

# Zadání diplomové práce

Název: Mobilní aplikace pro zobrazení výsledků hlasování Poslanecké

sněmovny

Student:Bc. Lukáš DangVedoucí:Ing. Ondřej JohnStudijní 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.

## Diplomová práce

# MOBILNÍ APLIKACE PRO ZOBRAZENÍ VÝSLEDKŮ HLASOVÁNÍ POSLANECKÉ SNĚMOVNY

Bc. Lukáš Dang

Fakulta informačních technologií Katedra webového inženýrství Vedoucí: Ing. Ondřej John 11. února 2023

České vysoké učení technické v Praze Fakulta informačních technologií

 $\ensuremath{{\mathbb O}}$  2023 Bc. Lukáš Dang. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Dang Lukáš. *Mobilní aplikace pro zobrazení výsledků hlasování Poslanecké sněmovny*. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

# Obsah

Po	oděkování	viii
Pr	rohlášení	ix
Al	bstrakt	x
Sh	nrnutí	xi
$\mathbf{Se}$	eznam zkratek	xii
1	Cíle	1
2	Poslanecká sněmovna2.1 Poslanecká sněmovna2.2 Hlasování v poslanecké sněmovně2.3 Webový portál psp.cz2.4 Motivace pro tuto práci	1 1 1 1
3	Funkční a nefunkční požadavky3.1Funkční požadavky3.2Nefunkční požadavky	1 1 2
4	Analýza existujících řešení         4.1 politiscope          4.1.1 Zhodnocení          4.2 Congress          4.2.1 Zhodnocení	1 1 2 2 3
5	Analýza zdrojových dat  5.1 Zdrojové soubory  5.2 Formát dat  5.3 Aktualizace  5.4 Datové typy  5.5 Licence  5.6 Datové soubory  5.7 Tabulky	1 1 2 2 2 2 3
6	Analýza softwarových architektur  6.1 Mobilní aplikace	1 1 3 3 3 3

iv Obsah

7	7.2	rh Uživat REST Datab	ΑI	PΙ										 						 			1 1 2 5
8	8.1 8.2 8.3	lemen Použit Impler Impler 8.3.1 8.3.2	té n mer mer In	ástr ntace	oje a e pre e do men	a te ezer méi itac	echi ntaò nov e da	nole éní é v ato	ogie vrs rstv vé v	tvy yy yrst	 tvy			 · ·		 		 		 · ·	•		1 9 15 15
9	9.1	Prezen 9.1.1 9.1.2 Zpracc 9.2.1 9.2.2 9.2.3 9.2.4 9.2.5	ntaò Da Ová St Ea Pa	éní v omé: atab	rstv nová ázov at . vání kce ( átění vání	va vá v vá v í zd dat í da í da	rstv vrst  roje ový at .	a . va . ový ch	  ch s sou 	sou bo	  .bo: rů 	  rů . 		 	 	    	 	 	 	 		 	19 20 21 22 24 24 25 25 26 26
10	10.1	<b>ování</b> Mobili Backei	ní a	_																			1 1 1
11		a <b>zení</b> Aplika Backer																					1 1 1
12		<b>śtění</b> Aplika Backer																					1 1 1
13	Závě	ér																					1
A	Přílo	oha																					3
Ol	osah j	přilože	ené	eho	mé	dia																	17

# Seznam obrázků

	Android aplikace politiscope											2 3
6.2	Architektura pro mobilní aplikaci pro Android [9] Složení UI vrstvy [9]											1 2 2
A.2 A.3 A.4 A.5 A.6 A.7 A.8 A.10 A.11 A.12 A.13 A.14 A.15 A.16 A.17	Diagram zdrojových dat Seznam hlasování Vyhledávání v seznamu hlasování Obrazovka pro seznam hlasování Detail hlasování Jak hlasovaly kluby Jak hlasovaly kluby Obrazovky pro detail hlasování Seznam poslanců Vyhledávání v seznamu poslanců Obrazovka pro seznam poslanců Detail poslance Jak hlasoval/a poslanec/kyně Obrazovky pro detail poslance Seznam nastavení Nastavení volebního období Obrazovka pro nastavení Adresářová struktura mobilní aplikace											4 5 5 9 9 9 10 10 11 11 11 12 12 13
		Se	<b>Z</b> ]	na	ar	$\mathbf{n}$	. 1	ta	ıb	)1	ıle	k
3.2 3.3	Funkční požadavky pro mobilní aplikaci											2 2 3 3
$5.2 \\ 5.3$	Datové typy dat											2 3 4 4

5.4	Tabulka zarazeni	5
5.5	Tabulka poslanec	5
5.6	Tabulka hl_hlasovani	6
5.7	Tabulka hl_poslanec	6
5.7	Tabulka hl_poslanec	7
5.8	Tabulka omluvy	7
	·	
A.1	Struktura agency	8
A.2	Struktura excuse	8
A.3	Struktura member	8
A.4	Struktura member_vote	10
A.5	Struktura membership	11
A.6	Struktura party	12
A.7	Struktura vote	13
11.,		10
	Seznam výpisů kóc	du
8.1	Příklad použití coroutiny	2
8.2	Příklad použití flow	4
8.3	Příklad použití DI pomocí knihovny Hilt	5
8.4	Ukázka konfigurace DI pro Hilt	6
8.5	Ukázka composable funkce	7
8.6	Ukázka composable funkce	7
8.7	Ukázka práce s Proto DataStore	9
8.8	Třída activity	9
8.9	Třída activity	10
	Komponenta pro seznam hlasování	10
		13
	Komponenta pro navigaci	13 14
	· · · · · · · · · · · · · · · · · · ·	
		16
	Ukázka datové vrstvy pro data o stavu aplikace	16
	Ukázka datového zdroje	
	Ukázka použití knihovny Retrofit pro získání seznamu hlasování z backendu	17
9.1	Ukázka kódu pro vytvoření endpointu	20
9.2	Ukázka nastavení hlaviček pro stránkování	21
9.3	Ukázka doménové vrstvy pro vrácení seznamu poslanců	21
9.4	Ukázka kódu pro sestavení objektu pro stránkování	22
9.5	Ukázka dopočtu statistik pro detail hlasování za běhu v doménové vrstvě	22
9.6	Ukázka použití streamu	22
9.7	Entita Vote reprezentující hlasování	23
9.8	Repozitář pro hlasování	23
9.9	Třída pro stahování zdrojových souborů	24
	Ukázka stahování dat pomocí knihovny commons-io	24
	Ukázka extrakce souborů ze zipu	25
	Skript pro odstranění duplicitních řádků	25
	Parsování datového souboru omluvy.unl	26
9.14	Parsování datového souboru omluvy.unl	26

9.15	Transformace objektu Omluva na databázový objekt Excuse	26
A.1	Tělo odpovědi pro dotaz GET /api/app	3
A.2	Tělo odpovědi pro dotaz GET /api/vote	3
A.3	Tělo odpovědi pro dotaz dGET /api/votei	3
A.4	Tělo odpovědi pro dotaz GET /api/party/vote/1	5
A.5	Tělo odpovědi pro dotaz "	6
A.6	Tělo odpovědi pro dotaz GET /api/member/1	7
A.7	Tělo odpovědi pro dotaz GET /api/member/1/vote	7

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.

	10.0	1/	_
Pr	Oh.	láše	mı
	OII.	last	

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 11. února 2023	
v Praze one 11. imora 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 naszení 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

 ${\bf REST} \quad {\bf Representational \ state \ transfer}$ 

API Application Programming Interface

Cíle

Prvním cílem práce je specifikace funkčních a nefunkčních požadavků, které jsou kladeny na mobilní aplikaci a backendu. Druhým cílem je analýza existujích 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.

2 Cíle

# 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

Poslanecká sněmovna

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

# 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 požada	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 požada	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 poža	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.

# Analýza existujících řešení

. . . . . . . . . . . . . . . . . . .

Tato kapitola se zabývá rešerší existujících řešení.

# 4.1 politiscope

Autor: Android Politiscope Developer

Počet stažení: více než 10 000

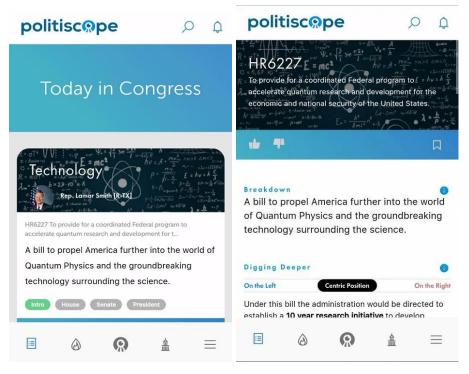
Analyzovaná verze: 2.4 (26. 1., 2023)

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 poskytuných z následujícíh 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)

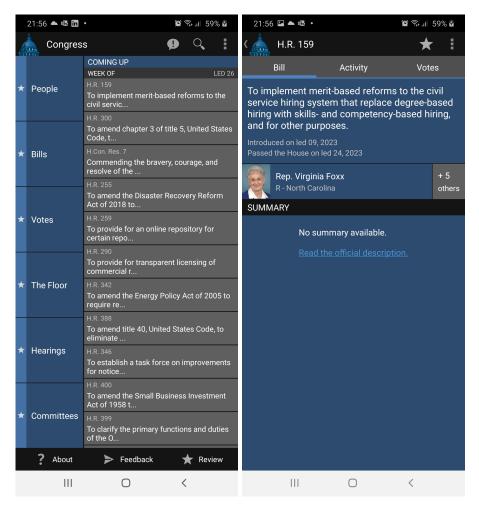
Aplikace Congress [5] dle popisu na Google Play poskytuje informace ohledně politických reprezentantů a jejich hlasováních, a návrzích zákona ve Spojených Státech. Návrhy a hlasování lze vyhledávat podle klíčových slov.

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.

Congress 3



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.

# 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		
Тур	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.

Tabulky 3

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	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	pec 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 zarazení 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	Tabulka zarazeni		
Sloupec	Тур	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.	

Tabulky 5

#### **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	Тур	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, kterí 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	Тур	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	

Tabulky 7

#### ■ 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. Podávámeli 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	Тур	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.	

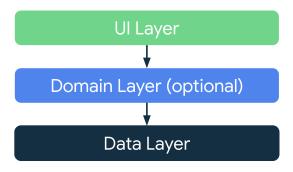
# Analýza softwarových architektur

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

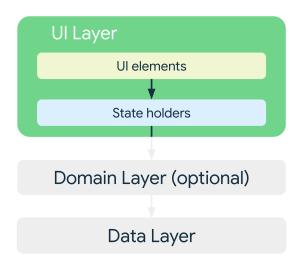
# 6.1.1 Architektura doporučená Googlem

Softwarovou architekturu doporučenou Googlem [9] ilustruje následující obrázek:



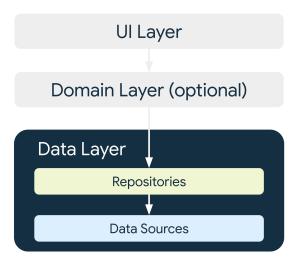
Obrázek 6.1 Architektura pro mobilní aplikaci pro Android [9]

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 vzdálěně) 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*.

Backend 3

### 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 držitěli 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.

2 Návrh

# Seznam poslanců

Obrazovka pro seznam poslanců (A.10) vypadá podobně jako obrazovka pro seznam hlasování. Obsahuje hlavičku s titulem, aktuálně nastaveným volebním obdobím, tlačítkém pro vyhledávání a tlačítkém 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 období. 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.

REST API 3

- **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).

4 Návrh

### 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

Databázový model 5

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).

6 Návrh

# 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 existujích 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 (8.1).

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 nedoběhla, 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 bežět 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 (8.3.2).

### **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

Android komponenta [17] je hlavním stavením kamenem aplikace, která má na starosti určitou funkcionalitu. V aplikaci se používá pouze komponenta Application. Pro jednoduchost textu budou považovány za komponenty i Activity a ViewModel, přestože podle dokumentace nejsou považovány za Android komponenty jako takové. V aplikaci jsou tedy použity následující komponenty:

- Application Reprezentuje aplikaci. Slouží pro inicilizaci aplikace a správu globálního stavu [17]. je inicializována před ostatními Android komponentami. Všechny ostatní Android komponenty existují v rámcí této. Pro použití vytvoříme podtřídu třídy Application a specifikujeme cestu k této podtřídě v rámci souboru AndroidManifest.xml.
- Activity Reprezentuje obrazovku a má na starosti správu UI a obsluhu událostí. Je to vstupní bod pro vytvoření UI. V aplikaci je použita pouze jedna aktivita a v rámci ní je dynamicky měněn obsah. Při navigace na jinou obrazovku se tedy nevytvoří nová aktivita, ale změní se pouze obsah existující aktivity. Pro použití této komponenty vytvoříme podtřídu třídy ComponentActivity a specifikujeme cestu k této podtřídě v rámci souboru AndroidManfiest.xml.
- ViewModel Reprezentuje držitele stavu (6.2), tj. definuje business logiku a drží v sobě stav pro UI elementy. Změna stavu vyvolá opětovné vykreslení UI. Pro použití této třídy vytvoříme podtřídu třídy ViewModel.

### Hilt

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

Výpis kódu 8.3 Příklad použití DI pomocí knihovny Hilt

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

Tento kód reprezentuje view model s jednou závislostí ve svém konstruktoru. Díky anotacím <code>@HiltViewModel</code> a <code>@Inject</code> z knihovny Hilt je uvedená závislost do tohoto objektu automaticky injektována Hiltem. Danou závislost nalezne Hilt ve svém interním kontejneru, který obsahuje všechny objekty a jejich závislostmi, které chceme, aby byly injektovatelné. Předpokladem pro DI je tedy existence tohoto interního kontejneru s grafem závislostí. Ten je nainicializován na základě konfiguračního souboru, v kterém specifikujeme objekty a jejich závislosti, které se mají do daného kontejneru přidat. Příkladem takové konfigurace je:

#### Výpis kódu 8.4 Ukázka konfigurace DI pro Hilt

```
// Soubor UseCaseModule.kt

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

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

    // dalsi konfigurace zavislosti
}
```

Konfigurace je specifikována v rámci singletonu. Anotace <code>@Module</code> říká, že daný objekt obsahuje konfigurace závislostí, která má Hilt přidat do svého grafu závislostí. Pomocí anotace <code>@InstallIn</code> specifikujeme životní cyklus pro dané závislosti. V tomto případě jsou závislosti vázané na životní cyklus view modelu. Pomocí anotace <code>Provides</code> specifikujeme již danou závislost. V tomto případě funkce vrací objekt typu <code>GetVotesUseCase</code>, a tím pádem je tento objekt Hiltem přidán do grafu závislostí a může být injektován do kteréhokoliv objektu. Všimněme si, že tento objekt má závislost na objektu typu <code>VoteRepository</code>. Tato závislost je také Hiltem injektována. Závislost je nakonfigurována podobným způsobem akorát v jiném souboru. Výhody DI jsou:

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

# Jetpack Compose

Jetpack Compose je Googlem doporučený způsob pro implementaci uživatelského rozhraní [18]. UI je implementováno pomocí programového kódu voláním *composable* funkcí, které reprezentují a na základě kterých se vykreslí nějaký UI element na obrazovce, ať už to je pouze tlačítko nebo celá obrazovka. Jetpack Compose poskytuje nativní composable funkce např. pro tlačítko, text a obrázek, ale také funkce reprezentující kontejnery pro seskupení UI elementů např. do řádku

nebo sloupce. Kombinací těchto funkcí lze vytvořit složitější funkce reprezentující složitější UI elementy. Jelikož UI elementy jsou reprezentovány funkcemi, lze je přepoužívat na více místech zavoláním dané funkce. Pomocí parametrů funkcí lze měnit vzhled a chování UI elementu. Lze mu např. změnit barvu, textový obsah, ale také přidat obsluhu pro události jako kliknutí na daný element a scrollování scrollovatelným kontejnerem. Jelikož je UI popisováno v programovým jazyce, lze využít jeho programové konstruktury jako cykly a podmínky. Zde je příklad composable funkce:

#### Výpis kódu 8.5 Ukázka composable funkce

```
// Soubor VoteListScreen.kt
@Composable
private fun Metadata(
  modifier: Modifier = Modifier,
  vote: Vote?,
  Column(modifier = modifier) {
    Description(vote?.description ?: "")
    Spacer(Modifier.height(15.dp))
    DateAndTime(vote?.date ?: "")
    // dalsi volani composable funkci
}
@Composable
fun Description(description: String) {
  Text(
    text = description,
    // dalsi atributy
}
```

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

#### Výpis kódu 8.6 Ukázka composable funkce

```
// Soubor VoteDetailsScreen.kt

@Composable
fun VoteDetailsScreen(
   viewModel: VoteDetailsViewModel = hiltViewModel(),
   // zbytek parametru
) {
```

```
// zbytek implementace

val voteDetailsUiState by
   viewModel.voteDetailsUiState.collectAsStateWithLifecycle()

Info(voteDetailsUiState = voteDetailsUiState)
}
```

Tento kód obsahuje composable funkci pro vykreslení obrazovky pro detail hlasování. Funkce obsahuje argument viewModel a proměnnou voteDetailsUiState. Nezávislě na tom, co tyto hodnoty přesně znamenají, dochází při jejich změně k rekompozici funkce. To znamená, že je funkce knihovnou Jetpack Compose znovu zavolána, ale tentokrát s novými daty. Důležitou vlastností composable funkcí je to, že k rekompozici dochází pouze ve všech funkcí v rámci podstromu funkce, kde došlo ke změně stavu. Další poznámkou je, že návratový typ funkce collectAsStateWithLifecycle(), která bude vysvětlena v popisu implementace prezentační vrstvy, je obecně State<T>, kde T je nějaký argument datového typu. Aby změna lokální proměnné vyvolala rekompozici funkce, musí být tohoto datového typu. Ke konkrétní hodnotě typu T se lze pak dostat prostřednictvím atributu value třídy State. Díky klíčovému slovu by má proměnná voteDetailsUiState datový typ T, a tudíž lze k ní přistupovat přímo. Mechanizmus rekompozice je přitom zachována. Výhody použití knihovny Jetpack Compose jsou tedy následující:

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

Alternativou ke knihovně Jetpack Compose pro vytváření UI jsou XML layouty, pomocí kterých lze vytvářet UI prostřednictvím XML tagů v XML souborech. Hlavní myšlenkou tohoto přístupu je to, že oddělujeme popis uživatelského rozhraní od programového kódu, díky čemuž je kód čitelnější a udržitelnější. XML soubory obsahují pouze popis elementů. Programový kód pracuje pouze s částmi aplikace mimo UI. Míchání popisu UI a programového kódu se však nikdy nevyhneme, jelikož layoutům se musí minimálně předat aspoň data. Toho lze dosáhnout pouze propojením programového kódu s layoutem. Dále např. implementace dynamického seznamu pomocí XML layoutu vyžaduje hodně boilerplate kódu [19]. Pro Jetpack Composu je vytvoření takového seznamu otázkou pár řádků kódu [20]. Z toho důvodu jsem se rozhodl implementovat UI pomocí knihovny Jetpack Compose.

#### Proto DataStore

V aplikaci je možnost nastavit si aktuální volební období. Toto nastavení by si měla aplikace pamatovat i po vypnutí a opětovném aplikace. Z toho důvodu je tento údaj ukládán lokálně v mobilním zařízení. Jsou dvě různé knihovny, které poskytují rozhraní pro ukládání do lokálního úložiště a čtení z něj: SharedPreferences [21] a DataStore [22]. Knihovna DataStore poskytuje asynchronní práci s lokálním úložištěm pomocí couroutinů, knihovna SharedPreferences pouze synchronní. Z toho důvodu byl vybrán DataStore. Knihovna DataStore poslytuje dvě různé implementace: Preferences DataStore a Proto DataStore. Implementace Proto DataStore

ukládá data jako instance vlastně definované třídy, která je vytvořena na základě konfiguračního souboru napsaný v jazyce používaného v Protocol Buffers. Díky implementace přes třídu je při ukládání a čtení dat poskytována typová kontrola. Implementace Preferences DataStore umožňuje přistupovat k datům pomocí klíčů, nevyžaduje předem definované schéma pro data a neposkytuje typovou kontrolu. Kvůli absenci typové kontroly byla zvolena implementace Proto DataStore, který funguje následovně:

#### Výpis kódu 8.7 Ukázka práce s Proto DataStore

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

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

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

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

Zpráva, jak ji nazývá Google, DataStore repezentuje strukturu lokálního úložiště. Na základě ní se při sestavení aplikace vygeneruje třídá UserPreferences. Třída DataStore poskytuje rozhraní pro práci s lokálním úložištěm. Proměnná userPreferencesDataStore.data je flow poskytující data z lokálního úložiště, tj. jakmile se data v úložišti změní, konzument dané flow na to zareaguje. Přes tuto proměnnou lze tedy získat aktuální volební rok. Pro aktualizaci volebního roku je pak použita funkce updateElectionYear.

# 8.2 Implementace prezentační vrstvy

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

# Vstupní bod

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

#### Výpis kódu 8.8 Třída activity

```
// Soubor MainActivity.kt
```

```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    // zbytek implementace
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        init()
        setContent {
            PspApp()
        }
    }
    // zbytek implementace
}
```

Anotace @AndroidEntryPoint je informace pro Hilt, aby dovoloval injetovat závislosti v rámci této třídy a všech komponent existujících v rámci této aktivity. Pomocí funkce setContent je zavolána kořenová composable funkce PspApp. V rámci funkce init je z backendu získán seznam volebních roků a uložen do lokálního úložiště:

#### ■ Výpis kódu 8.9 Třída activity

```
// Soubor MainActivity.kt
@Inject
lateinit var userPreferencesRepository: UserPreferencesRepository
@Inject
lateinit var pspApi: PspApi
private fun init() {
  initUserPreferences()
private fun initUserPreferences() {
  lifecycleScope.launch {
    userPreferencesRepository.userPreferencesFlow.collect {
      userPreferences ->
      if (userPreferences.electionYear == 0) {
        try {
          val appState = pspApi.getAppState()
          userPreferencesRepository
            .updateElectionYear(appState.electionYears.first())
        } catch (_: IOException) {
        }
      }
    }
  }
}
```

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

# UI elementy

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

- Column Sloupcový kontejner pro elementy. Hodí se pro pozicování UI elementů do sloupce. Nehodí se pro dlouhé seznamy, jelikož při rekompozici této komponenty dojde k rekompzici všech jejích podkompoennt, včetně těch které nejsou na obrazovce viditelné kvůli velikosti seznamu. Kontejnerem nejde scrollovat. Používám ji pozicování všech elementů, které se nachází pod sebou.
- LazyColumn Sloupcový kontejner pro elementy, který je zoptimalizovaný pro dlouhé seznamy. Při rekompozici komponenty dojde k rekompozici pouze viditelných podkomponent. Kontejnerem lze scrollovat. Používám ji pro dlouhé seznamy, tj. seznam hlasování, seznam poslanců, hlasování jednotlivých klubů a hlasování poslance.
- **Row** Řádkový kontejner pro elementy. Používám ji pro pozicování komponent vedle sebe.
- Text Textový element. Používám ji pro zobrazení textového obsahu.
- Image Obrázkový element. Obrázek lze získat pomocí URL adresy. Používám ho pro zobrazení profilových fotek poslanců a logy klubů.
- Icon Komponenta pro ikonku. Používám ji např. pro ikonky v hlavičce a v dolní liště.
- IconButton Tlačítko s ikonkou uprostřed. Používám ho pro vyhledávací tlačítko.
- OutlinedButton Tlačítko s kontrastem barev mezi pozadím a obvodem. Používám ho pro tlačíka v popupu na obrazovce nastavení.
- OutlinedTextField Textové pole s kontrastem barev mezi pozadím a obvodem. Používám ho pro vyhledávací pole.
- FloatingActionButton Plovoucí tlačítko. Používám ho pro skok na začátek dlouhých seznamu.
- Scaffold Pomocný kontejner pro seskupení komponent na obrazovce. Umožňuje jednoduchým způsobem přidat plovoucí tlačítko a automaticky ho napozicovat. Používám jeden sdílený všemi mezi obrazovkami. Každá obrazovka s dlouhým seznamem má taktéž tuto komponentu pro přidání plovoucího tlačítka.
- TopAppBar Horní lišta.
- Spacer Vytváří mezeru mezi dvěma UI elementy. zda je mezera horizontální nebo veritkální, specifikujeme skrz parametr.
- **Divider** Oddělovač. Dá se nastavit na vertikální nebo horizontální.
- CompositionLocalProvider Umožňuje přepsat hodnotu v kontextu, který obsahuje globální hodnoty dostupné v rámci určitého podstromu komponent. Používám ji pro lokální přepsání globální barvy, a tím nastylování textů pro data a časy.
- Box Kontejner umožňující skládat UI elementy na sebe. Používám ho pro vytvoření popupu v nastavení. Na pozadí je seznam nastavení a na něm se leží daný popup.
- SettingsMenuLink Composable funkce z knihovny Alorma pro snadné vytvoření nastavení v seznamu nastavení s již nějakým defaultním nastylováním.

- 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 seznamu hlasování.
- **TabRow** Lišta s taby. Používám ji na obrazovce pro detail hlasování a na obrazovce pro detail poslance.
- Tab Reprezentuje konkrétní obrazovku v rámci tabu.
- HorizontalPager Layout umožňující horizontálně scrollovat obsahem. Používám ho pro scrollování mezi taby.

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

- VoteListScreen.kt
- VoteScreen.kt
- MemberListScreen.kt
- MemberScreen.kt
- SettingsScreen.kt

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

#### Výpis kódu 8.10 Komponenta pro seznam hlasování

```
// Soubor VoteListScreen.kt.kt
@Composable
fun VoteListScreen(
  modifier: Modifier = Modifier,
  onVoteClick: (id: Int) -> Unit,
  viewModel: VoteListViewModel = hiltViewModel()
) {
  val fetchState by viewModel.fetchState
  val refreshing by viewModel.isRefreshing
  val pullRefreshState = rememberPullRefreshState(
    refreshing,
  { viewModel.refresh() }
  val votePagingItems = viewModel.votes.collectAsLazyPagingItems()
  val searchText by viewModel.searchText.collectAsStateWithLifecycle()
  val isSearchBarExpanded = viewModel.isSearchBarExpanded.value
  val electionYearRange by
    viewModel.electionYearRange.collectAsStateWithLifecycle("")
  Column(modifier = modifier) {
    // zbytek implementace
```

Na začátku funkce je definován stav:

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

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

# ■ Výpis kódu 8.11 Komponenta pro navigaci

```
// Soubor PspNavHost.kt
@Composable
fun PspNavHost(
  modifier: Modifier = Modifier,
  navController: NavHostController,
  onBackClick: () -> Unit
) {
  NavHost (
    modifier = modifier,
    navController = navController,
    startDestination = VotesDestination.route
    composable(route = VotesDestination.route) {
      VoteListScreen(
        onVoteClick = { id ->
          navController.navigateToVoteDetails(id)
      )
    }
    composable (
      route = VoteDetailDestination.routeWithArgs,
      arguments = VoteDetailDestination.arguments
    ) { navBackStackEntry ->
      val voteId = navBackStackEntry.arguments!!
        .getInt(VoteDetailDestination.voteIdArg)
      VoteScreen(
        voteId = voteId,
        onBackClick = onBackClick
      )
    }
```

```
}
}
```

Pomocí funkce NavHost vytváříme destinace pro navigaci. Každá destinace je vytvořena pomocí funkce composable, která v prvním parametru akceptuje název identifikující danou destinaci a v druhém callback funkci, která se zavolá, když dojde k navigaci k této destinaci. Samotnou navigaci má na starosti objekt NavHostController. Při navigaci na detail entity v seznamu specifikujeme navíc argumenty jak v definici destinace, tak i při navigaci.

## Držitelé stavů

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

#### Výpis kódu 8.12 Ukázka využití view modelu

```
// Soubor VoteListViewModel.kt
class VoteListViewModel(
): ListViewModel(...) {
  private val _votes: MutableStateFlow<PagingData<Vote>> =
    MutableStateFlow(PagingData.empty())
  val votes: StateFlow<PagingData<Vote>> = _votes
  init {
    viewModelScope.launch {
      currentElectionYear.combine(searchText, ::Pair)
      .collectLatest { pair ->
        getVotes(pair.first, pair.second)
        .cachedIn(viewModelScope)
        .collect { votes ->
          _votes.value = votes
      }
    }
  }
```

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

# 8.3 Implementace doménové vrstvy

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

- GetAppStateUseCase.kt
- GetMemberDetailUseCaseen.kt
- GetMembersUseCase.kt
- GetMemberVotesUseCase.kt
- GetPartiesUseCase.kt
- GetPartyVotesUseCase.kt
- GetVoteDetailUseCase.kt
- GetVotesUseCase.kt

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

Výpis kódu 8.13 Ukázka využití třídy doménové vrstvy pro získání stránkovaného seznamu hlasování

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

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

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

# 8.3.1 Implementace datové vrstvy

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

AppStateRepositoryImpl.kt

- MemberRepositoryImpl.kt
- PartyRepositoryImpl.kt.kt
- VoteRepositoryImpl.kt.kt

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

#### Výpis kódu 8.14 Ukázka datové vrstvy pro data o stavu aplikace

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

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

#### Výpis kódu 8.15 Ukázka datového zdroje

```
class PspRemoteDataSourceImpl @Inject constructor(
   private val pspApi: PspApi
) : PspRemoteDataSource {
   override suspend fun getAppState() =
       pspApi.getAppState()
   ...
}
```

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

- VotesPagingSource.kt
- MembersPagingSource.kt
- MemberVotesPagingSource.kt

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

# 8.3.2 Implementace síťové vrstvy

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

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

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

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

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

// zbytek dotazu
}
```

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

# Kapitola 9

# Implementace backendu

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

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

# Intellij IDEA

Backend byl vyvíjen ve vývojovém prostředí Intellij IDEA [23]. Výhody použití tohoto IDE jsou:

- Vestavěný inicializátor Spring Boot aplikací Pomocí tohoto inicializátor lze nakonfigurovat a nastavit potřebné závislosti ve Spring Boot¹ aplikaci jednoduše naklikáním v průvodci.
- Klávesové zkratky Toto IDE je vyvíjeno společností JetBrains, a tudíž obsahuje stejné klávesové zkratky jako Android Studio.

#### Maven

Maven je nástroj pro automatizaci sestavování programu. Původně byl použit Gradle kvůli čitelnější syntaxi. Poté se přešlo na Maven z historických důvodů. Původně bylo v plánu backend nasadit na Cloud Azure [24]. Ten poskytoval plugin pro maven, který umožňoval backend nasadit a zprovoznit pomocí jednoho příkazu. Avšak kvůli omezené možnosti využití výpočetního výkonu byl backend nakonec nasazen na jiný FIT cloud (více v kapitole o nasazení). Na funkčnosti aplikace to však nemá žádný vliv, a proto se už nepřešlo z Mavenu na Gradle.

# **Spring Boot**

Spring je open-source framework pro vývoj enterprise aplikací. Obshauje nástroje pro řešení různorodých problémů. Pro účely této práce byly využity nástroje pro vytvoření webových aplikací, které lze používat i pro implementaci REST API. Nástroje Springu jsou založeny na návrhovém vzoru dependency injection.

Spring Boot je framework, který je postavený na Springu a který má za cíl redukci boilerplate kódu a nutnost počáteční konfigurace. Toto realizuje prostřednictcvím autokonfigurace,

<sup>&</sup>lt;sup>1</sup>Technologie Spring Boot bude popsána později.

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

Alternativou ke knihovně Spring Boot je knihovna Ktor [25]. Výhodou této knihovny je, že vývoj v ní je určený pro psaní v Kotlinu. Lze tedy využít všechny výhody tohoto jazyka. Nevýhodou je, že nepodporuje defaultně dependency injection. Musí se tedy dodatečně nainstalovat a nakonfigurovat. Další nevýhodou je chybějící autokonfigurace. Mnoho závislostí se tedy musí manuálně nakonfigurovat.

#### Java

Java byla od začátku hlavním programovacím jazykem pro vývoj aplikací ve Spring Bootu. Od roku 2017 přišla integrace jazyka Kotlin do Spring Bootu [26] a v dokumentaci jsou ukázkové kódy psány jak v Javě tak i Kotlinu. Podle mého názoru má Java oproti Kotlinu větší podporu v komunitě, co se týče vývoje ve Spring Bootu. Nalezení řešení pro problém ve Spring Bootu s Javou bylo jednodušší než ve Spring Bootu s Kotlinem. Intellij DIEA však poskytuje nástroje pro automatickou trasnformaci souboru v jazyce Java do souboru v Kotlinu. Výsledný kód bylo však potřeba vždy pročistit, jelikož u všech proměnných vždy obsahoval datové typy, které lze v Kotlinu v některých případech pro čitelnost vynechat. Zároveň výstupní kód někdy nebyl kvůli drobnosti kompilovatelný. Z toho důvodu jsem se rozhodl pro použití Javy. Zdá se však, že podpora pro psaní aplikací Spring Boot pomocí Kotlinu je čím dál tím větší. Když bych měl možnost backend napsat znovu, pouvažoval bych znovu o použití Kotlinu, jelikož je to velmi dobrý programovací jazyk.

# 9.1 Prezentační vrstva

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

- VoteController
- PartyController
- MemberController
- AppStateController

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

Výpis kódu 9.1 Ukázka kódu pro vytvoření endpointu

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

Prezentační vrstva 21

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

Podle anotace <code>@RestController</code> Spring Boot pozná, že se jedná o controller a vytvoří implementaci endpointů vevnitř. Anotace <code>@GetMapping</code> definuje URL adresu endpointu. V parametru lze specifikovat URL id parametry pomocí <code>@PathVariable</code> a query parametry pomocí <code>@RequestParam</code>. Data získané z doménové vrstvy (service) jsou případně transformována (mapper). U stránkovaného obsahu jsou navíc přidány HTTP hlavičky pro informace o stránkování:

#### Výpis kódu 9.2 Ukázka nastavení hlaviček pro stránkování

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

   // zbytek implementace

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

   return responseHeaders;
}
```

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

## 9.1.1 Doménová vrstva

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

- MemberService
- PartyService
- VoteService

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

#### Výpis kódu 9.3 Ukázka doménové vrstvy pro vrácení seznamu poslanců

```
// Soubor MemberService.java
public Page<Member> getMembers(PagingParams pagingParams) {
   Pageable pageable = PageableGenerator.buildPageable(pagingParams);
   if (filterName == null) {
     return memberRepository
```

```
.findByElectionYear(electionYear, pageable);
} else {
   // kod pro filtrovani poslancu
}
```

Instance třídy Pageable je vytvořena následovně:

Výpis kódu 9.4 Ukázka kódu pro sestavení objektu pro stránkování

```
// Soubor PageableGenerator.java
public static Pageable buildPageable(...) {
   Pageable pageable;
   ...
   pageable = PageRequest.of(page, size, sort);
   // napr. page = 2, size = 20, sort = Sort.by("dateTime").descending())
   return pageable;
}
```

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

Výpis kódu 9.5 Ukázka dopočtu statistik pro detail hlasování za běhu v doménové vrstvě

```
// Soubor VoteService.java
public Vote getVote(int id) throws IOException {
    ...
    int excusedCount = ...
    int loggedOffCount = ...
    ...
    // nastaveni hodnot excusedCount a loggedOffCount
    // vraceni vysledku
}
```

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

Výpis kódu 9.6 Ukázka použití streamu

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

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

### 9.1.2 Databázová vrstva

Databázová vrstva je implementována pomocí repozitářů a databázových entit. Komunikace s databází je abstrahována pomocí objektově-relačního mapování, díky kterému jsme abstrahováni

Prezentační vrstva 23

od databázových tabulek a místo toho pracujeme s objekty (entitami). Pro tento účel používám knihovnu Hibernate [27]. Příkladem databázové entity je:

#### Výpis kódu 9.7 Entita Vote reprezentující hlasování

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

    @Id
    private int id;

    private LocalDateTime dateTime;

    // dalsi atributy
}
```

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

Pro implementaci repozitářů je použita knihiovna spring-data-jpa [28], která poskytuje rozhraní JpaRepository, který obsahuje základní metody pro manipulaci s danou entitou jako např. findAll() pro získání všech záznamů z tabulky nebo findById() pro získání záznamu s daným id. Výhodou této knihovny je možnost vytvoření vlastních metod, kterým se říká query metody. Implementace dotazu se vygeneruje na základě pojmenování query metody. Toto pojmenování se řídí podle určitých pravidel, které jsou popsány v dokumentaci [28]. Např. následující metoda vrací seznam hlasování v daném volebním období:

### Výpis kódu 9.8 Repozitář pro hlasování

```
// Soubor VoteRepository.java
@Repository
public interface VoteRepository extends JpaRepository < Vote, Integer > {
    ...
    List < Vote > findByElectionYear(int electionYear);
}
```

Zde je seznam souborů s repozitáři:

- VoteRepository.java
- MemberRepository.java
- MemberVoteRepository.java
- AgencyRepository.java
- ExcuseRepository.java
- MembershipRepository.java
- PartyRepository.java

# 9.2 Zpracování dat

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

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

# 9.2.1 Stahování zdrojových souborů

Stahování zdrojových souborů má na starosti třída následující třída:

Výpis kódu 9.9 Třída pro stahování zdrojových souborů

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

// dalsi metody
}
```

Pro stahování souborů je použita funkce copyURLToFile z knihovny commons-io [29]. Tato funkce stáhne soubor na dané URL do dané složky a s daným názvem:

Výpis kódu 9.10 Ukázka stahování dat pomocí knihovny commons-io

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

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

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

    FileUtils.copyURLToFile(downloadUrl, file);
    ...
}
```

Zpracování dat 25

# 9.2.2 Extrakce datových souborů

Pro extrakci souborů ze zip byla použita funkce funkce extractFile třídy ZipFile z knihovny zip4j [30]. Ta na základě názvu souboru, který se má vyextrahovat, vyextrahuje daný soubor do dané složky a soubor bude mít daný název:

#### Výpis kódu 9.11 Ukázka extrakce souborů ze zipu

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

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

### 9.2.3 Pročištění dat

Duplicitní záznamy jsou odstraněny pomocí skriptu napsaného v jazyce Bash:

#### Výpis kódu 9.12 Skript pro odstranění duplicitních řádků

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

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 rychlost Bashe pro tento účel a jednoduchá manipulace se soubory.

# 9.2.4 Parsování dat

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

#### Výpis kódu 9.13 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
}
```

Všechny parsery se nachází ve složce parsers. Pro parsování je použita knihovna [31]. Ta poskytuje třídu CsvToBeanBuilder, pomocí které lze parsovat CSV soubory a namapovat je na objekty vytvořené s anotacemi, jak popsáno výše. Dále poskytuje metody pro nastavení oddělovače, detekci nullové hodnoty a možnost ignorování uvozovek:

#### Výpis kódu 9.14 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();
}
```

### 9.2.5 Transformace a uložení dat

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

#### Výpis kódu 9.15 Transformace objektu Omluva na databázový objekt Excuse

```
// Soubor ExcuseLoader.java
public class ExcuseLoader extends BaseLoader {
```

Zpracování dat 27

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. 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.

## Testování

- 10.1 Mobilní aplikace
- 10.2 Backend

2 Testování

## Nasazení

- 11.1 Aplikace
- 11.2 Backend

2 Nasazení

# Kapitola 12 Spuštění

- 12.1 Aplikace
- 12.2 Backend

2 Spuštění

Závěr

2 Závěr

#### Příloha A

### Příloha

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

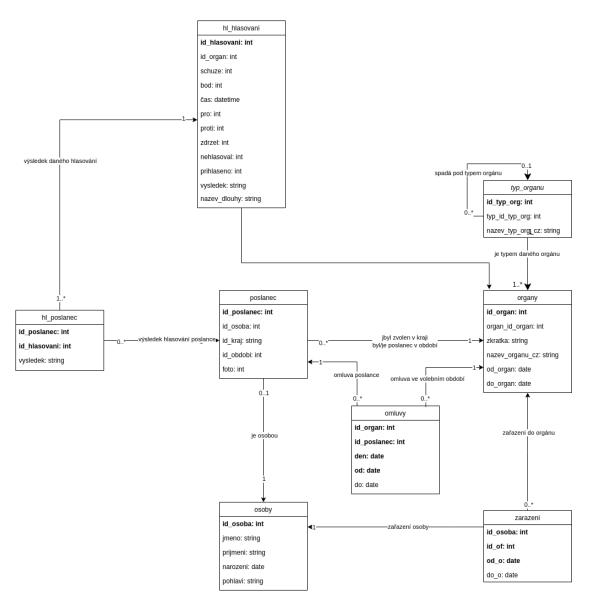
```
1
2
     "election_years": [
3
     2021,
     2017,
4
     2013,
     2010,
6
7
     2006,
     2002,
8
9
     1998,
     1996,
10
     1992
11
12
13
```

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

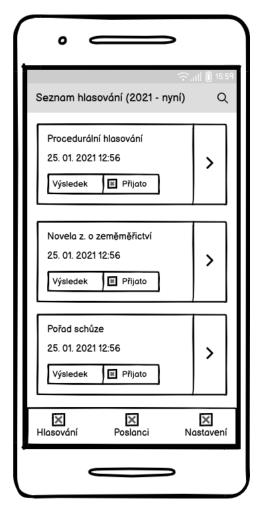
```
Е
1
2
       "id": 1,
3
       "date_time": "16. 12. 2022 13:29",
4
       "description": "Hlasovani 1",
5
6
       "result": "A"
7
8
       "id": 2,
9
       "date_time": "16. 12. 2022 13:26",
10
       "description": "Hlasovani 2",
11
       "result": "A"
12
13
14
```

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

```
1 {
2 "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",
3
    "description": "Hlasovani 1,
4
    "result": "A",
5
    "steno_protocol_url": "http://www.psp.cz/eknih/2021ps/stenprot/
6
        048 schuz/s048109.htm#h76",
    "yes_count": 100,
7
    "no_count": 0,
    "logged_off_count": 64,
    "excused_count": 0,
10
    "refrained_count": 36,
11
    "election_year": 0
12
13
```

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

```
1 [
2 {
```

```
"party_name": "Nazev klubu",
3
       "logo_url": "https://www.psp.cz/pics/klub/l-cps.jpg",
4
       "vote_id": 1,
5
       "party_results": {
6
         "yes_count": 2,
7
         "no_count": 0,
8
         "logged_off_count": 1,
9
         "excused_count": 0,
10
         "refrained_count": 0
11
12
       "member_results": [
13
14
           "member_name": "Poslanec 1",
15
           "vote_result": "@"
16
17
18
19
           "member_name": "Poslanec 2",
           "vote_result": "C"
20
21
22
           "member_name": "Poslanec 3",
23
           "vote result": "A"
24
25
26
           "member_name": "Poslanec 4",
           "vote_result": "A"
29
       ]
30
31
32
```

#### Výpis kódu A.5 Tělo odpovědi pro dotaz "

```
Е
1
2
       "id": 1,
3
       "name": "Poslanec 1",
4
       "party": "ANO",
5
       "photo_url": "https://www.psp.cz/eknih/cdrom/2021ps/eknih/202
6
          1ps/poslanci/i6474.jpg",
       "election_region": "Volebni kraj 1",
7
       "election_year": 2021
8
9
10
       "id": 2,
11
       "name": "Poslanec 2",
12
       "party": "ODS",
13
       "photo_url": "https://www.psp.cz/eknih/cdrom/2021ps/eknih/202
14
          1ps/poslanci/i6804.jpg",
       "election_region": "Volebni kraj 2",
       "election_year": 2021
16
17
18
```

#### Výpis kódu A.6 Tělo odpovědi pro dotaz GET /api/member/1

```
1
    "id": 1,
2
    "name": "Poslanec 1",
3
    "gender": "M",
4
    "party": "Poslanecky klub",
    "member_from": "12. 10. 2021",
    "member_to": null,
    "date_of_birth": "25. 09. 1970",
8
    "election_region": "Volebni kraj 1",
9
    "photo_url": "https://www.psp.cz/eknih/cdrom/2021ps/eknih/2021
10
        ps/poslanci/i6474.jpg",
    "election_year": 2021
11
12
```

#### Výpis kódu A.7 Tělo odpovědi pro dotaz GET /api/member/1/vote

```
1
2
       "vote": {
3
         "id": 1,
4
         "date_time": "16. 12. 2022 13:29",
5
         "description": "Hlasovani 1",
6
         "result": "A"
7
       "how_member_voted": "@"
9
10
11
       "vote": {
12
         "id": 2,
13
         "date_time": "16. 12. 2022 13:26",
14
         "description": "Hlasovani 2",
         "result": "A"
16
17
       "how_member_voted": "@"
18
19
20
```

#### ■ Tabulka A.1 Struktura agency

Název	Тур	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

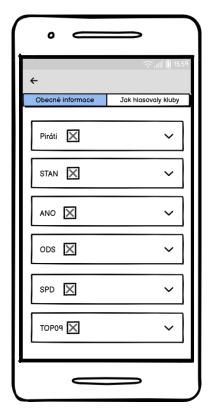
#### ■ Tabulka A.2 Struktura excuse

Název	Тур	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

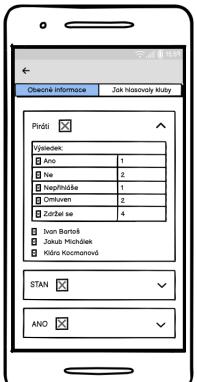
#### ■ Tabulka A.3 Struktura member

Název	Тур	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



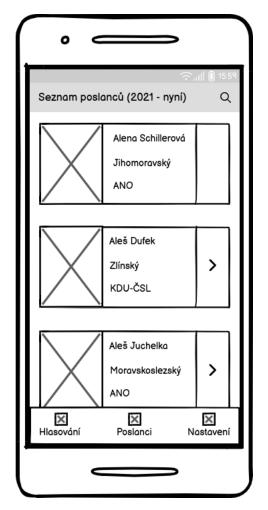


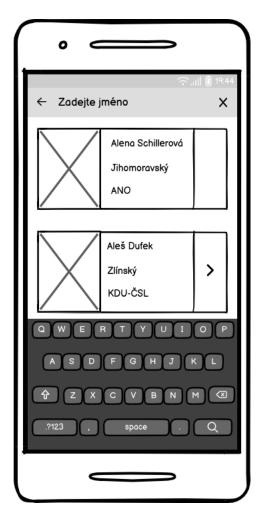
Obrázek A.5 Detail hlasování



Obrázek A.6 Jak hlasovaly kluby

- Obrázek A.7 Jak hlasovaly kluby
- Obrázek A.8 Obrazovky pro detail hlasování

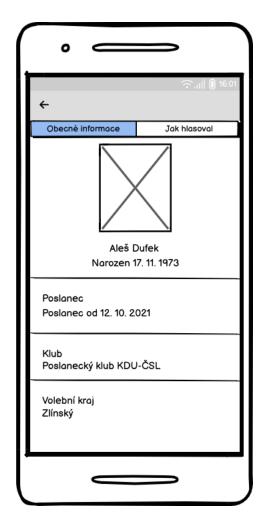


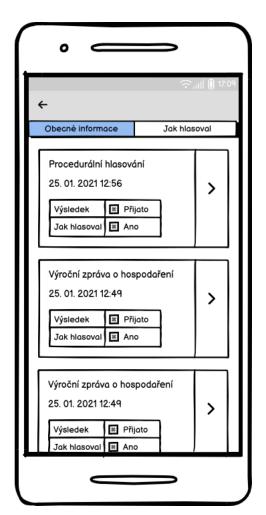


- Obrázek A.9 Seznam poslanců
- Obrázek A.10 Vyhledávání v seznamu poslanců
- Obrázek A.11 Obrazovka pro seznam poslanců

#### ■ Tabulka A.4 Struktura member\_vote

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

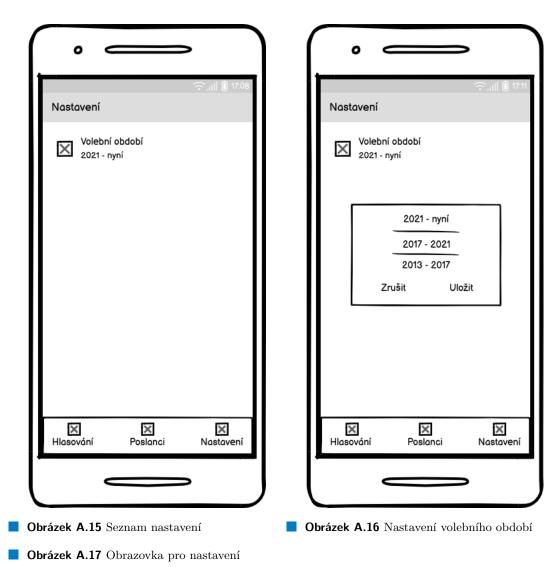




- Obrázek A.12 Detail poslance
- Obrázek A.13 Jak hlasoval/a poslanec/kyně
- Obrázek A.14 Obrazovky pro detail poslance

#### ■ Tabulka A.5 Struktura membership

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

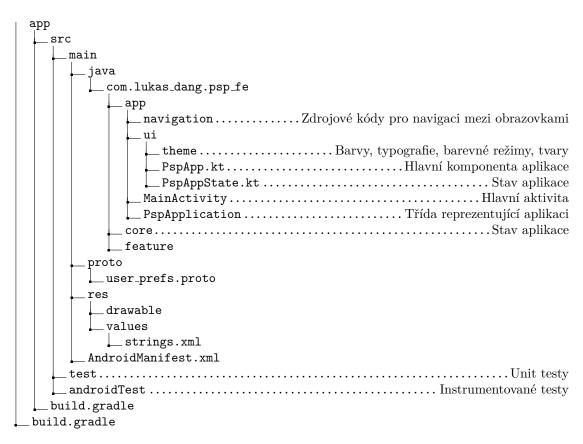


■ Tabulka A.6 Struktura party

Název	Тур	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	Тур	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 Bibliografie

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

- 17. GOOGLE. *Application* [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://developer.android.com/reference/android/app/Application.
- 18. GOOGLE. Build better apps faster with Jetpack Compose [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://developer.android.com/jetpack/compose.
- 19. GOOGLE. Create dynamic lists with RecyclerView [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://developer.android.com/develop/ui/views/layout/recyclerview.
- 20. GOOGLE. Lists and grids [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://developer.android.com/jetpack/compose/lists.
- 21. GOOGLE. Save key-value data [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://developer.android.com/training/data-storage/shared-preferences.
- 22. GOOGLE. Working with Proto DataStore [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://developer.android.com/topic/libraries/architecture/datastore.
- 23. JETBRAINS. Create dynamic lists with RecyclerView [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://www.jetbrains.com/idea/.
- 24. MICROSOFT. Azure [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://azure.microsoft.com/en-us.
- 25. JETBRAINS. Ktor [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://ktor.io/.
- 26. VMWARE, Inc. Introducing Kotlin support in Spring Framework 5.0 [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0.
- 27. HAT, Red. *Hibernate* [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://hibernate.org/.
- 28. VMWARE, Inc. Spring Data JPA Reference Documentation [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.
- 29. COMMONS, Apache. *Apache Commons IO* [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://commons.apache.org/proper/commons-io/.
- 30. LINGALA, Srikanth Reddy. zip4j [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://github.com/srikanth-lingala/zip4j.
- 31. SMITH, Glen. opencsv [online]. [B.r.]. [cit. 2023-02-10]. Dostupné z: https://opencsv.sourceforge.net/.

### Obsah přiloženého média

ı	readme.txt	stručný popis obsahu média
		adresář se spustitelnou formou implementace
	src	
	impl	zdrojové kódy implementace zdrojová forma práce ve formátu L <sup>A</sup> T <sub>E</sub> X
	thesis	zdrojová forma práce ve formátu I <sup>A</sup> T <sub>E</sub> X
		text práce
	thesis.pdf	text práce ve formátu PDF