ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Diplomová práce

Multiplatformní aplikace pro sběr a zpracování dat o kvalitě produktů

Jan Zdrha

12. 5. 2014

Prohlášení

Prohlašuji, že jsem zadanou diplomovou práci zpracoval sám s přispěním vedoucího práce a konzultanta a používal jsem pouze literaturu v práci uvedenou. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé diplomové práce se souhlasem katedry.

V Praze dne…….. podpis autora ……..

Zadání diplomové práce

Abstrakt

Diplomová práce se věnuje vývoji webové aplikace pro mobilní sběr dat o kvalitě produktů. V úvodu práce je diskutována důležitost odvětví zabývající se managementem kvality, doplněn je i podrobnější popis problémové domény. Dále již nastupuje část práce věnující se vývoji samotné aplikace. Struktura práce zde odpovídá postupu vývoje software dle metodiky Unified Process. V rámci vývoje aplikace prochází práce všemi aktivitami definovanými metodikou. Se zákazníkem bylo na začátku provedeno několik konzultací kvůli sběru požadavků, následně byly požadavky analyzovány a tak byla ověřena jejich konzistence. Dále byly zpracovány požadavky na platformu, technologie a software, jejichž využití požadovalo IT oddělení zákazníka. V dalších fázích byl vytvořen na platformě nezávislý návrh a provedena implementace. V implementační části jsou provedena srovnání několika knihoven a frameworků, které je možné pro realizaci různých částí aplikace využít. Následuje testování, které ověří funkčnost aplikace vzhledem k požadavkům. V závěru práce ukazuje další požadavky uživatelů na rozvoj.

Abstract

This thesis is dedicated to developing a web application for mobile data collection in field of quality management. The introduction discusses the importance of quality management, followed by a detailed description of the problem domain. Next few parts of the thesis are dealing with development of the application itself. Structure of these chapters corresponds to the Unified Process, a software developing methodology. As application develops, thesis goes through all the activities defined by methodology. At start of the development there were several meetings was held with customer. At these meetings the requirements were defined by customer. Then the requirements were analyzed and also their consistency was ensured. Furthermore, the application platform, technology and software to use through application development, were defined by customer's IT department. In subsequent phases a platform independent design was created and application was implementated. A library and framework comparison were made to decide which libraries and frameworks should be used. Finally the tests were performed and discussed because of verify application with respect to requirements. The conclusion shows another user requirements found during tests and discusses posibilities of future application development.

Obsah

1 Úvod

Kvalita potravin a obecně zboží je jedním z nejčastějších požadavků zákazníků, je proto přirozené, že všechny společnosti věnují tomuto tématu velkou pozornost. Společně s pozorností upřenou tímto směrem dochází ke značnému zlepšení zejména díky školení zaměstnanců, modernizaci a standardizaci postupů. V rámci stálého zlepšování, které je v konkurenčním prostředí nezbytnou podmínkou úspěchu je potřeba provádět pravidelné kontroly, ty následně vyhodnocovat a poskytovat zpětnou vazbu zodpovědným lidem.

Původně byla oddělení či zaměstnanci pečující o kontrolu kvality vybaveni primitivními nástroji: papír, tužka, teploměr a případně fotoaparát. Společně s pozorností věnovanou kvalitě se tento stav mění, je potřeba zaměstnance lépe vybavit a zefektivnit jejich práci tak, aby mohlo přibývat kontrol, staly se pružnějšími, usnadnilo se vyhodnocování a následný reporting, a také se standardizovalo řešení připomínek.

Právě na tuto vyhodnocovací část procesu zlepšování kvality se zaměřuje moje práce - Multiplatformní aplikace pro sběr a zpracování dat o kvalitě produktů.

1.1 Cíl práce

Cílem této práce je vytvořit softwarovou pomůcku – aplikaci, která bude uživatelům usnadňovat práci při sběru a zpracování dat. Základem je poznat práci uživatelů, zjistit její návaznost na jiné procesy a určit její výstupy. V návaznosti na to navrhnout a realizovat aplikaci, která nahradí dosavadní primitivní pomůcky (papír, tužka + Excel pro zpracování), umožní napojení na již existující systémy a zefektivní práci při vyhodnocování výsledků. V závěru bude aplikace otestována na scénářích vytvořených dle průběhu skutečného auditu.

1.2 Postup

Postup se bude řídit standartní metodikou vývoje softwaru Unified process (UP) [1]. Práce tedy postupně projde všemi kroky metodiky UP:

1. Průzkum problémové domény
2. Sběr a zpracování požadavků
3. Analýza
4. Návrh
5. Implementace
6. Testování
7. Nasazení

Dále podrobněji popíši jednotlivé části postupu.

1.2.1 Průzkum problémové domény

V této části se seznámím se současnými postupy a termíny z oblasti QA. Navštívím skutečný audit a prozkoumám současné pomůcky. Budu sledovat komunikaci budoucích uživatelů s odděleními a systémy se kterými komunikují a seznámím se s jejich požadovanými vstupy a výstupy. Výsledkem průzkumu problémové domény bude popis chování uživatelů a jejich okolí - tedy identifikace účastníků a jejich interakcí.

1.2.2 Sběr a zpracování požadavků

V rámci sběru požadavků vyjdu z průzkumu problémové domény a provedu diskusi se zadavatelem. Důležité bude definovat potřebnost a efektivnost jednotlivých částí optimalizace procesu, a tedy stanovení rozsahu jednotlivých fází realizace. To bude důležité pro definici postupného iterativního vývoje, tak aby uživatelé mohli jednotlivé funkční dodávky testovat a připomínkovat a měli tak pod kontrolou směřování vývoje systému. Výsledkem bude seznam požadavků na výslednou aplikaci.

1.2.3 Analýza

V analytické části analyzuji požadavky na architekturu aplikace, dále definuji aktory a případy užití a vytvořím analytické třídy. V této části budu řešit případné konflikty požadavků, zejména bude potřeba zpozornět v blízkosti hranice mezi jednotlivými částmi definovanými dle oprávnění uživatelů. Výstupem bude konzistentní seznam aktorů a akcí a ověření realizovatelnosti nefunkčních požadavků. Odhalené nedostatečně definované požadavky budou upřesněny.

1.2.4 Návrh

V rámci návrhu provedu rozdělení aplikace do jednotlivých modulů dle funkčních celků s jasně definovanými odpovědnostmi. Vytvořím diagram nasazení. Definuji datový model, návrhové třídy a jejich interakce, práva jednotlivých uživatelů.

1.2.5 Implementace

Ve fázi implementace vyberu knihovny a frameworky vhodné pro nasazení na definovaných systémech, tak aby bylo jejich použití efektivní pro daný účel a přitom byly otevřené rozšířením do budoucna (v dalších fázích). Následně realizuji – implementuji samotnou aplikaci. Výstupem tedy bude nasazená aplikace umožňující testování.

1.2.6 Testování

V této aktivitě provedu ověření splnění požadavků definovaných v kapitole Požadavky. Provedu testy aplikace nasazením v testovacích podmínkách s použitím sady testovacích dat. Výsledky testů porovnám s výstupy z původních auditů, ze kterých byla vytvořena testovací data. Tím bude prokázána správná funkce aplikace. Výstupem bude sada testovacích dat (která bude sloužit následně i uživatelům při inicializaci aplikace) a výsledky testů.

1.2.7 Nasazení

Pro účely této práce nasadím aplikaci pouze na testovacím serveru, ten umožní ověření funkčnosti. Testovací nasazení bude provedeno ve fázi Testování. Výstupem bude aplikace nasazená v testovacím prostředí zákazníka společně s připravenými inicializačními skripty databáze a sadou testovacích dat umožňující hladký přechod na provoz s dodanou aplikací.

1.3 Integrace, využití hardware

Důležitou myšlenkou je maximální využití možností, které jsou multiplatformním aplikacím v současné době poskytovány. V této kapitole se budu věnovat popisu těchto možností a jejich využití v realizované aplikaci. Dále budu diskutovat možnosti použití aplikace v jiných provozech.

1.4 Závěr

V závěru práce provedu vyhodnocení výsledků testů oproti požadavkům. Načrtnu možnosti rozšíření a praktického nasazení aplikace.

2 Rešerše

V rámci rešerše se naučím pracovat s pojmy problémové domény, poznám činnosti kontrolorů kvality, jejich pomůcky a postupy. Dále se seznámím s prostředím, kde pracují, kritérii a měřítky které používají.

2.1 Základní pojmy

V této kapitole proberu používané názvosloví kontextu práce, tak jak jsem je po seznámení s prostředím a s uživateli pochopil. Zde uvedené pojmy jsou používané přímo v oddělení, pro které aplikaci vyvinu. Tyto pojmy se mohou odchylovat od těch obecně používaných. Pro správnou a přesnou komunikaci se zákazníkem je ale výhodné používat jeho pojmy, přispívá to porozumění mezi oběma stranami.

2.1.1 Auditor

Auditor je pracovník, který provádí kontrolu, je vybaven zápisníkem a psacími pomůckami, teploměrem a případně fotoaparátem.

2.1.2 Store

Store neboli obchod, prodejna. Jako obchod uvažuji celý komplex, který čítá části pro příjem zboží, sklady, plochy kde je zboží vystaveno, prostory pro zákazníky i zaměstnance, kantýnu, restauraci která může být součástí obchodu, parkoviště a přilehlou infrastrukturu. Často je součástí také benzinová pumpa. Story jsou si navzájem velmi podobné, je tedy možné je pomocí auditu i srovnávat. Store je rozdělen na departmenty.

2.1.3 Department

Department je část obchodu skládající se ze subdepartmentů. Deparment zastřešuje logickou skupinu subdepartmentů. Například Department „Čerstvé zboží“ obsahuje zboží s relativně krátkou dobou trvanlivosti tedy pro představu zboží z oddělení masa, mléka, zeleniny, ryb atd. Na druhou stranu neobsahuje například zboží z oddělení drogerie nebo s dlouhou dobou trvanlivosti například nápoje.

2.1.4 Subdepartment

Depratment je oddělení obchodu. Oddělení jsou členěna podle druhu nabízeného zboží, například maso, mléko, drogerie atd. Reálně do departmentu patří všechny části storu se kterými přijde zboží daného oddělení do styku a kde není společně se zbožím jiného oddělení. Do departmentu tedy patří část obchodu, kde je zboží uskladněno, zpracováváno a vystavováno.

2.1.5 Typ auditu

Existuje několik typů auditů, každý z nich se zaměřuje na jinou část procesu, největším auditem je audit zpracování a prodeje zboží. Dalšími typy auditů jsou například audit přepravy a audit příjmu.

2.1.6 Audit

Audit je základní jednotkou práce oddělení, má samostatné vstupy a výstupy. Samotný audit je procedura, při které dojde ke kontrole jednoho storu z nějakého pohledu (dle typu auditu). Auditu se většinou účastní alespoň dva auditoři. Následně je daný audit vyhodnocen. Tím se získá hodnocení pro daný store a daný typ auditu v konkrétním čase. Audity jsou vzájemně porovnatelné, jednak mezi sebou historicky pro stejný store a také mezi jednotlivými story. Audit se skládá z checklistů.

Název společnosti	
Store:	**České Budějovice**
Audit date :	**15.4.2014**
Last audit date:	**15.10.2014**
Auditor :	**Jan Zdrha**
Report checking and approval:	**Administrátor**
Points achieved/ Získáno bodů	**724**
Maximum achievable points/ Maximálně dosažitelných bodů	**732**
Percent of points achieved/ Celkové hodnocení obchodu	**98,9%**
Target of audit 2013/ Plán auditu na rok 2013	**88,0%**

Obrázek 1: Shrnutí výsledků auditu

2.1.7 Checklist

Checklist je sada otázek, každá sada patří k určitému oddělení, jsou tedy například jiné otázky pro oddělení masa a jiné pro oddělení zeleniny. Checklist je organizován podle kategorií otázek. Vyhodnocení auditu vzniká z vyhodnocení checklistů, každý checklist je tedy vyhodnocen nezávisle na ostatních. V některých případech se audity mezi jednotlivými story porovnávají i na úrovni checklistů.

Summary of results and recommendation/ Shrnutí výsledků auditu a doporučení auditora:		
	Department	**Score**
Fresh	A. Meat / Maso	92,2%
	B. Dairy / Mléko	100,0%
	C: Delicatessen / Delikatesy	100,0%
	D. Frozen / Mražené	100,0%
	E. Fish / Ryby	100,0%
	F. F&V / O&Z	100,0%
Dry	G. Basic food / Koloniál	100,0%
	H. Beverages / Nápoje	100,0%
	I. Detergents, Cosmetics / Drogérie	100,0%
	J. Others / Obecně	100,0%
Findings	Main non-conformities found during audit:	
RESULT ZÁVĚR	**A- B – Satisfactory / Splněno** **C - D – Action required / Nutná akce** **E - F – Not satisfactory / Nesplněno**	

Obrázek 2: Shrnutí výsledků auditu - checklisty

2.1.8 Kategorie otázek

Otázky jsou, kvůli přehlednosti, členěny v rámci checklistů do kategorií. Jednotlivé kategorie jsou například Kvalita, Hygiena, Dokumentace atd. Kategorie jsou zavedené jen kvůli přehlednosti checklistů. Kategorie nemá kromě názvu a vazby na otázky, které do ní spadají žádné další atributy.

2.1.9 Otázka

Otázka je základním stavebním kamenem reportu, je tvořena textem otázky a vahou otázky. Různým otázkám je přikládána při vyhodnocení různá důležitost, která je vystižena právě vahou otázky, čím vyšší váha tím důležitější otázka. S otázkou je spojeno hodnocení, neboli „score“. Otázka je hodnocena známkou. Možné známky jsou A, B, C, NEHODNOCENO. Více k hodnocení otázek je uvedeno níže v části Hodnocení.

S hodnocením otázky jsou spjaté další doplňující informace, jedná se o seznamy výrobků, poznámku, popis neshody (problému) a fotodokumentaci.

Poznámka popisuje malý prohřešek nebo jinou připomínku auditora, za kterou není sníženo hodnocení. Jedná se například o doporučení, nebo upozornění na nově zaváděné normy, které budou v následujících auditech již zohledněny jako neshoda.

Neshoda upřesňuje, co konkrétně hodnotil auditor jako prohřešek při odpovědi na danou otázku, otázky jsou většinou obecné („Hygienické podmínky (WC pro zákazníky), záznamy o úklidu“) a tak je pro pozdější komunikaci dobré mít podrobnější popis problému („Na dámských toaletách ve druhém patře nebyl v době kontroly, tedy 13:30, dostatek toaletního papíru“). Většina otázek se sníženým hodnocením má popis neshody, není to ovšem podmínkou.

Seznamy výrobků je možné zadávat u otázek, které se týkají konkrétních zalistovaných výrobků. Jde například o otázky dotazující se na počet nalezených výrobků po datu minimální trvanlivosti, případně podobné otázky týkající se skladování za určitých podmínek (mrazák) nebo neporušenost obalů. V těchto seznamech je potřeba vyplnit kolik kusů daného zboží bylo nalezeno, jaký je jeho název, problematické datum a případně šarže.

Fotodokumentace slouží jako další užitečná pomůcka při zaznamenávání problémů a prezentaci výsledků. Dle hesla „obrázek vydá za tisíc slov“ je v terénu, kde je psaní zdlouhavé a nepohodlné, značně využíváno právě focení problémů.

A. FRESH - ODDĚLENÍ MASA				Hodnota parametru	Celkové hodnocení	kategorie
Kvalita	A.1	Prodej výrobků v souladu s legislativou (doba použitelnosti, minimální trvanlivosti, odpovídající kvalita...) **3 a více** artiklů v rámci celého oddělení automaticky hodnocení C	○ A ○ C ◉ B ○ nehod.	10	5	pole počet výrobků po expiry název + datum spotřeby
	A.2	Dodržování tabulek čerstvosti - čerstvé maso (nezahrnujeme výrobky po DMT nebo DS) **2 a více=C**	○ A ◉ C ○ B ○ nehod.	3	0	pole počet výrobků po TČ název + datum spotřeby
	A.3	Dodržování tabulek čerstvosti – mražené maso (nezahrnujeme výrobky po DS nebo DMT) 2 **a více=C**	◉ A ○ C ○ B ○ nehod.	3	3	pole počet výrobků po TČ název + datum spotřeby
	A.4	Dodržování tabulek čerstvosti – uzeniny (nezahrnujeme výrobky po DS a DMT) 3 a více=C	◉ A ○ C ○ B ○ nehod.	3	3	pole počet výrobků po TČ název + datum spotřeby
	A.5	Dodržení pravidla FIFO (prodejní plocha, zázemí)	◉ A ○ C ○ B ○ nehod.	3	3	
	A.6	Prezentace na prodejní ploše (nezakrytá chladící ventilace, množství zboží v regále)	◉ A ○ C ○ B ○ nehod.	1	1	
	A.7	Chladící řetězec - prodejní plocha (výrobky mimo chladící box, nedostatečná teplota zboží v regále...)	◉ A ○ C ○ B ○ nehod.	10	10	
	A.8	Chladící řetězec – zázemí (zboží na chodbách, skladování zboží v připravně...)	◉ A ○ C ○ B ○ nehod.	5	5	

Obrázek 3: Checklist s hodnocením

2.2 Příprava auditu

Audity vznikají jako sady otázek, které se postupně doplňují tak aby pokryly všechny oblasti, které chce management sledovat. V současnosti existuje víceméně ustálená sada, která prochází drobnými změnami. Pracuje se na dalších typech auditů.

Audit je složen z checklistů a otázek, jeho hierarchické dělení je určeno rozdělením storů na oddělení a je tedy v zásadě neměnné. Změny oddělení se očekávají, ale jsou spíše pomalé. Rychlejší je vývoj otázek, k jejich drobným úpravám dochází prakticky při každém auditu, změny se netýkají samotných otázek ale upřesňování toho co je hodnoceno a jakou srážkou score je to postihováno. Do textu otázky jsou tedy doplňována další doporučení co v rámci auditu kontrolovat. Tak dochází k postupnému zpřesňování otázek a kritérií.

2.2.1 Kalibrační audit

Kalibrační audit je školení auditorů konané alespoň jednou ročně, v rámci tohoto auditu je prováděno hodnocení storu jako při běžném auditu s tím rozdílem že všichni auditoři pracují společně a probírají a sjednocují si navzájem svá hodnotící kritéria tak, aby následně až budou hodnotit obchody samostatně, dosahovali na stejných problémech stejných hodnocení. Právě při tomto auditu dochází nejčastěji k úpravám otázek a kritérií hodnocení.

A.1	Prodej výrobků v souladu s legislativou (doba použitelnosti, minimální trvanlivosti, odpovídající kvalita...) **3 a více** artiklů v rámci celého oddělení automaticky hodnocení C	○ A ○ C ◉ B ○ nehod.	10	5

Obrázek 4: Otázka s hodnocením

2.3 Audit

Audity probíhají náhodně, praxe vyžaduje, pro praktické vykonání auditů v průběhu roku, jeden audit týdně, většinou se audity konají v úterý, středu nebo čtvrtek. Který den a store bude vybrán, není samozřejmě předem známo. Oddělení kontroly kvality v určený den ráno vyrazí do určitého storu, kde se ohlásí a s pracovníky jednotlivých oddělení postupně projde jim svěřená oddělení a hodnotí, dle zadaných otázek, situaci na daném místě. Vše je zapisováno a konzultováno s místním pracovníkem a případně fotodokumentováno. Místní pracovníci tak mají okamžitou zpětnou vazbu na to, co je špatně. Po prozkoumání celého obchodu je audit kompletní.

2.4 Vyhodnocení

Vyhodnocení následuje po vykonání auditu, většinou další den. Pracovníci, kteří vykonávali audit, vytvářejí k auditu dokumentaci. Jedná se o vyplnění sešitu MS Excel, kde jsou uvedeny jednotlivé otázky, ke kterým je potřeba zadat odpovědi, pomocí vzorců a zadaných (pevně daných) vah otázek je následně přepočítáno score jednotlivých otázek. Takto vyplněné checklisty jsou následně jeden po druhém hodnoceny. Každý checklist tak získá své score. Následně, po vyhodnocení všech checklistů je vyhodnocen i celý audit. A daný obchod obdrží známku dle hodnocení auditu. Podrobně je hodnocení popsáno níže. Takto vyhodnocený audit je odeslán na daný store jako draft hodnocení. Store může zadat k danému auditu připomínky, případně jednotlivé body auditu rozporovat, proto je potřeba mít vše řádně zdokumentované již z místa auditu. Po zpracování připomínek, případně dalších kol připomínek, je audit považován za hotový. Následně je odeslán vedení a archivován společně se všemi materiály. Doba archivace zatím není nijak omezena, archiv je uložen na sdíleném disku. Tímto je proces auditování daného obchodu u konce, každý obchod je zpravidla auditován jednou ročně pro daný typ auditu. Je snaha zvládnout všechny typy auditů v jeden den.

2.4.1 Hodnocení (Scoring)

Hodnocení vychází z bodových koeficientů za jednotlivé otázky a pak je vážením a procentuálními úrovněmi převedeno na score pro checklisty i pro audit.

2.4.1.1 Hodnocení otázek

Otázka je hodnocena pomocí score (A, B, C, NEHODNOCENO). Při hodnocení je využívána váha otázky. Jednotlivým otázkám je přidělena váha od 1 do 10 (není nijak omezeno, v budoucnu může být i vyšší váha). Score pak vystihuje koeficient, který je použit pro přepočet bodového hodnocení otázky. Koeficienty jsou uvedeny v tabulce Tabulka 1: Scoring otázek. Z důvodu rozdílnosti obchodů je zaveden i koeficient váhy tak, aby bylo možné otázku z hodnocení vynechat, pokud není možné store ohodnotit v kritériích vystižených otázkou.

Score	Koeficient hodnocení	Koeficient váhy
A	100%	100%
B	50%	100%
C	0%	100%
NEHODNOCENO	0%	0%

Tabulka 1: Scoring otázek

Příklady ohodnocených otázek jsou uvedeny na následujícím obrázku (Obrázek 5: Vyhodnocené otázky).

			Hodnota parametru	Pomocný parametr	Celkové hodnocení
A.1	Prodej výrobků v souladu s legislativou (doba použitelnosti, minimální trvanlivosti, odpovídající kvalita...) 3 a více artiklů v rámci celého oddělení automaticky hodnocení C	(•) A ( ) C ( ) B ( ) nehod.	10	10	10
A.2	Dodržování tabulek čerstvosti - čerstvé maso (nezahrnujeme výrobky po DMT nebo DS) 2 a více=C	( ) A ( ) C (•) B ( ) nehod.	10	10	5
A.3	Dodržování tabulek čerstvosti – mražené maso (nezahrnujeme výrobky po DS nebo DMT) 2 a více=C	( ) A (•) C ( ) B ( ) nehod.	10	10	0
A.4	Dodržování tabulek čerstvosti – uzeniny (nezahrnujeme výrobky po DS a DMT) 3 a více=C	( ) A ( ) C ( ) B (•) nehod.	10	0	Nehodnoteno

Obrázek 5: Vyhodnocené otázky

Otázka A1 má hodnocení A (4. sloupec). Váha neboli hodnota parametru (5. sloupec) je 10. Výsledná váha je po přepočtení koeficientem 100% také 10 (6. sloupec) a výsledný bodový zisk je po přepočtení koeficientem 100% také 10.

Otázka A2 má hodnocení B. Váha je 10. Výsledná váha po přepočtení koeficientem 100% je také 10. A celkové hodnocení po přepočtení koeficientem 50% je 5. Zde je vidět srážka za danou otázku (5 bodů z 10).

Otázka A3 má hodnocení C. Váha je 10. Výsledná váha po přepočtení koeficientem 100% je také 10. Celkové hodnocení po přepočtení koeficientem 0% je 0 bodů. Zde je také srážka tentokrát všech bodů za danou otázku.

Otázka A4 má hodnocení NEHODNOCENO. Otázku nebylo možné v daném storu vyhodnotit (otázka se týká uzenin, je tedy pravděpodobné že na daném storu se sortiment uzenin neprodává). Z tohoto důvodu je potřeba otázku z celkového hodnocení odstranit, tak aby byla dokumentována, ale neměla vliv na výsledné hodnocení. Otázka má váhu 10. Výsledná váha po přepočtení koeficientem 0% je 0 a celkové hodnocení je po přepočtení koeficientem také 0 (pro přehlednost označeno nápisem „Nehodnoteno“). Otázka má tedy nulovou váhu i nulové hodnocení, ve výsledném hodnocení se tedy neprojeví.

Pokud jsou takto ohodnocené všechny otázky v daném checklistu je možné checklist ohodnotit.

2.4.1.2 Hodnocení checklistu

Checklist je připraven k hodnocení v momentě, kdy jsou ohodnocené všechny jeho otázky. Pro checklisty se uvádí jen procentuální hodnocení, tedy poměr počtu bodů, které byly získány vzhledem k počtu bodů, které bylo možné získat. Pro přehlednost hodnocení se používá obarvení výsledků barvou. Pro výběr barvy slouží procentuální hladiny, z nichž se vybere barva dle celkového hodnocení checklistu.

Body, které bylo možné získat, jsou sumou vah otázek – výsledná váha (tedy ta která reflektuje vynechání otázek s hodnocením NEHODNOCENO). Body, které byly získány, jsou sumou celkových hodnocení jednotlivých otázek.

Například pokud budu uvažovat čtyři otázky zmiňované v kapitole Hodnocení otázek (Obrázek 5: Vyhodnocené otázky) jako jeden celý checklist bude hodnocení vypadat následovně: Checklist má 4 otázky z toho jedna hodnocena jako NEHODNOCENO, všechny o vahách 10. Celková váha otázek tedy bude 30 (10+10+10+0). Počet bodů získaných za jednotlivé otázky je 10+5+0+0 tedy 15. Za daný checklist získal tedy store 15 bodů z 30 možných a celkové hodnocení (procentuální) je tedy 50% (15/30). Otázka A4 byla z hodnocení pomocí koeficientů vynechána.

Pro obarvení hodnocení checklistů jsou určeny úrovně vypsané v tabulce Tabulka 2: Hodnocení checklistů a barevné úrovně

Hodnocení checklistu	Barva
88% - 100%	Zelená
70% - 88%	Oranžová
0% -70%	Červená

Tabulka 2: Hodnocení checklistů a barevné úrovně

Námi hodnocený audit by tedy měl v celkovém hodnocení červenou barvu. Jeho hodnocení je 50% tedy v rozsahu 0% - 70%.

2.4.1.3 Hodnocení auditu

Audit je možné hodnotit po vyhodnocení všech checklistů, ze kterých se skládá. Kritéria pro hodnocení auditu vycházejí z ohodnocených checklistů, hodnotí se jednak podle vah otázek a za ně získaných bodů a také podle procentuálního hodnocení jednotlivých checklistů.

Prvním kritériem je procentuální hodnocení získané jako poměr bodů získaných za všechny hodnocené otázky vzhledem k sumě vah hodnocených otázek.

Hodnocení auditu	Scoring auditu	Barva
88% - 100%	A	Zelená
70% - 88%	B	Oranžová
0% -70%	C	Červená

Tabulka 3: Hodnocení auditu, první kritérium

Druhým kritériem pro hodnocení je, že pokud má audit hodnocení prvního kritéria A musí pro získání tohoto hodnocení mít všechny checklisty ohodnocené známkami A nebo B. Pokud by tedy měl audit hodnocení 90%, ale měl jeden checklist s hodnocením pod 70%, byl by ohodnocen scorem B. Nesplnil by totiž druhé kritérium.

K celkovému hodnocení auditu se ještě doplňuje slovní komentář, kde jsou vyjmenovány nejzávažnější nedostatky. Ten je pak zobrazen ve výsledcích auditu.

2.5 Zveřejnění výsledků

Zveřejňování výsledků probíhá v několika fázích a na několika úrovních, dle fáze dokončení auditu a dle protistrany se kterou je komunikováno.

Audit je možné rozdělit na dvě části. První část je titulní strana auditu složená z parametrů auditu (Obrázek 1: Shrnutí výsledků auditu) a vyhodnocení checklistů a auditu (Obrázek 2: Shrnutí výsledků auditu - checklisty). Druhou částí je pak sada checklistů s odpověďmi (Obrázek 3: Checklist s hodnocením).

Ihned po návratu z auditu jsou údaje získané na auditu zaznamenány do sešitu MS Excel a po vyhodnocení odeslány zpět na store, kde byl audit proveden. Z auditu jsou přitom vyfiltrovány jen otázky se sníženým hodnocením (story rozporují jen tyto otázky), dále audit obsahuje titulní stranu, tedy celkové hodnocení.

Po vyřízení připomínek je audit odeslán vedení společnosti. Zde není potřeba zacházet do podrobností a ukazovat výsledky jednotlivých otázek. Z tohoto důvodu obsahuje tento výstup jen titulní stranu s parametry auditu, hodnocením checklistů (oddělení) a výsledné hodnocení storu.

Pro odeslání kompletních výsledků na store a pro archivaci pak slouží kompletní výstup, což je celý audit s titulní stranou a všemi otázkami i s hodnocením.

3 Sběr požadavků

V části sběr požadavků vyjdu ze znalostí problémové domény popsaných v předchozí kapitole, dále se budu soustředit na materiály předané uživateli, konzultace s nimi. Důležitým prvkem požadavků budou také materiály předané IT oddělením zákazníka, tedy různé specifikace, normy a zvyklosti, které je potřeba pro úspěšné nasazení aplikace v prostředí zákazníka dodržet.

3.1 Definice prostředí

Na začátku projektu byly po konzultacích s IT oddělením zákazníka určeny z jeho strany technologie, které budou při vývoji použity. Rozsahem se jedná o malou aplikaci, která doplňuje základní byznys zákazníka, proto bylo potřeba přizpůsobit se již existující infrastruktuře a zvyklostem. Vznikající aplikace tedy bude využívat existující infrastrukturu. Níže je uveden popis infrastruktury a možností, které poskytuje. Také jsou uvedena rozhodnutí, která vedla k výběru konkrétních technologií, definovaných zákazníkem. Kromě těchto požadavků nebyly k dispozici žádné další obecné (nefunkční) požadavky. Z podstaty vznikající aplikace bylo zřejmé, že se bude jednat o klient-server aplikaci.

3.1.1 Klient

Klientské zařízení mělo umožňovat zadávání scoringu, snadné psaní poznámek a neshod a také pořizování fotografií. Zároveň mělo umožňovat přenášení dat na server kvůli možnosti vzájemného sledování uživatelů, tak aby nevykonávali audit ve stejném oddělení.

Na základě těchto požadavků bylo vybráno zařízení iPad (verze 3) s operačním systémem iOS (verze 7.0). Dále má být aplikace také dostupná z běžného PC pro usnadnění zpracování a případné editace dat.

Co se týče software, byla vybrána webová aplikace určená pro webový prohlížeč. Důvodem tohoto rozhodnutí byla jednak možnost zajistit spuštění na PC a dále existence kontrol při zařazení aplikace do obchodu s aplikacemi, v případě realizace nativní aplikace.

Přes nepříjemnosti nativní aplikace bylo požadováno, aby aplikace vypadala pokud možno podobně jako by se jednalo o nativní aplikaci. Zástupci zákazníka za tímto účelem požadovali vytvoření aplikace ve frameworku Vaadin a jeho addonu Touchkit. Aplikace vytvořené za pomoci tohoto rozšíření jsou v prohlížeči pomocí stylů zobrazované tak že vypadají jako nativní aplikace.

3.1.2 Server

Aplikaci bylo v existujícím prostředí určeno místo na aplikačním serveru JBoss (7.1.1), ten definoval prostředí (knihovny a podmínky běhu) i jazyk implementace (Java). Dále bylo určeno, že jako úložiště dat bude využito databáze Oracle 11g.

Server je umístěn v interní síti zákazníka a je v jeho péči. Stejně tak zálohování a další údržbové práce budou po nasazení aplikace do provozu v režii zákazníka.

S umístěním serveru v intranetu souvisí i nutnost zřízení VPN pro uživatele pohybující se s klientskými zařízeními mimo intranet.

3.1.3 Konektivita

Důležitým aspektem klient sever aplikací je i kanál, který poskytuje spojení mezi klientskými stanicemi a serverem. Některé story jsou vybaveny Wi-Fi sítí, kterou budou klientská zařízení používat. Dále jsou zařízení vybavena mobilním připojením, které bude sloužit jako záložní v případě

problémům s Wi-Fi sítí. Počítače, na kterých budou probíhat finální úpravy a zpracování auditu, jsou vybavena pevným připojením (LAN) a jsou v intranetu.

3.2 Požadavky a zvyklosti IT zákazníka

IT oddělení zákazníka vzneslo požadavky, které je potřeba dodržet, aby mohl hladce proběhnout schvalovací proces a tedy přesun aplikace do produkčního prostředí. Jednalo se zejména o požadavky na pojmenování databázových objektů. Dále pak o požadavky na dokumentaci aplikace a také na verzování a předání zdrojových kódů.

3.2.1 Požadavky na pojmenování databázových objektů

Z důvodů snadného začlenění a následné údržby databázových objektů je potřeba dodržet pravidla, která předali zástupci zákazníkova IT oddělení[1]. Jde o soubor doporučení, z nichž vyberu jen ta, která se týkají aplikací malého rozsahu a která využívají databázi jen pro ukládání dat, tedy například nedefinují procedury a podobné, nadstavbové, vlastnosti. Následuje seznam požadavků na pojmenování.

1. Pro názvy objektů se používá angličtina
2. Používají se celá slova a ne zkratky, pokud to délka názvu dovoluje
3. Prefix zohledňuje jméno projektu (realizovanému projektu byl přidělen prefix „QCS_“)
4. Tabulky používají prefix projektu.
5. Sloupec primárního klíče má stejné jméno jako tabulka, s přidaným postfixem „_NO“
6. Primární klíč používá jméno tabulky s postfixem „_PK“
7. Sekvence používá prefix „SEQ_“ doplněný názvem sloupce pro který je sekvence určena
8. Indexy používají jméno tabulky a postfix „_I“ doplněný o číslo kvůli jedinečnosti pojmenování

3.2.2 Další požadavky

Požadavek na zabezpečení hesel v databázi – hesla budou kódována algoritmem MD5 nebo silnějším, z databáze tedy nebude možné přečíst heslo.

Dále je požadováno, aby aplikace při předání obsahovala stručnou dokumentaci poskytující základní představu o jejích funkcích a struktuře, knihovnách použitých při vývoji, způsobu nasazení a instalace.

Aplikace bude dále dodána spolu s instalačními SQL skripty, které vytvoří potřebné databázové objekty.

3.3 Materiály předané uživateli

Uživatelé mi dodali sešity MS Excel pro dva typy auditů, na osobní konzultaci a při ukázce pak vysvětlili, jak probíhá jeho vyplnění a vyhodnocení. Při samotném auditu jsem pak viděl vyplnění materiálů, které nosí na audit sebou a také přepsání do sešitu MS Excel. Po vyplnění je draft hodnocení auditu odeslán na store. Uživatelé mi také popsali částí auditů, které jsou odesílané dalším zainteresovaným osobám a oddělením.

Uživatelé dále, za pomoci kolegy z IT, vytvořili strukturovaný popis dat a jejich vazeb s popsanou terminologií. Tedy zjednodušený model byznys dat. Také byl definován postup, jak jsou jednotlivé struktury vytvářené (vzhledem k jejich vazbám a byznys procesům). Dále bylo definováno, s jakými daty mají jednotlivé uživatelské role pracovat. Z tohoto lze odvodit seznam uživatelských rolí.

[1] Na přání zákazníka není uveden celý dokument, dovoluji si tedy připojit volný přepis použitých pravidel.

Uživatelé společně s IT dále určili fáze realizace, tedy postup jak budou přidávány nové funkce do vznikající aplikace a termíny, kdy by měly být jednotlivé fáze dodány k testování a používání.

Po předání těchto podkladů následovala řada diskuzí o podobě aplikace, o jejích funkcích a možnostech, které zvolené technologie nabízejí. Z těchto diskuzí vzešel seznam funkčních požadavků na vznikající aplikaci.

3.4 Seznam požadavků

V této kapitole popíši všechny funkční požadavky, které zákazník definoval. Z důvodů stanovení konkrétních technologií a prostředí zákazníkem jsou obecné (nefunkční) požadavky již určeny a za výkon a údržbu prostředí nese odpovědnost zákazník. Z tohoto důvodu se omezím pouze na funkční požadavky.

3.4.1 Funkční požadavky

Funkční požadavky popisují sadu funkcí, které má aplikace z pohledu uživatele. Seznam požadavků je pak nejpřirozenější formou jak tyto požadavky s uživatelem formálně definovat.

1. Uživatelé aplikace
    1.1. Aplikace bude dostupná pouze přihlášeným uživatelům
        1.1.1. Přihlášení do aplikace bude možné pomocí uživatelského jména a hesla
        1.1.2. Uživatel bude mít možnost se z aplikace odhlásit
    1.2. Aplikace bude rozlišovat uživatele na základě jejich rolí
        1.2.1. Role uživatelů jsou: Administrátor, QA, Approval
        1.2.2. Pro roli administrátora budou dostupné všechny funkce aplikace
        1.2.3. Pro roli QA budou dostupné funkce mimo administrační část
        1.2.4. Pro roli Approval jsou dostupné stejné funkce jako pro QA jen s tím rozdílem že uživatel není zahrnut do seznamu auditorů při vstupu do auditu.
2. Dotazníková část
    2.1. V dotazníkové části bude vše potřebné k vykonání auditu
    2.2. Bude umožněno vybrat typ auditu
    2.3. Bude možné založit nový audit
        2.3.1. Nový audit má následující parametry: typ, název, datum vykonání, datum předešlého auditu stejného typu ve stejném storu, cíl auditu (v procentech), kontrolor auditu a store na kterém se audit konal
    2.4. Bude možné odstranit audit, při odstranění bude vyžadováno potvrzení
    2.5. Bude možné vybrat audit daného typu pro vyplnění
        2.5.1. Při přechodu do auditu bude zaznamenán uživatel, který do něj vstoupil, pokud se nejedná o uživatele s rolí Approval
        2.5.2. Audit bude členěn dle checklistů, bude zobrazen název checklistu ve formátu X.Y – Z, kde X je kód subdepartmentu, Y je název deparmentu a Z je název subdepartmentu, dále bude zobrazeno, zda je již checklist kompletně vyplněn, tedy zda bylo odpovězeno na všechny otázky
        2.5.3. Po výběru checklistu zobrazí aplikace seznam otázek daného checklistu
        2.5.4. Každá otázka bude zobrazena na vlastním řádku v seznamu
        2.5.5. Seznam otázek bude obsahovat následující položky
            2.5.5.1. Kategorii otázky

2.5.5.2. Text otázky, je potřeba aby se zobrazil celý
2.5.5.3. Indikátor neshody, zvýrazněný pokud je u dané odpovědi přidána neshoda
2.5.5.4. Indikátor poznámky, zvýrazněný pokud je u dané odpovědi přidána poznámka
2.5.5.5. Indikátor fotografie, zvýrazněný pokud je u dané odpovědi přidána fotografie
2.5.5.6. Váhu otázky
2.5.5.7. Prvek pro zadání score
2.5.5.8. Bodové hodnocení otázky, které se aktualizuje ihned po zadání score
2.5.6. Odpovědi na otázky se budou průběžně ukládat vždy po vybrání score
2.5.7. Bude možné zadat další detaily odpovědi
2.5.7.1. Fotografie
2.5.7.1.1. Přidat fotografii a zobrazit všechny přidané fotografie
2.5.7.1.1.1. Fotografii bude možné vybrat z již pořízených fotografií
2.5.7.1.1.2. Bude možné pořídit novou fotografii
2.5.7.1.2. Bude možné odstranit fotografii, pro odstranění bude vyžadováno potvrzení
2.5.7.1.3. Vkládané fotografie budou zmenšeny (pokud alespoň jedním rozměrem přesahují) na rozlišení 1024x768 pixelů při zachování původního poměru stran
2.5.7.2. Poznámka
2.5.7.2.1. Bude možné přidat k odpovědi textovou poznámku
2.5.7.3. Neshoda
2.5.7.3.1. Bude možné přidat k odpovědi textový popis neshody
2.5.7.4. Seznam výrobků
2.5.7.4.1. K odpovědi bude možné přidat seznam výrobků, pokud to povaha otázky umožňuje
2.5.7.4.2. Seznamy jsou dvou typů: položky po době trvanlivosti a položky po době čerstvosti, dále je potřeba u některých otázek zadat k oběma typům atribut šarže
2.5.7.4.3. Výrobky bude možné do seznamu přidávat i je z něj odstraňovat
2.5.7.4.4. Detaily bude možné průběžně ukládat

2.6. Bude možné prohlédnout výsledky auditu
2.6.1. Ve výsledcích auditu budou zobrazeny stejné informace jako na titulní straně reportu, viz obrázky Obrázek 1: Shrnutí výsledků auditu a Obrázek 2: Shrnutí výsledků auditu - checklisty
2.6.2. Ve výsledcích auditu bude možné doplnit slovní komentář k celému auditu
2.6.3. Bude možné editovat seznam uživatelů, kteří se auditu zúčastnili

2.7. Bude možné vytvořit report z daného auditu
2.7.1. Jsou požadovány 3 typy výstupních reportů (Manažerský, Kompletní a Jen neshody) a dva výstupní formáty (PDF, XLSX)
2.7.2. Manažerský report obsahuje jen titulní stranu
2.7.3. Kompletní obsahuje titulní stranu a všechny otázky a odpovědi i s detaily
2.7.4. Jen neshody obsahuje titulní stranu, a jen ty otázky u kterých je snížené hodnocení
2.7.5. Detailní popis reportů a jejich obsahu je uveden v kapitole Zveřejnění výsledků

2.7.6. Reporty vychází z uživateli předaného původního sešitu Excel s otázkami, kde probíhalo vyhodnocování, ukázky z tohoto sešitu jsou uvedeny na obrázcích: Obrázek 1: Shrnutí výsledků auditu, Obrázek 2: Shrnutí výsledků auditu - checklisty a Obrázek 3: Checklist s hodnocením

3. Administrační část
   - 3.1. V administrační části bude možné editovat všechny v aplikaci používané údaje
   - 3.2. Story
     - 3.2.1. Editace seznamu (přidání a mazání storů)
     - 3.2.2. Editace jednotlivých storů
       - 3.2.2.1. O každém bude aplikace uchovávat jeho název a číslo
   - 3.3. Departmenty a subdepratmenty
     - 3.3.1. Editace seznamu departmentů
     - 3.3.2. Editace departmentu
       - 3.3.2.1. O každém departmentu bude aplikace uchovávat jeho název
       - 3.3.2.2. Každý department má subdepartmenty
       - 3.3.2.3. Editace seznamu subdepratmentů
       - 3.3.2.4. Editace subdepartmentu
         - 3.3.2.4.1. O každém subdepartmentu bude aplikace uchovávat jeho kód, název (český) a anglický název
   - 3.4. Kategorie otázek a otázky
     - 3.4.1. Seznam kategorií otázek
       - 3.4.1.1. Editace seznamu otázek (přidávání a mazání kategorií)
     - 3.4.2. Editace kategorie – o každé kategorii bude aplikace uchovávat její název
     - 3.4.3. Editace otázek dané kategorie
       - 3.4.3.1. Seznam otázek s možností přidat otázku do kategorie
       - 3.4.3.2. Možnost výběru otázky pro editaci jejích atributů (viz Obrázek 6: Výběr otázek a detail otázky)
         - 3.4.3.2.1. Váha otázky
         - 3.4.3.2.2. Text otázky
         - 3.4.3.2.3. Kategorie otázky
         - 3.4.3.2.4. Detail otázky
           - 3.4.3.2.4.1. Detail otázky bude jeden z následujících typů: žádný, tabulka expirace, tabulka expirace s šarží, tabulka čerstvosti a tabulka čerstvosti s šarží
   - 3.5. Dotazníky
     - 3.5.1. Seznam dotazníků s možností přidání nového dotazníku
     - 3.5.2. O každém dotazníku bude aplikace uchovávat jeho typ a název
     - 3.5.3. Do dotazníku bude možné přidat checklisty dle existujících subdepartmentů
     - 3.5.4. Checklist bude obsahovat všechny dostupné kategorie
     - 3.5.5. Do jednotlivých kategorií checklistu bude možné přidat otázky výběrem ze všech otázek dané kategorie (viz Obrázek 6: Výběr otázek a detail otázky)
   - 3.6. Hotové dotazníky
     - 3.6.1. Seznam dotazníků kde bude možné editovat atribut dotazníku „dokončeno“, který určí, zda se dotazník zobrazí ve výběru dotazníků v uživatelské části (dokončeno = pravda => nezobrazí se)

3.7. Uživatelé

3.7.1. Seznam uživatelů s možností vybrat a editovat jednotlivé uživatele

3.7.2. O uživateli bude aplikace uchovávat jeho Jméno, Příjmení, uživatelské jméno, heslo a roli v aplikaci

3.7.3. Na sílu hesla nejsou definované žádné nároky

Obrázek 6: Výběr otázek a detail otázky

4 Analýza

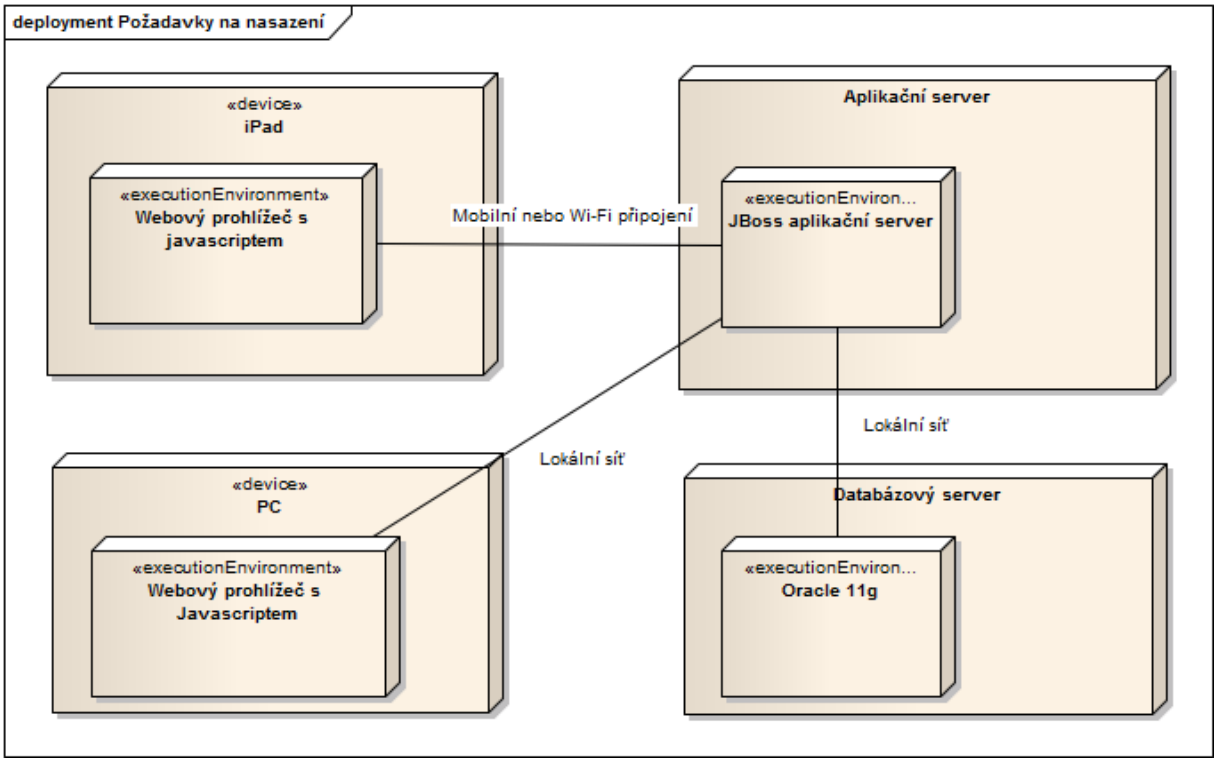
V analytické části analyzuji požadavky na architekturu platformy, dále definuji aktory a případy užití a vytvořím analytické třídy. Budu řešit případné konflikty požadavků, zejména bude potřeba zpozornět v blízkosti hranice mezi jednotlivými částmi realizované aplikace. Výstupem bude konzistentní seznam aktorů a akcí a ověření realizovatelnosti nefunkčních požadavků. Odhalené nedostatečně definované požadavky ve spolupráci s uživateli upřesněním.

4.1 Architektura aplikace

Z požadavků zákazníka je zřejmé, že jde o klient-server aplikaci.

Klientská část bude obsahovat pouze tenkého webového klienta a poběží na zákazníkem definovaném hardware (iPad 3), dle požadavků je bude klientská strana umožňovat jen zobrazování dat se základním ověření vstupů od uživatele, z náročnějších úkolů je zde pouze nutnost zmenšovat rozlišení fotografií už na klientské straně (z důvodu úspory dat při přenosu). Zákazník požaduje vytvoření klientské části aplikace pomocí frameworku Vaadin a jeho addonu Touchkit. Framework vaadin je založená na GWT, pro svůj běh tedy vyžaduje prohlížeč s povoleným Javascriptem.

Serverová část bude obsahovat úložiště dat (je požadováno využití Oracle 11g databáze) a veškerou business logiku, tedy zejména logiku pro manipulaci s daty, algoritmy pro vyhodnocování auditů a logiku pro vytváření reportů. Framework vaadin v serverové části využívá pro svůj běh JVM. Serverová část aplikace bude běžet v prostředí aplikačního serveru JBoss 7.1, další požadavky na realizační nástroje zákazník nevznesl.



Obrázek 7: Model nasazení dle požadavků

4.2 Uživatelské role

Uživatelské role jsou dobře definovány již v požadavcích a reflektují existující role v týmu oddělení, pro které je aplikace určena. Uživatelé jsou si vědomi svých rolí a dle nich již definovali požadavky na aplikaci. V aplikaci budou zastoupeny role: Admin, QA, Approval.

4.2.1 Admin

Hlavní úlohou této role je spravovat data. Zejména má tato role následující úkoly

- Spravovat uživatele
- Spravovat departmenty a subdepartmenty
- Spravovat story
- Spravovat kategorie a otázky
- Spravovat dotazníky
- Označovat dotazníky jako dokončené

Jedná se o roli, která plní funkce, které nejsou při běžném provozu příliš potřeba a je proto vhodné vytvořit pro ně zvláštní roli se speciálním okruhem práv.

4.2.2 QA

Role cílového uživatele, který bude s aplikací pracovat prakticky na denní bázi. Úkolem tohoto uživatele je vykonávat audity, což by mu měla naše aplikace usnadnit. Uživatelé v této roli budou používat dotazníkovou část se všemi jejími funkcemi, zejména

- Vyplňování dotazníků včetně
    - zadávání score
    - doplňování poznámek
    - doplňování neshod
    - vytváření a úpravy seznamů výrobků
    - pořizování a vkládání fotografií

4.2.3 Approval

Jedná se o roli, jejíž úkol je kontrolovat správnost a kompletnost vyplnění dotazníků. Po vyplnění a případných úpravách dotazníku uživateli role QA je na řadě uživatel role Approval, který finálně zkontroluje dotazník a jeho vyhodnocení předtím než tento dotazník opustí oddělení (stane se výstupem pro další účastníky auditování – management a pracovníky na storech). Tento uživatel má stejná práva jako role QA, jen s tím rozdílem že se jeho jméno nezapíše na seznam auditorů, když vstoupí do dotazníku.

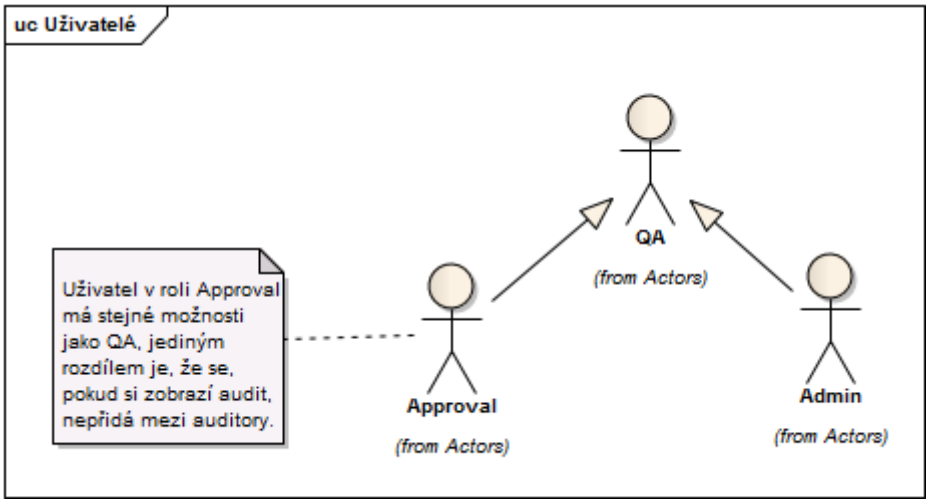
4.2.4 Profil uživatele

Realizovanou aplikaci hodnotím z hlediska náročnosti na znalosti uživatele jako nenáročnou, jednoduchou. Uživatelé, znalí problémové domény, nebudou mít problém aplikaci ovládat ani porozumět jejímu chování. Ve všech rolích budou vystupovat uživatelé, kteří mají již dostatek znalostí na to, aby mohli aplikaci snadno ovládat. Uživatelé jsou rozdělení do následujících rolí: *Admin, QA, Approval*. Uživatelé ve všech uživatelských rolích budou používat stejné rozhraní a budou dle potřeby proškoleni, aby byli schopni aplikaci efektivně využívat.

4.3 Případy užití

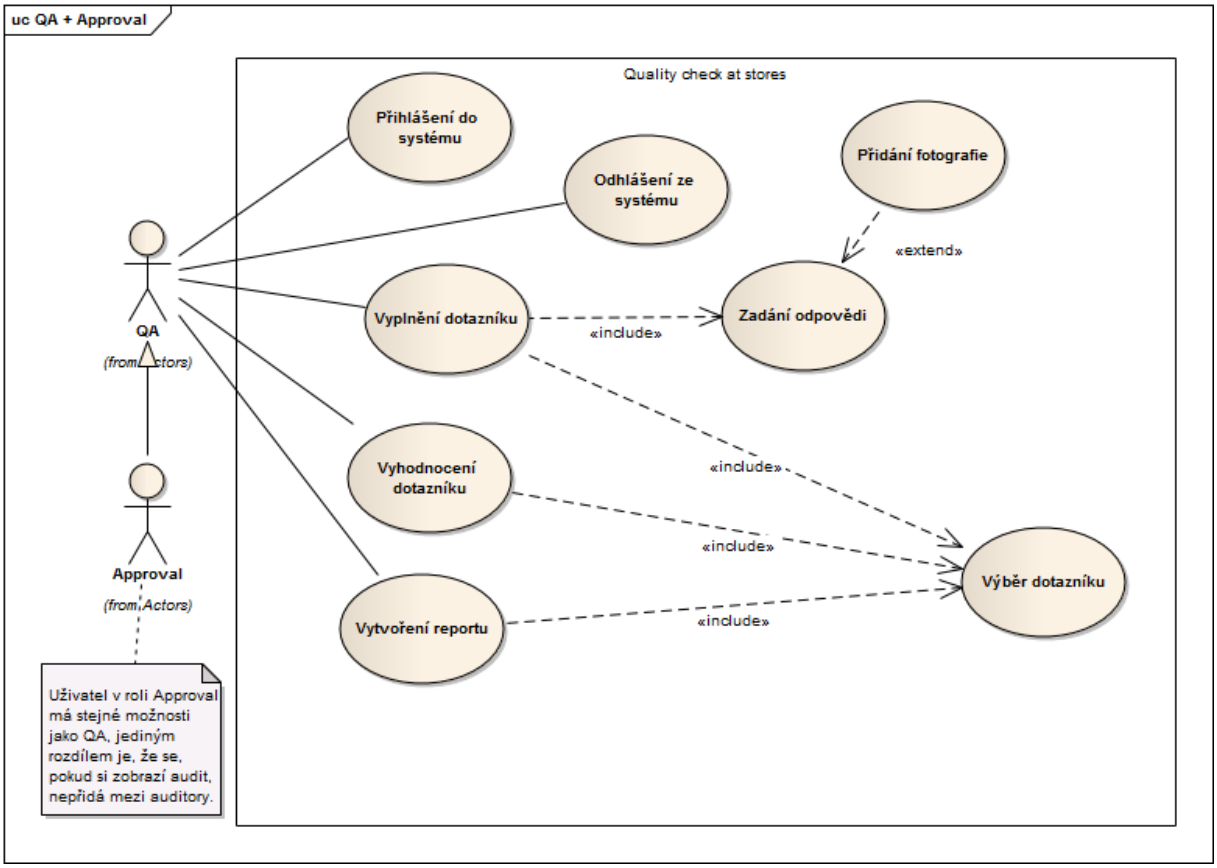
V předchozí kapitole jsem rozdělil funkční požadavky podle jejich příslušnosti k danému aktorovi, nyní pro přehlednost vytvořím diagramy, které zobrazují souvislosti mezi uživateli a jejich funkčními požadavky – případy užití.

První diagram (Obrázek 8: Uživatelské role) zobrazuje vzájemné vztahy uživatelů, „základním“ uživatelem je uživatel v roli QA. Uživatelská role Approval dědí všechny případy užití a jeho jedinou odlišností je že se nezobrazuje v seznamu auditorů, pokud si zobrazí audit. Dalším uživatelem je Admin, který rovněž dědí všechny případy užití od role QA, a navíc je mu umožněno spravovat aplikační data v rozsahu, který je uveden v kapitole Admin.



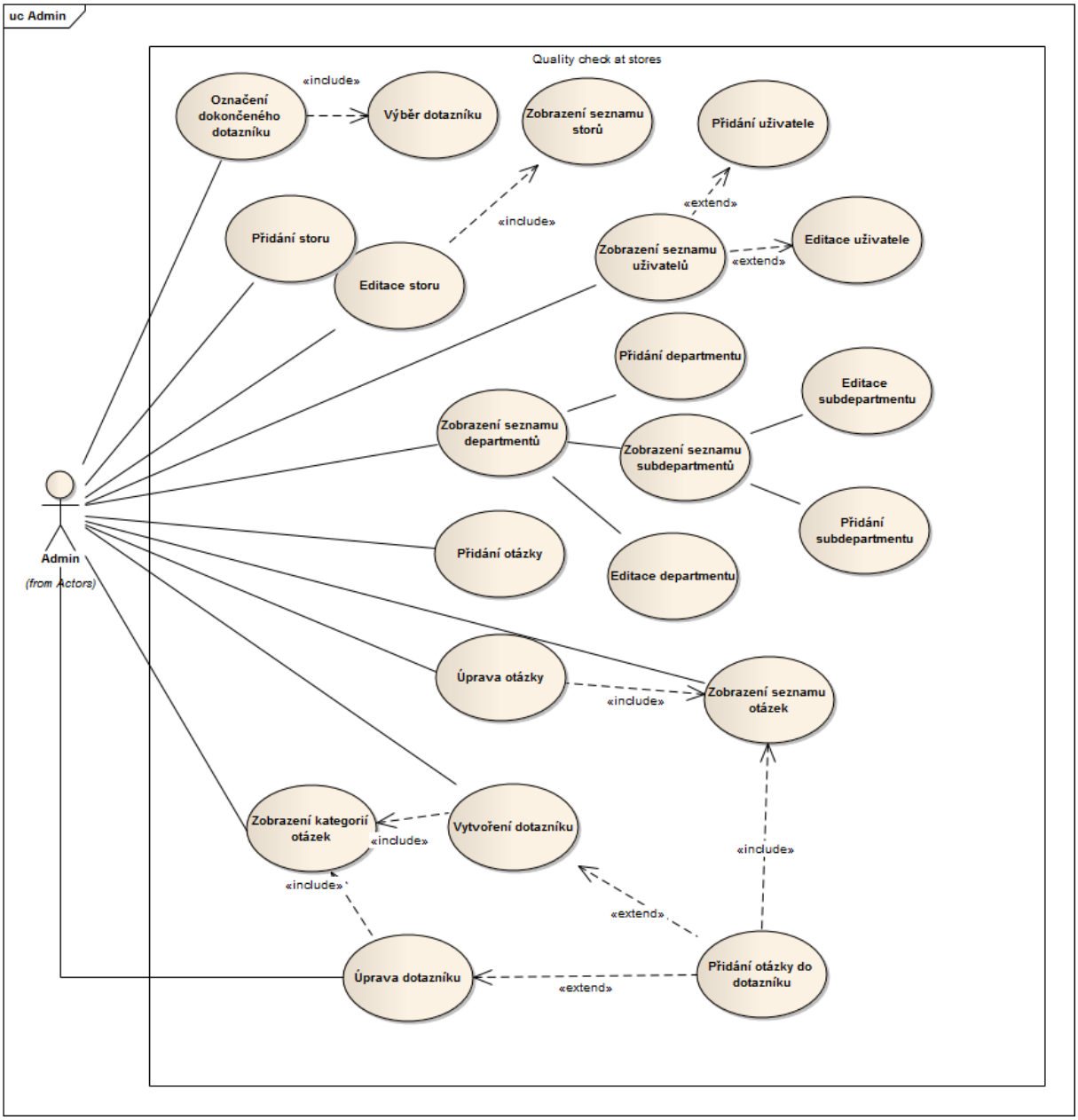
Obrázek 8: Uživatelské role

Na dalším diagramu (Obrázek 9: Případy užití pro roli QA a Approval) jsou zobrazeny případy užití pro uživatelské role QA a Approval.



Obrázek 9: Případy užití pro roli QA a Approval

Diagram případů užití pro roli Admin (Obrázek 10: Případy užití pro roli Admin) zobrazuje možnosti, které v rámci systému má uživatel v roli Admin. Tyto případy užití jsou zaměřeny na veškerou správu dat kromě vyplňování a vyhodnocování dotazníků, které má Admin k dispozici prostřednictvím jejich podědění od role QA.



Obrázek 10: Případy užití pro roli Admin

4.4 Analytické třídy

V této kapitole najdu analytické třídy a vytvořím jejich seznam a definuji jejich závislosti.

4.4.1 Hledání analytických tříd

K hledání analytických tříd slouží nejčastěji popis byznys domény případně seznam požadavků. Oba tyto vstupy mám k dispozici. Analytické třídy jsou objekty budoucí aplikace, jež jsou obrazem skutečných objektů reálného světa. Tyto třídy jsou většinou zřejmé po přečtení popisu byznys domény nebo požadavků. Jednoduše řečeno se jedná o nejčastěji se vyskytující podstatná jména, která mají vazbu na problémovou doménu.

Častým zdrojem analytických tříd je také slovníček pojmů z problémové domény, který si analytik buduje kvůli porozumění s uživateli. To je v našem případě asi nejsnazší cesta jak se k analytickým

třídám dobrat. Slovníček s popisem pojmů je možné nalézt v kapitole Základní pojmy. Zde zmiňované pojmy poslouží jako dobrý základ analytických tříd.

4.4.2 Analytické třídy ze slovníku pojmů

Základní analytické třídy jsou vidět v našem případě přímo ze slovníku pojmů.

- Auditor
- Store
- Department
- Subdepartment
- Typ auditu
- Audit
- Checklist
- Otázka

4.4.3 Analytické třídy z popisu problémové domény a požadavků

Další analytické třídy získané z popisu problémové domény a požadavků

- Uživatel, je spojen s třídou Auditor, kterou bude nahrazovat
- Kategorie otázek, slouží k členění otázek

4.4.4 Další analytické třídy

Pro komplexní řešení problému jsou potřeba další třídy, jejichž existence není z požadavků a slovníku přímo zřejmá. Jedná se o třídy vztahující se k vyplněným auditům. Z požadavků je zřejmé že je potřeba dotazník vyplňovat opakovaně, proto je potřeba oddělit třídy pro dotazník jako takový skládající se z checklistů a otázek od vyplnění dotazníku, které se skládá z výsledků dotazníku, listů s odpověďmi, které odpovídají vyplnění checklistů a odpovědi na otázky i s detaily (seznam výrobků, fotografie).

Pro tento účel tedy zavedu i následující třídy:

- Výsledky auditu
- List odpovědí
- Odpověď
- Výrobek (položka seznamu výrobků)
- Fotografie

4.4.5 Vztahy a atributy analytických tříd

Vztahy analytických tříd popisují souvislosti mezi jednotlivými třídami, jejich hierarchii a násobnost vztahů. Prostý výčet tříd je bez pochopení vztahů mezi nimi nepoužitelný pro další práci. Proto v této kapitole popíši jednotlivé vztahy mezi třídami, případně jejich skupinami.

Analytické třídy obsahují dvě základní hierarchie a pak řadu tříd pro kategorizaci a doplňující informace.

4.4.5.1 Hierarchie Audit-Checklist-Otázka

Tato hierarchie je základem aplikace, audit obsahuje N checklistů (1:N) a checklist obsahuje N otázek (1:N). Jedná se o soubor otázek, které jsou tříděné dle checklistu (oddělení). Audit má krom pole

checklistů ještě tyto atributy: název a typ auditu. Checklist má krom pole otázek ještě atribut subdepartment, ke kterému se vztahují dané otázky. Otázka má atributy text, váha a typ detailu.

4.4.5.2 Hierarchie Výsledky auditu-List odpovědí-Odpověď

Tato hierarchie je sadou odpovědí a výsledků auditu, jednotlivé položky mají vazbu na položky z předchozí hierarchie, ke kterému jsou výsledkem/odpovědí. Výsledky auditů mají N listů odpovědí (1:N) a List odpovědí má N odpovědí (1:N).

Výsledky auditu mají následující atributy: název, audit, ke kterému je výsledkem, datum uskutečnění auditu, datum předchozího auditu, store kde se audit konal, slovní popis výsledků auditu, seznam auditorů, kontrolora auditu, požadované score pro daný rok, počat získaných bodů, maximální možný počet bodů a seznam listů odpovědí.

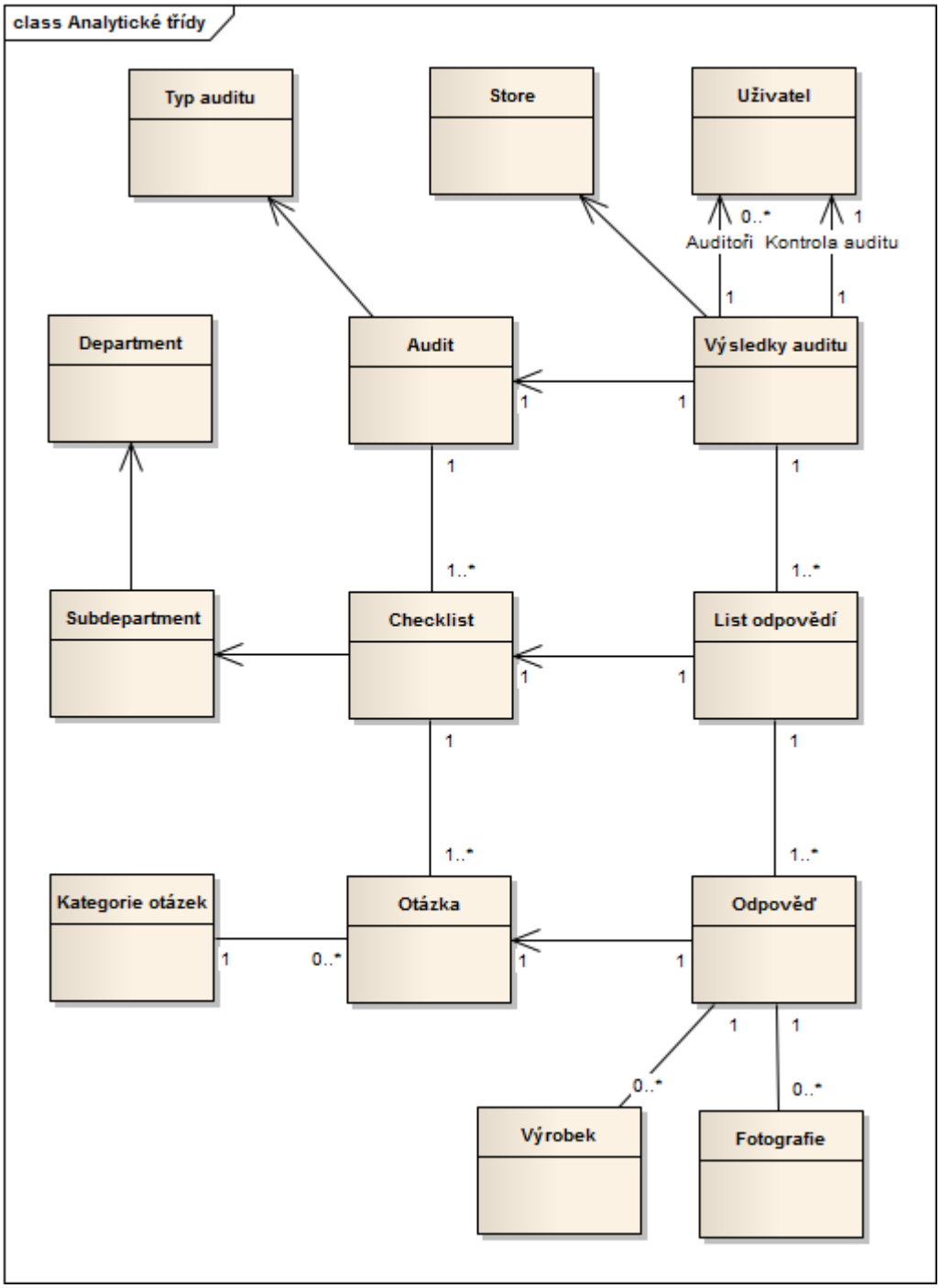
List odpovědí má následující atributy: Checklist ke kterému je odpovědí, seznam odpovědí, výsledky auditu kterého je součástí, maximum možných bodů které lze získat a bodový zisk.

Odpověď má následující atributy: otázku, ke které je odpovědí, list odpovědí do kterého patří, maximum bodů, které bylo možné získat, score, bodový zisk, poznámku, neshodu, seznam výrobků a seznam fotografií.

4.4.5.3 Kategorizace

Pro kategorizaci a doplnění informací slouží následující třídy:

- Typ auditu pro kategorizaci auditů (1:N)
- Store pro zadání místa kde byly výsledky auditu pořízeny (1:N)
- Subdepartment pro kategorizaci checklistů (1:N)
- Department pro kategorizaci subdepartmentů (1:N)
- Kategorie otázek pro kategorizaci otázek (1:N)
- Uživatele pro určení auditorů, kteří audit vykonali (M:N) a uživatele, který provedl kontrolu auditu (1:N)
- Výrobek pro určení výrobku v seznamu výrobků u odpovědi (N:1)
- Fotografie pro uložení fotografie v seznamu fotografií u odpovědi (N:1)



Obrázek 11: Analytické třídy

5 Návrh

V této kapitole proberu jednotlivé části aplikace z pohledu platformně nezávislého návrhu, budu se věnovat zejména jednotnému uživatelskému rozhraní a také rozdělení odpovědností mezi jednotlivé třídy mimo doménový model.

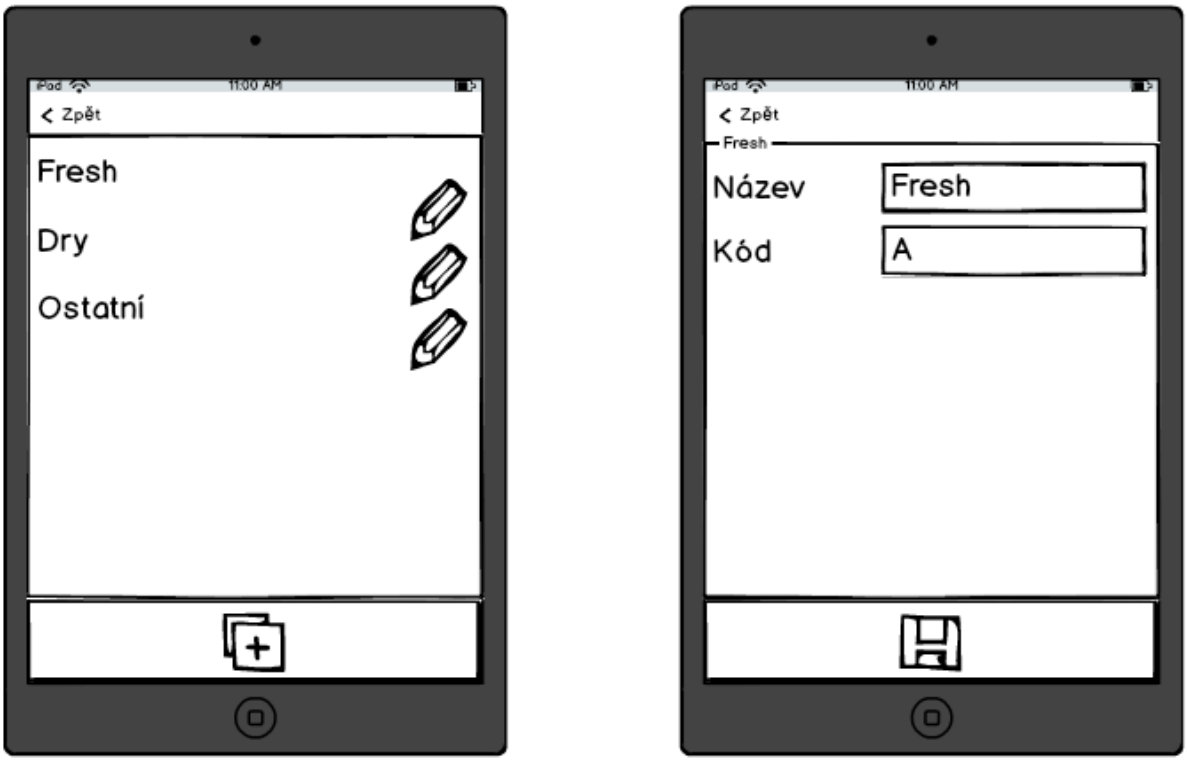
5.1 Uživatelské rozhraní

Pro jednoduchost a grafickou podobnost obrazovek aplikace jsem definoval dvě základní obrazovky: Seznam a Detail. V této kapitole popíši jejich význam, použití a rozložení. Obě obrazovky vychází z klasického zobrazování, tak jak je použito na mobilních zařízeních omezených menšími displeji.

5.1.1 Obrazovka Seznam

Obrazovka Seznam je použita pro zobrazení více stejných položek, tento seznam slouží jako rozcestník k detailům jednotlivých položek. Na této obrazovce je zobrazeno jen několik detailů (atributů) dané položky, které mají za úkol položku jednoznačně identifikovat. U jednotlivých položek je volitelně zobrazeno tlačítko umožňující jejich editaci. Při kliku na položku je možné přejít v hierarchii na její podpoložky, pokud existují. Dalším prvkem je tlačítko „+“ umožňující přidání nové položky stejného typu do seznamu. Po stisku tohoto tlačítka je uživateli zobrazena obrazovka Detail, kde je potřeba vyplnit potřebné atributy položky.

Například departmenty je možné editovat v administraci tak, že uživatel klikne na položku „department“. Zobrazí se obrazovka se seznamem jednotlivých departmentů, což je ukázka obrazovky Seznam (Obrázek 12: Obrazovka Seznam (vlevo) a obrazovka Detail (vpravo)). Z této obrazovky má uživatel možnost klikem na daný department přejít na jeho subdepartmenty, tedy níže v hierarchii objektů, nebo klikem na ikonu editace (tužka) zobrazit detail daného departmentu, což je příklad obrazovky detail. Další možností je přidat položku (nový department).



Obrázek 12: Obrazovka Seznam (vlevo) a obrazovka Detail (vpravo)

5.1.2 Obrazovka Detail

Obrazovka Detail (Obrázek 12: Obrazovka Seznam (vlevo) a obrazovka Detail (vpravo)) umožňuje editaci atributů vybrané položky seznamu. Realizuje dva scénáře: editaci a přidání položky. Oba scénáře začínají na Obrazovce seznam.

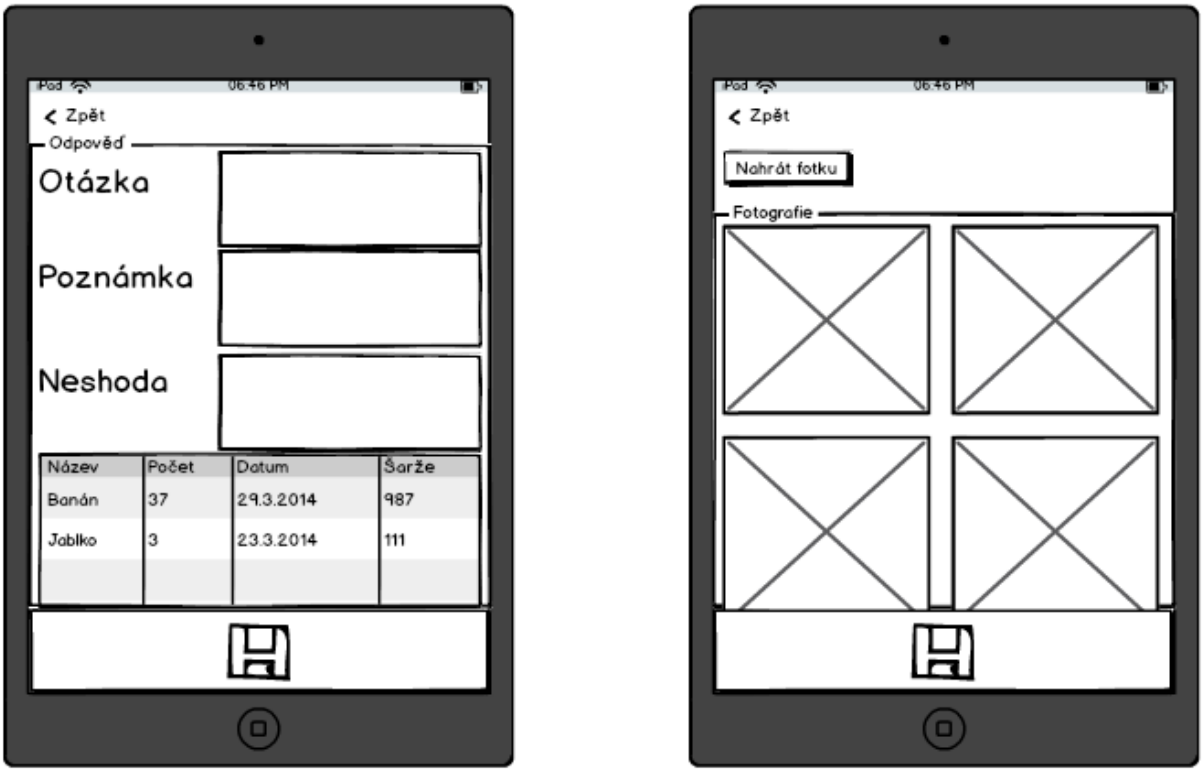
Pro přidání položky je potřeba stisknout tlačítko „+“, následně se zobrazí prázdný formulář, do kterého je potřeba vyplnit jednotlivé atributy a nakonec položku uložit (tlačítko disketa), následně je uživatel přesměrován na obrazovku Seznam.

Při editaci položky je potřeba u položky, kterou chce uživatel editovat, stisknout tlačítko „tužka“, to zobrazí formulář, kde jsou předvyplněné atributy dané položky. Ty pak uživatel edituje, když je hotov potvrdí editaci uložením (tlačítko „disketa“), následně je uživatel přesměrován na obrazovku Seznam.

Oba scénáře je možné ukončit bez uložení klikem na tlačítko „zpět“ které zruší editaci nebo přidání položky a přesměruje uživatele na obrazovku Seznam.

5.1.3 Obrazovky odpovědi

Aplikace využívá další obrazovky, které jsou specificky navržené pro daný úkol. Nejpoužívanějšími takovými obrazovkami jsou obrazovky pro zadání upřesnění odpovědi. První z nich umožňuje zadat poznámku a neshodu, případně seznam výrobků, pokud je požadován (vlevo na Obrázek 13: Detail odpovědi), na obrazovce se zobrazuje znovu i otázka ke které je detail doplňován. Druhá obrazovka umožňuje nahrát fotografie a zároveň zobrazuje galerii fotografií již přidaných k dané odpovědi (vpravo na Obrázek 13: Detail odpovědi).



Obrázek 13: Detail odpovědi

5.2 Servisní třídy

V rámci splnění požadavků aplikace je potřeba rozdělit její jednotlivé funkce do správně zapouzdřených jednotek, takové rozdělení následně usnadní údržbu aplikace a zpřehlední její fungování. Kromě níže vyjmenovaných kategorií mají své servisní třídy i entity, se kterými se komplexněji pracuje, například entita pro výsledky auditu.

5.2.1 Scoring

Scoring je samostatná třída určená pro vyhodnocování, její realizaci jsem oddělil záměrně, protože očekávám, že strategie vyhodnocování se bude měnit častěji než zbytek aplikace. Chtěl jsem tedy, aby v případě změny vyhodnocování bylo možné nalézt vše potřebné v jedné třídě. V této třídě jsou i výpočty, které jednotně používají zaokrouhlování a počet desetinných míst. Zároveň se tu vyhodnocuje, zda je již test považován za vyplněný nebo ne. Vyhodnocení testu a kontrola vyplnění se děje hierarchicky na úrovni otázek, pak na úrovni oddělení/checklistů a nakonec na úrovni celého auditu.

5.2.2 Reporty

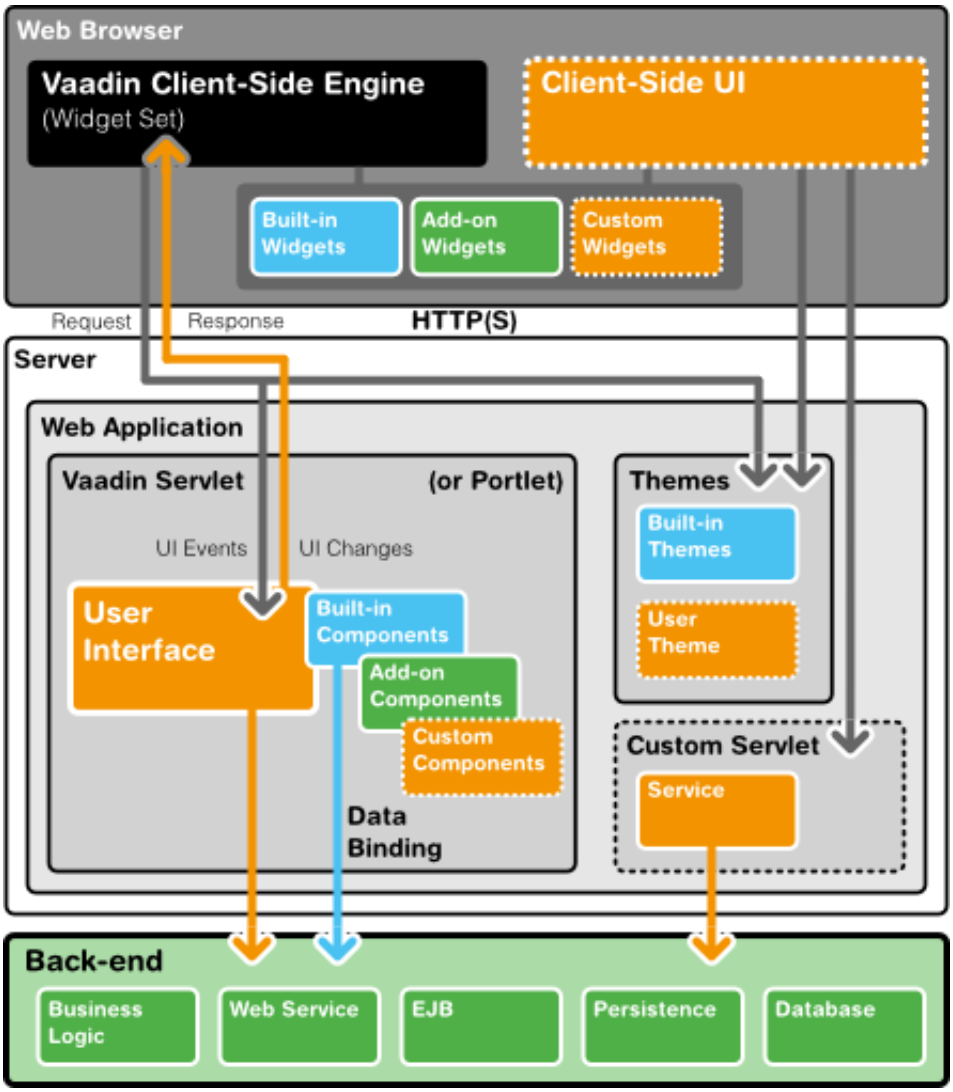
Pro reportování jsem použil samostatný balíček, který obsahuje odděleně třídy pro report do PDF a do XLSX, dále obsahuje všechny pomocné třídy, které jsou při vytváření reportu používány, tento samostatný balík tříd je možné vyjmout a kompletně nahradit jiným reportem, případně v servisních třídách pro jednotlivé druhy reportu provést jejich úpravu, nebo je nahradit na této úrovni.

5.3 Komunikace a vaadin

Komunikace mezi klientskými zařízeními a serverem bude zajišťována komponentami frameworku Vaadin, až na výjimky tedy nebude tedy potřeba řešit komunikaci programátorský. Framework pracuje tak, že k Javovským komponentám vytvoří jejich JavaScriptový obraz a komunikaci mezi nimi již řeší sám, celá aplikace je tedy jedna stránka, jejíž obrazovky jsou překreslovány JavaScriptem. Výjimku pro tento model komunikace tvoří v aplikaci tři místa: přihlašovací obrazovka, grafická data (CSS a obrázky) a komponenta pro upload obrázků.

5.3.1 Komponenty a widgety

Standartní aplikace založená na vaadinu má komunikaci mezi klientem a serverem již skrytou před programátorem, programátor pouze definuje, jaké komponenty se mají použít a jejich rozložení a akce (event, listener) a vše ostatní už zařídí framework. Vše se tváří jako by komunikace vůbec neprobíhala a vše se odehrávalo na serveru. Komunikace zajišťovaná frameworkem vaadin probíhá podle schématu na obrázku Obrázek 14: Architektura vaadin frameworku. Na serverové straně existuje model uživatelského rozhraní, k němu jsou připojeny listenery. Uživatelské rozhraní samo je pak na klientské straně, a jednotlivé eventy jsou přenášeny na server, kde jsou zachycovány patřičnými listenery stejně jako by se vše dělo na serveru. Pro využití tohoto chování stačí použít vestavěné komponenty frameworku (na obrázku „bulit-in widget“ a „component“, modře), použít pluginy z ekosystému vaadin (add-on widget a component, zeleně) nebo si vytvořit vlastní, jejichž chování bude podobné (custom widget a component, oranžově). Serverová část aplikace v servletu je tedy odstíněna od komunikace s klientem a z pohledu programátora servletu vše probíhá právě v tomto servletu. Další komunikace do aplikace se pak děje pomocí servisních tříd.



Obrázek 14: Architektura vaadin frameworku, převzato z [2]

5.3.2 Přihlašovací obrazovka

Z důvodu co nejjednodušší implementace modulu přihlašování jsem zvolil pro přihlášení HTML stránku s formulářem, který má prvky pojmenované tak jak to vyžaduje modul Spring Security. Díky tomu může být přihlášení celé realizováno a zpracováno přímo Springem a není potřeba integrovat Spring a vaadin a realizovat přihlášení pomocí vaadin formuláře. Takto předejdu problémům s injektováním dat do dané obrazovky. Díky tomu že celé rozhraní je realizované pomocí jedné stránky ovládané JavaScriptem je tato obrazovka umístěna ve složce „. /VAADIN/“, která je přístupná bez přihlášení. Takto se k ní tedy dostane i nepřihlášený uživatel a nedochází k renderování a zpracování pomocí Vaadinu. Obrazovka je zpracována Springem a následně je uživatel přesměrován na hlavní stránku aplikace.

5.3.3 Grafická data

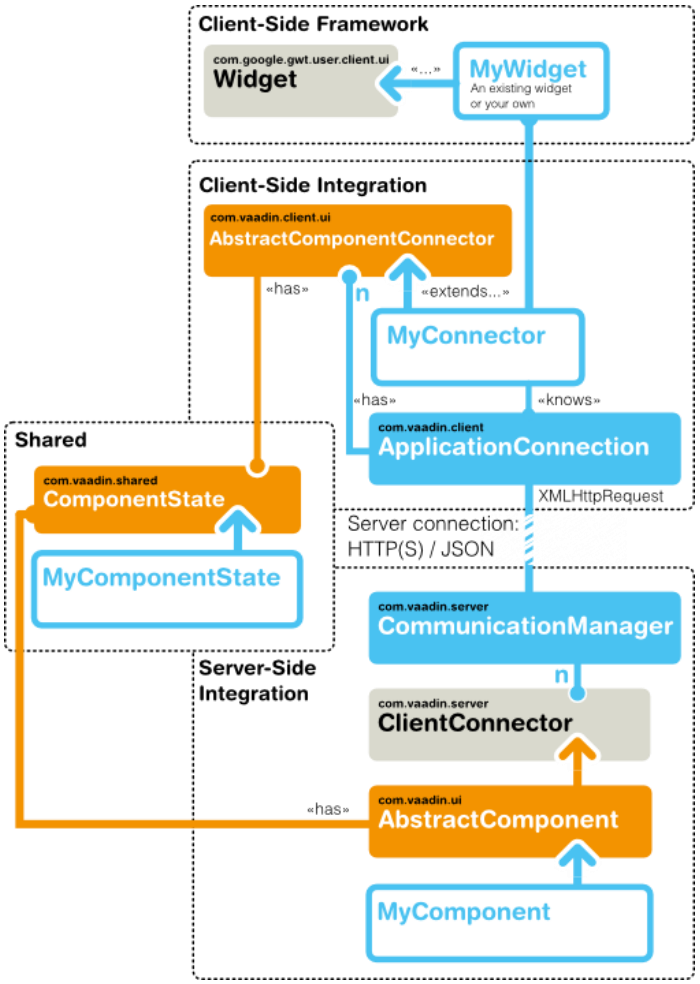
Grafická data, tedy zejména CSS a obrázky, jsou uloženy ve složce „. /VAADIN/“. Z této složky je možné je stahovat bez kontroly přihlášení a jsou tedy dostupná všem uživatelům, zároveň se tím vyhnu zatěžování serveru – jde totiž o data, která není potřeba zabezpečit.

5.3.4 Komponenta pro upload obrázků

V rámci realizace aplikace je potřeba vytvořit vlastní komponentu pro upload obrázků. Vaadin již obsahuje komponentu pro upload, dokonce je možné ji upravit tak, aby přijímala jen obrázky. Jediná odlišnost, kterou disponuje nově vytvářená komponenta je zmenšení obrázku před odesláním na server. Tento požadavek vzešel z analýzy rychlosti připojení a nemožnosti měnit rozlišení fotoaparátu klientského zařízení. Fotografie v původním rozlišení mají velikost cca 1MB, a svojí kvalitou daleko překonávají požadované rozlišení (1024x768). Je tedy potřeba obrázek před odesláním zmenšit tak aby zbytečně nezatěžoval přenosovou linku.

Jak jsem popsal v architektuře Vaadinu (kapitola Komponenty a widgety) programátor nemá, při použití základních komponent, možnost zpracovávat data na klientu, to se týká i zpracování obrázků. Standartní komponenta Upload by při použití listeneru a následném zpracování obrázku nefungovala správně – obrázek by se nejprve přenesl na server a až pak by byl zpracován patřičným listenerem kde by došlo ke zmenšení rozlišení. Obrázek by se tedy přenesl celý a zmenšil by se až na serveru. Z tohoto důvodu je potřeba vyvinout vlastní komponentu, která umožní zpracování obrázku na klientské straně.

Nová komponenta se bude na klientu i serveru chovat jako komponenta a widget Upload. Pouze v klientské části přibude část zajišťující zmenšení obrázku. Komunikace bude také realizována jiným způsobem – přenos obrázku jako data kódovaná v Base64 kódování. Pro vytvoření vlastní komponenty je potřeba vytvořit několik tříd, které dědí z již existujících tříd přítomných ve vaadin frameworku a určených právě pro snadné vytváření nových komponent (Obrázek 15: Vlastní komponenta s widgetem). Vlastní komponentu je potřeba umístit do zvláštních klientských balíčků (package) a použité knihovny zaregistrovat do widgetsetu.



Obrázek 15: Vlastní komponenta s widgetem, převzato z [2]

6 Implementace

V této kapitole popíši výběr technologií pro realizaci, nasazení, nastavení a spouštění. Provedu výběr a obhajobu knihoven pro realizaci. Předvedu způsob nasazení jednotlivých komponent a jejich konfigurace, doplním zajímavé detaily z implementace. A popíšu problémy vzniklé při implementaci a jejich řešení.

6.1 Framework Spring

Pro realizaci aplikace jsem na základě vlastních zkušeností vybral framework Spring. Jedná se v současné době o nejpoužívanější a nejrozšířenější řešení pro realizaci nejen webových aplikací.

Jádro frameworku umožňuje zejména vkládání závislostí do jednotlivých tříd. Dle návrhového vzoru Inversion of Control, je za vytváření tříd a navazování jejich závislostí zodpovědný samotný kontejner. Ve třídě, kde je jiná třída používána, již tedy není potřeba řešit její vytvoření a inicializaci jejích závislostí, to vše je dodáno kontejnerem na základě konfiguračního souboru, případně automatického rozpoznávání závislostí dle jejich příslušnosti k určité třídě nebo interface.

6.1.1 Inversion of control a Dependency injection

Jedná se o dva, ve Springu úzce spjaté, návrhové vzory, které společně dovolují oddělit jednotlivé znovupoužitelné kusy kódu (ve Springu třídy) a vkládat je na určitá, označená místa kde jsou potřeba a to bez znalosti procesu jejich vytváření a konkrétních atributů. Framework funguje zjednodušeně tak, že dle anotací a konfigurace vytvoří (control) seznam komponent, které má k dispozici a následně je dodává (injektuje) na místa kde jsou potřeba, tato místa označí uživatel, případně jsou sama identifikována frameworkem. Framework pak ze seznamu tříd, které má k dispozici vybere ty s požadovaným rozhraním a provede jejich injektování na daná místa. Tím je snížena vzájemná závislost tříd (Loose coupling) a zvětšena znuvupoužitelnost komponent.

Například při zabezpečení vyvíjené aplikace byl definován *passwordEncoder* (řádek 36 a 37 na obrázku Obrázek 16: Definice beans), ten je frameworkem vytvořen a vložen do seznamu známých objektů (beans). Následně je pomocí konfigurace injektován do *AuthenticationProvideru* (řádek 41 na obrázku Obrázek 16: Definice beans).

```
33      <beans:bean id="UserDetailsServiceImpl"
34                  class="cz.cvut.fel.zdrhajan.test1.UserDetailsServiceImpl">
35      </beans:bean>
36      <beans:bean id="passwordEncoder"
37                  class='org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder'/>
38
39      <authentication-manager alias="authenticationManager">
40          <authentication-provider user-service-ref="UserDetailsServiceImpl">
41              <password-encoder ref="passwordEncoder" />
42          </authentication-provider>
43      </authentication-manager>
```

Obrázek 16: Definice beans

Tentýž *passwordEncoder* je pak potřeba i ve třídě *UserDetailsServiceImpl* kam je potřeba ho injektovat a používat ho tak aby bylo zajištěno stejné kódování hesel v obou třídách. Injektování je v tomto případě dosaženo použitím anotace @Autowired (řádek 27 na obrázku Obrázek 17: UserDetailsServiceImpl). Framework objeví anotaci a dle typu atributu (*PasswordEncoder*) nalezne vhodného kandidáta na injektování. V rámci naší aplikace je definována jen jedna *bean* s implementující požadované rozhraní, proto je automaticky injektována právě ta. Pokud by bylo

bean implementujících toto rozhraní více bylo by potřeba frameworku pomoci s určením, kterou z nich použít toto je možné provést pomocí anotace @Qualifier("name") kde by slovo „name“ bylo nahrazeno jménem dané *bean* v našem případě „*passwordEncoder*“.

```
22    public class UserDetailsServiceImpl implements UserDetailsService {
23
24        @Autowired
25        private UserRepository userRepository;
26
27        @Autowired
28        private PasswordEncoder passwordEncoder;
```

Obrázek 17: UserDetailsServiceImpl

6.1.2 Konfigurace Springu

Konfiguraci kontejneru je možné provést v současné verzi Springu dvojím způsobem a to buď pomocí XML souboru, nebo pomocí anotací, případně kombinací obou způsobů. Pro tuto práci jsem si vybral kombinaci obou způsobů, a to zejména kvůli novosti konfigurace pomocí anotací a tedy malé komunitě uživatelů, kteří tento způsob používají. Konfiguraci je tedy možné rozdělit na dvě části. Pomocí XML konfigurace jsem konfiguroval modul Spring Security a základní část frameworku zodpovědnou za vytváření a správu servletů. Pomocí anotací jsem pak konfiguroval část zodpovědnou za komunikaci s databází.

6.1.3 Spring Security

Jedná se modul umožňující zabezpečit aplikaci (nejen webovou). Základem je přidání následujícího nastavení do konfiguračního souboru *web.xml*. Přidání tohoto filtru (pozor, filtr musí být první v řadě filtrů, aby zabezpečení fungovalo správně) způsobí, že každý příchozí požadavek bude kontrolován a bude u něj ověřeno, jaký zdroj se pokouší získat a zda k tomu má potřebná práva.

```
38    <filter>
39        <filter-name>springSecurityFilterChain</filter-name>
40        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
41    </filter>
42    <filter-mapping>
43        <filter-name>springSecurityFilterChain</filter-name>
44        <url-pattern>/*</url-pattern>
45    </filter-mapping>
```

Obrázek 18: Security filter

Zdroje a práva jsou nastavena v samostatném konfiguračním souboru *security.xml*, ten definuje informace o uživatelích (jménech, heslech a rolích), způsob přihlášení, zdroje a přístup k nim.

V realizované aplikaci budou prakticky jen dvě stránky, jedna bude sloužit k přihlášení do aplikace a na druhé bude celý zbytek JavaScriptové aplikace. Z dalších zdrojů jsou zde jen styly a obrázky umístěné v podsložkách složky *vaadin* a ikona favicon.ico umístěná v rootu webu, definice těchto zdrojů jako nezabezpečených je na Obrázek 19:Spring security na řádcích 11 a 12. Tyto zdroje jsou přístupné bez zabezpečení, vše ostatní je zabezpečeno.

Definice zabezpečení a přihlašování je pak na řádcích 14 až 31. Postupně proberu jednotlivé řádky a vysvětlím, o jaké nastavení jde. Na řádku 15 je definice přístupu ke všem zdrojům jen pro přihlášené uživatele v definovaných rolích. Z tohoto omezení jsou vyjmuty zdroje zmíněné v předchozím odstavci. Na řádcích 16 až 20 je definice chování v případě použití session, zejména pak přesměrování v případě její expirace (řádek 16) a množství session které může jeden uživatelský účet zároveň mít (řádek 19). Řádek 21 definuje chování odhlášení, pro odhlášení je potřeba přesměrovat uživatele na stránku /logout odkud při úspěchu odhlášení přejde na stránku /login.

Na řádcích 22 až 24 je definice pomůcky pro automatické opakované přihlášení „remember me“ (zapamatuj si mě). Tato funkce zajistí automatické přihlášení pomocí uložení cookie na straně klienta, ten je pak automaticky přihlášen, pokud se pokusí přistoupit na zabezpečenou stránku, například po zavření prohlížeče. Toto se ukázalo při testech jako důležitá funkce, vzhledem ke slabému signálu ve storech a přecházení mezi jednotlivými vysílači bylo spojení občas přerušeno a uživatel byl po novém připojení nucen se znovu ručně přihlásit. Pomocí této funkce je možné uživatele přihlásit automaticky, pokud se dotáže na zabezpečenou stránku v definovaném časovém intervalu (1den = 86400s).

Na řádcích 25 až 30 je definice přihlašování uživatelů. Uživatelé se přihlašují pomocí HTML formuláře */VAADIN/login.html*. Zpracování formuláře provádí přímo modul Spring security. To je možné díky dohodnutému označení polí (name) a adrese, na kterou je formulář odeslán. Formulář musí být odeslán na adresu definovanou na řádku 21 (j_spring_security_check) a jednotlivá pole musí být označena dohodnutými řetězci v atributu name (viz následující seznam).

- Uživatelské jméno - j_username
- Heslo - j_password
- Checkbox pro funkci remember me - _spring_security_remember_me

Pokud je formulář takto nastaven, je automaticky zpracován modulem Spring security, konkrétně instancí třídy *AuthenticationManager*. Pro zpracování je potřeba dodat *AuthenticationManageru* zdroj uživatelských přihlašovacích údajů, tedy třídu implementující rozhraní *UserDetailsService*. Takovouto třídou je ve vyvíjené aplikaci třída *UserDetailsServiceImpl*. Ta je do AuthenticationManageru injektována na základě definice na řádku 40. Na řádku 41 je pak definováno injektování PasswordEncoderu.

```
11      <http security="none" pattern="/favicon.ico"/>
12      <http security="none" pattern="/VAADIN/**"/>
13
14      <http auto-config='true'>
15          <intercept-url pattern="/**" access="ROLE_USER, ROLE_QA, ROLE_ADMIN, ROLE_APPROVAL" />
16          <session-management invalid-session-url="/login"
17                              session-fixation-protection="none"
18                              session-authentication-error-url="/accessdenied">
19              <concurrency-control max-sessions="1" expired-url="/login?session=expired" />
20          </session-management>
21          <logout logout-url="/logout" logout-success-url="/login?logout_successful=1" />
22          <remember-me key="qcs_ap"
23                       user-service-ref="UserDetailsServiceImpl"
24                       token-validity-seconds="86400"/>
25          <form-login
26              login-page="/VAADIN/login.html"
27              login-processing-url="/j_spring_security_check"
28              authentication-failure-url="/login_error.html"
29              default-target-url="/"
30              always-use-default-target="true"/>
31      </http>
32
33      <beans:bean id="UserDetailsServiceImpl"
34                  class="cz.cvut.fel.zdrhajan.test1.UserDetailsServiceImpl">
35      </beans:bean>
36      <beans:bean id="passwordEncoder"
37                  class='org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder'/>
38
39      <authentication-manager alias="authenticationManager">
40          <authentication-provider user-service-ref="UserDetailsServiceImpl">
41              <password-encoder ref="passwordEncoder" />
42          </authentication-provider>
43      </authentication-manager>
```

Obrázek 19:Spring security

6.1.3.1 Třída UserDetailsServiceImpl

Funkcí třídy *UserDetailsServiceImpl* je zprostředkovat přihlašovací údaje uživatele na základě uživatelského jména, kterým se uživatel přihlásil. Tyto údaje mohou být uloženy v jakémkoli objektu, který implementuje rozhraní *UserDetails*, pro snadnost implementace jsem toto rozhraní implementoval přímo u entity *User*. Třída *UserDetailsServiceImpl* tedy jen prostřednictvím injektovaného *UserRepository* načítá z databáze uživatele na základě jejich uživatelského jména.

Další funkcí této třídy je poskytnout administrátorské přihlášení pevně v kódu napsanému uživateli. Jedná se o užitečnou možnost servisního přihlášení i v případě kdy nejsou v aplikaci zadáni žádní uživatelé.

Další funkcí této třídy pak je poskytnutí funkce zakódování hesla tak aby bylo možné hesla v aplikaci editovat a pak je ukládat do databáze již v zakódovaném tvaru, tak jak to požaduje AuthenticationManager a také požadavky IT oddělení zákazníka.

6.1.3.2 Třída PasswordEncoder

Toto rozhraní je výchozím rozhraním, které implementují algoritmy vhodné pro kódování hesel pro jejich bezpečné uložení v databázi. Důvod, proč se hesla před uložením do databáze kódují je, že je nutné znečitelnit heslo i pro například administrátory databáze, zároveň je nežádoucí aby se v aplikaci heslo používalo v nekódovaném tvaru. Proto je heslo co nejdříve po přijetí zakódováno a dále se porovnává jen se zakódovanou podobou uloženou v databázi.

Přímo v modulu Spring security je několik implementací takovýchto algoritmů. IT oddělení zákazníka požadovalo algoritmus MD5 a silnější, z tohoto důvodu jsem vybral algoritmus BCrypt (*BCryptPasswordEncoder*). Definice tohoto *PasswordEncoderu* je na řádku 36 a 37 na obrázku Obrázek 19:Spring security. AuthenticationManageru je pak injektován na základě definice na řádku 41 (Obrázek 19:Spring security).

6.1.4 Spring data

Spring data je další z modulů Springu, tento je určený pro práci s daty uloženými v databázi a jejich přenášení z a do databáze, jejich mapování, dotazování oproti nim a pro další podobné úkoly. Konfigurace probíhá pomocí souboru *persistence.xml* nebo pomocí anotování a Javovské konfigurace. Ve vyvíjené aplikaci je využito Java konfigurace a anotací, celé nastavení je soustředěno do třídy *Application.java*, jedná se o konfiguraci *JpaVendorAdapter* bean, *DataSource* bean a *LocalContainerEntityManagerFactoryBean* bean. Dalším nastavením je anotace *@EnableJpaRepositories* s určením package kde jsou umístěny třídy s jednotlivými repository.

Pro přístup k datům a manipulaci s nimi je použito knihovny Hibernate realizující JSR 317 prostřednictvím *HibernateJpaVendorAdapter*. Knihovna využívá pro mapování objektů do databáze anotace, které jsou v nich uvedené, jedná se o označení entitních tříd anotací @Entity a doplnění dalších nepovinných údajů o struktuře a atributech třídy a dalších detailech.

6.1.4.1 Entitní třída

Entitní třída je datová třída reprezentující soubor dat určitého typu. Na úrovni aplikace je reprezentací dat uložených v jedné tabulce databáze, každá instance této třídy pak reprezentuje jeden řádek takové tabulky.

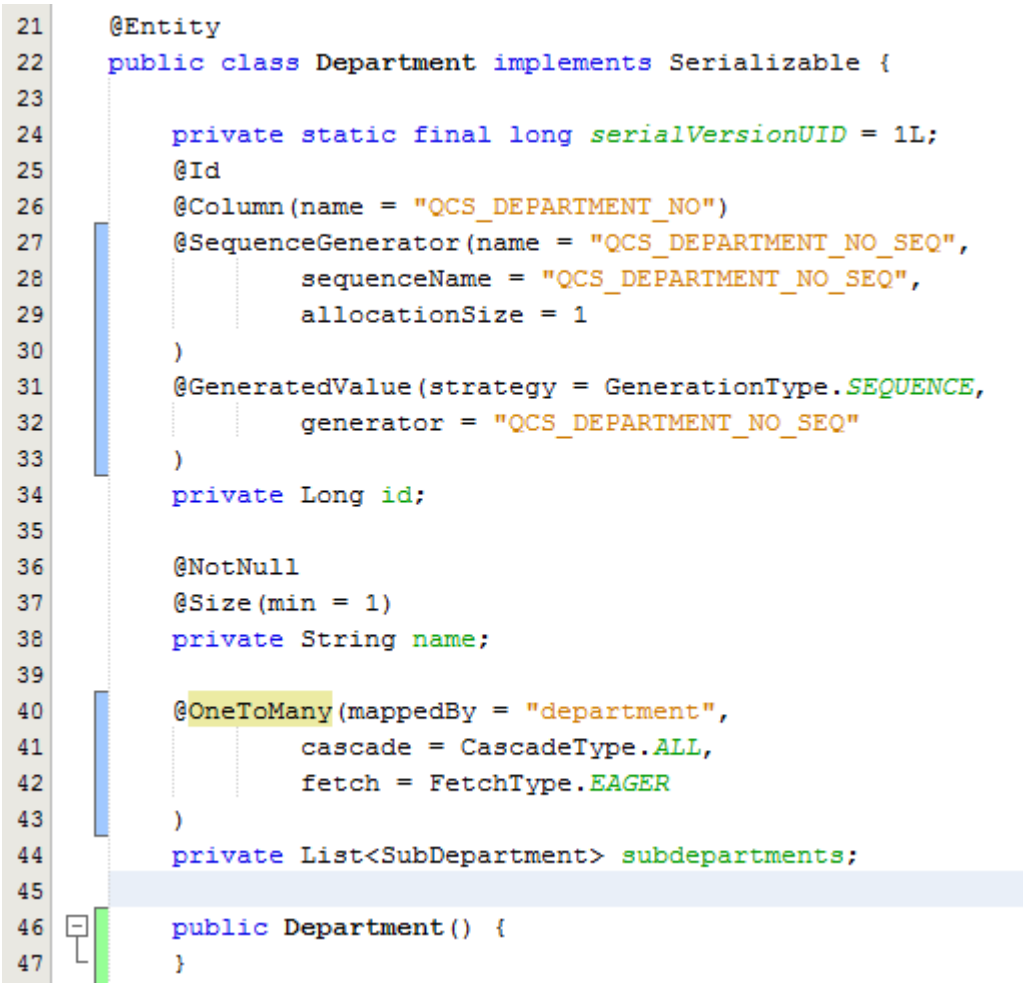
Entitní třídu je nutné anotovat pomocí *@Entity* (řádek 21 na obrázku Obrázek 20: Entita), dále musí mít taková třída bezparametrický konstruktor (řádek 46 na obrázku Obrázek 20: Entita), aby ji mohla persistující knihovna instanciovat a konečně musí mít jednoznačný identifikátor (id) označená anotací *@Id* (řádek 25 na obrázku Obrázek 20: Entita).

Další doplňkové (nepovinné) detaily použité v ukázce jsou @Column, @SequenceGenerator, @ GeneratedValue, @NotNull, @Size a @OneToMany.

Anotace @Column slouží k zadání specifických parametrů sloupce, v tomto případě (řádek 26 na obrázku Obrázek 20: Entita) slouží atribut name k definování názvu sloupce. Název sloupce s identifikátorem se, dle požadavků IT oddělení zákazníka, skládá z názvu tabulky a postfixu „_NO“.

Anotace @SequenceGenerator (řádek 27 na obrázku Obrázek 20: Entita) vytvoří v databázi sekvenci určenou ke generování posloupnosti čísel použitých jako jedinečný identifikátor položek tabulky. Opět je zde požadavek na konkrétní název sekvence, ten je uspokojen definicí jména vytvářené anotace na řádku 28 (sequenceName).

Anotace @GeneratedValue (řádek 31 na obrázku Obrázek 20: Entita) určí sekvenci, která se má použít pro získání hodnot unikátního identifikátoru, jde o sekvenci vytvořenou na řádku 27. Na tuto sekvenci je odkazováno pomocí jejího jména (řádky 32 a 27 na obrázku Obrázek 20: Entita).



```
21  @Entity
22  public class Department implements Serializable {
23
24      private static final long serialVersionUID = 1L;
25      @Id
26      @Column(name = "QCS_DEPARTMENT_NO")
27      @SequenceGenerator(name = "QCS_DEPARTMENT_NO_SEQ",
28              sequenceName = "QCS_DEPARTMENT_NO_SEQ",
29              allocationSize = 1
30      )
31      @GeneratedValue(strategy = GenerationType.SEQUENCE,
32              generator = "QCS_DEPARTMENT_NO_SEQ"
33      )
34      private Long id;
35
36      @NotNull
37      @Size(min = 1)
38      private String name;
39
40      @OneToMany(mappedBy = "department",
41              cascade = CascadeType.ALL,
42              fetch = FetchType.EAGER
43      )
44      private List<SubDepartment> subdepartments;
45
46      public Department() {
47      }
```

Obrázek 20: Entita

Anotace @NotNull a @Size (řádek 36 a 37 na obrázku Obrázek 20: Entita) slouží k validaci ukládaných dat, @NotNull ověří, že je daný atribut různý od hodnoty null. @Size pak ověří délku řetězce, v našem případě je požadován řetězec delší než 1 znak, horní hranice délky není omezena.

Anotace @OneToMany (řádek 40 až 43 na obrázku Obrázek 20: Entita) označuje atribut typu List, který obsahuje instance entitní třídy, která je v relaci 1:N. Takovýto list je možné automaticky naplnit instancemi, které jsou v relaci s touto instancí. Atribut *mappedBy* (řádek 40) pak označuje, jaké vazby má být využito, tedy v jakém sloupci je uložen identifikátor této instance. Atribut *cascade* (řádek 41) určuje chování při změnách, tedy jak se zachovat například v případě odstranění nadřazené entity (v tomto případě dojde i k odstranění objektů obsažených v listu). Atribut *fetch* (řádek 42) určuje chování při načítání entit listu, buď mohou být načteny s načtením tohoto objektu („eager“), nebo až ve chvíli kdy bude k atributu přistoupeno (*„lazy“*).

6.1.4.2 Repository

Repository je návrhový vzor, který má zajistit, že v celé aplikaci bude existovat jedno místo (rozhraní) pomocí, kterého se budou měnit data dané entity v úložišti. Změnou dat je myšlena sada operací CRUD (create - vytvoř, read - přečti, update – aktualizuj/změň, delete -smaž) nad danou entitou. Repository by mělo umožňovat pouze tyto základní operace, případně umožňovat složitější (filtrované) dotazy read.

V rámci implementace bude použito balíku Spring data který umožňuje snadné vytvoření takovýchto repository. Použití repository se dělí na část konfigurační a část programovací. Zatímco v konfigurační části je potřeba repository předat zdroj dat (úložiště) a nasměrovat Framework k daným rozhraním které mají jednotlivá repository, v části programovací je potřeba jednotlivá rozhraní definovat.

6.1.4.2.1 Konfigurační část

Pro správné použití repository v projektu je potřeba je nakonfigurovat v konfiguraci, opět je zde možnost provést konfiguraci pomocí java-konfigurace nebo XML-konfigurace. Konfigurace je v tomto případě snadná a dobře zdokumentovaná, použij tedy pro mě přirozenější java-konfiguraci.

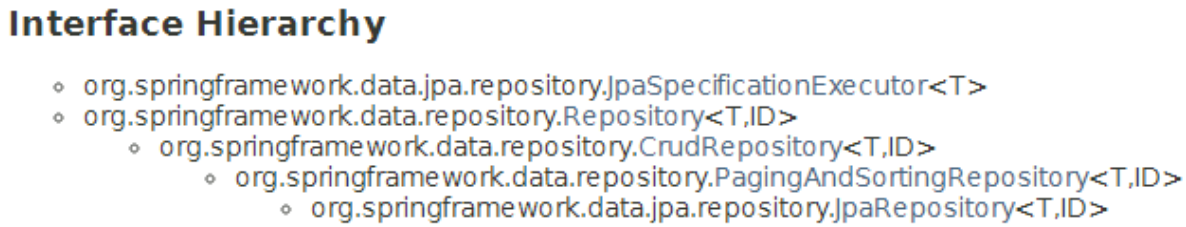
Konfigurace má dvě části, jednou částí je povolení repository pomocí anotace *@EnableJpaRepositories* a zadání cesty (balíčku), kde jsou jednotlivé implementace potřebných repository do pole parametru *basePackages* (řádek 24 na obrázku Obrázek 21: EnableJpaRepositories), dále pak definice *EntityManager* a *TransactionManager* což jsou třídy potřebné pro práci a připojení k databázi (obě definovány v *Application.java*). Podrobněji je konfigurace těchto tříd zmíněna v kapitole Spring data.

```
23  @Configuration
24  @EnableJpaRepositories(basePackages = {"cz.cvut.fel.zdrhajan.test1.repository"})
25  @EnableTransactionManagement
26  @ComponentScan(basePackages = {"cz.cvut.fel.zdrhajan.test1.repository", "cz.cvut
27  public class Application {
```

Obrázek 21: EnableJpaRepositories

6.1.4.2.2 Programovací část

Repository jako takové je dodané kontejnerem, úkolem programátora je vytvořit rozhraní a tedy pouze definovat funkce, které bude repository mít. Implementace je následně vytvořena frameworkem. Jako výchozí rozhraní pro rozšíření je potřeba použít rozhraní *CrudRepository<T,ID extends Serializable>*, které poskytuje funkce pro základní práci s daty (viz Obrázek 23: CRUDRepository funkce). Funkce rozhraní se svými názvy od CRUD liší, v zásadě jde ale jen o změnu pojmenování, místo read je find, místo create a update je save a delete je stejné, navíc je zde z praktických důvodů findAll (vrať všechny položky) a count (vrať počet položek).



Obrázek 22: Hierarchie repository rozhraní

Další rozhraní vhodná pro dědění vlastních rozhraní jsou specifičtí potomci tohoto rozhraní (viz Obrázek 22: Hierarchie repository rozhraní) nebo vlastní rozhraní (vytvořená děděním, často anotovaná *@NoRepositoryBean*), například rozhraní rozšířené o stránkování a řazení (*PagingAndSortingRepository*) nebo rozhraní rozšířené o možnost definovat vlastní dotazy (*JpaRepository*).

Method Summary	
long	count() Returns the number of entities available.
void	delete(ID id) Deletes the entity with the given id.
void	delete(Iterable<? extends T> entities) Deletes the given entities.
void	delete(T entity) Deletes a given entity.
void	deleteAll() Deletes all entities managed by the repository.
boolean	exists(ID id) Returns whether an entity with the given id exists.
Iterable<T>	findAll() Returns all instances of the type.
T	findOne(ID id) Retrives an entity by its primary key.
Iterable<T>	save(Iterable<? extends T> entities) Saves all given entities.
T	save(T entity) Saves a given entity.

Obrázek 23: CRUDRepository funkce

Repository umožňuje generování implementace funkcí na základě jejich názvů nebo na základě definovaného *@NamedQuery* nebo *@Query*.

Generování na základě názvu vychází z manuálu a standardu pro pojmenovávání funkcí v repository. Na základě těchto názvů je Framework schopen automaticky generovat jednotlivé funkce. Nejčastěji se tohoto využívá při specifických find-funkcích. Ve vyvíjené aplikaci je toto využito například v *UserRepository,* kde je potřeba najít uživatele dle uživatelského jména (username). Není potřeba složitě definovat nějaký filtr nebo psát ručně dotaz v JPQL. Stačí pouze definovat funkci se správným názvem (řádek 13 Obrázek 24: UserRepository) a framework dodá sám její implementaci. V tomto případě je název funkce složen z „findBy“, která určuje, že se bude hledat a budou potřebné parametry a dále je zde „Username“, což značí, že se bude filtrovat podle atributu username, jehož hodnota je předána ve stejně pojmenovaném parametru. Další klíčová slova pro tvoření názvů funkcí je možné najít v tabulce 2.3 na [3].

```
11    public interface UserRepository extends CrudRepository<User, Long> {
12
13        List<User> findByUsername(String username);
14    }
15
```

Obrázek 24: UserRepository

Existují i další možnosti definice funkcí repository. Jednak pomocí anotace *@NamedQuery* přímo v entitní třídě, která může přepsat jednotlivé funkce i v repository pokud se jmenují stejně jako definovaný dotaz (Obrázek 25: NamedQuery).

```
@NamedQueries({
        @NamedQuery(
        name = "findStockByStockCode",
        query = "from Stock s where s.stockCode = :stockCode"
        )
})
@Entity
@Table(name = "stock", catalog = "mkyong")
public class Stock implements java.io.Serializable {
```

Obrázek 25: NamedQuery

Další možností je pak definice dotazu přímo v repository u dané funkce, dotaz je pak zapsán v JPQL, přímo u funkce která ho realizuje (Obrázek 26: Query v Repository).

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query("select u from User u where u.emailAddress = ?1")
  User findByEmailAddress(String emailAddress);
}
```

Obrázek 26: Query v Repository

6.1.4.3 *ImprovedNamingStrategy*

ImprovedNamingStrategy je třída realizující převod jmen anotovaných entitních tříd a jejich atributů na jména databázových objektů. Poskytuje základní strategii pro pojmenování a jejím poděděním a přepsáním jednotlivých funkcí je možné vytvořit vlastní způsob pojmenování objektů. Toho je možné výhodně využít v rámci této práce pro naplnění části požadavků na pojmenování databázových objektů.

V současné chvíli bohužel tato třída neumožňuje programově určovat jména všech generovaných databázových objektů, proto je potřeba některé objekty pojmenovat ručně. Tam, kde to umožňuje, je výhodné použít tento způsob. Z tohoto důvodu je v aplikaci využito kombinace ručního a automatického pojmenování objektů.

Na příkladu entitní třídy (Obrázek 20: Entita) je dobře vidět jak je ručně uvedeno jméno sloupce identifikátoru (@Id), je potřeba využít anotace *@Column* a jejího parametru *name* a ručně definovat název sloupce („QCS_DEPARTMENT_NO“).

Naproti tomu jméno tabulky, pokud není ručně definováno, je určeno podle názvu třídy. Toto je i náš případ, neexistující anotace *@Table(name="MyEntityTableName")* způsobí, že výchozí instance třídy *ImprovedNamingStrategy* pojmenuje databázovou tabulku stejně jako třídu, tedy „DEPARTMENT“. Toto chování lze změnit právě zděděním z *ImprovedNamingStrategy* a přepsáním funkce *classToTableName* (řádek 14 na obrázku Obrázek 27: DatabaseNamingStrategy). Tato funkce dostává parametrem název třídy a měla by vrátit požadovaný název tabulky. V našem případě jde tedy jen o přidání prefixu „QCS_“, který byl určen IT oddělením zákazníka. V současné době je bohužel možné takto upravovat jen jména tabulek.

```
9   public class DatabaseNamingStrategy extends ImprovedNamingStrategy {
10
11      private static final String PREFIX = "QCS_";
12
13      @Override
14      public String classToTableName(final String className) {
15          return this.addPrefix(super.classToTableName(className));
16      }
17
18      private String addPrefix(final String composedTableName) {
19          return PREFIX + composedTableName.toUpperCase().replace("_", "");
20      }
```

Obrázek 27: DatabaseNamingStrategy

Vytvořenou třídu *DatabaseNamingStrategy* je ještě potřeba zaregistrovat přidáním jejího umístění a názvu do *JpaProperties* entitymanageru (*LocalContainerEntityManagerFactoryBean*). V našem případě vložením hodnoty *"cz.cvut.fel.zdrhajan.test1.DatabaseNamingStrategy"* k property *"hibernate.ejb.naming_strategy"*. Toto se děje ve třídě *Application*, na řádku 49 (Obrázek 28: DatabaseNamingStrategy nastavení)

```
43      @Bean
44      public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource dataSource, JpaVendorA
45          LocalContainerEntityManagerFactoryBean lef = new LocalContainerEntityManagerFactoryBean();
46          lef.setDataSource(dataSource);
47          lef.setJpaVendorAdapter(jpaVendorAdapter);
48          Properties props = new Properties();
49          props.put("hibernate.ejb.naming_strategy", "cz.cvut.fel.zdrhajan.test1.DatabaseNamingStrategy");
50          lef.setJpaProperties(props);
51          lef.setPackagesToScan(new String[]{"cz.cvut.fel.zdrhajan.test1.repository", "cz.cvut.fel.zdrhaja
52          return lef;
53      }
```

Obrázek 28: DatabaseNamingStrategy nastavení

6.2 Framework pro vytváření reportů

S reportovacím frameworkem jsem, na začátku vývoje, neměl prakticky žádné zkušenosti, bylo tedy potřeba tento framework vybrat z dostupných řešení. Základním parametrem byla dostupnost frameworku zdarma, a možnost výstupu do obou požadovaných formátů – PDF a XLSX.

Framework na vytváření reportů měl dále splňovat řadu požadavků. Například možnost programově doplňovat data, dynamicky měnit strukturu reportů, možnost formátování textu i jiných částí reportu

možnost stránkování a možnost sloučení několika reportů dohromady, report v PDF měl navíc umožnit vkládání obrázků.

6.2.1 Formáty PDF a XLSX

V první fázi jsem objevil framework JasperReports Library [4], jako jediný nalezený umožňoval výstup do obou formátů. Navíc se jednalo o volně distribuovanou součást komerčního produktu, čemuž odpovídala i kvalita podpory na fórech a dokumentace. Po vytvoření prototypu jsem ale narazil na problém s reportem v Excelu, report vypadal stejně jako report do PDF, tedy obsahoval i záhlaví a zápatí stránek a číslování stránek a vše bylo v jednom sešitu. Výsledný report vypadal stejně jako PDF dokument v Excelu. To bylo samozřejmě pro zveřejnění nepoužitelné. Dalším problémem byla nemožnost dynamicky manipulovat s reportem, to bylo možné jen vytvořením rozdílných šablon pro jednotlivé dokumenty.

6.2.2 Formát PDF

Vzhledem k tomu, že jsem nenašel další vhodné kandidáty, kteří by uměli export do obou formátů, byl jsem nucen použít dva nástroje, pro každý formát jiný. Pro PDF jsem byl spokojen s výstupy z JasperReports, jen chyběla možnost skládat dohromady reporty dle potřeby a programově manipulovat s jejich strukturou. Při dalším hledání jsem narazil na DynamicReports [5], výhodou knihovny bylo, že nebylo potřeba vytvářet šablonu, ale vše se dělo programově až ve chvíli vytváření reportu. Byla zde tedy řada možností jako report dynamicky upravovat, formátovat dle daných hodnot a slučovat jednotlivé reporty dohromady, tento nástroj plně vyhověl požadavkům, které jsem na vytváření reportů do PDF kladl.

Při použití této knihovny je potřeba definovat strukturu reportu, k čemuž slouží ReportBuilder a následně tomuto builderu předat zdroj dat pro report (datasource). Jak je vidět z ukázky kódu (Obrázek 29: Report PDF) je definování struktury snadno čitelné. Na začátku vytvořím samotný builder (řádek 206), definuji hlavní nadpis (title, řádek 207), který se skládá z textové komponenty (řádek 208) a mezery pod nadpisem (řádek 209). Následuje definice formátování, která se postará o zvýraznění sudých řádků pro lepší čitelnost výsledného reportu (řádek 211). Pomocí funkce *addColumn* jsou definovány jednotlivé sloupce reportu (řádky 212-214). Funkce *addField* slouží k definici atributu objektu, který se bude v reportu používat (řádek 215, ostatní atributy jsou definovány v definicích jednotlivých sloupců). Pomocí funkce *subtotalsAtSummary* je možné definovat kumulativní hodnoty do zápatí jednotlivých sloupců (řádky 216-220). Funkce *summary* pak přidá do zápatí reportu jeho celkové hodnocení (řádek 222). A konečně na řádku 223 předám builderu zmiňovaný *datasource* obsahující jednotlivé položky (řádky) výsledného reportu.

```
206         JasperReportBuilder answerListsReport = report()
207                 .title(
208                         cmp.text(ReportConstants.VYHODNOCENI_SUMMARY).setStyle(stl.style().bold()),
209                         cmp.gap(1, 10)
210                 )
211                 .highlightDetailEvenRows()
212                 .addColumn(department)
213                 .addColumn(subDepartment)
214                 .addColumn(percentage)
215                 .addField(field("questionList.subDepartment", SubDepartment.class))
216                 .subtotalsAtSummary(
217                         sbt.text("RESULT", department).setStyle(resultStyle),
218                         sbt.text("", department).setStyle(resultStyle),
219                         sbt.text(auditResults.getResultScore().toString(), percentage).setStyle(stl.
220                         sbt.text(percentFormater.format(AuditResultsService.getPercentScore(auditRes
221                 )
222                 .summary(cmp.gap(1, 5), cmp.text(ReportConstants.VYHODNOCENI_FINDINGS).setStyle(stl.
223                 .setDataSource(answerLists);
```

Obrázek 29: Report PDF

6.2.3 Formát XLSX

Výběr knihovny pro reportování do Excelu vycházel z podobného problému jako PDF, v JasperReports nebylo možné přistupovat k jednotlivým listům sešitu, nebylo možné report dynamicky upravovat při generování a report neodpovídal struktuře, na jakou jsou uživatelé v Excelu zvyklí (například obsahoval po každých 20 řádcích reportu číslo stránky). Z těchto důvodů bylo potřeba nalézt jiný nástroj. Jako nejčastější alternativa k JapsperReports pro Excel je uváděna knihovna Apache POI [6], která je vyvíjena v rámci Apache Software Foundation a zaměřuje se na manipulaci s daty v dokumentech MS Office.

V tomto nástroji je potřeba procházet jednotlivé buňky a doplňovat do nich jednotlivé hodnoty, následně buňky slučovat, případně formátovat k čemuž je možné využít oblasti větší než jedna buňka. Jako největší problém se ukázalo vytváření modelu vznikajícího sešitu, kde jsem využíval vytváření buněk (řádek 517 na Obrázek 30: Report XLSX) a řádek, framework tyto objekty vytvářel jako nové, když byly předtím vytvořeny, to způsobovalo rozházení hodnot tím, jak se na místo jedné buňky vytvořily dvě. Tomuto chování jsem nakonec předešel použitím tovární funkce getOrCreateRow (řádek 515 na Obrázek 30: Report XLSX), která se pokusila objekt získat a pokud již existoval, vrátila ho, a pokud ne vytvořila objekt nový. Na ukázce kódů (Obrázek 30: Report XLSX) je dále vidět nastavení výšky řádku (řádek 516) a nastavení hodnoty (řádek 518) a stylu (řádek 519) buňce.

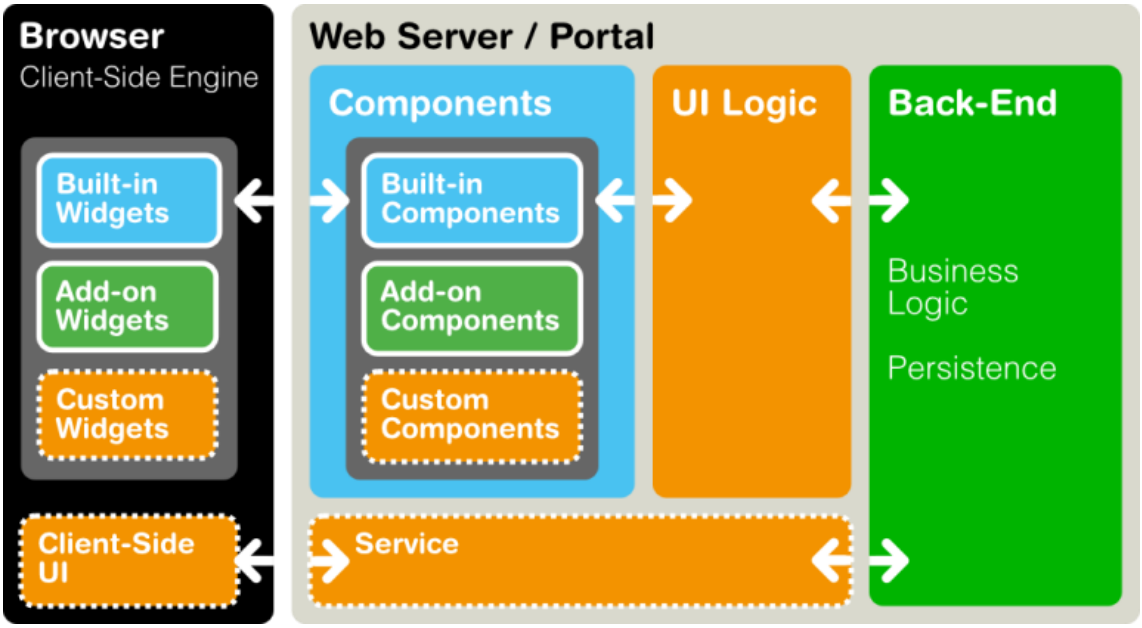
```
515             row = getOrCreateRow(sheet, rowIndex++);
516             row.setHeightInPoints((short) 27);
517             cell = row.createCell(1);
518             cell.setCellValue(answerList.getPointsMaximum().doubleValue());
519             cell.setCellStyle(answerListSummaryValueStyle);
```

Obrázek 30: Report XLSX

6.3 Vaadin & Touchkit

Vaadin [2] je framework určený pro vývoj webových aplikací. Jeho hlavním zaměřením je vývoj uživatelského rozhraní a prezentační vrstvy (View a Controller z MVC). Jeho základem je GWT framework od Google, nad kterým Vaadin vytváří jeho serverovou část a to tak, že je programátor naprosto oddělen od nutnosti programovat klientskou část aplikace jako takovou v jejích jazycích. Klientská část je programována v jazyku Java, stejně jako zbytek aplikace a následně je kompilována do JavaScriptu, čímž zvyšuje produktivitu programátora a zbavuje ho nutnosti učit se další jazyk a

koncepce. Díky tomu se programátor může soustředit více na logiku aplikace, místo na HTML a JavaScript samotný.



Obrázek 31: Vaadin architektura, převzato z [2]

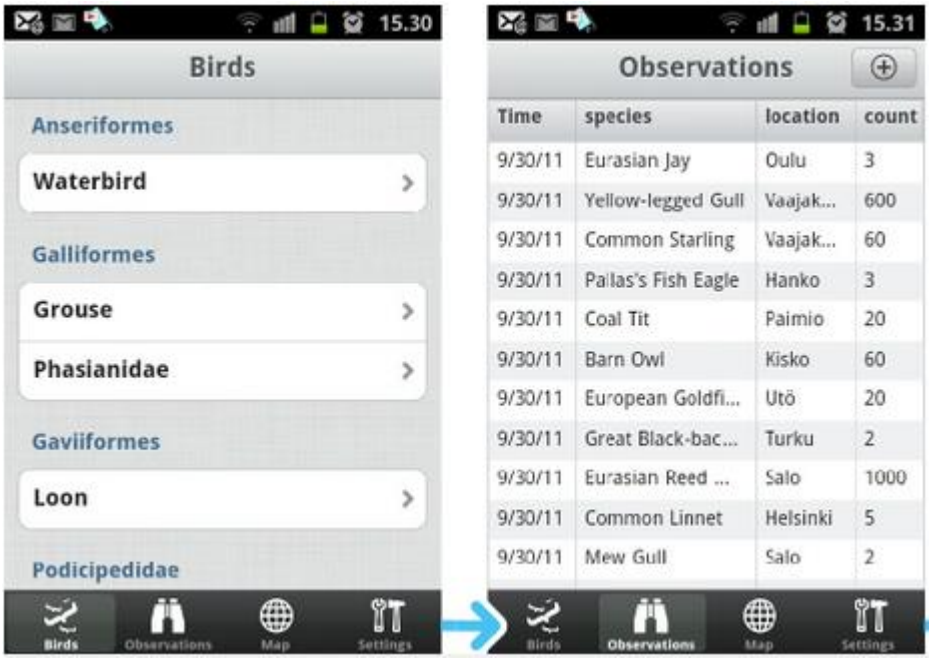
Obrázek (Obrázek 31: Vaadin architektura) ukazuje základní architekturu webové aplikace vytvořené pomocí frameworku vaadin serverová část se skládá ze serverové části frameworku vaadin a komunikační části a logiky aplikace. Komunikační část komunikuje s klientskou částí, které běží na klientské stanici ve webovém prohlížeči jako JavaScriptový kód. Logika uživatelského rozhraní je pak v Java servletu a běží na aplikačním serveru.

Klientská část, jak už jsem zmínil, je tvořena pouze JavaScriptovým kódem, ten může být snadno vykonáván v běžně používaných prohlížečích, to je velmi důležité, protože pro běh takové aplikace není potřeba žádný plugin nebo jakýkoliv jiný program, který by bylo nutné na klientské straně instalovat. To je značný problém například u frameworků založených na Flashi, Java appletech a podobně. Vaadin vychází z frameworku GWT (Google web toolkit) a využívá jeho komponenty, díky tomu podporuje širokou škálu prohlížečů a jejich verzí, stejně jako GWT samotný.

Propojení klientské a serverové části zaštiťuje také vaadin, prakticky to tedy znamená, že na serveru není potřeba se vůbec zabývat tím, zda je aplikace webová nebo ne. Vývoj pak vypadá podobně jako při vývoji desktopové aplikace za použití standartních javovských knihoven pro GUI - AWT nebo SWING. Na druhou stranu je možné vytvořit si vlastní komponenty a tak rozšířit již existující sadu komponent o vlastní specifické funkce. Vývoji vlastní komponenty se věnuji v kapitolách Komponenta pro upload obrázků a Vytvoření vlastní komponenty.

6.3.1 Touchkit

Touchkit [7] je addon vaadinu vyrobený stejnou společností (Vaadin Ltd.) jeho hlavní součástí jsou komponenty specifické pro tvorbu uživatelských rozhraní na mobilních zařízeních a také grafika uživatelských rozhraní (témata, themes), která připomínají nativní uživatelské prvky (viz Obrázek 32: Rozhraní Vaadin Touchkit).



Obrázek 32: Rozhraní Vaadin Touchkit, převzato z [7]

Takovou webovou aplikaci je možné na platformě iOS posunout ještě blíže k nativní aplikaci pomocí několika parametrů a nastavení. Zmiňovaná nastavení ukazuje ukázka kódu (Obrázek 33: HTML header). Nejdůležitějším parametrem je *apple-mobile-web-app-capable* (řádek 7), který umožňuje nastavit aplikaci jako nativní, tedy například nezobrazovat pruh prohlížeče při jejím otevření z plochy zařízení. Dále je potřeba přidat ikonu aplikace, která bude na ploše zařízení použita (řádek 9). Takto nastaven ou stránku je potom potřeba navštívit pomocí prohlížeče Safari a vytvořit pro ni odkaz na ploše zařízení (AirDrop -> Add to Home screen), tím se na ploše vytvoří ikona dle dodaného obrázku s popiskem dle atributu title stránky, která po kliknutí spustí aplikaci ve speciálním režimu prohlížeče, kdy není vidět jeho záhlaví, aplikace tak zabírá celou obrazovku.

```
7   <meta name="apple-mobile-web-app-capable" content="yes">
8   <link rel="icon" href="./themes/qcsTheme/favicon.ico" type="image/x-icon" />
9   <link rel="apple-touch-icon" href="./themes/qcsTheme/favicon.ico">
```

Obrázek 33: HTML header

6.3.2 Vytvoření vlastní komponenty

Jak bylo zmíněno v kapitole Komponenta pro upload obrázků, bylo v rámci projektu potřeba vytvořit novou komponentu, která by umožnila zmenšení vyfocených obrázků ještě před jejich odesláním na server. Toto se ukázalo jako komplikovaný problém, který jsem vyřešil až s pomocí podpory frameworku na jejich fóru. Doporučeným řešením bylo vytvořit novou komponentu, která by využívala knihovny lib-gwt-imageupload [8] pro manipulaci s obrázky a odeslat výsledný obrázek na server pomocí ServerRPC. Ukázka manipulace s obrázky je v demu, které je poskytnuté k Touchkitu [9], respektive jeho zdrojových kódech [10].

Nově vzniklá komponenta i s klientskou (widget) a komunikační částí (Connector, ClientRPC a ServerRPC) se nachází v balíčku *cz.cvut.fel.zdrhajan.test1.vaadin.gwt.client*. Widget je prakticky komponenta Upload, která místo standartní akce (nahrání na server) provede zmenšení obrázku a následně jeho odeslání přes komunikační část. Ta je napojena na serverovou komponentu

UploadImageResize, která se chová stejně jako standartní komponenta Upload a je tedy možné zaregistrovat k ní standartní listenery, které se postarají o zpracování příchozího obrázku.

Vzhledem k použití doplňkové knihovny pro manipulaci s obrázky je potřeba zaregistrovat tuto knihovnu do widgetsetu, tak aby byla v klientské části aplikace dostupná. To provedu přidáním definice knihovny do konfiguračního souboru *AppWidgetSet.gwt.xml* (řádek 16, Obrázek 34: Widgetset konfigurace).

```
9     <!-- Inherit DefaultWidgetSet -->
10    <inherits name="com.vaadin.DefaultWidgetSet" />
11
12    <inherits name="com.vaadin.addon.touchkit.gwt.TouchKitWidgetSet" />
13
14    <inherits name="org.vectomatic.libgwtfile"/>
15
16    <inherits name="org.vaadin.juho.imageupload.libgwtimageupload"/>
17
```

Obrázek 34: Widgetset konfigurace

7 Nasazení

V rámci nasazení aplikace do jednotlivých prostředí bylo potřeba vytvořit instalační sadu pro vyvíjenou aplikaci. Tato sada se skládá zejména z databázových skriptů potřebných pro vytvoření databázových struktura samotné aplikace. Dále jsou pak dodány skripty určené pro import dat a samotná aplikace jednak ve formě war souboru a také ve formě zdrojových kódů. Aplikace byla doplněna o dokumentaci, která umožňuje rychlý start případným budoucím vývojářům, tak aby byli schopni aplikaci upravit, sestavit a nahrát na server.

Uživatelé při testování aplikace již vytvořili užitečná data, která by potřebovali přenést i na produkční prostředí, toto bude vykonáno pomocí zálohování databáze testovacího serveru a následného nahrání této zálohy na server produkční, aplikace bude při tomto procesu odstavena.

7.1 Scénář instalace

Při instalaci aplikace je potřeba postupovat podle následujícího scénáře. Při instalaci předpokládám existenci aplikačního serveru JBoss 7.1, s potřebnými závislostmi (případně možno upravit v pom.xml) a databáze Oracle 11g.

1. Na databázovém serveru bude spuštěn skript pro instalaci databáze (createdb.sql) který vytvoří strukturu databáze (tabulky, sequence, indexy, constrains a synonyma pro tabulky a sequence)
2. Nyní je možné do databáze nahrát data, pokud je to potřeba (existují užitečná data ze záloh, nebo předchozí uživatelské práce před migrací)
3. V aplikačním serveru bude přítomen driver pro připojení do databáze a bude nastaven datasource do vytvořené databáze. Očekávané jméno datasource je „QualityCheck“. Konfigurace bude podobná jako na Obrázek 35: Konfigurace datasource. Po nastavení je potřeba restartovat aplikační server
4. Nyní je možné na aplikační server nahrát sestavenou aplikaci (war soubor) a ten spustit
5. Nyní je možné aplikaci otevřít v prohlížeči na adrese serveru
6. Pokud nejsou v aplikaci zadáni žádní uživatelé, je možné se přihlásit výchozím jménem a heslem, obojí je možné nalézt ve zdrojových kódech v *UserDetailsServiceImpl.java*

```
104         <datasource jndi-name="java:jboss/datasources/QualityCheck" pool-name="QualityCheck"
105             <connection-url>jdbc:oracle:thin:@localhost:1521:orcl</connection-url>
106             <driver>OracleJDBCDriver</driver>
107             <security>
108                 <user-name>system</user-name>
109                 <password>********</password>
110             </security>
111         </datasource>
112         <drivers>
113             <driver name="OracleJDBCDriver" module="oracle.jdbc"/>
114         </drivers>
```

Obrázek 35: Konfigurace datasource

8 Testování

Testování probíhalo po celou dobu vývoje aplikace, nejdříve se jednalo o testování návrhů uživatelského rozhraní, kde na prototypech uživatelského rozhraní pozorovali uživatelé, zda se jim s rozhraním dobře pracuje. Následně uživatelé dostávali postupně jednotlivé samostatné funkční celky aplikace, které na týdenní bázi ověřovali a předávali připomínky k nim. Připomínky jsem do aplikace postupně zapracovával.

Když byla již aplikace použitelná při skutečném auditu, vzali ji uživatelé sebou do provozu. Zaznamenávali výsledky duplicitně do aplikace i do původních pomůcek a tak aplikaci testovali. Po několika kolech připomínek bylo takto ověřeno, že aplikace splňuje jejich původní požadavky. Tímto vzetím aplikace do provozu byly prakticky nahrazeny pilotní i akceptační testy a aplikace tak začala sloužit svému účelu ještě před oficiálním dokončením a předáním.

V rámci testování uživatelé zadávali již užitečná data z auditů, což jim následně umožňovalo testovat i výstupy (reporty) a seznámit s těmito výstupy další oddělení, kterým výstupy předávají. Takto byla prakticky celá aplikace testována a otestována ještě dříve než k testovací fázi prakticky došlo.
Uživatelé při testech v praxi narazili na několik dalších možností rozšíření, které uvádím v samostatné kapitole

Rozšíření a vylepšení požadovaná uživateli.

Správnost výpočtů aplikace byla ověřena oproti historickým auditům, audity byly zadány do aplikace a následně vyhodnoceny. Bylo porovnáno hodnocení jednotlivých oddělení a celého auditu a při shodě třech takovýchto testů bylo vyhodnocování auditů prohlášeno za funkční a otestované.

Správnost reportů byla taktéž ověřována oproti historickým auditům. Report do XLSX byl sestavován tak aby odpovídal původnímu zadání, kdy byly výsledky auditu zapisovány do Excelu. Bylo tedy snadné následně porovnat vytvořený report s původním historickým reportem, ze kterého byla data opsána do aplikace. Zároveň se při testování do aplikace doplnila část historických dat.

Uživatelé akceptovali aplikaci jako odpovídající jejich požadavkům.

Dalším testováním aplikace byla na straně IT oddělení zákazníka konzultace s jeho pracovníky a nasazení aplikace do testovacího prostředí. Při tomto testu bylo ověřeno splnění požadavků tohoto oddělení, tedy zejména správnost pojmenování databázových objektů, smysluplnost a efektivita databáze. Zároveň byla uživatelskými testy při běhu na testovacím prostředí prokázána stabilita aplikace.

Zaměstnanci IT oddělení zákazníka akceptovali aplikaci jako odpovídající jejich požadavkům.

9 Rozšíření a vylepšení požadovaná uživateli

Při testování aplikace a její prezentaci dalším uživatelům byly nalezeny další možnosti rozšířen aplikace za účelem usnadnění práce uživatelů. Jedná se o možnosti čtení čárových kódů, zlepšení možností fotografování, úprava aplikace pro použití na jiných klientských zařízeních, možnost hromadného stahování obrázků a možnost importu starých auditů. V této kapitole prodiskutuji možnosti rozšíření aplikace v těchto směrech.

9.1 Čtení čárových kódů

Uživatelé při práci na storech zaznamenávají jednotlivé výrobky do seznamů v aplikaci, ke každému výrobku je potřeba zadat název, datum spotřeby, počet kusů výrobku a případě šarži výrobku, pokud se jedná o výrobky, kde se šarže sleduje. V současné době je potřeba tyto údaje zapisovat ručně, což je zdrojem chyb a zdržení při auditu. Aby se zabránilo těmto negativům, bylo navrženo, že by aplikace mohla pomocí vestavěného fotoaparátu číst čárové kódy z výrobků a z centrální databáze tyto údaje sama doplnit.

Toto rozšíření je možné, ale relativně náročné. Je potřeba realizovat další komponentu, která bude z klientské části spouštět externí aplikaci určenou pro čtení čárových kódů pomocí vestavěného fotoaparátu. Takovéto aplikace existují, ale jen zlomek z nich má možnost spouštění přímo z webového prohlížeče (pomocí přesměrování webové adresy na klientském zařízení). Následně je potřeba zpracovat výsledek předaný zpět aplikací (opět pomocí webové adresy). Poté je možné přidat položku do seznamu před čímž je ještě potřeba spojit se se serverem a v databázi výrobků najít příslušnou položku.

9.2 Zlepšení fotografování

Uživatelé pořizují při auditech fotografie i na špatně osvětlených místech. V těchto případech je kvalita výsledné fotografie vlivem nedostatku světla špatná. Dodané klientské zařízení bohužel nedisponuje přisvětlovací diodou k fotoaparátu ani jiným zdrojem světla. Na aplikační straně bohužel není možné z fotografie „dostat co tam není“. Z tohoto důvodu bylo uživatelům vysvětleno, že jedinou možností je mít jiný zdroj světla, například baterku, a tou dané místo před vyfocením osvítit.

9.3 Úprava aplikace pro použití na jiných klientských zařízeních

Při práci na auditech byly zaznamenány stížnosti uživatelů na velikost klientského zařízení, toto zařízení bylo pořízeno zákazníkem, není tedy v možnostech dodavatele s tímto problémem něco udělat. V případě vyvinuté aplikace není problém změnit zařízení, na kterém bude aplikace nasazena. Někteří uživatelé požadovali aplikaci použít na zařízení iPhone, což teoreticky není problém. Problém bude pouze s ergonomií při používání, uspořádání jednotlivých grafických prvků aplikace je přizpůsobeno většímu displeji. Pomocí frameworku vaadin je možné přizpůsobit aplikaci různým velikostem displeje a zařízením. Případná změna formátu a velikosti displeje tedy nebude problém a všechny úpravy budou motivované pouze ergonomií, nikoli funkcemi samotnými.

9.4 Hromadné stahování obrázků

Uživatelé vznesli při přípravě na kalibrační audit (školení auditorů) požadavek na možnost stažení všech fotografií obsažených v aplikaci, případně na možnost stažení fotografií z jednotlivých auditů. Tento požadavek je snadno řešitelný, základní možnosti jsou dvě.

První možností je realizovat v rámci aplikace další akci, která umožní stažení balíku (zip soubor) fotek ze serveru kde bude balík připraven načtením fotek z databáze. Toto řešení lépe vyhovuje zapouzdření funkcionality do aplikace (co se týká aplikace, je, pokud možno, v aplikaci).

Druhou možností je vytvoření databázového jobu, který by v pravidelných intervalech četl soubory z databáze a nahrával je prostřednictvím sítě na sdílený disk, kam mají přístup všichni uživatelé. Toto řešení přenáší logiku mimo aplikaci a bude tedy náročnější na údržbu. Z pohledu zákazníka se ale jedná o řešení efektivnější, zejména kvůli přenášení souboru po síti jen jednou a následně jeho nalezení na síťovém disku.

9.5 Import historických auditů

Po vytvoření několika auditů pomocí aplikace uživatelé narazili na potřebu vytvořit historické porovnání auditů, do tohoto porovnání měly vstoupit i audity vykonané před nasazením aplikace. Bylo tedy požadováno, aby aplikace obsahovala modul pro import historických auditů. Po analýze dodaných historických auditů bylo zjištěno, že data v těchto auditech nejsou správně (konzistentně) strukturována (například neshody byly uvedeny jako pokračování textu v buňce otázky, jen jinou barvou, stejně tak byly pokaždé jinak organizované seznamy výrobků u otázek). Z těchto důvodů a jednorázovosti využití bylo rozhodnuto, že tato funkce nebude realizována a data budou do aplikace vyplněna ručně.

10 Integrace, využití hardware

Ve vyvinuté aplikaci jsou tři prvky, které zasluhují z hlediska vývoje mobilních webových aplikací zvláštní pozornost, jde o pseudonativní mód aplikace na iOS, pořizování fotografií z prohlížeče a konečně chystané propojení s externí aplikací pro čtení čárových kódů.

10.1 Pseudonativní mód aplikace

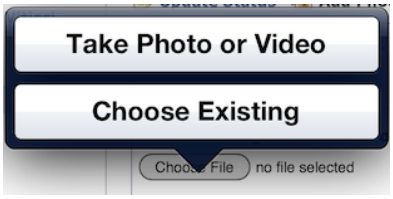
Tento mód jsem již zmínil v kapitole Touchkit, jde o speciální mód webového prohlížeče, kdy je aplikace zobrazena na celou obrazovku. Tohoto chování docílím pomocí nastavení několika parametrů (metatagů stránky) tak jak je prezentovaného na obrázku Obrázek 33: HTML header. Po navštívení takové stránky a uložení jejího odkazu na Home screen zařízení se bude aplikace spouštět ve fullscreen módu. Takto se bude aplikace, společně s nativním tématem grafiky, tvářit jako by se jednalo o skutečně nativní aplikaci. Získávám tak další výhodu, která byla webovým aplikací donedávna odepřena, celoobrazovkový mód a ikona s názvem a přímým spuštěním aplikace přímo na ploše. Toto nastavení je specifické pro prostředí iOS. Zároveň je nastavení tohoto parametru podporováno i přímo vaadinem, stačí provést nastavení při inicializaci servletu (řádek 23, Obrázek 36: WebAppCapable parameter).

```
20      @Override
        protected void servletInitialized() throws ServletException {
22          super.servletInitialized();
23          getTouchKitSettings().getWebAppSettings().setWebAppCapable(true);
```

Obrázek 36: WebAppCapable parameter

10.2 Pořizování fotografií z prohlížeče

Další vlastností prohlížeče Safari na mobilních zařízeních, je možnost vložit do pole výběru souboru fotografii pořízenou fotoaparátem. Výchozí chování pro toto pole je nabídnutí bubliny, kde si uživatel zvolí, zda chce vyfotit fotografii, natočit video nebo vybrat soubor ze zařízení (Obrázek 37:Upload a jeho možnosti).



Obrázek 37:Upload a jeho možnosti

Tyto možnosti je možné ovlivnit pomocí nastavení atributů tagu *input*, toto je vidět na obrázku Obrázek 38: Tag input a jeho atributy, kde dochází k nastavení atributu *accept* a *capture*. Atribut *accept* určuje typ filtru, který má být na soubory nebo zdroje aplikován - v tomto případě by byly k dispozici jen obrázky, volba pro pořízení záznamu videa by tedy nebyla zobrazena. Dále je zde nastaven atribut *capture*, který nastavuje zdroj, odkud má být soubor požadován, provedené nastavení by tedy nabídlo jen možnost pořídit fotografii pomocí fotoaparátu. Správná funkce těchto atributů není zatím realizována ve všech prohlížečích. Vzhledem k definovanému cílovému zařízení jsem si ji po otestování dovolil použít.

```
<input type="file" accept="image/*" capture="camera">
```

Obrázek 38: Tag input a jeho atributy

11 Závěr

Mým cílem v rámci této práce bylo vyvinout aplikaci dle zákazníkových požadavků. Pro splnění tohoto úkolu jsem se rozhodl postupovat podle metodiky Unified proces (UP). Postupně jsem prošel jednotlivými částmi procesu vývoje aplikace, tak jak jsou v této metodice definovány.

V počátečních fázích sběru požadavků a analýzy, jsem se věnoval zejména práci s uživateli a ověření komplexnosti a konzistence požadavků. Za nejsložitější v této části považuji zpracování požadavků z několika směrů - oddělení, které se u mě postupně scházely. Tyto požadavky uživatelé doplňovali k „základnímu“ zadání, které mi předali jako úvod do problematiky ještě před začátkem prací. Zákazník požadoval, abych se za pomoci základního zadání seznámil s problémovou doménou a byl tak při další práci budoucím uživatelům rovnocenným partnerem.

Po ustálení požadavků jsem pracoval na návrhu aplikace a vytváření prototypů. V této části jsem vytvořil modely aplikace, které zajistí splnění požadavků. Přitom jsem se věnoval sjednocení uživatelského rozhraní, tak aby bylo jednoduché a srozumitelné a pokud možno podobné ve všech částech aplikace. To bylo vzhledem k jednoduchosti aplikace poměrně snadné a tak jsem skončil se dvěma základními (Seznam a Detail) a dvěma speciálními obrazovkami pro usnadnění zadávání odpovědi na otázku.

Při návrhu aplikace jsem se seznamoval s možnostmi jak systém realizovat, zkoumal možnosti, které nabízejí knihovny a další softwarové komponenty a servery, které zákazník požadoval při vývoji použít. Zároveň jsem sledoval jejich kompatibilitu s mě známými knihovnami a komponentami.

UP je značně variabilní a je možné jej upravit v závislosti na specifikách projektu, toho jsem využil v části návrh. V aktivitě návrh se standardně nemluví o konkrétních technologiích a postupech realizace, přičemž tyto úvahy se přenechávají až do fáze implementační. Toto bylo specifické pro tuto práci, kde zákazník již na začátku dodal jako součást zadání i konkrétní technologie, které mají být pro realizaci použité a bylo tedy potřeba i tyto požadavky zahrnout do úvah ještě před fází implementace.

V implementační fázi jsem realizoval navrhnutou aplikaci dle modelů v části návrh. V kapitole implementace popisuji technologie a jejich použití na konkrétních příkladech z vytvářené aplikace (Spring Security, Repository atd.). V průběhu vytváření aplikace jsem se setkal s řadou problémů, které taktéž popisuji i s jejich řešeními (například vlastní komponenta pro nahrání zmenšených obrázků). Našel jsem mnohdy snadná, i když ne úplně komplexní řešení (DatabaseNamingStrategy). Dále v této části rozebírám architekturu a možnosti použitých frameworků a knihoven.

V průběhu implementační fáze jsem v týdenních intervalech předváděl aplikaci uživatelům s komentářem, co jsem za danou periodu vyvinul. Aplikaci jsem také nahrával na vývojový server, kde si ji mohli uživatelé vyzkoušet. To se ukázalo jako velmi vhodné z několika důvodů. Nejdůležitějším bylo to, že uživatelé se při těchto konzultacích přirozeně školili na práci s aplikací, zároveň probíhala i výměna připomínek a zodpovídání dotazů k funkcím a možnostem aplikace. Důležitým faktorem bylo také nadšení uživatelů a probuzení jejich zájmu o novou aplikaci prostřednictvím jejich aktivního zapojení do vývoje.

S tím jak uživatelé aplikaci zkoušeli, došlo přirozeně i k jejím testům v provozu. Nejprve uživatelé využívali aplikaci v kombinaci s původními pomůckami a výsledky zaznamenávali duplicitně i do aplikace. Tím vlastně přešla aplikace do testovací fáze. V této fázi je aplikace asi 2 měsíce a postupně

dochází k odstranění chyb nalezených při provozu. V této fázi jsou také definovány nové požadavky na vylepšení či rozšíření aplikace.

Kromě drobných úprav v uživatelském rozhraní nebylo uživateli nalezeno příliš chyb. Nejzávažnějším problémem aplikace je v současných podmínkách její závislost na připojení k serveru. Při realizaci aplikace bylo počítáno s tím, že toto připojení bude stabilní. Jak se ale ukázalo v praxi, připojení je problematické. K problémům dochází na prodejní ploše, kde je signál stíněn například množstvím nápojů vyrovnaných v regálech do značné výšky, v prostorách zázemí například v mrazácích kde je uskladněno maso a které jsou tvořeny kovovými kontejnery, podobné problémy nastávají i v částech auditu, který se odehrává mimo obchod tedy vnější prostory příjmu zboží. Problematickým se také ukázalo přirozené chování klientského zařízení (iPad), které se snaží přepnout vždy ke zdroji konektivity s nejsilnějším signálem. To je v závislosti na místě buď Wi-Fi nebo mobilní připojení. Aplikace se pak při připojení k jinému zdroji resetuje, protože dojde ke ztrátě stavu. Tento problém je při současné koncepci aplikace řešitelný jen těžko (uživatelé mají mít přehled o práci jejich kolegů) a to vytvořením „tlustšího“ klienta, který by využíval úložiště zprostředkovaného prohlížečem.

Mimo zmíněného problému s připojením přicházejí od uživatelů prakticky jen pozitivní ohlasy, zejména se týkají ruční práce, kterou jim aplikace ušetří při zpracování výsledků. Výsledky byly původně zaznamenávány ručně a následně v kanceláři přepisovány do Excelu. Všechny reporty byly vytvářeny z těchto Excelů také ručně. Oproti tomu současný stav umožňuje zadávání dat rovnou do aplikace, sledování průběžných výsledků ve chvíli, kdy audit probíhá a okamžitě po jeho dokončení je možné předat jeho výsledky. Výsledky zároveň vypadají lépe (formát PDF, XLSX) a je možné je případně filtrovat podle uživatele, kterému jsou předávány (3 varianty reportů).

Aplikace bude předána do ostrého provozu v polovině května 2014.

12 Seznam zkratek

Zkratka	Vysvětlení
API	Application Programing Interface
AWT	Abstract Window Toolkit
GUI	Graphical User Interface
GWT	Google Web Toolkit
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IT	Information Technology
JSON	JavaScript Object Notation
MS	Microsoft
MVC	Model-View-Controller
PC	Personal Computer
PDF	Portable Document Format
POJO	Primitive Old Java Objects
QA	Quality Assurance
REST	Representational State Transfer
RPC	Remote Procedure Call
SQL	Structured Query Language
UI	User Interface
UP	Unified Process – metodika vývoje SW
XLSX	Open XML spreadsheet file format

13 Seznam obrázků

14 Bibliografie

[1] J. ARLOW a I. NEUSTADT, UML 2 a unifikovaný proces vývoje aplikací, Brno: Computer Press, 2007.

[2] Marko Grönroos, „Book of Vaadin," Vaadin Ltd., 2014. [Online]. Available: https://vaadin.com/book/-/page/preface.html. [Přístup získán 3 2014].

[3] T. D. Oliver Gierke, „JPA Repositories," 2013. [Online]. Available: http://docs.spring.io/spring-data/jpa/docs/1.4.3.RELEASE/reference/html/jpa.repositories.html. [Přístup získán 28 4 2014].

[4] „JasperReports Library," Jaspersoft Corporation, 2014. [Online]. Available: http://community.jaspersoft.com/project/jasperreports-library. [Přístup získán 3 2014].

[5] R. Mariaca, „DynamicReports," Ricardo Mariaca, 2014. [Online]. Available: http://www.dynamicreports.org/. [Přístup získán 03 2014].

[6] G. S. A. S. R. K. D. F. Andrew C. Oliver, „Apache POI - the Java API for Microsoft Documents," 2014. [Online]. Available: http://poi.apache.org/. [Přístup získán 3 2014].

[7] V. Ltd., „Vaadin TouchKit," Vaadin Ltd., 2014. [Online]. Available: https://vaadin.com/add-ons/touchkit. [Přístup získán 3 2014].

[8] J. Nurminen, „lib-gwt-imageupload," Juho Nurminen, 2014. [Online]. Available: http://vaadin.com/directory#addon/lib-gwt-imageupload:vaadin. [Přístup získán 3 2014].

[9] V. Ltd., „Parking demo," Vaadin Ltd., 2014. [Online]. Available: http://demo.vaadin.com/parking/. [Přístup získán 3 2014].

[10] T. Virkki, „vaadin/parking-demo," Tomi Virkki, 2014. [Online]. Available: https://github.com/vaadin/parking-demo. [Přístup získán 3 2014].