Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

DIPLOMOVÁ PRÁCE

Systém pro rezervaci nabízených služeb od různých poskytovatelů

Plzeň, 2011 Josef Hošna

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 10. května ….....

Abstract

Systém for reservation offered services from diferent providers

Cílem mé práce je navrhnout a částečně implementovat webový systém pro rezervaci nabízených služeb od různých poskytovatelů. Pro návrh takového systému je zapotřebí zpracovat problematiku rezervačních systémů, specifikace požadovaných druhů služeb a možné principy rezervování těchto služeb i možné způsoby plateb přes webové rozhraní. Součástí návrhu aplikace je také vytvoření datového modelu a architektury aplikace, tak aby odpovídala požadavkům zadavatele. Pro tvorbu aplikace budou vybrány vhodné technologie podporující Jazyk Java.

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Martinovi Zímovi , Ph.D. za cenné rady pro tvorbu diplomové práce. Dále bych rád poděkoval své rodině a přátelům za podporu ve studiu a trpělivost při tvorbě této práce.

Obsah

Úvod

Pro vytváření rozsáhlejších J2EE aplikací je na výběr ze dvou přístupů. Jeden z přístupů je, využití při tvorbě základní možnosti J2EE a napsat vše sami bez využití pokročilejších nástrojů. Zde se nemusí zkoumat nové technologie a může se začít dříve psát, ale bude toho ke psaní víc. Druhý přístup je využít některé z pokročilejších nástrojů nebo frameworků, které ušetří práci a čas při vývoji. Vývoj aplikace tímto přístupem, je při znalosti těchto nástrojů kratší. Jiná situace nastává, pokud je používáme poprvé. Zde se musí nové technologie i nástroje nastudovat a vzniká riziko, že čas strávený studováním bude příliš dlouhý. Pak by mohl přesáhnout čas ušetřený ulehčením vývoje a doba tvorby aplikace se nakonec protáhne. Prodloužení doby vývoje je nežádoucí efekt, který vývojářům nikdo nezaplatí. Z toho důvodu většina vývojářů váhá, zda se cestou změny technologie vydat, nebo ne.

V mé Vás seznámím s druhým případem, tzn. že pro vývoj své aplikace jsem zvolil cestu využití nástrojů, které pomáhají při vývoji ušetřit práci a čas. Aplikace, na které budou tyto nástroje představeny, bude webový rezervační systém pro rezervování služeb. Tento systém navrhnu tak, aby bylo možné jej použít pro rezervaci více druhů služeb. Služby bude možné rezervovat od jednoho nebo i více poskytovatelů.

Systém bude mít jedno veřejné rozhraní, ve kterém si zákazníci budou služby vyhledávat, prohlížet a rezervovat. Aby rezervace nebyla úplně bezcennou, bude spojena se zaplacením zálohy. V tomto rozhraní se budou zákazníci registrovat a následně i přihlašovat, aby se dostali k přehledu svých objednávek. Dále bude aplikace obsahovat rozhraní pro poskytovatele služeb, kde budou poskytovatelé zadávat nabízené služby, kontrolovat objednávky a mít přehled svých zákazníků a obsazenosti nabízených služeb.

Aplikace bude webová a data bude mít uložena v databázi PostgeSQL. Bude muset umožňovat zasílání emailů (registrace, faktury) a generovat PDF dokumenty. Pro zjednodušení našeho vývoje jsem zvolil následující nástroje: nástroj pro perzistenci dat Hibernate, aplikační rámec Spring a šablonovací systém Tiles 2.

V druhé kapitole popíši problematiku rezervačních systémů. Pak se budu zabývat druhy služeb pro rezervaci a jejich úskalími. Dál popíši potřebnou funkčnost systému, aby odpovídal požadavkům

zadavatele. Pak bude následovat postup návrhu datového modelu pro databázi tak, aby uchovávala všechna data pro správnou funkčnost. Další kapitola bude popsání nástroje Hibernate a jeho použití, kde bude reverzní vytvoření mapovacích souborů, POJO objektů a DAO tříd, podle databázových tabulek. Dále bude popis aplikačního rámce Spring, jeho využití a konfigurace v našem konkrétním případě, kde bude propojení s Hibernate a Tiles 2. Také popíši, jaké problémy se vyskytli a postup jejich řešení. Nakonec již jen shrnu výsledky práce a zhodnotím, kolik času zabralo naučit se s těmito nástroji pracovat a kolik práce nám ušetřily.

Rezervační systémy

Co to vlastně rezervační systém je? Rezervační systém je obdoba e-shopu, který je zaměřen pouze na produkty. Rezervační systém umožňuje rezervovat především služby, jako jsou masáže, pobyty nebo lety letadlem. Lze také rezervovat například zasedací místnost v práci nebo auto v půjčovně. Tato rezrvace se na první pohled nemusí zdát jako služba, pokud se na ní pohlíží jako rezervování auta či místnosti, ale když je zpoplatněná, tak si vlastně rezervujete službu propůjčení věci. Rezervační systémy jsou všude kolem nás a každý se s nimi již někde setkal. Určitě jste si kupovali letenky přes internet, zamlouvali pobyty na dovolenou nebo nějakou wellness službu (cvičení, vířivku atd.).

Účelem všech rezervačních systémů je vydělat peníze. Pokud je tento systém pro jednoho jediného poskytovatele (ten kdo poskytuje služby), vydělává tím, že umožňuje lepší přístupnost jím poskytovaných služeb. Je-li systém pro více poskytovatelů a zastřešuje ho jeho provozovatel, který si účtuje procentuální část z celkové ceny objednávky za zprostředkování, vydělává z těchto poplatků.

Každý z těchto systémů pracuje stejně. Vy, jako zákazník si vyberete službu a čas poskytnutí, zarezervujete si ji, dostanete nějaké potvrzení o rezervaci služby, a pak přijdete s tímto potvrzením k poskytovateli služby v daný čas a služba je Vám poskytnuta. V ideálním případě by se za rezervaci ani nemuselo platit, ale v ideálním světě samozřejmě nežijeme, a s tím je spojeno několik problémů. Prvním problémem jsou internetoví záškodníci, kteří by mohli buď sami, nebo skrze využití k tomu určených programů, zarezervovat veškeré kapacity poskytovatele, ale nedostavit se k vybrání služby a samozřejmě ji ani nezaplatit. Dalším problémem je nezodpovědnost lidí, kteří

rezervace provádějí, jelikož stačí, aby se třeba špatně vyspali a už se jim na zarezervovanou masáž nechce, tak tam prostě nepřijdou a ani o tom nedají vědět. Nebo rodině nakonec nezbudou peníze či jim onemocní děti dva dny před odjezdem a dovolená se nekoná a zamluvený pobyt propadne. Z pohledu zákazníka je mu to jedno, jelikož ho to nic nestojí a není to žádný problém. Z pohledu poskytovatele služby to je už horší, jelikož pro zákazníka vyhradil část své kapacity a třeba i odmítal jiné zákazníky, kteří si chtěli službu objednat po něm a když zjistí 5 minut po stanoveném čase, že se zákazník nedostaví, tak s tím už nic udělat nemůže. Těmto problémům lze předejít několika způsoby.

Zpoplatnění zálohou

Prvním, nejrozšířenějším způsobem, je rezervaci zpoplatnit procentuální zálohou z celkové ceny rezervovaných služeb. To znamená, že si objednáte pobyt v ceně 4000 korun a záloha bude např. 20% (800 Kč), což si lidé už rozmyslí zda přijedou. Nebo se pokusí kontaktovat poskytovatele a zkusit pobyt přesunout na jiné datum. V tomto případě záškodníky úplně vyřadíme, protože takové částky investovat nebudou. Tato metoda je velmi často používaná na rezervačních portálech, jako například na „hostels.com“ nebo „hostelworld.com“. Záloha se pohybuje kolem 10 % z ceny objednávky, tato částka v těchto případech putuje na účet provozovatele portálu a poskytovatel dostane pouze informaci o tom, kdo a na kdy si jeho službu objednal. Tato metoda je s touto výší zálohy vhodná hlavně pro pobyty, kde zákazníka trochu nutí dodržet svou část dohody, i kdyby ji zákazník nedodržel, může ubytovací zařízení během dne sehnat jiné zákazníky na zbytek plánované doby pobytu. Vhodné je to např. pro velké kempy, které mají stovky ubytovacích míst a pokud se někdo nedostaví, tak mají dostatek zákazníků, kteří tato volná místa mohou zaplnit. Problém by to mohl být pro hotel s několika málo pokoji, tam by mohl být problém narychlo zaplnit danou kapacitu. Nevhodné je to např. pro například pro poskytování masáží, kde by částka z deseti procentové zálohy byla velmi malá a dostala by se pouze např. k provozovateli sytému, ale ne k poskytovateli. V případě těchto služeb je vhodnější zvýšit zálohu na vyšší procento, např. 40 %.V tom případě by bylo lepší zálohu zvýšit např. na 40% z ceny a ta by se v případě nedostavení zákazníka rozdělila mezi poskytovatele a provozovatele. Při tak velké záloze se nedostavení moc neočekává. U záloh, ať malých, nebo velkých, může nastat problém třeba při rezervování jazykových kurzů. V jazykových školách je běžnou praxí to, že na jeden čas vypíšou kurzy od všech jazyků, které nabízejí a čekají, které se nejvíce naplní, ty potom budou učit. Zákazníky, kteří jsou zapsáni v kurzech, které se nakonec neuskuteční, pak zkusí přeřadit do kurzů v jiný čas. Někdy se přeřazení nepovede a zákazník do žádného kurzu nenastoupí, pak by mu měla být vrácena záloha, která byla poslána na účet provozovatele sytému, a ne školy. Proto je třeba u rezervačních

systémů správně navrhnout finanční politiku.

Zaplacení celé částky

Druhou možností, ke které jsem se již přiblížil, je vybrat při rezervaci rovnou celou částku najednou. Tato možnost se v praxi také hojně využívá, ale opět není vhodná na všechny druhy služeb. Kdyby byl například nějaký portál zaměřený na jazykové kurzy pro rozsáhlou oblast, tak si tam zákazník nejprve kurz rezervuje a poté si začne zjišťovat o dané Jazykové škole, jaké má reference, zda za to stojí, nebo ne. Rozhodně ale nebude chtít hned platit celý kurz naráz. Naopak u hotelu se host ubytuje a částku zaplatí najednou. V tu chvíli nemá možnost moc vybírat, jaká je v hotelu kvalita poskytovaných služeb, zbylé informace najde na stránkách hotelu. U služeb, kde jednotlivá služba není drahá, nepohybuje se v řádech tisíců, si zákazník zarezervuje jednu na zkoušku, kterou zaplatí celou. Pro další využívání služeb se rozhodne podle předvedené kvality provedení. V tomto případě, pokud existuje nějaký provozovatel, dostane podíl z vybrané částky.

Bezplatná rezervace

Poslední možnost je vlastně ta první, bez vybírání záloh. Jen se k ní musí přidat starost poskytovatele, aby kontaktoval zákazníka, zda objednávku myslí vážně nebo ne. V praxi to vypadá tak, že provozovatel služby kontaktuje určitou dobu před objednaným časem vykonání služby zákazníka, zda si objednávku nerozmyslel a zda se opravdu dostaví. Musí tak provést v dostatečném časovém předstihu, aby mohl kapacitu zaplnit jiným zákazníkem v případě, že si to ten původní rozmyslel. Takto je to pohodlné pro zákazníka i poskytovatele. Jen se poskytovatel musí opravdu ujistit, že se zákazník nedostaví nebo mu emailem oznámit skutečnost, že z důvodu nekomunikace z jeho strany je rezervace stornována. Pokud by totiž zákazník nakonec dorazil např. do hotelu a hotel pro něj neměl kapacitu a rezervace by byla stále platná, musí mu hotel zajistit náhradní ubytování ve stejné nebo vyšší kvalitě za cenu uvedenou v rezervaci, čímž by hotel rozhodně prodělal. S placením provozovateli takového portálového systému je to tak, že si provozovatel spočítá všechny rezervace, např. pro daný hotel, které byly v jeho systému provedeny. Tyto rezervace pošle poskytovateli služby (v tomto případě hotelu) a ten mu je buď potvrdí nebo ne, podle toho zda zákazníci rezervaci dodrželi nebo ne. Poté provozovatel potvrzené rezervace vyfakturuje. Tomuto systému lze vytknout, že poskytovatel může některé rezervace schválně prohlásit za propadlé. Bude-li to však dělat ve větší míře než ostatní poskytovatelé v daném systému, bude to zřetelné, nebo minimálně podezřelé a provozovatel ho bude moci ze sytému vyřadit a tím poskytovatel ztratí zakázky, což rozhodně není v zájmu žádného podnikatele.

Co se týká cen, můžeme se zaměřit ještě na poměr cen na internetu a přímo u poskytovatele. Obecná politika je taková, že ceny nabízené na internetu jsou stejné nebo nižší, a zákazníci jsou tím nuceni tyto služby využívat. Například u nás v Čechách je možnost zamluvit na internetu kvalitní hotely v Praze za cenu pod tisíc korun na noc, ale když přijedete přímo na hotel ubytují vás za cenu o dost překračující hranici tisíce korun. Obecně se ceny spíše shodují a smlouvy s rezervačními portály většinou obsahují podmínku, že se služby nesmějí na portálu nabízet dráž, než kolik v reálu stojí. Ale existují i případy, kdy si rezervační systém chce jako provizi vzít i 40% a to poskytovateli nezbývá nic jiného, než trochu cenu nadsadit, aby mu pak z ceny zbylo alespoň něco pro sebe.

Stávající řešení

Když zadáte do Googlu „rezervační systém“ najdete jich jistě spoustu. Jako příklad mohu uvést SuperSaaS, Bizzy, Previo nebo BOOKER. Každý z těchto systémů je zaměřen na určitý typ nebo i skupinu služeb. Většinou mají vytvořený určitý kalendář, ke kterému je připravena logika pro rezervování určitých zdrojů, jako například sportovní hřiště, nebo zasedací místnosti na určitou omezenou dobu. Některé mají logiku upravenou i tak, aby bylo možné zamlouvat zdroje i na delší čas, a dají se použít i na rezervaci pokojů v ubytovacím zařízení. Všechny tyto systémy jsou kvůli přístupnosti zákazníkům samozřejmě webové a jsou určeny pro jednotlivé poskytovatele. Žádný z nich není určen pro použití s více poskytovateli a není určen pro specifičtější služby, jako například kurzy. Poskytovatelé těchto služeb musí volit komplikovaná řešení skrze systémy pro kalendářové akce a trochu si je upavit, aby alespoň trochu vyhovovaly jejich potřebám.

Taková je i momentální situace u zadavatele mé práce. Zadavatel provozuje jazykovou školu a má na svých webových stránkách možnost rezervace kurzu online. Toto momentální řešení je založeno na redakčním systém Joomla a extenzi **DT Register.** DT registr v dřívějších verzích byla spojena s další komponentou a to JEvents, což je kalendář pro Joomlu. V tomto kalendáři se dají vytvářet různé události, buď z veřejného rozhraní nebo z administrátorského, podle nastavení práv. DT Register slouží k možné rezervaci těchto akcí. Znamená to, že se musely akce nejdříve vytvořit v kalendáři a pak převést do registračního podsystému. To bylo dost zdlouhavé, ale v nejnovější verzi se dají události vytvářet přímo v DT registru a to práci velmi usnadňuje. Je stále potřeba mít nainstalovanou i druhou extenzi, protože se využívá její tabulka v databázi, ale nemusí se využívat přímo.

V DT registru to funguje tak, že nejdříve vytvořím registrační formulář. Ten se vytváří po

jednotlivých polích. Tato pole lze vytvořit i pro každou událost jiné, pokud je potřeba, ale není to podmínkou. Příklad pole, které se může hodit jen k určité službě, je třeba checkbox pro objednání slovníku angličtiny (nebude se nabízet ke kurzu němčiny). Pak se vytvoří kategorie pro produkty, ty se dají i zanořovat do sebe, ale jen ve dvou úrovních, což je zrovna pro kurzy dost nevyhovující, jak ukáži v následující kapitole. Poslední krok je vytvoření dané aktivity, při jejím vytváření se nastaví od kdy do kdy, pro kolik lidí, článek s popisem, přiřazenou kategorii, název atd.. Důležité je to, že musím všechny informace, které chci sdělit, napsat do názvu, protože zobrazení vyplněných atributů pro tento druh služby není vyhovující. Pro příklad, je kurz jednou týdně v úterý od 11:00 do 12:30 po dobu tří měsíců, ale v DT Registru událost nastavíte, že je od 11:00 1.1.2011 do 12:30 3.4.2011, a to by se Vám nakonec zobrazilo i u popisu kurzu. Z takového popisu by zákazník moc moudrý nebyl, proto se musí ve výpisu zobrazovat jen název a vše do něj napsat. Tento systém prostě není připraven na zadávání kurzů. Dále si při vytváření aktivity vyberete, které položky chcete zobrazit v registračním formuláři pro tuto konkrétní službu, což je výhoda v tom, že můžete každé akci přiřadit něco jiného, ale pokaždé to vyplňovat je práce navíc. Toto zobrazení by se dalo nastavit třeba ke kategorii, kdyby jich bylo možné zanořit více. Další nevýhoda tohoto řešení je i to, že v přehledu nabízených akcí (kurzů) není vidět, kolik je obsazených/volných míst. Při tomto řešení se do systému ani neregistruje a není tedy možný žádný přehled zákazníků.

Výhodou tohoto řešení je relativně nízká cena, která je 75$, což je při současném kurzu asi 1300Kč na pořízení a bez dalších poplatků. Další výhoda je možnost zakomponování do systému Joomla a možnost propojení s jinými sociálními pluginy. Zde by bylo možné systém doplnit o možnost registrace a následného přihlášení uživatelů, ale pokud systém nepodporuje dobrý přehled zákazníků, tak je to v našem případě k ničemu.

V následující kapitole si popíši, jaké služby by měl pokrýt rezervační sytém, aby byl vhodný pro většinu trhu a jakou by měl mít funkčnost, aby vyhovoval jako rezervační sytém a dal se použít i pro správu zákazníků.

Požadavky na systém

Vyvíjený systém bude internetový rezervační systém pro rezervaci vybraných druhů služeb. Tento systém nabídne k rezervaci více typů služeb a v tomto ohledu bude poskytovat jistou univerzálnost. Rezervovat bude možné vždy jen jeden druh z nabízených druhů služeb. Tento typ se vybere při instalaci před nakonfigurováním systému a později se již nebude měnit.

Systém bude mít 3 uživatelské role. Jednak zákazníky, kteří si budou vybrané služby rezervovat. Pak poskytovatele, což jsou firmy poskytující nabízené služby. Třetí rolí budou provozovatelé, to bude v případě, že u jedné ze služeb nebude pouze jeden poskytovatel, ale bude jich víc. Provozovatelem bude osoba, která provozuje server s tímto rezervačním systémem. Pokud bude jen jeden poskytovatel bude zároveň provozovatelem.

Vyvíjený systém umožní rezervaci služeb od jednoho nebo i od více poskytovatelů, kteří poskytují služby stejného typu. V prostředí nabídky služeb od více poskytovatelů bude mít systém vlastní vzhled a postup výběru služby se bude lišit pouze v počtu kroků výběru, např. místo samostatné volby služby si uživatel vybere také místo poskytování, po tomto kroku už se uživatel opět dostane na výběr samotné služby, jako v případě, kdy je pouze jeden poskytovatel. Výsledkem bude zobrazení vybrané služby od více poskytovatelů.

Systém bude samostatná aplikace. Nebude se integrovat do žádných dalších internetových stránek. Bude mít vlastní doménun na které bude provozován. Systém bude umožňovat snadnou úpravu stylu stránek do stejné podoby, jako jsou stránky poskytovatele. To proto, aby si zákazník nemyslel, že se dostal na jiné stránky a následně je raději neopustil. Součástí tohoto vzhledu bude i zachování, alespoň základního menu, z původních stránek. Vzhled se bude upravovat podle vzhledu stránek poskytovatele pouze v případě, že bude poskytovatel jen jeden, jinak bude mít systém vzhled vlastní.

Jak jsem uvedl v předchozí kapitole, systém umožní, aby zákazník platil zálohu za zarezervování služby. To zda se bude záloha za vybranou službu platit, a kolik procent bude činit, bude možné nastavit. Toto nastavení bude pro všechny služby stejné a nebude se lišit u jednotlivých poskytovatelů. Tuto platbu bude možné provést několika způsoby. Způsoby plateb popíši v kapitole „Placení“.

Aby byl systém univerzální a mohl pokrýt poptávku, bude umožňovat rezervování následujících služeb:

- kurzy (jazykové, kuchařské, programátorské, …),
- pobyty (hotel, pension, hostel)
- a hodinové služby (masáže, pneuservis, kadeřnictví, …).

Tyto služby pokrývají většinu služeb na trhu, které mají potenciál k požadavkům rezervování. Systém bude navržen tak, aby se dal snadno doplnit o další typ služby a nebylo potřeba měnit jádro programu.

Systém bude zákazníkům umožňovat nejenom objednat službu bez nutnosti registrace, ale budou se moci do systému také registrovat a přihlásit. Po přihlášení uvidí uživatel historii svých objednávek a bude moci editovat své údaje.

Poskytovatel po přihlášení uvidí všechny své zákazníky, se všemi jejich údaji a historií objednávek. Také uvidí přehled jím nabízených služeb, včetně aktuálních stavů kapacit a historií obsazenosti služeb. Bude moci také jednotlivé objednávky editovat, např. z důvodu vyhovění žádosti zákazníka o přesun poskytnutí služby na jiný termín. Zákazník ovšem tato práva mít nebude.

Systém bude uchovávat všechny informace o historiích objednávek, nabídek a zákaznících daného poskytovatele. Bude tak poskytovateli sloužit jako přehledná databáze jeho služeb a zákazníků.

Bude zde také umožněno hodnocení poskytovaných služeb formou zpětné vazby od zákazníků. Po provedení služby přijde zákazníkovi email s formulářem, kde se formou škálového hodnocení odpoví na několik základních otázek ohledně kvality služby. Po vyplnění a odeslání formuláře, se hodnocení promítne do ohodnocení produktu.

Systém bude pro ukládání dat používat databázi. Měla by být použita taková databáze, aby nebylo potřeba kupovat licence pro její používání, aby zbytečně nenarůstala cena systému. Systém musí být navrhnut tak, aby byl odstíněn od databázové vrstvy a nebyl velký problém změnit databázový stroj, či dokonce typ databáze.

Detailní analýza systému

Vlastnosti služeb

Kurzy

Jako příklad uvedu jeden kurz s jeho vlastnostmi, tato ukázka bude obsahovat velké množství vlastností, které v reálné situaci nastat nemusí, ale mohou:

Kurz obecné angličtiny úrovně B1, 2 vyučovací hodiny jednou týdně v Českých Budějovicích (Jihočeský kraj), pro 8 žáků, hodiny v pondělí od šesti do půl osmé. V termínu 20. 9. 2010 až 10. 1. 2011. Cena 4000 Kč.

Kurz má následující vlastnosti: kraj a město výuky, jazyk, úroveň, vyučovaná slovní zásoba, cena, kapacita, zda je semestrální/intenzivní/ individuální, data od kdy do kdy, ve které dny a v jaký čas se vyučuje.

Jak je vidět, největší část vlastností kurzů by se dala označit jako jeho kategorie, jelikož se tyto vlastnosti můžou většinou opakovat u více kurzů a jiné kurzy se mohou lišit v jedné či více vlastnostech. Ukázka

- **kraj:** Jihočeský
- **město:** České Budějovice
- **jazyk:** angličtina
- **vyučovaná slovní zásoba:** obecná angličtina
- **úroveň:** B1
- **typ kurzu:** semestrální
- **počet hodin:** jednou týdně 2h

Pokud tyto kategorie vhodně zanořím pod sebe, vytvořím z nich strom, kde jako list bude jeden či více kurzů. Pak pouhým přiřazením kurzu do správné koncové kategorie získá kurz většinu svých vlastností.

Zbytek vlastností si pak ponese kurz jako takový. Tzn. kapacita, cena, termín od kdy do kdy, dny v týdnu a časy konání.

Toto rozdělení se bude moci aplikovat na jakýkoli kurz. Budou se lišit jen kategorie a jejich počet.

Pokud bude systém nainstalovaný pouze pro jednoho poskytovatele, odpadnou první dvě kategorie. Kurz může být třeba kuchařský se zaměřením na italskou kuchyni a intenzivní, tzn. měsíc každý pracovní den, ale pořád bude mít svou cenu, kapacitu a termíny od kolika, ve které dny a od kterého do kterého data. Proto bude nutné, aby bylo možné kategorie editovat i se zařazováním kurzů.

Ke kurzům budou nabízeny i doplňkové produkty. To budou produkty nebo služby poskytované jen jednorázově, budou tedy vázány jen na data kurzu. Budou mít svou cenu a bude možné je objednat ve zvoleném množství.

Ve školním prostředí těchto produktů bude celá řada. Mohou to být učebnice, slovníky, sešity a jiné pomůcky k výuce, ale třeba i rozmanité spektrum psacích potřeb. Musí se definovat, ke kterému kurzu se bude smět nabízet který produkt, jelikož ke kurzu obecné němčiny na úrovni B1 by systém neměl nabízet slovník právnické španělštiny a učebnici angličtiny pro úroveň C1. Tyto produkty se budou také řadit do kategorií jako např.

- **typ:** učebnice
- **jazyk:** angličtina
- **vyučovaná slovní zásoba:** obecná angličtina
- **úroveň:** B1

Tyto kategorie a jejich produkty se budou muset při instalaci systému naplnit a během jeho používání se budou moci měnit, mazat a přidávat, tytéž úpravy bude možné provádět i u cen.

Pobyty

Pobyt v hotelu, penzionu či hostelu se vždy vztahuje k pokoji a dni, myšleno 24h, tzn. i s nocí z jednoho dne na druhý. V praxi se tomu říká lůžkoden, i když to neznamená celých 24h, protože hosté mohou nastoupit např. až po 14 h a odjet musí do 11 h, aby se stihli uklidit a připravit pokoje pro další hosty. Cena se v průběhu roku může měnit, i když jde o ten samý pokoj. Další důležitou vlastností je kapacita a vybavení pokoje a pak už jen doplňkové produkty.

Doplňkové produkty v hotelu budou služby poskytované navíc k samotnému pobytu. Budou to služby typu polopenze, plná penze, šampaňské na pokoj, buzení, parkoviště atd. Může existovat také možnost, že se nebudou nabízet všechny služby ke všem pokojům, proto se bude definovat, ke kterým pokojům se bude nabízet která služba.

Tyto služby budou mít opět svou cenu za jednorázové využití a budou se moci objednat v určitém množství. Tzn., že pokud si zákazník ke třem dnům pobytu ve dvoulůžkovém pokoji objedná 1x polopenzi, dostane jí pouze jeden den. Pokud jí chce na celé tři dny, měl by si ji objednat 3x. Tato možnost volby je zde, protože pokud by si zákazník objednal polopenzi a dostal ji automaticky k celému pobytu, tak ji nemusí využít celou z důvodu výletů atd., ale zaplatit by ji musel celou. Musí zde být kontrola, zda si host neobjednal službu vícekrát než je možné, např. v tomto případě 5x.

Hostu bude nabízena polopenze vždy na jeden den, ale pro různý počet lidí, od maximálního počtu osob na pokoji až po jednoho hosta, aby host nemusel objednávat polopenzi po jednom dni pro každou osobu na pokoji zvlášť. Může si tedy k 5ti dnům, na 5ti lůžkovém pokoji objednat 2x polopenzi pro 5osob, jednou pro 3 a jednou pro čtyři. Toto rozdělení bude záležet na definici a přidělení doplňkových služeb poskytovatelem, ale musí mu to být systémem umožněno.

Pokud si host neobjedná službu na celou dobu pobytu, tak si přidělení k jednotlivým dnům musí domluvit s poskytovatelem, který ho včas kontaktuje. Tuto skutečnost systém zajišťovat nebude, jelikož většina hotelů umožňuje hýbat s těmito službami i přímo na místě, kde se host rozhodne podle aktuálních podmínek jako je počasí, nabídka jídel atd. Na konci pobytu se pak finančně vyrovnají. To již ale neřeší systém.

Hodinové služby

Hodinové služby znamená všechno, kde se objednává na určitou hodinu, jsou to např. masáže, fyzioterapeutická centra, pneuservisy atd. Poskytovatelé těchto služeb poskytují určitou skupinu služeb. Poskytovatelé mají určitá pracoviště, což jsou prostory, ve kterých mohou službu poskytovat, masážní místnost, stojan na auta atd. Na každém pracovišti se dají vykonávat některé služby, ale nemusí to být všechny.

Na jednom pracovišti muže být zařízení, které je nutné pro vykonávání určité služby A, ale nemusí být na všech. Naopak na dalším pracovišti může být další zařízení, které umožňuje poskytovat jinou službu B. Ale pro ostatní služby C, D, E není potřeba žádného speciálního zařízení, které by bylo jen na některém pracovišti. To znamená, že pokud má poskytovatel jen dvě pracoviště, tak má v jednu chvíli kapacitu na provádění dvou služeb C, D nebo E najednou nebo jen jedné A a B a nebo jejich různou kombinaci, kde ale nesmí být dvakrát A nebo B. Pokud si zákazník zamluví službu D

na určitou hodinu, tak poskytovatel bude mít v tu dobu volnou kapacitu na poskytnutí pouze jedné služby C nebo E, nebo jedné služby A nebo B, podle toho na které pracoviště jde zákazník, který si zamluvil službu D.

Pro ujasnění uvedu konkrétní příklad. Máme 2 pracoviště, zelené a modré, a 5 služeb, využití rázových vln, využití laseru, masáže zad, masáže nohou a masáže lávovými kameny. Všechny masáže je možné vykonávat na zeleném i modrém pracovišti, ovšem rázová vlna je pouze na zeleném pracovišti a laser je pouze na modrém pracovišti. Poskytovatel může tedy nabídnout v pondělí v 11:00 kombinaci 2 jakýchkoliv z masáží, 1x využití rázových vláken a 1x využití laseru. Ovšem jakmile si zákazník zarezervuje využití laseru, může už poskytovatel nabídnout pouze 1x jakoukoliv masáž, nebo využití rázových vln.

Každá poskytovaná služba zabere určitý čas např. 30 min, 1 hodinu. Pracovní den poskytovatele se bude skládat z časových slotů např. po 30ti min. a jednotlivé služby se do nich budou rezervovat. Služba může zabrat jeden nebo více těchto slotů. Slot je pevný celek, který se nedá dělit, nebude se moci zarezervovat služba na jeden a pul slotu. Tyto sloty budou vyplňovat celou otevírací dobu, na kterou se bude moci zákazník rezervovat. Pracovní doba se může lišit u každého pracoviště, takže ji nebude určovat poskytovatel. Podle pracovních dob jednotlivých pracovišť se budou lišit i možné nabízené služby.

Registrace

Systém umožní zákazníkovi se buď registrovat, nebo objednávat bez registrace. Registrovaný zákazník bude mít tu výhodu, že po přihlášení uvidí přehled svých potvrzených i nepotvrzených objednávek (viz dále), plateb k nim, své údaje a seznam svých osob, tento pojem bude vysvětlen v podkapitole „Objednávání“. V tomto seznamu se bude možné dostat na detail jednotlivých produktů z objednávek.

Pokud se zákazník rozhodne registrovat, musí vyplnit tyto údaje:

- jméno
- příjmení
- email
- heslo
- zopakovat heslo
- ulice
- město
- telefon

- PSČ
- rok narození
- položka, zda chce zákazník dostávat novinky

Všechny tyto údaje budou povinné, protože je velmi důležité, aby mohl poskytovatel některou z cest kontaktovat zákazníka z důvodu potvrzení objednávky, či domluvy posledních detailů, které systém nemůže zachytit.

Položka email bude použita jako přihlašovací jméno. Na tuto emailovou adresu bude odeslán email pro potvrzení registrace. Pokud zákazník registraci nepotvrdí, bude jeho registrace považována za neplatnou a bude se muset registrovat znovu.

Pokud se zákazník rozhodne objednat bez registrace, bude muset vyplnit ty samé údaje jako při registraci, až na heslo a položku, zda chce dostávat novinky. Ale systém o tohoto zákazníka nepřijde, i tak se mu heslo vygeneruje a pošle na mail, s oznámením, že pokud bude v systému chtít příště objednávat, stačí se přihlásit. Pokud se zákazník přihlásí, účet se automaticky aktivuje, jelikož jeho heslo bylo také posláno na mail.

Pokud zákazník někdy zapomene své heslo, bude mít možnost si heslo resetovat a nechat si nové poslat na email.

Nebude možnost nechat si zaslat zapomenuté přihlašovací jméno na email, protože přihlašovacím jménem je emailová adresa, kterou zákazník zapomněl.

Po přihlášení bude mít zákazník oproti normálním funkcím také k dispozici filtrovatelný seznam svých objednávek, plateb, osob a vlastní údaje.

Výběr služby

Systém bude moci obsahovat velké množství nabízených služeb, ale kdyby se zákazníkovi zobrazily všechny najednou, byl by to nepřehledný seznam plný služeb, které zákazník nehledá a stránky by raději opustil.

Proto bude systém umožňovat vybrat si službu podle požadovaných vlastností, nebo alespoň zúžit množství nabídnutých služeb na přehledný seznam. Vyhledávání služby se bude u jednotlivých typů služeb lišit.

U každého typu služby bude možnost podrobnějšího vyhledávání přes všechny možnosti služeb, aby se zákazník mohl rychle dostat ke specifické službě, kterou hledá.

Kurzy

U kurzů se bude zákazník chtít dostat na seznam kurzů, které budou patřit do určité zanořené kategorie. To znamená, že by mu systém měl umožnit vyhledávat podle jednotlivých kategorií. Zákazník by měl mít možnost si postupně vybrat jednu kategorii za druhou. Např. kraj, město, jazyk, slovní zásobu, úroveň a typ. Pak se dostane např. na seznam semestrálních kurzů obecné němčiny v Českých Budějovicích, které se budou lišit jen v počtu hodin týdně a jejich časech.

Systém musí zajistit postupné načítání zanořených kategorií pro další krok výběru. Dále musí zajistit, aby nevznikl problém, když by určité kategorie byly více zanořené než jiné. Rozdíl mezi vybíráním u více poskytovatelů a jedním bude jen v počtu zanořených kategorií. U více poskytovatelů bude navíc např. ještě kraj a město.

Po vybrání požadovaných kategorií se zobrazí přehledný seznam kurzů s jeho vlastnostmi - počet hodin týdně, dny výuky i s časy a cena kurzu. Přes tento výpis se bude moci uživatel dostat na detail kurzu s jeho podrobnějším popisem, možností objednání apod. Systém bude zobrazovat jen kurzy, u kterých je ještě volná kapacita. Kurzy bude nabízet, i když již začaly, ale je na nich stále volno. Nebudou se nabízet kurzy, které již skončily.

Pokud by kategorií bylo příliš, systém umožní rychlé vyhledání např. přes 2-3 kategorie, pak zobrazí delší seznam s možností filtrace podle ostatních vlastností.

Pobyty

Na většině serverů pro rezervování pobytů se vybírá podle několika kritérií. Je to čas příjezdu, čas odjezdu nebo eventuálně počet dní, země, město a počet osob, nebo přímo počet pokojů s počtem lůžek. Když se vybírají přímo pokoje, lze zadat počet pokojů a jejich typ, ale nelze zkombinovat více druhů pokojů, což není dobré řešení. Pokud se zadá počet osob a jede např. skupina tří párů, tj.

6 lidí, musí být ošetřeno, jaké pokoje se mají nabídnout. Ideální řešení bude vybírat bez osob a pokojů a na další obrazovce zobrazit seznam všech nabízených pokojů, které mají na zvolený termín volno, a uživatel si sám vybere, kolik kterých pokojů chce objednat.

Toto řešení minimalizuje změnu vyhledávání mezi více poskytovateli a jedním. Aby se dalo vybrat mezi více poskytovateli, přidá se k datu kraj a město, což budou opět kategorie produktu a než se zobrazí seznam pokojů nabízených jedním poskytovatelem, vloží se obrazovka se seznamem nabízených hotelů (poskytovatelů), a až proklikem na položku poskytovatele se zobrazí seznam pokojů, které se nabízejí. Pokud bude poskytovatel jen jeden, nezobrazí se výběr kategorií a přeskočí se obrazovka se seznamem poskytovatelů.

Seznam nabízených pokojů se bude skládat z popisu hotelu, nebo aspoň polohy a fotky hotelu. Pak na této stránce bude zobrazen termín, pro který se hledá. Pod těmito informacemi budou v řádcích jednotlivé pokoje s cenou za noc a s možností výběru počtu pokojů. Také zde bude výsledná cena vybraných pokojů, za celý požadovaný čas, a tlačítko objednat.

Hodinové služby

U hodinových služeb je problém s výběrem podle času v jejich vlastnostech. Na určitou hodinu by se systém mohl tvářit, že má poskytovatel volno, ale když přejdete k výběru služby, zjistíte, že nabídka služeb je dost omezená z toho důvodu, že je již plné pracoviště s vybavením na určité služby. Tuto skutečnost by zákazník neměl jak zjistit a mohl by si myslet, že služby nejsou v nabídce vůbec. Musel by se přepínat mezi různými časy, aby zjistil, kdy má volno na jím požadovanou službu.

Jednou z možností by bylo zobrazovat na jeden čas vedle sebe všechny služby, které jsou k dispozici. Tato možnost ale skrývá nebezpečí toho, že služeb bude moc a zobrazení se nevejde na stránku a zobrazení volných časů, např. na týden, by nepřicházelo v úvahu vůbec.

Řešením bude, nejdříve vybrat požadovanou službu, a až potom zobrazit časový rozvrh, kdy je volno a kdy není. Zde se může vyskytnout problém v tom, že systém přiřadí zákazníka do pracoviště se specializovaným zařízením pro službu, která ho nebude vyžadovat a tím se tato služba bude na tento čas zbytečně blokovat, i když by poskytovateli mohla přinést větší zisk. Tuto skutečnost musí systém řešit, ale půjde to pouze za předpokladu, že alespoň jedno z pracovišť nebude obsahovat specializovaný přístroj. Pokud by bylo každé pracoviště specializované na

některou službu, pak by rozhodování o pracovišti nemělo smysl.

Pokud se bude vybírat mezi více poskytovateli, tak si zákazník nejdříve vybere dle kategorií (opět nejspíš kraj a město). Pak si vybere jednu ze služeb, která do této kategorie patří. Tyto služby by se měly zobrazit v přehledném kompaktním přehledu s názvem vybrané služby, velmi stručným popisem a cenou. Vybráním položky služby se zákazník dostane na její detail, kde bude název, delší popis a kalendář s volnými termíny pro tuto službu, vybráním volného termínu se přidá služba na tento termín do aktuální objednávky.

Pokud bude systém pouze pro jednoho poskytovatele, tak odpadne vybírání podle kategorií a zobrazí se rovnou seznam služeb, kde budou podmínky stejné.

Seznam služeb by mohl být na jednu stránku příliš dlouhý, proto se služby rozdělí také do kategorií, podle kterých je bude možné filtrovat. Jedno filtrování by mělo proběhnout ještě před vstupem na seznam (u více poskytovatelů se pouze dodá další výběr), např. na hlavní stránce pod úvodním textem. Tato možnost filtrování by měla být i přímo na stránce se seznamem.

Objednání

Při výběru služby k rezervaci, si ji zákazník přidá na objednávku a dále může vybírat další služby. Rezervace po jednotlivých službách, by pro zákazníka nebyla pohodlná. Systém bude umožňovat objednat více služeb na jednu objednávku. Místo pro uložení služeb k rezervaci se nebude jmenovat „Košík“ jako v normálních e-shopech, ale „Aktuální objednávka“ nebo „Vaše aktuální objednávka“ podle místa na stránce.

Při objednávání nepřihlášeného uživatele bude vyžadováno pro ověření objednávky opsání kódu z obrázku, aby nemohli objednávat roboti. U přihlášeného zákazníka, tato kontrola nebude vyžadována. Před potvrzením objednávky bude možnost zadat kód ze slevového kupónu. Tato sleva se pak promítne do konečné ceny objednávky.

Objednávky se budou dělit na potvrzené a nepotvrzené. Pouze potvrzené objednávky se budou promítat do kapacit služeb. Objednávka bude potvrzená automaticky, pokud zákazník zaplatí zálohu za objednávku, anebo ji může potvrdit poskytovatel po kontaktu se zákazníkem a potvrzení jeho úmyslů. Stejně tak bude moci poskytovatel objednávku zrušit, pokud zákazník, i přes zaplacení zálohy, nebude s poskytovatelem komunikovat. Je v zájmu poskytovatele kontaktovat každého svého zákazníka, toto systém již řešit nemůže.

U každého typu služby budou požadovány jiné údaje na objednávce. Jelikož poskytovatel bude potřebovat údaje specifické pro daný typ služby. Musí se klást velký důraz na to, že službu může objednat jedna osoba, ale využít jí nakonec může osoba úplně jiná.

Kurzy:

- údaje o kurzu / kurzech
- údaje o zákazníkovi
- údaje o osobách navštěvujících kurzy
- poznámka
- přídavné produkty
- ceny všech položek
- slevy
- celková cena
- zaplacená částka
- zbývá doplatit

Pobyty

- od kd
- do kdy
- pokoje
- počet osob
- další objednané služby
- údaje o zákazníkovi
- údaje o osobách, které pobyt využijí - jedna osoba na pokoj
- jazyk
- poznámka
- ceny všech položek
- slevy
- celková cena
- zaplacená částka
- zbývá doplatit

Hodinové služby:

- seznam služeb
- data služeb
- časy služeb
- pracoviště
- údaje o zákazníkovi
- údaje o osobách, které služby využijí
- poznámka
- ceny všech položek

- slevy
- celková cena
- zaplacená částka
- zbývá doplatit

Při srovnání těchto údajů u jednotlivých typů služeb zjistíme, že je to vždy stejný seznam. Údaje o objednaných službách, údaje o zákazníkovi, údaje o osobách využívajících službu, poznámka a údaje týkající se cen, slev, celkové ceny a zaplacené částky. Jen u pobytů je navíc jazyk, ve kterém budou zákazníci komunikovat.

Z těchto údajů již všechny máme k dispozici, jen chybí někde uchovávat informace o osobách využívajících služby. Tato osoba bude buď sám zákazník, nebo i osoba jiná. Jde o případy, kdy rodiče objednají kurz pro své děti, nebo šéf pobyt pro své zaměstnance.

Údaje o těchto osobách budou stejné jako informace o zákazníkovi jméno, telefon, email, adresa, jen nebudou mít heslo. Aby zákazník nemusel tyto informace vyplňovat vždy znova, bude si moci definovat seznam „svých osob“, ze kterého si bude moci vybrat osobu a přiřadit ji k vybrané službě či službám. Tento seznam zákazník uvidí po přihlášení a bude ho moci i editovat. Pokud při objednání nebude mít zákazník požadovanou osobu ve svém seznamu, jednoduše si ji tam přidá. Pokud bude objednávat nepřihlášený zákazník, seznam osob se mu uloží k jeho vygenerovanému účtu.

Osoby se budou ke službám přiřazovat tak, že ke každé službě bude jedna osoba. U kurzu je to jasné, protože u každého kurzu bude osoba, která ho bude navštěvovat. U hodinové služby je to obdobné, ke každé službě musí být osoba, která tuto službu využije. U pobytů by měla být jedna osoba ke každému pokoji, aby se snadno jednalo při ubytování a předávání klíčů.

Objednávka se bude skládat z několika kroků. První bude výběr všech služeb k rezervaci, druhý bude vstup na aktuální objednávku a přiřazení osob k rezervovaným službám. Pak zákazník tyto služby musí potvrdit a vybrat způsob platby. Pokud nebude přihlášený, tak ho systém vyzve k přihlášení nebo vyplnění údajů. Další krok se bude odvíjet od zvoleného způsobu platby. Poslední krok bude již jen zobrazení údajů proběhlé objednávky.

Po objednání bude zákazníkovi doručen email s informacemi o rezervaci. Pokud nebude objednávka potvrzena, bude v emailu také upozornění, že rezervace zatím není závazná. Po pozdějším potvrzení objednávky, je nutné tuto skutečnost oznámit zákazníkovi, zasláním emailu

s potvrzením. Email s informacemi o rezervaci přijde zákazníkovi po každé editaci jeho rezervace poskytovatelem.

Pokud objednávka nebude potvrzena do určité doby, bude automaticky zrušena. Systém bude evidovat i tyto objednávky.

Placení

Systém bude umožňovat platit zálohu několika způsoby. Jedním ze způsobů bude platba kartou přes platební rozhraní. Druhým způsobem bude platba přes účet. Poslední alternativou bude platba na místě. Ne vždy musí být povoleny všechny typy plateb v systému.

Platba kartou je jednoduché, rychlé a bezpečné řešení. Systém bude mít hned potvrzení o zaplacení, zákazník bude mít ihned potvrzenou objednávku a vše je vyřešené.

Platba přes účet je komplikovanější v tom, že trvá určitou dobu, než dorazí peníze poskytovateli nebo provozovateli na účet. Po tuto dobu nebude objednávka potvrzena. Systém nebude mít přístup k údajům z účtu, kam mají peníze přijít, proto musí tuto platbu potvrdit provozovatel ručně. Po potvrzení objednávky se promítne do kapacity produktu. Pokud mezi tím někdo jiný zaplnil potřebnou kapacitu, poskytovatel si musí domluvit se zákazníkem změnu termínu vykonání služby.

Platba přes účet bude povolena pouze určitou dobu před provedením služby, jelikož by se nemusela stihnout provést. Systém pro tyto platby musí vygenerovat a uložit potřebné údaje a zaslat je zákazníkovi na email.

Poslední alternativou je platba na místě. Ta je podobná platbě na účet, jen se musí peníze složit u poskytovatele nebo provozovatele na kontaktní adrese. Pro tuto platbu bude stanovena také lhůta, do kdy se musí provést. Tato lhůta bude kratší, než pro platbu na účet, např. 48h. Při složení peněz, musí poskytovatel nebo provozovatel tuto rezervaci potvrdit.

Systém bude pro poskytovatele evidovat i platby ostatní, aby o nich měl přehled. Když je zaplacena záloha, zbytek částky se uloží jako platba k zaplacení v nastavené lhůtě. Pokud tuto platbu zákazník do stanovené lhůty nezaplatí, systém na tuto skutečnost poskytovatele upozorní. Zaplacení platby bude v systému zaznamenáno ručně poskytovatelem. Tato funkce slouží pouze pro přehlednost zaplacených a nezaplacených pohledávek.

Platbu bude možno rozdělit na několik splátek. Toto rozdělení však nesmí udělat sám zákazník, ale musí to udělat poskytovatel. Ten nastaví, na kolik splátek se má částka rozdělit a systém platbu automaticky rozdělí. Platba rozdělená do splátek se bude automatiky rozdělovat do rovnoměrných částek. Pokud bude částka po rozdělení obsahovat i necelé koruny, tak se automaticky zaokrouhlí nahoru na celé. Poskytovatel pak jen nastaví data splatnosti.

K objednávce bude možnost přidat slevu. Sleva se bude moci přidat vložením slevového kódu při objednávání. To znamená, že se rovnou započítá do celkové ceny objednávky. Druhou možností je, že slevu přidá poskytovatel k objednávce manuálně. V případě, že platba není rozdělena na splátky, je jen jedna splátka, částka se od této splátky odečte. Pokud je více splátek, sleva se odečte od poslední. Když je splátka menší než sleva, tak se poslední splátka nastaví na částku 0, tedy jako splacená, a zbytek se odečte od předchozí splátky. Objednávka si bude evidovat i svojí cenu před slevou.

Jelikož se bude muset rozdělení platby do splátek i přidání slevy dělat ručně, bude se editování objednávky dělat v několika krocích, tak aby přidání slevy bylo před rozdělením do splátek. Pokud přesto bude třeba splátky editovat, bude se moci měnit už pouze datum splatnosti, ale ne počet splátek.

Systém bude u záloh, pokud budou placeny kartou, evidovat variabilní symbol, čas, číslo účtu, číslo karty atd. U záloh neplacených kartou a ostatních plateb bude evidovat pouze částku, datum splatnosti, datum splacení a způsob splacení. Při placení zálohy přes kartu, přijde zákazníkovi na mail faktura za tuto platbu, pokud bude záloha placena jinak, fakturu obstarává poskytovatel nebo provozovatel. S těmito platbami nemá systém nic společného.

Pokud platbu zákazník nezaplatí, a aby systém stále neupozorňoval na nesplacenou platbu, bude moci být nastavena na nedostupnou.

Aby poskytovatel nemusel slevu pokaždé vyplňovat, bude mít k dispozici svůj seznam slev, které bude moci po přihlášení editovat. Při přidávání slevy k objednávce bude moci vybrat ze seznamu slev, nebo vytvořit slevu novou. Sleva bude vždy buď v procentech, nebo v penězích, bude mít své jméno, popis (důvod slevy) a hodnotu slevy. Při slevě v procentech hodnota nesmí překročit 100 %.

Všechny lhůty pro platby budou v systému nastavitelné a musí být uvedeny i v obchodních podmínkách.

Poskytovatel

Poskytovatel po přihlášení bude moci spravovat vše, co je pro jeho funkci potřebné. Poskytovatel bude vždy moci spravovat, své údaje, kategorie přídavných produktů, přídavné produkty, své slevy, CRM statusy a kategorie svých služeb. Bude zde mít přehled objednávek svých služeb, plateb a svých zákazníků. Samozřejmě v systému uvidí i své služby samotné, jen se to bude měnit podle typu služeb.

Seznam služeb

Kurzy

Seznam kurzů bude jednoduchý, poskytovatel bude mít seznam aktuálních kurzů, kde budou kurzy, které ještě neproběhly, nebo právě probíhají a archív kurzů, kde budou kurzy, které již proběhly. Seznam bude zajisté velmi dlouhý, proto by u něj měla být možnost filtrace přes různé atributy, jako jsou kategorie kurzu, vlastnosti kurzu a zda je na něm ještě volná kapacita.

U jedné položky kurzu bude jeho název, kategorie, cena a kapacita. Po výběru se dostane poskytovatel na jeho detail, kde bude popis a výpis přihlášených zákazníků, ať potvrzených nebo nepotvrzených. Z výpisu jednotlivých zákazníků se bude moci poskytovatel dostat na detail, zákazníka nebo detail objednávky.

Seznam objednávek bude filtrovatelný podle vlastností kurzů, data, jména zákazníka a podle statusu zaplacená/nezaplacená. Nad seznamem bude jen několik filtrovacích kritérií, ale bude zde možnost podrobného vyhledávání podle všech kritérií.

Seznam objednávek bude rozdělen do dvou seznamů, na seznam aktuálních, to budou ty, u kterých služby ještě neproběhly, nebo teprve probíhají, a archív, to budou ty, kde služby již proběhly.

Pobyty

U pobytů, bude seznam nadefinovaných pokojů. U položky pokoje bude možnost pokoj editovat, kde se zobrazí rozvrh na měsíc nebo dva s aktuálním měsícem. Mezi měsíci bude možnost přepínat nebo si rovnou nějaký vybrat. K výběru bude měsíc i s rokem. V kalendáři bude u každého dne

znak pro cenu, a zda je pokoj volný, obsazený nebo neaktivní. Cena nebude vypsaná, to by zabíralo moc místa, bude barevně označena a pod kalendářem bude legenda vysvětlující cenu. Stejně bude určeno, i zda je pokoj plný, volny, neaktivní. Pokud bude pokoj na určitý den plný, výběrem tohoto dne se poskytovatel dostane na objednávku.

Pod kalendářem bude editace ceny pro daný jeden pokoj (viz dále), editace vypnutí/ zapnutí pokoje pro různá data a editace údajů o pokoji jako číslo, kapacita, kategorie atd.

Dále bude mít poskytovatel možnost zobrazit si celkový přehled obsazenosti pokojů. To bude tabulka, kde budou na ose x jednotlivé pokoje a na ose y data. V každém poli pro den a pokoj bude zobrazeno, zda je volný, obsazený, neaktivní a cena. Obojí bude barevně jako v minulém případě a výběru obsazeného pokoje, se poskytovatel dostane na objednávku.

Další funkcí pro poskytovatele pobytů bude změna ceny pokojů. U pokoje bude definována základní cena, se kterou se bude u pokoje dopředu generovat přiřazení ke dnům, pokud nebude určeno jinak. Přiřazení ke dnům se bude poprvé generovat automaticky na určitou dobu dopředu, např. 8 měsíců. Každé další generování bude probíhat na měsíc se základní cenou, opět pokud nebude stanoveno jinak.

Jiné stanovení ceny se může stanovit právě v záložce k tomu určené. Zde bude výpis aktuálních pravidel. Jedno pravidlo bude pro kategorii pokojů, které budou např. dvoulůžkové. Bude zde nastavena cena a datum od - do. První se nastaví perioda opakování pravidla, kde bude týden, měsíc a rok. Podle periody se zobrazí data, od kdy do kdy bude pravidlo platit pro týdny, dny v týdnu, pro měsíce, den v měsíci a pro rok, měsíc a den. Dále se musí vyplnit u pravidla od kdy do kdy platí. Poslední se samozřejmě vyplní cena. Při přigenerovávání nového měsíce (v našem případě 8 měsíců dopředu) se zkontroluje, zda neplatí na toto datum nějaké pravidlo a pokud ne, tak se další měsíc přigeneruje se základní cenou. Pokud se bude měnit cena u jednoho pokoje, bude mít toto pravidlo přednost před pravidlem pro kategorii. Při editaci ceny na detailu pokoje, se bude měnit cena pouze u jednoho pokoje a neovlivní další pokoje v kategorii. Při změně ceny na detailu pokoje, bude postup stejný, jen se nebude vybírat kategorie.

Další záložkou bude záložka pro deaktivaci pokojů. To bude probíhat úplně stejným způsobem jako pravidla pro cenu. Také se vybere kategorie, perioda pravidla (pravidlo se bude opakovat např. 1x za měsíc), vyplnění dat akorát místo ceny se bude vyplňovat aktivní/ neaktivní. Na záložce bude také výpis aktuálních pravidel. Tyto pravidla se budou moci v seznamu mazat, ale ne měnit. Změna

pravidla bude probíhat tak, že ho provozovatel smaže a vytvoří si nové, protože by stejně měnil většinu údajů. Stejně tak to bude i u změny cen. Pravidlo by ale nemělo zasáhnout do již zamluvených pobytů. Pokud tato skutečnost nastane, cena nebo aktivace pokoje se nezmění a poskytovatel na tuto skutečnost bude upozorněn. Sám si pak musí vyjednat se zákazníkem změnu.

Při vytvoření pravidla se projdou všechny již vygenerované dny podléhající pravidlu a upraví se podle pravidla. U pravidla bude ještě tlačítko „Zkontrolovat“ a při potvrzení se provede stejná akce, jako při vytvoření. Bude zde pro případ vyřešení konfliktu pravidla s již zamluveným pobytem. Pravidla bude možné vytvořit i pro ještě nevygenerované dny, v těchto případech nebude možnost konfliktu.

Poslední záložkou systému budou seznamy objednávek. Ty budou znovu filtrovatelné podle čísla pokoje, podle jeho kategorie, data vystavení a data pobytu atd. Pro filtrování platí stejná pravidla jako u kurzů.

Hodinové služby

U hodinových služeb bude seznam, který bude u jednotlivých služeb zobrazovat název kategorie, délku služby a cenu. Po označení jedné služby se dostane poskytovatel na její detail, kde navíc bude i popis a bude zde možnost editování jednotlivých údajů.

Dál zde bude seznam pracovišť, ve kterém se bude zobrazovat jen název, z této položky bude možnost se dostat na detail nebo na rozvrh. V detailu bude jméno, popis, seznam přiřazených služeb a týdenní výpis pracovních hodin. Bude možné tyto údaje editovat.

V rozvrhu bude zobrazen rozvrh na aktuální týden a na týden následující. Zobrazení bude formou tabulky, kde na ose y budou dny v týdnu a na ose x budou časové sloty. Bude zde zobrazeno, zda je slot volný nebo ne. Pokud slot nebude volný, zobrazí se zde jméno zákazníka, který má být obsloužen. Po označní jména se poskytovatel dostane na objednávku služby na tento čas. Seznam se bude moci posouvat na další týdny.

Je jasné, že tyto hodiny nebudou moci být vždy dodrženy. Přijdou do toho dovolené, státní svátky, nemoci. Proto na každém volném časovém slotu bude možnost ho deaktivovat, a aby pro vypnutí celého dne nemusel poskytovatel klikat na hodně slotů, bude u každého dne možnost ho celý vypnout. Pro vypínání bude platit to samé co pro pobyty. Pokud slot již nebude volný, nepůjde

vypnout a poskytovatel si se zákazníkem musí domluvit přesun výkonu služby, až poté ho lze deaktivovat.

Časové sloty se budou generovat automaticky v určité době dopředu, po týdnech, pro všechny pracoviště, podle jejich pracovních hodin a nastavené délky slotu. Tuto dobu bude možné si nastavit, nebude tak dlouhá, např. jeden měsíc, aby bylo možno dělit ceny na sezónní a nesezónní. To je další důvod, proč je cena přímo u služby, a zdražení se promítne ihned do všech nově vytvořených objednávek. Objednávek, které byly potvrzeny před touto změnou, se změna nijak nedotkne. Povede to sice k situaci, že zákazníci objednáni na ten samí den mohou platit za služby rozdílné ceny, ale o tom se těžko dozvědí, a když ano, tak jim to poskytovatel snadno vysvětlí, že ten kdo platí míň, si stihl udělat rezervaci před zdražením.

Další funkce pro poskytovatele

Objednání za zákazníka

Každý poskytovatel bude mít možnost objednat za zákazníka. Tato možnost je zde, aby mohl službu prodat zákazníkovi na místě v centru poskytování služby. Jednoduše se přijde zákazník objednat přímo k poskytovateli a on za něj vyplní všechny údaje a objedná za něj a zákazník může služby zaplatit rovnou na místě.

Poskytovatel takto může objednat za zákazníka stávajícího, nebo úplně nového, s tím že vyplní indicie a vytvoří mu tak účet. Problém může být, pokud bude zákazník starší a nebude mít email. Pro tuto variantu zde bude možnost vytvořit zákazníka interního, který bude mít login a ne email. Pak bude mít účet pouze u poskytovatele v jeho centru a nedostane se k němu přes internet, ale lidé co nemají email, nemají ani internet, takže to ničemu nevadí. Tato situace většinou nenastane, bude to spíše výjimečně, ale pokryje to možné mezery.

Poskytovatel bude mít možnost objednat stávajícímu zákazníkovi, vytvořit nového webového zákazníka, nebo nového interního zákazníka a těm objednat. Poskytovatel bude moci objednat jakémukoli zákazníkovi v systému, kterého může vyhledat podle jména nebo emailu či jiných vlastností. Poskytovatel, mimo údajů o zákazníkovi a objednávek služeb přímo u něj, o zákazníkovi nezjistí nic víc. To znamená, že pokud u daného poskytovatele nebude mít žádné rezervace, tak

uvidí pouze kontaktní údaje. Nemůže zjišťovat, kdo kde a kdy používá jaké služby. Nebude moci ani přímo prohlížet jeho osoby, ale při objednávání v nich bude moci vyhledávat a přiřazovat, ale pouze pokud bude objednávat on pro tohoto zákazníka.

Při takovémto objednání poskytovatel rovnou zákazníkovi objednávku vytiskne, samozřejmě mu ji systém pošle také na email. Pokud se bude jednat o nově vytvořeného webového zákazníka, systém mu pošle na mail žádost o potvrzení, až se zákazník přihlásí na email a potvrdí email, přijdou mu na něj přihlašovací údaje a údaje o objednávce. Pokud zadal email špatně, má alespoň vytištěnou rezervaci v ruce a při příští návštěvě může s poskytovatelem změnit email. Poskytovatel bude moci editovat nepotvrzené zákazníky, které sám vytvořil.

Seznam zákazníků

Poskytovatel bude mít k dispozici i seznam svých zákazníků. V tomto seznamu nebudou všichni zákazníci systému, ale jen ti, kteří již nakupovali od daného poskytovatele. Pokud to bude jediný poskytovatel v systému, uvidí všechny zákazníky. Tento seznam bude obsahovat jméno, kontaktní údaje a celkovou částku utracenou u daného poskytovatele, což bude součet cen všech objednávek zákazníka u tohoto poskytovatele. Seznam bude filtrovatelný podle jména, roku narození, utracené částky nebo místa bydliště.

Bude zde i možnost podrobného vyhledávání podle všech parametrů zákazníka i jeho čísla, podle parametrů jeho objednávek, jako datum poskytnutí služby, pořízené přídavné služby a také další atributy. To bude záležet na druhu poskytované služby, jazyk a jeho úroveň atd. Vyhledání zákazníka bude moci být rozděleno na více druhů vyhledání podle oblasti hledání, např. vlastnosti zákazníka, objednávek, plateb, služeb a přídavných produktů.

Po vybrání zákazníka se dostane na jeho detail, kde budou jeho kontaktní údaje, celková utracená částka a seznam jeho objednávek od daného poskytovatele i s možností filtrování. Po vybrání objednávky se poskytovatel dostane na její detail.

Editace objednávky

Když se dostane poskytovatel na detail objednávky, bude zde moci objednávku editovat. Je to z

důvodu, kdyby zavolal zákazník, že onemocněl a službu nebude moci využít a potřeboval by objednávku změnit na jiný volný termín. Nebo když zjistí, že přecenil svou úroveň v jazyku a bude jí chtít změnit.

Poskytovatel se bude moci přepnout do editovacího módu, kde bude moci u objednávky změnit službu nebo čas, v případě kurzů bude měnit jenom službu (kurz). Bude se moci editovat, každá položka na objednávce. Pokud ji lze editovat, naběhne klasický výběr služby pro zákazníky, vybere se nová položka a přidá na objednávku místo stávající. Přepočítá se cena a upraví podle toho objednávka. Pokud se změní cena, tak se změní velikost potřebné platby k doplacení služeb. Pokud bude cena rozdělená do splátek, rozdíl ceny se projeví ve výši poslední splátky. To i v případě, že bude splátka již zaplacená a případný doplatek si musí vyřešit poskytovatel sám. Tyto výjimečné situace nebude systém více řešit. Pokud bude objednávka již potvrzená, změna se promítne do kapacit služeb, pokud ne nic se nezmění, protože nepotvrzené objednávky se nepromítají do kapacit. Takto bude umožněn i přestup zákazníka například z kurzu do kurzu již v probíhajícím kurzu, z důvodu nestíhání učiva. Systém bude zobrazovat probíhající kurzy, na kterých je ještě volno, proto nebude problém ho najít a přiřadit k objednávce.

Služby bude možné z objednávky i rušit. Například pro případ nevyužití jedné polopenze u pokoje, kde se pobyt bude platit až na konci pobytu. Změna ceny se opět projeví změnou velikosti potřebné platby k doplacení služeb. Pokud bude cena rozdělena do splátek, rozdíl ceny se opět projeví ve výši poslední splátky. To i v případě, že bude splátka již zaplacená, případný doplatek si musí vyřešit poskytovatel sám. Tyto výjimečné situace nebude systém více řešit.

Objednávku bude možno také celou zrušit, i když již byla celá potvrzená, jelikož si zákazník může svoji objednávku rozmyslet a poskytovatel by tím zbytečně postrádal danou kapacitu. Je na poskytovateli, aby sám kontaktoval zákazníky, zda objednávku dodrží. Systém bude editovat i zrušené objednávky.

Jednou zrušenou objednávku bude moci poskytovatel znovu obnovit, např. když zákazník pozdě zaplatí zálohu.

CRM statusy

Jednotliví zákazníci budou moci získat od poskytovatele tzv. CRM status označující velikost slevy při nákupu. Tato sleva je odměnou poskytovatele, za to že u něj zákazník za určitou dobu utratí určitý obnos peněz. Tento obnos bude součet cen všech objednávek po slevách u jednoho poskytovatele za určité období. Toto období bude buď plovoucí, nebo neplovoucí.

Plovoucí znamená, že se součet bude brát za období ode dneška zpět, nebude mít tedy pevné datum začátku a konce. Neplovoucí znamená, že to bude období od přihlášení uživatele dopředu a po jeho vypršení se začne součet nanovo. Zda uživateli náleží některý CRM status, bude automaticky kontrolovat systém při objednávání služby. Pokud ano, tak rovnou přiřadí slevu od statusu k objednávce. Sleva se tedy projeví na konečné ceně. V určitých intervalech se bude kontrolovat, zda statusy nevypršely. U neplovoucího bude pro využití ještě určitá doba po skončení období. U plovoucího nebude žádná doba na vybrání po skončení, protože tam doba neskončí, jen se při objednávání systém podívá zpět na objednávky za definovanou dobu a pokud je tam požadovaná útrata, tak bude mít nárok na danou slevu.

Pokud zákazník tohoto statusu dosáhne, přijde mu email s oznámením této skutečností. Pokud naopak jeho trvání vyprší, systém ho o této skutečnosti také informuje.

Poskytovatel bude mít k dispozici svůj seznam těchto statusů. Bude je moci i vytvářet, mazat a editovat a přiřazovat k nim procentuální slevy ze svého seznamu slev. Pokud změní poskytovatel tyto statusy a tedy podmínky jejich získání musí o tom informovat zákazníky emailem. Tuto skutečnost, ale musí provést poskytovatel sám, tuto záležitost systém v této verzi řešit nebude.

Provozovatel

Provozovatel je osoba, která provozuje a nastavuje systém. Po přihlášení bude mít k dispozici seznam poskytovatelů, objednávek, zákazníků a nastavení systému.

Seznam poskytovatelů bude obsahovat všechny poskytovatele v systému. Po vybrání položky poskytovatele se dostane provozovatel na jeho detail. U každého poskytovatele bude možnost editace, nebo vypnutí. Editovat se budou moci všechny jeho údaje pro případ, že si provozovatel

všimne, že je poskytovatel vyplnil špatně, nebo do nich vyplnil něco, co neměl. Každý poskytovatel bude mít své jméno, adresu, telefon, email, popis. Do jeho popisu se uvede krátký popis společnosti a také doplňující informace, jako např. jaké nabízí slevy při množstevním nákupu (CRM statusy), a ostatní doplňující informace specifické pro daného poskytovatele, včetně čísla účtu a ostatních platebních údajů.

Poskytovatel jako takový je jedna společnost, ale ta bude mít v systému více uživatelů. Tito uživatelé se do systému nebudou sami registrovat, ale bude je zadávat do systému provozovatel. Provozovatel bude také jen jedna společnost, ale bude mít také více uživatelů. Tyto uživatele bude do systému zadávat společnost prodávající systém. Tito uživatelé se budou zadávat přímo do databáze a nebude žádné další rozhraní, přes které by se uživatelé zadávali. Uživatelé, na rozdíl od provozovatele a poskytovatele budou mít pouze email, jméno, příjmení a heslo. Adresu ani další kontakty být nemusí, jelikož se s nimi samotnými nikdy jednat nebude, ale pouze s jednatelem společnosti poskytovatele, nebo provozovatele.

Pod popisem poskytovatele bude editovatelný seznam jeho uživatelů. Provozovatel bude moci uživatele mazat, editovat, přidávat. Bude zde i možnost dostat se na přehled poskytovatele jako má poskytovatel sám. U kurzů to je seznam kurzů s kapacitami, u pobytů tabulka s pokoji a dny a u služeb tabulka s dny, hodinami a časovými sloty.

Seznam objednávek bude obsahovat všechny objednávky ze systému. Bude zde opět možnost filtrace i podle poskytovatele a data vykonání služby či podrobné vyhledání. Tento seznam je pro provozovatele nejdůležitější, protože si vyfiltruje objednávky od jednoho poskytovatele pro jeden měsíc a ty mu vyfakturuje.

Seznam zákazníků bude obsahovat všechny zákazníky systému. Seznam bude opět filtrovatelný a bude zde také možnost podrobného vyhledávání. Po výběru na položky zákazníka se dostane provozovatel na stejný detail jako poskytovatel, akorát zde budou zobrazeny všechny objednávky zákazníka.

Další položka bude nastavení systému, zde budou všechny položky nastavení potřebné k běhu systému, jako např. povolené způsoby plateb, procentuální velikost zálohy, limity pro platby. Mezi nastavením systému bude i definování hodnotících kritérií pro služby v systému, které budou pro všechny stejné.

Poslední položkou budou obchodní podmínky. Ty budou stejné pro celý systém. Zde budou práva a povinnosti zákazníků a poskytovatelů a všichni se jich musí držet. Pro podmínky mezi poskytovatelem a provozovatelem si mezi sebou musí uzavřít smlouvu, kterou systém neřeší.

Ostatní funkčnost

Hodnocení služeb

Systém bude umožňovat, za pomoci zpětné vazby, hodnotit služby zákazníky. Po provedení služby zašle na email zákazníka a přiřazených osob email s formulářem a prosbou o jeho vyplnění. Na formuláři bude několik základních otázek o provedení služby a spokojenosti s ní. Odpovědi budou na výběr, nebudou se vyplňovat zákazníkem, budou se pouze vybírat. Odpovědi budou typu od - do, v odpovědi bude spokojenost v bodech jako 5 - velmi spokojen nebo 1 - velmi nespokojen. Tyto otázky bude spravovat provozovatel.

Odpověď se odešle potvrzením formuláře a systém ji automaticky zpracuje a promítne do hodnocení služby. Hodnocení se bude stahovat u kurzů k poslední kategorii, u hodinových služeb ke službě a u pobytů k celému hotelu.

Tisk

Systém bude umožňovat vytisknout každý zobrazený seznam v systému i detail objednávek, zákazníků, poskytovatelů a služeb. Při zobrazování seznamů, musí být zřetelně rozeznatelné jednotlivé řádky.

Generování unikátních čísel

Systém bude generovat v systému unikátní čísla pro zákazníky, objednávky, služby, kategorie, přídavné produkty a jejich kategorie.

Přidávání vlastních položek

V systému bude možno přidat vlastní položky k entitám zákazník, služba a objednávka. Při přidání položky se položka objeví u nových entit i u těch, které již v systému byly vytvořeny. Každá

položka bude buď zákaznická, nebo poskytovatelská. To znamená, že si poskytovatel bude moci k zákazníkovi přidat vlastní položku, která bude označovat např., zda je zákazník neplatič. Tato položka bude sice u zákazníka, ale uvidí jí jenom poskytovatel, bude to tedy checkbox. Položky, které se budou moci k entitám přidat, budou ze skupiny předdefinovaných elementů jako textové pole, checkbox, dropdown menu s položkami, textarea.

Zhodnocení

Nyní jsem provedl detailní sběr a analýzu požadavků, které by měl splňovat kvalitní, použitelný, rezervační sytém. Je z něj vidět, že požadované funkčnosti je opravdu velké množství a jeden programátor by jí vyvíjel opravdu dlouho, je to tedy projekt spíše pro vícečlenný tým. Proto budu dále počítat s tím, že vyvíjím systém pouze pro jeden typ služby a to pro kurzy.
Systém se pokusím navrhnout tak, aby byl snadno rozšířitelný o další typy služeb. Již teď z této analýzy požadavků je vidět, že vybrané služby mají dost odlišné rysy, a proto by byl už jen vzhled aplikace dosti odlišný. V dalších částech se při představování použitých technologií zaměřím i na ono snadné rozšíření a vyberu nejjednodušší způsob jeho provedení.

Datový návrh

V této kapitole se budu zabývat popisem datového modelu databáze, kde musím zachytit všechna potřebná data pro aplikaci, která zároveň musí odpovídat databázovým normám. Celý datový model je zachycen v příloze „Datový model“. Tato příloha obsahuje všechny tabulky databáze, jejich sloupečky a propojení. V datovém modelu popíši jednotlivé tabulky. Začnu na nejvyšší úrovni uchovávaných dat a postupně se propracuji níž. Nejvýše v systému je provozovatel a tím také začnu.

Datový model je vytvořen pouze pro jeden typ služeb a to kurzy, jak jsem již zmínil v minulé kapitole. Pro snadné odlišení tabulek související s tímto typem!služby budou všechny označeny prefixem „courses_“. Tím vyřeším problém stejně se jmenujících tabulek pro různé typy služeb, které se zabývající stejnou věcí, např. objednávkou. Tento problém je možné řešit schématy, ale ne všechny databáze schémata podporují. To by mohlo způsobit problém při přechodu na jiný typ databáze.

Provozovatel

Tabulka **operator,** jak jsem již řekl, uchovává data o provozovateli systému. Obsahuje jeho název, popis, cestu k obrázku a kontaktní údaje jako adresu, telefon a email. Tyto údaje budou uchovávány pro možnost zobrazení údajů o provozovateli zákazníkům.

Provozovatel je pouze jeden jako společnost, ale bude potřeba vytvořit pro něj uživatele, kteří se budou do systému přihlašovat a pracovat s ním. Údaje o těchto uživatelích bude uchovávat tabulka **operator_users**. O uživatelích se bude uchovávat jen jejich jméno, příjmení, tituly, email, který se bude používat jako přihlašovací jméno, heslo a telefon. Heslo se bude uchovávat v zakódované podobě, která je o hodně delší, proto bude jeho možná délka až 100 znaků, i když heslo bude omezeno, např. na 20 znaků.

Poskytovatel

Tabulka **vendors** bude uchovávat informace o jednotlivých poskytovatelích služeb. O poskytovatelích budou uchovávány stejné informace jako o provozovateli, navíc zde bude uvedeno bankovní spojení a informace, zda je aktivní nebo ne. Tato informace bude sloužit k možnosti vypnutí poskytovatele v sytému bez jeho smazání.

Stejně jako u provozovatele, tak i poskytovatel bude potřebovat své uživatele, kteří se do systému budou moci přihlásit a pracovat s ním. Informace o nich se uloží do tabulky **vendors_users.** U těchto uživatelů se budou uchovávat stejná data jako u uživatelů provozovatele. Zde je velmi důležitý odkaz na poskytovatele, ke kterému patří. Nebude uvedena jejich adresa apod., protože provozovatel bude komunikovat přímo s poskytovatelem jako společností a ne s jednotlivými uživateli.

Zákazník

Informace o zákaznících systému uchovává tabulka **customers**. Jsou v ní informace zadané při registraci zákazníka do systému, které jsem popsal v předchozí kapitole. Navíc je zde sloupec pro hash kód do aktivačního odkazu a sloupeček pro krátkodobé uložení hesla v nešifrované formě. Heslo v nešifrované formě je pro možnost jeho odeslání zákazníkovi v emailu, který potvrzuje aktivaci. Pak se heslo v této podobě smaže.

Přidávání vlastních položek

V minulé kapitole jsem mezi požadavky uvedl i možnost přidávání vlastních položek k základním entitám systému, kterými jsou zákazník, služba (v našem případě kurz) a objednávka. Jak to zachytit v databázi? Systém bude muset uchovávat informace o tom, jakého typu položka je, které entitě přísluší a také data, která se do entity budou zadávat pro každou instanci.

Řešení je následující: jako první musím udělat číselník typů položek, což bude tabulka **element_type.** Do ní uložím její název, html tag a také typ pro případ tagu input. Podporované tipy v aplikaci budou textarea, text field, checkbox a dropdown menu. Dále vytvořím tabulku **elements**, která bude uchovávat všechny položky od všech entit. Zde bude uložen typ položky a číslo elementu. Dále si pro každou tabulku uchovávající informace o entitě vytvořím tabulku, která uchová informace o její dodatečné struktuře. Zde bude jméno položky, její popis, cesta k obrázku, odkaz na element a její číslo. Pro zákaníka to bude tabulka **customers_structure**. V této tabulce jsou téměř všechny informace o položce, tak proč zde není i typ položky a nezruším tabulku element? Je to z důvodu, že položka může být i typu drop down menu, neboli html tag select a toto menu má také své vlastní položky. Tyto položky od všech drop donwnů budou uchovány v jedné tabulce **drop_down_menus**, kde bude odkaz na element, číslo položky v elementu a její hodnota. Právě kvůli odkazu na element potřebujeme tabulku elements, protože je jen jedna, oproti tomu tabulek s dodatečnou strukturou entit bude několik, a to jako cizí klíč v databázi nelze realizovat.

Nyní mám zachyceno jakého typu položka je, jaké má možnosti výběru v případě drop down menu,

a ke které entitě náleží. Chybí už jen její hodnota k instanci entity, a proto vytvořím ke každé entitě tabulku s hodnotami pro tyto položky. Tabulka bude obsahovat odkaz na instanci entity, na položku a její hodnotu. V případě zákazníka se tabulka bude jmenovat **customers_values**.

K zákazníkovi budu moci kromě volitelných položek přiřazovat i statusy. Statusy se nebudou přiřazovat k zákazníkovi jako entitě, ale k jednotlivým zákazníkům (instancím). Tyto statusy budou uloženy v tabulce **status** a budou obsahovat jméno, popis, cestu k obrázku a odkaz na provozovatele, jenž status vytvořil. Pro přiřazení statusů k jednotlivým zákazníkům musí existovat tabulka, jelikož jde o vazbu many to many. Tato tabulka se jmenuje **status2customers** a neobsahuje nic jiného, než odkaz na status a zákazníka.

Kurzy

Další důležitou entitou je kurz jako takový. Informace o nich budou uchovány v tabulce **courses_courses**. Jsou zde uloženy informace popsané v předchozí kapitole, což je jeho unikátní id, cena, kapacita, datum od - do, volná místa, počet opakování týdně, jeho kategorie a odkaz na poskytovatele.

Určitě jste si všimli, že tabulka s informacemi o kurzech neuchovává, které dny v týdnu jsou hodiny kurzu. Zde je tento počet dynamický a každý kurz jich může mít jiný počet. To znamená, že jde o vazbu one to many a tyto údaje bude uchovávat tabulka **courses_courses_hours**. Obsahuje pouze čas od kdy, čas do kdy a odkaz na den v týdnu. Dny v týdnu jsou obsaženy v tabulce **days_in_week**, což je jejich číselník. Toto řešení umožňuje mít vícero hodin v jeden den, což se dá využít např. při týdenních pobytových kurzech , kde se učí i třikrát denně.

Ke kurzu budu také přiřazovat volitelné položky jako k zákazníkovi, proto stejným postupem jako v minulém případě vytvořím tabulky **courses_courses_structure** a **courses_courses_values**. Ostaní k tomu potřebné tabulky mám již vytvořené.

Kategorie kurzů

Také jsme si v požadavcích uvedli, že kurzy budou patřit do kategorií, které se budou do sebe zanořovat. Pro tyto kategorie máme tabulku **courses_categories**, kde jsou sloupečky pro jejich jména, popis a odkaz na nadřazenou kategorii

Přídavné produkty

Bude potřeba někde ukládat informace o přídavných produktech. K tomu slouží tabulka **courses_add_products**, kde bude uloženo jméno, popis, cena, cesta k obrázku a odkaz na kategorii produktu. O kategoriích přídavných produktů se ukládají stejné informace jako o kategoriích samotných kurzů a jsou uloženy v tabulce **courses_ap_categories**.

Pro opatření nabízení přídavných produktů pouze ke správným kurzům je vytvořena tabulka **courses_apcategori2categori**, která zajišťuje přiřazení kategorií přídavných produktů ke správným kategoriím kurzů.

Objednávky

Další velmi důležitou entitou systému bude objednávka. Pro uložení dat o objednávkách existuje tabulka **courses_orders,** zde budou uloženy základní informace o objednávce, např. datum vytvoření, odkaz na zákazníka, cena, odkaz na slevu a informace o tom, zda je objednávka nová, potvrzená, vytvořená poskytovatelem či stornovaná. Samozřejmě zde nejsou služby k objednávce, jelikož je jich dynamický počet a znovu jde o vazbu many to many. Pro toto přiřazení mám vlastní tabulku **courses_cours2order,** kde je odkaz na objednávku, kurz a osobu využívající službu, tedy odkaz na zákazníka nebo jeho osobu. Tabulka sloužící stejnému účelu bude i v případě přiřazení přídavných produktů k objednávce. Zde nebude osoba využívající službu, ale cena produktu v době objednání. Ta je uvedena proto, že se ceny produktů budou v průběhu času měnit, a to by ovlivnilo ceny na objednávce. Tabulka pro toto přiřazení je **courses_ap2order**. K objednávce, jako k poslednímu objektu, budu přiřazovat i vlastní položky, proto jsou zde znovu tabulky **courses_orders_structure** a **courses_orders_values.**

U objednávek budeme spravovat i platby patřící k objednávce. Tyto platby budou uloženy v tabulce **payments**, kde bude odkaz na objednávku, částka, informace o tom, zda je již zaplacená, datum splatnosti a datum zaplacení.

Slevy

Co ještě chybí? Teď už to vypadá, že jsem pokryl všechny důležité části programu, stále to ale není vše. Zatím nemám zachycené v systému například slevy. Slevy jsou také samostatná entita, která se bude přiřazovat k objednávkám a tudíž jde o vazbu one to many. Budou uloženy v jedné tabulce **discounts,** a nebude potřeba žádná propojovací tabulka. Zde se bude ukládat jméno slevy, její popis, hodnota, jednotky a odkaz na poskytovatele, který slevu vytvořil. Jelikož mezi požadavky je i zadávání slevového kódu, musím vytvořit také tabulku se slevovými kódy. To bude tabulka

discounts_codes, kde bude uložen kód slevy, čas použití, informace, zda byla sleva již použita a odkaz na slevu.

CRM statusy

Další funkčností systému jsou tzv. crm statusy, které určují slevu od poskytovatele zákazníkovi, pokud u poskytovatele utratí určitý obnos za určitý čas. Tabulka uchovávající tyto informace je **crmstatus**. V ní se uchovává jeho název, popis, čas pro nasbírání sumy, sumu a odkaz na slevu a na poskytovatele. Přiřazení těchto statusů k zákazníkům je opět vazba many to many a potřebuji k tomu spojovací tabulku, což je v tomto případě **crm2customers**, kde je pouze odkaz na zákazníka a na status.

Osoby zákazníka

Další potřebnou součástí je i seznam osob zákazníka, které přiřazuje k objednaným službám, pokud je nebude chtít sám pro sebe. Tyto osoby budou uloženy v tabulce **guests**. Zde budou uloženy stejné informace jako u zákazníků, ovšem bez hesla a hash, a bude v ní odkaz na zákazníka, kterému patří.

Emaily

V aplikaci budu posílat také různé emaily, např. pro potvrzení registrace atd. Tyto emaily se budou vytvářet z určitých šablon, které budou obsahovat text emailu a určité proměnné, které se v nich budou přepisovat na aktuální data jako např. datum nebo jméno zákazníka. Proměnných bude určitá skupina a jedna proměnná se bude moci vyskytovat ve více emailech a jeden email bude obsahovat i více proměnných. Jedná se tedy opět o vazbu many to many.

Data pro email musím rozdělit do několika tabulek. První bude **email_tempplates** uchovávající jméno šablony, předmět a text šablony. Druhá tabulka **email_variables,** bude obsahovat množinu proměnných k přepsání a u každé bude její jméno a znak, který se má přepsat za aktuální hodnotu. Pro jejich přiřazení použiji spojovací tabulku **email_template2variable**.

Hodnocení

Data pro hodnocení se budou skládat z množiny otázek a z množiny odpovědí od zákazníků, vztahující se k jednotlivým kategoriím kurzů. Proto data budou ve dvou tabulkách jedna je **courses_evaluations_questions** s množinou otázek a jejich názvem (pro zobrazení výsledků) a druhá bude **courses_evaluations_values** s odpověďmi. U odpovědí bude bodová hodnota

odpovědi, odkaz na zákazníka nebo hosta, který to vyplnil, odkaz na otázku, datum hodnocení a odkaz na kategorii, ke které se otázka vztahovala.

Nastavení

V aplikaci budu potřebovat i další nastavení, jako např. časové lhůty, velikost zálohy atd. Všechny tyto hodnoty budou v tabulce **settings.**

Další nastavení by mohl být druh služby, kterou aplikace zrovna podporuje, jde o to, jestli bude zrovna rezervovat kurzy, nebo jiné druhy služeb. V této práci to není aktuální, jelikož jsem uvedl, že tato práce bude obsahovat postup tvoření pouze aplikace pro typ služby, kurzy. Ale chci aplikaci připravit, tak aby se dala snadno rozšířit, proto zde bude i tabulka **services** s výčtem druhů služeb.

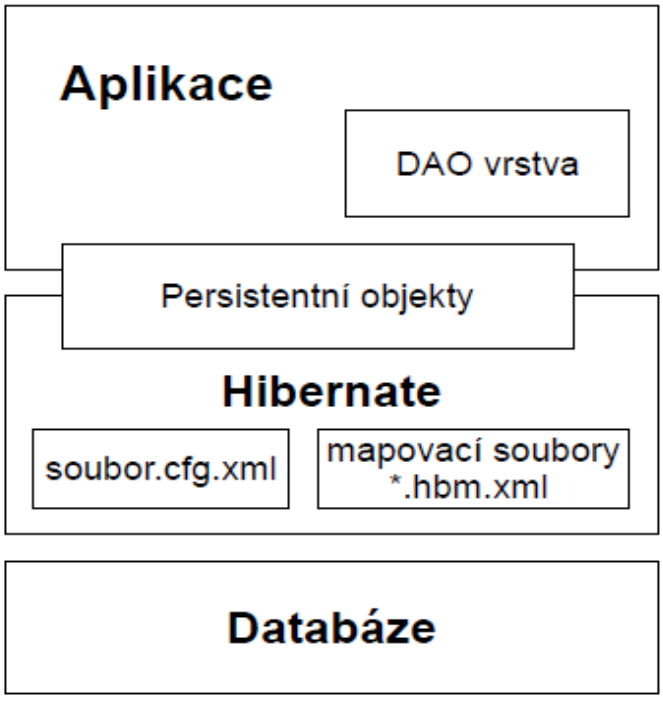
Hibernate

Bylo dokázáno, že 70 – 90 % času vývoje aplikací, využívajících pro ukládání dat databázi, zabere vytvoření kódu manipulujícího s daty z databáze. Hibernate je nástroj obsahující prostředky pro objektově-relační mapování tříd jazyka Java na tabulky relační databáze (ORM) . Jinak řečeno poskytuje techniku pro převod dat z objektového modelu do relačního modelu a zpět. Tak dokáže usnadnit práci s daty nad databází a snížit čas potřebný na vývoj kódu pro jejich manipulaci.

Mimo to Hibernate poskytuje nástroje, pro práci s těmito daty včetně objektově orientovaného jazyka HQL, který dokáže na základě objektové notace uložená data z databáze vybírat v podobě objektů. Hibernate dále poskytuje mechanismy pro transakční zpracování dat, polymorfní perzistenci, kešování, objektově orientované sestavování dotazů, vlastní datové typy a mnoho dalších funkcionalit. S řadou z nich Vás seznámím v následujících podkapitolách. Důležitá výhoda Hibernate je, že poskytuje jednotné API pro práci se všemi podporovanými databázemi a databázovými servery.

Architektura

Na následujícím obrázku je zachycen příklad architektury třívrstvé aplikace s využitím nástroje hibernate.



Na obrázku je vidět, že architektura je složena z několika částí. Přímo s databází komunikuje Hibernate pomocí konfigurace uložené v konfiguračním souboru s názvem *.cfg.xml. Ten pak pomocí mapovacích souborů s názvy *.hbm.xml mapuje tabulky z databáze na persistentní třídy nebo tzv. POJO (Plain Old Java Object) objekty. S těmito objekty pak manipuluje DAO vrstva v aplikaci, která obstarává manipulaci s daty využitím hibernate funkcí.

Persistentní třídy (POJOs)

Persistentní objekty jsou obyčejné třídy, na které Hibernate nemá žádné speciální nároky. Musí obsahovat jen bezparametrický konstruktor a je doporučeno, aby obsahoval atribut pro uchování hodnoty primárního klíče. Další vlastnosti nejsou povinné, ale dle zvyklostí obsahují manipulační metody pro každou instanční proměnou a implementují rozhraní **Serializable** pro jejich serializaci. Ukázkou persistentní třídy z aplikace je například třída **DaysInWeek.**

```
public class DaysInWeek implements java.io.Serializable {

	private int dayNr;
	private String name;
	private String shortcut;

	public DaysInWeek() {
	}

	public String getShortcut() {
		return shortcut;
	}

	public void setShortcut(String shortcut) {
		this.shortcut = shortcut;
	}

	.
	.
	.
}
```

Konfigurace (*.cfg.xml)

V tomto konfiguračním souboru je konfigurace uložena v kořenovém elementu `<hibernate-configuration>` a hlavní nastavení je v elementu `<session-factory>.` Význam session faktory vysvětlím později. Nyní vysvětlím jen k čemu slouží nastavení v tomto elementu. První část této konfigurace je v elementech property a vztahují se hlavně k databázi. Struktura elementu je

```
<property name="název nastavení">hodnota</property>
```

Nastavení jsou následující

hibernate.connection.driver_class

- určuje třídu s ovladačem databáze, hodnota je jiná pro každý typ databáze

Hodnota v aplikaci : „`org.postgresql.Driver`“ – pro databázi PostgreSQL

hibernate.dialect

- určuje jakým dialektem má komunikovat hibernate s databází, ale opět záleží na typu databáze

Hodnota v aplikaci: „org.hibernate.dialect.PostgreSQLDialect“

hibernate.connection.url

-druh databáze, adresa databázového serveru a název databáze

Hodnota v aplikaci: „`jdbc:postgresql://`localhost`/DDB“`

hibernate.connection.password - heslo pro připojení k databázi

hibernate.connection.username - uživatelské jméno pro připojení k databázi

hibernate.default_schema

- určuje schéma v databázi, se kterým má hibernate komunikovat

Hodnota v aplikaci: „public“ – Jde nastavit jakékoli, ale jen jedno. Doporučuje se tabulky vytvořit ve veřejném schématu, aby při změně typu databáze nevznikl problém s právy.

hibernate.show_sql

- Určuje zda se mají vypisovat do konzole sql dotazy prováděné hiberantem. Hodnota je defaultně „false“.

hibernate.search.autoregister_listeners

-

Hodnota v aplikaci : „false“

hibernate.bytecode.use_reflection_optimizer

-

Hodnota v aplikaci : „false“

Další nastavení je v elementu mapping, kde je jen jeden atribut a to `resource`. Ten určuje cestu k jednomu mapovacímu souboru. Těchto souborů může být více. Příklad:

```
<mapping resource="dip/hosna/mapping/Customers.hbm.xml" />
```

Mapování (*.hbm.xml)

Hibernate poskytuje několik strategií mapování:

Jedna třída mapovaná na jednu tabulku

Nejjednodušší mapování, které neumožňuje dědičnost ani polymorfismus.

Hierarchie tříd mapovaná na jednu tabulku

Při tomto mapování pomocí speciálního atributu (diskriminátoru) Hibernate rozpozná, který objekt daná relace zachycuje, a pak objekt vytvoří. Mapování podporuje polymorfismus i dědičnost.

Každá třída hierarchie mapovaná na jednu tabulku

Také umožňuje používat dědičnost i polymorfismus, ale na rozdíl od minulé strategie mapovaní hierarchie na jednu tabulku, tato strategie produkuje normalizované databázové schéma.

Každá konkrétní třída hierarchie mapovaná na jednu tabulku

I tato strategie umožňuje používat dědičnost i polymorfismus. V tabulkách ale dochází k duplikaci společných atributů. Tato strategie je často používaná při práci s již existujícím databázovým schématem, které nemáme možnost měnit.

Jedna třída na jednu tabulku

Mapovací soubory obsahují mimo DTD kořenový element `<hibernate-mapping>`. Pro mapování třídy slouží element `<class>`, který má sadu následujících atributů:

Atribut name - název perzistentní třídy i s balíkem

Atribut table - název tabulky v relační databázi, na kterou se bude třída mapovat.

Atribut proxy - obsahuje rozhraní, které bude objekt implementovat, pokud bude součástí líně vyhodnocovaných kolekcí nebo objektů.

Atribut where - omezí podmínkou výběr dat reprezentovaných touto třídou.

Atribut lazy - zapne / vypne líné získávání dat z kolekcí a asociovaných objektů

Mapování primárního klíče

Všechny instanční proměnné v třídách jsou určitého typu. Hibernate podporuje všechny základní javovské datové typy. Typem tohoto atributu může být ale i objekt, kolekce objektů nebo i datový typ vlastní. První mapovaná proměnná vždy zachycuje primární klíč (id). Pro toto mapování slouží element `<id>`. Příklad z mapovacího souboru pro tabulku DaysInWeek

```
<id name="dayNr" type="int">
    <column name="day_nr" />
    <generator class="assigned" />
</id>
```

Z definice je vidět, že proměnná „dayNr“ v této třídě typu int odpovídá primárnímu klíči ze sloupce „day_nr“ a je generován generátorem typu „assigned“. Element obsahuje atributy, které obsahuje každá mapovaná proměnná.

Atribut name - název mapované instanční proměnné.
Atribut column - název sloupce databázové tabulky, na kterou se bude mapovat. (pokud není zadán, jméno atributu je shodné s názvem proměnné)
Atribut type - typ této proměnné. Pokud není zadán, Hibernate se ho pokusí zjistit pomocí reflexe.

Element `<generator>` se svým attributem „class“ udává, jak bude hodnota klíče získávána, buď bude zadána programem před uložením, nebo bude určitým způdobem generována. Hibernate nabízí i několik vlasních generátorů. Typ „assigned“ udává, že hodnota klíče bude zadána programem před uložením do databáze. Pokud tak nechci učinit, mohu využít například typ „uuid“,který generuje 32 znakové řeťezce hexadecimálních čísel. Dále se nabízí generátor vytvářející sekvenci čísel, inkrementální generátor atd.

V aplikaci využívám i možnost přiřazení hodnoty klíče jako daší hodnoty z určité sequence. Je to obdoba autoinkrementu z MySQL pro PosgreSQL. Pak by element `<generator>` vypadal názsledovně:

```
<generator class="sequence" >
      <param name="sequence">název sequence</param>
</generator>
```

Mapování primitivních typů

K mapování proměnných primitivních typů slouží element `<property>`, který umožňuje mapovat proměnné základních datových typů jazyka. Příklad mapování proměnné základního typu:

```
<property name="name" type="string">
      <column name="name" length="50" />
</property>
```

Příklad je ze souboru CoursesStructure.hbm.xml a je v něm namapována proměnná „name“ typu String na sloupec „name“ s maximální délkou omezenou na 50 znaků. Mimo těchto atributů může element obsahovat ještě atributy not-null, update a insert. Ty určují ve kterých příkazech má být proměnná zahrnuta a zda smí být (zda může nabývat hodnotu) null.

Jiným typem atributu tohoto elementu je atribut se jménem formula, který obsahuje SQL dotaz. Tato proměnná pak nabývá po načtení objektu z databáze hodnot právě tohoto vyhodnoceného dotazu. V tomto případě obsahuje element pouze tento atribut a atribut name.

Mapování vazeb mezi tabulkami

Pokud je v databázi vaba cizím klíčem na jinou relaci ve vztahu 1:1 nebo 1:N, tak persistentní třídy budou mít jako proměnou přímo odkazovaný objekt. Pro toto mapování slouží elementy `<many-to-one>` a `<one-to-one>`. Příklad z VendorsUsers.hbm.xml :

```
<many-to-one name="vendors" class="dip.hosna.pojos.Vendors" fetch="select" >
            <column name="vendor_id" />
    </many-to-one>
```

Zde je vidět opět název proměnné a sloupce, který v tomto případě obsahuje cizí klíč. Místo typu je zde název persistentní třídy odkazovaného objektu. Atribut `fetch` určuje, kdy se má daný objekt vybrat z databáze. Hodnotou *`select`* určuje, že se daný objekt má načíst z databáze při první manipulaci s objektem. Tato hodnota je implicitní. Mimo této hodnoty se nabízí ještě hodnota *`join`*. Při této hodnotě se načte celý objekt hned při načítání persistentní třídy, která ho má jako instanční proměnou. Element `<one-to-one>` obsahuje stejné atributy.

Mapování kolekcí

Pokud mám vazbu 1:N, tak mohu namapovat objekty i opačně. To znamená, že by objekt, který se v databázi odkazuje na daný objekt cizím klíčem, měl jako instanční proměnou kolekci objektů, které se na něj odkazují. K tomuto mapování slouží element `<set>` v kombinaci s elementem `<one-to-many>`. Příklad z CoursesCourses.hbm.xml, kde je namapována kolekce jeho vyučovacích hodin, které mají odkaz na svůj kurz:

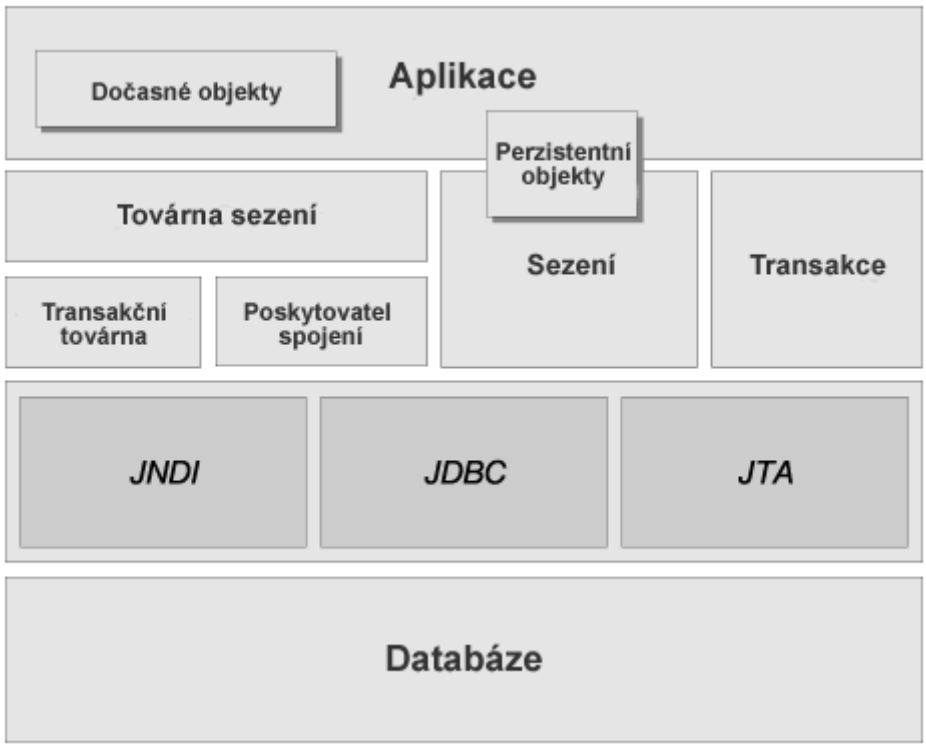
```
<set name="coursesCoursesValueses" table="courses_courses_values"
                    inverse="true" lazy="true" fetch="select" cascade="all">
                <key>
                    <column name="cours_id" />
                </key>
                <one-to-many class="dip.hosna.pojos.CoursesCoursesValues" />
```

```
</set>
```

V elementu `<set>` jsou atributy pro název proměnné a tabulky. Dále je zde opět atribut `fetch` a k tomu `lazy` , ale oba znamenají to samé. Strategie vybírání dat z databáze při prvním použití se nazývá líná. Atribut slouží k určení skutečnosti, že jde o oboustranné mapování, a že jde druhou stranou mapování. Pokud je uveden tento atribut, tak se změny provedené v této kolekci nepromítnou do databáze. Tomuto chování se říká tranzitivní perzistence.

Pokud chci, aby se tyto změny projevily do databáze, uvádí se atribut `cascade,` jehož hodnota udává při kterých operacích se změny mají projevit. Možné hodnoty jsou all, create, merge, delete, save-update, evict, replicate, lock a refresh .

Vnitřní architektura Hibernate



Základním kamenem práce s Hibernatem je sezení (implementace rozhraní **Session**). Tento objekt obsahuje JDBC spojení a stará se o konverzi mezi objekty a relacemi. Sezení je snadné vytvořit a opět uvolnit, jelikož je to velmi častá operace, která se provádí minimálně jednou u každého HTTP požadavku. Sezení také slouží jako továrna pro transakce a udržuje cache první úrovně.

Session se vytváří za pomoci továrny sezení (rozhraní SessionFactory), která se konfiguruje v již

zmíněném souboru *.cfg.xml v elementu `<session-factory>`, který byl popsán dříve. Tato továrna udržuje cache druhé úrovně. Na rozdíl od samotného sezení je náročné vytvořit objekt implementující toto rozhraní, proto v programu existuje jen jedna instance na jednu databázi. Pro každou databázi musí být nová továrna. Továrna získává JDBC spojení od poskytovatele spojení. Ten připravuje množinu spojení a pak je předává jednotlivým továrnám.

Dále je na obrázku transakce, zde je rozhraní **Transaction**, které není povinné, ale doporučené. Toto rozhraní a jeho API slouží k odproštění aplikačního kódu od implementace transakcí na nižších úrovních. To umožní následnou přenositelnost aplikace. Samotné transakce tvoří transakční továrna. To znamená objekt implementující rozhraní **TransactionFactory**. V aplikaci jsou pak transakce získávané ze sezení.

Stavy perzistentních objektů

Perzistentní objekty se liší ve vztahu k Hibernate sezení. Podle tohoto vztahu se rozlišují tři stavy perzistentních objektů. Podle stavu se pak s objekty v aplikaci pracuje.

Dočasné(pomocné) objekty
To jsou objekty, které nikdy nebyly v žádném vztahu s Hibernate sezením a tudíž nemají odpovídající řádek v databázi ani perzistentní identitu. Tyto objekty nemají nastavený primární klíč. Jedná se zejména o nově vytvořené objekty.

Odstavené objekty (Detached Instances)
Za tyto objekty jsou považovány objekty, které mají svoji perzistentní identitu a pravděpodobně i odpovídající řádek v databázi. Nejsou ale v oblasti působnosti sezení. Jsou to objekty, které byly vybrány z databáze a nyní se používají v aplikaci mimo působnost sezení, ve kterém byly vybrány z databáze. Hibernate nemůže zaručit, že změny těchto objektů budou promítnuty do databáze.

Perzistentní objekty
Mají svoji perzistentní identitu a jsou v oblasti působnosti Hibernate sezení. Každou změnu provedenou na těchto objektech je možno provést také v databázi.

Reverzní vytvoření perzistentních

Běžně je Hibernate využíván k vytvoření databáze podle předložených objektů a mapovacích souborů. V mé práci popíši postup opačný a to vytvoření mapovacích souborů a persistentních tříd podle již vytvořené databáze. Postup je opačný, proto název reverzní. Pro jejich vytvoření použiji nástroje obsahožené v HibernateTools. Takto vytvořená podoba objektů nebude jejich finální, bude to jen základ pro další práci. Postup vytvoření těchto objektů se skládá z několika kroků.

1.krok – konfigurační soubor Hibernate

Hibernatu se musí říci, kde je databáze a jak k ní přistoupit. Tyto informace uchovávám v již popsaném souboru *.cfg.xml. V souboru pro toto použití musí být informace o databázi, ale nikoli o umístění mapovacích souborů. V této fázi vývoje mapovací soubory ještě neexistují a budou se teprve vytvářet.

2.krok – konfigurace Hibernate console

Tuto konfiguraci v Eclipsu vytvořím jednoduše, protože je v nabídce „Nový“ mezi ostatními typy souborů pod názvem „Hibernate Console configurations“. Při jejím vytváření stačí vybrat projekt aplikace a konfigurační soubor Hibernate *.cfg.xml. Není potřeba zadávat žádné další údaje. Seznam konfigurací se zobrazuje v Eclipsu v Hibernate perspektivě v okně „Hibernate configurations“.

3.krok - konfigurační soubor reverze

Ke konfiguraci reverzního postupu slouží soubor *.reveng.xml. Soubor obsahuje informace o tabulkách a jejich sloupečcích, pro které se budou vytvářet potřebné soubory. Dále soubor obsahuje informace o převodu typů z databázových na Javovské. V eclipse se dá soubor sestavit graficky, kde se jeho obsah skládá z několika záložek. V první se vybere Hibernate konfigurace, v druhé se vyberou tabulky i se sloupečky a v poslední se vyberou databázové typy a jejich Javovské protějšky. Výsledná struktura dokumentu se skládá z kořenového elementu `<hibernate-reverse-engineering>` a elementů `<type-mapping>` a `<table>`. První obsahuje přiřazení datových typů a a druhý tabulky s jejich sloupečky.

4.krok – vytvoření souborů

Pro samotné vytvoření souborů je potřeba se v Eclipsu přepnout do perspektivy „Hibernate“ a v menu se zadat **„Run → Hibernate Code generation → Hibernate Code generation Configurations“**. Zde se musí vytvořit nová konfigurace, kde se v první záložce vybere již

vytvořená Hibernate Console konfigurace, složka a balík pro vytvořené soubory a hlavně *.reveng.xml soubor. Dále se zde musí vybrat všechny nastavení pro vyhledání many-to-many tabulek atd. V druhé záložce se vybírá, jaké soubory se budou generovat. Pro mojí aplikaci jsem vybral DomanCode (kód persistentních POJO tříd), Hibernate XML Mappings (mapovací *.hbm.xml soubory), DAO code (DAO třídy pro manipulaci s daty) a Hibernate XML Configurations (původní *.cfg.xml soubor doplněný o cesty k mapovacím souborům). Po zadání údajů stačí zmáčknout Run a soubory se vygenerují.

Nyní jsou v zadaném balíku vygenerované všechny zadané soubory. Vytvořené mapovací soubory jsou vytvořeny v adekvátním stavu a zatím se měnit nemusí. Při vývoji se pouze doplní kolekce o odpovídající cascade atribut. Pouze pokud se přemístí z místa vygenerování, musí se změnit jejich umístění v *.cfg.xml souboru, který se také upravovat nemusí. Předposlední vygenerované objekty, což jsou persistentní třídy, jsou poslední které se také upravovat nemusí. Poslední vygenerované DAO třídy jsem pro přehlednost v aplikaci přejmenoval z přívlastku Home na Dao. Další úprava těchto objektů spočívá ve smazání metody `getSessionFactory` a změně řádku deklarace této proměnné na `private SessionFactory sessionFactory;`.

Vygenerované dao třídy jsou vygenerované i metodami pro manipulaci s daty. Metody jsou `persist`, `attachDirty`, `attachClean`, `delete`, `merge`, `findById` a `findByExample`. Metoda `delete` slouží ke smazání z databáze. Metoda `persist` ukládá nový objekt do databáze. Metoda `attachClean` ve svém tě volá metodu `~~lock~~(instance, LockMode.NONE)` a ta má stejný význam jako metoda `persist`, proto se v aplikaci ani nevyužívá. Metoda `findByExample` dle názvu hledá a vrací instanci podle primárního klíče.

Metody `attachDirty` a `merge` slouží pomalu ke stejnému účelu. První metoda volá ve svém těle metodu `saveOrUpdate`. To znamená, že se pokusí najít podle primárního klíče daný objekt. Pokud ho nalezne, tak ho aktualizuje, pokud ne, tak uloží nový. Skutečnost, že objekt nenalezla, ale uložila nový, sdělí aplikaci vyhozením výjimky. Metoda `merge` slouží jen k aktualizaci objektu.

Poslední metoda v pořadí je metoda `findByExample`, která v těle vytvoří `Example` podle instance a kritéria podle třídy a vyhledává podle nich. Za kritéria bere všechny proměnné dané třídy primitivních typů, které se nerovnají null. Pokud chci nějaké kritérium dané třídy vynechat, musí se její název přidat v příkladu (`Example`) do kolekce `excludeProperty` před vyhledání. Celý kód metody rozšířené o vynechané kritéria vypadá ve třídě `CustomersDao` následovně:

```
public List findByExample(Customers instance, ArrayList<String> excluded) {
```

```
        log.debug("finding Customers instance by example");
        try {

              Example e = Example.create(instance);

              for(int i = 0;i<excluded.size();i++)
              {
                    e.excludeProperty(excluded.get(i));
              }


              List results = sessionFactory.getCurrentSession()
                          .createCriteria("dip.hosna.pojos.Customers")
                          .add(e).list();
              log.debug("find by example successful, result size: "
                          + results.size());
              return results;
        } catch (RuntimeException re) {
              log.error("find by example failed", re);
              throw re;
        }
}
```

Aplikační rámec Spring

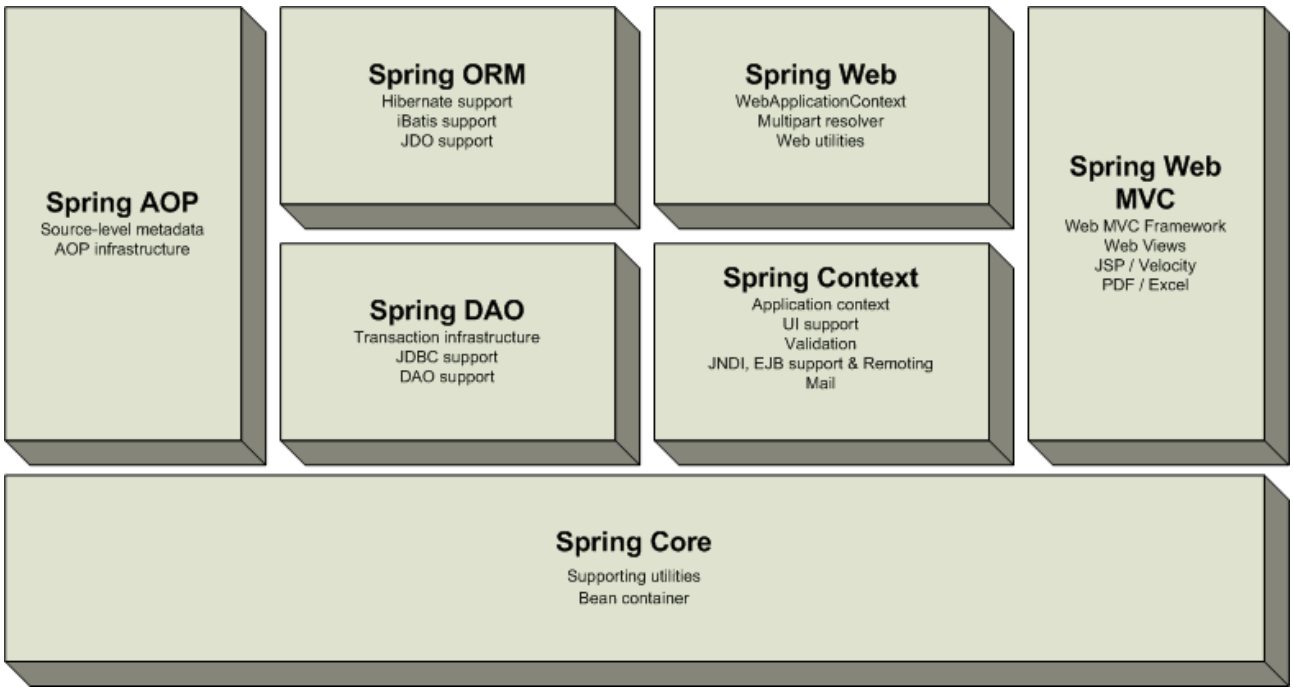
V mé práci popíši novou verzi springu, a to verzi 3.0. Spring má tak velké možnosti použití a nabízí tolik funkčnosti, že je velmi těžké shrnout jeho popis do několika řádků. Z tohoto důvodu bude následující popis velmi obecný. Spring je aplikační rámec určený k ulehčení vývoje Java / J2EE aplikací jehož hlavní předností je neinvazivnost. Neinvazivnost znamená, že neomezuje návrh architektury aplikace, ani neurčuje její strukturu. Jedna z mnoha výhod neinvazivnosti je možnost zaměnění tohoto aplikačního rámce za jiný bez nutnosti velkých změn v aplikaci.

Další výhody Springu je možné shrnout do následujících okruhů:

- Odstranění těsných programových vazeb mezi jednotlivými *POJO* objekty a vrstvami za pomoci návrhového vzoru **Inversion of Control.**
- Možnost volby implementace (**EJB** - Enterprise Java Beans, POJO) logické vrstvy pro aplikační architekturu a ne naopak (architektura by předurčovala implementaci).
- Řešení aplikačních domén bez nutnosti použití EJB, například transakční zpracování, podpora pro logické vrstvy formou webových služeb či **RMI** (Java Remote Method Invocation).
- Podpora implementace komponent pro přístup k datům, ať již formou přímého **JDBC** či **ORM** (objektově-relační mapování) technologií a nástrojů, jako již dříve popsaný **Hibernate** nebo **TopLink**, **iBatis** či **JDO**.
- Odstranění roztroušených konfigurací a pracného dohledávání jejich významu.
- Abstrakce vedoucí ke zjednodušenému používání dalších části J2EE. Například **JMS** (Java Messaging Services), **JMX** (Java Management Extensions), **JavaMail**, JDBC, **JCA** (Java Connector Architecture) nebo **JNDI** (Java Naming & Directory Interface).
- Ulehčení psaní a práce s unit testy.
- Správa a konfigurační management logických komponent.

1 Složení Springu

V předchozí podkapitole jsem vyjmenoval většinu vlastnosti a funkcí Springu. Tyto vlastnosti lze rozdělit do sedmi modulů, které jsou graficky zachyceny na následujícím přehledném schématu.



Všechny moduly lze využít kdekoli ve Spring kontejneru, jehož inicializace je záležitost několika řádek kódu. Kontejner je prostředí, ve kterém se odehrává život všech objektů, které pomocí Springu spravujeme.

Core modul

Je jádro tvořící základ celého frameworku a poskytuje funkčnost **IoC** (Inversion of Control), nebo nově značeno i **DI** (Dependency Injection) pro řízení celého kontejneru. Základ jádra tvoří implementace BeanFactory, která se stará o životní cyklus jednotlivých *POJO* objektů - vytvoření, nastavení vazeb mezi nimi, inicializace, poskytování objektů k použití a ukončení při zastavení kontejneru.

V minulém odstavci jsem uvedl, že technologie **IoC** a **DI** jsou základem jádra Springu. Z tohoto důvodu je zde vysvětlím podrobněji a i s rozdíly mezi nimi. IoC (česky „obrácení řízení“) je dlouho používaný návrhový vzor, který představoval techniku, pomocí které dochází k přenesení řízení běhu programu z programovaného kódu na podpůrný aplikační rámec. Dá se říci, že Spring je rámec tohoto návrhového vzoru. Postupem času se ale význam IoC posunul k popisování způsobu, jakým se zajišťuje provázání spravovaných objektů v aplikačních rámcích.

To ale neodpovídalo jeho původnímu významu, a proto se zavedl nový pojem DI (česky „Injektáž

závislostí“). Tento pojem znamená, že pokud budeme mít dva objekty A a B, kde objekt A bude obsahovat odkaz na objekt B. Při použití DI se vytvoří oba objekty při startu kontejneru a v objektu A bude inicializován i odkaz na objekt B. Existují dva základní Springem podporované způsoby, jak této inicializace dosáhnout.

První způsob se nazývá „Setter injection“, kde se vložení provádí pomocí klasických setrů, čili nastavovacích metod s názvem „set + Název proměnné“ s parametrem obsahující atribut pro nastavení proměnné. Druhý případ se nazývá „Constructor injection“, zde se vložení provádí skrze konstruktor, kterému se zadají jako parametry všechny potřebné instanční proměnné.

Context modul

Context modul staví na základě poskytnutém jádrem a přidává funkčnost navíc v podobě podpory pro internacionalizaci, JDNI a Java EE funkce, jako jsou EJB a JMX. Důležitá funkce tohoto modulu je odstínění od BeanFactory, jelikož modul Context je prostředník mezi klientským kódem a BeanFactory. Inicializace odstínění je pak možné provést mnohem transparentnějším způsobem, jako servlet kontejnerem či JUnit testem. Ústředním bodem modulu Context je rozhraní ApplicationContext. Jeho možnosti jsou větší než možnosti továrny tříd, jako například podpora zdroje zpráv (MessageSource support), podpora práce s externími zdroji dat (ResourceLoader support), nebo podpora událostí (framework events).

DAO modul

DAO (Data Access Object) modul obstarává abstraktní vrstvu pro práci s JDBC. Díky němu se nemusí opakovat kód při práci s databází (získání connection, vytváření statementu, iterování přes result set). Toto opakování je typické pro většinu aplikací pracujících s JDBC. Zajišťuje také podporu deklarativní definice transakcí a především „exception handling“, který převádí `java.sql.SQLException` na inteligentní hierarchii běhových výjimek.

ORM

Modul ORM poskytuje integrační vrstvy pro populární API objektově relačních mapovacích nástrojů, jako již popsaný **Hibernate** nebo **TopLink**, **iBatis, JDO** či **SPS**. Pomocí ORM balíčku můžete využít všech uvedených frameworků pro objektově-relační mapování v kombinaci se všemi

ostatními funkcemi, které SpringFramework nabízí, jako je například podpora transakcí a AOP.

AOP

AOP (Aspect Oriented Programming) je modul obsahující funkcionalitu pro Aspektově orientované programování. To slouží pro správu částí kódu, které se prolínají celou aplikací (autorizace, logování, transakce), ale přitom je nelze nikam zařadit. AOP tyto segmenty kódu seskupí do tzv. aspektů a ty pak umožňuje aplikovat kdekoli v aplikaci. Využití tohoto modulu se prolíná celým frameworkem a je to jedna z nejsilnějších vlastností Springu.

Web

Modul **Web** ve Spring Frameworku poskytuje základní webově orientované funkce, jako je funkce multipart file-upload, lokalizaci (i18n) či práci s cookies. Modul také slouží k podpoře integrace s webovými nástroji jako Struts, WebWork, JSF implementacemi nebo Tiles.

Web MVC

MCV (Model, Controller, View) obsahuje podporu springu pro webové třívrstvé aplikace, ve kterých je oddělena pohledová, logická a modelová vrstva.

2 Spring MVC

Z předchozího přehledu je jasné, že základem webové aplikace vyvíjené za pomoci springu je jeho část MVC. Každá taková aplikace musí mít třívrstvou architekturu. Účelem této architektury je sdružit objekty mající jeden účel do stejné vrstvy a nemíchat do nich jinou funkcionalitu. Aplikace obsahuje tyto vrstvy:

- **modelová** - model je tvořen objekty, které nesou data od kontroleru k pohledu, který je má následně zobrazit.
- **pohledová** – pohled (view) přijímá model s daty od kontroleru a zobrazuje (formátuje) je, následně zasílá odpověď zpátky
- **kontrolní** – kontroler (controller) zpracovává požadavky a na jejich základě načítá data a vytváří z nich model, který pak odesílá pohledu

Životní cyklus požadavku

Z popsaných vrstev jsou vidět i základní části životního cyklu požadavku. Pokud by cesta požadavku byla takto jednoduchá, nenabízela by moc možností úprav a zpracování. Celý životní cyklus požadavku ve spring MVC je mnohem delší a složitější. Požadavek je vždy typu `javax.servlet.http.HttpServletRequest` a po přijmutí třídou `DispatcherServlet` (třída bude popsána později) se dějí následující kroky:

kontext - Aplikační kontext daného servletu je vložen do atributu požadavku `DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE`, kde je k dispozici pro další objekty, které mají co dočinění s požadavkem dále

národní prostředí – Další atribut přidaný do požadavku je detektor národního prostředí. Detektor musí implementovat rozhraní `org.springframework.web.servlet.LocaleResolver`. Dále se využívá k identifikaci národního prostředí (Locale) klienta. Tento detektor musí být v aplikačním kontextu definován pod názvem *`localeResolver`*. Pokud tak učiněno není, je použita instance třídy `org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver`.

téma – Do požadavku je přidán i detektor motivu. Tento detektor musí implementovat rozhraní `org.springframework.web.servlet.ThemeResolver` a dále je využíván k identifikaci uživatelsky nastavených motivů vzhledu (themes). V aplikačním kontextu musí být definován pod názvem `themeResolver`. Pokud tomu tak není je do atributu přidána instance třídy `org.springframework.web.servlet.theme.FixedThemeResolver`.

soubory – pro snadnější práci se soubory, které jsou požadavkem poslány serveru, je do objektu implementujícího `org.springframework.web.multipart.MultipartHttpSertvletRequest` přidán i objekt v aplikačním kontextu definovaný pod názvem `multipartResolver`, který implementuje rozhraní `org.springframework.web.multipart.MultipartResolver`. Pokud definován není, nepřidá se do požadavku nic.

mapování – Vyhledají se v aplikačním kontextu všechny objekty typu `org.springframework.web.servlet.HandlerMapping`. Ty se pak podle definovaného pořadí dotáží na kontroller a interceptor pro daný požadavek. Interceptor požadavek před a

postzpracovává, ale kontroler provádí jeho zpracováni. Není-li v aplikačním kontextu definován žádný objekt třídy `HandlerMapping`, je vytvořena instance třídy `org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping`.

předzpracování – Pokud byl vybrán alespoň jeden interceptor, požadavek je mu předán k předzpracování, než se předá kontroleru k samotnému zpracování. Interceptor s požadavkem může provést jakoukoli akci. Může požadavek např. zastavit nebo ho přesměrovat. V aplikaci je využíván k ověření, zda má uživatel přístup k daném cíli a pokud ne, je požadavek přesměrován na přihlašovací obrazovku.

typ kontroleru - Před předáním požadavku konkrétnímu kontroleru, se musí určit, jaký typ kontroleru bude použit pro jeho zpracování. Tento krok při zpracovávání požadavku slouží k umožnění snadné výměny MVC Spring rámce za jiný s jinými kontrolery, nebo jiné úpravy způsobu doručení požadavku ke kontroleru. Určení typu se provádí na základě objektu typu `org.springframework.web.servlet.HandlerAdapter`, který je vyhledán v Aplikačním kontextu na základě svého typu. Na rozdíl od předchozích objektu, které se vyhledávali podle názvu definice, zde smí tento název být libovolný. Výchozím objektem při jeho nenalezení se stává `org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter`, který umožňuje použití kontrolerů ze Spring MVC, které implementují rohraní `org.springframework.web.servlet.mvc.Controller.`

kontroler – požadavek je předán vybranému kontroleru ke zpracování. Na základě požadavku kontroler vytvoří model s daty ve spolupráci s aplikační vrstvu. Mimo modelu určí kotroler i logický název pohledu pro zobrazení daného modelu. Nakonec předá oba objekty informace dál.

postzpracování – požadavek je znovu předán interceptoru, pokud byl některý vybrán, k postzpracování. V aplikaci se toto post zpracování využívá k obohacení požadavku o kolekce odkazů pro menu podle role přihlášeného uživatele, pokud je některý vůbec přihlášen.

výběr pohledu – v minulém kroku byl do požadavku přidán i logický název pohledu. Tento název ale neobsahuje konkrétní soubor či soubory pro zobrazení. Na ty se musí název převést. Pro tento převod musí být definovány v aplikačním kontextu objekty typu

`org.springframework.web.servlet.ViewResolver`, kterým je postupně v definovaném pořadí předložen název pohledu. K převodu je použit první, který vrátí výsledný pohled k danému názvu. Pro tento účel je v aplikacích typicky využíváno některého ze šablonovacích systémů, které jsou k dispozici, jako FreeMaker nebo Velocity. V aplikaci se využívá šablonovací systém **Tiles** 2.

zobrazení – vrácený soubor/soubory pohledu je využit ke zobrazení daného modelu.

Interceptory

Při popisu života požadavku byl mnohokrát zmíněn objekt s označením interceptor. Tzv. interceptory jsou objekty, které implementují rozhranní `org.springframework.web.servlet.handler.HandlerInterceptor` a jsou k určené předzpracování a postzpracování požadavků. Toto rozhraní pro ně definuje následující metody:

`preHandle` - metodě je předán požadavek, před předáním kontroleru ke zpracování.

`PostHandle` - metodě je předán požadavek, po zpracování kontrolerem, ale před zobrazením modelu pohledem.

`AfterCompletion` – metoda je volána po zobrazení modelu pohledem. Lze využít k úklidu prostředků po životním cyklu požadavku

3 Integrace springu

Základem konfigurace každé webové J2EE aplikace je soubor „**WebContent/ WEB-INf /web.xml**“ ve kterém se definují jednotlivé servlety. Zde pro integraci springu musím přidat následující řádky:

```
<servlet>
  <servlet-name>spring</servlet-name>
      <servlet-class>
          org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>spring</servlet-name>
  <url-pattern>*.html</url-pattern>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Tímto se aplikace dozví o tom, že se bude používat jako servlet pod názvem spring třída `org.springframework.web.servlet.DispatcherServlet`, a že na ní budou přesměrovány

všechny požadavky, i ty s příponou „html“. Tato třída je základní kámen Sringu MVC. Je to tzv. přední kontroler, na který jsou směrovány všechny požadavky a on je pak předává dál jednotlivým kontrolerům, které je zpracují. Dalo by se říct že je to vstupní brána do springu. Po inicializaci bude tento dispečer hledat svou vlastní konfiguraci a ta musí být implicitně uložena ve stejné složce pod názvem „[název-servletu]-servlet.xml“. V tomto souboru je uložena konfigurace a definice potřebných objektů pro daný dispečer. Jelikož se v jedné aplikaci může vyskytovat dispečerů více, může se vyskytovat i více konfigurací s různými, v nich vytvořenými objekty. Tyto konfigurace o sobě ale navzájem neví. Jeden dispečer tedy neví o objektech druhého a nemůže je proto použít. Pro tyto účely se používá globální konfigurační soubor s označením „aplication context“, který je společný pro všechny dispečery. Pro zavedení toho kontextu do aplikace se musí do souboru web.xml přidat tento posluchač:

```
<listener>
      <listener-class>
            org.springframework.web.context.ContextLoaderListener
      </listener-class>
</listener>
```

Po zavedení posluchače se musí vytvořit soubor „applicationContext.xml“, do kterého se umístí definice všech globálních objektů. V mé aplikaci není vytvoření této konfigurace zapotřebí, ale pro přehlednost je zde vytvořena také a je do ní umístěno nastavení Hibernate, které popíši níže.

4 Konfigurace springu

V těchto konfiguracích se definují jednotlivé objekty a jejich závislosti. Tyto objekty spring při startu aplikace pomocí DI vytvoří, zajistí jejich provázáni podle definovaných závislostí, drží je při běhu aplikace a poskytuje je svým klientům. Konfigurační soubory jsou xml soubory s kořenovým elementem `<beans>`, ve kterém jsou jednotlivé objekty definovány v elementech `<bean>`. Normální definice objektů by mohla vypadat následovně:

```
<bean id="nejakyObjekt" class="com.balik.NejakyObjektImpl" />
<bean id="druhyObjekt" class="com.balik.DruhyObjektImpl" >
      <property name="ob" ref="nejakyObjekt" />
</bean>
```

V tomto případě musí existovat třída *NejakyObjektImpl* implementující určité rozhraní např. *NejakyObjekt*. Každý objekt který potřebuji springem takto vytvořit musí implementovat své rozhraní, aby bylo vyhověno základnímu požadavku OOP a to programování do rozhraní. Druhým objektem musí být třída *DruhyObjektImpl*, který bude implementovat také své rozhraní a k tomu

bude mít referenci na `NejakyObjektImpl`. Spring při startu vytvoří oba objekty a i s jejich provázáním. Objekty mohou samozřejmě obsahovat i jiné proměnné, jen nebudou naplněny springem. Definice objektů v xml souborech umožňuje způsob definice pro „Setter injection“ i „Constructor injection“. Dokonce umožňuje i definici základních datových typů, jejich kolekcí nebo čehokoli jiného.

Mimo tohoto způsobu je zde i možnost novějšího přístupu a to skrze anotace. Ty se píšou přímo do objektů, ušetří se práce s vytvářením dlouhých konfiguračních elementů se složitou syntaxí. Nevýhoda tohoto novějšího přístupu je, že není všechna konfigurace na jednom místě, ale je roztroušena po zdrojovém kódu. V mé aplikaci používám novější přístup s anotacemi, a proto zde základní typy anotací vysvětlím.

@Repository

Tato anotace se umisťuje do řádku nad definici třídy. Anotuje tedy celou třídu. Anotací označená třída bude přidána do seznamu springem vytvářených bean. Touto anotací se označují většinou Dao třídy.

@Service

Tato anotace má stejné použití a důsledek jako anotace předešlá. Rozdíl je v tom, že by se měla používat pro objekty z logické nebo servisní vrstvy. Momentálně nejsou nástroje, které by dělaly rozdíly mezi třídami označenými těmito anotacemi.

@Autowired

Tato anotace se používá pro označení instančních proměnných které mají být springem naplněny. Anotace lze použít jak pro proměnou, tak i pro konstruktor. Pokud je proměnou odkaz na jiný objekt, musí tento objekt být označen ve své definici jednou z předchozích anotací, aby byl před přiřazením springem již vytvořen.

@Controller

Tato anotace slouží k označení tříd které budou pracovat jako kontrolery. Používá se v kombinaci s anotacemi **@RequestMapping**, které mapují na jaký požadavek má kontroler, nebo jeho metoda reagovat. Přesné použití a všechny možnosti těchto anotací popíši dále v podkapitole zabývající se mapováním.

Propojení s ORM Hibernate

Jak jsem popsal již dříve, základní třídou pro práci s hibernate je SessionFacetory. Tato třída poskytuje jednotlivé sezení pro práci s persistentními objekty. Aby spring mohl používat hibernate musí si vytvořit třídu SessionFactory jako jednu z bean. To se provede přidáním následujících řádek do souboru applicationContext.xml, kde bude tato třída přístupná pro všechny dispečery.

```
<bean id="sessionFactory"
          class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
     <property name="configLocation">
          <value>classpath:hibernate-generated.cfg.xml</value>
     </property>

 </bean>
```

Je vidět, že objekt typu SessionFactory je třída poskytovaná Hibernatem a to přesně *org.springframework.orm.hibernate3.LocalSessionFactoryBean*. Po její definici je možné kdekoli v kódu vytvořit proměnnou typu `SessionFactory` a pokud jí označíme již zmíněnou anotací `@Autowired`, spring do ní automaticky po startu aplikace přiřadí zde vytvořený objekt. Proměnné tohoto typu se vyskytují pouze v dao třídách, které jsou jediné určené k manipulaci s daty.

Pro správnou funkčnost Hibernatem vygenerovaných dao tříd ve springu, je potřeba v nich provést tři úpravy. První úprava spočívá v předělání statické konstanty typu SessionFactory získané statickou metodou na instanční proměnou toho samého typu a označit ji anotací `@Autowired`. Konkrétně z

```
private static final SessionFactory sessionFactory = getSessionFactory();
```

na

```
@Autowired
private SessionFactory sessionFactory;
```

Vygenerovaná statická metoda `getSessionFactory()` se stává nepotřebnou a mohu ji ve všech třídách smazat. Druhá úprava spočívá v označení třídy anotací `@Repository`, aby jí spring vytvořil jako bean použitelnou jinými třídami. Třetí úpravou je vytvoření rozhraní pro každou z těchto tříd. Rozhraní těchto tříd je potřebnou podmínkou k možnosti vytvořit je jako spring bean. Pokud budu tuto třídu používat jako instanční proměnou (označenou opět anotací `@Autowired`) v jiné třídě, tak se jako její typ uvede rozhraní a ne přímo třída. Podle rozhraní spring pozná, kterou bean do proměnné má přiřadit.

Nyní je možno kdekoli v aplikaci využít dao třídy pro manipulaci s daty. Manipulace s daty by byla velmi nepohodlná, jelikož se nyní sezení, které manipuluje s daty, vytváří pouze na dobu jednoho požadavku do databáze. To znamená, že po načtení dat z databáze se ihned z persistentního objektu

stává objekt odstavený. Problém vznikne pokud tento objekt bude mít kolekci jiných objektů s línou strategií vybírání dat z databáze. Pokud se k nim pokusím přistoupit, musí se tyto objekty načíst z databáze, ale objekt s kolekcí již nemá svou persistentní identitu a sezení neví, které objekty má vrátit. Jedno řešení je nepoužívat línou strategii vybírání dat z databáze, ale to by se při vybrání jednoho objektu mohla načíst i půlka databáze.

Řešením tohoto problému je říci aplikaci, ať vytváří sezení na dobu životního cyklu jednoho požadavku. To se nastaví vytvořením filtru `OpenSessionInViewFilter` v souboru „web.xml“. Nastavení filtru vypadá následovně:

```
<filter>
     <filter-name>hibernateFilter</filter-name>
     <filter-class>
           org.springframework.orm.hibernate3.support.OpenSessionInViewFilter
     </filter-class>
</filter>

<filter-mapping>
     <filter-name>hibernateFilter</filter-name>
     <url-pattern>/*</url-pattern>
</filter-mapping>
```

Tímto nastavením se zavedl `hibernateFilter` representovaný třídou `org.springframework.orm.hibernate3.support.OpenSessionInViewFilter`, který zařídí potřebnou dobu života sezení. Aby byl filtr aplikován na všechny požadavky v aplikaci je namapován na řetězec „/*“, který representuje všechny požadavky. Pokud by zde byla určitá adresa, nebo část adresy, byl by filtr aplikován pouze na požadavky směrované na adresy odpovídajícím zadanému vzoru.

Poslední nastavení se týká definice transakcí. Někdy je potřebné, aby se určité operace, prováděné s daty, provedli v transakci. Aby tomu tak bylo, musí se v kódu obvykle definovat začátek a konec všude, kde je transakce zapotřebí. Za použití Hibernate tomu tak být nemusí. Pro Hibernate lze jednoduchou anotací nad metodou určit, že celá metoda je jedna transakce. Anotace je `@Transactional` a vkládá se do řádku nad metodu. Pro použití se nabízí metody v dao třídách, kde je vždy potřeba, aby se provedla metoda celá.

Někdy se v aplikaci může stát, že je potřeba jako transakci definovat i činnost s daty využívající více metod dao třídy najednou. Z těchto důvodů je v aplikaci definována ještě tzv. servisní vrstva. V této vrstvě je ke každé dao třídě vytvořena jedna servisní třída, která má svou dao třídu jako proměnou. V této třídě jsou definovány stejné metody jako v dao třídě, ale je zde i možnost

vytvořit metodu další, která by využívala více dao metod. Pak stačí tuto metodu opět označit anotací `@Transactional`. V aplikaci se pak nikde nepoužívá samotných dao tříd, ale manipuluje se s daty skrze servisní třídy. Každá servisní třída má definované své vlastní rozhraní a je označena anotací `@Service`, aby byla vytvořena jako bean při startu aplikace.

Nastavení lokalizace

Proces lokalizace aplikace se skládá ze dvou částí. Všechny texty v aplikaci se musí umístit na jedno místo mimo zobrazovací soubory, a také se musí rozpoznat národní prostředí, aby bylo jasné, které texty použít. Pro první část nabízí spring možnost použití tzv. zpráv. Princip je jednoduchý, vytvoří se soubor kam se umístí požadované texty (zprávy) pod určitým klíčem, pod kterým se pak budou volat ze zobrazovacích souborů. V souborech pro zobrazení se pak zavolá pouze zpráva uložená pod určitým klíčem a ta se následně vypíše.

Zprávy

Aby bylo možné zprávy používat, musí se ve springu nastavit jejich zdroj. To se provede přidáním následující beany do konfigurace:

```
  <bean id="messageSource" class=
"org.springframework.context.support.ReloadableResourceBundleMessageSource">
          <property name="basename" value="classpath:messages" />
          <property name="defaultEncoding" value="UTF-8"/>
      </bean>
```

Zde je zaveden zdroj zpráv *messageSource* representovaný třídou *org.springframework.context.support.ReloadableResourceBundleMessageSource*. Tato třída umožňuje i znovu načítání zpráv v určitém časovém limitu. Tuto možnost v aplikaci nevyužívám, jelikož není žádný předpoklad, že by se zprávy měnily a musely se znovu načíst. Pokud se čas nenastaví jsou zprávy načteny pouze jednou. Tato třída je zde použita, jelikož umožňuje snadné nastavení kódování souboru se zprávami.

Parametry zdroje jsou *basename* a *defaultEncoding*. První určuje řetězec, kterým musí začínat názvy s uloženými zprávami a druhý jejich kódování. Výsledkem tohoto nastavení je, že spring načte jako zdroje zpráv všechny soubory s názvem „`messages*.properties`“, které nalezne v classpath. Formát ukládání zpráv v těchto souborech je jednoduchý. Na každé řádce je pouze jedna zpráva ve formátu „kód.zprávy=Text zprávy“. Příklad: `label.save=Ulož.` Formátování umožňuje prázdné řádky, mezery v textu zprávy i znaky '.' a '–' v kódu zprávy.

Detekce prostředí

Pro detekci prostředí se musí v konfiguraci vytvořit bean s názvem *localeResolver,* jak jsem již popsal v životním cyklu požadavku. Pro rozpoznání národního prostředí existují 4 metody. Je to na základě HTTP hlaviček, uživatelského prostředí, cookies a nebo předdefinovaného parametru. Určení na základě http hlaviček je implicitní konfigurace a je vytvořena, pokud není určeno jinak. Pro určení podle uživatelského prostředí se využívá třída *org.springframework.web.servlet.i18n.SessionLocaleResolver*, pro určení podle cookies třída *org.springframework.web.servlet.i18n.CookieLocaleResolver*. V aplikaci jsem využil poslední možnost podle předefinovaného parametru. Konfigurace vypadá následovně:

```
<bean id="localeResolver"
      class="org.springframework.web.servlet.i18n.FixedLocaleResolver">
      <property name="defaultLocale" value="cs_CZ" />
</bean>
```

Tato konfigurace je zvolena z důvodu určení aplikace pouze pro českou veřejnost. V této verzi se zde neuvažuje o vícejazyčné podpoře, ale pro snazší rozšiřitelnost jsou zde texty umístěné také ve zprávách. Předdefinovaný parametr *"cs_CZ"* určuje, že zprávy se hledají v souboru „`messages_cs_CZ.properties`“. Pokud by se zvolila detekce podle uživatelského prostředí a na aplikaci se někdo podíval z anglického prostředí, zprávy by se hledaly v souboru „`messages_en_US.properties`“. Pro vypsání zprávy v zobrazovacím JSP souboru se musí vložit následující řádek pro import spring tagů:

```
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
```

A pro vypsání zpráv se pak kamkoli v souboru umístí tag:

```
<spring:message code="kód_zprávy"/>
```

Odesílání emailů

Jednou z velkých předností springu je snadné odesíláním emailů. V obyčejné Javě se dá mailem poslat prostý text, ale když je potřeba něco víc, je to velmi náročné. Se springem stačí v konfiguraci nastavit MailSender bean, která se využívá k samotnému odesílání (třída *org.springframework.mail.javamail.JavaMailSenderImpl*). Tam se zadají údaje o emailovém účtu. Poté se musí vytvořit vlastní tzv. MailManager třída, která uchovává jednotlivé části jedné emailové zprávy. Jsou to položky předmět, adresa pro koho, tělo zprávy atd. Dále obsahuje i metodu `sendEmail,` ve které pomocí MailSendru email odešle. Konfigurace pro oba objekty s příkladem konfigurace pro gmail vypadá následovně:

```
<bean id="IRezervMailManager"
   class="dip.hosna.utils.RezervMailManager">
         <property name="mailSender" ref="mailSender"/>
         <property name="commonFrom" value="rezervacee@gmail.com" />
   </bean>

 <bean id="mailSender"
   class="org.springframework.mail.javamail.JavaMailSenderImpl">
         <property name="username" value="rezervacee@gmail.com" />
         <property name="password" value="nějaké_heslo" />
         <property name="javaMailProperties">
         <props>
           <prop key="mail.smtp.user">rezervacee@gmail.com</prop>
           <prop key="mail.smtp.password">nějaké_heslo</prop>
           <prop key="mail.smtp.host">smtp.gmail.com</prop>
               <prop key="mail.smtp.port">587</prop>
           <prop key="mail.smtp.auth">true</prop>
           <prop key="mail.smtp.starttls.enable">true</prop>
          </props>
         </property>
   </bean>
```

Samotné odeslání emailu v kódu může vypadat například takto:

```
    try {

    MimeMessage mimeMessage = mailSender.createMimeMessage(  );
    MimeMessageHelper helper = new MimeMessageHelper( mimeMessage, "UTF-8" );

          helper.setTo( to );
          helper.setFrom( getCommonFrom(  ) );
          helper.setSubject( subject );
          helper.setText( body );
          mailSender.send( mimeMessage );

      } catch ( MessagingException expc ) {

          log.warn( expc.getMessage(  ) );
     }
```

Zde ji vidět, že odeslání emailu, kde tělo není prostý text, ale MimeMessage, je opravdu snadné. Pro usnadnění práce se využívá třída `org.springframework.mail.javamail.MimeMessageHelper`.

Tiles 2

Je šablonovací systém, který ve spojení se springem překládá logické názvy pohledu na pohledy definované ve svých šablonách. Pro napojení na spring se musí definovat Tiles jako *viewResolver*, kterému se předávají logické názvy pohledů, jak jsem již popsal v životním cyklu požadavku. Definice vypadá následovně:

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="viewClass">
        <value>
            org.springframework.web.servlet.view.tiles2.TilesView
        </value>
    </property>
</bean>
```

Dalším krokem je definovat umístění konfiguračního *.xml souboru pro Tiles, kde jsou definovány jednotlivé pohledy. V aplikaci je konfigurace uložena v souboru tiles.xml a její definice pro spring vypadá následovně:

```
<bean id="tilesConfigurer"
    class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
    <property name="definitions">
        <list>
            <value>/WEB-INF/tiles.xml</value>
        </list>
    </property>
</bean>
```

Konfigurační soubor je xml soubor, má kořenový elementem `<tiles-definitions>`, ve kterém se nacházejí definice jednotlivých pohledů v elementech `<definition>`. Výhodou těchto definic je možnost dědění. Princip vysvětlím na příkladu :

```
<definition name="vendorIndex" template="/WEB-INF/jsp/default/vendor/index.jsp">
        <put-attribute name="title" value="rezervace-vendor" />
        <put-attribute name="meta"
              value="/WEB-INF/jsp/default/vendor/meta.jsp" />
        <put-attribute name="header"
              value="/WEB-INF/jsp/default/vendor/header.jsp" />
        <put-attribute name="menu"
              value="/WEB-INF/jsp/default/vendor/userMenu.jsp" />
        <put-attribute name="toolbar"
              value="/WEB-INF/jsp/default/vendor/toolbar.jsp" />
        <put-attribute name="footer"
              value="/WEB-INF/jsp/default/vendor/footer.jsp" />
        <put-attribute name="body"
              value="/WEB-INF/jsp/default/vendor/body.jsp" />
        <put-attribute name="courses"
              value="/WEB-INF/jsp/default/vendor/menuCourses.jsp" />
</definition>
```

Zde je definice základního pohledu pro poskytovatele označené jako *vendorIndex*. Tento pohled obsahuje svojí základní šablonu v podobě JSP souboru umístěného v *"/WEB-INF/jsp/default/vendor/index.jsp"* a 8 atributů v podobě jiných JSP souborů. Atributy a jejich použití jsou stejné jako include v obyčejných JSP souborech. To znamená, že v základním JSP souboru (šabloně) jsou vloženy tagy pro vložení daných atributů. Tag vypadá následovně:

```
<tiles:insertAttribute name="název_atributu"/>
```

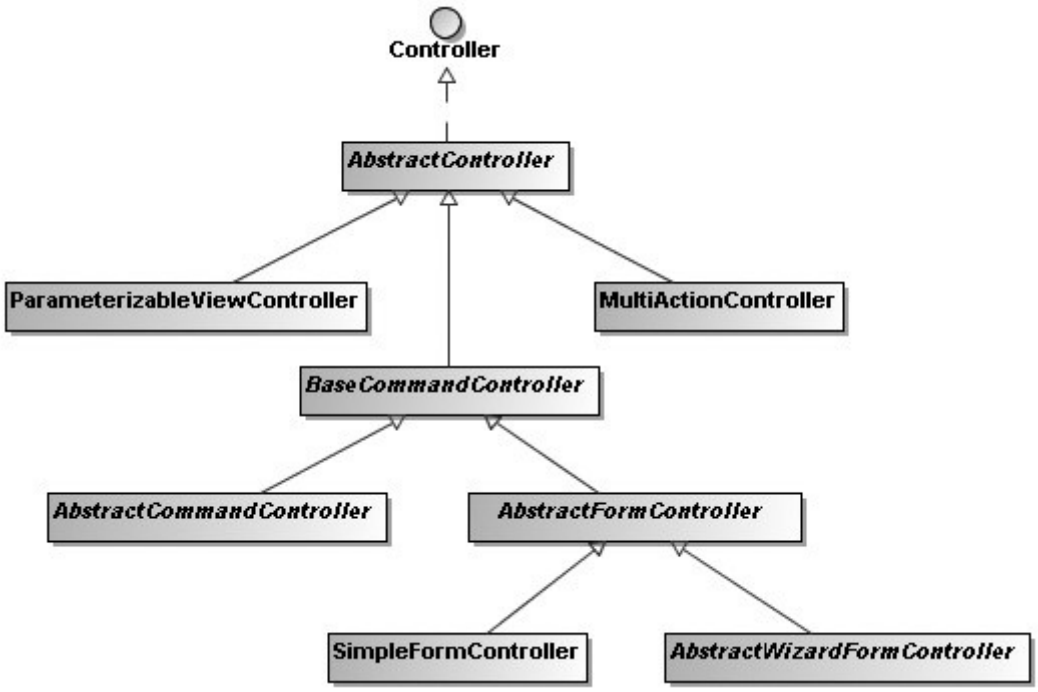
Tento základní soubor slouží jako šablona do které se vkládají jednotlivé prvky. Dědění těchto definic je dáno atributem `extends` v elementu `<definition>`, který označuje název pohledu od kterého se dědí. Zděděný pohled zdědí celou definici včetně atributů a jejich naplnění. Tento pohled pak může, ale nemusí do atributů přidělit jiné hodnoty. Může změnit například jen hodnotu atributu určujícího soubor s tělem stránky. Příklad od děděného pohledu:

```
<definition name="categories" extends="vendorIndex">
      <put-attribute name="title" value="Kategorie kurzů" />
      <put-attribute name="body"
            value="/WEB-INF/jsp/default/vendor/categories.jsp" />
</definition>
```

Tento zděděný pohled bude vypadat stejně jako šablona, jen změní titulek a vnitřní tělo stránky.

Mapování

Jednotlivé požadavky, jak jsem již uvedl, zpracovávají kontrolery. Kontrolerů je ve springu více druhů a existuje jich členitá hierarchie. Hierarchie kontrolerů je zobrazena na následujícím obrázku:



Každý tento kontroler má jiný účel a tomu odpovídající strukturu. Dříve se musely třídy pro kontrolery dědit od těchto jednotlivých typů a implementovat potřebné metody. Nyní je možno pro kontroler implementovat jakoukoli třídu a pouze jí označit anotací `@Controller`. Spring si pak sám detekuje k čemu kontroler slouží podle mapovacích anotací nad jednolitými metodami.

Dříve se mapování požadavků na jednotlivé kontrolery muselo zapisovat do konfigurace dispečerů.

Nyní pro nastavení mapování stačí použít k tomu určené anotace. Anotacemi pro mapování se mohou označit třídy, nebo jen metody. Pokud se adresou požadavku označí třída, tak se musí ještě označit metoda pro zpracování. Anotace pro mapování je `@RequestMapping`. K anotaci se připojuje závorka s parametry. Parametr `value` určuje adresu požadavku, kterou má kontroler zpracovat. Pokud je tento parametr v závorce jediný, nemusí se označovat. Stačí uvést v závorkách daný řetězec. Druhý možný parametr je `method`, který určuje jakou metodou má parametr přijít. Pokud není uveden, tak bude metoda zpracovávat požadavek ať přijde jakoukoli metodou. Příklad označení anotací pro mapování:

```
@RequestMapping(value ="/vendor/upravKategorii", method = RequestMethod.POST)
```

Metody pro zpracování požadavku musí mít návratový typ `String` a musí vracet logický název pohledu. Parametry metody mohou být typů `Map<String, Object>`, `BindingResult` a `HttpSession`. Mapa nese všechny objekty uložené v požadavku. `BindingResult`, slouží k uchovávání chyb nasbíraných za život požadavku. Atribut typu `HttpSession` nese aktuální session z prohlížeče a může se použít například k získáni uložených parametrů.

Pokud chceme například s požadavkem poslat objekt pro formulář, uložíme ho do mapy. Když se ale formulář s upraveným objektem odešle zpět, objekt nenalezneme v mapě, ale jako atribut modelu. Ten se získá tak, že se z něj udělá parametr metody požadovaného typu označený anotací `@ModelAttribute` s názvem objektu, pod kterým byl uložen při poslání na formulář. Celá definice metody může vypadat následovně:

```
@RequestMapping(value ="/mojeudaje", method = RequestMethod.POST)
public String editCustomerSave(@ModelAttribute("customer")
      Customers customer, BindingResult result, Map<String, Object> map,
                                                        HttpSession session)
```

Další možností je mít parametr v adrese. Zde se nemůže mapovat každá možná hodnotu zvlášť, ale musí se použít dynamická adresa. Parametr se z adresy dostane za pomoci anotace `@PathVariable` s názvem proměnné z adresy. Posléze se použije opět jako vstupní parametr metody. Příklad použití:

```
@RequestMapping(value ="/vendor/smazKurz-{courseId}", method=RequestMethod.GET)
public String deleteCourse(@PathVariable("courseId")String courseId,
                                                    Map<String, Object> map)
```

Nahrávání souborů

I zde spring nabízí nástroje, které tuto činnost velmi usnadní. Ve springu existují dohromady tři způsoby, jak nahrání souborů docílit. Já představím ten nejjednodušší a zároveň nejefektivnější způsob, který je použit i v aplikaci. Prvním krokem je definování beany s názvem *multipartResolver* v konfiguraci dispečeru. Jak jsem již zmínil v životním cyklu požadavku, tento název se musí dodržet. V aplikaci je tento objekt instance třídy *org.springframework.web.multipart.commons.CommonsMultipartResolver*. Jejím jediným využitým nastavením je *maxUploadSize*, které určuje maximální velikost nahrávaného souboru. Definice v aplikaci vypadá následovně:

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
            <property name="maxUploadSize" value="2000000"/>
</bean>
```

Jak vytvořit ve formuláři políčko pro výběr souboru bylo již uvedeno. Pak je jen potřeba, aby proměnná spojená s tímto políčkem byla typu `org.springframework.web.multipart.commons.CommonsMultipartFile`. Aby jí uměl formulář zpracovat doplníme do značky pro formulář k atributům `method`, `action`, `commandName` i `enctype` s hodnotou *multipart/form-data*. Na kontroler přijde v objektu z formuláře již nahraný obrázek a stačí ho metodou `image.transferTo(File file);` uložit do požadovaného souboru a serveru. Pro pomoc se zvolením názvu poskytuje soubor tohoto typu metodu `getOriginalFilename()`. Obrázky musí být pro možnost přístupu ze zobrazovacích JSP souborů umístěny ve složce „WebContent“. Jinak by na ně tyto soubory neviděly a nebylo by možné soubory zpětně zobrazit na stránkách aplikace.

Validátory

Když se v kontroleru získá objekt poslaný formulářem je třeba zjistit, zda jsou zadaná data validní. Aby se nemusela provádět kontrola v každé metodě, která zpracovává požadavek týkající se jednoho typu objektu, používají se validátory. Validátor je třída implementující rozhraní `org.springframework.validation.Validator`, které určuje implementaci metody `supports(Class<?> arg)` a `validate(Object arg0, Errors errors)`. První slouží k identifikaci, zda je objekt k validaci požadovaného typu a druhá k samotné validaci.

K ulehčení implementace validace spring poskytuje nápomocné metody ze třídy

`org.springframework.validation.ValidationUtils`. Těmto metodám se jako parametry vloží kolekce s chybami, název atributu ke kontrole a kód zprávy, která má být vypsána jako oznámení chyby. V kontroleru se při zpracování požadavku zavolá validační metoda s objektem z formuláře. Jako druhý parametr validační metody (kolekce `errors`) se zadává parametr `BindingResult result` z metody zpracovávající požadavek. Zda byla validace úspěšná se pak zjistí zavoláním metody `result.hasErrors()`, která vrátí **`true`** pokud se nějaké problémy vyskytly.

Nemapované soubory

Při tvorbě pohledových souborů vzniká problém při pokusu zobrazit soubor s obrázkem nebo zařadit soubor s css styly. Problém spočívá v tom, že v obou případech vložení souboru obsahuje odkaz na tento soubor. Tento odkaz, tak jako všechny ostatní, se snaží zpracovat definovaný dispečer. Je zapotřebí vytvořit výjimky, které nemá mapovat. To se provádí definicí implicitního serverletu, který bude potřebné výjimky zpracovávat. Ten pak umožňuje soubory normálně používat. Definice ze souboru „web.xml“ vypadá následovně:

```
 <servlet-name>default</servlet-name>
       <url-pattern>*.png</url-pattern>
       <url-pattern>*.js</url-pattern>
       <url-pattern>*.css</url-pattern>
       <url-pattern>*.JPG</url-pattern>
       <url-pattern>*.jpg</url-pattern>
</servlet-mapping>
```

Z příkladu je patrné, že adresy končící na požadované přípony bude zpracovávat definovaný implicitní servlet a jsou vyjmuty ze zpracování servletem `spring`.

Zabezpečení

V aplikaci jsou tři typy uživatelů, pro každého je jiné rozhraní, pod jinou adresou. Aby se rozhraní rozlišily, obsahuje adresa rozhraní pro poskytovatele navíc `/vendor/` a pro provozovatele `/operator/`. To komplikuje odkazy, jelikož musí obsahovat i tuto část a ta se v každém dalším odkazu duplikuje. Proto je v aplikaci vytvořen pro poskytovatele i pro provozovatele jeden kontroler, který přesměruje požadavky na správný cíl. Adresa rozhraní pro přihlášeného zákazníka neobsahuje navíc žádnou část. V aplikaci se musí ošetřit, aby se na tyto adresy nemohl dostat každý, ale jen ti přihlášení.

Spring nabízí pro zabezpečení celý balík zaměřený na tuto problematiku. Název balíku je **Spring Security** a obsahuje velké množství funkcí zaměřených na tuto problematiku. Balík není zařazen přímo v balíku spring, ale musí se stáhnout zvlášť. Zprovoznit základní přihlašování i s několika

uživatelskými rolemi a možností pamatování uživatele je velmi jednoduché. Jen se musí využívat entity přímo z tohoto balíku a počítá se s jedním zdrojem uživatelů, kteří budou mít jen jiný atribut určující roli. Pokud je potřeba udělat to jinak, je projení s tímto balíkem těžší.

Z těchto důvodů tuto možnost ve své aplikaci nevyužívám. Pro každou uživatelskou roli je v aplikaci zvláštní přihlašovací formulář na jiné adrese. Pro každý z těchto formulářů je zvláštní kontroler, který ověří zadaná data v databázi. Hesla se v databázi ukládají v zašifrované formě. Po úspěšném přihlášení se uloží do HttpSession atribut s danou identitou. Při pokusu o přístup na adresu přístupnou až po přihlášení poskytovatele, nebo provozovatele se vždy v interceptoru ověřuje, zda je uživatel korektně přihlášen pomocí atributu v HttpSession. Pokud tomu tak není je přesměrován na příslušnou přihlašovací stránku. Adresa pro ověření přihlášení u těchto dvou rolí je snadno rozeznána díky vloženému řetězci. U přihlášeného zákazníka je to horší, zde se provádí ověření až v kontroleru u každého konkrétního požadavku.

Formuláře

Pro tvorbu webových formulářů spring nabízí vlastní sadu značek pro usnadnění tvorby. V JSP souboru, kde se tyto značky použijí, je třeba je definovat přidáním této řádky:

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
```

Nyní budou všechny značky pro formulář začínat prefixem `<form:`. Základní značkou pro formulář je `<form:form>`, což je obdoba značky `<form>`. Tento tag obsahuje atributy `method` a `action`, které mají stejnou funkci jako v kalsickém HTML. Navíc je zde atribut `commandName`, který určuje název objektu, který má formulář zobrazit. Tento objekt se musí vložit v kontroleru do mapy s objekty pod daným názvem.

Poté se do formuláře vloží značky pro zobrazení jednotlivých atributů objektů, které je třeba zobrazit. Značky pro jednotlivé prvky formuláře jsou obdobné jako v klasickém HTML.
`<form:input` zobrazí textbox (input type=“text“)
`<form:password` zobrazí textbox se zakrytými znaky (input type=“password“)
`<form:textarea` zobrazí textové pole (textarea)
`<form:radiobutton` zobrazí jedno ze skupiny políček pro výběr, kde je možné vybrat pouze jedno (input type=“radio“)
`<form:checkbox` zobrazí políčko pro výber (input type=“checbox“)

`<form:checkboxes` zobrazí skupinu políček pro výběr. Musí se zadat kolekce hodnot do atributu `items`

`<form:select` zobrazí dropdown menu pro výběr hodnoty (select). Musí se opět zadat kolekce s hodnotami do atributu `items`. Je na výběr z více možností zápisu.

První možnost (příklad ze souboru „registrace.jsp“):

```
<form:select items="${years}" path="year_of_birth" />
```

Druhá možnost (příklad ze souboru „rnewCrmStatus.jsp“)::

```
<form:select  path="discounts.id" >
      <form:options items="${discounts}" itemValue="id" itemLabel="name" />
</form:select>
```

Třetí možnost (příklad ze souboru „toolbar.jsp“)::

```
<form:select path="selected" onchange="this.form.submit();">
  <form:option value="0">Vyberte</form:option>
  <form:options items="${select.categories}" itemValue="id" itemLabel="name" />
</form:select>
```

`<form:hidden` nezobrazí nic, ale vypíše v kódu hodnotu, aby při odeslání formuláře daný objekt hodnotu obsahoval a neztratila se. Typickým příkladem hodnoty, kterou potřebujeme, ale nechceme jí zobrazit, je např. id objektu z databáze.

Chybí zde značka pro položku sloužící k nahrávání souborů. Pro tuto značku nemá Spring žádnou alternativu. Lze pouze využít značky `<form:input` a dopsat do ní atribut `type` s hodnotou *`file`*. Tato je řešení pro nahrávání souborů v aplikaci.

Každá položka se musí přiřadit k nějakému atributu daného objektu. Přiřazení se provádí skrze atribut `path`, do jehož hodnoty se zadá název daného atributu objektu. K položkám ve formuláři lze přiřadit i popis pomocí značky `<form:label> </form:label>`, také s atributem `path`. Ke každé položce je možnost vypsat i chybové hlášení pokud nějaké je. K tomu slouží značka `<form:errors` také s atributem `path`. Vypsání jedné řádky formuláře v tabulce může vypadat následovně (příklad ze souboru „newApCategory.jsp“):

```
<tr>
    <td>
        <form:label path="description">
              <spring:message code="label.description"/>
        </form:label>
    </td>
    <td><form:input path="description" /></td>
    <td>
        <form:errors path="description" cssClass="error" />
    </td>
</tr>
```

Atribut `cssClass` označuje třídu css stylů, která má být aplikována.

Zobrazení kolekcí položek

Pokud objekt ve formuláři obsahuje dynamickou kolekci položek vzniká problém s jejich zobrazením. Ten spočívá v přiřazení položky formuláře ke konkrétnímu atributu v objektu. Za tímto účelem se používá značka `<spring:bind` v kombinaci s atributem `varStatus` při procházení kolekce. První značka je z kolekce značek definované vložením řádku :

```
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
```

Druhá je atribut značky `<c:forEach` z kolekce definované vložením řádku:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

`<spring:bind` má jen jeden parametr a to `path`, který určuje, ke které proměnné se vztahuje obsah této značky. Ohraničuje kód stahující se k jedné proměnné. V jejím těle lze umístit políčko formuláře, nebo jen výpis dané proměnné. Nyní se musí ještě zjistit index daného prvku v kolekci.

Příkaz `<c:forEach` se v JSP souborech používá pro procházení kolekcí. Příkaz může mít tři atributy:

`items` slouží k zadání kolekce s daty,

`var` slouží k označení právě procházeného prvku z kolekce,

`varStatus` slouží k označení statusu právě procházeného prvku (obsahuje index prvku).

Příklad použití ze souboru „coursStructure.jsp“ :

```
<c:forEach items="${item.elements.dropDownMenuses}" varStatus="item">
        <tr>
         <td><spring:message code="label.course.item"/>${item.index}</td>

            <td>

            <spring:bind path="elements.dropDownMenuses[${item.index}].value">

              <input type="text" name="<c:out value="${status.expression}"/>"
                id="<c:out value="${status.expression}"/>"
                value="<c:out value="${status.value}"/>" />

            </spring:bind>
            </td>
             <td></td>
             <td></td>
       <tr>

      </c:forEach>
```

Příklad demonstruje použití pro výpis dynamické kolekce textboxů pro řetězce uložené v kolekci `item.elements.dropDownMenuses`. Příklad se používá při přidání nové položky do struktury kurzu. Pokud je nová položka typu drop dow menu, naplní se v ní, po stisknutí příslušného tlačítka a zpracování kontrolerem, kolekce položek menu zadaným počtem položek. Příklad zobrazí tuto kolekci o dynamické délce. Z příkladu je také vidět, že při použití parametru `varStatus` se stává parametr `var` nepotřebným. Podobný princip je použit i pro přidání položek do struktury objednávky, zákazníka, jejich následné zobrazení, a také při přidávání vyučovacích hodin ke kurzu.

Při odesílání objektu s dynamickou velikostí kolekce vzniká problém. Jde o to, že při odeslání formuláře s objektem se vytváří nový objekt a plní se daty z formuláře. Při vytvoření objektu je kolekce prázdná a formulář do ní nemůže vložit prvek na určitou pozici. Kdyby se o to pokusil, pokus skončí vyhozením výjimky. Je možné toto řešit naplněním kolekce prázdnými daty při vytvoření objektu. V takovém případě by formulář mohl zobrazit právě prázdné položky, což je nežádoucí. Z tohoto důvodu v aplikaci v objektech s dynamickými kolekcemi existuje metoda, která kolekci prochází a vymaže z ní prázdné položky, nebo položky přesahující nastavenou velikost kolekce. Metoda se volá při nastavení požadované velikosti kolekce. Nebo je volána z kontroleru při zpracování objektu, pokud není velikost předem známa.

Příkladem může být objekt `CoursesCourses`, který representuje kurz. Ten obsahuje kolekci `coursesCoursesValueses`, která obsahuje hodnoty položek přidaných do struktury kurzu. Kolekce se načte z databáze s objektem a pošle na formulář. Při poslání zpět není nikde řečeno kolik prvků má kolekce mít. Před zpracováním objektu se proto zavolá metoda `checkValues()`, která přesune prvky do jiné dočasné kolekce a vrátí tam jen ty, u kterých se proměnná

`coursesCoursesStructure` nerovná **`null`**. Tím z kolekce zmizí prázdné objekty vložené při vytvoření objektu. Stejná metoda se volá i před posláním nového objektu do formuláře, aby formulář nezobrazil, nežádoucí prázdné položky. Před jejím zavoláním se musí do kolekce vložit základní hodnoty pro operátorem definované volitelné položky struktury kurzu.

Zobrazení dynamických položek

Jak jsem již zmínil, entity kurz, zákazník a objednávka budou obsahovat operátorem přidané položky. Položky mohou být z předem definované množiny textfield, textarea, drop down menu (dále jen select) a checkbox. Každá z těchto položek má v html i jsp jinou značku, proto je jejich zobrazení komplikované. Textfield a chcekbox se shodují v počáteční značce `<input` a atributu `type`, hodnoty atributu se již liší. Oba obsahují i atribut `value`, ale u prvního jeho hodnota znamená text v poli a u druhého určuje, zda se má zařadit do odeslaného požadavku. U chceckboxu je tedy hodnota vždy **`true`**. U druhé dvojice typů se liší počáteční a koncová značka. U typu textarea je text v ní vypisován mezi značky. Select má také úplně jiné značky, než všechny ostatní a k tomu obsahuje kolekci hodnot. Hodnota uložená v databázi znamená, jaká z položek je vybrána. V aplikaci vypadá kód pro jejich univerzální zobrazení takto (příklad ze souboru newCourse.jsp):

```
<c:forEach items="${course.coursesCoursesValueses}" varStatus="valueS" var="value">
<tr>
<spring:bind path="coursesCoursesValueses[$
                                  {valueS.index}].coursesCoursesStructure.columnNr">
        <input type="hidden"
              name="<c:out value="${status.expression}"/>"
              id="<c:out value="${status.expression}"/>"
              value='<c:out value="${value.coursesCoursesStructure.columnNr}"/>'
        />
</spring:bind>

<spring:bind path="coursesCoursesValueses[${valueS.index}].values">
  <td>
       <c:out value="${value.coursesCoursesStructure.name}"/> :
  </td>
  <td>
  <<c:out value="${value.coursesCoursesStructure.elements.elementType.tag}"/>
     name="<c:out value="${status.expression}"/>"
     id="<c:out value="${status.expression}"/>"

  <c:if  test="${!empty value.coursesCoursesStructure.elements.elementType.type}">
       type="${value.coursesCoursesStructure.elements.elementType.type}"

  <c:if  test="${value.coursesCoursesStructure.elements.elementType.id==2}">
         value="true"
         <c:if  test="${value.values==true}">
               checked="checked"
         </c:if>
  </c:if>

  <c:if  test="${value.coursesCoursesStructure.elements.elementType.id!=2}">
       value="<c:out value="${status.value}"/>"
  </c:if>
   /
  </c:if>>

  <c:if  test="${empty value.coursesCoursesStructure.elements.elementType.type}">
       <c:out value="${status.value}"/>

<c:forEach items="${value.coursesCoursesStructure.elements.dropDownMenuses}"
                                                                    varStatus="vS" var="v">

         <option value="${v.id.valueNr}"
              <c:if  test="${value.values==v.id.valueNr}">
                     selected
              </c:if>
            >
               ${v.value}
         </option>
     </c:forEach>
       </<c:out value="${value.coursesCoursesStructure.elements.elementType.tag}"/>  >

  </c:if>

  </spring:bind>

  </td>
       <td></td><td></td>
  </tr>

  </c:forEach>
```

Kód začíná cyklem, který prochází kolekci `course.coursesCoursesValueses`, kde jsou tyto položky uloženy. Jako první se do skryté proměnné uloží hodnota `value.coursesCoursesStructure.columnNr`, která v databázi representuje primární klíč v relaci s

položkami. Pak se zobrazí první část položky, což je její název, to je pro všechny typy shodné. Dalším krokem je zobrazení otvírací závorky samotné značky a názvu značky uložené v proměnné `value.coursesCoursesStructure.elements.elementType.tag`. Důležité je, aby byl `tag` vypsán hned za závorku, jinak by prohlížeč nezobrazil položku korektně.

Další krok zobrazení je test, zda objekt obsahuje neprázdnou proměnnou `value.coursesCoursesStructure.elements.elementType.type`, což je případ pro textfield a checkbox. Pro další dva typy v databázi uložen není. Pokud není prázdný, typ se vypíše. V té samé podmínce je také test, zda se nejedná o typ položky 2, která representuje checkbox. Pokud ano, vypíše se řetězec `value="true",` a také následuje test, zda není hodnota položky **`true`**. Když je i druhá podmínka splněna, je vypsán řetězec `checked="checked"`, aby byl checkbox zobrazen jako vybraný. Další kontrola v pořadí je nahrazení else větve podmínky kontrolující typ číslo 2, jelikož JSTL nepodporuje else. Při splnění je jasné, že se jedná o položku typu textfield a její hodnota se vypíše do hodnoty atributu `value` pomocí příkazu `<c:out value="${status.value}"/>`. Posledním krokem v dané podmínce, které vyhovují pouze typy textfield a checkbox, je vypsání zpětného lomítka. Posléze se vypíše už mimo podmínky pro všechny společná uzavírací závorka.

Nyní jsou již vypsané celé položky prvních dvou typů. Další pokračování kódu se týká již jen typu textarea a select. Opět je zde podmínka na prázdnost proměnné reprezentující atribut type, což je obdoba else větve minulé podmínky testující opak. Hned po ní následuje vypsání hodnoty položky pro případ textové arey příkazem `<c:out value="${status.value}"/>`. Následuje vypsání kolekce `value.coursesCoursesStructure.elements.dropDownMenuses`, ve které má položky jen typ select, proto se pro ostatní typy nevypíše nic. Uvnitř cyklu se vypisují jednotlivé `<option>` položky a testuje se shodnost čísla položky s hodnotou z databáze pro vypsání řetězce `selected`, který určí, že tato položka bude ve zobrazeném selectu vybraná. Poslední kus tohoto kódu vypíše uzavírací značku.

Závěr

Pro potřeby diplomové práce bylo v aplikaci vyvinuto základní funkční jádro. Toto jádro obsahuje rozhraní pro všechny tři uživatelské role. Každé rozhraní obsahuje základní funkce nezbytné pro danou uživatelskou roli, jako je pro provozovatele správa jednotlivých poskytovatelů nebo pro zákazníka vybrání a objednání dané služby. Každé rozhraní je zabezpečené, aby se do něj nebylo možné dostat bez úspěšného přihlášení přihlášení. Po domluvě se zadavatelem své práce budu ve vývoji aplikace pokračovat v domluvených iteracích. Dokončení vývoje je odhadováno na polovinu léta. Aplikace byla navržena, tak že není problém přidat další požadovanou funkcionalitu.

Zdroje

1. http://www.javabeat.net/tips/230-spring-framework-form-tags.html
2. http://morosystems.cz/java/spring/
3. http://www.podolinsky.cz/2010/09/jak-vytvorit-java-web-aplikaci-s-podporou-spring-a-jpa/
4. http://viralpatel.net/blogs/2010/06/spring-3-mvc-create-hello-world-application-spring-3-mvc.html
5. http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/validation.html
6. http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html#configuration-optional
7. http://eclipse.dzone.com/news/springide-using-spring-eclipse
8. http://vyvojari.oxyonline.cz/spring-framework-aplikacni-server-jinak
9. http://www.mkyong.com/struts/struts-tiles-framework-example/
10. http://www.vaannila.com/spring/spring-tiles-integration-1.html
11. http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/validation.html
12. http://www.mkyong.com/spring-mvc/spring-mvc-form-handling-annotation-example/
13. http://www.mularien.com/blog/2008/07/07/5-minute-guide-to-spring-security/
14. http://static.springsource.org/spring-security/site/docs/3.1.x/reference/springsecurity.html
15. http://www.mkyong.com/spring/spring-aop-examples-advice/
16. http://maestric.com/doc/java/spring/hello_world
17. http://www.theserverside.com/news/1363858/Introduction-to-the-Spring-Framework
18. http://www.javabeat.net/tips/232-spring-message-and-theme-tags-ltspringmes.html
19. http://www.benmccann.com/dev-blog/sample-log4j-properties-file/
20. http://springcert.sourceforge.net/2.5/9-study-web-app.html

21. http://www.wikihow.com/Generate-Hibernate-Pojo-Classes-from-DB-Tables
22. http://www.javaworld.com/javaworld/jw-10-2004/jw-1018-hibernate.html
23. http://j2eereference.com/2011/01/many-to-many-mapping-in-hibernate/
24. http://www.mkyong.com/spring/maven-spring-hibernate-mysql-example/
25. http://www.stevideter.com/2008/12/07/saveorupdate-versus-merge-in-hibernate/
26. http://blog.xebia.com/2009/03/jpa-implementation-patterns-saving-detached-entities/
27. http://solutionsfit.com/blog/2007/12/04/extending-query-by-example-through-annotation/
28. http://www.roseindia.net/hibernate/hibernateidgeneratorelement.shtml

29. LINWOOD, Jeff; MINTER, Dave. *Beginning Hibernate : An itroduction to persistence using Hibernate 3.5*. 2nd Edition. USA : Apress, 2010. 379 s. ISBN 978-1-4302-2851-6.

30. MACHACEK, Jan, et al. *Pro Spring 2.5 : The Spring Framework 2.5 release reflects the state of the art in both the Spring Framework and in enterprise JavaTM frameworks as a whole. A guidebook to this critical tool is necessary reading for any conscientious JavaTM developer. —Rob Harrop, author Pro Spring*. Vyd. 1. USA : Apress, 2008. 890 s. ISBN 978-1-4302-0506-7.

31. MATULÍK, Petr; PÁRAL, Tomáš. *MoroSystem, s.r.o.* [online]. 2007 [cit. 2011-04-19]. Moderní JEE™ technologie a nástroje. Dostupné z WWW: <http://morosystems.cz/java/>.

32. ETIKALA, Abhilash. Javabeat [online]. Wed Nov 10th, 2010 [cit. 2011-04-19]. Spring Framework FORM Tags. Dostupné z WWW: <http://www.javabeat.net/tips/230-spring-framework-form-tags.html>.

33. PODOLINSKÝ, Martin. Martin Podolinský : Freelance Software Engineer and Consultant [online]. September 08, 2010 [cit. 2011-04-19]. Jak vytvořit java web aplikaci s podporou Spring a JPA . Dostupné z WWW: <http://www.podolinsky.cz/2010/09/jak-vytvorit-java-web-aplikaci-s-podporou-spring-a-jpa/>.

34. PATEL , Viral. ViralPatel [online]. June 22, 2010 [cit. 2011-05-19]. Spring 3 MVC: Create Hello World application in Spring 3.0 MVC. Dostupné z WWW: <http://viralpatel.net/blogs/2010/06/spring-3-mvc-create-hello-world-application-spring-3-mvc.html>.

35. Springsource COMUNITY [online]. 2010 [cit. 2011-04-19]. 5. Validation, Data Binding, and Type Conversion Prev Part III. Core Technologies. Dostupné z WWW: <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/validation.html>.

36. Mkyong.com [online]. August 20, 2010 [cit. 2011-04-19]. Spring MVC form handling annotation example. Dostupné z WWW: <http://www.mkyong.com/spring-mvc/spring-mvc-form-handling-annotation-example/>.

37. HIBERNATE : CommunityDocumentation [online]. June 24, 2009 [cit. 2011-05-19]. Chapter 3. Configuration. Dostupné z WWW: <http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html#configuration-optional>.

38. VannNila [online]. 2011 [cit. 2011-05-19]. Spring Tiles Integration. Dostupné z WWW: <http://www.vaannila.com/spring/spring-tiles-integration-1.html>.

39. Jerome Jaglale [online]. 2008 [cit. 2011-05-19]. Spring MVC Fast Tutorial: Hello World. Dostupné z WWW: <http://maestric.com/doc/java/spring/hello_world>.

40. *Javabeat* [online]. 30.11.2010 [cit. 2011-04-19]. Spring MESSAGE and THEME Tags (and). Dostupné z WWW: <*http://www.javabeat.net/tips/232-spring-message-and-theme-tags-ltspringmes.html*>.

41. *Spring study note* [online]. 2010 [cit. 2011-04-19]. Using Spring in Web Applications. Dostupné z WWW: <*http://springcert.sourceforge.net/2.5/9-study-web-app.html*>

42. *Wiki how* [online]. 20. 10. 2010 [cit. 2011-04-19]. How to Generate Hibernate Pojo Classes from DB Tables. Dostupné z WWW: <http://www.wikihow.com/Generate-Hibernate-Pojo-Classes-from-DB-Tables>.

43. *J2ee reference* [online]. 27. 1. 2011 [cit. 2011-04-19]. Many to Many mapping in hibernate . Dostupné z WWW: <http://j2eereference.com/2011/01/many-to-many-mapping-in-hibernate/>

44. *Solutions fit* [online]. 4. 12. 2007 [cit. 2011-04-19]. Extending query by example through annotation. Dostupné z WWW: <http://solutionsfit.com/blog/2007/12/04/extending-query-by-example-through-annotation/>

45. *Rose india* [online]. 12. 3. 2008 [cit. 2011-04-19]. Understanding Hibernate element. Dostupné z WWW: <http://www.roseindia.net/hibernate/hibernateidgeneratorelement.shtml>.