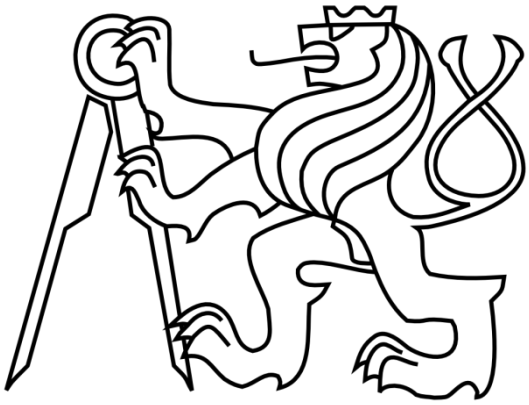
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

WEBOVÉ ROZHRANÍ RESTAURAČNÍHO SYSTÉMU

Tomáš Červenka

Vedoucí práce: Ing. Komárek Martin

Studijní program: Softwarové technologie a management, Bakalářský
Obor: Softwarové inženýrství

Zadání Bakalářské práce

Poděkování

Děkuji Ing. Martinu Komárkovi za pomoc při odborné vedení bakalářské práce a rady, které vedli k jejímu zpracování. Děkuji také Vítovi Samkovi za pomoc při plánování a při seznámení s projektem, na který tato bakalářská práce navazuje.

Prohlášení

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

V Praze dne 27. 5. 2016

Abstrakt

Hlavním cílem této práce je přepsání stávajícího restauračního systému Cashbob do webové aplikace. Aplikace je napsaná v jazyce Java za použití Play! Frameworku. Mezi základní implementovanou funkčnost patří dostupnost REST rozhraní, které umožňuje vzdálenou práci se systémem a implementaci pokladny systému přes webové rozhraní. K práci patří analýza, testování, optimalizace uživatelské přívětivosti, tvorba dokumentace a nasazení aplikace.

Abstract

The main goal of thesis is to rewrite current restaurant system Cashbob to web application. Application is to be written in Java programming language with use of Play! Framework. The key implemented functionality is REST interface availability which allows remove operations on system and implementation of web cash desk interface. Analysis, testing, user accessibility optimization, documentation creation and application deployment belongs to project as well.

Obsah

Slovník pojmů

HTML request – požadavek odeslaný z klienta (prohlížeče) na server.

HTML response – odpověď serveru na request klienta, zaslaný klientovi.

ETag – údaj v hlavičce HTML requestu sloužící k validaci cache.

REST – architektura rozhraní sloužící primárně k přístupu k datům pomocí HTML requestů.

1. Úvod

1.1. Motivace

Jedna z prací, která se musí řešit v restauracích je evidence objednávek zákazníků. V minulosti všechny restaurace tento problém řešili do různé míry propracovaným systémem ručně psaných papírových poznámek. Postupem času se i do restaurací, jako i do jiných odvětví našeho života, začala protlačovat technika. Proto ve většině restaurací můžeme pozorovat používání elektronických pokladen a zároveň obsluhy používající přenosné zařízení k záznamu objednávek. Na tuto problematiku cílí restaurační systém CashBob [1].

Použití pokladního systému má několik výhod. Zefektivňují práci, protože umožňuje přehledně evidovat objednávky, umožňuje je sdílet mezi více zařízeními, potažmo místy, bez toho, aby si informace o nich pracovníci personálu fyzicky předávali. Také snižuje chybovost procesu zpracovávání objednávek, protože se v něm nikde neztratí a zároveň je v něm řada informací (např. týkajících se objednatelných položek) zadaná na pevno, takže si je personál nemůže splést. Informace sesbírané při běhu podniku se dále dají použít ke správě skladu a inventurám. Dá se porovnat, kolik zboží se prodalo na pokladně, kolik ho reálně bylo vydáno na pokladně a následně případný rozdíl, který by měla obsluha doplatit.

1.2. Stávající práce

Na systému CashBob se pracovalo už 5 let v rámci více projektů [2,22-34]. V rámci nich vznikl systém pro restaurace skládající se z více modulů, umožňující správu podniku, skladu, směn zaměstnanců a poskytující funkci pokladny spojené se správou objednávek.

Systém je psaný v jazyce Java. Skládá se ze serverové části, kde běží REST rozhraní a databáze. K němu se přes REST připojuje klient psaný ve Swingu.

Momentálně je systém CashBob ke stažení na stránkách http://www.cashbob.cz, kde ho publikoval Vít Samek, který na něm pracoval v rámci své bakalářské práce [2]. Nabízí následující moduly: Pokladna, Správu podniku, Správa skladu a Plánování směn. V textu ho dále budu zmiňovat jako „stávající systém“ nebo „stávající aplikaci“.

1.3. Na čem se bude pracovat

V rámci této bakalářské práce přepracuji stávající aplikaci tak, aby běžela jako webová aplikace a dalo se k ní přistupovat přes webového prohlížeče z libovolného zařízení. Výhoda oproti běhu v desktopovém režimu je, že jednotliví uživatelé nemusejí nic stahovat ani instalovat. To zároveň umožní nasazování nových verzí systému bez toho, aby to zasáhlo uživatele, kteří se systémem pracují – je potřeba pouze provést update serveru.

Protože celkový je rozsah systému CashBob příliš veliký, v rámci tohoto projektu se zaměřím na předělání jeho části týkající se pokladny a objednávek a na REST rozhraní.

V textu budu novou aplikaci zmiňovat jako webová aplikace (WA).

2. Plán práce

Protože vývoj aplikace bude probíhat delší dobu, je třeba stanovit si časový odhad. Toho dosáhnu definováním jednotlivých úkolů, které je třeba na práci provést. Díky tomuto rozpadu na menší části je snadnější odhadnout časovou náročnost celého projektu a definovat si postup, jak ho zpracovávat.

Tato kapitola řeší rozpad práce na jednotlivé úkoly, jejich časové ohodnocení a určení časového harmonogramu, podle kterého je budu zpracovávat.

2.1. Plánování a časový odhad

2.1.1. Příprava

Celková doba: 5 hod
Připojení a správa GITu *(2 hod)*
Seznámení se se stávajícím projektem *(3hod)*

- → Spuštění aplikace
- → Spuštění aplikace ze zdrojových kódů
- → Seznámení se se systémem

2.1.2. Analýza

Celková doba: 13 hod
Návrh rolí a práv *(4 hod)*
Návrh cachování *(4 hod)*
Definování coding standards *(1 hod)*
Návrh použitých technologií *(4 hod)*

2.1.3. Aplikační základ

Celková doba: 86 hod
Implementace modelu *(24hod)*

- → Nastavení v Play!
- → Rozchození ORM proti databázi
- → Přenesení entity tříd a tříd na ně navazující
- → Import testovacích dat

Implementace loginu a zabezpečení = autentizace *(8 hod)*

- → Udržení uživatele v cachi
    - Implementace session timeout

Implementace autorizace – rolí a práv *(4 hod)*
Zkopírování business tříd a jejich úprava (týká se balíčků rest, dto, svc, exceptions, conv, utils)

- → Odstranění závislostí na již nepoužité knihovny *(4 hod)*
- → Refactoring podle coding standards *(32 hod)*

Implementace REST *(14 hod)*

- → Přepis tříd poskytující rozhraní do Play

2.1.4. Implementace pokladny

Celková doba: 54 hod
Implementace tvorby nového účtu (modální okno se stoly) *(4 hod)*
Implementace obrazovky se seznamem otevřených účtů *(4 hod)*
Implementace práce s účtem

- → Implementace práce s menu *(8 hod)*
  - Implementace vykreslení menu a pohybu v něm
  - Implementace přesunu položek menu do objednávky
- → Implementace přidání jednorázové položky do objednávky *(2 hod)*
- → Implementace rámečku objednávky *(8 hod)*
  - Implementace odebrání položky z objednávky
  - Implementace hromadného zrušení a objednání celé objednávky
  - Implementace tisku objednávky
- → Implementace rámečku účtu *(8 hod)*
  - Implementace editace účtu (jméno, stůl, objednávka)
  - Implementace přidání položky menu z účtu do objednávky

Implementace zaplacení účtu *(20 hod)*

- → Implementace rámečku K zaplacení
  - Implementace převedení položky zpátky do účtu
  - Implementace hromadného převedení všech položek zpátky do účtu
- → Implementace rámečku účtu
  - Implementace převedení položky do K zaplacení
  - Implementace hromadného zaplacení
- → Implementace zaplacení položek v rámečku K zaplacení
  - Implementace uzavření účtu
- → Implementace tisku účtenky

Implementace přesunutí položek mezi účty *(20 hod)*

- → Implementace rámečku účtu a rámečku K přesunutí
  - Implementace přesunutí položky z účtu do rámečku K přesunutí
- → Implementace rámečků Cílový účet a Detail
  - Implementace zobrazení všech účtů v rámečku Cílový účet
  - Implementace zobrazení položek účtu v Detailu
- → Implementace přesunutí položek z rámečku K přesunutí do Cílového účtu
  - Implementace uzavření účtu
- → Implementace tvorby nového účtu

2.1.5. Implementace nastavení

Celková doba: 10 hod

Implementace změny vlastního hesla *(2 hod)*

Implementace správy stolů *(8 hod)*

2.1.6. Implementace tisku

Celková doba: 20 hod

Zjištění, jakým způsobem se tiskárna obsluhuje z Javy

Implementace převedení dat do formátu pro tiskárnu

Implementace tisku

2.1.7. Testování

Celková doba: 24 hod

Testování uživatelské přívětivosti *(8 hod)*

Oprava zjištěných chyb *(16 hod)*

Selenium IDE testy *(40 hod)*

2.1.8. Nasazení

Celková doba: 16 hod

Nasazení na linux server *(16 hod)*

2.1.9. Dokumentace

Celková doba: 82 hod
Psaní bakalářské práce *(40 hod)*
Dokumentace REST API *(18 hod)*
Instalační postup *(2 hod)*
Popis Java package *(2 hod)*
Doménový model *(2 hod)*
Vedení wiki *(18 hod)*

2.2. Harmonogram projektu

Projekt bude zpracováván a odevzdáván v iteracích. Každá iterace bude trvat 2 až 3 týdny a jejím výsledkem bude funkční aplikace s příslušnou dokumentací a testy. Díky tomu bude na konci iterace uzavřená předem daná část projektu. Například se nestane se, že by psaní dokumentace nebo testování iterace bylo odloženo až na konec projektu. Zároveň pokud na nějaký úkol nezbude dost času, aplikace bude uzavřená a alespoň její část funkční [3].

2.2.1. Iterace 1 – Úprava stávajícího kódu

Odevzdání: 16. 11. 2015
Trvání: 2 týdny

- Úprava použitého stávajícího kódu
    - Změny v použitých balíčcích
    - Změny v REST API

2.2.2. Iterace 2 – Zabezpečení aplikace

Odevzdání: 30. 11. 2015
Trvání: 2 týdny

- Analýza stávajícího zabezpečení
- Návrh implementace autorizace
- Implementace loginu a zabezpečení = autentizace
- Implementace autorizace – rolí a práv
- Tvorba Selenium testů pro login
- Příprava instalačního balíčku a návodu ke spuštění

2.2.3. Iterace 3 – Správa stolů

Odevzdání: 14. 12. 2016
Trvání: 2 týdny

- Analýza stávající správy stolů
- Implementace správy stolů
- Tvorba Selenium testů pro správu stolů
- Příprava instalačního balíčku

2.2.4. Iterace 4 – Seznam účtů a tvorba nového účtu

Odevzdání: 4. 1. 2016
Trvání: 3 týdny

- Analýza a implementace tvorby nového účtu

- Analýza a implementace obrazovky se seznamem otevřených účtů
- Tvorba Selenium testů pro tvorbu nového účtu a zobrazení otevřených účtů
- Příprava instalačního balíčku

2.2.5. Iterace 5 – Menu

Odevzdání: 25. 1. 2016
Trvání: 3 týdny

- Analýza stávajícího menu a práce s ním
- Implementace práce s menu
- Uživatelské testování menu
- Příprava instalačního balíčku

2.2.6. Iterace 6 – Práce s účtem

Odevzdání: 15. 2. 2016
Trvání: 3 týdny

- Analýza stávající práce s účtem
- Implementace práce s účtem
- Tvorba Selenium testů pro práci s účtem
- Příprava instalačního balíčku

2.2.7. Iterace 7 - Přesunutí položek mezi účty

Odevzdání: 29. 2. 2016
Trvání: 2 týdny

- Analýza stávajícího přesunutí položek mezi účty
- Implementace přesunutí položek mezi účty
- Tvorba Selenium testů pro přesunutí položek mezi účty
- Příprava instalačního balíčku

2.2.8. Iterace 8 - Zaplacení účtu

Odevzdání: 21. 3. 2016
Trvání: 3 týdny

- Analýza stávajícího zaplacení účtu
- Implementace zaplacení účtu
- Tvorba Selenium testů pro zaplacení účtu
- Příprava instalačního balíčku

2.2.9. Iterace 9 - Tisk

Odevzdání: 4. 4. 2016
Trvání: 2 týdny

- Analýza a implementace tisku
- Otestování tisku
- Příprava instalačního balíčku

2.2.10. Iterace 10 - Odevzdání bakalářské práce

- Odevzdání bakalářské práce ke kontrole (hotovo 18. 4. 2016 – 2 týdny)
- Finální verze bakalářské práce (hotovo 9. 5. 2016 - 3 týdny)

2.2.11. Další termíny

- Popis Java package (hotovo k 4. 1. 2016)
- Návrh cachování (hotovo k 4. 1. 2016)
- Dokumentace REST API (hotovo k 4. 1. 2016)
- JavaDoc (hotovo 11. 1. 2016)
- Zhodnocení dosavadní práce (hotovo 11. 1. 2016)

2.3. Celkový časový a finanční odhad

Z plánu jednotlivých iterací vychází celková časová náročnost tohoto projektu na 370 hodin. To odpovídá 47 osmihodinovým pracovním dnům.

Uvážím-li náklady na zaměstnance za jeden den 3000 Kč, náklady na řešení tohoto projektu by celkově činili 141 tisíc Kč.

3. Technologie

Tato kapitola rozebírá, které zásadní technologie budou použité při implementaci webové aplikace a jejím následném testování.

3.1. Server

Webová aplikace bude psaná v jazyce Java 8 a poběží na frameworku Play! [4] ve verzi 2.4.x. Play je MVC framework, což znamená, že umožňuje vytvořil datový model a logiku aplikace a zároveň spravuje uživatelské rozhraní. Používá webový server Netty [5].

3.2. Databáze

Jako databáze bude použita H2 [6]. Ta je pro Play nativní (po správné konfiguraci se spouští spolu s Playem), používá SQL a je a pro účely projektu nabízí dostatečnou funkcionalitu. Mezi její základní vlastnosti patří vysoká rychlost, je open-source (zdarma), nabízí JDBC api (tzn. dá se volat přímo z Javy) a má malou velikost. Také nabízí webový prohlížeč, takže není potřeba instalovat zvlášť, jako u jiných databází.

Play sice používá vlastní ORM framework, ale protože v předchozím projektu byl použit Hibernate spolu s JPA, rozhodli jsme se použít tyto technologie. Ty po nakonfigurování zajišťují (především díky anotacím v entitách) automatickou práci s databází bez nutnosti přímého psaní SQL příkazů. Ty jsou ve výsledku použity jenom zřídka u komplikovanějších dotazů.

3.3. Uživatelské rozhraní

Vzhledem k tomu, že je cílem projektu je tvorba webové aplikace, uživatelské rozhraní je zajištěné webovými stránkami ve formátu HTML.

3.3.1. Scala template

Ke generování stránek se používá Play! template engine založený na Scale. Umožňuje do HTML kódu vnořovat scala výrazy. Pomocí nich je možné předávat parametry a používat různé programovací výrazy, jako je if-else, iterování, znovu použitelné bloky kódu apod.

3.3.2. CSS/JS frameworky

Dále je možné použít CSS/JS frameworky pro snadnější stylizování stránek a další view funkcionalitu. Nejrozšířenější dva jsou Bootstrap [7] a Foundation [8]. Na internetu se nachází řada recenzí rozbírající tyto dva frameworky. Oba jsou volně dostupné, nabízí stejnou základní funkcionalitu a jsou podporované hlavními prohlížeči. Mezi jejich vlastnosti a funkce patří: responzivita, modularita, možnost práce s Less [9] a Sass [10]. Pro použití v projektu jsem vybral Bootstrap. Je rozšířenější, což slibuje větší komunitu, potažmo více dokumentace a materiálů pro práci s ním. Zároveň jsem ho již v minulosti používal, takže jeho výběr znamená časovou úsporu.

Některé pluginy Bootstrapu používají knihovnu jQuery [11] a v případě jejich použití je nutné ji importovat.

3.3.3. React.js

Při iteraci 5, v rámci které jsem implementoval jídelní lístek restaurace (dále menu), bylo potřeba vyřešit, jakým způsobem budu menu překreslovat při jeho průchodem. Z hlediska technologie by bylo nejsnadnější stránku obsahující menu pokaždé poslat ze serveru znovu. Tato varianta by bohužel byla pro účely menu, od kterého chceme, aby reagovalo okamžitě, nepoužitelná. Spojení se serverem, poslání nové response stránky a její vykreslení zaberou příliš mnoho času a uživatele by to zdržovalo.

Jediným způsobem, jak provést překreslení zobrazeného obsahu v browseru je pomocí Javascriptu. V samotném Javascriptu je poměrně náročné přepisovat větší množství obsahu stránky, protože se musí vybrat každý element stránky a nahradit jeho část (například textový obsah nebo název CSS třídy) nebo větší blok stránky, pak se ale musí z textu vyescapovat HTML značky.

React.js [12] oproti tomu umožňuje do div elementu vložit a následně dynamicky měnit standardní HTML kód. React třídy si drží svůj stav, například aktuální pozici v menu. Ten je možné měnit při spuštění uživatelských akcí, jak je klik na tlačítko, najetí na něj myší apod. Při změně stavu dojde k překreslení HTML kódu divu, který danná React třída spravuje – takže při rozkliknutí skupiny položek menu se změní aktuální pozice v menu a dojde k jeho překreslení.

3.4. Technologie pro testování

Součástí zadání bakalářské práce je testování implementace jednotlivých iterací. Aby se mohlo snadno a opakovaně provádět zpětně, je potřeba ho zautomatizovat. Pro tento účel jsem se rozhodl využít software Selenium [13]. Ten slouží k automatizaci prohlížení webových stránek. Testy budou psané v doplňku pro Mozillu Firefox, v Selenium IDE. Ten slouží jako vývojové prostředí, ve kterém je možné nahrávat průchod aplikací a následně ho spouštět a debugovat. Jako výsledný produkt bude HTML script.

4. Standardy kódování

Pro dlouhodobou udržitelnost projektu je důležité psát kód tak, aby byl při vrácení se k němu dobře čitelný, pochopitelný a upravitelný [14][15]. To jak psát takový kód, zajistí určení si standardů, které bude splňovat. Dále jsou popsaná kritéria, která by měl kód splňovat.

Použití správných typů – u atributů a parametrů budou použity typy, které v rámci daného použití odpovídají jazyku Java. Například atributy uchovávající binární stav (true/false) budou definovány jako boolean a ne jako integer.

Komentování – všechny složitější metody a třídy budou obsahovat komentář ve formátu Javadoc popisující jejich chování. Kód také bude popsaný uvnitř metod. Komentáře budou vždy účelné (vyhnout se generickému generování komentářů). Komentáře budou v angličtině[16].

Pojmenování proměnných, metod, tříd, balíčků – pojmenování bude v angličtině a bude výstižně popisovat danou entitu. Jména budou formátovaná podle CamelCase.

Délka metod – metody nebudou (až na výjimečné případy) větší délky, než 30 řádků. To z toho důvodu, že jsou pak náročně pochopitelné.

Skladba tříd – metody budou sdružené do tříd tak, aby to dávalo smysl a aby třída pokaždé obsahovala zásadní funkčnost a nevznikali přebytečné třídy s minimem funkcí. Metody mající logickou souvislost budou sdružené v jedné třídě.

Umístění JPA anotací – JPA anotace budou pro přehlednost umístěny nad a tributy třídy a ne u GET a SET metod.

Folding GETrů a SETrů – v případě, že řádky obsahující tyto metody budou obsahovat pouze generickou funkcionalitu, budou zakomentovány způsobem, který umožní tyto řádky v IDE schovat.

Opakující se kód – v případě, že bude určité chování použito na více místech, bude sdruženo do opakovaně volané metody.

5. Používání cache

Při používání restauračního systému je velmi důležitá rychlá odezva systému. Jeden způsobů, jak jí dosáhnout, je použití cachování. To funguje tak, že si výsledky dotazů, u kterých očekáváme, že se budou opakovat, uložíme do cache (mezipaměti). Když následně dojde k jejich zavolání, místo toho, aby se dotazy musely znovu zpracovávat nebo odjinud vzdáleně načítat, načtou se přímo z cache. Tím dojde k úspoře času a výpočetních prostředků.

Tato kapitola popisuje, jak je možné řídit cachování na databázi a na serveru.

5.1. Řízení cachování na databázi

Použitá H2 databáze má vlastní (volitelně konfigurovatelnou) cache. V praxi to znamená, že si ukládá výsledky často volaných dotazů a v případě jejich volání dojde rovnou k jejich načtení z cache. Tuto funkcionalitu není třeba speciálně zapínat. Je zapnuta implicitně.

5.2. Řízení cachování na serveru

Nejvíce času při volání serveru zabírá komunikace s DB. Tento čas můžeme ušetřit tak, že u dotazů, u kterých čekáme časté volání, budeme cachovat jejich výsledky na serveru. Také tím ušetříme čas strávený zpracováním výsledků (jako je namapování na Java entity a jiné).

5.2.1. Strategie cachování

Play! k cachování používá svojí implementaci Ehcache [17]. Umožňuje ukládat Java objekty pod string klíčem a nastavit jim určitý timeout. V případě cachovaní požadavků na REST bude jako takový klíč sloužit URL dotazu, které je možno svázat s session ID requestu tak, aby výsledek mohl být jedinečně určený pro každého připojeného klienta.

Při cachování je potřeba hlídat, aby cachovaná data byla aktuální. To se zařídí tím, že každá metoda, která mění data v DB, bude mazat záznamy v cachi obsahující objekty na tyto data navázané. V praxi to znamená, že záznamy v cache vztahující se k jedné skupině dat uložené metodami GET budou smazány při použití metod POST, PUT a DELETE.

Při každém dotazu na objekty a REST API requesty, o kterých víme, že jsou cachované se nejdřív podívám do cache, jestli v ní jsou. Pokud ano, použijí se, pokud ne, provede se dotaz na databázi a výsledek se uloží do cache.

V průběhu testování bude vhodné sledovat vytížení paměti JVM při delší práci s aplikací, aby v důsledku ukládání cache dat nedošlo k jejímu naplnění.

5.2.2. Cachované objekty

Uživatel bude cachován podle session.
Při přihlášení se uživatel uloží.
Když dojde ke změně hesla, heslo se bude testovat oproti databázi.
Při odhlášení uživatele dojde k jeho odstranění z cache.

Když dojde k dotazu na otevřené účty, jejich seznam se uloží do cache.
Seznam otevřených účtů bude smazán z cache, když dojde k vytvoření nového účtu; přidání do seznamu otevřených účtů; přidání, odebrání, přesunutí nebo zaplacení jednotlivých položek nebo zaplacení účtu.

Když dojde k dotazu na otevřené položky menu, uloží se do cache.
Menu bude z cache smazáno, když dojde k jeho úpravě v nastavení - vymazání buňky, přidání menu, přidání položky, nastavení barvy nebo změně názvu.

5.2.3. Timeout cache

Objekt uživatele bude mít timeout. Jeho délka se bude načítat z konfiguračního souboru, kde bude zadána číselně v minutách. Pro prodloužení v cachi se bude objekt uživatele znovu ukládat při každém requestu. Všechny ostatní budou mít nastavený timeout na nekonečno.

5.2.4. Implementace

K Play cache přistupuji pomocí vlastní třídy CacheUtils (viz kód 1), kterou pomocí anotace @Include volám z ostatních tříd. Třída CacheUtils slouží jako rozhraní pro přístup do cache pro každou session zvlášť.

Kód 1: Třída CacheUtils . Soubor CacheUtils.Java.

```
/**
 * Serves as facade to objects in cache saved specifically for every
 * session.
 */
public class CacheUtils {

   private final int TIMEOUT = 60* Integer.parseInt(
      configUtils.getProperty("timeout"));
   @Inject ConfigUtils configUtils;
   @Inject SessionUtils sessionUtils;
   @Inject CacheApi cache;

   /**
    * Saving objects and strings to cache
    */
   public void setCacheObject(String key, Object object) {
      String fullKey = sessionUtils.getSessionKey() + "." + key;
      cache.set(fullKey, object, TIMEOUT);
   }

   /**
    * Saves string to cache
    * @param key    name of the string (choosable)
    * @param string value itself
```

```
     */
    public void setCacheString(String key, String string) {
        cache.set(sessionUtils.getSessionKey()
            + "." + key, string, TIMEOUT);
    }

    /**
     * Retrieving object from cache
     * @param key name of the object (choosable)
     * @param type value itself
     * @param <T>  class of object we expect to get
     * @return object itself
     */
    public <T extends Object> T getCacheObject(String key,
            Class<T> type) {
        String fullKey = sessionUtils.getSessionKey() + "." + key;
        return type.cast(cache.get(fullKey));
    }

    /**
     * Retrieving object from cache
     * @param key name of the object (choosable)
     * @param type class of object we expect to get
     * @param <T>  class of object we expect to get
     * @return object itself
     */
    public <T extends Object> T getCacheObjectAndRemove(String key,
            Class<T> type) {
        String fullKey = sessionUtils.getSessionKey() + "." + key;
        T object = type.cast(cache.get(fullKey));
        cache.remove(fullKey);
        return object;
    }

    /**
     * Removes object from cache
     */
    public void removeCacheObject(String key) {
        String fullKey = sessionUtils.getSessionKey() + "." + key;
        cache.remove(sessionUtils.getSessionKey());
    }
}
```

5.3. Využití cache pomocí HTML hlaviček

5.3.1. Návrh

HTML verze 1.1. umožňuje nastavit response HTML hlavičku, která říká, jak dlouho si má prohlížet držet response na URL. Pak když je na toto URL proveden request, prohlížeč ho neposílá na server, ale vrátí vlastní nacachovanou stránku. To použiji v případě statických stránek a zdrojů. Do zdrojů budou spadat soubory stylů, Javascriptu a obrázky.

V případě, že je stránka dynamická, ale bude s vysokou pravděpodobností opakovaně dotazovaná, budu používat ETagy [18]. Jako hodnota etagu bude sloužit hash objektů nebo parametrů způsobující dynamičnost stránky. Tuto hodnotu si při odesílání takové stránky uložím do cache pro danou session a URL. Devalidace proběhne v případě, že dojde ke změně objektů nebo parametrů určující stránku nebo odhlášení uživatele.

5.3.2. Implementace

Statické zdroje – css styly, Javascipt a obrázky jsou Playem cachované automaticky.

5.4. Shrnutí

Cachování můžeme implementovat především na dvou místech – při ukládání často používaných objektů používaných pro generovaní HTTP response a cachování celých HTTP response na straně klienta pomocí HTML hlaviček. Tím dosáhneme zrychlení systému.

6. Úprava použitého stávajícího kódu

Veškerý použitý kód z projektu CashBob bylo potřeba přepsat tak, aby byl použitelný v novém projektu. Zároveň bylo potřeba držet se nadefinovaných standardů kódování (viz kapitola 4.).

Největší změny proběhli při přepisování REST API, protože bylo potřeba přepsat všechny třídy tak, aby pracovali s Play! REST API.

6.1. Změny v použitých balíčcích

6.1.1. Změny v cz.cvut.fel.cashbob.model.dao

Všechny třídy service měly vlastní interface, který implementovaly. V nich byly nadefinované hlavičky metod. To bylo důležité v moment, kdy byl jejich životní cyklus řízen Springem a byly z nějakého místa volány přes dependency injection pomocí anotace @Autowired . Protože Play používá místo toho Java anotaci @Include a dependency injection má implementovanou způsobem, který interfacy nevyžaduje, nejsou anotace @Include dále potřeba a tak jsem je smazal.

Podobně jako v případě balíčku cz.cvut.fel.cashbob.controllers.svc jsem odstranil nadbytečné interfacy.

Dále jsem odstranil Spring anotace @Repository.

6.1.2. Změny v cz.cvut.fel.cashbob.model.conv

Podobně jako v případě balíčku cz.cvut.fel.cashbob.model.dao jsem odstranil nadbytečné interfacy a @Autowired anotace.

6.1.3. Změny v cz.cvut.fel.cashbob.model.entity

Kvůli přehlednosti jsem přesunul JPA anotace od GET metod k atributům entit.

Metody GET a SET jsem zakomentoval tak, aby se dali schovat pod jeden nadpis. Protože v nich většinou není žádná logika, není třeba je primárně vidět a můžou být schované.

6.1.4. Změny v cz.cvut.fel.cashbob.controllers.svc

Podobně jako v případě balíčku cz.cvut.fel.cashbob.model.dao jsem odstranil nadbytečné interfacy.

Z důvodu dalšího nepoužívání Springu jsem odstranil anotace @Service a @Transactional.

Dále jsem všechny atributy service, které odkazovali na třídy dao, covertor nebo service přepsal na private.

6.1.5. Vyčistění kódu od nepotřebných komentářů

Ve většině tříd byli komentáře členící třídu do více sekcí (viz kód 2). Byli ale i ve třídách, kde moc kódu nebylo, často k danému nadpisu nebyl kód žádny. Proto jsem je odstranil.

Kód 2: Příklad původních komentářů, co jsem odstranil ze stávajícího kódu.

```
// ======================== STATIC MEMBERS ========================
// ======================== INSTANCE MEMBERS ======================
// ======================== STATIC METHODS ========================
// ======================== CONSTRUCTORS ==========================
// ======================== OVERRIDEN METHODS =====================
// ======================== INSTANCE METHODS ======================
// ======================== PRIVATE METHODS =======================
// ======================== GETTERS / SETTERS =====================
```

6.1.6. Změny v cz.cvut.fel.cashbob.model.dao

Podobně jako v případě balíčku cz.cvut.fel.cashbob.model.dao jsem odstranil Spring dependency injection. Spring anotace @Transactional jsem nahradil stejnojmennými Play anotacemi a tak zajistil Playem vyžadovanou transakcionalitu dotazů volajících databázi.

6.2. Změny v REST API

6.2.1. Změny v REST API

Bylo nutné odstranit všechny Javax.ws.rs anotace (@Context, @Component). Mezi ně patřili i anotace @GET, @POST, @PUT, @DELETE a @Path, které ukazovali, které metody budou vystavené pod jakým URL a HTTP metodami (viz kód 3). Pak jsem je nahradil cestami v souboru routes, který přiřazování metod k URL řeší (viz kód 4). Zároveň k metodám obsahující přístup k databázi bylo třeba přidat anotaci @Transactional a injectovat potřebné atributy (viz kód 5).

Kód 3: Původní Javax.ws.rs anotace - soubor AccountApi.Java (stávající aplikace).

```
@Component
@Path(AccountApi.ACCOUNT_PATH)
public class AccountApi extends AbstractApi {

  public static final String ACCOUNT_PATH = "/account";

  @Autowired
  private IAccountService accService;

  @GET
  @Produces(RestConst.MIME_JSON)
  @Transactional
  public Response getAll() {
```

Kód 4: Příklad směrování - soubor routes.

```
GET  /rest/account
cz.cvut.fel.cashbob.controllers.rest.AccountApi.getAll()
```

Kód 5: Nové Play anotace - soubor AccountApi.Java.

```
@Transactional
public class AccountApi extends AbstractApi {

    public static final String ACCOUNT_PATH = "/account";

    @Inject private AccountService accService;
    @Inject private AccountConvertor accountConvertor;

    public Result getAll() {
```

6.2.2. URL requestů

Při tvorbě JSONů, které se používají jako odpověď na požadavky na REST API se často používá URL příchozího requestu. To bylo opět třeba přizpůsobit Play.

Po testování bylo třeba opravit bug, který vznikal, když byla metoda zavolaná z číselné url (např. když místo localhost obsahovala 127.0.0.1). V tu chvíli použité metody nedovedli předřadit řetězec „http://“. To bylo vyřešeno manuálním přidáním.

6.2.3. Mapování JSONů

Příchozí requesty v metodě POST obsahující objekty JSON, které slouží jako “parametry”, často reprezentují entity obsažené v systému. Při příjmu requestu je proto potřeba je na entity navázat, aby bylo možné využít jejich chování. Protože jde o opakovaně použitou funkcionalitu, vytvořil jsem si na zpracování takových požadavků metodu (viz kód 6).

Kód 6: Namapování JSON requestu na Java entitu – soubor AbstractApi.Java.

```
/**
 * Maps JSON object from request to DTO object.
 * @param type Class of Java DTO object on which we went to map JSON.
 * @return JSON object mapped on Java DTO.
 */
protected <T extends Object> T getFromJson(Class<T> type) {
    JsonNode json = request().body().asJson();
    if (json == null) {
        return null;
    }
    Object request = Json.fromJson(json, type.getClass());
    T object = type.cast(request);
    return object;
}
```

6.2.4. Mapování odpovědi

Jako odpověď na všechny REST requesty se posílá DTO objekt namapovaný na JSON. Podle toho bylo potřeba projít všechny rozhraní a přepsat return hodnoty tak, aby odpovídali Play (viz kód 7).

Kód 7: Namapování response DTO na JSON a jeho odeslání.

```
return ok(Json.toJson(responseDTO));
```

6.2.5. Odchytávání REST výjimek

Téměř všechny API mohou v případě problému vrátit chyby EntityException nebo ValidationException. Ty je potřeba odchytit a následně informovat uživatele, že k chybě došlo, případně jí specifikovat. Odchytávání takových výjimek je řešeno v třídě Global metodou onError. V ní zjišťuji, jestli se jedná o jednu z výše zmíněných chyb. Pokud ano, pošlu zpátky response DTO s chybovou hláškou (viz kód 8).

Kód 8: Odchytávání chyb na REST rozhraní – soubor Global.Java.

```
@Override
public Promise<Result> onError(Http.RequestHeader requestHeader,
Throwable throwable) {

   //<editor-fold desc="Odchytavani REST chyb">
   if(throwable instanceof EntityException){
      EntityException e = (EntityException) throwable;
      return Promise.<Result>pure(
        AbstractApi.getBadRequestResponse(
           e.getErrorCode(), e.getMessage()));
   }else if(throwable instanceof ValidationException){
      ValidationException e = (ValidationException) throwable;
      return Promise.<Result>pure(
        AbstractApi.getBadRequestResponse(
           e.getErrorCode(), e.getMessage()));
   }
   //</editor-fold>

   ...
}
```

6.3. Opravy chyb

6.3.1. Název parametru podle databáze

Při testování jsem narazil na chybu v moment, kdy bylo z databáze potřeba získat jídelní lístek. Jeho položku reprezentuje třída CustMenuNode. Problém byl v Named Queries, ve kterých se

filtrovalo podle **`c.available`** (jak je parametr pojmenován v databázi). Místo toho bylo třeba použít **`c.isAvailable`** (jak je parametr pojmenován v javě) (viz kód 9).

Kód 9: Přejmenování parametru v NamedQueries – soubor CustMenuNode.Java.

```
@NamedQueries({
    ...,
    @NamedQuery(name = "CustMenuNode.findByMenuId", query = "
       SELECT c FROM CustMenuNode c
       WHERE c.menu.id = :id AND c.isAvailable = TRUE"
    )
})
public abstract class CustMenuNode implements Java.io.Serializable,
IEntity {

    ...

    @Column(name = "AVAILABLE")
    protected boolean isAvailable = true;
```

6.4. Shrnutí

Použitý původní kód se podařilo přepsat tak, aby byl plně použitelný v projektu pro webový systém. Většina změn se týkala odstranění Spring anotací a jejich případném nahrazení anotacemi Play!. Další podstatné úpravy proběhly v REST API, které opět bylo potřeba přizpůsobit syntaxi Play! a zároveň udržet původní funkcionalitu. To se podařilo.

Doba odhadnutá na úkoly zpracované v této kapitole odpovídala reálnému času.

7. Zabezpečení

7.1. Autentizace

7.1.1. Analýza stávajícího řešení

Ve stávajícím řešení je implementovaná přihlašovací obrazovka zobrazující login formulář s tlačítky Přihlásit a Ukončit (viz obr. 1). Ta se hodí i pro webovou aplikaci, jenom bez tlačítka Ukončit. To ve stávající aplikaci ukončovalo program, na webové stránce již nemá smysl. Přihlášení je řešeno přes REST api.

Obr. 1: Přihlašovací obrazovka (stávající aplikace).

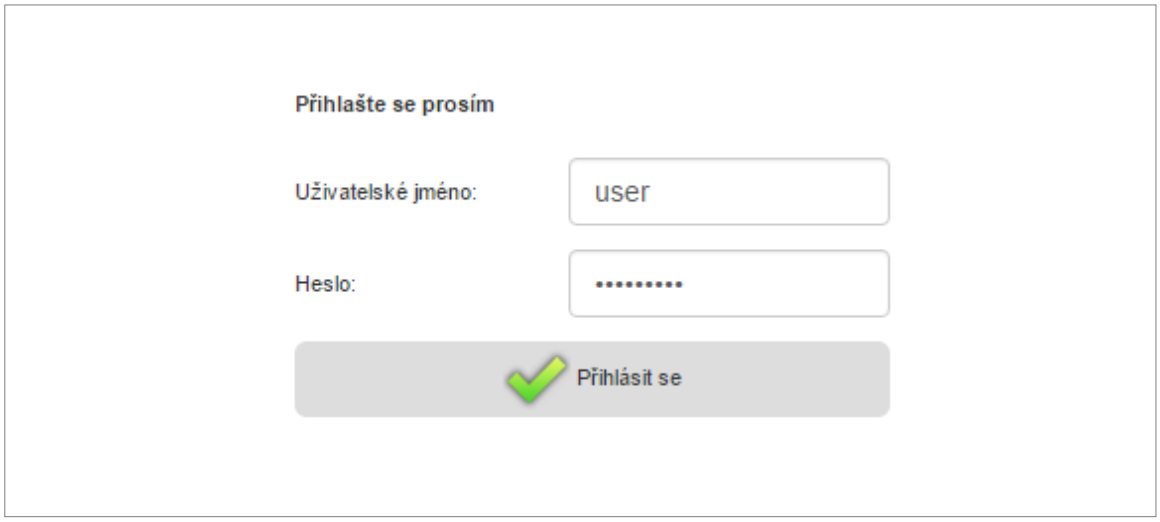
Login údaje se následně vyhodnocovali přes REST API oproti serveru. Tento způsob bude nahrazen standardním POST požadavkem následovaným reloadem stránky.

7.1.2. Implementace přihlášení

Na přihlašovací obrazovce jsem vytvořil formulář, který se vyhodnocuje metodou POST na URL /login (viz kód 10, obr. 2).

Kód 10: Login formulář (odstraněná část kódu zajišťující design) – soubor login.scala.html.

```
<form method="post" id="loginForm" action="/login">
    <label>@text("login.username")</label>
    <input type="text" name="name" >

    <label>@text("login.password")</label>
    <input type="password" name="password">

    <button type="button"
onclick="document.getElementById('loginForm').submit()">
    @text("login.button")
</form>
```



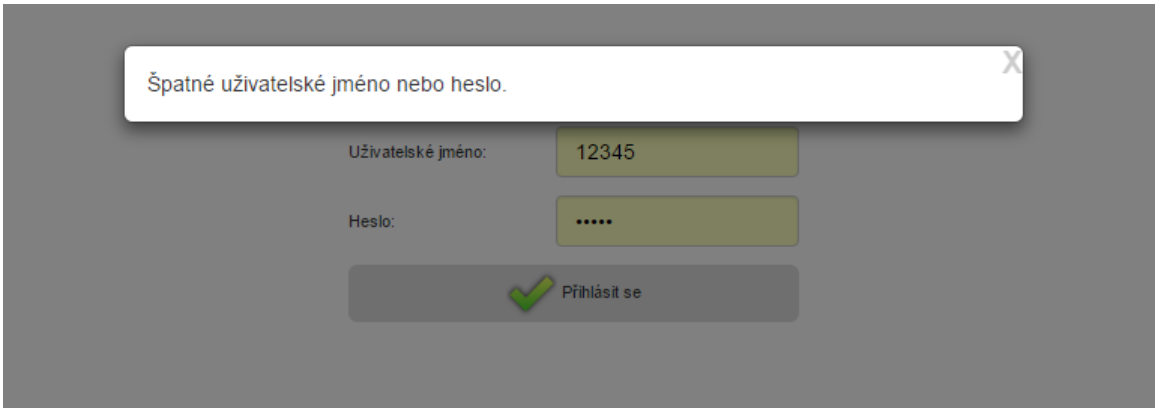
Obr. 2: Přihlašovací obrazovka (WA).

Ten je zpracovávaný metodou Login.loginUser(). Ta zajišťuje namapování login formuláře na objekt, který ho reprezentuje v Javě. V něm zkontroluji, že obsahuje vyplněné uživatelské jméno a heslo. Následně ve třídě SecurityUtils uživatele podle uživatelského jména načtu z databáze. Zadané heslo pomocí springové třídy BCryptPasswordEncoder zvaliduji oproti databázovému. V tuto chvíli vím, jestli uživatel zadal korektní login údaje.

Pokud uživatelské jméno nebo heslo neodpovídají, přesměruji uživatele na login stránku a vykreslím mu modální okno s hláškou, že zadané informace nejsou správné (viz obr. 3).

Pokud uživatel zadal korektní login údaje, uložím mu čas loginu do session (pod klíčem lastTime). Následně se mu podívám do session, kde může mít uloženou URL odkazující do zabezpečené části systému, na kterou se již před tím snažil dostat nebo ze které byl v důsledku

timeoutu odhlášen. Pokud systém tuto URL v session najde, přesměruje request na ní. V ostatních případech přesměruje na domovskou stránku.



Obr. 3: Modální okno s hláškou, že zadané login informace jsou chybné (WA).

7.1.3. Zabezpečení proti přístupu nepřihlášeného uživatele a timeout

Veškeré třídy kontrolující části systému pro přihlášené uživatele dědí od třídy Logged (viz kód 11). Ta je opatřena anotací, která volá metodu call(), která dále volá metodu callProblem() ve třídě CheckLogged (viz kód 12). Pokud přistupujeme do kterékoli metody třídy dědící od třídy Logged, je tato anotace zavolána.

V ní se kontroluje, jestli je uživatel zalogovaný - v metodě SecurityUtils.getLogged() se ověří, že je v session prohlížeče nastavený čas poslední akce přihlášeného uživatele (pod klíčem lastTime). Pokud není, uložím do session prohlížeče URL, na kterou se uživatel snažil dostat, abych ho na ní při příštím úspěšném zalogování přesměroval (viz 7.1.1.). Následně přesměruji na login stránku.

Dále kontroluji timeout uživatele. Ten ověřuji v metodě SecurityUtils.timeout() (viz kód 13). V ní znovu dívám na čas poslední akce přihlášeného uživatele (pod klíčem lastTime). Pak si spočítám, jak dlouhá doba uběhla od této poslední akce do teď. Dál si z konfiguračního souboru načtu dobu, po které má nastat timeout (položka pod klíčem timeout). Tyto dva časy porovnám a vyhodnotím, jestli k timeoutu došlo nebo ne. Pokud ano, uložím do session prohlížeče URL, na kterou se uživatel snažil dostat. Dále přesměruji na login stránku a v modální oknou vypíšu uživateli hlášku, že byl z důvodu timeoutu odhlášen.

Pokud bude po proběhnutí metody callProblem() uživatel stále zalogovaný, obnoví se na čas poslední akce přihlášeného uživatele (pod klíčem lastTime). Dále se v cache obnoví záznam s objektem uživatele, aby nedošlo k jeho odstranění z důvodu timeoutu cache.

Kód 11: Použití anotace s třídou CheckLogged – soubor Logged.Java.

```
@With(CheckLogged.class)
public class Logged extends Controller {
```

Kód 12: Metody call() a callProblem() – soubor CheckLogged.Java.

```
public Promise<Result> call(Http.Context context) throws Throwable {
    Promise<Result> resultPromise = callProblem(context);

    // If there is a problem
    if(resultPromise != null){
        return resultPromise;
    }

    // If everything is cool
    sessionUtils.setLastTime();
    logged.recacheUser();
    return delegate.call(context);
}

/**
 * If there is a problem, returns Result, where it's solved.<br>
 * If everything is ok, returns null.
 * Called from {@ling #call}
 */
public Promise<Result> callProblem(Http.Context context) throws
Throwable {

    // Checks logged
    if( !securityUtils.getLogged() ) {
        Logger.info("NO user => STOP");
        sessionUtils.setLastUrl(context._requestHeader().uri());
        System.out.println((context._requestHeader().uri()));
        return Promise.<Result>pure(login.loginPage());
    }

    // Checks timeout
    if( !securityUtils.timeout()) {
        Logger.info("Timeout => STOP");
        sessionUtils.setSessionAlert(
            textUtils.text("error.timeout"));
        sessionUtils.setLastUrl(context._requestHeader().uri());
        return Promise.<Result>pure(login.logout());
    }

    return null;
}
```

Kód 13: Metoda timeout() – soubor SecurityUtils.Java.

```
/**
 * Checks how long it is after last user action.<br>
 * Called from {@link CheckLogged#call(Http.Context)}.<br>
 * Warning: Doesn't do logout, just checks.
 * @return timeout = FALSE, ok = TRUE
 */
public boolean timeout() {
   long lastTime = sessionUtils.getLastTime();
   // If user is not logged, lastTime is 0, returning true
   if (lastTime == 0) {
      return true;
   }

   // Counts timeout
   long timeout = new Date().getTime() - lastTime;
   // Loading requested maximal timeout from configu
   String property = configUtils.getProperty("timeout");
   long maxTimeout = Integer.parseInt(property) * 1000 * 60;

   // If user is after timeout...
   if (maxTimeout < timeout) {
      Logger.info("Time difference is over "
         + timeout / 1000 / 60 + " minutes.");
      return false;
   }

   // If everything is allright...
   return true;
}
```

Při testování jsem narazil na problém, že po restartu serveru byl v session stále uložen čas poslední akce přihlášeného uživatele (ta se ukládá na klientovi), ale v cache (na serveru) již nebyl uložen přihlášený uživatel. To vyústilo v NullPointerException. Tento problém jsem vyřešil tím, že na začátek každého klíče, pod kterým si tento čas ukládám, přidávám náhodný řetězec. Ten je generován jednou při každém spuštění serveru (viz kód 14).

Kód 14: Práce s časem poslední akce přihlášeného uživatele v session - soubor SessionUtils.Java.

```
private static final String SERVER_RUN = RandomString.getRandom();

/**
 * If it's already generated
 * (saved in session session as "uuid"), returns session ID.
 * Else generates new one.
 * @return session ID
 */
public String getSessionKey() {
   String sessionID = session(SERVER_RUN + UUID);
   if (sessionID == null) {
      sessionID = Java.util.UUID.randomUUID().toString();
      session(SERVER_RUN + UUID, sessionID);
```

```
    }

    return sessionID;
}

/**
 * Sets current time in miliseconds.<br>
 * Called from {@link CheckLogged#call(Http.Context)}
 * (it means it's called every time user goes to secured page).
 */
public void setLastTime() {
    Date lastTime = new Date();
    session(getSessionKey()
        + LAST_TIME, String.valueOf(lastTime.getTime()));
}

/**
 * Called for example from {@link SecurityUtils#getLogged()}.
 * @return
 */
public long getLastTime() {
    String stringLastTime = session(getSessionKey() + LAST_TIME);

    if (stringLastTime == null) {
        return 0;
    }

    return Long.parseLong(stringLastTime);
}
```

7.2. Autorizace

7.2.1. Analýza stávajícího řešení

Ve stávající aplikaci byla autorizace řešená pomocí objektu rolí, které měl přidělený každý uživatel v libovolném množství. Následně byly jednotlivé service metody opatřena spring anotací @PreAuthorize s parametrem požadované role. Tento postup ve webové aplikaci nebude použit, protože se v ní nepoužívá spring.

7.2.2. Návrh implementace

Popis řešení

V rámci webové aplikace restauračního systému bude implementovaný systém autorizace. Jeho účelem bude omezení fungování uživatele v rámci restauračního systému. Omezení bude na úrovni vizuální, tedy že uživatel neuvidí v aplikaci prvky, které jsou svázány se zakázanou funkčností a na úrovni kódu, kde budou metody nebo třídy přístupné jenom uživatelům s určitou uživatelskou rolí.

V systému budou v kódu nadefinované uživatelské role. Budou definovány jako statické promněné typu short s přímo přiřazenou hodnotou. V databázi bude mít každý uživatel číselný atribut role. V jednotlivých anotacích bude hlídáno, bude hodnota role přistupujícího uživatele binárně sečtena s požadovanou rolí a výsledná hodnota pak oproti požadované roli porovnaná.

Pokud se hodnoty budou shodovat, uživatel bude puštěn dál, v opačném případě bude přesměrován na chybovou stránku.

Konkrétní role

V systému budou nadefinovány následující role:

číslo v desítkové soustavě (číslo v binární soustavě): Název role – práva role.

1 (0001): Číšník – tvořit účty, vytvářet účty a zavádět objednávky, přesouvat položky mezi účty.
3 (0011): Vrchní číšník – práva číšníka, zaplatit objednávku.
7 (0111): Admin 1 – práva vrchního číšníka, upravovat menu pokladny.
15 (1111): Admin 2 – práva admina 1, upravovat stoly, editovat položky menu podniku.

Změny v implementaci

V databázi budou odstraněny tabulky USER_ROLE, ROLE_RIGHT a ROLE.

Z tabulek TYPEWORKSHIFT, ROLE_DISCOUNTTYPE a USER musí být odstraněny závislosti na tabulku ROLE a USER_ROLE. Místo nich budou přidány atributy SMALLINT určující roli.

Tyto vazby musí být zároveň odstraněny v Java entitách reprezentující zmiňované tabulky. Proběhnou následující změny:

- Bude odstraněna třída Role.
- V třídě User bude atribut roles převeden na short.
- V třídě RoleDiscounttype bude atribut role převeden na short.
- V třídě Typeworkshift bude atribut role převeden na short.

Další změny budou muset proběhnout v následujících třídách, které obsahují závislosti na třídu Role: RoleDao, RoleService, RoleDiscounttype, TypeWorkshiftConvertor, UserApi, UserConvertor a WorkshiftService.

7.2.3. Implementace změny atributu

Databáze

Pro změnu databáze bylo potřeba přepsat její instalační script. Byly provedeny změny popsané v návrhu, totiž odstranění tabulek USER_ROLE, ROLE_RIGHT a ROLE. Dále byly odstraněny závislosti na tyto tabulky z tabulek TYPEWORKSHIFT, ROLE_DISCOUNTTYPE a USER. V nich byl místo nich přidán sloupec role (viz kód 15).

Kód 15: Sloupec ROLE – soubor Import_all.sql.

```
ROLE INTEGER NOT NULL
```

V tom došlo ke změně oproti návrhu (kde se počítalo s datovým typem SMALLINT). Bylo to nutné kvůli tomu, že se při použití datového typu SMALLINT pomocí JPA nedařilo namapovat

roli z databáze na Java atribut. Ten zůstával null. Při změně typu na INTEGER se problém vyřešil.

Zároveň bylo třeba odstranit importy dat do starých tabulek a upravit je vůči aktuálnímu databázovému schématu.

Java

Bylo třeba odstranit třídy týkající se role. To znamenalo odstranění třídy Role, nepoužívání třídy Rights a odstranění tříd, které je ovládaly. Odstranil jsem třídy RightsApi, RoleApi, RightConvertor, RoleConvertor, RoleDao a Role.

U dalších tříd bylo potřeba změnit atribut role tak, aby nepoužíval třídu Role, ale vlastní atribut Integer role (viz kód 16).

Kód 16: Atribut role.

```
@Column(name = "ROLE")
private Integer role;
```

Tyto změny se projevily v třídách AbstractApi , TemplateApi, TypeWorkshiftApi, UserApi, WorkshiftApi, TypeWorkshiftService, UserService, WorkshiftService, TypeWorkshiftConvertor, UserConvertor, RoleDiscounttype, Typeworkshift a User.

Změny se také odrazily do DTO objektů. Ty jsem doposud bral ze stávající knihovny CashBobRestLib, ve které nešly upravovat. Musel jsem je zkopírovat do aktuálního projektu (do balíčku cz.cvut.fel.cashbob.model.dto) a zde je upravit. Tím došlo ke značnému snížení závislosti na knihovně CashBobRestLib. Po přesunutí třídy JsonDateAdapter do aktuálního projektu (do balíčku cz.cvut.fel.cashbob.utils) bylo možné tuto knihovnu úplně odstranit.

7.2.4. Ověření role přihlášeného uživatele

Zabezpečení metod je řešeno vlastní anotací CheckRoleAnn, která volá třídu CheckRole. Anotace check role má jako parametr integer (viz kód 17).

Kód 17: Anotace CheckRoleAnn – soubor CheckRoleAnn.

```
/**
 * Ref. {@link CheckRole}.
 */
@With(CheckRole.class)
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface CheckRoleAnn {
    int value();
}
```

V třídě CheckRole je zavolaná metoda call, která načte hodnotu zadanou v anotaci (požadovaná role) (viz kód 18). Na tu použije binární AND spolu s rolí přihlášeného uživatele. Pokud se výsledná hodnota rovná požadované roli, vše je v pořádku a request se forwarduje. Pokud uživatel nemá dostatečná práva, je přesměrován na login stránku a vypíše se mu modální okno s chybovou hláškou.

Kód 18: Anotace CheckRoleAnn – soubor CheckRoleAnn.

```
/**
 * Checks if user's role includes required role. <br>
 * If it is, forward action,<br>
 * else redirects to login page.
 */
public class CheckRole extends Action<CheckRoleAnn> {

   @Inject SessionUtils sessionUtils;
   @Inject TextUtils textUtils;
   @Inject private CacheUtils cacheUtils;

   public Promise<Result>call(Http.Context context) throws Throwable{
      // Loads required user role for method and current user's type
      int requiredRole = configuration.value();
      int currentRole = cacheUtils.getCacheObject(
         "user", User.class).getRole();

      // Checks access
      if ((currentRole & requiredRole) == requiredRole) {
         return delegate.call(context);
      }else {
         sessionUtils.setSessionAlert(
            textUtils.text("error.notAllowed"));
         return Promise.<Result>pure(redirect(Login.URL));
      }
   }
}
```

Příklad použití anotace je v kódu 19. V tomto případě pokud přistoupí uživatel, jehož role nemá poslední 4 bity ve formě 1111 (tzn. 15 dekadicky), systém ho přesměruje na login stránku. Pokud má dostatečnou roli, bude přesměrován na stránku, ve které se mu vypíše požadovaná hláška.

Kód 19: Použití anotace CheckRoleAnn.

```
@CheckRoleAnn(15)
public Result test2(){
    return ok("You shall pass!");
}
```

7.3. Cross-site request forgery (CSRF)

CSRF je druh útoku na internetovou aplikaci, proti kterému se bránit přidáním unikátního řetězce (tokenu) pro každý požadavek, ve kterém uživatel odesílá nějaká data na server.

Play tento problém řeší pomocí třídy CSRFFilter, kterou je možné volat při každém requestu automaticky (viz kód 20). Token je také nutné přidat ručně do všech HTML formulářů (viz kód 21).

Kód 20: Přidání CSRF filtru pro každý požadavek – soubor Filters.Java.

```
/**
 * Filters - classes, which are called on every request.<br>
 * This class must be configured in file conf/application.conf.
 */
public class Filters implements HttpFilters {

    @Inject CSRFFilter csrfFilter;
    ...

    /**
     * Adding filters.
     * This method is called by Play core.
     */
    @Override
    public EssentialFilter[] filters() {
        return new EssentialFilter[]{csrfFilter,...};
    }
}
```

Kód 21: Příklad - přidání CSRF tokenu do login formuláře.

```
<form method="post" id="loginForm" action="/login">
    @CSRF.formField
    ...
</form>
```

7.4. Testování

V rámci testování se prochází aplikace pomocí selenium scriptu (soubor security.html) způsobem popsaným níže. Testování probíhá za použití předem vytvořeného uživatele user1, heslo 12345, role 1.

1. Úspěšné přihlášení.
   1.1. Přechod na login stránku.
   1.2. Vyplnění login údajů.
   1.3. Klepnutí na tlačítko přihlásit.
   1.4. Zkontrolování úspěšného přihlášení.
2. Úspěšné odhlášení.
   2.1. Klepnutí na link Odhlásit se.
   2.2. Přechod na domovskou stránku (ta je dostupná pouze po přihlášení).
   2.3. Zkontrolování, že domovská stránka je nedostupná a prohlížeč je stále na login stránce.
3. Neúspěšné přihlášení.
   3.1. Vyplnění chybných login údajů.
   3.2. Klepnutí na tlačítko přihlásit.
   3.3. Zkontrolování, že se zobrazil modální okno s chybovým hlášením.
4. Autorizace
   4.1. Provedení všech kroků z bodu 1.
   4.2. Přechod na testovací stránku vyžadující oprávnění 15.
   4.3. Zkontrolování, že se zobrazil modální okno s chybovým hlášením.

Testy proběhly úspěšně.

7.5. Shrnutí

Podařilo se aplikaci zabezpečit tak, aby do jejích částí, které jsou dostupné jenom pro přihlášené uživatele nemohli přistupovat nepřihlášení uživatelé. Dále jsem přepsal databázové schéma a upravil Java třídy tak, aby podporovali nově navržené role a vytvořil anotaci, která umožní hlídat, aby se přihlášení uživatelé dostali pouze do částí systému jim určeným.

Doba odhadnutá na úkoly zpracované v této kapitole byla 16 hodin. Reálný čas strávený na nich byl přibližně 40 hodin. Čas je odhadnutý velmi hrubě, protože ve fázi, kdy bylo zabezpečení implementováno jsem zároveň pracoval na dalších částech práce. Jeden z důvodu přetažení časového odhadu byl, že při plánu jsem zapomněl započítat čas strávený na CSRF.

8. Domovská stránka

8.1. Analýza stávajícího řešení a návrh

Ve stávající aplikaci je domovská stránka, která slouží jako rozcestník mezi základními funkcemi aplikace. Jsou v ní odkazy, které otevírají okna Pokladny, Správy podniku, Správy skladu, Plánování směn, Nastavení, Nastavení práv a dál tlačítko pro odhlášení se z aplikace (viz obr. 4).

Tuto stránku se budu snažit v co největší míře zachovat. Tlačítka nebudou otevírat nová okna, ale odkazovat na URL na konkrétní stránky.

Obr. 4: Obrazovka domovské stránky (stávající aplikace).

8.2. Implementace domovské stránky

Stránka je implementovaná jako seznam tlačítek, do kterých je pomocí tagu img vložen obrázek (převzatý ze stávající aplikace) (viz obr. 5).

Obr. 5: Obrazovka domovské stránky (WA).

8.3. Shrnutí

Domovskou stránku se povedlo bez problémů nasadit.

Na tvorbu domovské stránky v časovém plánu nebyl specificky vyhrazen žádný čas. Aktivitami popsanými v této kapitole jsem strávil přibližně 2 hodiny.

9. Seznam účtů a tvorba nového účtu

Tato kapitola popisuje vytvoření šablony pro stránky poklad (především řádka s menu) a vytvoření stránek pro zobrazení seznamu otevřených účtů a tvorby nového účtu.

9.1. Analýza stávajícího řešení pokladny

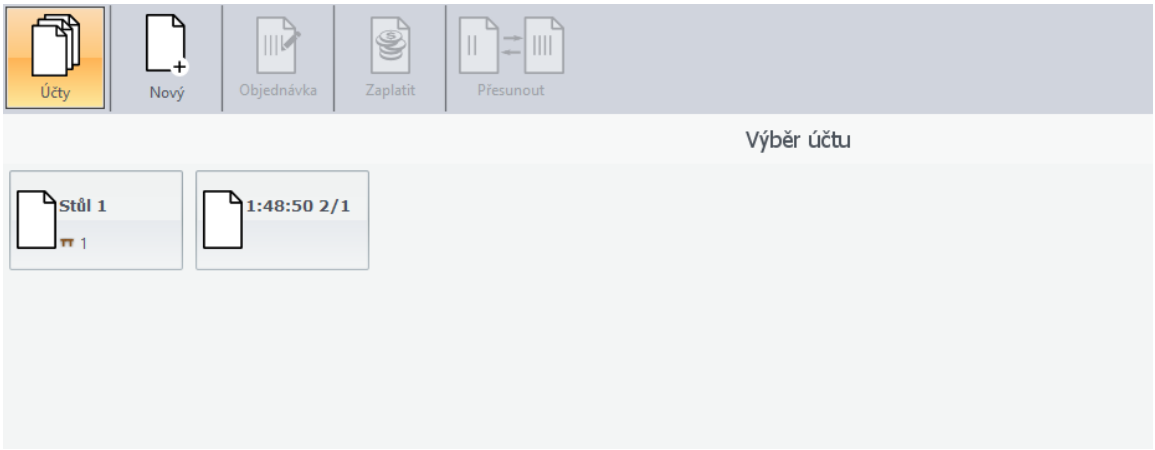
9.1.1. Pokladna

Ve stávající aplikaci se pro Pokladnu otevře nové okno. V něm je nahoře umístěná řádka s možnostmi zobrazení nezaplacených účtů, tvorby nového účtu, zobrazení otevřeného účtu, zaplacení otevřeného účtu a přesunutí položek z otevřeného účtu na jiný účet. Dál vrchní řádka umožňuje rozbalení menu nastavení, ve kterém je Editace menu a Bilance.

Aktuální menu (ve kterém se zrovna nacházíme je zvýrazněný a menu, na které se najede myší, se také zvýrazní. Menu, která nejsou použitelná (v případě, že například není otevřená žádná objednávka) jsou zašedlá a nereagují.

9.1.2. Seznam účtů

Pod menu Účty se zobrazí obrazovka pro výběr účtu. V ní je seznam tlačítek, která reprezentují jednotlivé objednávky, případně zobrazí stůl, ke kterému je objednávka přiřazena (viz obr. 6).



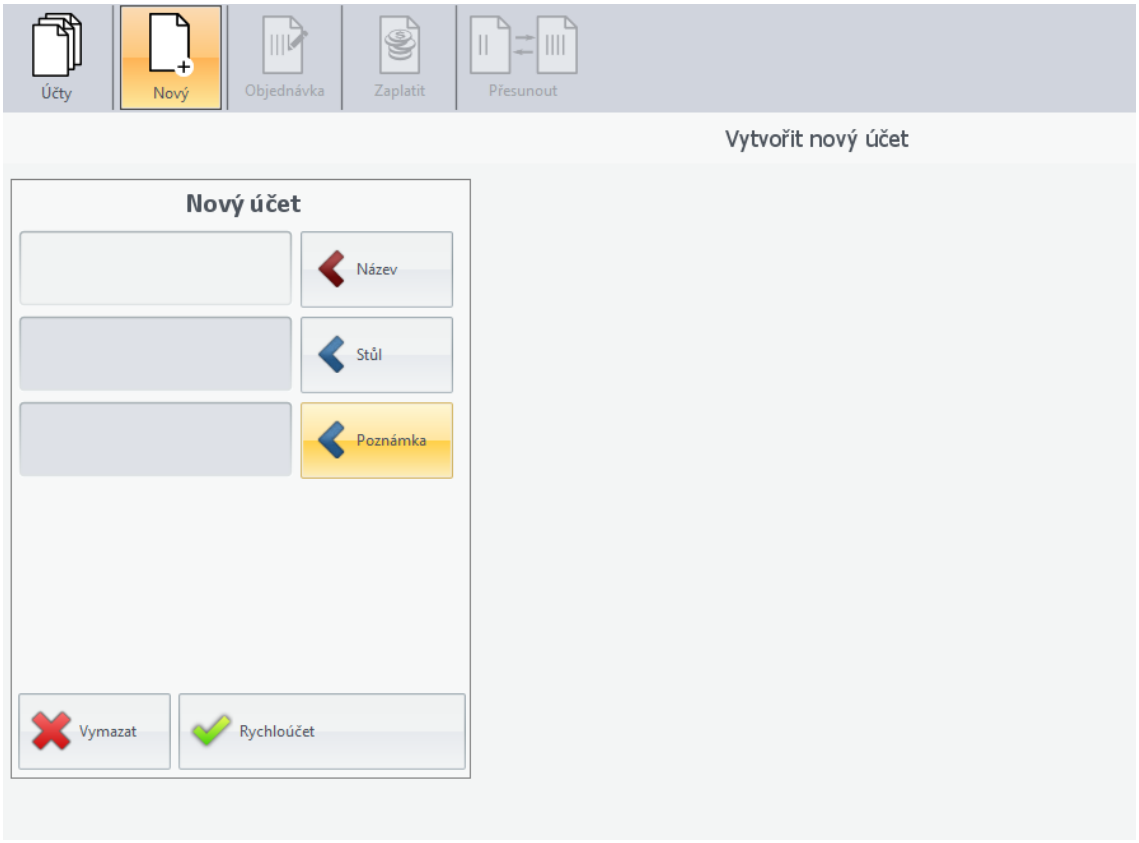
Obr. 6: Obrazovka seznamu účtů (stávající aplikace).

9.1.3. Tvorba nového účtu

Pod menu Nový se vytváří nový účet. Obrazovka, která mu náleží obsahuje input texty pro název účtu, stůl, ke kterému náleží a nabízí připsat poznámku (viz obr. 7). Zároveň umožňuje vytvořit tzv. rychloúčet. Ten se hodí například u baru, kde zákazník restaurace nepatří k žádnému stolu.

Pokud při tvorbě účtu nedojde k vyplnění názvu účtu, dojde automaticky k pojmenování podle stolu, ke kterému je účet navázán.

Při výběru stolu se po kliknutí na tlačítko zobrazí modální okno, ve kterém jsou zobrazeny všechny stoly.



Obr. 7: Obrazovka tvorby nového účtu (stávající aplikace).

9.2. Návrh

9.2.1. Pokladna

Oproti stávající aplikaci se ve webové aplikaci pro pokladnu nebude otevírat nové okno, dojde pouze k přesměrování. Menu bude zachováno.

9.2.2. Seznam účtů

V seznamu účtu nedojde k žádným změnám.

9.2.3. Tvorba nového účtu

Veškerá funkční část stránky nového účtu bude zachovaná. Protože tlačítka Název a Poznámka sloužily pro zobrazení digitální klávesnice na obrazovce, což ve webové aplikaci není potřeba, budou odstraněna.

Ve webové aplikaci bude odstraněn pojem Rychloúčet. Potvrzovací tlačítko bude zobrazovat text Vytvořit účet po celou dobu. V případě, že pole jména účtu nebude vyplněno, dojde k jeho

automatickému vyplnění aktuálním datem a časem. Pokud nebude vyplněno číslo stolu, nedojde k přiřazení žádného stolu k účtu.

9.3. Implementace

9.3.1. Šablona stránek v pokladně

Protože část všech stránek pokladny je stejná, používám pro ně šablonu, do které vkládám obsah jednotlivých stránek pokladny a jejich nadpis.

Tuto šablonu vkládám do hlavní šablony všech stránek (soubor main.scala.html), včetně souboru se styly. Ten je jeden pro všechny stránky pokladny. Dále šablona obsahuje řádku s menu.

Pomocí Javascriptu pak zvýrazňuji na každé stránce pokladny aktuální menu (viz kód 22).

Kód 22: Příklad – zvýraznění tlačítka Účty – soubor accountList.scala.html.

```
<script>
    $(document).ready(function() {
        $('#cashdesk').addClass("top-row-icon-active")
    });
</script>
```

Tlačítka nastavení v pravé části řádky menu jsou defaultně schovaná a po kliku na tlačítko Nastavení se pomocí jQuery funkce toggle() přepíná jejich viditelnost. Po načtení stránky nastavení je třeba tyto tlačítka manuálně zobrazit. (viz kód 23)

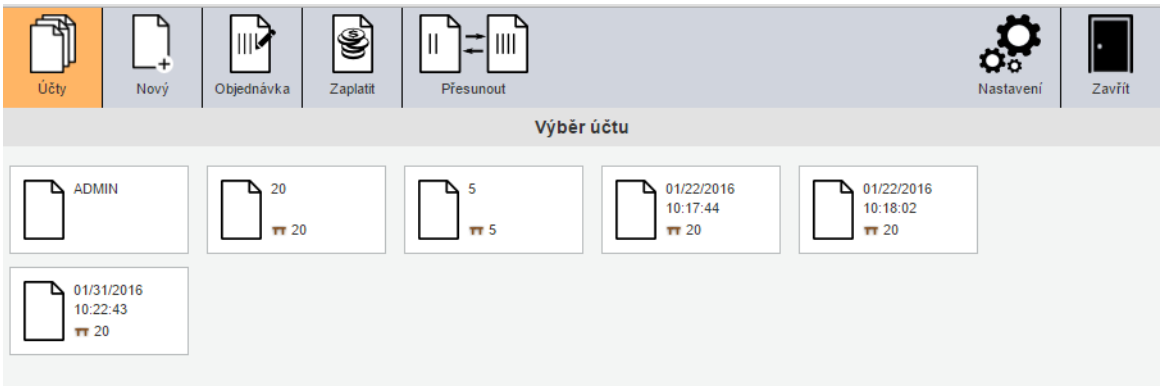
Kód 23: Schování a zobrazení tlačítek nastavení – soubor cashdesk.js.

```
/**
 * ACCOUNTS ALL
 * Toggles visibility of Settings menu.
 */
$(document).ready(function() {
    $('#settings').click(function() {
        $('#editTable').toggle("slide");
        $('#editMenu').toggle("slide");
    });
});
```

9.3.2. Seznam účtů

Stránka seznamu všech otevřených účtů a možnosti prokliknout se na jejich detaily je velice jednoduchá (viz obr. 8). V controlleru se zavolá servisní metoda pro získání seznamu všech otevřených účtu.

Tento seznam se jako parametr předá do šablony, ve které se pomocí cyklu vypíše jako tabulka tlačítek odkazujících na detaily účtů. Tlačítka se po najetí myši zvýrazní oranžově. V tlačítku je zobrazen název účtu, případně číslo stolů, ke kterému je účet přiřazený.



Obr. 8: Obrazovka seznamu otevřených účtů (WA).

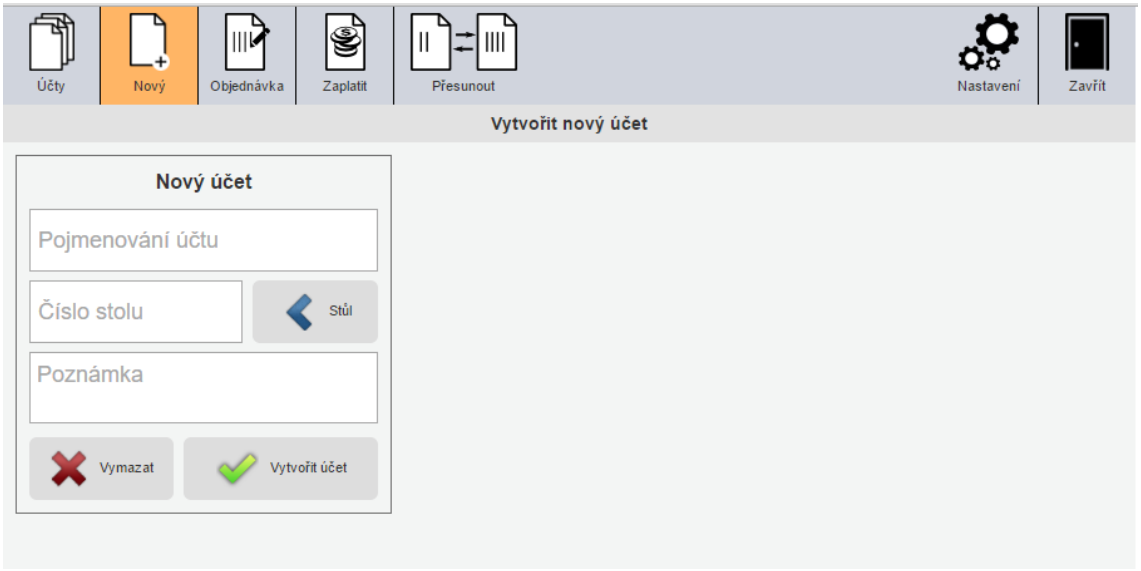
9.3.3. Tvorba nového účtu – původní implementace

Při generování stránky pro vytvoření nového účtu si do Javascriptové proměnné uložím seznam všech stolů v restauraci převedený do JSON objektu. Tento objekt využívám při odesílání formuláře (viz níže).

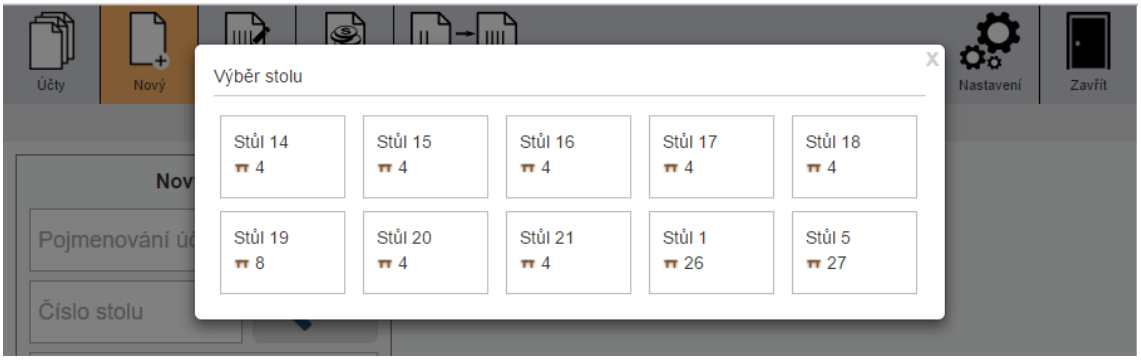
Stránka obsahuje formulář s třemi inputy, pro jméno účtu, číslo stolu a poznámku. Ani jedno z polí není povinné (viz obr. 9).

Číslo stolu je možné zadat manuálně nebo je možné kliknout na tlačítko pro výběr stolu. Po kliknutí na něj se zobrazí modální okno se zobrazením tlačítek reprezentující stoly v restauraci (viz obr. 10). Po vybrání stolu se pomocí Javascriptu zapíše číslo vybraného stolu do příslušného inputu. Následně se zkontroluje, jestli je zadán nějaký text v inputu pro jménu účtu. Pokud není, vyplní se do něj text ve formátu Stůl č. X.

Po kliknutí na tlačítko Vymazat se pomocí Javascriptu formulář vymaže.



Obr. 9: Obrazovka tvorby nového účtu – původní implementace (WA).



Obr. 10: Obrazovka výběru stolu (WA).

Při zpracovaní formuláře v controlleru je třeba ID stolu, ne jeho čísla, které se zadává do inputu. Proto před odesláním přidám do formuláře input element, do kterého vyplním ID stolu (viz kód 24).

Kód 24: Přidání input elementu s ID stolu – soubor cashdesk.Java.

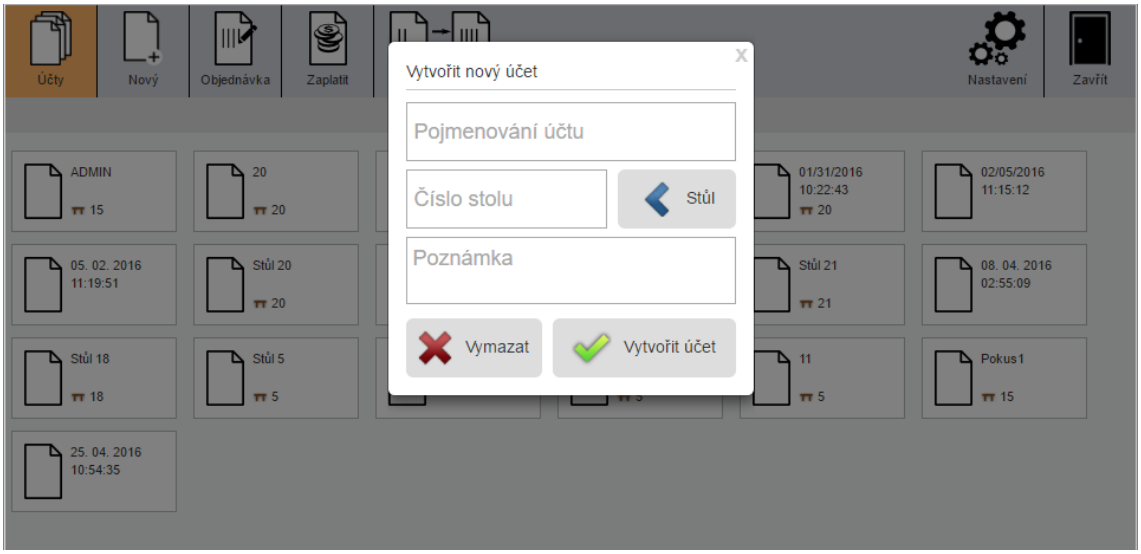
```
/**
 * ACCOUNTS NEW
 * Submits form, which serves for creating new account.
 * Adds input with table ID.
 */
$(document).ready(function() {
    // Fast account
    $('#newButtonFast').click(function() {
```

```
        var table;
        var tableNumber = $('#tableNumber').val();
        tables.forEach(function(t){
            if(t.tablenumber == tableNumber){
                var input = $("<input>")
                    .attr("type", "hidden")
                    .attr("name", "tableId").val(t.id);
                $('#newForm').append($(input));
                return;
            }
        });
        $('#newForm').submit();
    });
});
```

V controlleru se provede kontrola, jestli je vyplněné jméno účtu. Pokud není, vyplní se do něj aktuální datum a čas. Následně je účet vytvořen. Pokud vytvoření proběhlo úspěšně, proběhne přesměrování na detail účtu. Pokud nastala chyba, proběhne přesměrování na stránku tvorby nového účtu a vypíše hláška, že došlo k chybě.

9.3.4. Tvorba nového účtu – finální implementace

Při implementaci Přesunutí položek mezi účtu (viz kapitola 13.) jsem zjistil, že bych mohl znovu použít formulář pro tvorbu nového účtu v modalovém okně. Proto jsem odstranil samostatnou stránku pro tvorbu nového účtu a kód z ní přenesl do šablony všech stránek pokladny. Díky tomu je možné, že když se kdekoli v pokladně stiskne tlačítko pro nový účet, zobrazí se modalové okno pro tvorbu nového účtu (viz obr. 11).



Obr. 11: Obrazovka tvorby nového účtu – finální implementace (WA).

9.4. Testování

V rámci testování se prochází aplikace pomocí selenium scriptů (soubory newAccount.html, accountList.html) způsobem popsaným níže. Testování probíhá za použití předem vytvořeného uživatele user1, heslo 12345, role 1.

9.4.1. Test tvorby nového účtu

1. Přechod na stránku tvorby nového účtu.
    1.1. Přechod na login stránku.
    1.2. Vyplnění login údajů.
    1.3. Přihlášení.
    1.4. Přechod do pokladny.
    1.5. Přechod na stránku tvorby nového účtu.
2. Tvorba účtu s detaily.
    2.1. Výběr stolu (kliknutí na tlačítko stůl, vybrání prvního tlačítka).
    2.2. Kontrola vyplnění jména účtu (Stůl X).
    2.3. Přepis jména účtu.
    2.4. Vyplnění poznámky.
    2.5. Vytvoření účtu (proběhne přesměrování na detail).
    2.6. Kontrola jména a čísla stolu účtu.
3. Tvorba účtu bez detailů.
    3.1. Přechod na stránku tvorby nového účtu.
    3.2. Vytvoření účtu (proběhne přesměrování na detail).
    3.3. Kontrola vygenerovaného jména účtu.

9.4.2. Test výpisu všech otevřených účtů

Tento test navazuje na test tvorby nového účtu (počítá s tím, že je vytvořený alespoň jeden otevřený účet).

1. Přechod na stránku tvorby nového účtu.
    1.1. Přechod na login stránku.
    1.2. Vyplnění login údajů.
    1.3. Přihlášení.
    1.4. Přechod do pokladny (vede na stránku seznamu účtů).
2. Kontrola přechodu do detailu účtu.
    2.1. Kliknutí na tlačítko účtu.
    2.2. Kontrola, že došlo k přechodu na detail účtu.

Testy proběhly úspěšně.

9.5. Shrnutí

V rámci této iterace se úspěšně podařilo zanalyzovat, implementovat, otestovat a nasadit stránku pro tvorbu nového účtu a stránku se seznamem otevřených účtu. Oproti stávající verzi došlo na stránce pro tvorbu nového účtu k odstranění tlačítek pro vyvolání virtuální klávesnice. Zároveň dochází ke správnému řešení nevyplněných polí jména účtu a čísla stolu (viz 10.3. Návrh tvorby nového účtu).

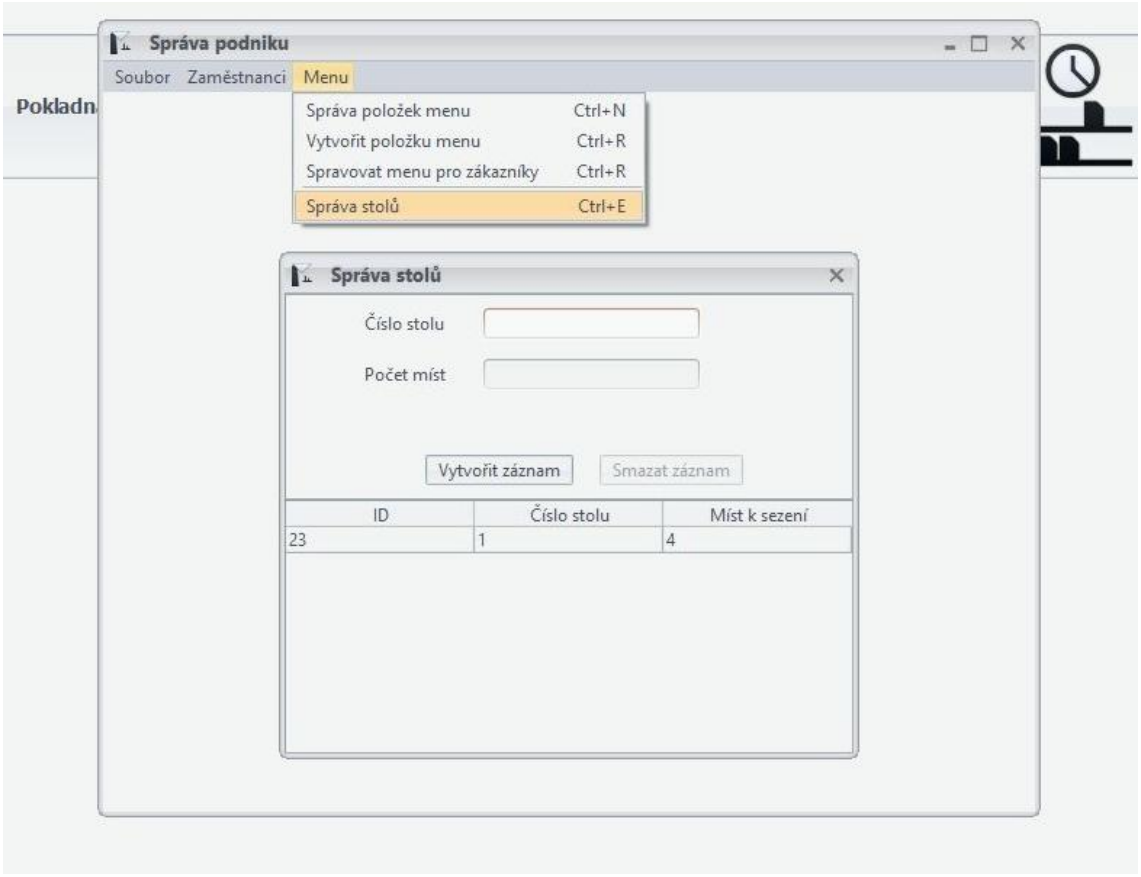
Na aktivity popsané v této kapitole bylo vyhrazeno 8 hodin. Reálný čas strávený na nich byl přibližně 22 hodin. Přesáhnutí bylo způsobeno tím, že v rámci této kapitoly jsem implementoval šablonu všech stránek v pokladně (které spadají do jiných kapitol). Ve shrnutí následujících kapitol je vidět časová úspora na ostatních stránkách.

10. Správa stolů

Stávající aplikace umožňuje přidružovat objednávky k jednotlivým stolům. Tyto stoly je potřeba spravovat – vytvářet, upravovat a mazat. Tato kapitola řeší tvorbu prostředí, které to umožní.

10.1. Analýza stávajícího řešení

Ve stávající aplikaci se do Správy stolů dostaneme přes hlavní menu > Správa podniku > Menu > Správa stolů. V otevřeném okně vidíme seznam existujících stolů, jejich ID, čísla a počet míst k sezení. Tyto stoly je možné upravovat a mazat. Dál zde můžeme vytvořit nový stůl (viz obr. 12).



Obr. 12: Obrazovka Správa stolů (stávající aplikace).

10.2. Návrh

Správu stolů jsem se rozhodl změnit, protože okno, ve kterém se provádí, nemá dostatečně intuitivní ovládání. Rozhodl jsem se místo řádku v tabulce pro Správu stolů vytvořit stránku v Pokladně v Nastavení (na stejné úrovni jako úprava menu). Zde se každý stůl zobrazí jako

tlačítko s ikonkou stolu. Po kliknutí se zobrazí modální okno, kde je možné stůl upravit nebo smazat. Dál se na stránce zobrazí tlačítko pro přidání nového stolu.

10.3. Implementace editace stolů

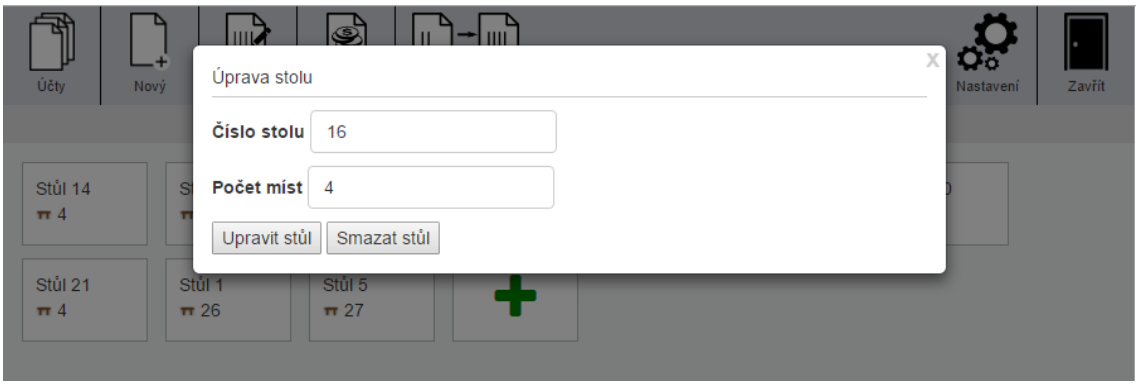
Při generování stránky pro editaci stolů si načtu seznam všech stolů, převedu si ho do DTO a v Javascriptu uložím jako JSON do proměnné tables. Seznam stolů vypíšu jako seznam tlačítek.

Na konec seznamu přidám tlačítko pro tvorbu nového stolu (viz obr. 13).

Obr. 13: Obrazovka Editace stolů (WA).

Po kliknutí na tlačítko stolu se zobrazí modální okno s jeho detailem (viz obr. 14). Pomocí Javascriptu se do něj z proměnné tables načteno číslo stolu a počet míst. Do proměnné tableIndex se uloží ID stolu. To se pak používá při kliknutí na tlačítko smazat stůl. Také se při otevření modální oknou změní atribut action tak, aby zahrnoval ID otevřeného stolu.

Po kliknutí na tlačítko ‚Smazat stůl' prohlížeč zažádá o potvrzení akce, aby nedošlo ke smazání stolu omylem. V controlleru se testuje, jestli na stůl nejsou navázané otevřené účtu. Pokud ano, uživateli se zobrazí hláška, že stůl nejde smazat.



Obr. 14: Obrazovka Úprava stolu (WA).

10.4. Testování

V rámci testování se prochází aplikace pomocí selenium scriptu (soubor editMenu.html) způsobem popsaným níže. Testování probíhá za použití předem vytvořeného uživatele user1, heslo 12345, role 1.

1. Přechod na stránku Editace stolů.
   1.1. Přihlášení
   1.2. Přechod na stránku Pokladny.
   1.3. Přechod na stránku Editace stolů.
2. Tvorba stolu.
   2.1. Kliknutí na tlačítko Plus.
   2.2. Vyplnění čísla stolu.
   2.3. Vyplnění počtu míst.
   2.4. Tvorba stolu.
   2.5. Ověření, že se nový stůl zobrazil.
3. Editace stolu.
   3.1. Otevření Úpravy stolu.
   3.2. Upravení čísla stolu.
   3.3. Upravení počtu míst.
   3.4. Upravení stolu.
   3.5. Ověření, že byl stůl upraven.
4. Smazání stolu.
   4.1. Otevření detailu stolu.
   4.2. Smazání stolu.
   4.3. Potvrzení smazání stolu.
   4.4. Ověření, že byl stůl smazán.

10.5. Shrnutí

Stránku editace stolů se podařilo se implementovat tak, aby působila intuitivně. Stránka umožňuje stoly tvořit, upravovat a mazat. Při mazání hlídá je ohlídaná integrita dat v databázi.

Čas strávený na aktivitách popsaných v této kapitole byl v plánu odhadnut na 8 hodin, což přibližně odpovídá reálnému času.

11. Menu

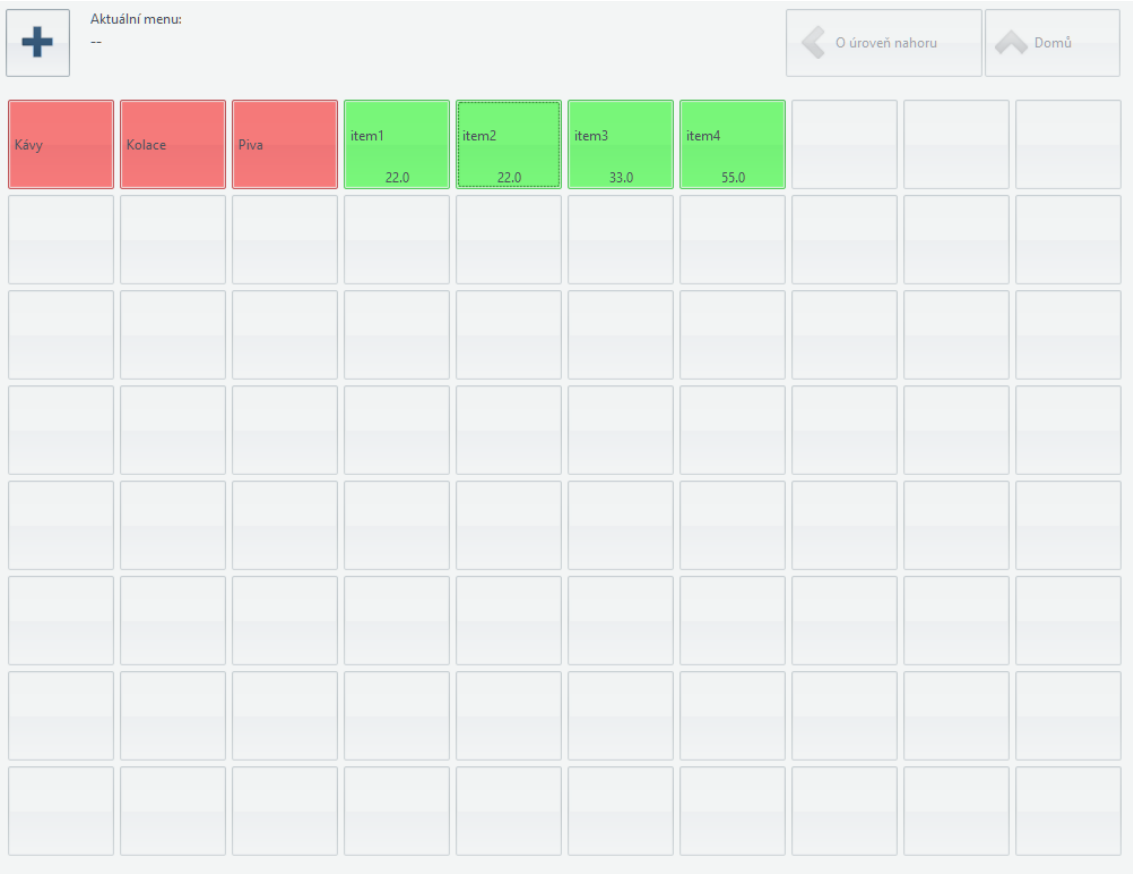
Tato kapitola se věnuje práci s jídelním lístkem neboli menu restaurace.

11.1. Analýza

Stávající aplikace obsahuje menu 10 krát 10 tlačítek (viz obr. 15). Tlačítka reprezentují buď skupiny položek menu, nebo položky samotné. Při kliknutí na skupinu položek se přejde do vybrané skupiny a zobrazí se skupiny a položky menu, které jsou její podmnožinou. Při kliknutí položku menu se položka přidá do objednávky. Všechna tlačítka mají svojí barvu, pozici (X, Y) v poli tlačítek a název. Položky navíc obsahují cenu.

Nad menu je tlačítko +, pomocí kterého lze jednorázově přidat položku, která není v menu, do objednávky. Položka se uloží do seznamu položek podniku, nezobrazuje se však v menu.

Nad menu jsou tlačítka umožňující pohyb v něm. V případě vstupu do skupiny položek tlačítko O úroveň výše ze skupiny vystoupí. Tlačítko Domů zobrazí hlavní menu podniku (tzn. takové, jehož skupiny a položky nepatří do žádné skupiny položek).



Obr. 15: Obrazovka detailu účtu, výřez menu (stávající aplikace).

11.2. Návrh

11.2.1. Přenos menu

Načítání stránky obsahující menu a samotný průchod menu je třeba optimalizovat tak, aby uživateli připadal okamžitý. Toho bude dosaženo cachováním menu na straně serveru při načítání dat z databáze. Samotné načtení stránky trvá 100 až 200 ms, objem dat obsahující menu je zanedbatelné a tak není třeba ho na straně prohlížeče nijak cachovat. Po přijetí stránky prohlížečem bude načtené celé menu prohlížečem a překreslování při průchodu menu pomocí Javascriptu bude probíhat okamžitě a není třeba dále optimalizovat.

11.2.2. Rozložení menu

Při navrhování rozložení menu pro webovou aplikaci jsem se rozmýšlel, jestli při zmenšení stránky zmenšit počet tlačítek menu v jedné řádce. Tak by se dosáhlo toho, že i při zobrazení v menších oknech, např. na mobilních zařízeních bude celé menu zobrazené a tlačítka na něm budou v příjemné velikosti. Z důvodu zachování funkčnosti pozice tlačítek, která umožňuje nemít všechny položky menu hned za sebou, ale na předem dané X, Y pozici, tento způsob rozložení menu nebudu neimplementovat. Menu bude optimalizované pro minimální šířku prohlížeče větší, než 1250px.

CSS umožňuje zmenšovat a zvětšovat velikost tlačítek při změně velikosti okna. Díky tomu budou tlačítka vždy v největší možné velikosti tak, aby se vešla na stránku. Text v tlačítkách bude zalamován. V případě, že se text na řádku nevejde, bude jeho konec nahrazen třemi tečkami „…".

11.2.3. Vykreslování

Při návrhu vykreslování menu a objednávání položek z něj jsem zjistil, že psát zpracování všech akcí uživatele v čistém Javascriptu by bylo neefektivní. Pro tento účel jsem vybral framework React.js. O jeho výběru více v kapitole 3.3.3. React.js.

11.2.4. Funkčnost

Funkčnost menu a tlačítek navigace bude zachována.

Tlačítko + nebude implementováno. Jeho funkčnost spadá do podobné kategorie jako uživatelská úprava menu (resp. editace položek menu). Při přidání jednorázové položky se totiž vytváří nová položka restaurace, jenom se nezobrazuje v menu. Funkcionalita úpravy menu v rámci této práce není implementovaná.

11.3. Implementace

Při zpracování requestu na stránku detailu objednávky, která obsahuje menu, se načtou všechny položky menu, které nemají žádného rodiče (tzn., nepatří do žádné skupiny položek). Těm se rekurzivně v databázi dohledají potomci a poskládá se strom menu. Ten se uloží do cache, aby se volání databáze a sestavování položek do stromu nemuselo volat znovu.

Menu se následně převede do JSON formátu, předá se scala template jako parametr a při skládání HTML stránky se uloží do Javascriptového objektu.

Menu se v Javascriptu skládá ze 4 React.js tříd. Třída Menu vykresluje tlačítko +, aktuální menu a třídy MenuNavigation a MenuTable. Z těchto tříd jsou do ní propagovány a zpracovávají se v ní reakce na události kliknutí na tlačítka menu a navigace. Zároveň se v ní ve funkci arrangeMenu (viz. kód 25) uspořádává menu tak, aby bylo připravené k vykreslení na pozice v souřadnicích X, Y.

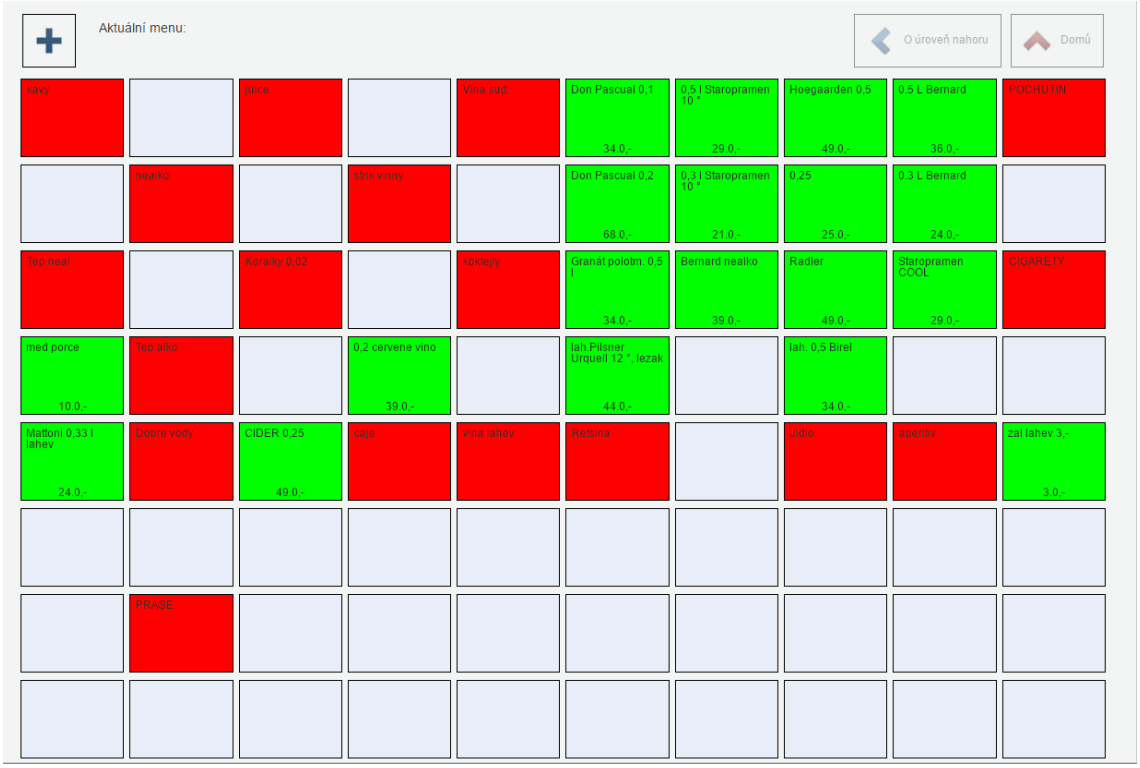
Kód 25: Funkce arrangeMenu, která uspořádává menu pro vykreslení – soubor accountDetail.js.

```
arrangeMenu: function(array) {
    // Menu dimensions are 10 * 8
    var sortedMenu = new Array(80).fill(null);

    // For every item of array (unsorted menu)...
    for(var x = 0; x < array.length; x++){
        if(array[x] != null) {
            //...put item in sortedMenu to correct position.
            sortedMenu[(parseInt(array[x].x) - 1) * 10
                + parseInt(array[x].y) - 1] = array[x];
        }
    }

    return sortedMenu;
},
```

Připravené pole reprezentující menu se předává třídě MenuTable. Ta celé pole projde a pro každou položku ověří, jestli se jedná o prázdné tlačítko, skupinu položek nebo list. Podle toho tlačítko vykreslí různě a přidá mu různé onClick funkce. Samostatná tlačítka jsou vykreslena pomocí třídy MenuButton, která je do MenuTable vnořena.

Třída MenuNavigation obsahuje tlačítka O úroveň nahoru a Domů a zaručuje, že tlačítka jsou funkční v moment, kdy nejsme v kořenovém menu.

Všechny akce se propagují do třídy Menu, která při průchodu menu menu překreslí (viz obr. 16), nastaví funkčnost tlačítek navigace a přepíše cestu Aktuální menu. Z třídy Menu se ven propagují kliknutí na tlačítko položky, kterou chceme objednat, a počítá se s tím, že bude zpracováno jinou React.js komponentou.



Obr. 16: Obrazovka detailu účtu, výřez menu (webová aplikace).

11.4. Testování

V rámci testování se prochází aplikace pomocí selenium scriptu (soubor menu.html) způsobem popsaným níže. Testování probíhá za použití předem vytvořeného uživatele user1, heslo 12345, role 1. Test počítá s tím, že v aplikaci je již vytvořen alespoň jeden otevřený účet.

Test počítá s tím, že je v aplikaci vytvořeno menu pomocí testovacího databázového scriptu. Při testování v instalaci, kde menu není vytvořeno tímto způsobem je potřeba test provést ručně.

1. Přechod na Detail účtu.
   1.1. Přihlášení
   1.2. Přechod na stránku Pokladny.
   1.3. Výběr prvního otevřeného účtu (přechod na jeho detail).
2. Test pozice.
   2.1. Test, že položka PRASE je na pozici X = 2 a Y = 7.
3. Test pohybu v menu.
   3.1. Přechod do skupiny položek „Koralky 0,02“.
   3.2. Kliknutí na tlačítko „O úroveň nahoru“.
   3.3. Přechod do skupiny položek „Koralky 0,02“.
   3.4. Přechod do skupiny RUMY.
   3.5. Kontrola pozice položky „Malteco 15yr 0.05“ na pozici X = 8 a Y = 5.
   3.6. Kontrola správného vypsání aktuálního menu (obsahuje „Koralky 0,02 > RUMY“).
   3.7. Kliknutí na tlačítko Domů.
   3.8. Kontrola správného vypsání aktuálního menu (je prázdné).

11.5. Shrnutí

V této iteraci se mi úspěšně podařilo implementovat menu do detailu objednávky. Menu se přizpůsobuje velikosti stránky a vypadá a funguje stejně, jako ve stávající aplikaci. Při tvorbě menu jsem se naučil nový Javascript framework React.js, který umožňuje dynamicky překreslovat webové stránky.

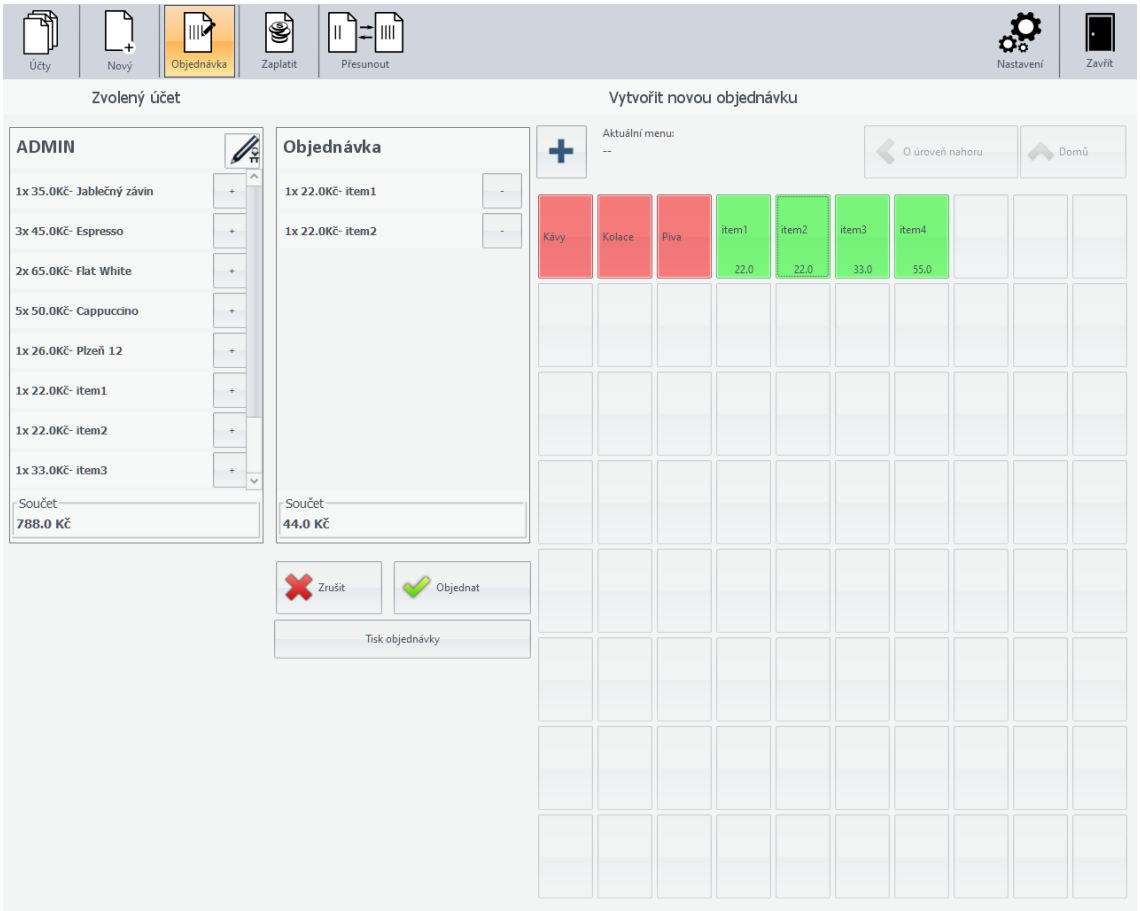
Implementace menu se prolínala s implementací obrazovky práce s účtem (konkrétně s implementací objednávky a rámečku účtu), popsané v následující kapitole. Společně s ostatními aktivitami popsanými v této kapitole zabrala přibližně 25 hodin. V plánu bylo implementaci práce s menu přiřazeno 8 hodin. Nejvíc času navíc zabralo učení se pro mě nového frameworku React.js.

12. Práce s účtem

Tato kapitola se věnuje implementaci obrazovky, která umožňuje přenést položky z menu podniku do objednávky a z ní na zvolený účet. Odtud je zároveň možné upravit detaily objednávky a objednávku vytisknout. Na obrazovce se zobrazuje menu, jehož implementace je popsaná v minulé kapitole.

12.1. Analýza a návrh

Na stránce se mimo menu zobrazují rámečky zvoleného účtu, který obsahuje již objednané položky a rámeček Objednávka (viz obr. 17). Rámečky zobrazují aktuální sumu ceny položek v sobě zobrazených. Pod rámečkem Objednávka je skupina tlačítek, které umožňují práci s položkami v tomto rámečku.

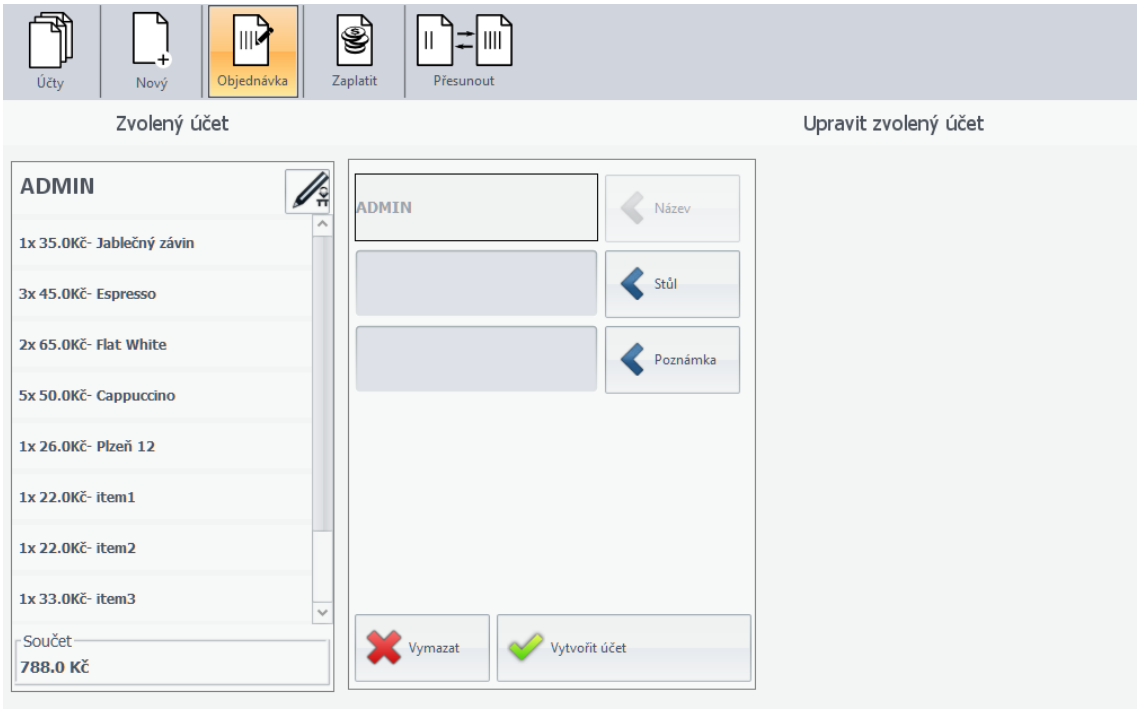


Obr. 17: Obrazovka detailu účtu (stávající aplikace).

Po kliku na položku menu se položka přidá do rámečku Objednávka. Z ní je možné položky po jedné odebírat nebo je odebrat všechny najednou tlačítkem Zrušit. Do rámečku Objednávka je zároveň možné přidávat položky na základě těch, co jsou pro účet již objednané (v rámečku zvoleného účtu).

Při kliknutí na tlačítko Objednat jsou položky z rámečku Objednávka přesunuty do rámečku zvoleného účtu, což znamená, že jsou reálně objednané. Tlačítkem Tisk objednávky je objednávka vytištěna podobně, jako když se tiskne účet.

V rámečku zvoleného účtu je tlačítko, které místo rámečku Objednávka a menu zobrazí rámeček úpravy detailů účtu (viz obr. 18). Zde je možné upravit stůl, ke kterému je účet přiřazen a poznámku k účtu.



Obr. 18: Obrazovka úpravy detailů účtu (stávající aplikace).

Funkcionalita výše popsaného bude do webové aplikaci přenesena v plném rozsahu. Na obrazovce úpravy detailů účtu nebude zobrazen rámeček detailu účtu, protože zde nemá žádný význam.

12.2. Implementace

Většina stránky práce s účtem je implementovaná pomocí React.js. Obsah stránky vykresluje React.js třída AccountDetail. Vykresluje rámeček zvoleného účtu, rámeček objednávky a React.js třídu Menu popsanou v minulé kapitole. Obsahuje funkce zpracovávající následující akce:

- Klik na tlačítko menu – funkce je předaná jako parametr třídě Menu. Při spuštění přidá položku, na kterou bylo kliknuto v menu do pole obsahující položky objednávky a abecedně ho seřadí.
  Při přesunutí položky do jakého pole položek používám funkci addItemToList (viz kód 26). Ta zkontroluje, že položka není zrušená, zaplacená nebo přijatá. Následně projde celé pole

a zkontroluje, jestli se v něm již tato položka nevyskytuje. Pokud ne, přidá ji do něj. Pokud ano, jenom zvýší počet těchto položek. Díky tomu se například při přidání jedné kávy do účtu obsahujícího dvě kávy vypíše jednou položka „3x káva“, ne 3x položka „1x káva“.
Tato funkce je dále použita na všech místech, kde se v Javascriptu přesouvá položka z jednoho pole položek do druhého.

Kód 26: Funkce přidávající položku do pole položek – soubor cashdesk.js.

```
/**
 * Adds one item to list (OrderDTO list).
 * @param list Array, where item is added (format array[ITEM][count]).
 * @param item Item to add to array.
 * @returns {*} List with added item.
 */
function addItemToList(list, item){
    if(item.isaccepted || item.iscanceled || item.ispaid){
        return list;
    }

    for(var i = 0; i < list.length; i++){

        if(list[i][0].menuItem.menuitemid == item.menuItem.menuitemid){
            list[i][1]++;
            return list;
        }
    }

    list.push([item, 1]);
    return list;
};
```

- Klik na tlačítko + u položky zvoleného účtu - při spuštění přidá položku z účtu do pole obsahující položky objednávky a abecedně ho seřadí.
- Klik na tlačítko – u položky objednávky – při spuštění odebere položku z pole objednávky a abecedně ho seřadí.
- Klik na tlačítko Vymazat – přepíše pole objednávky prázdným polem.

- Klik na tlačítko Objednat
  - Zkontroluje, že pole objednávky obsahuje nějaké položky.
  - Pokud ano, přetransformuje je na takové, které je možné poslat v requestu na serverové REST rozhraní zpracovávající vytvoření nové objednávky.
  - Vytvoří AJAX request (viz kód 27) a pošle ho na url '/rest/account/'+ account.id +'/order', kde account.id je číslo zvoleného účtu.
  - V případě, že byl request úspěšně odeslán, vypíše se do Javascript konzole prohlížeče odpověď serveru, do pole zvoleného účtu je připojeno pole objednávky a pole objednávky je přepsáno prázdným polem.

Kód 27: AJAX request volaný při objednání položek – soubor accountDetail.js.

```
$.ajax({
    context: this,
    type: 'POST',
    url: '/rest/account/'+ account.id +'/order',
    data: JSON.stringify({
        isUnaccepted: true,
        items: items,
        targetAccId: account.id
    }),
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    success: function (data) {
        console.log(data);

        this.setState({
            orderItems: [],
            accountItems: addTwoLists(
                this.state.accountItems,  this.state.orderItems)
        });
    }
})
```

12.3. Testování

V rámci testování se prochází aplikace pomocí selenium scriptu (soubor cashbobDetail.html) způsobem popsaným níže. Testování probíhá za použití předem vytvořeného uživatele user1, heslo 12345, role 1.

Test počítá s tím, že je v aplikaci vytvořeno menu pomocí testovacího databázového scriptu. Při testování v instalaci, kde menu není vytvořeno tímto způsobem je potřeba test provést ručně.

1. Vytvoření testovacího účtu.
   1.1. Přihlášení
   1.2. Přechod na stránku Pokladny.
   1.3. Klinutí na tvorbu nového účtu-
   1.4. Vyplnění jména účtu „test account“.
2. Objednání z menu a práce s rámečkem Objednávka.
   2.1. 4x kliknutí na položku „Don Pascual 0,1“ v menu.
   2.2. 1x kliknutí na položku „med porce“ v menu.
   2.3. 4x kliknutí na odebrání položky „Don Pascual 0,1“ v rámečku objednávky.
   2.4. Kontrola, že se položky zobrazily v rámečku objednávky.
   2.5. Kontrola součtu ceny v rámečku objednávky.
   2.6. Objednání položek.
   2.7. Kontrola, že se položky zobrazily v rámečku zvoleného účtu.
3. Objednání z menu z účtu.
   3.1. 2x kliknutí na položku „Don Pascual 0,1“ v rámečku zvoleného účtu.
   3.2. Objednání položek.
   3.3. Kontrola, že se položky zobrazily v rámečku zvoleného účtu.
4. Úprava detailů účtu.
   4.1. Přechod na obrazovku úpravy detailů účtu.
   4.2. Výběr stolu, ke kterému bude účet přiřazen.
   4.3. Potvrzení úpravy účtu.
   4.4. Přechod na obrazovku úpravy detailů účtu.
   4.5. Kontrola, že stůl byl přiřazen.

12.4. Shrnutí

Tato kapitola, práce s účtem, navazovala na tvorbu menu. Podařilo se ho spojit s rámečky zvoleného účtu zobrazující již objednané položky a rámečkem objednávky. Implementace úpravy detailů zvoleného účtu proběhla rovněž v pořádku.

Na aktivity popsané v této kapitole bylo v plánu vyhrazeno 16 hodin. Do plánu nebylo zahrnuto, že strávím čas na úpravě detailů účtu. Reálně jsem na něm strávil přibližně 14 hodin, což aspoň částečně ubírá ze zpoždění nabraného při implementaci menu.

13. Přesunutí položek mezi účty

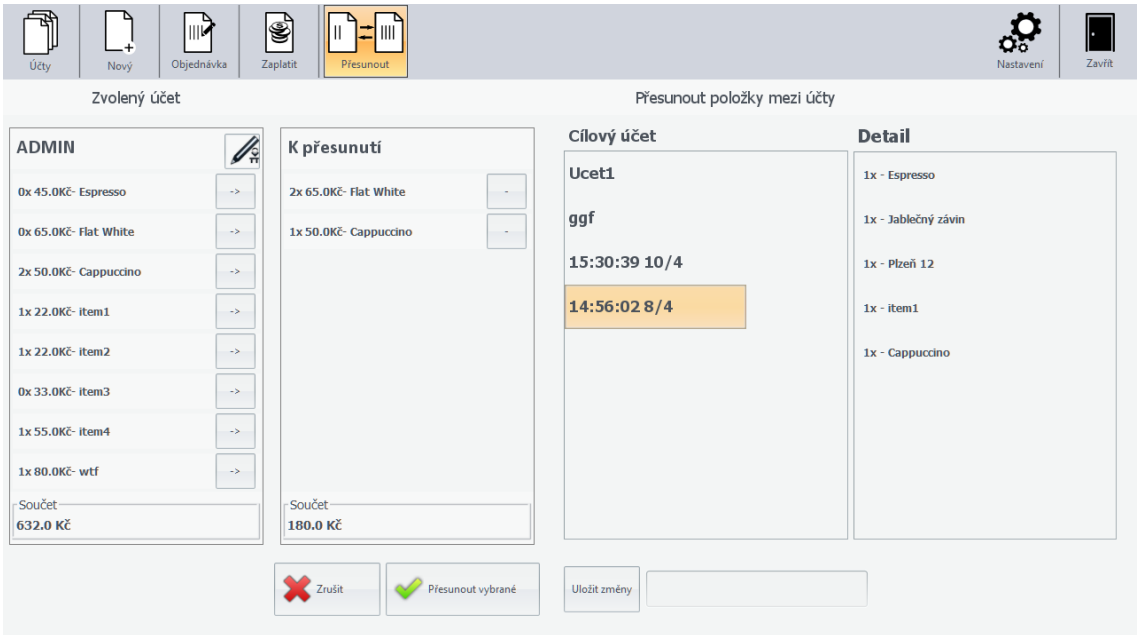
Tato kapitola popisuje práci na obrazovce, která umožňuje přesunout objednané položky ze zvoleného účtu na jiný účet – již vytvořený nebo na nový.

13.1. Analýza a návrh

Stávající aplikace obsahuje následující rámečky (viz obr. 19):

- rámeček zvoleného účtu – zobrazuje položky zvoleného účtu,
- rámeček K přesunutí – zobrazuje položky, které uživatel chce přesunout,
- rámeček Cílový účet – nabízí všechny otevřené účty, kam je možné položky z rámečku K přesunutí přesunout,
- rámeček Detail – zobrazuje položky v účtu, který je vybrán v rámečku Cílový účet.

Pod rámečkem K přesunutí jsou tlačítka pro odstranění všech položek z rámečku K přesunutí a tlačítko Přesunout vybrané, které přesun provede. Pod rámečkem Cílový účet se nachází tlačítko Nový účet. Po kliknutí na něj se zobrazí textbox, do něhož je možné zadat jméno účtu a pak ho vytvořit kliknutím na tlačítko Uložit změny. To slouží k rychlému přesunutí položek na nový účet.



Obr. 19: Obrazovka Editace stolů (stávající aplikace).

Rozhodl jsem se změnit tlačítko Nový účet. Pro uživatele může být matoucí, že je možné vytvářet účty dvěma způsoby – jednak přes tlačítko Nový v menu a jednak v na obrazovce přesunutí mezi účty pomocí tlačítka Nový účet. Formulář pro tvorbu nového účtu jsem navíc

měl připravený z implementace tvorby nového účtu. Proto jsem ho předělal tak, aby se zobrazoval v modálním okně. Na obrazovce Přesunutí mezi účty se v modálním okně navíc zobrazí tlačítko pro vytvoření účtu bez přechodu na detail. Díky tomu se při vytvoření nového účtu nepřejde do detailu objednávky, nový účet se pouze zobrazí v rámečku Cílový účet. Tak nedojde ke ztrátě žádné funkcionality.

13.2. Implementace

Většina stránky práce s účtem je implementovaná pomocí React.js. Rámečky a jejich obsah vykresluje React.js třída Borders. Obsahuje funkce zpracovávající následující akce:

- Klik na tlačítko + u položky zvoleného účtu – přesune položku z rámečku zvoleného účtu do rámečku K přesunutí. Rámeček K přesunutí abecedně seřadí.
- Klik na tlačítko Vybrat všechny objednávky – sloučí pole položek z rámečku zvoleného účtu rámečku K přesunutí. Rámeček zvoleného účtu přepíše prázdným polem. Rámeček k přesunutí abecedně seřadí.
- Klik na tlačítko – u položky k přesunutí – při spuštění odebere položku z pole k přesunutí a přidá ji do pole zvoleného účtu. Rámeček zvoleného účtu abecedně seřadí.
- Klik na tlačítko Zrušit – sloučí pole položek z rámečku K přesunutí k zvoleného účtu. Pole položek z rámečku K přesunutí přepíše prázdným polem. Rámeček zvoleného účtu abecedně seřadí.
- Klik na položku v rámečku Cílový účet – zjistí se ID účtu, na který se kliklo. V poli otevřených účtů podle toho účet najde, jeho položky vypíše do rámečku Detail a abecedně seřadí.
- Klik na tlačítko Přesunout:
  - Zkontroluje, že je vybrán cílový účet.
  - Zkontroluje, že pole K přesunutí obsahuje nějaké položky.
  - Přesune položky K přesunutí do pole cílového účtu (pouze mezi Javascriptovými poli).
  - Přetransformuje položky K přesunutí na takové, které je možné poslat v requestu na serverové REST rozhraní zpracovávající přesun položek mezi účty.
  - Vytvoří AJAX request a pošle ho na url '/rest/account/'+ currentAccount.id +'/moveItems', kde account.id je číslo cílového účtu.
  - V případě, že byl request úspěšně odeslán, vypíše se do Javascript konzole prohlížeče odpověď serveru.

13.3. Testování

V rámci testování se prochází aplikace pomocí selenium scriptu (soubor cashdeskMove.html) způsobem popsaným níže. Testování probíhá za použití předem vytvořeného uživatele user1, heslo 12345, role 1. Test počítá s tím, že v aplikaci je již vytvořen alespoň jeden otevřený účet.

Test počítá s tím, že je v aplikaci vytvořeno menu pomocí testovacího databázového scriptu. Při testování v instalaci, kde menu není vytvořeno tímto způsobem je potřeba test provést ručně.

1. Přechod na Detail účtu.
   1.1. Přihlášení

1.2. Přechod na stránku Pokladny.
1.3. Výběr prvního otevřeného účtu (přechod na jeho detail).
2. Objednání testovacích položek.
2.1. 4x kliknutí na položku „Don Pascual 0,1“ v menu.
2.2. 1x kliknutí na položku „med porce“ v menu.
3. Tvorba 2. účtu ze stránky přesunutí.
3.1. Přechod na stránku přesunutí.
3.2. Kliknutí na tvorbu nového účtu.
3.3. V modálním okně kliknutí na tlačítko Vytvořit bez přechodu na detail.
4. Test přesouvání.
4.1. Kliknutí na tlačítko Vybrat všechny objednávky.
4.2. Kontrola, že se všechny položky přesunuly do rámečku k přesunutí.
4.3. Kliknutí na tlačítko Zrušit.
4.4. Kliknutí na tlačítko + u položky „Don Pascual 0,1“ v rámečku otevřeného účtu.
4.5. Kliknutí na tlačítko + u položky „med porce“ v rámečku otevřeného účtu.
4.6. Kliknutí na tlačítko Přesunout.
4.7. Kliknutí na tlačítko + u položky „Don Pascual 0,1“ v rámečku otevřeného účtu.
4.8. Kliknutí na tlačítko Přesunout.
4.9. Kontrola, že rámeček detail obsahuje 2x položku „Don Pascual 0,1“ a jednou položku „med porce“.

13.4. Shrnutí

V rámci práce na této kapitole jsem implementoval stránku pro přesunutí položek mezi účty. Zjednodušil jsem tvorbu nového účtu, která na této stránce neprobíhá pomocí samostatného tlačítka, jak to bylo ve stávající aplikaci, ale ve stejném formuláři v modálním okně jako při běžné tvorbě nového účtu. Navíc ve formuláři přibylo tlačítko pro tvorbu účtu bez přechodu na jeho detail, takže uživatel zůstane na obrazovce přesunutí položek mezi účty.

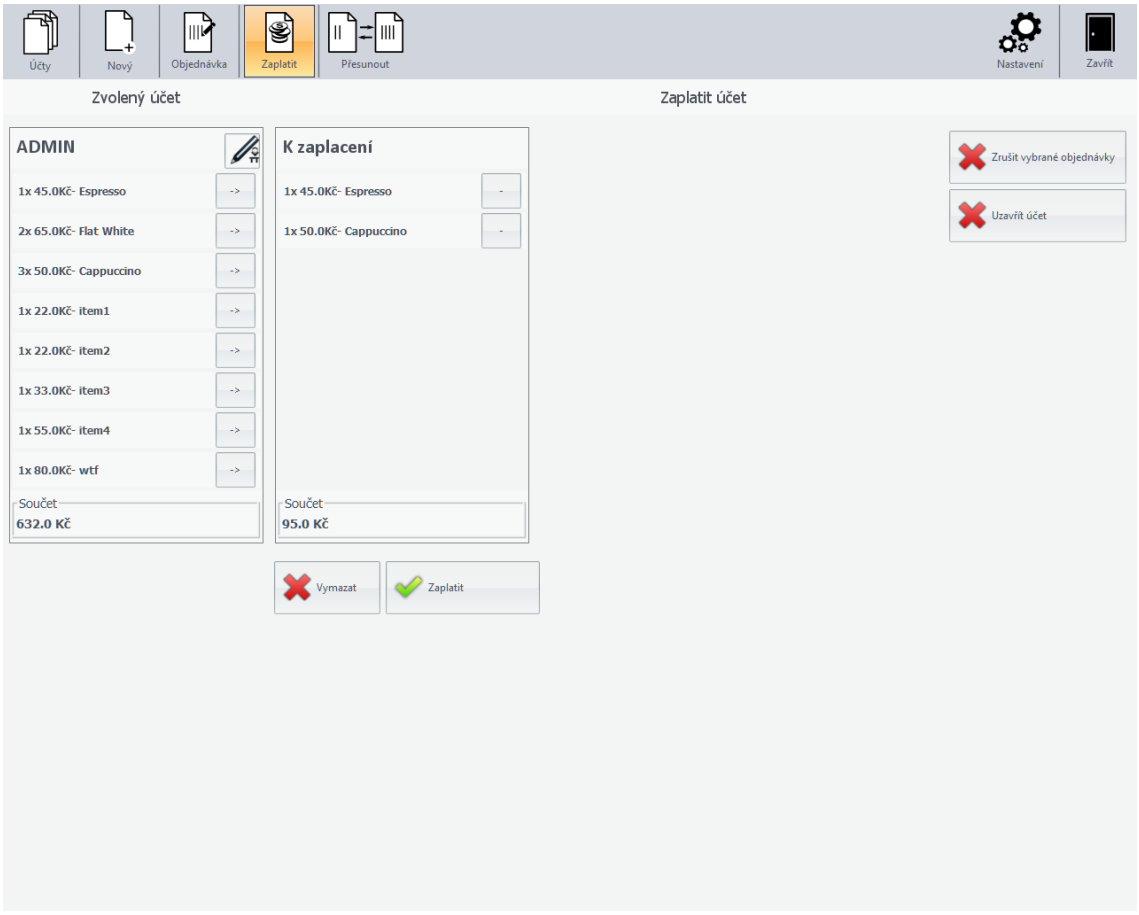
Na aktivitách spojených s prací na této kapitole jsem strávil přibližně 16 hodin. V plánu bylo 20 hodin. K úspoře došlo díky tomu, že část funkcionality byla implementovaná nebo převzatá z práce na předchozích kapitolách.

14. Zaplacení účtu

Tato kapitola popisuje placení položek účtu a uzavření účtu.

14.1. Analýza

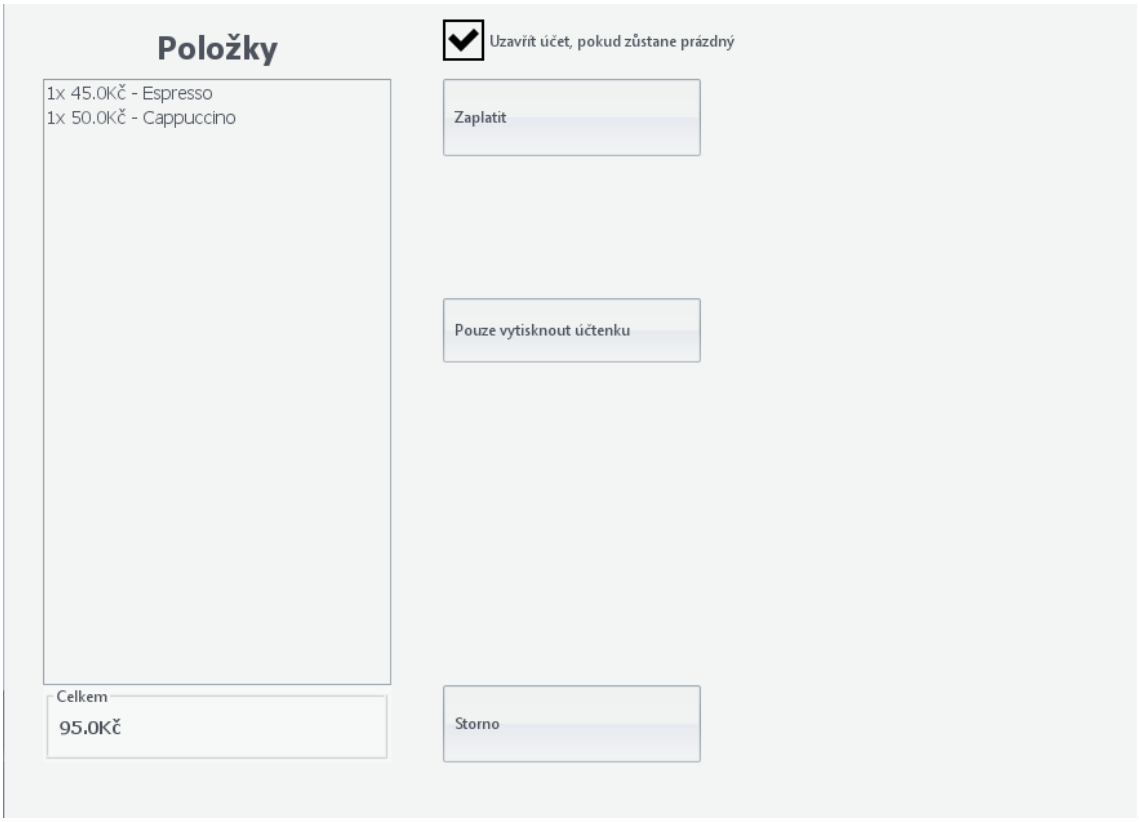
Obrazovka uzavření účtu umožňuje vybrání položek k zaplacení ze zvoleného účtu. Ty je následně možné zaplatit nebo je zrušit (viz obr. 20).



Obr. 20: Obrazovka zaplacení účtu (stávající aplikace).

Při platbě vybraných položek se zobrazí modální okno obsahující souhrn všech položek vybraných k zaplacení (viz obr. 21). Umožňuje vybrat, jestli se při zaplacení všech položek má účet uzavřít a nebo zůstat otevřený k dalšímu použití. Akce je potvrzena tlačítkem Zaplatit. Okno zároveň umožňuje vytisknout účtenku zobrazující položky vybrané k zaplacení.

Kliknutí na tlačítko Zrušit vybrané objednávky označí položky vybrané v rámečku K zaplacení jako zrušené a už se v systému dále nezobrazují. Tlačítko Uzavřít účet celý účet uzavře a ten se již dále nezobrazuje. Po kliknutí na obě tlačítka vyskočí okno požadující potvrzení akce.



Obr. 21: Modální okno zobrazené po kliknutí na tlačítko zaplatit (stávající aplikace).

14.2. Návrh

Pod rámeček zvoleného účtu umístím tlačítko pro přesun všech položek do rámečku k zaplacení. Díky tomu bude možné rychleji manipulovat s položkami.

Tlačítka Zrušit vybrané objednávky a Uzavřít účet posunu přímo vlevo vedle rámečku K zaplacení, protože nevidím důvod, proč by měla být až na kraji obrazovky. K nim přesunu tlačítko Zaplatit. Díky tomu pod rámečky budou pouze tlačítka sloužící k manipulaci s položkami mezi rámečky a vpravo budou kontrolní tlačítka.

Nevidím důvod pro to zobrazovat modální okno obsahující souhrn informací o zaplacených položkách, protože nepřináší navíc žádnou funkčnost ani dodatečné informace. Checkbox Uzavřít účet, pokud zůstane prázdný pak přesunu mezi kontrolní tlačítka vpravo od rámečku k zaplacení.

Tlačítko Zaplatit přejmenuji na tlačítko Vytisknout účet a zaplatit. Před zaplacením není důvod účet tisknout, proto není třeba zvláštního tlačítka. Naopak při placení je tisk třeba vždy, proto je vhodné tyto akce provést zároveň.

14.3. Implementace

Většina stránky práce s účtem je implementovaná pomocí React.js. Obsah stránky vykresluje React.js třída AccountPay. Vykresluje rámeček zvoleného účtu, rámeček K zaplacení a tlačítka. Obsahuje funkce zpracovávající následující akce:

- Klik na tlačítko + u položky zvoleného účtu – přesune položku z rámečku zvoleného účtu do rámečku K zaplacení. Rámeček K zaplacení abecedně seřadí.
- Klik na tlačítko Vybrat všechny objednávky – sloučí pole položek z rámečku zvoleného účtu rámečku K zaplacení. Rámeček zvoleného účtu přepíše prázdným polem. Rámeček K zaplacení abecedně seřadí.
- Klik na tlačítko – u položky k zaplacení – při spuštění odebere položku z pole k zaplacení a přidá ji do pole zvoleného účtu. Rámeček zvoleného účtu abecedně seřadí.
- Klik na tlačítko Vymazat – sloučí pole položek z rámečku K zaplacení k zvoleného účtu. Pole položek z rámečku K zaplacení přepíše prázdným polem. Rámeček zvoleného účtu abecedně seřadí.
- Klik na tlačítko Vytisknout účet a zaplatit:
  - Ověří, že rámeček K zaplacení obsahuje nějaké položky.
  - Přetransformuje položky K zaplacení na takové, které je možné poslat v requestu na serverové REST rozhraní zpracovávající zaplacení položek účtu. .
  - Na URL '/rest/account/'+ account.id +'/payItems', kde account.id je číslo účtu, pošle AJAX request.
  - Pokud byli zaplaceny všechny položky a účet se podařilo uzavřít, dojde k přesměrování na seznam účtů.
- Klik na tlačítko Zrušit vybrané objednávky:
  - Ověří, že rámeček K zaplacení obsahuje nějaké položky.
  - Zobrazí upozornění s žádostí o potvrzení akce.
  - Přetransformuje položky K zaplacení na takové, které je možné poslat v requestu na serverové REST rozhraní zpracovávající zrušení položek účtu.
  - Na URL '/rest/account/'+ account.id + '/cancelItems', kde account.id je číslo účtu, pošle AJAX request.
  - V případě úspěšného odeslání účtu je vypsána odpověď serveru do Javascriptové konzole.
- Klik na tlačítko Uzavřít účet:
  - Ověří, že rámeček K zaplacení obsahuje nějaké položky. Pokud ano, zobrazí upozornění, že účet obsahuje nezaplacené položky.
  - Pokud účet neobsahuje nezaplacené položky, zobrazí upozornění s žádostí o potvrzení akce.
  - Na URL '/rest/account/close/'+ account.id, kde account.id je číslo účtu, pošle AJAX request.
  - Pokud se účet podařilo uzavřít, dojde k přesměrování na seznam účtů.

14.4. Testování

1. Přechod na Detail účtu.
    1.1. Přihlášení
    1.2. Přechod na stránku Pokladny.
    1.3. Výběr prvního otevřeného účtu (přechod na jeho detail).
2. Objednání testovacích položek.
    2.1. 4x kliknutí na položku „Don Pascual 0,1“ v menu.
    2.2. 1x kliknutí na položku „med porce“ v menu.
3. Test zaplacení.
    3.1. Přechod na stránku Zaplacení
    3.2. 1x kliknutí na položku „Don Pascual 0,1“ v rámečku zvoleného účtu.
    3.3. Kliknutí na tlačítko Vytisknout účet a zaplatit.
    3.4. Potvrzení akce.
4. Test zrušení položky
    4.1. 1x kliknutí na položku „med porce“ v rámečku zvoleného účtu.
    4.2. Kliknutí na tlačítko Zrušit vybrané objednávky.
    4.3. Potvrzení akce.
5. Test Uzavření účtu.
    5.1. Kliknutí na tlačítko Uzavřít účet.
    5.2. Potvrzení akce.
    5.3. Ověření, že došlo k přesměrování na seznam účtů.

14.5. Shrnutí

V rámci této iterace jsem zpracoval placení a uzavírání účtu. V obrazovce jsem změnil rozložení tlačítek, takže je obrazovka přehlednější, odstranil modální okno a přesunul tlačítka, které obsahovalo, přímo na stránku.

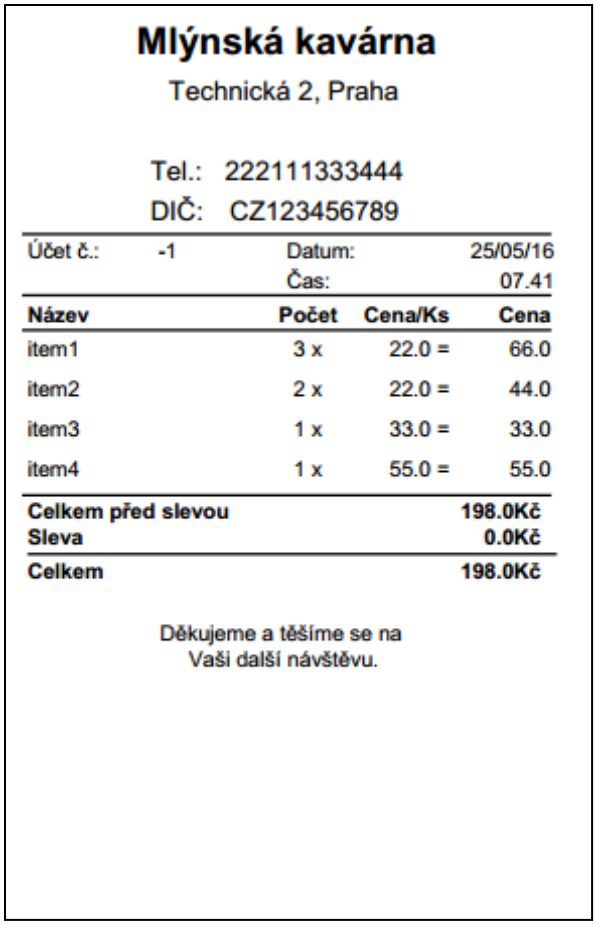
Na tuto iteraci jsem měl vyhrazeno 20 hodin. Reálně jsem na ní strávil přibližně 8 hodin, díky tomu, že část funkcionality byla implementovaná nebo převzatá z práce na předchozích kapitolách.

15. Tisk

Tato kapitola popisuje tisk účtenek do souborů PDF a PNG v restauračním systému.

15.1. Analýza a návrh

Stávající aplikace umožňuje na obrazovce detailu účtu tisk objednávek a na obrazovce zaplacení účtu tisk účtenky (viz. obr. 22). Na účtence je jméno restaurace, její adresa, DIČ a telefonní číslo. Následně obsahuje číslo účtu (v případě účtenky). V seznamu objednávek je u každé položky její jméno, počet položek, cena za kus a celková cena položek (cena za kus krát počet položek). Na konci seznamu je suma ceny, sleva a cena po slevě.



Mlýnská kavárna

Technická 2, Praha

Tel.: 222111333444
DIČ: CZ123456789

Účet č.:	-1	Datum:	25/05/16
		Čas:	07.41

Název	Počet	Cena/Ks	Cena
item1	3 x	22.0 =	66.0
item2	2 x	22.0 =	44.0
item3	1 x	33.0 =	33.0
item4	1 x	55.0 =	55.0

Celkem před slevou	**198.0Kč**
Sleva	**0.0Kč**
Celkem	**198.0Kč**

Děkujeme a těšíme se na
Vaši další návštěvu.

Obr. 22: Účtenka (stávající aplikace).

Rozložení účtenky a informace, které obsahuje, nechám zachované. Vypustím údaj o slevě a ceně po slevě, protože momentálně není v systému možné nastavit účtu slevu, takže by byla informace zbytečná. Při budoucí implementaci by bylo vhodné tento údaj zobrazovat pouze v případě, kdy je nějaká sleva nastavena a v ostatních případech ho skrýt.

K tisku používá stávající aplikace knihovnu Jasper [19] a tiskne účtenky na defaultní tiskárně systému. K tisku Jasper využívá knihovnu AWT. Ve webové aplikaci jsem se rozhodl účtenku nejdříve převést do formátu, který bude možné ze serveru odeslat na klienta a tam ho zpracovat (například právě vytisknout) libovolným způsobem. Pro webovou aplikaci jsem se rozhodl použít knihovnu Apache FOP [20], protože jsem s ní pracoval již v minulosti. Ta umožňuje na základě vytvořené XSL-FO šablony zformátovat data do PDF dokumentu. Pro tisk na menších přenosných tiskárnách spojených s mobilní verzí aplikace restauračního systému je vhodné mít podklad pro tisk ve formátu PNG. Proto implementuji převod z PDF do PNG pomocí knihovny Apache PDFBox [21] a výsledný soubor vystavím na webovém rozhraní.

15.2. Implementace

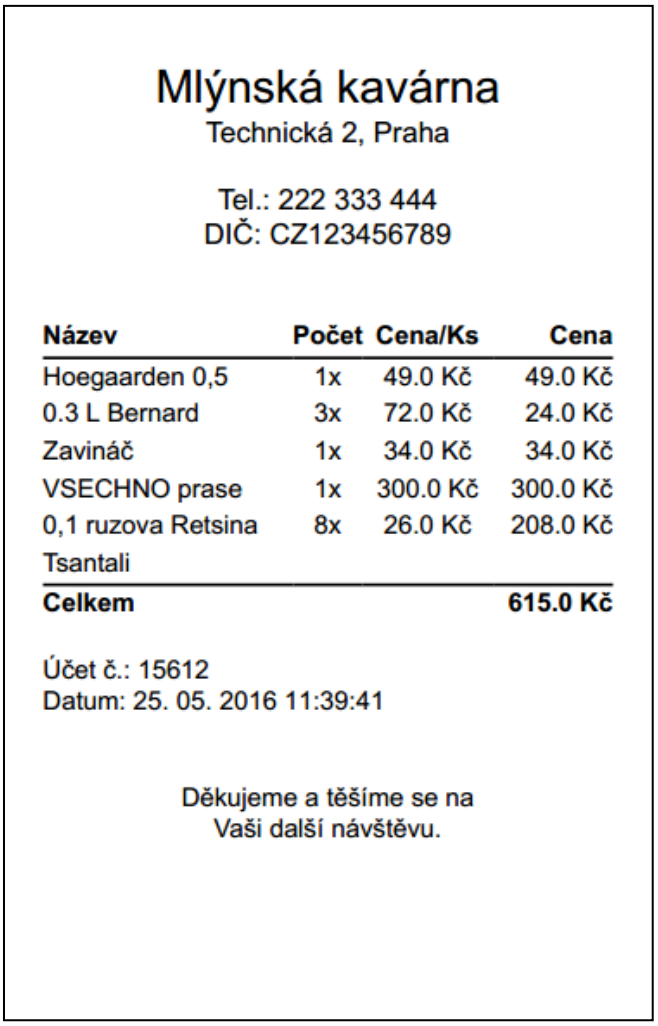
Vrácení souboru účtenky ve formátu vystavuji přes REST rozhraní na URL '/rest/fop/pdf' a ve formátu PNG na '/rest/fop/png'. Rozhraní přijímá POST request, ve kterém je třeba specifikovat číslo účtu (nebo ho nechat nevyplněné) a objednávky, které se mají na účtu zobrazit. Tyto údaje jsou předány k tisku třídě FopService.

Apache FOP je potřeba předložit na vstup data ve formátu XML a šablonu, která určuje vzhled a rozložení dokumentu ve formátu XLS-FO. XML data si připravím do dočasného souboru pomocí JAXB. To je standardní Java framework umožňující na základě anotacích obsažených v Java entitě vygenerovat XML soubor obsahující informace v jejích určených parametrech a metodách. Tento soubor je dočasný a po použití je smazán.

Šablonu pro účtenky mám uloženou jako receiptTemplate.xsl. Zde je v XSL formátu nadefinovaný vzhled dokumentu. Obsahuje informace o vzhledu stránky, jako je její velikost a okraje (viz kód 28) a následně umístění textu. Ten je při generování načten z připraveného XML souboru a cílí se pomocí XPath dotazů. Text je možný pozicovat a formátovat pro dosažení chtěného výsledku (viz obr. 23).

Kód 28: Ukázka kódu XLS-FO – vzhled stránky s dynamickou délkou – soubor receiptTemplate.xsl.

```
<xsl:variable name="page-height" select="//height"/>

<fo:layout-master-set>
    <fo:simple-page-master master-name="receipt"
        page-width="80mm" page-height="{$page-height}">
        <fo:region-body margin-bottom="1cm" margin-top="0.7cm"
            margin-left="0.5cm" margin-right="0.5cm" border="0"/>
        <fo:region-before extent="0cm" border="0"/>
        <fo:region-after extent="0cm" border="0"/>
        <fo:region-start extent="1cm" border="0"/>
        <fo:region-end extent="1cm" border="0"/>
    </fo:simple-page-master>
</fo:layout-master-set>
```



Mlýnská kavárna
Technická 2, Praha

Tel.: 222 333 444
DIČ: CZ123456789

Název	Počet	Cena/Ks	Cena
Hoegaarden 0,5	1x	49.0 Kč	49.0 Kč
0.3 L Bernard	3x	72.0 Kč	24.0 Kč
Zavináč	1x	34.0 Kč	34.0 Kč
VSECHNO prase	1x	300.0 Kč	300.0 Kč
0,1 ruzova Retsina	8x	26.0 Kč	208.0 Kč
Tsantali			
Celkem			**615.0 Kč**

Účet č.: 15612
Datum: 25. 05. 2016 11:39:41

Děkujeme a těšíme se na
Vaši další návštěvu.

Obr. 23: Účtenka (WA).

Při dotaz na účtenku ve formátu PNG je účtenka nejdříve vygenerovaná jako PDF a následně pomocí knihovny Apache PDFBox převedena do PNG a odeslána.

15.3. Shrnutí

V rámci této kapitoly se mi podařilo implementovat převod dat účtenky do souboru formátu PDF, které je následně možné vytisknout nebo převést na PNG a poslat přes REST rozhraní na mobilního klienta.

Na implementaci tisku jsem plánoval 20 hodin. Protože jsem s knihovnou Apache FOP měl zkušenosti z dřívějška, podařilo se mi převod do PDF stihnout za 7 hodin.

16. Další možnosti rozvoje aplikace

Tato kapitola rozebírá, které témata by bylo možné dále zpracovat a jak smysluplným způsobem rozšířit pokladnu webového restauračního systému.

16.1. Uchování dat na klientovi v případě odpojení

Momentálně dochází na klientovy pouze ke cachování souborů, které se v průběhu běhu aplikace nijak nemění – obrázků, CSS a Javascriptových souborů. Toto cachování by se dalo rozšířit o uchovávání dat, jako je například seznam otevřených účtů nebo podrobnosti o právě otevřeném účtu. Uložení dat na úrovni klienta by se dalo řešit pomocí HTML5 paměti local storage. Ta umožňuje například pomocí Javascriptu ukládat data přímo do prohlížeče klienta. Takový postup ušetří datovou komunikaci klienta se serverem v případě, kdy se pracuje s již dříve volanými daty. Zároveň by to umožnilo klientovi v omezené míře pracovat i v případě krátkodobého výpadku připojení k serveru.

16.2. Zamykání dat na serveru z důvodu zachování integrity

Ve webové aplikaci není nejsou nijak ošetřeny stavy, kdy s jedním účtem pracuje více uživatelů naráz. Byla by potřeba udělat analýza na to, jakým způsobem bude aplikace používaná, jestli bude ke konfliktům při práci na stejných datech docházet často, je potřeba je nějakým způsobem ošetřit tak, aby se těmto konfliktům předcházelo ještě před tím, než nastanou a jak tyto situace řešit.

Nemůže dojít k porušení integrity dat v databázi, protože při jejím volání se kontroluje, v jakém jsou zasažená data stavu. Může však dojít například k tomu, že když dva uživatelé budou mít otevřený stejný účet a jeden z nich ho zavře. Druhý uživatel účet uvidí jako otevřený a veškeré akce na něm mu budou zamítnuty. Účet uvidí jako zavřený až při následující akci, kterou v systému provede.

Jednou z možností, jak těmto situacím předcházet by bylo uzamčení účtu (nebo jiné entity) pro otevření v průběhu doby, co s ní bude pracovat jiný uživatel. Druhým řešením by bylo dynamické obnovování stránek všech aktuálně připojených uživatelů o změny, které nastaly na účtu, se kterým momentálně pracují. Taková funkcionalita by se dala zajistit například použitím websocketů.

16.3. Zobrazení stolů

Funkcionalita, kterou by řada podniků uvítala, je rozšíření všech obrazovek zobrazující stoly. Týkalo by se to především obrazovky správy stolů a modálních oken umožňujících přiřazení stolu k účtu. Stoly jsou momentálně zobrazené jako tlačítka v tabulce. Ideální by bylo, kdyby bylo možné zobrazit stoly tak, jak jsou zobrazeny fyzicky v restauračním zařízení. To by obsluze usnadnilo jejich výběr.

Tato funkcionalita by se dala promítnout i do zobrazení seznamu otevřených účtů, aby bylo vidět, že konkrétní účet náleží stolu s konkrétním fyzickým umístěním.

16.4. Design

Aplikace má momentálně poměrně strohý, funkcionalistický vzhled. Pokud by byla zaměřena na komerční sféru, bylo by třeba vzhled zmodernizovat tak, aby byl více přitažlivý. Nejlepší by bylo najmout grafika, který by vzhled navrhl a v ideálním případě i naprogramoval kostru front-endu bez funkcionality. Tu by později mohl dodat programátor.

Webová aplikace má zároveň možnost responzivního zobrazení pro různě veliké displeje. Vzhledem k tomu by stálo za zvážení jak moc design přizpůsobovat zařízením s méně velkým displejem.

17. Shrnutí

Na projektu Webové rozhraní restauračního systému jsem pracoval v rámci semestrálního projektu a předmětu Softwarové inženýrství, ve kterých jsem dokončil úkoly náležící 1. a 2. iteraci naplánované v harmonogramu projektu.

V práci jsem pokračoval v iteracích 3 až 10 v rámci bakalářské práce.

17.1. Iterace 1 a 2

Seznámil jsem se s stávajícím projektem a připravil si nový projekt v Play frameworku pro webovou aplikaci. V rámci přípravy jsem vytvořil návrh řešení zabezpečení aplikace, cachování, definoval jsem coding standards pro projekt a navrhl technologie, které budou při tvorbě aplikace použity.

Pro webovou aplikaci jsem použil model ze stávající aplikace a upravil jeho kód tak, aby v ní byl použitelný. Zároveň jsem upravil a importoval testovací data vytvořená pro stávající projekt.

Ze stávající aplikace jsem použil kód pro REST rozhraní a upravil ho tak, aby bylo vhodné pro projekt webové aplikace.

Implementoval jsem login stránku a zabezpečení aplikace, tzn. autorizaci a autentizaci. Aplikace umožňuje zabezpečit přístup k určené funkcionalitě pouze pro přihlášené uživatele a umožňuje ověřit práva uživatele. K otestování zabezpečení jsem vytvořil selenium skripty.

Všechny úkoly naplánované pro první dvě iterace se podařilo splnit.

17.2. Iterace 3 až 10

V iteracích 3. až 10. jsem se zaměřil na implementování domovské stránky a pokladny. Domovská stránka slouží jako rozcestník mezi jednotlivými částmi aplikace, z nichž nejdůležitější je pokladna.

Při vstupu do pokladny se zobrazí obrazovka seznamu účtů. Spolu s ní jsem implementoval kostru všech stránek pokladny, která se skládá z vrchního menu obsahující odkazy na seznam všech účtu, tvorbu nového účtu, detail účtu, zaplacení účtu, přesunutí položek mezi účty, tlačítka nastavení a odkazu na domovskou stránku. Seznam účtů zobrazí všechny otevřené účty s jejich názvem a případně s číslem stolu, ke kterému náleží.

Při kliknutí na tlačítko pro tvorbu nového účtu nedojde k přesměrování na stávající aplikaci, jako v případě stávající aplikace, ale zobrazí se modální okno s formulářem pro tvorbu účtu. Díky tomu je například z obrazovky přesunutí položek mezi účty možné vytvořit nový účet bez opuštění stránky.

V rámci obrazovky detailu účtu jsem implementoval jídelní lístek menu a objednávání položek z něho. Naučil jsem se pracovat s pro mě novým Javascriptovým frameworkem React.js, který

umožňuje pohodlně dynamicky překreslovat HTML obsah stránky bez nutnosti jejího opětovného načtení.

Obrazovka Zaplatit nabízí platbu za položky účtu, jejich rušení a uzavření účtu i v případě, že všechny položky na něm nebyly zaplaceny.

Aplikace dovoluje přesouvat objednané položky mezi zvoleným účtem a libovolným jiným.

Funkcionalitu tisku jsem implementoval do té míry, že je možné vytisknout si účtenku do souboru PDF.

17.3. Zhodnocení

Původní časový odhad náročnosti tohoto projektu byl 370 hodin. Reálný čas na něm byl přibližně 330 hodin.

Všechny hlavní části systému jsou zdokumentované a otestované Selenium testy. Při práci se mi z časových důvodů nepovedlo provést testování uživatelské přívětivosti. Aplikace je nasazená a funkční na školním testovacím prostředí na linuxovém serveru.

18. Bibliografie

[1] „Cashbob,“ [Online]. Available: www.cashbob.cz.

[2] V. Samek, Nasazení a propagace restauračního systému, květen 2015.

[3] M. Dráb, „Iterativní vývoj,“ Goshoom.NET Dev Blog, 7 12 2015. [Online]. Available: http://dev.goshoom.net/2011/09/iterativni-vyvoj/.

[4] „Play Framework,“ 7 10 2015. [Online]. Available: https://www.playframework.com/.

[5] „Netty: Home,“ 13 5 2016. [Online]. Available: http://netty.io/.

[6] „H2 Database Engine,“ 22 9 2015. [Online]. Available: http://www.h2database.com/html/main.html.

[7] „Bootstrap • The world's most popular mobile-first and responsive front-end framework,“ [Online]. Available: http://getbootstrap.com/.

[8] „Foundation | The Most Advanced Responsive Front-end Framework from ZURB,“ [Online]. Available: http://foundation.zurb.com/.

[9] „Getting started | Less.js,“ 13 5 2016. [Online]. Available: http://lesscss.org/.

[10] „Sass: Syntactically Awesome Style Sheets,“ 13 5 2016. [Online]. Available: http://sass-lang.com/.

[11] „jQuery,“ [Online]. Available: https://jquery.com/.

[12] „A JavaScript library for building user interfaces | React,“ 13 5 2016. [Online]. Available: https://facebook.github.io/react/.

[13] „Selenium - Web Browser Automation,“ 1 12 2015. [Online]. Available: http://www.seleniumhq.org/.

[14] t. f. e. Wikipedia, „Coding conventions,“ 7 12 2015. [Online]. Available: https://en.wikipedia.org/wiki/Coding_conventions.

[15] Oracle, „Code Conventions for the Java Programming Language: 1. Introduction,“ 7 12 2015. [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-139411.html#16712.

[16] Oracle, „Code Conventions for the Java Programming Language: 5. Comments,“ 7 12 2015. [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-141999.html#385.

[17] „Ehcache,“ [Online]. Available: http://www.ehcache.org/.

[18] „HTTP ETag - Wikipedia, the free encyclopedia,“ 13 5 2016. [Online]. Available: https://en.wikipedia.org/wiki/HTTP_ETag.

[19] „JasperReports® Library | Jaspersoft Community,“ 25 5 2016. [Online]. Available: http://community.jaspersoft.com/project/jasperreports-library.

[20] „Apache(tm) FOP - a print formatter driven by XSL formatting objects (XSL-FO) and

an output independent formatter.,“ 25 5 2015. [Online]. Available: https://xmlgraphics.apache.org/fop/.

[21] „Apache PDFBox | A Java PDF Library,“ 26 5 2016. [Online]. Available: https://pdfbox.apache.org/.

[22] J. Krutina, Ladění restauračního systému, květen 2014.

[23] J. Kubíček, Nasazení a údržba restauračního systému, květen 2014.

[24] A. Makarič, Modulární architektura restauračního systému, květen 2014.

[25] J. Pýcha, Příprava restauračního systému na reálné nasazení, květen 2014.

[26] F. Dyrčík, Modul pro plánování směn systému CashBob, květen 2014.

[27] J. Rohlíček, Úprava a příprava na reálné nasazení restauračního systému CashBob, leden 2014.

[28] T. Lukáš, CashBob – Walk, leden 2013.

[29] T. Apeltauer, Úprava skladového a invetarizačního modulu systému Cashbob, květen 2012.

[30] M. Kosek, Modul pro plánování směn systému CashBob, leden 2012.

[31] Š. Tesař, Úprava a příprava na reálné nasazení systému CashBob, květen 2012.

[32] T. Hnízdil, Pokladní modul a jeho integrace do systému pro správu restaurace, květen 2010.

[33] J. Jambor, Fyzická architektura a nasazení systému pro restaurace, květen 2010.

[34] J. Mlejnek, Skladový a administrativní modul softwaru pro správu restaurace, červen 2009.

19. Příloha: Obsah CD

CD obsahuje následující:

- složka **cashbob_play** – složka obsahující zdrojové soubory aplikace restauračního systému včetně knihoven a testovací databáze.
- **Webové rozhraní restauračního systému.pdf** – bakalářská práce.

20. Příloha: Návod na spuštění

Návod na spuštění webové aplikace CashBob na systému Windows.

20.1. Prerekvizity

1. Pokud si nejste jistí, jestli máte správnou verzi Javy, otevřete příkazový řádek a napište:
   Java –version
   Pokud máte nainstalované Java Runtime Environment (JRE), version 1.8.x, můžete následující kroky přeskočit a pokračovat kroky 20.2.
2. Stáhněte si a nainstalujte JRE, nejlépe verzi 8u60 na stránce Oracle. Odkaz na stažení 64-bitové verze: http://download.oracle.com/otn-pub/Java/jdk/8u60-b27/jre-8u60-windows-x64.exe .
3. Ve Windows proměnných nastavte vytvořte proměnnou JAVA_HOME (návod, jak se tam dostat: http://www.computerhope.com/issues/ch000549.htm), která odkazuje do adresáře, kde je Java nainstalovaná (typicky *c:\Program Files\Java\jre1.8.x*). Do proměnné Path dopište *";%JAVA_HOME%\bin;".*
4. Restartujte počítač.

20.2. Spuštění aplikace

1. Aplikaci je třeba stáhnout. Je zabalena v souboru **cashbob-x.x.zip**, kde je x.x číslo verze aplikace.
2. V balíčku je složka **cashbob-x.x**. Tu si vykopírujte do libovolného místa na disku.
3. Konfigurace v souboru **conf/application.conf**.
   3.1. Ve složce cashbob-x.x/db je obsažený databázový soubor cashbob.h2. K tomu je potřeba nastavit cestu. Ta je pod klíčem **db.default.url**. Cesta je ve formátu **jdbc:h2:cesta_k_cashbob-x.x/db/cashbob**.
   3.2. Volitelně můžete nastavit **timeout**, po kterém bude přihlášený uživatel odhlášen z aplikace. Defaultně je nastavený na 20 minut.
4. Volitelně můžete nastavit port, na kterém bude aplikace běžet. Defaultně je nastavený na 9000. Pokud si ho přejete změnit, otevřete soubor **cashbobStart.bat**. Zde přepište číslo na řádku **SET PORT=9000**.
5. Spuštění aplikace se provádí spuštěním souboru **cashbobStart.bat**.

21. Příloha: Dokumentace REST API

21.1. Account

21.1.1. GET /rest/account

Popis: Vrátí seznam všech otevřených účtů

Ukázka request: prázdný

Kód 29: Ukázka response – GET /rest/account.

```
{
  "uri":"http://localhost:9000/rest/account",
  "accs":[
    {
      "uri":"http://localhost:9000/account/2",
      "table":{viz Table},
      "id":2,
      "name":"ADMIN",
      "createdate":1325372400000,
      "note":"",
      "user":{
        viz User
      },
      "category":null,
      "opened":true,
      "orders": null // musí se pro ně zavolat detail
    }
  ]
}
```

21.1.2. GET /rest/account/{id}

Popis: Vrátí detail účtu (včetně orders).

{id}: ID účtu, jehož detail chceme.

Kód 30: Ukázka response – GET /rest/account/{id}.

```
{
    "uri":"http://localhost:9000/rest/account/1",
    "table":null,
    "id":1,
    "name":"First account",
    "createdate":1325372400000,
    "note":null,
    "user":null,
    "category":null,
    "opened":false,
    "orders":[
        {
            "uri":null,
```

```
            "id":942007,
            "ispaid":true,
            "time":1365756884700,
            "isaccepted":true,
            "iscanceled":false,
            "price":39.0
        }
    ]
}
```

21.1.3. POST /rest/account

Popis: Vytvoření nového účtu.

Kód 31: Ukázka request – POST /rest/account.

```
{
    "table": {
      "id":1 // nepovinne
    },
    "user": {
      "id":2 // nepovinne
    },
    "category": {
        "id":3 // nepovinne
    },
    "name":"text",
    "note":"text"
}
```

Ukázka Response: výsledek operace

21.1.4. POST /rest/account/{id}/order

Popis: Přidání objednávky k účtu.
{id}: ID účtu, ke kterému chceme přidat objednávku.

Kód 32: Ukázka request – POST /rest/account/{id}/order.

```
{
    "isUnaccepted": false,
    "items": [
        {
            "menuItemId":1,
            "count": 1
        }
    ]
}
```

Ukázka Response: výsledek operace

21.1.5. POST /rest/account/{id}/payItems

Popis: Zaplatí vybrané položky objednávky.
{id}: ID účtu, který chceme platit.

Kód 33: Ukázka response – POST /rest/account/{id}/payItems.

```
{
    "items": [
        {
            "menuItemId":1,
            "count": 1
        }
    ]
}
```

Ukázka Response: výsledek operace

21.1.6. POST /rest/account/{id}/moveItems

Popis: Zaplatí vybrané položky objednávky.
{id}: ID účtu, ze kterého přesouváme.

Kód 34: Ukázka request – POST /rest/account/{id}/moveItems.

```
{
    "targetAccId":28,
    "items": [
        {
            "menuItemId":1,
            "count": 1
        }
    ]
}
```

Ukázka Response: výsledek operace

21.2. Table

21.2.1. GET /rest/table

Popis: Vrátí seznam všech stolů.
Ukázka request: prázdný

Kód 35: Ukázka response – GET /rest/table.

```
{
    "uri":"http://localhost:9000/rest/table",
    "tables":[
        {
            "uri":"http://localhost:9000/table/0",
            "id":0,
            "tablenumber":14,
            "numberofplaces":4
        }
    ]
}
```

21.2.2. GET /rest/table/{id}

Popis: Vrátí seznam všech stolu podle ID.

Ukázka request: prázdný

Kód 36: Ukázka response – GET /rest/table/{id}.

```
{
    "uri":"http://localhost:9000/rest/table/2",
    "id":2,
    "tablenumber":16,
    "numberofplaces":4
}
```

21.3. Custom menu node

21.3.1. GET /rest/custmenunode/{id}

Popis: Vrátí kompletní menu.

Ukázka request: prázdný

Kód 37: Ukázka response – GET /rest/custmenunode/{id}.

```
{
            "isMenu":true,
            "name":"Jídelní lístek",
            "nameShort":null,
            "price":null,
            "quantity":null,
            "parentMenu":null

        },
        {
            "uri":"http://localhost:9000/custmenunode/60",
            "custMenuNodeId":60,
            "itemId":25,
            "isMenu":false,
            "name":"Kofola",
```

```
            "nameShort":null,
            "price":36.0,
            "quantity":"1",
            "parentMenu":{
                "uri":null,
                "menuid":2,
                "name":"nealko",
                "date":1351852894000,
                "isCustMenu":false
            }
        }
    ]
}
```

21.4. User

21.4.1. GET /rest/user

Popis: Vrátí seznam všech uživatelů.

Ukázka request: prázdný

Kód 38: Ukázka response – GET /rest/user.

```
{
  "uri":"http://localhost:9000/rest/user",
  "users":[
    {
        "uri":"http://localhost:9000/rest/user/admin",
        "id":1,
        "version":1,
        "firstName":"Admin",
        "lastName":"predefined",
        "username":"admin",
        "password":null,
        "personalId":"1",
        "credit":0.0,
        "role":null,
        "updatePassword":null
    }
  ]
}
```

22. Příloha: Popis struktury projektu

Tato kapitola popisuje strukturu projektu CashBobPlay (popis projektu pro development, ne buildu).

22.1. app/

V kořeni projektu je třída Global, která je základním ovladačem pro Play. Globální class, která obsahuje metody spouštějící se při startu aplikace, při zavolní každého requestu nebo zpracování jinak nezachycených exception. Nesmí ani přemístit (umístění muselo někde nakonfigurovat).

TYP	NÁZEV	
class	Global	extends GlobalSettings

22.2. app/cz.cvut.fel.cashbob

V tomto balíčku je obsažená veškerá Java část aplikace.

22.2.1. controllers.actions

Slouží jako anotace nebo jsou v nich použity. Slouží pro ověření autorizace a autentizace při přístupu k metodám a třídám. @interface tvoří vlastní anotaci. Třída CheckLogged je použita v Play anotaci @With.

TYP	NÁZEV	
class	CheckLogged	extends Action.Simple
class	CheckRole	extends Action<CheckRoleAnn>
@interface	CheckRoleAnn	

22.2.2. controllers.exc

Obsahuje třídy vlastních výjimek. Tyto třídy jsou použity převážně v balíčku controllers.rest.

TYP	NÁZEV	
abstract class	CashBobException	extends Exception
class	CashBobRuntimeException	extends RuntimeException
class	EntityException	extends CashBobRuntimeException
class	Messages	
class	ValidationException	extends CashBobException

22.2.3. controllers.pages

Controllery obsahující metody zpracovávající requesty uživatelů. Dále obsahují metody přímo na tyto metody navázané.

Všechny třídy, které slouží pro obsluhu requestů od zalogovaných uživatelů dědí od třídy Logged. Ta mimo jiné hlídá autentizaci.

TYP	NÁZEV	
class	cashdesk.AccountDetail	extends Logged
class	cashdesk.AccountList	extends Logged
class	cashdesk.AccountMove	extends Logged
class	cashdesk.AccountNew	extends Logged
class	cashdesk.AccountPay	extends Logged
class	cashdesk.AccountNew	extends Logged
class	cashdesk.Settings	extends Logged
class	Logged	extends Controller Všechny třídy, které slouží pro obsluhu requestů od zalogovaných uživatelů dědí od této třídy. Hlídá autentizaci.
class	Login	extends Controller Slouží pro obsloužení nezalogovaných uživatelů na stránce login (jediná stránka pro nezalogované uživatele).

22.2.4. controllers.rest

Balíček obsahuje třídy, které obsluhují requesty směřované na REST api. Všechny tyto třídy dědí ze třídy AbstractApi, která obsahuje základní funkcionalitu, jako je například zpracování URL příchozích requestů nebo odesílání univerzálních odpovědí (např. potvrzení, že byl request v pořádku zpracován nebo se naopak nepodařilo ho zpracovat).

Z důvodu velkého množství typově velmi podobných tříd v tomto balíčku nejsou všechny vypsané.

TYP	NÁZEV	
abstract class	AbstractApi	extends Controller
class	zbylé rest api třídy	extends AbstractApi

22.2.5. controllers.svc

Balíček obsahuje servisní třídy, které převážně přidávají logiku před dao třídy.

Z důvodu velkého množství typově velmi podobných tříd v tomto balíčku nejsou třídy vypsané.

TYP	NÁZEV	
class	service třídy	Jsou nezávislé (tzn. žádné třídy nerozšiřují ani neimplementují).

22.2.6. filters

Obsahuje třídy, která přidává funkcionalitu filtrů, totiž tříd, které se volají při každém requestu. Použity jsou filtry pro obranu proti CSRF útokům, pro logování časů requestů a Gzip filter pro kompresi dat odesílaných na klienta.

Umístění této třídy musí být nakonfigurováno v souboru conf/application.conf pod klíčem play.http.filters.

TYP	NÁZEV	
class	Filters	implements HttpFilters
class	LoggingFilter	extends Filter Tato loguje do souboru conf/time.log časy requestů. Třída je napsaná v Scale, nejde napsat v Javě.

22.2.7. model

Tento balíček obsahuje balíčky conv, dao, dto, entity, forms a fop.

Třídy v něm obsažené zastupují záznamy v databázi nebo se zahrnují chování spojené přímo a pouze práce s nimi.

Třídy v něm obsažené nemají závislostí na třídy mimo balíček model.

22.2.8. model.conv

Balíček obsahuje třídy konvertující objekty z balíčku model.entity do objektů model.dto.

Třídy v tomto balíčku jsou závislé pouze na balíčku model.entity.

Z důvodu velkého množství typově velmi podobných tříd v tomto balíčku nejsou třídy vypsané.

TYP	NÁZEV	
class	třídy konvertorů	Jsou nezávislé (tzn. žádné třídy nerozšiřují ani neimplementují).

22.2.9. model.dao

Balíček obsahuje třídy sloužící k přímé práci s databází.

Třídy neobsahují SQL příkazy, ale volají se z nich NamedQueries definované pomocí JPA anotací nad třídami model.entity. Do volaných NamedQueries se dosazují proměnné předané v parametrech dao metod.

Výsledky databázových volání se mapují na objekty model.entity.

Základem je třída AbstractDao<E extends IEntity>, ze které dědí všechny ostatní dao třídy. Obsahuje především univerzální metody společné pro všechny ostatní třídy, které jsou rozlišené podle jednotlivých tříd model.entity. Díky tomu není potřeba pro každou třídu implementovat zvlášť metody pro získání všech entit daného typu, získání entity podle jejího ID nebo odstranění entity z databáze.

Třídy v tomto balíčku jsou závislé pouze na balíčku model.entity.

Z důvodu velkého množství typově velmi podobných tříd v tomto balíčku nejsou všechny třídy vypsané.

TYP	NÁZEV	
abstract class	AbstractDao<E extends IEntity>	
class	zbylé dao třídy	extends AbstractDao<E extends IEntity >

22.2.10. model.dto

Balíček obsahuje obrazy tříd z model.entity připravené tak, aby mohli být například vypsány do XML nebo serializovány do JSON objektů.

Třídy v tomto balíčku nejsou závislé na jiných balíčcích.

Z důvodu velkého množství typově velmi podobných tříd v tomto balíčku nejsou třídy vypsané.

TYP	NÁZEV	
abstract class	AbstractResource	Slouží jako předpis pro řadu dto tříd v tomto balíčku (ne všechny), které z ní dědí parametr URI.
class	zbylé dto třídy	

22.2.11. model.entity

Balíček obsahuje třídy reprezentující záznamy v databázy.

Třídy obsahují JPA anotace, které řeší ORM mapování a NamedQueries obsahující šablony pro databázové dotazy (volají se z tříd v balíčku model.dao).

Z důvodu velkého množství typově velmi podobných tříd v tomto balíčku nejsou třídy vypsané.

TYP	NÁZEV	
interface	IEntity	Nutí implementovaným třídám metody pro získání ID entity.
class	zbylé třídy entit	

22.2.12. model.fop

Obsahuje třídy, z nichž se generují XML soubory obsahující data pro tisk do PDF.

TYP	NÁZEV	
class	OrderXml	
class	ReceiptXml	

22.2.13. model.forms

Obsahuje třídy, na které se mapují POST requesty obsahující data z vyplněných formulářů.

TYP	NÁZEV	
class	FormTable	
class	FormUser	

22.2.14. utils

Třídy obsahující pomocné metody a konstanty.

TYP	NÁZEV	
class	CacheUtils	Třída sloužící pro práci s cache.
class	ConfigUtils	Slouží pro práci s konfiguračním souborem conf/application.conf.
class	DateUtils	Převedení datumů na textové řetězce.
class	EntityUtils	
class	JsonDateAdapter	extends XmlAdapter<Long, Date>
class	LangUtils	Třída pracující s jazykem.
class	MoneyUtils	Převedení peněžních částek na text.
class	PasswordEncoder	Třída poskytující šifrování hesel.
class	RandomString	
class	SecurityUtils	Třída zajišťující zabezpečení.
class	SessionUtils	Cokoli, co se ukládá do session v prohlížeči uživatele prochází tudy.
class	TextUtils	Slouží pro vypisovaní lokalizovaných textů.

22.3. app/views

Obsahuje scala šablony (template), ze kterých se generují HTML stránky pro zobrazení.
Všechny šablony mají koncovku *.scala.html.

NÁZEV	POPIS
main	Hlavní šablona pro všechny ostatní stránky. Obsahuje css, js, font importy. Obsahuje favicon.

22.3.1. bits

Složka obsahuje šablony, které neslouží pro vykreslení celých stránek. Tyto šablony jsou opakovaně použity pro skládání aplikace (jsou použity v jiných šablonách). Jsou krátké.

NÁZEV	POPIS
puretextModální okno	Modální okno, který se zobrazí automaticky po načtení stránky, pokud v něm něco je (ukotvený v main šabloně). Obsahuje pouze textové upozornění.
text	K zobrazené lokalizovaného textu ze souborů messages.cz-CZ a .en-GB.

22.3.2. cashdesk

Složka obsahuje šablony pro vykreslení stránek Pokladny

NÁZEV	POPIS	DO ŠABLONY
accountDetail	Stránka detailu účtu.	cashdesk_template
accountEdit	Stránka úpravy účtu.	cashdesk_template
accountList	Stránka se seznamem účtů.	cashdesk_template
accountMove	Stránka pro přesun položek mezi účty.	cashdesk_template
accountNew	Stránka pro tvorbu nového účtu.	cashdesk_template
accountPay	Stránka pro zaplacení účtu.	cashdesk_template
cashdesk_template	Slouží jako šablona pro ostatní stránky v Pokladně.	main
editMenu	Stránka pro úpravu menu (neimplementovaná).	cashdesk_template
editTable	Stránka pro úpravu stolů v restauraci.	cashdesk_template
editTable	Stránka Editace stolů.	cashdesk_template

22.3.3. home

Obsahuje šablonu pro Domovskou stránku aplikace (rozcestník po přihlášení).

NÁZEV	POPIS	DO ŠABLONY
home	Domovská stránka.	main

22.3.4. login

Složka obsahuje šablonu pro vykreslení login stránky.

NÁZEV	POPIS	DO ŠABLONY
login	Login stránka.	login

22.4. public/

Složka pro obrázky, Javascript a css soubory.

22.5. conf/

Složka obsahuje konfigurační a jazykové (lokalizační) soubory.

SOUBOR	POPIS
application.conf	Hlavní konfigurační soubor pro aplikaci. Nastavení umístění filtrů, připojení k databázi, JPA, timeoutu přihlášení a jiné.
fopConfLin.xconf	Soubor s konfigurací pro knihovnu FOP na Linuxu.
fopConfWin.xconf	Soubor s konfigurací pro knihovnu FOP na Windows.
logger.xml	Soubor s konfigurací logování.
messages.cz	Všechny texty pro zobrazení lokalizované v češtině.

messages.en	Všechny texty pro zobrazení lokalizované v angličtině.
routes	Zde je řečeno, jakou URL bude obsluhovat jaká Java metoda.
META-INF/ persistance.xml	Nastavení hibernate.

22.6. db/

Složka obsahuje databázi.

SOUBOR	POPIS
casbob.h2.db	Soubor databáze.
Import_all.sql	Instalační script včetně testovacích dat.

22.7. lib/

Složka obsahuje knihovny nutné k běhu aplikace. Knihovny zde nesmí být dále větvené do podsložek, jinak si je Play framework nedovede bez dodatečné konfigurace načíst.

22.8. logs/

SOUBOR	POPIS
application.log	Zde se loguje běh aplikace. Z javy do něj zapisuji pomocí Logger.info().
times.log	Zde se loguje čas, za který se provedou requesty a z jeké session jsou volány. Může sloužit pro statistiku a ladění.

22.9. selenium/

Testovací selenium scripty.