Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

# BAKALÁŘSKÁ PRÁCE

Mobilní aplikace pro promotéry a jejich koordinátory

2019

Kamil Kotlář

Vedoucí práce: Mgr. Radek Janoštík

Studijní obor: Aplikovaná informatika, prezenční forma

**Bibliografické údaje**

| | |
|---|---|
| Autor: | Kamil Kotlář |
| Název práce: | Mobilní aplikace pro promotéry a jejich koordinátory |
| Typ práce: | bakalářská práce |
| Pracoviště: | Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci |
| Rok obhajoby: | 2019 |
| Studijní obor: | Aplikovaná informatika, prezenční forma |
| Vedoucí práce: | Mgr. Radek Janoštík |
| Počet stran: | 45 |
| Přílohy: | 1 CD/DVD |
| Jazyk práce: | český |

**Bibliograhic info**

| | |
|---|---|
| Author: | Kamil Kotlář |
| Title: | Mobile application for promoters and their coordinators |
| Thesis type: | bachelor thesis |
| Department: | Department of Computer Science, Faculty of Science, Palacký University Olomouc |
| Year of defense: | 2019 |
| Study field: | Applied Computer Science, full-time form |
| Supervisor: | Mgr. Radek Janoštík |
| Page count: | 45 |
| Supplements: | 1 CD/DVD |
| Thesis language: | Czech |

## Anotace

*V práci jsem prozkoumal současná řešení v oblasti sledování denních propagačních akcí a na základě zjištěných poznatků jsem se pokusil navrhnout a implementovat univerzální systém pro jejich správu. Snažil jsem se poučit z chyb stávajících řešení a uspokojit obecné požadavky reklamních agentur. Práce obsahuje aplikační a databázový server, web pro administraci a nativní Android aplikaci pro promotéry.*

## Synopsis

*In my thesis I explored the current solutions of tracking the daily promotions and tried to design and implement an universal system for their management based on the findings. I tried to learn from the mistakes of existing solutions and to satisfy the general requirements of advertising agencies. The thesis consists of application and database server, web for administration and native Android application for promoters.*

**Klíčová slova:** server; Android; web; marketing

**Keywords:** server; Android; web; marketing

Děkuji Mgr. Radkovi Janoštíkovi za příkladné vedení práce a ochotu pohotově odpovídat mým dotazům.

*Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.*

datum odevzdání práce podpis autora

# Obsah

# Seznam obrázků



# Seznam zdrojových kódů

# 1 Úvod

Nejlepším způsobem propagace produktu je přirozeně osobní kontakt. Toho jsou si vědomy i marketingové agentury, a proto ze strany velkých firem roste zájem o propagační akce. Agentury pro tyto firmy shání komunikativní studenty se zájmem o danou oblast prodeje a na oplátku studentům nabízí lukrativní brigádu na pozici promotéra.

Osobně jsem práci vykonával dva roky a za doby mého působení jako promotér jsem měl možnost detailně poznat fungování promo akcí a získat dostatek kontaktů pro posouzení kvality sledování denních propagačních akcí různých firem. Většinu používaných řešení pro správu proma jsem shledal jako nedostatečnou, a proto jsem se s nasbíranými zkušenostmi a poznatky rozhodl pro vývoj vlastního komplexního systému. Motivací pro jeho vytvoření mělo být především ulehčení záznamů prodejů pro promotéry.

Základní důraz při vývoji systému byl kladen na jeho univerzálnost tak, aby se dal využít v jakékoli marketingové agentuře zabývající se přímou podporou prodejů a neobsahoval žádný proprietární prvek specifický pouze pro jednu společnost. Cílem bylo vytvořit především silné jádro serveru, schopné rozšiřitelnosti a komunikace s libovolným klientem.

## 1.1 Základní pojmy

- ***Podpora prodejů*** neboli ***sales promotion*** je jednou ze čtyř částí marketingového mixu. Jedná se o techniku, která zákazníka různými nástroji stimuluje k uskutečnění nákupu (soutěže, hry, akce na místě prodeje, slevové kupony, prémie, vzorky, ...). [1]

- ***Shop-in-shop*** je mini-obchod uvnitř místa prodeje, který je tvořen výrobky a/nebo prezentačními prvky značky. Většinou je doplňují tématické promostolky, promopulty. [2]

- ***Partnerská prodejna*** je z pohledu promo agentury právě taková prodejna, ve které se nachází shop-in-shop a probíhá v ní promo.

- ***Propagační akce, promo akce*** či ***promo*** lze chápat jako formu osobní komunikace mezi firmou a potenciálním zákazníkem. Jejím principem je seznamovat zákazníka s výhodami nabízeného produktu a přimět zákazníka k jeho koupi. Místem konání promo akcí jsou shop-in-shopy v partnerských prodejnách. [1]

- ***Denní promo*** je typ promo akce, která se koná pravidelně v rozmezí jednoho až sedmi dnů v týdnu.

- ***Promo agentura*** je marketingová agentura, která pro svého zákazníka zajišťuje promo, zprostředkovává zpětnou vazbu na produkty a reportuje výsledky promo akcí.

## 1.2 Aktéři systému

- ***Promotér*** je člověk, jehož úlohou je zviditelnit určitý produkt, značku nebo službu a učinit je pro zákazníky atraktivnější a viditelnější. Má podpořit jejich prodej, pomoci utvářet dobré jméno společnosti a zvýšit obecné povědomí o samotném produktu, značce nebo službě. Kromě dobré znalosti promované věci musí mít promotér schopnost odhadnout povahu a momentální náladu člověka a tomu přizpůsobovat svůj projev. Během své směny odvádí úspěšné prodeje agentuře. Převážně jde o studenty, kteří si touto formou přivydělávají. [3]

- ***Koordinátor proma***, zkráceně ***koordinátor***, je osoba, která se stará o správný chod proma. Je přímým nadřízeným promotéra. Má za úkol přijímat nové promotéry, plánovat promotérům směny, kontrolovat aktivitu proma a v neposlední řadě řešit konflikty mezi promotéry a partnerskými prodejnami. V závislosti na přístupu agentury může vytvářet výsledné statistiky pro zákazníka a v ojedinělých případech může sám být také promotérem.

- ***Administrátor*** je uměle vytvořená role systému. V reálném světě se může jednat o některého z vedoucích marketingové agentury nebo tuto funkci může zastávat přímo koordinátor. Jeho úkolem je přidávat koordinátory, promotéry a další administrátory do systému. Jeho další práva jsou stejná jako u koordinátora.

# 2 Průzkum současných řešení vybraných firem

Před samotným návrhem systému jsem nejdříve provedl průzkum u čtyř agentur poskytujících denní promo. U dvou ze čtyř uvedených agentur jsem nějakou dobu pracoval, a to mi umožnilo poměrně přesně zjistit, jak promo v daných firmách funguje. U zbylých agentur poznatky poskytli jejich promotéři, které považuji jako důvěryhodný zdroj.

U každé společnosti je popsán způsob evidence docházky promotéra, zapisování a odvodu popsaných produktů a způsob, jakým se utváří směnový plán. Pod popisem se vždy nachází můj názor k danému řešení. Názor není kritikou, ale spíše uvědoměním si dobrých a špatných stránek daného řešení.

## 2.1 HP

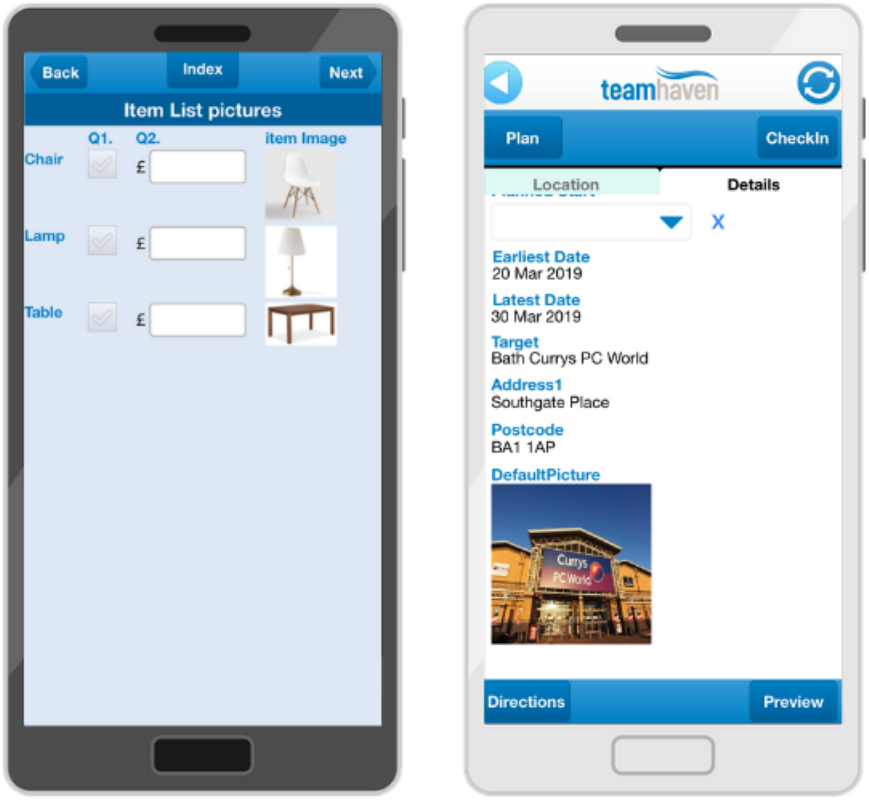
V průběhu mého průzkumu denní promo i jednorázové projekty pro společnost HP řešila česká marketingová agentura Eventuality, s.r.o..

Příchod na směnu byl řešen odesláním fotografie promotéra s shop-in-shopem HP v pozadí do patřičné skupiny v mobilní aplikaci WhatsApp. Ukončení proma probíhalo odesláním textové zprávy. Docházka se dále řešila papírově jako „karta promotéra“, která obsahovala datum a čas směny a razítko prodejny. Karta se na konci měsíce oskenovala a zaslala e-mailem koordinátorovi.

Prodeje se odváděli prostřednictvím vyplnění řádku Excelové šablony (tzv. ***product listu***) pro každou prodanou věc. Tabulka obsahovala tyto sloupce: země, partner, prodejna, jméno promotéra, datum, začátek a konec směny, počty konzultací, modelové označení, příslušenství, celý název produktu, kategorie a cena s DPH. Product list se odesílal koordinátorovi v polovině a na konci měsíce.

Tvorbu směn zastávaly tabulky, sdílené skrze cloudové úložiště. Ke konci měsíce koordinátor do cloudu vložil šablonu s prodejnami jako řádky a jednotlivými dny měsíce reprezentujícími sloupce. V každé buňkce se nacházel buď počet hodin směny (příznak existence směny v daný den) nebo nic (směna v daný den není k dispozici). Promotéři k buňkám s počty hodin připisovali svá jména. Tabulku mohl libovolně editovat kdokoli s přístupem k souboru. Editace se uzavírala počátkem daného měsíce, kterého se tabulka týkala.

Později tabulky doplnil systém TeamHaven (obrázek 1 [4]), který má HP licencovaný globálně, a který se prezentuje jako jednička mezi systémy pro proma. Do systému se promotéři přihlašovali skrze mobilní aplikaci, kde přihlašovací údaje byly tvořeny z názvu prodejny a města. Aplikace vyžadovala být neustále spuštěná, po vypnutí a opětovném zapnutí si nepamatovala, zda se promotér nacházel na směně. Pokud byla aplikace spuštěna na pozadí, za 6hodinovou směnu spolehlivě vybila plnou baterii mobilního zařízení. Evidence prodejů byla horší než vyplňování tabulky v Excelu, šlo totiž opět o vyplňování tabulky, jen na mobilu. Sloupce byly téměř totožné s tabulkou popsanou výše, řádky tvořily jednotlivé prodané produkty, u kterých byla předem navolena kategorie produktu. Přidat dva produkty z jedné kategorie bylo možné pouze tak, že se data v řádku




Obrázek 1: Ukázka aplikace TeamHaven

oddělily čárkou a cena se sečetla, aby prošla validátorem. Na konci směny se aplikace musela ručně synchronizovat se serverem.

### 2.1.1 Komentář

Řešení evidence docházky je nejednotné, vyžaduje kombinaci komunikátoru, elektronické pošty a fyzického papíru. Výhodou je možnost přidělit přístup do WhatsApp skupiny libovolné osobě jak z reklamní agentury, tak od promované značky. Kontrola docházky zde nemá moc velký smysl, WhatsApp umožňuje nahrávat i fotky uložené v albu, lze tedy nafotit několik fotek dopředu a poté je vydávat za právě vyfocené. Také je přísně kontrolován čas odchodu, ale textová zpráva neobsahuje žádnou informaci o poloze, pouze čas odeslání. Správný odchod promotéra má tedy nejspíše jistit „Karta promotéra“. Vzhledem k tomu, že se karta zpravidla nepodepisuje při odchodu, ale když si promotér zažádá (popřípadě sám orazí), tak nejde o velmi spolehlivou kontrolu.

Z mobilního telefonu je vkládání prodejů do tabulek nepohodlné, proto promotéři prodeje píší zpravidla „bokem“ a vkládají je do tabulky až po směně z PC. S tím souvisí i zapomínání si prodeje zapisovat a také nemožnost evidovat vytíženost prodejny dle jednotlivých hodin. Mimo to má každý řádek zbytečně spoustu duplicitních položek. Není přece potřeba u každé prodané věci evidovat jméno promotéra, zemi, počet oslovených zákazníků za den a další.

Plánování směn je neférové. Často dochází ke komplikacím, zejména kvůli neexistujícímu systému práv.

Co se týče systému TeamHaven, ten je z hlediska aplikace pro promotéry naprosto neintuitivní, koordinátor musí posílat promotérům rozsáhlý návod. V aplikaci je problém vůbec najít seznam směn, překlad do češtiny byl proveden strojově. Design aplikace je velmi zastaralý, nejsou použity žádné nativní komponenty, ovládací prvky nejsou rozmístěny v souladu s doporuČením UI, jeví prvky náhodného uspořádání. Vyplňování tabulky je velmi nepohodlné, na mobilní displej se vleze pouze její malá část. Při vkládání dat do tabulky není vidět záhlaví a skrze chybu dokonce ani text, který promotér zadává. Kvůli porušení atomicity dat musí být odvod prodejů nadále veden i formou Excelové tabulky, pouze se sníženou frekvencí odesílání na jedenkrát za měsíc.

## 2.2 Epson

Epson využíval služeb agentury PROMOTERI.EU, s.r.o..

Evidenci docházky řešila opět aplikace WhatsApp. V případě příchodu i odchodu se odesílala poloha. Namátkově bylo potřeba na konci směny odeslat fotku koordinátorovi. Koordinátor během dne náhodně objížděl prodejny a kontroloval, zda jsou promotéři v práci.

Prodeje se zapisovaly do tabulek Google. Stačilo zapsat modelové označení produktu spolu s cenou do patřičné kolonky ke svému jménu.

Promotéři ke konci měsíce odeslali, které dny jsou v následujícím měsíci k dispozici, koordinátor tyto informace zpracoval a vytvořil rozvrh směn, který zaslal nazpět e-mailem.

### 2.2.1 Komentář

Řešení docházky je sice jednoduché, ale neúčinné. Agentura bojuje se stejným problémem s aplikací WhatsApp, kdy WhatsApp umožňuje odeslat libovolnou polohu na mapě. Fotky lze samozřejmě opět mít v zásobě na následující směny. Za spolehlivou požaduji fyzickou kontrolu, která promotéry udržuje „ve střehu“. Je však náročná z hlediska finančních zdrojů agentury a času koordinátora.

Vkládání prodejů je jednoduché a umožňuje pohodlnější vkládání a editaci z mobilu. Vytvořit ale z těchto tabulek rozumné statistiky zabere agentuře mnoho práce.

Tvorba směn je řešena poměrně dobře. Pro koordinátory může být tvorba směn náročnější, a proto mohou být směny mezi promotéry rozloženy nerovnoměrně.

## 2.3 Lenovo

Kompletní reklamní služby pro Lenovo v době psaní práce obstarávala společnost Fox Hunter, s.r.o.

Po příchodu do práce se museli promotéři přihlásit do webové aplikace a přepnout svůj stav na „pracuji“. Žádná kotrola dle polohy se nekonala.

Webová aplikace, sloužící k ověření docházky, se používala i k odvodu prodejů. Po prodeji promotér vybral produkt ze seznamu, přidal cenu a stručné informace o zákazníkovi (muž/žena, věk, poznámky zákazníka).

Tvorba směn fungovala velmi podobně jako u Epsonu.

### 2.3.1 Komentář

Webová aplikace nijak nekontroluje polohu, není tedy problém si webovou aplikaci otevřít kdekoli a „pracovat“.

Odvod prodejů u Lenova je teoreticky ideálním řešením. V praxi už je to horší, spousta produktů v seznamu chybí, databáze produktů je většinou neudržovaná a neaktuální. To ale samozřejmě závisí na přístupu společnosti/agentury. Na systému oceňuji nutnost přidat informace o zákazníkovi. Metadata poskytnou Lenovu a agentuře nejlepší zpětnou vazbu (cílová skupina produktu, žádanost, atd.).

Škoda, že řešení není „vše v jednom“ a promotéři na jednom místě nenaleznou i nadcházející směny.

## 2.4 Acer

U Aceru veškerý marketing řešili přímo zaměstnanci firmy.

V prodejnách Alzy v Praze si kontrolu docházky řešili zaměstnanci Aceru v rámci kontroly celého stánku. V ostatních prodejnách se kontrola docházky příliš neřešila. Na konci měsíce poslal zaměstnanec Aceru e-mail s otázkou: „Které dny jste tedy byl v práci?“.

Z prodejen Alzy získával Acer informace o prodejích jako exportované tabulky z vlastního systému obchodu, protože promotér zde sám prodával produkty bez nutnosti obtěžovat zaměstnance prodejny. Na ostatních prodejnách odvod prodejů Acer neřešil (kvůli nedůvěře v poctivost promotérů).

Co se týká směn, koordinátor vždy zadal promotérům maximální počet hodin v měsíci, které si mohou odpracovat. Promotéři nazpět posílali své vytvořené rozvrhy směn ke schválení. Většinou již nedocházelo k úpravám. V prodejnách Alzy koordinátor vytvořil rozvrh směn, ten promotéři odsouhlasili, popřípadě si mezi sebou směny prohodili. Veškerá komunikace probíhala skrze e-mail.

### 2.4.1 Komentář

Je na uvážení, zda je občasná fyzické kontrola dostatečná. Mimo prodejny Alzy v Praze se samozřejmě jedná nejhorší (ne)řešení evidence docházky. Promotéra nic nemotivuje na prodejnu chodit.

V případě prodejen, do kterých dochází pouze jeden promotér, je tvorba směn pohodlná pro obě strany. Na Alze se promotéři musí přizpůsobit nebo si směny

mezi sebou vyměnit. Pokud však směny prohodit nelze, je to komplikace pro koordinátora. Pro daný den musí vyčlenit náhradníka, postupným kontaktováním jednotlivých promotérů.

Na prodejnách Alzy je řešení pro Acer i promotéry ideální. Je to bez práce a odvody jsou pravdivé. Mimo Alzu by samozřejmě Acer měl mít nějaký způsob evidence prodejů kvůli zpětné vazbě a motivačním provizím pro promotéry.

## 2.5 Závěr průzkumu

Průzkum mi umožnil přesně definovat činnosti promotéra i koordinátora. Také jsem na jeho základě mohl definovat společné prvky všech řešení a uvědomit si, které body je potřeba vylepšit nebo předělat.

### 2.5.1 Řešení vyvozené z průzkumu

Evidence směn by měla být realizována ověřením polohy na začátku i na konci směny v aplikaci, bez možnosti polohu modifikovat (popřípadě v systému zaznamenat, že poloha byla nesprávná). Server rovněž zaznamená do databáze skutečný čas příchodu a odchodu, kde budou tyto údaje k dispozici pro další zpracování.

V případě evidence prodejů bych rád vycházel z řešení Lenova. Klíčovým prvkem je rychlost odvodů, a proto by mělo být možné přidat nový prodej z jedné ze záložek úvodní obrazovky. Od promotéra se při odvodu očekává pouze zadání prvních písmen z produktového čísla/jména a následný výběr produktu z našeptávače. Rovněž bude nutné zadat cenu. Cena se na jednotlivých prodejnách liší a neustále se mění, proto by nebylo vhodné nutit koordinátora ceny neustále aktualizovat. Pokud odvod prodeje nebude pro promotéra dostatečně rychlý, bude promotér odeslání prodejů odkládat a agentura přijde o možnost sledovat prodeje podle denní doby.

Aplikace bude poskytovat možnost zápisu na směnu. Bez této inicializace systém nesmí povolit odvod prodejů. V souvislosti se zápisem na směnu by měl promotér mít k dispozici stručný přehled o nadcházejících směnách.

# 3 Použité technologie

## 3.1 Programovací jazyky

Při volbě programovacího jazyka pro tvorbu serveru i Android aplikace jsem nejprve uvažoval o Javě. Po menším průzkumu možností jsem se rozhodl pro Kotlin, zejména kvůli podobnosti s jazykem Java.

U webu jsem se nechtěl pouštět do experimentů, a tak zvítězila kombinace HTML, CSS, JavaScriptu a jQuery.

### 3.1.1 Kotlin

Kotlin je Turingovsky úplný, open-source a staticky typovaný programovací jazyk. V projektu jsem využil variantu, která běží nad JVM.

- ***JVM*** neboli ***Java Virtual Machine*** je sada počítačových programů a datových struktur, která využívá modul virtuálního stroje ke spuštění dalších počítačových programů a skriptů vytvořených v jazyce Java nebo v jazycích na Javě postavených (například Kotlin nebo Groovy). [5]

V Kotlinu nad JVM je možno nejen využívat knihovny napsané v jazyce Java, ale i v rámci jednoho projektu kombinovat oba tyto jazyky. Jazyk je oficiálně podporovaný pro tvorbu nativních Android i serverových aplikací. Kotlin je zaměřen na bezpečnost, přehlednost a pohodlnou práci s kolekcemi.

Datové typy jsou oproti Javě **`null`**-safety [6], což eliminuje nebezpečí v podobě `NullPointerException`. Pokud datový typ může nabývat hodnoty **`null`**, pak je za datovým typem symbol `?`. V jazyce Kotlin neexistuje ternární operátor tak, jak jej známe z jazyka Java či C. Naproti tomu **`if`** lze použít kdekoli v kódu, podobně jako to umožňuje LISP. V Kotlinu se často pracuje s Elvis operátorem (`?:`), který je binární. Pokud kód na pravé straně operátoru skončí jako **`null`** , provede se inline kód na pravé straně binárního operátoru (ukázka použití v kódu 1).

```
1 fun getJobs(city: City? = null, foo: Boolean): List<Job> {
2    val users = city?.let { getUsersByCity(city) } ?: getUsers()
3    return users.filter { it.eyes == EyesColor.GREEN }
4              .forEach { if (foo) it.bar += 1 else it.bar = 0 }
5              .mapNotNull { it.job }
6 }
```

Zdrojový kód 1: Práce s kolekcemi a null-safety v jazyce Kotlin

I přesto, že v Kotlinu není potřeba psát plno konstruktorů, getterů a setterů, není nijak ohrožena bezpečnost kódu. Více konstruktorů v jednom lze vytvořit za pomoci uvedení výchozích hodnot jednotlivých slotů objektu (příklad

```
data class ErrorOutDto(
      val errorCode: Int = 0,
      val message: String = "Unknown error"
)
```

Zdrojový kód 2: Hlavička Kotlin třídy s gettery i konstruktory

viz. zdrojový kód 2). Gettery a settery jsou generovány podle toho, zda je vlastnost označena `val` (pouze ke čtení, hodnota) nebo `var` (proměnná).

Kotlin je v současné době vyvíjen společností JetBrains. Je možno jej využít i ve variantě překládané do ECMAScript 5.1 či jako Kotlin Native (LLVM, v beta testování). [7]

### 3.1.2 HTML a CSS

Dvojice jazyků HTML a CSS byla prakticky jedinou volbou pro definici syntaxe, sémantiky a vzhledu webu.

### 3.1.3 JavaScript

JavaScript je programovací jazyk, který umožňuje implementaci komplexních částí do webových stránek. Je to pomyslná třetí vrstva ve standardních webových technologiích. Umožňuje vytvářet dynamicky aktualizovaný obsah, ovládání multimédií, animování obrázků a další. [8]

## 3.2 Knihovny a frameworky

### 3.2.1 Spring Framework

Spring Framework nabízí komplexní programovací a konfigurační model pro moderní enterprise aplikace založené na Javě.

Klíčovým prvkem Springu je infrastrukturní podpora na aplikační úrovni. Spring se zaměřuje na to, aby nebylo potřeba se vázat na konkrétní prostředí, ve kterém bude aplikace nasazena. Vývojáři se tak mohou lépe soustředit na samotnou logiku aplikací. [9]

#### 3.2.1.1 Spring Boot

usnadňuje vytváření aplikací v ekosystému Spring. [10]

- Umožňuje použití „starter“ závislostí, které zjednodušují konfiguraci výsledného sestavení.
- Pokud je to možné, automaticky konfiguruje knihovny Springu i třetích stran.
- Poskytuje funkce pro produkční nasazení jako jsou metriky, statistiky, kontroly stavu aplikace a externalizované konfigurace serveru.

- Přímo poskytuje podporu pro Tomcat, Jetty nebo Undertow.

**3.2.1.2 Spring Security** je výkonná a přizpůsobitelná knihovna pro autentizaci a kontrolu přístupu. Je standardem zabezpečení Spring aplikací. Poskytuje mimo jiné ochranu proti útokům typu session fixation, clickjacking, cross site request forgery a dalším. [11]

**3.2.1.3 Spring for Android** je REST klient pro Android navržený tak, aby poskytnul co nejlepší komunikaci se serverem vytvořeným použitím Spring Frameworku, včetně podpory autentizace. Ukázku základní práce s knihovnou naleznete v kódu 3. [12]

```
fun getPlainPasswords(): List<Book> {
   val url = "www.foo-bar.it/books"
   val resp = RestTemplate().getForObject(url, String::class.java)
   return if (resp.statusCode.is2xxSuccessful) {
      jacksonObjectMapper().readValue(resp.body)
   } else throw Exception("blah")
}
```

Zdrojový kód 3: Ukázka použití knihovny Spring for Android

### 3.2.2 Hibernate

Framework z dílen firem JBoss a Red Hat, který umožňuje objektově-relační mapování (ORM). Usnadňuje řešení otázky zachování dat objektů i po ukončení běhu aplikace. Ulehčuje programátorovi práci tím, že nemusí transformovat objekty do relací ručně, ale přenechá to perzistentní vrstvě. Je jednou z implementací Java Persistence API (JPA). [13]

Více o technologiích JPA a ORM v sekci 4.3.1.

### 3.2.3 Anko

ANdroid-KOtlin knihovna od firmy JetBrains, která umožňuje jednodušší a rychlejší vývoj Android aplikací. [14]

**3.2.3.1 Anko SQLite** poskytuje nástroje pro práci s SQLite databází v úložišti Android zařízení.

**3.2.3.2 Anko Commons** zprostředkovává rychlou tvorbu dialogů a menší drobnosti zjednodušující práci při vývoji Android aplikací.

**3.2.3.3 Anko Coroutines** výrazně zjednodušují práci s úlohami na pozadí. Jsou postaveny na Kotlin Coroutines. Jedná se o mocnou část Anko knihovny, díky které není třeba pro každou akci na pozadí vytvářet novou implementaci třídy `AsyncTask`. Jednoduchá práce s Coroutines je znázorněna v kódu 4.

```
1 fun someFunction() = doAsync {
2    val result = someLongAction()
3    uiThread { displayResult(result) }
4 }
```

Zdrojový kód 4: Ukázka použití Anko Coroutines

### 3.2.4 Bootstrap

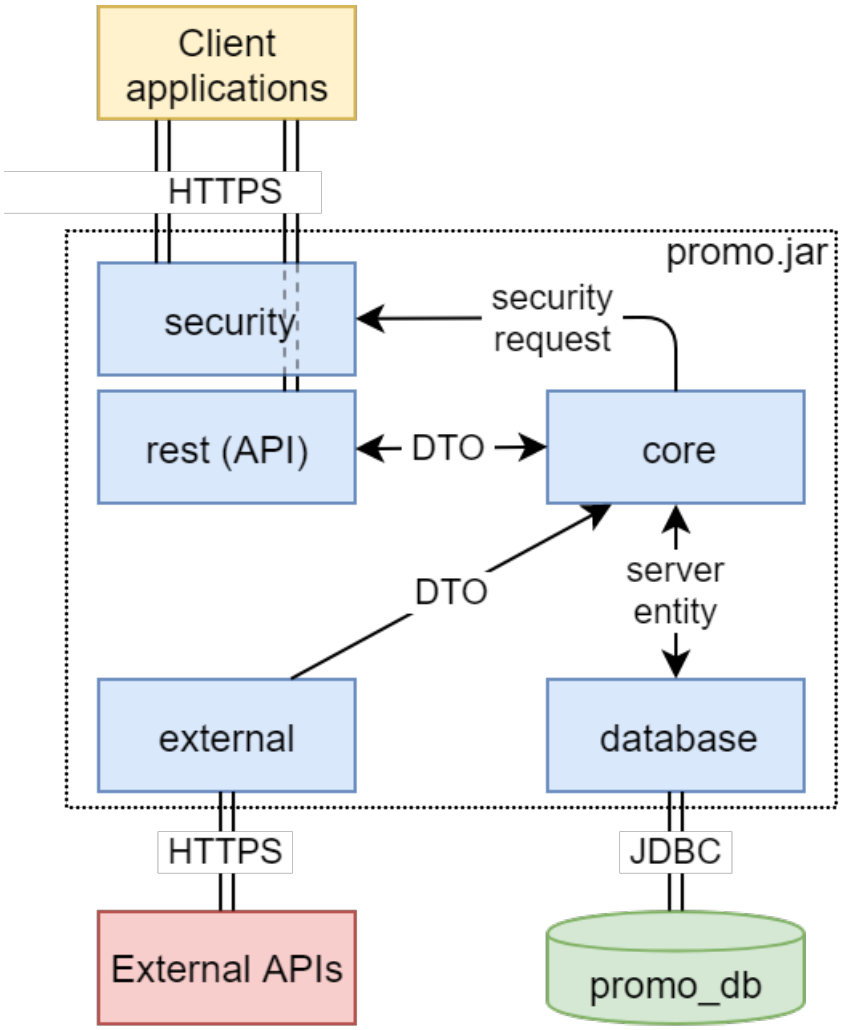
Bootstrap je populární framework pro rychlou tvorbu responzivních webů. Jedná se o open-source sadu nástrojů pro vývoj pomocí HTML, CSS a JavaScriptu. [15]

### 3.2.5 jQuery

Jedná se o malou, ale funkčně bohatou knihovnu JavaScriptu. Usnadňuje práci s událostmi, animacemi a balíkem technologií Ajax. [**?**]

# 4 Server

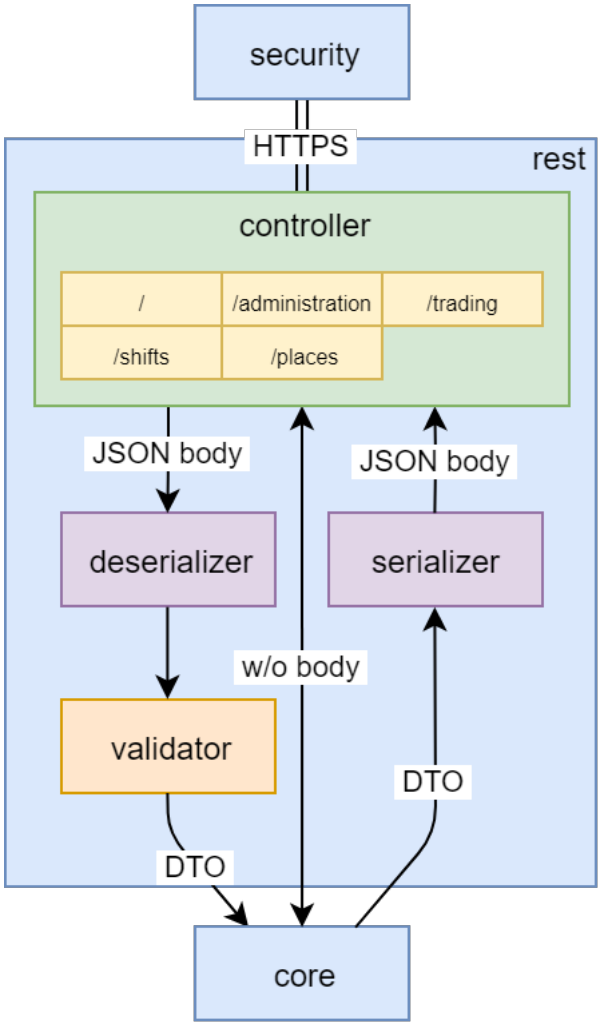
V této sekci bych se rád věnoval implementaci jednotlivých částí serveru. Server jsem rozdělil na pět základních částí: *rest*, *core*, *database*, *security* a *external*. V *jar* souboru je zdrojový kód rozdělen do stejnojmenných package (znázorňuje schéma 2). [16]




Obrázek 2: Server - architektura

## 4.1 Rest

Package rest obecně zprostředkovává komunikaci mezi klientem a serverem. Nachází se zde webové API. Architekturou webového API je REST. Obrázek 3 zobrazuje vnitřní logiku v package rest.




Obrázek 3: Rest - architektura

- ***API, Application Programming Interface*** neboli ***aplikační programové rozhraní*** slouží k předávání dat mezi softwarovými aplikacemi formalizovaným způsobem. [17]

- ***REST, REpresentational State Transfer*** je architektura rozhraní, navržená pro distribuované prostředí. REST navrhnul a popsal v roce 2000 Roy Fielding (jeden ze spoluautorů protokolu http) v rámci disertační práce Architectural Styles and the Design of Network-based Software Architectures. Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům. Zdrojem mohou být data, stejně jako stavy aplikace. Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim:

  1. ***GET*** umožňuje načíst jakékoli informace identifikované jednotným identifikátorem zdroje (URI). Jestliže se požadavek na zdroj vztahuje k získání dat, jsou to právě data, která jsou vrácena v odpovědi

ve formě entity.

2. ***POST*** se používá k vytvoření nového podzdroje ke zdroji identifikovaného pomocí URI. Skutečná funkcionalita prováděná metodou POST je určena serverem. Vykonáním POST metody nemusí vzniknout zdroj, který by byl identifikovatelný pomocí URI.
3. ***PUT*** slouží k nahrazení zdroje identifikovaného URI v adrese požadavku. Pokud URI neexistuje, může metoda PUT sloužit k jeho alokaci, podobně jako metoda POST. Tomu musí být přizpůsoben status kód, který server vrátí.
4. ***DELETE*** požaduje, aby server odstranil zdroj identifikovaný pomocí URI v požadavku. Klientovi nelze zaručit, že operace byla provedena.

Metoda GET se považuje za bezpečnou metodu, nedochází při ní k vytváření ani modifikaci zdrojů. Metoda POST je datově nebezpečná, při každém volání se vytváří nový zdroj, kdežto metody PUT a DELETE jsou idempotentní, a tedy libovolný počet opakovaných volání bude mít stále stejný výsledek. [18] [19]

### 4.1.1 API controllery

V package controller se nachází soubory **ApiController.kt*, které obsahují jednotlivé endpointy, ke kterým se klient může dovolat. Jednoduchý ApiController si ukážeme a popíšeme v kódu 5.

```
1  @RestController
2  @RequestMapping("/places", produces = "application/json")
3  class PlacesApiController(private val storesDao: StoresDao) {
4
5      @DeleteMapping(value = ["/stores"])
6      fun deleteStores(
7              @RequestParam(value = "id")
8              ids: Set<Long>
9      ) = storesDao.deleteStores(ids)
10
11     @PutMapping(value = ["/stores/{storeId}"])
12     @ResponseStatus(HttpStatus.NO_CONTENT)
13     fun updateStore(
14             @PathVariable(value = "storeId")
15             storeId: Long,
16
17             @RequestBody store: StoreInDto
18     ) = storesDao.updateStore(storeId, store)
19 }
```

Zdrojový kód 5: Příklad Spring REST controlleru

(1) Anotace, která Springu říká, že se jedná o controller a je typu REST. Pokud bychom používali Spring MVC, jako anotaci bychom uvedli pouze `@Controller`.

(2) Mapování controlleru na část URL adresy. Do třídy `PlacesApiController` tedy budou spadat všechny endpointy, které mají na prvním místě adresy hodnotu `"/places"`. Vlastnost `produces` určuje, co se vrací v tělech endpointů této třídy (v tomto konkrétním případě jde o JSON).

(3) V konstruktoru třídy `PlacesApiController` jde o tzv. ***Dependencies injection (vkládání závislostí)***. Jedná se o techniku pro vkládání závislostí mezi jednotlivými komponentami tak, aby jedna komponenta mohla používat druhou, aniž by na ni měla v době sestavování programu referenci. [20]

(5)-(6) Mapování endpointu na funkci. Z anotace můžeme vyčíst, že se jedná o metodu DELETE, která bude mapovaná na cestu `"/places/stores"`. Uvádí se relativní cesta. Anotace `@DeleteMapping` je pouze syntaktickým cukrem pro `@RequestMapping(method = RequestMethod.DELETE)`. Obdobně je možno použít anotace `@GetMapping`, `@PostMapping` a další.

(7)-(8) Parametry funkce je potřeba v controlleru označit anotací `@RequestParam` s názvem parametru v URL jako hodnotou. U parametru můžeme specifikovat např. zda je volitelný spolu s výchozí hodnotou, pokud klient tento parametr neuvede.

(9) Přes endpoint se volá funkce z core package, kde je umístěna logika pro manipulaci s objekty a databází.

(11) U tohoto mapování je důležité si povšimnout, že se část cesty (parametr `storeId`) nachází ve složených závorkách. Tento poznatek se váže k parametru funkce na řádcích (14)-(15).

(12) V anotaci `@ResponseStatus` je možné uvést status kód, který se bude vracet klientovi. Tento status kód bude použit pouze v případě, že při obsluze klienta nedojde k chybě. Ve výchozím stavu se jako kód vrací 200. V tomto případě je použitý kód 204, protože změna je provedena úspěšně a zároveň není potřeba vracet změněnou entitu.

(14)-(15) Zde se dostáváme k poznatku z řádku (11). Anotací `@PathVariable` udáváme parametr, který se vkládá do URL adresy a dotváří URI zdroje. Na místo {storeId} se dosadí ID objektu, se kterým budeme operovat.

(17) Jako druhý parametr funkce přijímáme tělo požadavku. Ten označíme anotací `@RequestBody`. Jako datový typ těla můžeme zvolit řetězec (a ten později parsovat na objekt) nebo rovnou zvolíme datový typ, který bychom očekávali po deserializaci JSONu.

(18) Získané parametry opět předáme funkci z core package.

### 4.1.2 Deserializace a validace vstupních dat

Pokud požadavek klienta obsahuje tělo (zpravidla JSON), je potřeba toto tělo deserializovat na objekt a vlastnosti objektu validovat. O to se v restu stará mapping package. Jako příklad uvedu `StoreInDto` (zdrojový kód 6). Jedná se o ***DTO (data transfer object)***, na který jsme narazili v ukázce 5. Struktura DTO objektu je identická s JSON objektem, který je od uživatele vyžadován.

```
1  data class StoreInDto(
2        @JsonDeserialize(using = GeneratedIdJsonDeserializer::class)
3        val chainStoreId: Long,
4
5        @JsonDeserialize(using = StringJsonDeserializer::class)
6        val houseNumber: String,
7
8        @JsonDeserialize(using = StringJsonDeserializer::class)
9        val street: String,
10
11       @JsonDeserialize(using = StringJsonDeserializer::class)
12       val zipCode: String
13 )
```

Zdrojový kód 6: Ukázka vstupního DTO objektu

(1) Jako typ objektu je zvolena **`data class`**. V jazyku Kotlin se jedná o typ třídy, který slouží pouze pro uchování dat. Pro takovou třídu se vygenerují metody `equals()`, `hashcode()`, `copy()` (pro vytvoření kopie objektu s možností jeho modifikace) a `toString()` (ve formátu **`"User(name=John, age=42)"`**) [21]

(2),(5),(8),(11) Nepovinná anotace, která umožňuje přepsat výchozí deserializer. Upravené deserializery jsem využil k validaci vstupu uživatele.

Pro úplnost uvedu ještě krátký deserializer v kódu 7. Ten přetěžuje metodu **`fun`** `deserialize(p: JsonParser, ctxt: DeserializationContext?): T` třídy `JsonDeserializer<T>`. Metoda ve svém těle volá statickou funkci `validate(str: String, fieldName: String?)` objektu `StringValidator`. Pokud při validaci nedojde k chybě, celý DTO objekt se předává ke zpracování příslušné třídě v coru.

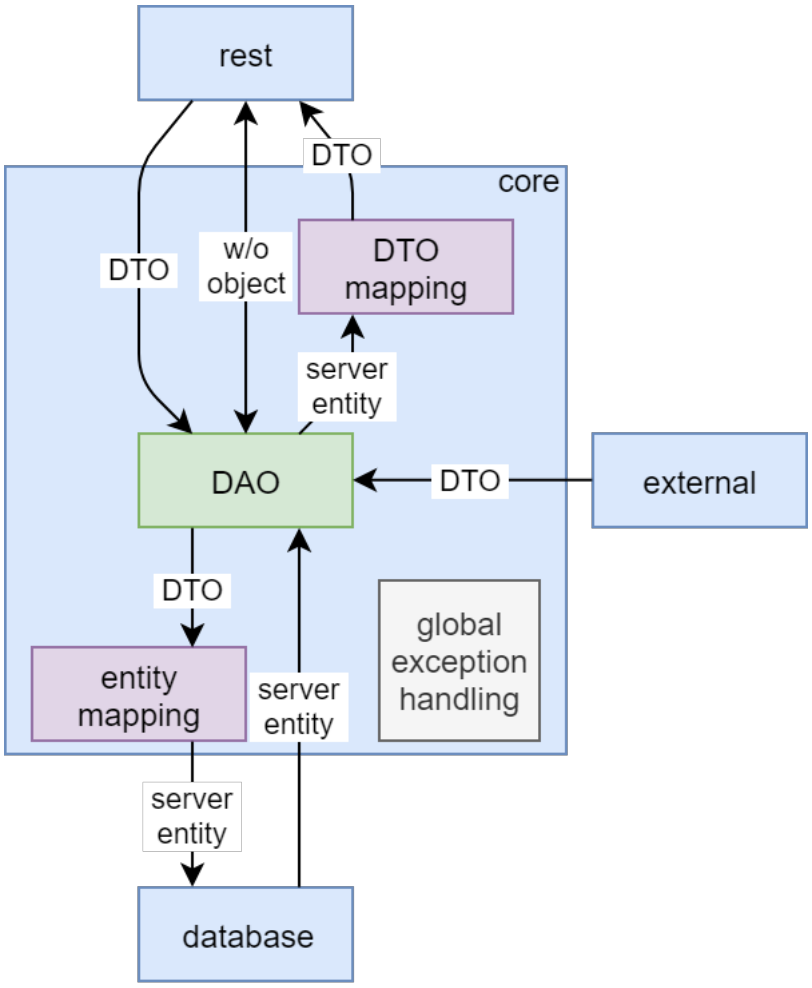
```
class StringJsonDeserializer : JsonDeserializer<String>() {

   override fun deserialize(
         p: JsonParser,
         ctxt: DeserializationContext?
   ) = StringValidator.validate(p.text.trim(), p.currentName)
}
```

Zdrojový kód 7: Deserializace a validace vstupu

## 4.2 Core

V package core se nachází veškerá mapovací a početní logika. Core přijme DTO z rest package, vytvoří entitu, kterou uloží do databáze, nebo načte entitu z databáze či externího zdroje a tu vrátí nazpět do package rest ve formě DTO. To je schématicky naznačeno na obrázku 4.




Obrázek 4: Core - architektura

### 4.2.1 Řízení toku dat

V package dao se nachází soubory **Dao.kt*, které řídí veškerý tok dat v core. Mimo to se v nich zpracovávají informace od uživatele, z databáze a z externích API. Tyto soubory jsou důležitou vrstvou mezi endpointy a databází. Není žádoucí do databáze zapisovat nezpracovaná data od uživatele, stejně tak by uživatel neměl mít k dispozici všechna data objektu, který se nachází v databázi. Implementace DAO komponenty je znázorněna ve zdrojovém kódu 8.

```
1  @Component
2  class CategoriesDao(private val categoriesRepo: CategoriesRepo) {
3
4     @Transactional
5     fun getCategory(
6           id: Long,
7           details: Boolean = false
8     ): CategoryOutDto {
9
10       val category = categoriesRepo.getByDeletedFalseAndId(id)
11       return if (!details) {
12          category.toDto()
13       } else {
14          category.toDetailsDto()
15       }
16    }
17 }
```

Zdrojový kód 8: Příklad DAO komponenty

(1) Anotace `@Component` zajišťuje, aby byla třída nalezena frameworkem Spring při inicializaci serveru.

(2) Podobně jako u rest controlleru i zde využijeme ***Dependency injection*** a do konstruktoru vložíme příslušnou třídu z package database.

(4) Funkci je nutné anotovat jako `@Transactional`. Důvod je uveden u popisu řádku (14).

(5)-(7) Funkce, která vrací kategorii bere jako identifikátor `id` a parametr `details`. V případě, že je parametr `details` roven **`true`**, bude výstupní DTO obsahovat i jednotlivé produkty, které daná kategorie obsahuje. Výchozí hodnotu argumentu je možné v jazyce Kotlin uvést ihned k argumentu. Vnitřně tak dochází k ***parametrickému polymorfismu***. Jestliže má argument uvedenou výchozí hodnotu, tento parametr při volání funkce nemusí být specifikován.

(8) Jako návratová hodnota je uvedena `CategoryOutDto`. Jedná se o DTO objekt, který vznikl mapováním z objektu `Category`.

(10) Z package database získáme kategorii dle jejího identifikátoru. Pokud kategorie neexistuje nebo má příznak `deleted`, klientovi je vrácena informace o neexistujícím zdroji.

(11) V závislosti na parametru `details` se objekt `Category` dále převede na požadovaný DTO.

(12) Objekt se dále mapuje na `CategoryOutDto`.

(14) V případě mapování na DTO s detaily, je spolu s kategorií vrácena i kolekce produktů, které do kategorie patří. Vzhledem k tomu, že ne vždy je potřeba spolu s kategorií z databáze vrátit i všechny produkty, je zde využito principu líného vyhodnocování. Z toho důvodu je funkce anotována jako `@Transactional`, protože se opět přistupuje k databázi a bez anotace by došlo k chybě.

### 4.2.2 Vytváření DTO objektů

Mapování výsledných entit, které core pro rest zpracoval, se provádí v mapping package. Veškerá mapovací logika je umístěna v souboru `DtoMapping.kt`. Zde jsem využil ***Kotlin Extensions***. Ty umožňují libovolné třídě (i z jazyka Java) přidat metody bez nutnosti vytvářet potomka dané třídy. To se hodí zejména pokud je nutné přidat funkcionalitu třídě, která se nenachází v aktuálním projektu, ale je součástí externí knihovny nebo frameworku. V tomto případě jsem chtěl docílit toho, aby se všechno mapování na objekty DTO nacházelo na jednom místě.

```
1  fun Category.toDto() = CategoryOutDto(
2        id = this.id,
3        name = this.name
4  )
5
6  fun Category.toDetailsDto() = CategoryOutDto(
7        id = this.id,
8        name = this.name,
9        products = this.products.mapNotNull {
10          if (!it.deleted) it.toDto() else null
11       }
12 )
```

Zdrojový kód 9: Ukázka mapování

Ukázka 9 znázorňuje mapování objektu `Category` na `CategoryOutDto` (DTO ve zdrojovém kódu 10). V závislosti na zvolené metodě mapování jsou mapovány do DTO i jednotlivé produkty.

```
1  @JsonInclude(JsonInclude.Include.NON_DEFAULT)
2  data class CategoryOutDto(
3        val id: Long,
4        val name: String,
5        val products: List<ProductOutDto> = listOf()
6  ) {
7
8     @JsonInclude(JsonInclude.Include.NON_NULL)
9     data class ProductOutDto(
10          val partNumber: String,
11          val name: String? = null
12    )
13 }
```

Zdrojový kód 10: Ukázka výstupního DTO objektu

(1) V případě, že kategorie neobsahuje žádné produkty, není ve výsledném JSON objektu parametr `products` vůbec přítomen.

(8) Obdobně jako na řádku (1) bude JSON díky `@JsonInclude` anotaci obsahovat pouze ty položky, které nebudou **`null`**.

### 4.2.3 Systém obsluhy výjimek

Důležitou součástí package core je globální systém obsluhy výjimek. Každá výjimka, která je na serveru vyvolána, se řeší ve třídě `GlobalExceptionHandler`. Ve třídě se při zachycení výjimky vytvoří objekt `ErrorOutDto`, který obsahuje vlastní chybový kód serveru (z objektu `PPErrorCodes`) a detailní chybovou hlášku v anglickém jazyce. Výsledný objekt je vrácen spolu s HTTP status kódem kategorie 4xx nebo 5xx. Pokud handler nenalezne metodu pro obsluhu výjimky, bude obsloužena metodou pro obsluhu generické výjimky. Díky tomutu systému je vrácení chyby *„500 - Internal server error“* nejhorší možná situace, do které se server může dostat.

### 4.2.4 Služba elektronické pošty

Poslední drobností, kterou core obsahuje, je třída `PasswordEmailService`. Komponenta slouží k zasílání registračních e-mailů a e-mailů potřebných k obnovení hesla. Podrobnosti k tomu, jak správa účtu funguje, naleznete v sekci 4.4.2.

## 4.3 Database

Tato část serveru slouží ve spolupráci s databázovým systémem k perzistenci dat a zprostředkování přístupu k nim. Jako databázový systém jsem zvolil MySQL, ale díky použitým technologiím může být bez nutnosti výrazných změn nahrazen libovolným jiným databázovým systémem (stačí změnit ovladač JDBC).

### 4.3.1 Objektově-relační mapování

- ***Objektově-relační mapování***, ***ORM*** je programovací technika v softwarovém inženýrství, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. Hlavním cílem ORM je synchronizace mezi používanými objekty v aplikaci a jejich reprezentací v databázovém systému tak, aby byla zajištěna persistence dat. [22]

- ***JPA*** neboli ***Java Persistence API*** je standard programovacího jazyka Java, který umožňuje objektově-relační mapování. Usnadňuje práci s ukládáním objektů do databáze a naopak. [23]

- ***Entita*** je objekt, který reprezentuje data v databázi. Typicky entitní třída reprezentuje tabulku v relační databázi a každá instance této třídy pak koresponduje s jedním řádkem tabulky. [24]

K obsluze databázového systému byl využit framework Hibernate, který je implementací standardu JPA.

Na příkladu 11 si ukážeme, jak vypadá entita vytvořená v jazyce Kotlin s pomocí `javax.persistance` package.

```
1  @Entity
2  data class Category(
3        @Column(unique = true, nullable = false)
4        var name: String
5  ) {
6
7     @Id
8     @GeneratedValue
9     @Column(updatable = false)
10    val id: Long = 0
11
12    @OneToMany(mappedBy = "category")
13    val products: List<Product> = listOf()
14
15    @Column(nullable = false)
16    var deleted = false
17 }
```

Zdrojový kód 11: Příklad databázové entity

(1) Anotace, která určuje, že jde o databázovou entitu.

(3),(9),(15) V anotaci `@Column` můžeme podrobněji specifikovat vlastnosti sloupce v tabulce.

(7)-(8) Jednoduše určíme ID, které bude automaticky generované. V databázi se vytvoří tabulka `hibernate_sequence`, která uchovává poslední použitá ID. ID se inkrementuje pro každou nově vytvořenou instanci entity.

(12) `@OneToMany` je protějškem k anotaci reprezentující vztah `@ManyToOne`. Parametr `mappedBy` odkazuje na vlastnost `category` v entitě `Product` a umožňuje jednoduše z kategorie přistoupit k seznamu produktů, které kategorie obsahuje. Ve skutečnosti se v databázi nachází pouze sloupec `category`, který je umístěn v tabulce `Product`. Tento zápis tedy neporušuje atomicitu dat. V případě, že bychom použili `@ManyToMany`, došlo by k vytvoření spojovací tabulky mezi dvěma entitami.

### 4.3.2 Práce s daty v databázi

Pro vytvoření dané tabulky v databázi a práci s ní je potřeba ještě vytvořit rozhraní. Rozhraní označíme `@Repository` a podědíme jej z generické třídy `JpaRepository<T, S>`, kde `T` určuje entitu a `S` je primárním klíčem dané entity. Na příkladu 12 si ukážeme jeden repozitář spolu s ekvivalentní metodou v jazyce SQL.

```
1 @Repository
2 interface CategoriesRepo : JpaRepository<Category, Long> {
3
4    fun getAllByIdIn(ids: Set<Long>): List<Category>
5 }
```

Zdrojový kód 12: Ukázka rozhraní JpaRepository

Deklarace funkce `getAllByIdIn(ids: Set<Long>): List<Category>` odpovídá následujícímu zápisu v jazyku SQL:

```
1 SELECT * FROM category
2 WHERE id IN (id1, id2, ...);
```

## 4.4 Security

Tento package je velmi důležitou částí serveru. Významně se podílí na bezpečnosti dat. Využívá knihovnu Spring Security, která je doplňkem pro framework Spring. Než se přistoupí k samotné obsluze požadavku, dojde k ověření totožnosti uživatele a v závislosti na výsledku je uživatelův požadavek vyřízen kladně či záporně.

### 4.4.1 Konfigurace

Spíše než o programování je práce s knihovnou Spring Security o správné konfiguraci. Hlavní konfiguraci nalezneme ve třídě `WebSecurityConfig`. Přepsáním metod `fun configure(*)` třídy `WebSecurityConfigurerAdapter` můžeme zvolit vlastní pravidla pro autentizaci uživatelů, šifrování hesel a určit přístupy k jednotlivým endpointům prostřednictvím rolí.

### 4.4.2 Správa účtů

Práva pro vytvoření nových účtů má přidělena pouze admin. Pokud by bylo žádoucí, aby toto bylo umožněno i koordinátorovi, je to otázka změny jednoho řádku v konfiguračním souboru. Registrace nového uživatele probíhá následujícím způsobem:

1. Uživatel s příslušnými právy zavolá *POST /administration/users* spolu s tělem, obsahujícím základní informace o uživateli.

2. Pokud vytvoření nových uživatelů proběhne úspěšně, je vygenerován bezpečnostní token, který je uložen do databáze. Pro token jsem zvolil platnost v délce jednoho měsíce.

3. Uživateli přijde registrační e-mail, který obsahuje odkaz spolu s příslušným tokenem. Odkaz je jednorázový a po jeho odkliknutí dojde k aktivaci účtu a vygenerování bezpečného hesla. Uživatel najde heslo v odkazu.

Možnost obnovy zapomenutého hesla pracuje velmi obdobně jako registrace. Po zavolání *POST /administration/users/{email}/obtain-password* dojde opět k vytvoření tokenu a jeho odeslání na příslušný e-mail. V tomto případě má token platnost 60 minut od jeho vygenerování. Dále uživatel postupuje jako u registrace. Z důvodu, že je endpoint přístupný bez autentifikace, bylo potřeba udělat bezpečnostní opatření, aby nedošlo k jeho zneužití. O to se stará objekt *ResetPasswordTimer*. Objekt si udržuje mapu, kde klíčem je e-mailová adresa a hodnotou je čas posledního úspěšného požadavku o změnu hesla. Při požadavku o změnu hesla nejdříve objekt zkontroluje, zda tento požadavek již nebyl v posledních 5ti minutách vykonán. Pokud ano, server vrátí zápornou odpověď.

### 4.4.3 Autentifikace uživatele

Uživatel se přihlašuje a odhlašuje skrze dvojici endpointů *POST /login* a *POST /logout*. Implementace je ponechána výchozí, z knihovny Spring Security. Jedinou nevýhodou je použití *application/x-www-form-urlencoded* namísto *application/json* v těle požadavku. Lehce je upravena pouze odpověď, kterou */login* endpoint vrací. Sezení uživatele se udržuje za pomoci JSESSIONID, která se uloží na straně klienta a stará se o to, aby se uživatel nemusel opakovaně prokazovat přihlašovacími údaji.

### 4.4.4 Bezpečnost hesel

Pro šifrování hesel byla využita třída `BCryptPasswordEncoder` z knihovny Spring Security. Ta jako hash využívá BCrypt, který je pro hesla vhodnější než hashe z rodiny SHA*. Důvodem pro použití hashe BCrypt je paradoxně jeho pomalá „rychlost“. Ta znemožňuje útočníkovi generovat kvanta hesel v krátkém čase. Tím se významně snižuje šance na prolomení hesla hrubou silou. [25]

Výpočet hashe trvá přibližně 200 milisekund, což je zároveň akceptovatelná čekací doba pro uživatele.

Při první inicializaci serveru se navíc vygeneruje náhodná sůl, kterou nezná ani programátor. To umocňuje bezpečnost hesel.

### 4.4.5 Oprávnění přístupu mimo security package

Vzhledem k tomu, že oprávnění k přístupu lze vázat pouze na jednotlivé endpointy, je v některých situacích potřeba řešit oprávnění přístupu i mimo package security. Jedná se zejména o situace, kdy endpoint je sice odemčen pro více rolí, ale pouze uživatel nemůže využívat určité parametry. O bezpečnost mimo security package se stará komponenta `SecurityManager`. Ta jednoduše vytahuje informace o uživateli aktuálního sezení a určuje, zda je uživatel oprávněn k dané činnosti.

## 4.5 External

Balík external je místo, kde se server dostává do role klienta. Jako klient komunikuje s API externích systémů pomocí protokolu HTTPS. Výsledky požadavků se posílají do coru, který záznamy cachuje do databáze, aby se omezil počet volání a zlepšila odezva systému.

Každý klient je implementací abstraktní třídy `AbstractClient`. Komunikace se servery je implementována s pomocí `java.net` package z Javy 11.

### 4.5.1 REST Countries v2

API REST Countries poskytuje informace o zemích celého světa. Stačí odeslat dvou nebo tří místný kód země a vybrat ta pole, která od API potřebujeme. REST Countries používám k získání názvu země a kódu hlavní místní měny.

### 4.5.2 Nominatim

Nominatim API má k dispozici informace o lokaci na základě ulice a PSČ. Poskytuje název města, kód země a GPS souřadnice daného místa. Ulice a PSČ jsou tedy jediné dva údaje, které se od uživatele vyžadují. Následně Nominatim spolu s REST Clientem poskytnou všechny informace, které server potřebuje. Měna se využívá k odvodu prodejů, souřadnice zase ke kontrole polohy.

## 4.6 Ostatní

### 4.6.1 Inicializace serveru

Komponenta `ServerInitializer` se stará o inicializaci serveru. S pomocí anotace `@PostConstruct` lze označit metodu, která se vykoná ihned po spuštění základních služeb serveru.

Při inicializaci serveru se vytváří výchozí administrátor. Pokud je tento administrátor upraven/smazán, po restartu serveru je účet výchozího administrátora opraven do výchozí podoby.

### 4.6.2 Dokumentace Swagger

Swagger a Swagger UI jsou nástroje, které vytvoří API dokumentaci na základě */META-INF/resources/*. Swagger UI umožňuje vytvořit přehlednou dokumentaci a REST klienta v jednom.

Pro účely produkčního nasazení je Swagger deaktivován, aby nebyla ohrožena bezpečnost (z důvodu znalosti všech endpointů).

### 4.6.3 Resources

V esources se nachází soubor *application.properties*, kde je soustředěna konfigurace serveru, databáze a e-mailového klienta. Všechny parametry se navíc dají přepsat přes příkazovou řádku při spouštění skrze *„java -jar“*.

# 5 Android klient

V této sekci stručně popíši fungování Android aplikace pro promotéry z hlediska uživatelského rozhraní i práce na pozadí. Aplikaci jsem testoval především v emulátoru. Reálné chování aplikace jsem prověřil na low-end tabletu (Android 7.0, 4jádrový procesor s taktem 1.2 GHz, 1 GB RAM, GPS, 3G).

Jako minimální podporovaná verze systému Android byla zvolena 5.1, což zajišťuje funkčnost i na starších zařízeních. Podpora starších verzí Androidu přinesla i některé ústupky a nutnost zachovat kompatibilitu. Můžu zmínit například nemožnost pracovat s `LocalDateTime`, `LocalDate` a `LocalTime` nebo chybějící streamy. Většinou lze ale najít backportované varianty, které je možno využít.

Největším krokem vpřed ve vývoji Android aplikace je za mě knihovna Anko, která značně vývoj ulehčuje, zejména v práci na pozadí aktivit. Ve spolupráci s jazykem Kotlin je vývoj mnohem pohodlnější než s použitím jazyka Java.

Aplikace je vyhotovena v anglické a české jazykové verzi a její stav závisí pouze na serveru. Pokud se tedy promotér například přihlásí na směnu a aplikaci vypne, aplikace po opětovném zapnutí zjistí informaci o započaté směně ze serveru.

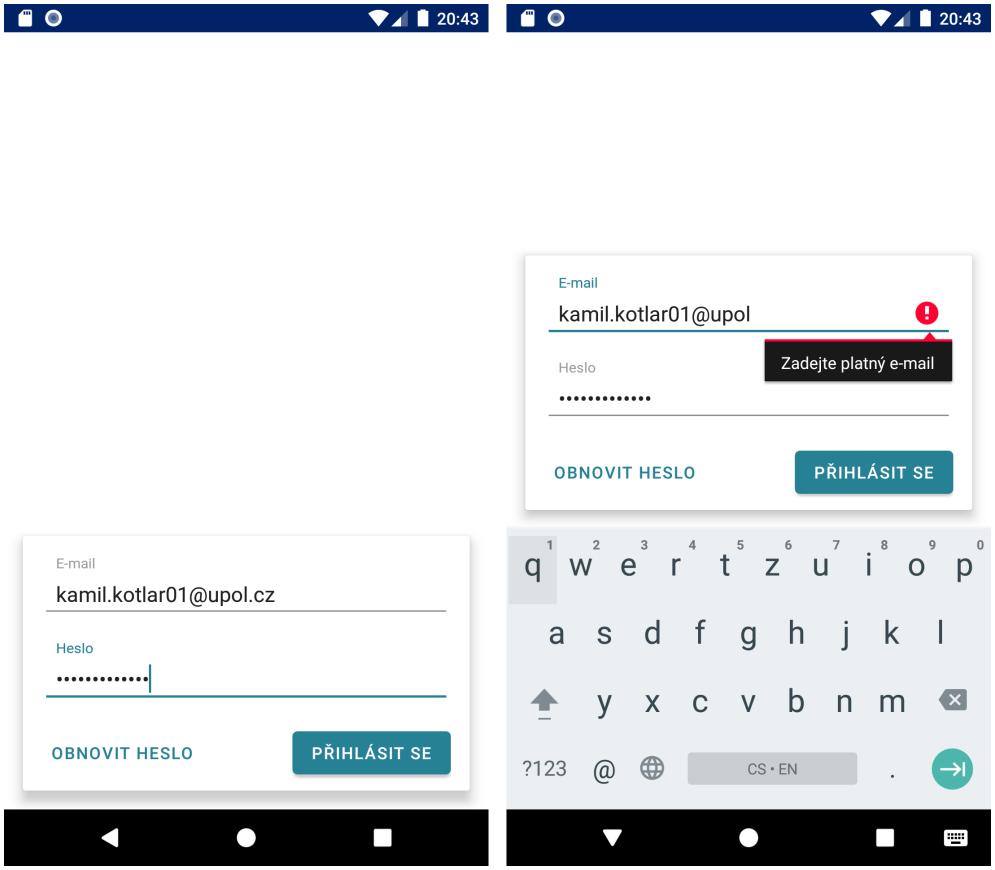
Vzhled aplikace je laděn v duchu Material Designu se snahou vyhnout se jakýmkoli designovým výstřelkům či nestandardnímu uživatelskému rozhraní. Google poskytuje velmi přesné podklady pro tvorbu UI v Material Designu [26], bohužel v package `com.google.android.material` implementace mnohých komponent chybí a bylo potřeba je dotvořit (Google vsází spíše na Flutter, pro který nenajdete komponentu se stavem „planned“). Barevné schéma i font aplikace lze jednoduše změnit a přinést tak každé společnosti personalizovaný produkt.

## 5.1 Přihlášení

### 5.1.1 Přihlašovací obrazovka

`LoginActivity` (obrázek 5) je aktivita, skrze kterou se promotér přihlásí do aplikace. Nejedná se o výchozí aktivitu. Do této aktivity uživatele přepne aplikace v momentě, kdy během pokusu o získání dat ze serveru zjistí, že uživatel není přihlášen a zároveň na zařízení nejsou uloženy validní přihlašovací údaje. Nejedná se tedy o aktivitu, na kterou by byl uživatel přiveden při každém spuštění aplikace.

Z hlediska UI jsem použil CardView s poměrně výaznou elevací, abych kartu lépe oddělil od stejnobarevného pozadí. Karta je posuvná, aby ji nepřekryla softwarová klávesnice a záměrně je ve spodní části displeje, aby byla dobře dosažitelná. Nad kartou je vcelku velký prostor, jehož využití má záviset na zákazníkovi. Počítá se především s logem firmy nebo agentury. Logo se při posuvu karty nahoru zmenší.




Obrázek 5: Přihlašování z obrazovky LoginActivity

### 5.1.2 Přihlašování a obnova hesla

Uživatelské údaje jsou automaticky vyplněny, pokud se nachází v lokální databázi. Validace e-mailu není řešena pouze na serveru, ale i na klientovi, aby se zkrátila doba odezvy aplikace. Dokud není vyplněn e-mail i heslo, tlačítko „PŘIHLÁSIT SE“ je deaktivované.

Po stisknutí přihlašovacího tlačítka dojde k zaslání přihlašovacích údajů. Autentizace uživatele byla popsána v sekci 4.4.3. Tuto akci je nutné udělat na pozadí, aby nedošlo k blokaci UI vlákna. Pokud je přihlášení úspěšné, přihlašovací údaje se uloží do lokální SQLite databáze. Z databáze se záznam maže při odhlášení, z bezpečnostních důvodů. Z hlavičky odpovědi se rovněž extrahuje JSESSIONID, které má Android k dispozici pro vyřízení dalších požadavků.

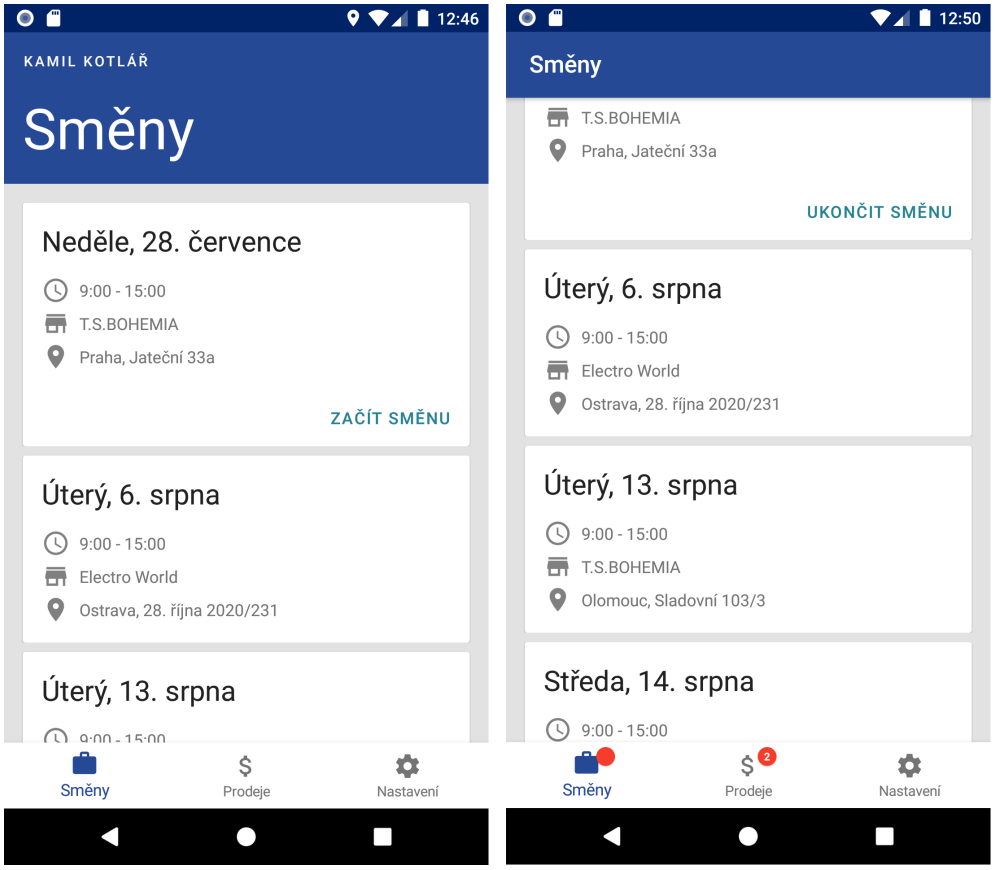
Z přihlašovací obrazovky je možné obnovit zapomenuté heslo. Při stisknutí tlačítka „OBNOVIT HESLO“ se před odesláním požadavku objeví potvrzovací dialog. Podrobně byla obnova hesla probrána v sekci 4.4.2.

## 5.2 Směny

### 5.2.1 Obrazovka se směnami

Po přihlášení do aplikace je uživateli k dispozici seznam směn z období následujících 30ti dnů. Skrze tuto obrazovku se promotér také přihlašuje na směnu. Obrazovka je zachycena na snímku 6.

Jednotlivé směny jako hlavní údaj zobrazují datum směny, v podrobném rozpise pak čas a konec směny, prodejnu a místo. Pokud je pro daný den k dispozici směna, na kartě se objeví tlačítko „ZAČÍT SMĚNU“/„UKONČIT SMĚNU“.




Obrázek 6: Směny na obrazovce ShiftsFragment

Při prvotním přihlášením na směnu je nutné, aby uživatel aplikace povolil přístup k poloze. Pokud výzvu odmítne, aplikace odmítne promotéra na směnu přihlásit. Jestliže se promotér nenachází v okruhu 100 metrů od místa směny, musí dát promotér souhlas k odeslání požadavku. V požadavku se nachází i informace o nevalidní poloze promotéra. Tato informace je následně uložena do databáze serveru. Obdobná je situace s ukončením směny. V momentě, kdy uživatel začne směnu, dojde ke změně tlačítka a zobrazení odznaku na spodním panelu.

V rozhraní aplikace bylo použito RecyclerView, protože se rozhraní jednotlivých karet opakuje. Použití RecyclerView namísto ListView šetří systémové pro-

středky a scrollování je mnohem plynulejší. Vrchní panel se dynamicky skrývá.

Co se týče činnosti na pozadí, tak pro seznam směn bylo nutné implementovat adaptér, který jej obsluhuje. Řeší přidávání i odebírání položek. Aplikace veškeré potřebné informace ze serveru načítá asynchronně a jakmile jsou odpovědi k dispozici, zobrazí je. Mezitím obrazovka zobrazuje placeholder rozložení.

### 5.2.2 Ověřování polohy

Ověřování polohy se děje pouze lokálně. Zasílat v požadavku polohu nemá smysl, server nemá možnost ověřit validitu polohy. Informace o místu směny má aplikace k dispozici. Souřadnice porovná s aktuálními a vydá výstup z dvouprvkové množiny výstupů {shoduje, neshoduje}. Teoreticky se jedná o triviální činnost.

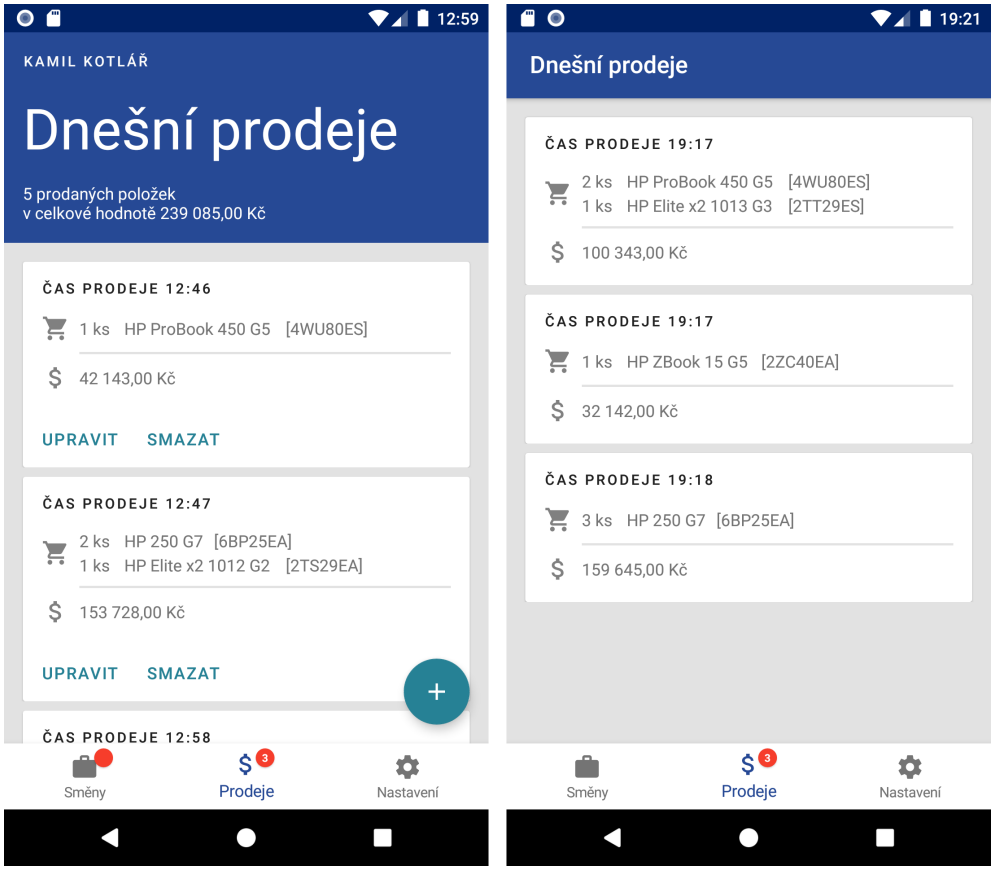
V praxi je situace horší, je třeba počítat s několika situacemi, které mohou nastat. Může se tak jednat o nepovolení přístupu k poloze, vypnutí GPS služby nebo dokonce vypnutí GPS uživatelem během získávání polohy.

Získání polohy se děje ve třídě `LocationService`. Jako poskytovatele nevolím násilím GPS a volbu nechávám na OS. Jediné kritérium, podle kterého Android vybere poskytovatele polohy, je přesnost (tu požaduji vysokou, odpovídá maximální odchylce 100 metrů). V některých případech je tedy poloha určena na základě IP a MAC adres (dle referenční příručky). Na získání polohy má aplikace 10 vteřin, poté se objeví chybová hláška.

Se situací, kdy uživatel vypne GPS v průběhu získávání souřadnic, byl vcelku problém. `LocationListener` sice obsahuje metody, které jsou volány v určitých situacích (`onLocationChanged()`, `onStatusChange()`, `onProviderEnabled()` a `onProviderDisabled()`), ale žádná z nich se neaktivuje při vypnutí lokace. Tato situace tedy byla vyřešena zavedením časového limitu 10ti vteřin, jinak docházelo k cyklení.

## 5.3 Prodeje

### 5.3.1 Obrazovka s prodeji




Obrázek 7: Obrazovka s prodeji během a po směně

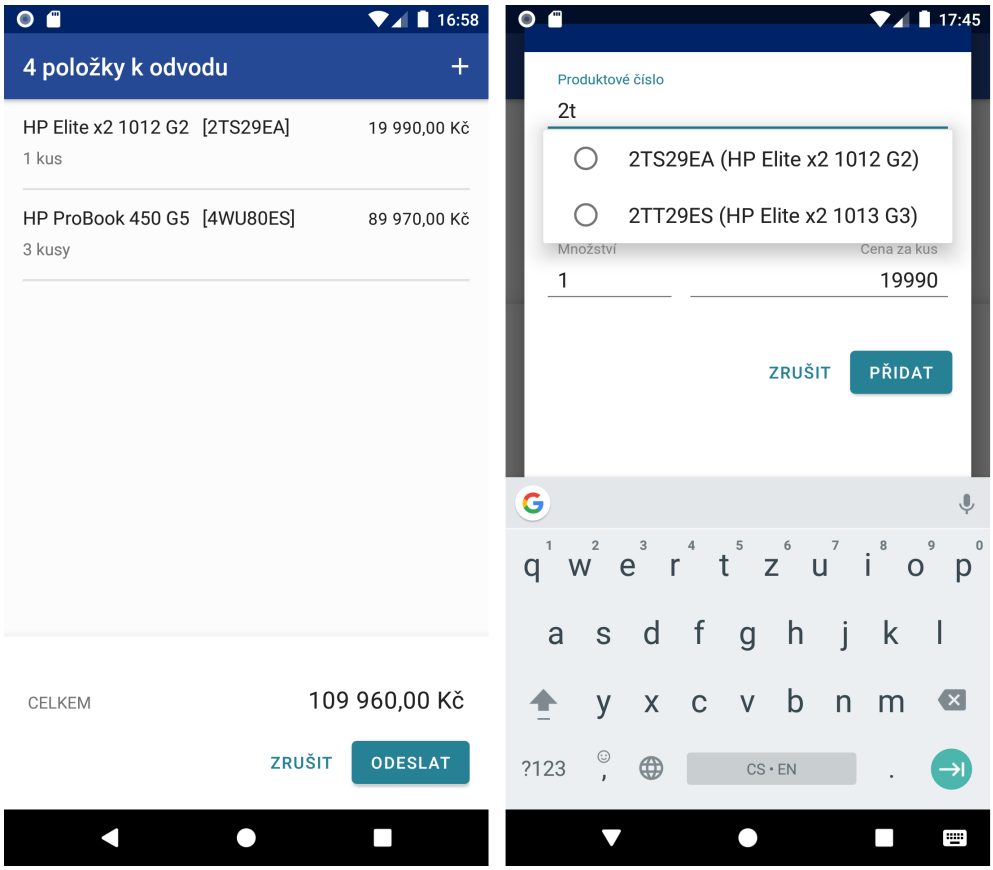
Fragment na obrázku 7 je po vzhledové stránce velmi podobný jako obrazovka se směnami. Opět byl použit stejný horní panel i RecyclerView. V horním panelu se tentokrát schovává i souhrn prodejů, naproti tomu odznak na spodním panelu počítá jednotlivé prodeje. Navíc je zde plovoucí tlačítko „+“, nacházející v pravém dolním rohu. Tlačítko slouží k přidání prodeje a je aktivní pouze tehdy, je-li promotér přihlášen ke směně. Totéž platí tlačítkách „UPRAVIT“ a „SMAZAT“ na kartách s prodeji. Nadpis karty je tentokrát zvolen menší, protože důležitější je její obsah. Prodeje se opět stahují z databáze při startu aplikace.

### 5.3.2 Synchronizace produktů

Produkty, které je možno přidat do prodeje se stahují ze serveru a cachují v zařízení. Aplikace se při startu dotáže serveru, jaká verze databázové tabulky produktů je na jeho straně a verzi porovná s lokální. Pokud verze nejsou ekvivalentní, klient stáhne ze serveru novou verzi a tu uloží do lokální SQLite databáze. Stažení celé tabulky produktů není nijak datově náročné, v praxi se nebude jednat

o více než 200-300 záznamů. Půjde řádově o kilobajty. Porovnávání záznamů v jednotlivých verzích a následné částečné stahování by bylo výpočetně náročnější a náchylnější ke vzniku desynchronizační chyby. Stáhnutím celé tabulky je toto riziko eliminováno.

### 5.3.3 Přidávání produktů




Obrázek 8: Přidávání produktů a ukázka našeptávače

Během přidávání prodejů se ze směny využijí informace o prodejně, konkrétně měna, ve které se v dané zemi platí. Informace mezi aktivitami/fragmenty se předávají přes `Intent`.

Tlačítkem „+“ je možno přidat položky prodeje. Dialog, který se po stisknutí tlačítka objeví, je navržen tak, aby přidání produktu bylo co nejrychlejší. K rychlému odvodu prodeje napomáhá našeptávač, který filtruje produkty z lokální databáze. Našeptávač se aktivuje po zadání dvou znaků. Obrazovku s produkty a našeptávač lze vidět na snímku 8.

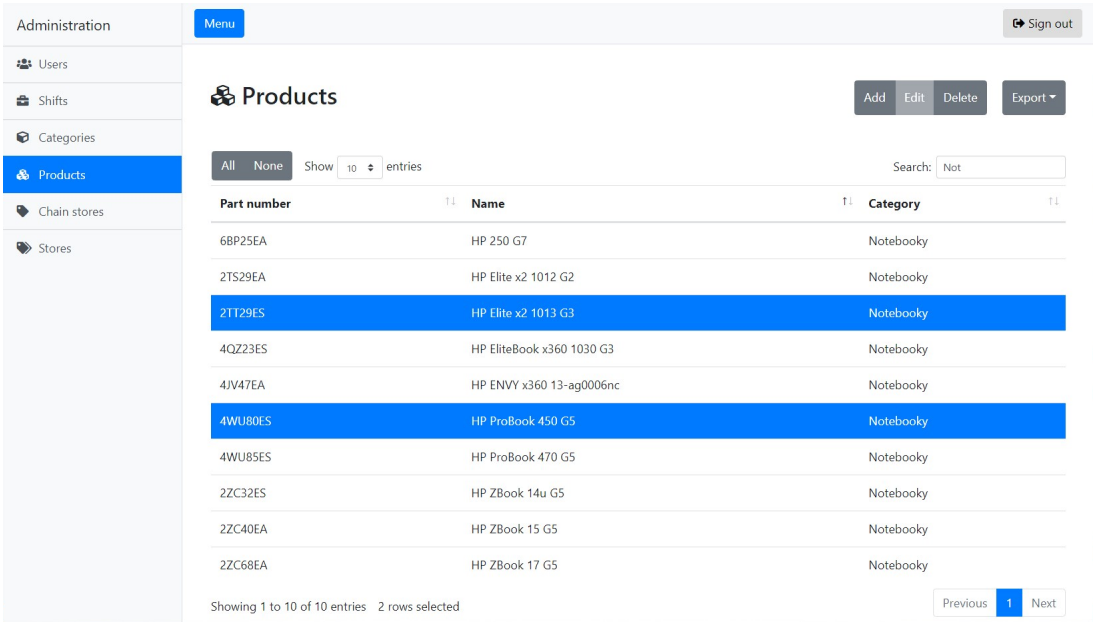
Z implementačního hlediska filtrace produktů skutečně probíhá již v databázi, za pomoci klíčového slova **`LIKE`**. Nemusí se tak z databáze vytahovat všechny záznamy, ale jen ty, které je potřeba zobrazit uživateli.

# 6 Webový klient

Původně byl projekt plánován pouze jako Android aplikace pro promotéry a server s možností napojení libovolného klienta v budoucnu. Po konzultaci s vedoucím práce jsem se rozhodl na základě jeho doporučení zhotovit jednoduchý web pro demonstraci API serveru.

Při vývoji webu byly technologie vybrány s ohledem na to, aby byl jeho vývoj co nejrychlejší. Významně k tomu pomohla knihovna DataTables, která poskytla výkonné a přizpůsobitelné nástroje pro práci s daty.

Web je zhotoven v anglické lokalizaci a rozlišuje uživatele dle jeho role. Vstupy uživatele jsou validovány na straně klienta. Pokud by však někdo manipuloval se zdrojovým kódem, pak nesrovnalosti zachytí validátory na straně serveru.




Obrázek 9: Snímek web klienta

Po vzhledové stránce jsem respektoval výchozí styl i barevné schéma knihovny bootstrap. Vzhled podtrhují ikony ze služby Font Awesome. Snímek z webu lze vidět na obrázku 9.

Požadavky na server zprostředkovává Ajax. Při výskytu chyby se využívají výchozí chybové hlášky zasílané v odpovědi serveru.

Napříč jednotlivými kategoriemi webu je práce s tabulkami velmi podobná. Jedná se o akce přidání položky, editaci položky a smazání položek. Data lze rovněž exportovat pro tisk, do PDF, CSV nebo XLS formátu. Tabulky podporují stránkování, volbu jedné i více položek a vyhledávání.

# 7 Plány do budoucna

Vytvořenou platformu bych rád nadále rozvíjel v mnoha směrech a implementovat do ní nápady, které jsem dostal v průběhu vývoje, a na které prozatím nebyl čas. Jmenovitě by se mohlo jednat o tyto funkce a vylepšení:

- Rozdělit server na samostatné moduly. Modulární server by mohl plnit personalizované požadavky jednotlivých agentur.
- Vytvořit nástroje pro generování a zobrazování statistik z prodejů.
- Vytvořit iOS aplikaci nebo zvážit přechod na progresivní webovou aplikaci.
- Prozkoumat možnosti validace přijatých souřadnic na straně serveru.
- V Android aplikaci využít notifikace pro upozornění promotérů o nadcházející směně.
- Umožnit změnu hesla jiným způsobem než vygenerováním řetězce serverem (na druhou stranu server takto zaručuje, že každé heslo je bezpečné).
- Implementovat jednoduchý widget pro připnutí na plochu mobilního zařízení nebo část aplikace portovat na nositelnou elektroniku (pravděpodobně lepší možnost vzhledem k rostoucí popularitě „wearables“).
- Vytvořit systém pro automatické generování rozvrhů směn.
- Vytvořit burzu směn pro promotéry.
- Rozšířit povědomí o platformě mezi agentury.

# Závěr

V době vytváření zadání práce jsem měl takřka mlhavou představu o tom, jak se vyvíjí server. Překvapivě ale velké množství práce bylo potřeba vykonat také na Android klientovi. Přišel jsem na to, že vytvoření kvalitní nativní aplikace není zdaleka tak jednoduché jako vytvoření webu. Na druhou stranu by se investovaná práce měla vrátit v podobě spolehlivosti a větší úspory energie mobilních zařízení.

Výsledkem mé bakalářské práce je systém, který by měl být schopen obstát v ostrém provozu. Jak již bylo zmíněno, cílem bylo vytvořit především silný základ, na kterém je možné dále pracovat. Server tento ohled dle mého názoru splnil, možnosti rozšíření jeho funkcionality jsou téměř neomezené.

Co se týče budoucího rozvoje platformy, je potřeba implementovat nástroje pro tvorbu a zobrazení statistik pro prodeje. Jedná se o funkcionalitu, která by pro marketingové agentury mohla být velkým lákadlem. Pro reálné použití bude potřeba webu přidat funkcionalitu pro import většího množství dat najednou. Server je na možnosti hromadného přidávání položek připraven.

Posledním střípkem, který brání praktickému použití je neexistence iOS klienta, což nelze ignorovat. Za úvahu stojí, zda by z dlouhodobého hlediska nedávala více smysl progresivní webová aplikace.

# Conclusions

At the time of creating the thesis assignment I barely knew what the server development is. Surprisingly, however, a large amount of work had to be done even on the Android client. I found that creating a quality native app is not nearly as easy as creating a website. On the other hand, invested work should return in the form of reliability and greater energy savings for mobile devices.

As the result of my Bachelor thesis is a system that should be able to withstand the live operation. As already mentioned, the aim was to create a strong extensible basis. In my opinion, the server fulfilled this request. The possibilities of extending its functionality are almost unlimited.

For the future development of the platform, it should be implemented some tools for the manipulation with sales statistics. This is the functionality that could be a great bait for marketing agencies. For real use, it will be necessary to add functionality to the website to unlock the possibility to import more data at once. The server is ready for it.

The last piece that prevents the platform to be used in a real world is the absence of an iOS client, which cannot be ignored. It is worth considering whether a progressive web application would make more sense in the long run.

# Literatura


[1] WIKIPEDIE. *Marketingová propagace — Wikipedie: Otevřená encyklopedie* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://cs.wikipedia.org/w/index.php?title=Marketingov%C3%A1_propagace&oldid=16978914

[2] KING STYLE S.R.O. *Slovník reklamy* [online]. 2008. [cit. 2019-07-26]. Dostupný z: http://www.kingstyle.cz/cs/slovnik-reklamy/?mt=4

[3] WIKIPEDIE. *Promotér — Wikipedie: Otevřená encyklopedie* [online]. 2017. [cit. 2019-07-26]. Dostupný z: https://cs.wikipedia.org/w/index.php?title=Promot%C3%A9r&oldid=14508253

[4] GRIMES, Steve. *Display Images of Locations, Users and Activities*. 2019. [cit. 2019-07-26]. Dostupný z: http://teamhaven.blogspot.com/2019/03/brief-field-teams-using-images-store.html

[5] WIKIPEDIE. *Java Virtual Machine — Wikipedie: Otevřená encyklopedie* [online]. 2019. [cit. 2019-07-29]. Dostupný z: https://cs.wikipedia.org/w/index.php?title=Java_Virtual_Machine&oldid=16987955

[6] JETBRAINS S.R.O. *Null Safety* [online]. [cit. 2019-07-26]. Dostupný z: https://kotlinlang.org/docs/reference/null-safety.html

[7] JETBRAINS S.R.O. *Kotlin* [online]. [cit. 2019-07-26]. Dostupný z: https://kotlinlang.org

[8] MOZILLA *Co je JavaScript?* [online]. 2005-2019. [cit. 2019-07-26]. Dostupný z: https://developer.mozilla.org/cs/docs/Learn/JavaScript/First_steps/Co_je_JavaScript

[9] PIVOTAL SOFTWARE, INC. *Spring Framework* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://spring.io/projects/spring-framework

[10] PIVOTAL SOFTWARE, INC. *Spring Boot* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://spring.io/projects/spring-boot

[11] PIVOTAL SOFTWARE, INC. *Spring Security* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://spring.io/projects/spring-security

[12] PIVOTAL SOFTWARE, INC. *Spring for Android* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://projects.spring.io/spring-android/

[13] WIKIPEDIE. *Hibernate — Wikipedie: Otevřená encyklopedie* [online]. 2018. [cit. 2019-07-26]. Dostupný z: https://cs.wikipedia.org/w/index.php?title=Hibernate&oldid=16504878

[14] JETBRAINS S.R.O. *Anko* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://github.com/Kotlin/anko/blob/master/README.md

[15] BOOTSTRAP TEAM *Bootstrap* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://getbootstrap.com/

[16] BAELDUNG SRL. *REST with Spring Tutorial*. 2019. [cit. 2019-07-26]. Dostupný z: https://www.baeldung.com/rest-with-spring-series

[17] APPLE INC. *Co je API?* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://support.apple.com/cs-cz/guide/shortcuts/apd2e30c9d45/ios

[18] WIKIPEDIE. *Representational State Transfer — Wikipedie: Otevřená encyklopedie* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://cs.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=17357691

[19] INTERNET SOCIETY, The. *RFC 2616* [online]. 1999. [cit. 2019-07-26]. Dostupný z: https://tools.ietf.org/html/rfc2616#page-53

[20] WIKIPEDIE. *Vkládání závislostí — Wikipedie: Otevřená encyklopedie* [online]. 2018. [cit. 2019-07-26]. Dostupný z: https://cs.wikipedia.org/w/index.php?title=Vkl%C3%A1d%C3%A1n%C3%AD_z%C3%A1vislost%C3%AD&oldid=16084552

[21] JETBRAINS S.R.O. *Data Classes* [online]. [cit. 2019-07-26]. Dostupný z: https://kotlinlang.org/docs/reference/data-classes.html

[22] WIKIPEDIE. *Objektově relační mapování — Wikipedie: Otevřená encyklopedie* [online]. 2014. [cit. 2019-07-26]. Dostupný z: https://cs.wikipedia.org/w/index.php?title=Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD&oldid=11440457

[23] ORACLE CORPORATION. *JSR 338: JavaTM Persistence 2.2* [online]. 2019. [cit. 2019-07-26]. Dostupný z: https://jcp.org/en/jsr/detail?id=338

[24] ORACLE CORPORATION. *The Java EE 5 Tutorial* [online]. 2010. [cit. 2019-07-26]. Dostupný z: https://docs.oracle.com/javaee/5/tutorial/doc/bnbqa.html

[25] RIETTA, Frank. *Use Bcrypt or Scrypt Instead of SHA* for Your Passwords, Please!* [online]. 2016. [cit. 2019-07-27]. Dostupný z: https://rietta.com/blog/bcrypt-not-sha-for-passwords/

[26] GOOGLE LLC. *Design guidance and code* [online]. 2019. [cit. 2019-07-27]. Dostupný z: https://material.io/

# A Obsah přiloženého CD/DVD

| | |
|---|---|
| **server/** | |
| **src/** | Zdrojový kód serveru. |
| **promo-1.0.0.jar** | Spustitelný .jar soubor. |
| **android/** | |
| **src/** | Zdrojový kód Android klienta. |
| **app-release.apk** | Podepsaný .apk soubor. |
| **web/** | |
| **src/** | Zdrojový kód webu. |
| **kidiplom/** | Tento text a jeho zdrojový kód. |
| **server_requests.http** | Ukázka API volání. |
| **README.txt** | Instalační instrukce. |