na atributy objektu je implementováno přidáním anotací k příslušným atributům. Vysvětlíme si na příkladu. Nejdříve je potřeba vytvořit třídu, do jejíž instancí se napamatují záznamy ve zdrojovém souboru. Ta bude mít u atributů anotaci určující pozici neboli číslo sloupce záznamu. Hodnota tohoto sloupce se pak napamatuje na tento atribut:

Výpis kódu 8.32 Parsování datového souboru omluvy.unl

```
// Soubor Omluva.java
public class Omluva {
  @CsvBindByPosition(position = 0)
  private int idOrgan;

  @CsvBindByPosition(position = 1)
  private int idPoslanec;

  // dalsi atributy
}
```

Nyní lze datový soubor zparsovat a namapovat na tuto třídu. Knihovna poskytuje třídu `CsvToBeanBuilder`, které předáme objekt typu —Reader—. Ta slouží pro čtení znaků ze streamu. V tomto případě je do streamu posílány data z datového souboru. Třídá `CsvToBeanBuilder` poskytuje metody, kterými specifikujeme:

- Třídu, na kterou se mají záznamy namapovat.
- Oddělovač hodnot v záznamu.
- Určení, kdy je hodnota null. V tomto případě hodnota interpretována jako null, pokud je prázdná.
- Ignorování složených závorek.

Výpis kódu 8.33 Parsování datového souboru omluvy.unl

```
// Soubor OmluvyParser.java

public class OmluvyParser extends UnlParser {

  public List<Omluva> read() {
    String filePath = PspPath.Unl.OMLUVY;
    BufferedReader reader = getReader(filePath);

    return new CsvToBeanBuilder<Omluva>(reader)
    .withType(Omluva.class)
```

```
    .withSeparator(UNL_SEPARATOR)
    .withFieldAsNull(UNL_NULL_SEPARATOR)
    .withIgnoreQuotations(true)
    .build()
    .parse();

  }
}
```

zdroj: https://opencsv.sourceforge.net/

8.4.5 Transformace a uložení dat

Po zpasrování datových souborů a namapování záznamů na objekty lze tyto objekty začít ztransformovat do žádané podoby. Kód pro transformaci má u všech typů dat podobný tvar:

Výpis kódu 8.34 Transformace objektu Omluva na databázový objekt Excuse

```
// Soubor ExcuseLoader.java

public class ExcuseLoader extends BaseLoader {

    private final OmluvyParser omluvyReader;
    private final ExcuseRepository excuseRepository;

    public void load() {
      excuseRepository.deleteAllInBatch();
      List<Omluva> omluvaList = omluvyReader.read();

      List<Excuse> excuses = omluvaList.parallelStream()
            .map(omluva -> {
              // transformace
            })
          .collect(Collectors.toList());

      Lists
        .partition(excuses, BATCH_SIZE)
        .forEach(excuseRepository::saveAll);

  }

}
```

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, což stream, který potenciálně zpracuje objekt v seznamu paralelně. Každý objekt je ztransformován a namapován na databázový objekt. Konkrétní implementace transformace se liší od typu entity. Ztransformovaná data jsou perzistetně uložena 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 9

Testování

Kapitola 10

Nasazení

Kapitola 11

Spuštění

Kapitola 12

Závěr

Příloha A

Příloha

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

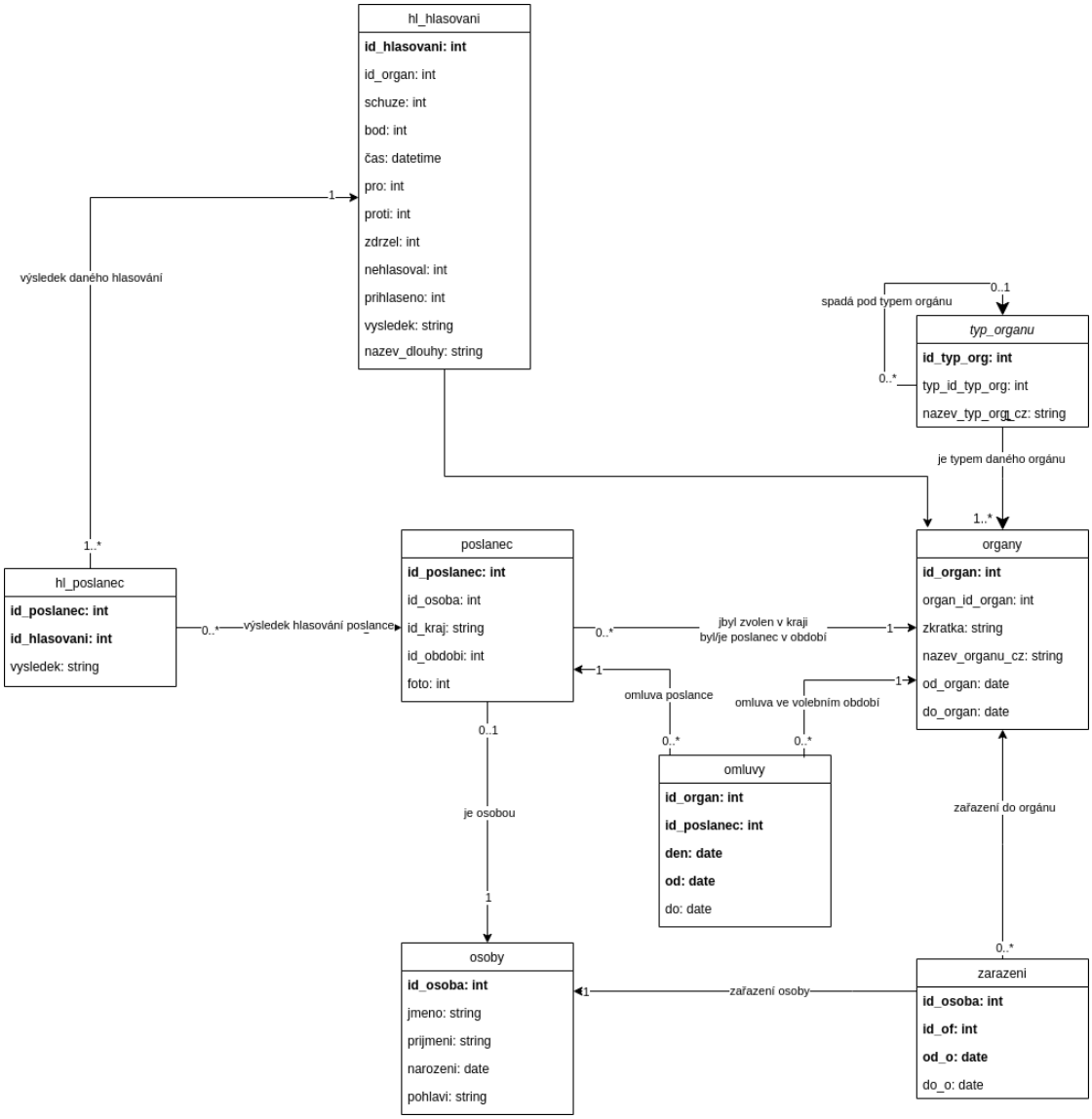
```
{
  "election_years": [
  2021,
  2017,
  2013,
  2010,
  2006,
  2002,
  1998,
  1996,
  1992
  ]
}
```

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

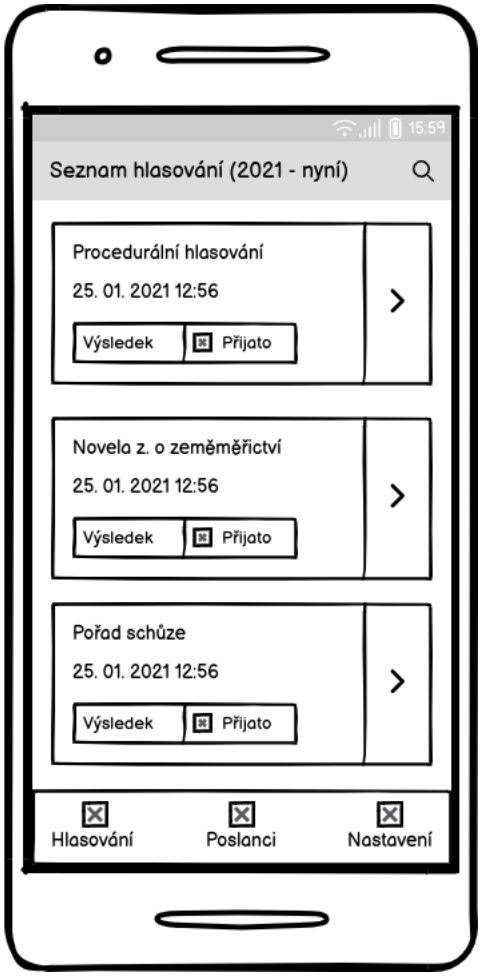
```
[
  {
    "id": 1,
    "date_time": "16. 12. 2022 13:29",
    "description": "Hlasovani 1",
    "result": "A"
  },
  {
    "id": 2,
    "date_time": "16. 12. 2022 13:26",
    "description": "Hlasovani 2",
    "result": "A"
  },
]
```

Výpis kódu A.3 Tělo odpovědi pro dotaz `dGET /api/votei`

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



Obrázek A.1 Diagram zdrojových dat



Obrázek A.2 Seznam hlasování

Obrázek A.3 Vyhledávání v seznamu hlasování

Obrázek A.4 Obrazovka pro seznam hlasování

```
  "date_time": "16. 12. 2022 13:29",
  "description": "Hlasovani 1,
  "result": "A",
  "steno_protocol_url": "http://www.psp.cz/eknih/2021ps/stenprot/
     048schuz/s048109.htm#h76",
  "yes_count": 100,
  "no_count": 0,
  "logged_off_count": 64,
  "excused_count": 0,
  "refrained_count": 36,
  "election_year": 0
}
```

Výpis kódu A.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 A.5 Tělo odpovědi pro dotaz ¨

```
[
  {
    "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 A.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/2021
     ps/poslanci/i6474.jpg",
  "election_year": 2021
}
```

Výpis kódu A.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": "@"
  }
]
```

Tabulka A.1 Struktura agency

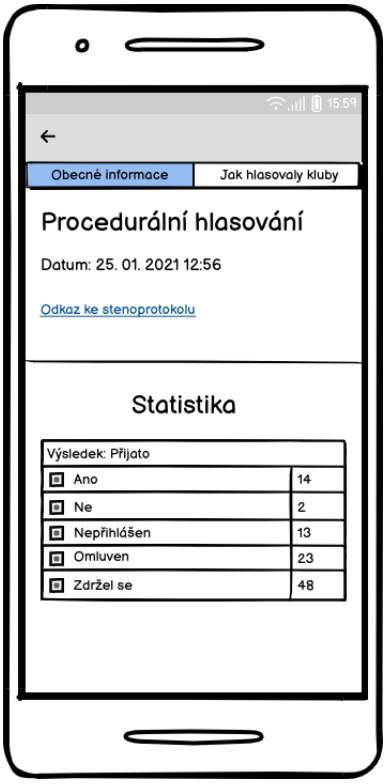
Název	Typ	Popis
id	int	identifikátor orgánu
abbreviation	varchar(255)	zkratka názvu orgánu
end_date	date(255)	datum zániku orgánu
name	varchar(255)	název orgánu
start_date	date	datum založení orgánu
_id	int	identifikátor typu orgánu

Tabulka A.2 Struktura excuse

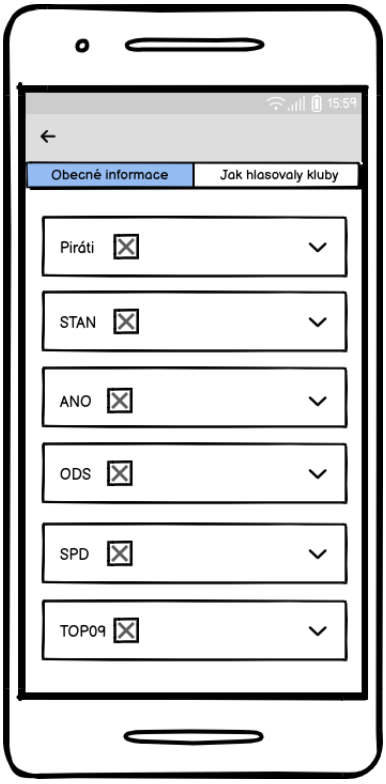
Název	Typ	Popis
member_id	int	identifikátor poslance, který je omluven
date	date	datum, kdy je poslanec omluven
start_time	time	čas, od kterého byl poslanec omluven
end_time	time	čas, do kterého byl poslanec omluven
election_year	int	první rok volebního období

Tabulka A.3 Struktura member

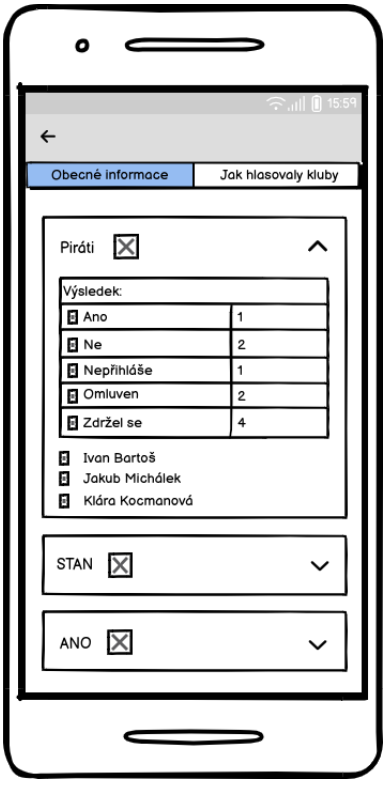
Název	Typ	Popis
id	int	identifikátor poslance, který je omluven
date_of_birth	date	datum narození
election_region	varchar(255)	volební kraj
election_year	int	první rok volebního období
gender	varchar(255)	pohlaví
member_from	date	datum začátku členství
member_to	date	datum konce členství
name	varchar(255)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



Obrázek A.5 Detail hlasování

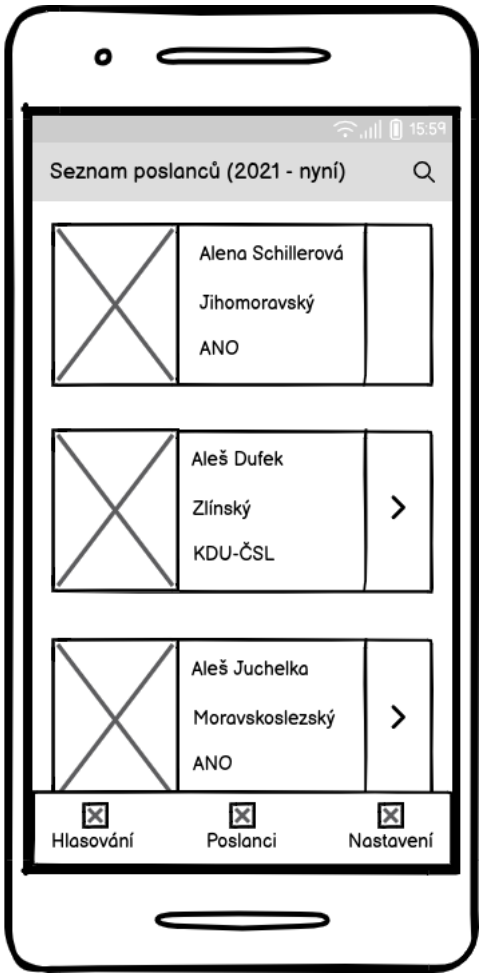


Obrázek A.6 Jak hlasovaly kluby

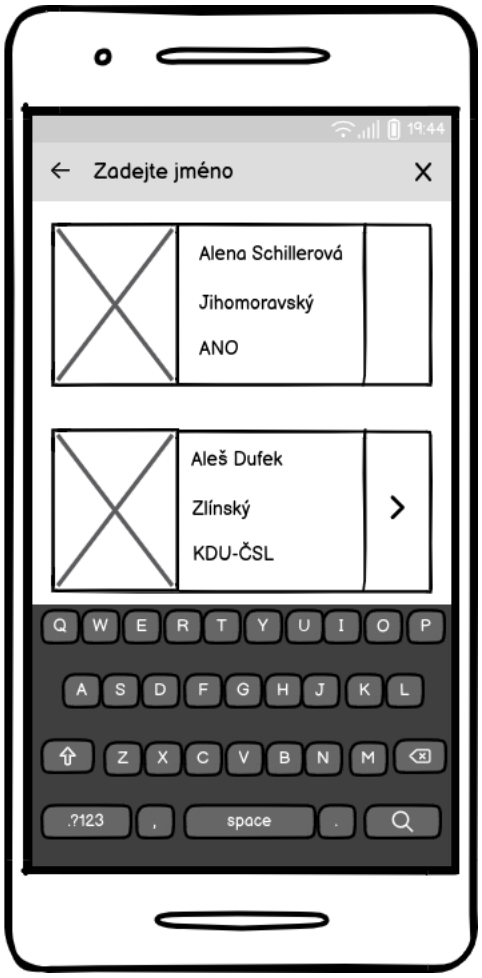


Obrázek A.7 Jak hlasovaly kluby

Obrázek A.8 Obrazovky pro detail hlasování



Obrázek A.9 Seznam poslanců

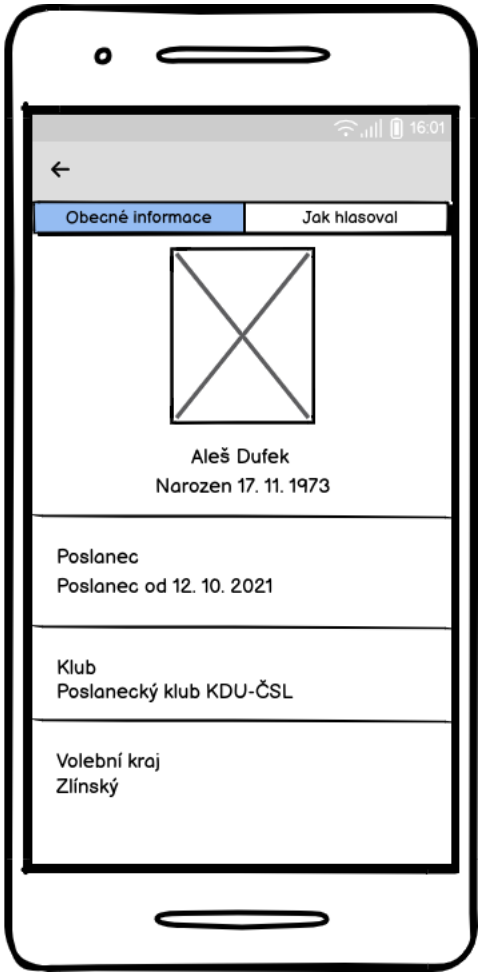


Obrázek A.10 Vyhledávání v seznamu poslanců

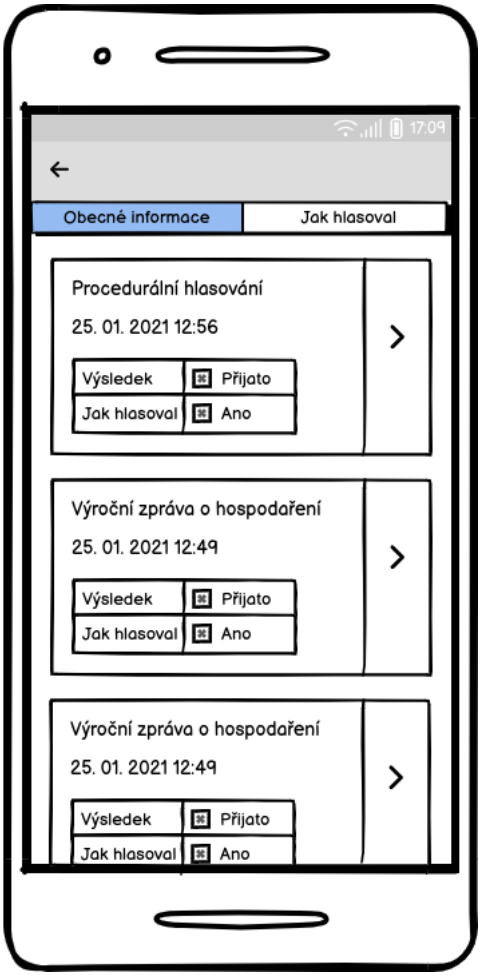
Obrázek A.11 Obrazovka pro seznam poslanců

Tabulka A.4 Struktura member_vote

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



Obrázek A.12 Detail poslance

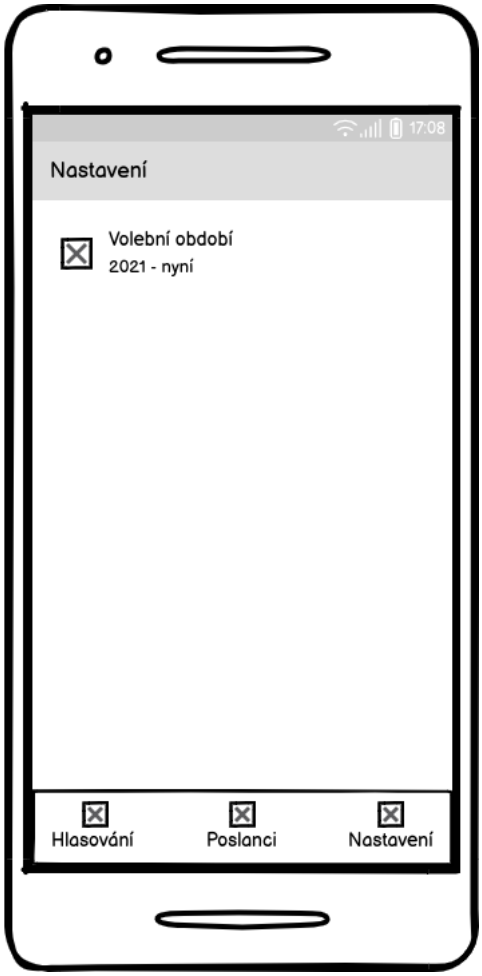


Obrázek A.13 Jak hlasoval/a poslanec/kyně

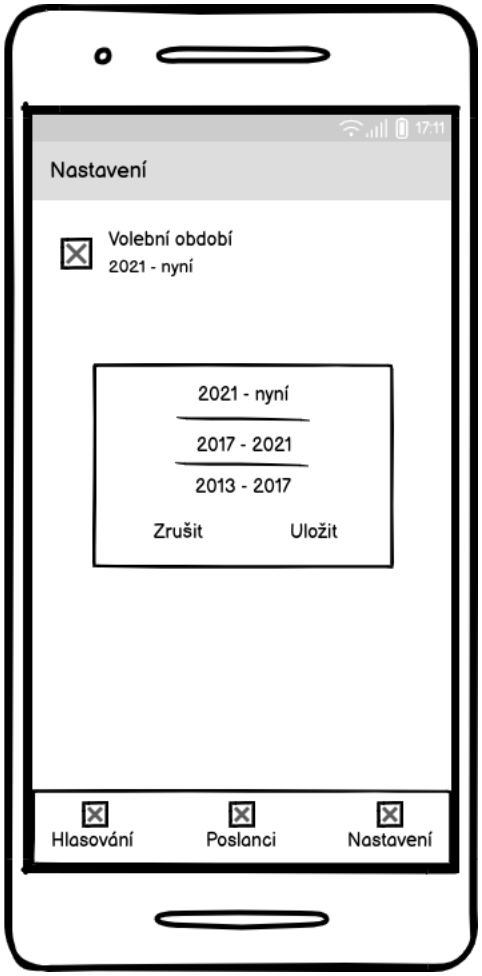
Obrázek A.14 Obrazovky pro detail poslance

Tabulka A.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



Obrázek A.15 Seznam nastavení



Obrázek A.16 Nastavení volebního období

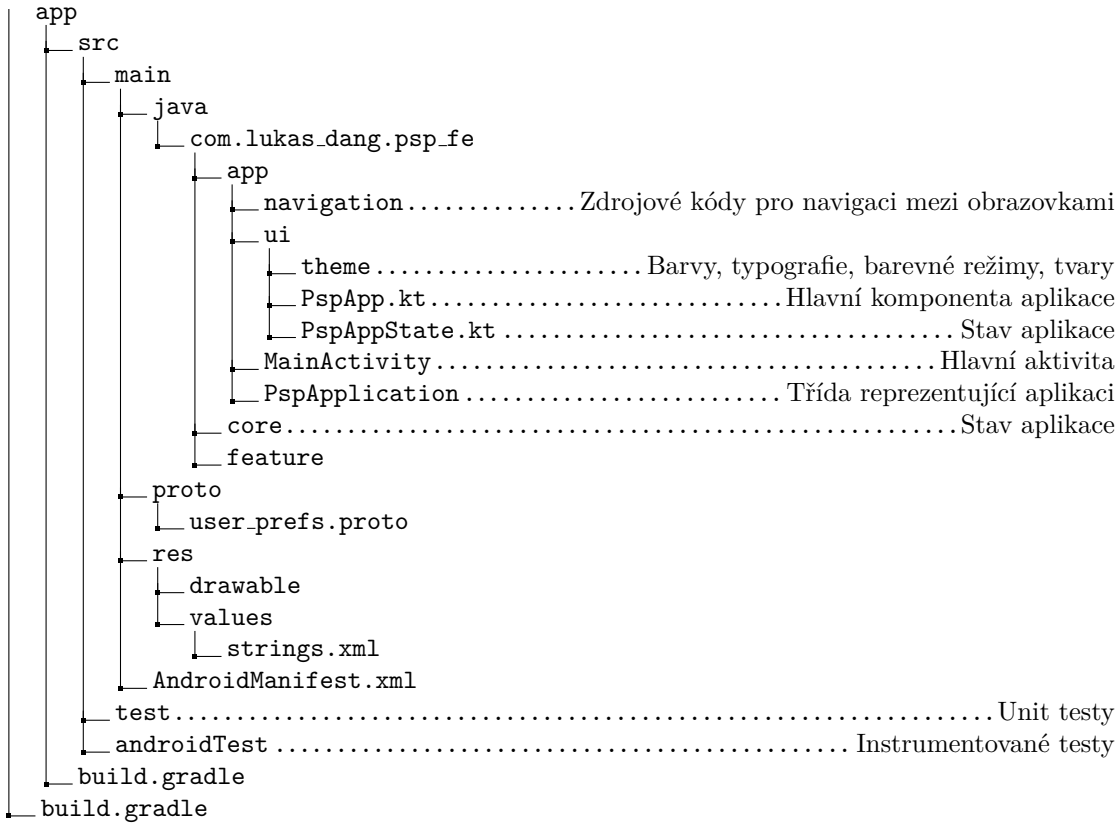
Obrázek A.17 Obrazovka pro nastavení

Tabulka A.6 Struktura party

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

Tabulka A.7 Struktura vote

Název	Typ	Popis
`date_time`	`datetime`	datum a čas hlasování
`description`	`varchar(255)`	popis hlasování
`election_year`	`int`	první rok volebního období
`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



Obrázek A.18 Adresářová struktura mobilní aplikace

Bibliografie

1. 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.
2. DEVELOPER, Android Politiscope. *politiscope* [online]. Jan 2020. [cit. 2023-01-26]. Dostupné z: `https://play.google.com/store/apps/details?id=com.junkie.android.politiscope&hl=en&gl=US`.
3. [online]. [B.r.]. [cit. 2023-01-26]. Dostupné z: `https://www.propublica.org/`.
4. [online]. [B.r.]. [cit. 2023-01-26]. Dostupné z: `https://theunitedstates.io/`.
5. 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`.
6. [online]. [B.r.]. [cit. 2023-01-26]. Dostupné z: `https://api.open.fec.gov/developers/`.
7. PARLAMENT ČESKÉ REPUBLIKY, Poslanecká sněmovna. *Poslanci a osoby* [online]. [B.r.]. [cit. 2023-02-08]. Dostupné z: `https://www.psp.cz/sqw/hp.sqw?k=1301`.
8. PARLAMENT ČESKÉ REPUBLIKY, Poslanecká sněmovna. *Hlasování* [online]. [B.r.]. [cit. 2023-02-08]. Dostupné z: `https://www.psp.cz/sqw/hp.sqw?k=1302`.
9. GOOGLE. *Guide to app architecture* [online]. [B.r.]. [cit. 2023-02-09]. Dostupné z: `https://developer.android.com/topic/architecture`.
10. TEAM, Indeed Editorial. *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`.
11. GOOGLE. *Android Studio* [online]. [B.r.]. [cit. 2023-02-09]. Dostupné z: `https://developer.android.com/studio`.
12. JETBRAINS. *Kotlin* [online]. [B.r.]. [cit. 2023-02-09]. Dostupné z: `https://kotlinlang.org/`.
13. 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/`.
14. ORACLE. *Java* [online]. [B.r.]. [cit. 2023-02-09]. Dostupné z: `https://www.oracle.com/java/`.
15. JETBRAINS. *Object expressions and declarations* [online]. [B.r.]. [cit. 2023-02-09]. Dostupné z: `https://kotlinlang.org/docs/object-declarations.html`.

16. JETBRAINS. *Asynchronous Flow* [online]. [B.r.]. [cit. 2023-02-09]. Dostupné z: `https://kotlinlang.org/docs/flow.html`.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF