Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství

Diplomová práce

# Webová aplikace Šoupátka

***Bc. Lukáš Kmoch***

Vedoucí práce: Ing. Martin Skřenek

4. května 2015

# Poděkování

Chtěl bych poděkovat Ing. Martinu Skřenkovi za vedení této práce, stejně tak bych chtěl poděkovat Miroslavu Bradičovi za konzultace ohledně implementace jednotlivých částí. V neposlední řadě děkuji rodičům a Barboře Čadyové za obrovskou trpělivost, kterou se mnou měli při psaní práce a za pomoc při korektuře práce.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. května 2015 .....

### Odkaz na tuto práci

Kmoch, Lukáš. *Webová aplikace Šoupátka.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

# Abstrakt

Diplomová práce Šoupátka se zabývá návrhem a implementací webové aplikace s možností dynamického generování databázových tabulek za běhu aplikace. Práce ukazuje, jaké jsou možnosti řešení takovéhoto požadavku v technologii ASP.NET MVC a následně popisuje implementaci jednoho z navržených řešení.

**Klíčová slova** Šoupátka, ASP.NET MVC, Entity Framework, AssemblyBuilder, dynamicky generované tabulky, scaffolding

# Abstract

Master's thesis Web application Šoupátka deals with designing and implementation of a web application with the possibility to generate database tables dynamically at the application runtime. The thesis shows the way how to solve such a requirement in ASP.NET MVC and how to implement one of the proposed solutions.

**Keywords** Šoupátka, ASP.NET MVC, Entity Framework, AssemblyBuilder, dynamically generated tables, scaffolding

# Obsah

# Seznam obrázků

# Úvod

Webová aplikace Šoupátka je aplikace, která uživateli umožňuje za běhu aplikace vytvořit rozšiřitelnou datově řízenou webovou aplikaci. Šoupátka se skládají ze dvou částí nazvaných Návrhář a Aplikace[1]. Návrhář slouží k definici výsledné aplikace, tj. k definici jaké bude výsledná aplikace obsahovat modely, řadiče a pohledy. Druhá část Aplikace je vygenerovaná na základě nastavení, které uživatel provedl v Návrháři a obsahuje tak již výslednou aplikaci. Šoupátka tedy podporují, podobně jako například ASP.NET Dynamic Data, scaffolding. Scaffolding je technika podporovaná MVC frameworky pro automatické generování kódu na základě databázového schématu. Pomocí vygenerovaného kódu pak umožňuje aplikace vytvářet, číst, upravovat a samozřejmě smazat databázové záznamy. Uživatel Šoupátek tento kód může generovat pomocí jednoduchého grafického prostředí a vytvořit si tak vlastní aplikaci bez jakýchkoliv znalostí zdrojového kódu. Šoupátka se už postarají o veškerou další práci, jako je generování databázového schématu.

Jednoduchým příkladem využití aplikace je uživatel Franta, který pro svoji firmu již delší dobu hledá systém, do kterého by mohl vkládat informace o svých zaměstnancích. Informace jako jsou například: kdy uživatel nastoupil do práce, na jakém dělá projektu a mnoho dalších. Bohužel ale u veškerých aplikací narazil na problém, že neobsahují možnost uložit si všechny informace, které potřebuje. Smutný Franta tak začal uvažovat o tom, nechat si vytvořit vlastní aplikaci. Když se ale dozvěděl, jakou částku vytvoření takové aplikace stojí, málem se už rozhodl uchovávat si veškeré informace jednoduše pomocí Excelovské tabulky. V té době ho naštěstí jeden ze zaměstnanců upozornil na aplikaci Šoupátka. Franta se tak do aplikace zdarma zaregistroval a zjistil, že je schopný během pár minut vytvořit vlastní aplikaci, která mu umožní uchovávat všechny potřebné informace a to navíc v přehledné formě s možností vyhledávání, řazení atd.

[1]Pokud dále v této práci narazíte na zmínku o Aplikaci s velkým prvním písmenem, je tím myšlena právě tato specifická část aplikace. Nejde tedy o typografickou chybu.

Kapitola 1

# Cíl práce

Cíl této práce se skládá z několika částí. Prvním cílem práce je navrhnout architekturu aplikace v ASP.NET MVC tak, aby na ní bylo možné vytvořit výslednou webovou aplikaci neboli Návrhář. Výsledná architektura by měla splňovat požadavky ze zadání této práce a zároveň umožnit snadné vytvoření a postupné rozšiřování funkcionality aplikace. Druhým cílem práce je na výsledné architektuře vytvořit Návrhář. Třetím a hlavním cílem je navrhnout a implementovat součást aplikace, která se bude starat o vygenerování výsledné aplikace z definice vytvořené v Návrháři. Důležitým aspektem při implementaci aplikace je vzít v potaz další rozšíření, která by v aplikaci měla být v budoucnu implementována, ale nejsou součástí této diplomové práce.

Při plnění tohoto cíle je diplomová práce podrobněji zaměřena na analýzu dalších speciálních požadavků na aplikaci. Mezi tyto požadavky patří:

- Přidávání modelů, pohledů a řadičů z grafického prostředí aplikace.
- Možnost změn vlastností modelů.
- Vhodné grafické komponenty pro úpravu údajů.
- Možnost konfigurace rozložení grafických komponent v pohledech.
- Dynamické generování položek v menu.

O těchto požadavcích je více napsáno v sekci 2.1

Kapitola 2

# Analýza

## 2.1 Požadavky

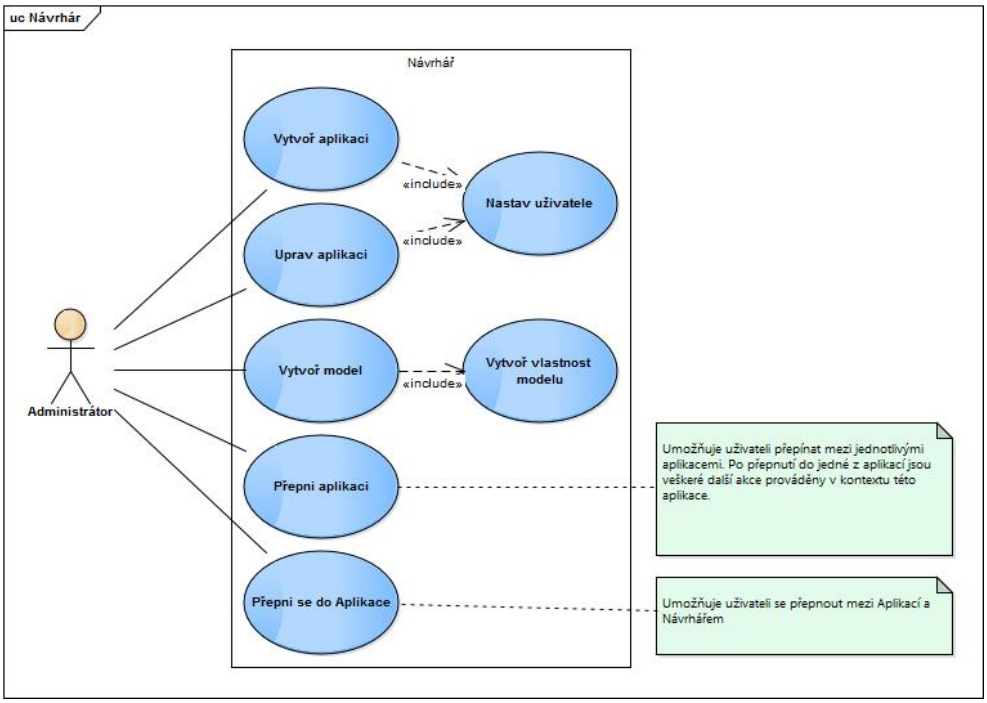
### 2.1.1 Zadání

Ze zadání vyplývají tyto požadavky:

- Přidávání modelů, pohledů a řadičů z grafického prostředí aplikace.
- Možnost změn vlastností modelů.
- Vhodné grafické komponenty pro úpravu údajů.
- Možnost konfigurace rozložení grafických komponent v pohledech.
- Dynamické generování položek v menu.
- Implementace v ASP.NET MVC.

### 2.1.2 Funkční požadavky

- Správa aplikací.
- Správa modelů.
- Automatická aktualizace databáze.
- Verzování vytvořených assembly.
- Rychlé přepínání mezi aplikací a modelem.
- Možnost pozvat uživatele do aplikace.
- Lokalizace do českého a anglického jazyku.




Obrázek 2.1: Případ použití Návrháře

## 2.2 Use case

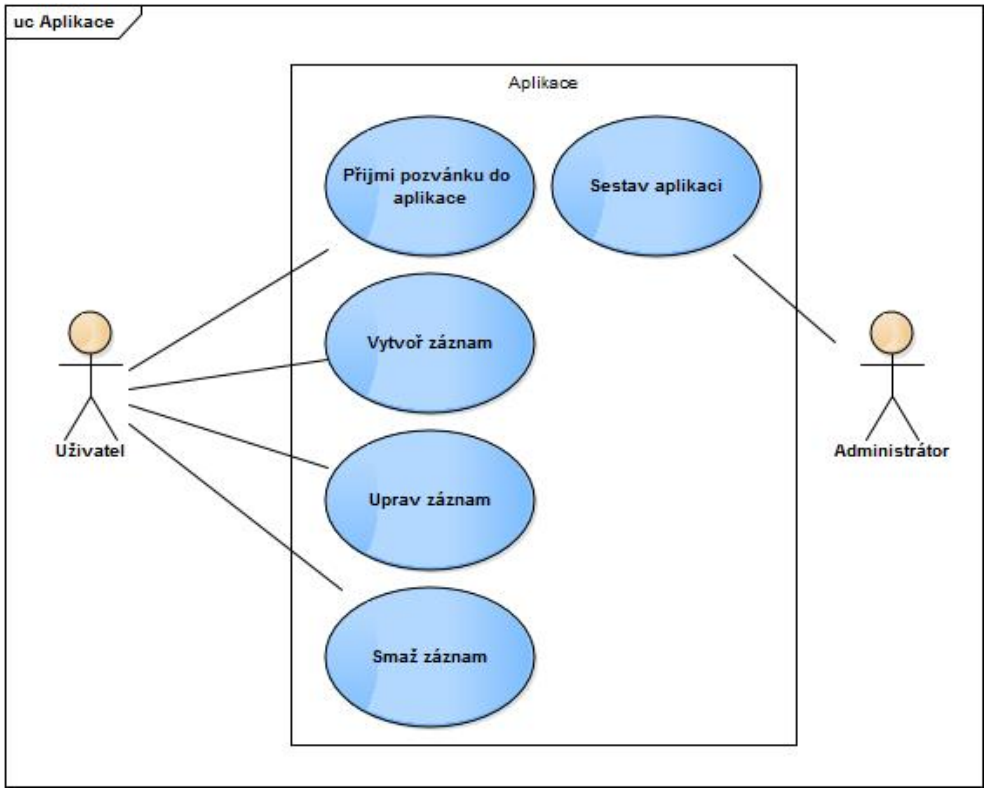
Požadované funkční požadavky aplikace byly zakresleny do use case modelů. První z use case diagramů 2.1 zobrazuje akce, které jsou uživateli umožněny, pokud se nachází v části aplikace Návrhář.

Druhý diagram 2.2 zobrazuje akce, které uživatelé aplikace mohou provádět, pokud se nachází v Aplikaci.

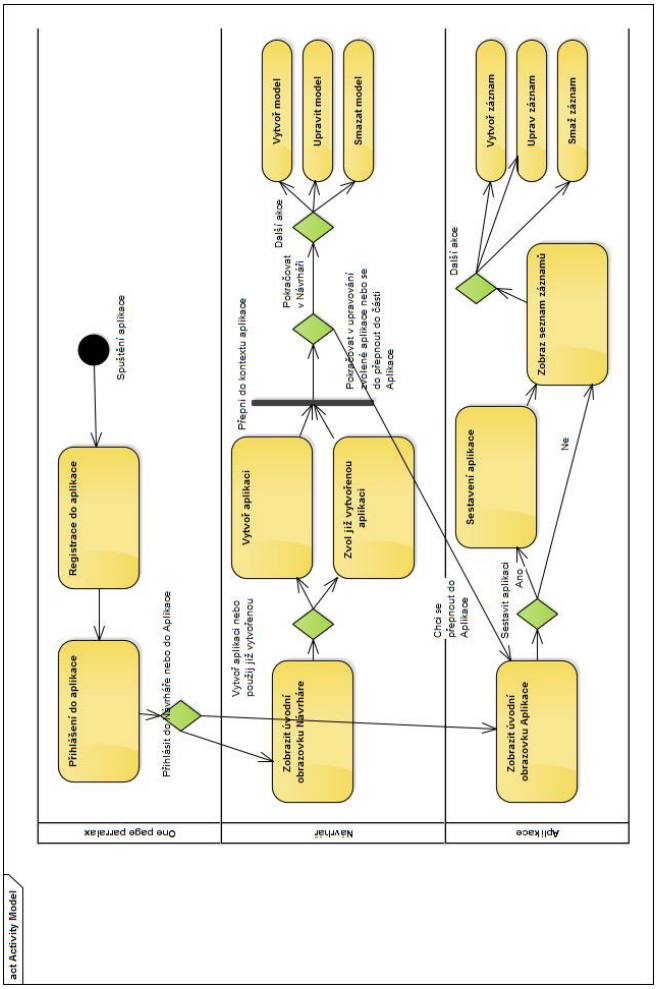
## 2.3 Diagram aktivit

Pro lepší pochopení posloupnosti provádění jednotlivých požadavků vznikl diagram aktivit 2.3. Pro jednoduchost tento diagram aktivit zobrazuje pouze základní akce a je tak zjednodušenou verzí skutečného diagramu. Neobsahuje tak například návrat zpět na úvodní obrazovku po úpravě modelu. Cílem diagramu je zobrazit průběh přihlašování s logikou rozhodnutí, zda uživatele přepnout do Aplikace nebo do Návrháře. Dalším účelem diagramu je zobrazit přepínání mezi Aplikací a Návrhářem a hlavně volbu kontextu aplikace [2].

[2]Kontext aplikace slouží k uchování informace s jakou aplikací uživatel pracuje. Veškeré další akce, jako je vytváření modelů, jsou dále prováděny v kontextu zvolené aplikace.




Obrázek 2.2: Případ použití Aplikace




Obrázek 2.3: Vytvoření a práce s aplikací

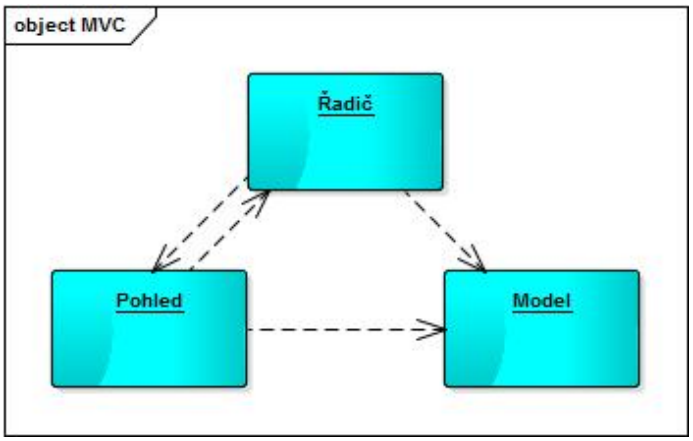
Kapitola 3

# Návrh

## 3.1 ASP.NET MVC

Jedním z požadavků na aplikaci je implementace pomocí ASP.NET MVC. Tato volba je pro návrh aplikace zcela zásadní a následující kapitola je proto věnována základnímu popisu této architektury. V další části je na této architektuře navržen základní koncept aplikace 3.5. Na závěr v části věnované implementaci 4 je popsána samotná implementace potřebných součástí.

ASP.NET MVC je technologie určená pro vytváření webových aplikací. Tato technologie je nástupcem starší technologie Web Forms a je založena na návrhovém vzoru MVC. Aplikace se tak skládá ze tří částí: model, view (pohled) a controller (řadič) 3.1




Obrázek 3.1: Model Pohled Řadič

### 3.1.1 Model

Slouží v aplikaci jako vrstva pro uchovávání dat a přístup k datům. Jde tedy o třídu, kterou používá ORM framework pro uchování entity. Tato entita pak odpovídá jednomu řádku tabulky[3].

### 3.1.2 Pohled

Součást, která na základě modelu generuje výsledné HTML a stará se tedy o zobrazení dat.

### 3.1.3 Řadič

V této části je implementována business logika. Slouží tak k získání dat z databáze a ke zpracování dat z pohledů.

### 3.1.4 Komunikace

Z diagramu architektury 3.1 lze vidět, jak spolu jednotlivé části komunikují. Nejlépe jde tato komunikace ukázat na příkladu. Pro jednoduchost si představme, že uživatel přistoupí k webové stránce, která mu zobrazí tabulku s různými daty. Komunikace v takovém případě probíhá následujícím způsobem. Řadič pomocí ORM frameworku získá data potřebná k zobrazení. Tyto data získá formou seznamu modelů. Zjednodušeně lze říci, že každý model je jeden databázový řádek. Řadič dále tento seznam modelů odešle do pohledu. Nakonec pohled, na základě získaných dat, vytvoří vzhled výsledné stránky s tabulkou.

Tento příklad může působit až příliš triviálně, ale jedna z hlavních funkcí aplikace Šoupátka je právě takové zobrazení tabulky dat. V dalších sekcích se jí tato práce bude dále zabývat.

[3]Obecně je model třída implementující business logiku, může tak jít o libovolnou třídu vytvořenou v projektu. Tato třída však nemusí být použita ORM frameworkem a nemusí odpovídat entitě. V aplikaci Šoupátka je ale možné pojmy entita a model používat libovolně, protože jedna instance modelu skutečně odpovídá jedné entitě.

## 3.2 Varianty řešení

Z popisu architektury lze vidět, že částí, která je pro naši implementaci důležitá, je model. Model slouží jako entita uchovávající data. Protože jsou Šoupátka určena ke správě entit, je tak model hlavní částí, která je pro implementaci důležitá.

Pokud uvažujeme běžnou implementaci aplikace v ASP.NET MVC, je model vytvářen v době implementace aplikace. Není důležité, zda je vytvářen na základě databáze nebo přímo implementován programátorem. Při spuštění aplikace jsou vždy všechny modely známy a je možné s nimi pracovat. V Šoupátkách by však mělo být umožněno uživateli vytvářet vlastní záznamy v databázi, tedy vlastní databázové tabulky anebo jinak řečeno modely. Proto je v aplikaci potřeba zajistit možnost vytvářet tyto modely dynamicky za běhu aplikace.

## 3.3 Dynamický model

Implementace takové funkcionality je možná několika způsoby. I když existují i další možná řešení, uvedu dvě řešení, která je možné najít. Nakonec pak uvedu navržené řešení, které stojí na vlastní myšlence.

### 3.3.1 Databázové schéma

První z navržených řešení je nahradit mapování jednotlivých modelů na jednotlivé tabulky za mapování na databázové schéma, které by umožňovalo uchovat informace o modelu. Ve skutečnosti by tak existovalo v databázi stále stejné schéma, které by uchovávalo definici o vytvořených modelech a o vztazích mezi jednotlivými modely.

### 3.3.2 ModelBuilder

Druhá možnost je implementace DynamicDbContextu[5]. Tato implementace je založena na myšlence generovat assembly s jednotlivými modely za pomoci AssemblyBuilderu. Pomocí MagicDbModelBuilderu[6] pak vytvořit dynamický databázový kontext, který bude obsahovat jednotlivé modely z vytvořené assembly.

### 3.3.3 Dynamické assembly

Posledním řešením tohoto problému je dynamicky generovat kód těchto modelů a z nich pak následně vytvořit assembly. Tedy podobně jako u druhého řešení vytvořit assembly, která by obsahovala nově vytvořený model. Stejně jako u druhého řešení je dále třeba řešit, jak o této změně modelu říct databázovému kontextu. Protože toto řešení již generuje dynamické assembly s modely, je možné stejně tak generovat i assembly s databázovým kontextem. Řešení tak spočívá v dynamickém generování assembly s celým databázovým kontextem a modely. Tím jsou při vytvoření nového databázového kontextu automaticky nastavena i metadata a není třeba využívat MagicDbModelBuilder.

### 3.3.4 Zvolené řešení

Každé z uvedených řešení má své výhody i nevýhody. První řešení je ale možné ihned zavrhnout. Toto totiž ve skutečnosti není řešení! Ve skutečnosti jde pouze o způsob jak tento požadavek obejít. Požadované řešení by mělo využívat již implementovaný ORM nejlépe Entity Framework. Toto řešení vyžaduje ve své podstatě implementaci vlastního ORM a je nutnost tak implementovat všechny požadované funkcionality do tohoto vlastního ORM. Došlo by tak například ke ztrátě change trackingu, který poskytuje Entity Framework. Druhé řešení je již mnohem přijatelnější. Jednou z jeho nevýhod je například ztráta generických dbSetu a celkově vysoká náročnost na implementaci. Mezi další nevýhody patří například také závislost na knihovně MagicDbModelBuilder. Posledním řešením se bude zabývat tato diplomová práce.

## 3.4 Dynamické řadiče

Další důležitou součástí aplikace jsou řadiče. Řadiče se starají o implementaci business logiky do aplikace. Stejně jako u modelů jsou tyto řadiče známé již v době spuštění aplikace a je tak třeba znovu vyřešit jak vytvářet řadiče dynamicky. Protože již bylo rozhodnuto, jak vytvářet dynamicky modely, nabízí se stejné řešení i pro řadiče. Ve výsledku tak bude pro každou vytvořenou aplikaci existovat assembly s modely, databázovým kontextem a řadiči.

## 3.5 Výsledný koncept aplikace

Na základě návrhu vznikl popis prvotní představy funkčnosti aplikace a návrhu architektury. Jde pouze o prvotní návrh konceptu aplikace. Slouží tak pouze k jednoduchému popisu výsledné aplikace. K popisu samotné implementace slouží v této diplomové práci kapitola implementace 4.

Návrh implementace Šoupátek je tedy založen na několika myšlenkách, které slouží ke splnění požadavků ze zadání. Jak již bylo řečeno, celá implementace Šoupátek by měla fungovat na základě generování assembly se zdrojovým kódem aplikace 4.8.3. Jinak řečeno pro každou uživatelem vytvořenou aplikaci vznikne assembly. Dále pak dojde při přístupu uživatele do samotné aplikace k použití řadiče z vytvořené assembly. Tato funkcionalita je vyřešena pomocí ControllerFactory 4.8.2. Každý z řadičů se bude starat o jeden dynamicky vytvořený model. To znamená, že bude obsahovat funkce jako: vytvořit entitu typu model, upravit entitu, odstranit entitu a další. Samotné uchovávání informací o jednotlivých vytvořených entitách a jejich úpravách bude na pozadí prováděno pomocí Entity Frameworku nad MS SQL databází. Samotné změny vlastností modelů pak budou řešeny pomocí zapnutí automatických migrací Entity Frameworku 4.8.4.1.

## 3.6 Grafický návrh a Bootstrap

### 3.6.1 Kritéria

Jedním z požadavků na výslednou aplikaci je volba vhodné Bootstrap šablony pro výslednou aplikaci. Cílem této diplomové práce tak nebylo navrhnout vzhled aplikace a jejích jednotlivých komponent, ale pouze na základě požadavků zvolit vhodnou šablonu. Při výběru šablony je několik základních kritérií, které jsou pro výslednou aplikaci kritické:

- Cena
- Podporované prohlížeče
- Použití Less případně Sass
- Komponenty
- Aktualizace
- Podpora ASP.NET MVC

Prvním, i když ne zásadním kritériem při výběru šablony, je její cena. Cena se ve většině případů pohybuje někde mezi 200 Kč až 1000 Kč za jednu licenci. V případě, kdy by šablona měla být použita pro více aplikací, může cena vzrůst až na několik desítek tisíc korun českých. Cena se odvíjí převážně od počtu komponent. Mezi základní komponenty patří popisky, poznámky, upozornění, stránkování, formulářové prvky, validace, tlačítka, ikonky a mnoho dalšího. Mezi ty, které nejsou běžnou součástí všech šablon, patří přihlašovací stránka, stránka s chybou, profilová stránka a spousta dalších věcí, které se při tvorbě vlastního webu můžou hodit.

Dalším zásadním požadavkem je počet podporovaných prohlížečů. Pokud aplikace počítá například pouze s podporou Internet Exploreru verze 10 a vyšší není tento požadavek kritický. V případě, kdy by měla aplikace podporovat i starší prohlížeče, tak se seznam možných šablon velmi rychle zkrátí.

Pokud nejste se vzhledem aplikace absolutně spokojeni, je dalším důležitým faktorem volby šablony použití Less případně Sass. Tyto CSS preprocesory umožňují provádět ve výsledném vzhledu aplikace změny pomocí úpravy jednoho parametru. Můžete tak například velmi snadno změnit barvu aplikace na jinou.

Předposledním kritériem je aktualizace. Aktualizací je zde myšleno, kdy byla naposledy na šabloně provedena změna. Pomocí tohoto údaje je možné poznat nebo alespoň odhadnout, zda je možné počítat například s přidáním dalších komponent, či jak dlouho trvá opravení nalezených chyb v šabloně.

Posledním kritériem je podpora ASP.NET MVC. Některé šablony podporují technologii ASP.NET MVC. Podporou je v tomto případě myšlena implementace HTML helperu pro jednotlivé grafické komponenty aplikace. Je

tak možné například jednoduše bez znalosti HTML vygenerovat bootstrap panel nebo jinou z komponent. Programátor si tak nemusí pamatovat veškeré potřebné třídy a další atributy, které je potřeba HTML nastavit.

### 3.6.2 Poskytovatelé

Jednotlivé šablony jsou v drtivé většině případů poskytovány přes web třetích stran. Díky tomu je možné vybírat ze stovek šablon. Mezi nejčastější a nejoblíbenější poskytovatele patří:

- https://wrapbootstrap.com/
- http://startbootstrap.com/
- https://bootswatch.com/
- http://bootstrapzero.com/

### 3.6.3 Zvolená šablona

Výslednou zvolenou šablonou je šablona se jménem Color Admin. Tuto šablonu je možné získat ze stránek https://wrapbootstrap.com/ za cenu $18 za jednu licenci případně za $70 pro vícenásobnou licenci. Šablona podporuje veškeré nejnovější prohlížeče a Internet Explorer od verze 8. Celá šablona je postavena s využitím css pre-processoru LESS a umožňuje tak snadné upravení jednotlivých grafických komponent aplikace. Color Admin za $18 nabízí obrovské množství komponent. Mezi komponenty, které naleznete v aplikaci Šoupátka, patří:

- Datové tabulky
- Stránkování
- Validace
- Datepicker a další formulářové prvky
- Přihlašovací stránka
- Chybová stránka 404
- One Page Parallax
- Icony

Celý seznam komponent a celou šablonu naleznete na stránkách `http://wrapbootstrap.com/preview/WB0N89JMK`. Aplikace Šoupátka je postavena na verzi COLOR ADMIN 1.5.

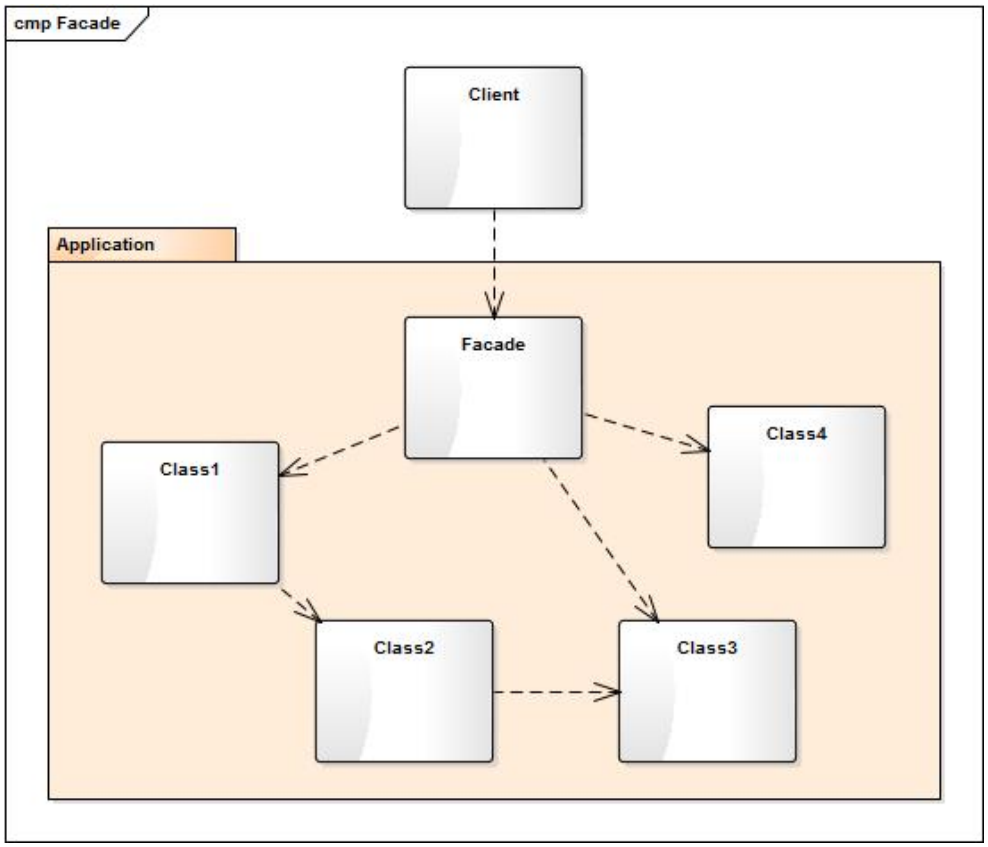

Obrázek 3.2: ColorAdmin 1.5

#### 3.6.3.1 Java Script knihovny

Zvolená šablona je postavena na několika Java Script knihovnách. Mezi ty nejzajímavější patří:

- DataTable
- Wizard
- Font awesome
- Gritter
- Morris
- Slimscroll

Některé z těchto knihoven využívá aplikace Šoupátka i pro vlastní účely. O jaké knihovny se jedná, a jaké je použití těchto knihoven v aplikaci, se dozvíte více v sekci 4.1




Obrázek 3.3: Facade [7]

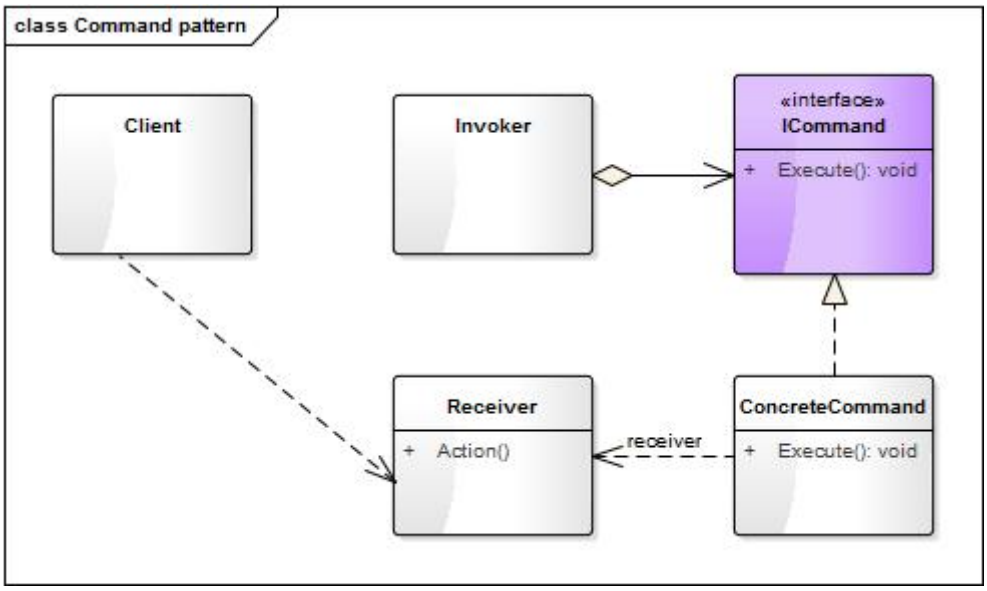
## 3.7 Architektura aplikace

### 3.7.1 Jádro aplikace

Jádro aplikace slouží jako jednotné místo pro přístup a manipulaci s daty. Výsledná architektura jádra aplikace je postavena na několika návrhových vzorech.

#### Façade

Základem celé architektury je Façade pattern. Tento návrhový vzor slouží ke schování složitého systému za jednoduchý interface. Přes tento interface může aplikace se systémem jednoduše pracovat. Zároveň tak může sloužit jako společný přístupový bod k různým částem systému a poskytovat z jednotlivých součástí systému pouze potřebné metody. Jde tedy o strukturální návrhový vzor. Funkci návrhového vzoru Façade znázorňuje diagram 3.3. V doménovém modelu 3.7.2 aplikace se s návrhovým vzorem Façade setkáte u objektu Kernel.




Obrázek 3.4: Command Pattern [7]

### Command pattern

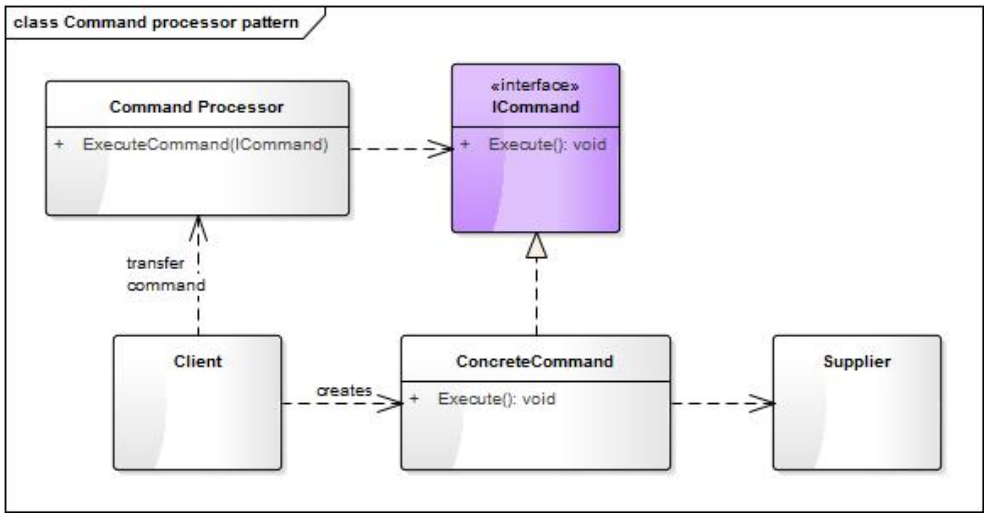
Command pattern je jeden z návrhových vzorů patřících do skupiny behaviorálních. Tento návrhový vzor řeší problém vyvolání požadavku na objektu bez detailní znalosti operace, která bude na výsledném objektu provedena. Command pattern tedy zapouzdří požadavek do jednoho objektu. Tímto objektem je ve většině implementací interface ICommand. Funkci Command patternu znázorňuje diagram 3.4.

### Command processor pattern

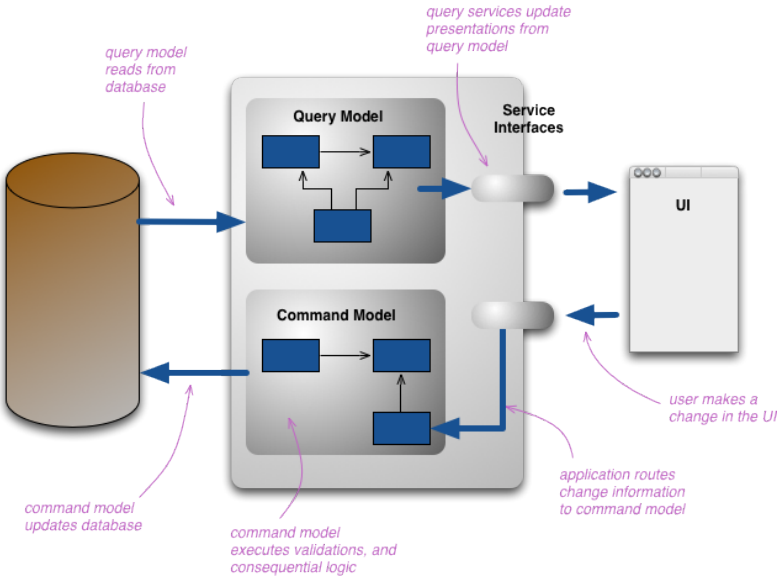
Návrhový vzor Command processor je postaven na již zmíněném Command patternu. Využitím Command patternu dojde k zapouzdření požadavku do objektu. Dále je třeba se rozhodnout, jak s daným objektem pracovat, a právě k tomu slouží Command processor pattern. Rozšiřuje tedy Command pattern o specifikaci, jak s výsledným objektem neboli Commandem zacházet. Funkci Command processoru znázorňuje obrázek 3.5.

#### 3.7.1.1 Command Query Responsibility Segregation

CQRS je návrhový vzor, který řeší jak pracovat s informačními systémy. Pro práci s informačními systémy je zvykem používat CRUD operace, což znamená, že můžeme data číst, zapisovat, aktualizovat a smazat. Na rozdíl od CRUD pohlíží CQRS na práci s daty jinak. Odděluje úpravy informací a čtení informací. Úpravy jsou prováděny pomocí příkazů a čtení informací pomocí

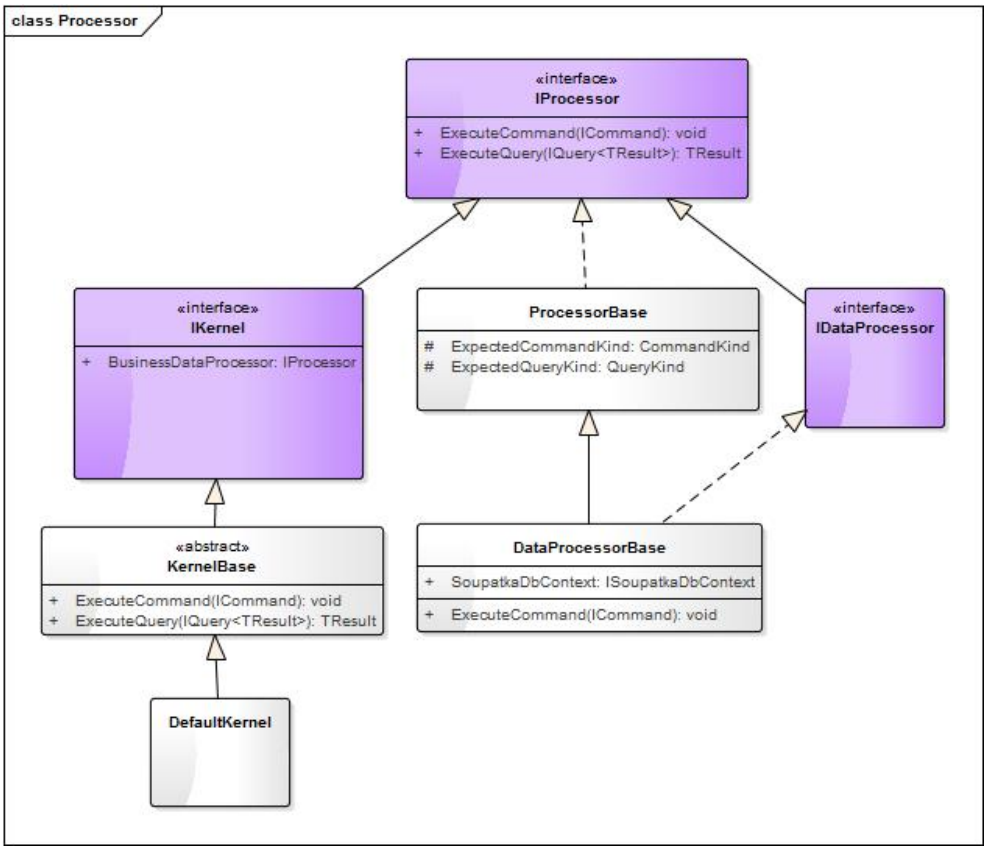



Obrázek 3.5: Command Processor Pattern [7]




Obrázek 3.6: Command Query Responsibility Segregation [8]

dotazů. Implementace jednotlivých příkazů a dotazů se pak může lišit. Pomocí tohoto rozdělení je kupříkladu možné vytvořit dotaz, který čte seznam záznamů a další dotaz, který čte pouze jeden záznam z informačního systému. Operace čtení je tak nahrazena za dva dotazy. Tento návrhový vzor je proto vhodný především v případech, kdy jsou pouze běžné operace CRUD nedostatečné a je třeba dalších operací 3.6.




Obrázek 3.7: Jádro aplikace

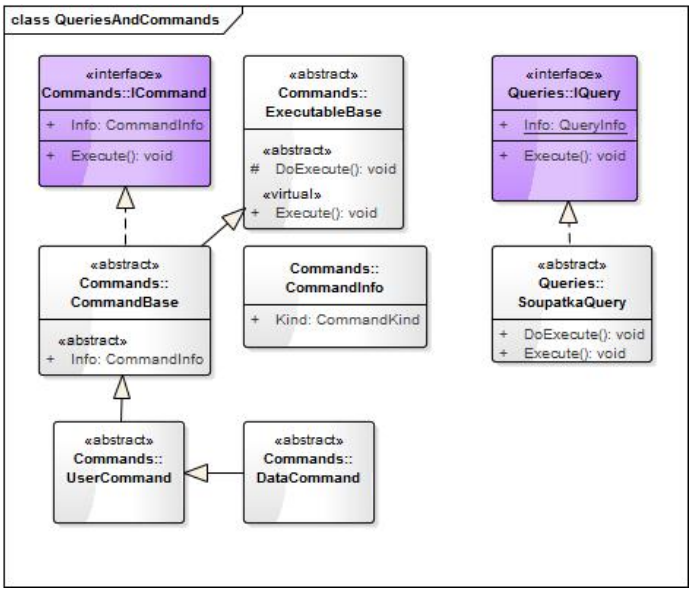
### 3.7.2 Doménový model

Spojením těchto návrhových vzorů vznikl doménový model. Z modelu 3.7 můžeme vidět využití interface IKernel jako Façady, která umožnuje provádět metody ExecuteCommand respektive ExecuteQuery, tedy je zde použit návrhový vzor CQRS. Tyto metody dále přijímají interface ICommand respektive IQuery, které slouží jako výsledný zapouzdřený objekt z Command patternu. O spuštění jednotlivých příkazů a dotazů se stará IProcessor.

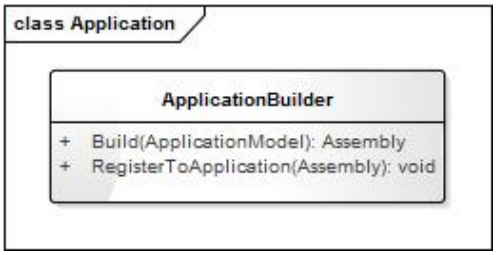
### 3.7.3 Application builder

Hlavní částí aplikace je Application builder 3.9. Application builder se stará o sestavení výsledné aplikace z definice, která byla vytvořena pomocí grafického rozhraní návrháře. Měla by tedy umožnit vygenerovat výsledné assembly, které budou obsahovat zdrojový kód jednotlivých aplikací vytvořených uživateli.

Pozor slovo builder v názvu je v tomto případě mírně matoucí, protože ve




Obrázek 3.8: Příkazy a dotazy




Obrázek 3.9: Doménový model ApplicationBuilder

skutečnosti nemá model s návrhovým vzorem Builder nic společného.

Kapitola 4

# Implementace

## 4.1 Přední část aplikace

Implementace přední části aplikace nebyla hlavním cílem této práce. Z tohoto důvodu je přední část postavena na zakoupené bootstrap šabloně. Šoupátka využívají mnoho externích knihoven, které jsou součástí této šablony 3.6.3, stejně tak jako několik dalších.

### 4.1.1 JQuery

Základním kamenem dnes již drtivé většiny webových aplikací je JQuery a ani aplikace Šoupátka nejsou žádnou výjimkou. JQuery je malá, rychlá JavaScriptová knihovna obsahující velké množství funkcí. Umožňuje snadnou manipulaci s HTML elementy, jejich animaci, reakci na události a mnoho dalšího.

### 4.1.2 ImgAreaSelect

ImgAreaSelect [12] je JQuery zásuvný modul sloužící k ořezávání obrázků. Implementace ale ve skutečnosti obrázek skutečně neořízne, pouze vytvoří grafické rozhraní pro oříznutí obrázku. Samotné oříznutí je nutné provést až na serveru ze získaných souřadnic. Tento zásuvný modul umožňuje nastavit několik parametrů, jako jsou například poměry jednotlivých stran nebo maximální velikost obrázku. Je tak možné například nastavit, aby šel obrázek oříznout pouze na čtverec. Tento modul je v aplikaci použit u nahrávání profilového obrázku.

### 4.1.3 Font-awesome

Dalším zajímavým modulem je Font-awesome [13]. Font awesome poskytuje vektorové ikony, které mohou být dále upravované, jako například jejich barva, velikost, stíny a to všechno jen za pomoci CSS. Navíc jde skutečně o vektorovou grafiku, takže ikony vypadají dobře v jakékoli velikosti.

### 4.1.4 DataTable

Jednou z hlavních částí grafického rozhraní aplikace je tabulka sloužící k zobrazení seznamu entit. Protože tabulka může potencionálně obsahovat velké množství záznamů, je třeba ji rozšířit o možnost filtrování, stránkování, vyhledávání a dalších možností. Pro tyto účely je v aplikaci použit další zásuvný modul pro JQuery DataTable [11]. Tento modul umožňuje jednoduše nastavit, jaké z funkcí chcete, aby vámi požadovaná tabulka měla, a modul již vše automaticky vyřeší.

## 4.2 Barevné menu

Jednou z funkcionalit, o kterou byla zvolená šablona rozšířena, je implementace dvoubarevného menu. V případě, kdy se uživatel nachází v Návrháři, je menu tmavé, zatímco pokud se nachází v Aplikaci, menu je světlé. Uživatel tak snadno rozpozná, v které části se zrovna nachází. Do menu bylo navíc přidáno tlačítko, které umožňuje extra rychlé přepínání mezi Aplikací a Návrhářem. Uživatel tak může během jednoho kliknutí vidět, jak se změny provedené v Návrháři projevily v Aplikaci samotné.

## 4.3 Notifikace

Jednou z důležitých částí aplikace jsou notifikace. Notifikace v současnosti slouží k zobrazení pozvánek do Aplikace. Zobrazují informaci o tom, do jakých Aplikací je uživatel pozván, stejně tak jako který uživatel přijmul pozvánku do námi vytvořené Aplikace. Uživatelé si také mohou zobrazit historii pozvánek a přijmout například pozvánku, kterou dříve odmítli. Sekce notifikací dává velký prostor k vylepšení aplikace, a je to jedna z částí, u které se počítá s dalším budoucím vývojem 6.2.

## 4.4 Lokalizace

Jedním z požadavků na aplikaci je také lokalizace do anglického a českého jazyka. V ASP.NET MVC je lokalizace řešena pomocí zdrojových resx souborů. Tento soubor obsahuje záznamy uchovávané formou XML [10].

```
<data name="Key">
<value>Hello world!</value>
</data>
```

K jednotlivým hodnotám v resx souboru je pak možné přistupovat jako k statickým proměnným. Proto, aby byl tento soubor dostupný ze všech částí aplikace, je ale nutné vytvořit pro tyto lokalizační soubory vlastní projekt a dále nastavit reference na tento projekt. Oddělení souborů do vlastního projektu však může znamenat komplikace se získáváním těchto informací pro NuGet balíčky, které pro přístup k lokalizaci využívají resource provider. Jedním z těchto balíčků je například MVC SiteMapProvider 4.9.1.

### 4.4.1 Resource provider

Aby bylo možné lokalizaci použít i v těchto případech, je nutné implementovat vlastní ResourceProviderFactory a ResourceProvider. Implementaci těchto částí je možné najít [16]. Tato implementace je pro náš případ absolutně dostačující. Šoupátka tak neimplementují vlastní logiku a jsou postaveny na této implementaci. Po implementaci ResourceProviderFactory a ResourceProvider je ještě třeba nastavit v aplikaci používání této implementace. To je provedeno jednoduše přidáním řádku globalization do web.configu aplikace a nastavením parametru ResourceProviderFactoryType.

### 4.4.2 Lokalizace enum

Další částí, u které je třeba řešit lokalizaci, jsou enumerace. Enumerace slouží k zobrazení dropdown listu v GUI aplikace a musí být lokalizované. Aplikace tuto situaci řeší implementací vlastního description atributu. Tento atribut přepisuje základní Description atribut a rozšiřuje ho o možnost nastavit typ souboru s lokalizací. Dále je třeba vytvořit vlastní display template pro zobrazení enumu v pohledech. Tento template na základě nově vytvořeného atributu a nastaveného typu souboru a klíče získá lokalizovanou hodnotu.

## 4.5 Dotazy a příkazy aplikace

Návrh samotného jádra aplikace již byl popsán v sekci 3.7.1. Na základě tohoto návrhu byla provedena implementace. Implementaci bylo třeba oproti návrhu rozšířit o samotné dotazy a příkazy, tedy o implementaci tříd, které implementují abstraktní třídy DataCommand a BusinessDataQuery.

### 4.5.1 Dotazy

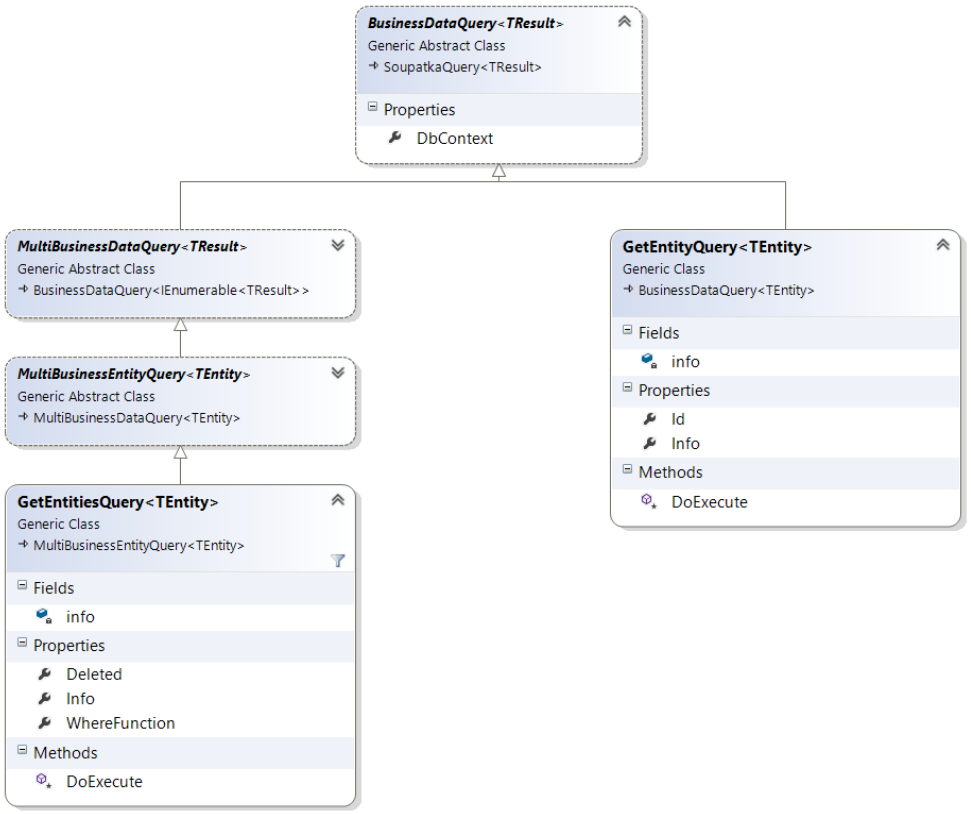
Oproti běžnému CRUD modelu, kde existuje pouze operace čtení pro získávání dat, jsou v Šoupátkách implementované funkce dvě. První z nich slouží ke čtení pouze jedné entity (GetEntityQuery) a druhá ke čtení seznamu entit (GetEntitiesQuery). Oba tyto dotazy jsou implementované jako generické třídy, kde generický atribut rozhoduje o typu databázového setu, který bude pro dotaz použit.

Implementace GetEntityQuery je velmi jednoduchá - stačí, aby dotazu bylo zadáno Id entity, kterou chce získat a ta je vyhledána v příslušném databázovém setu.

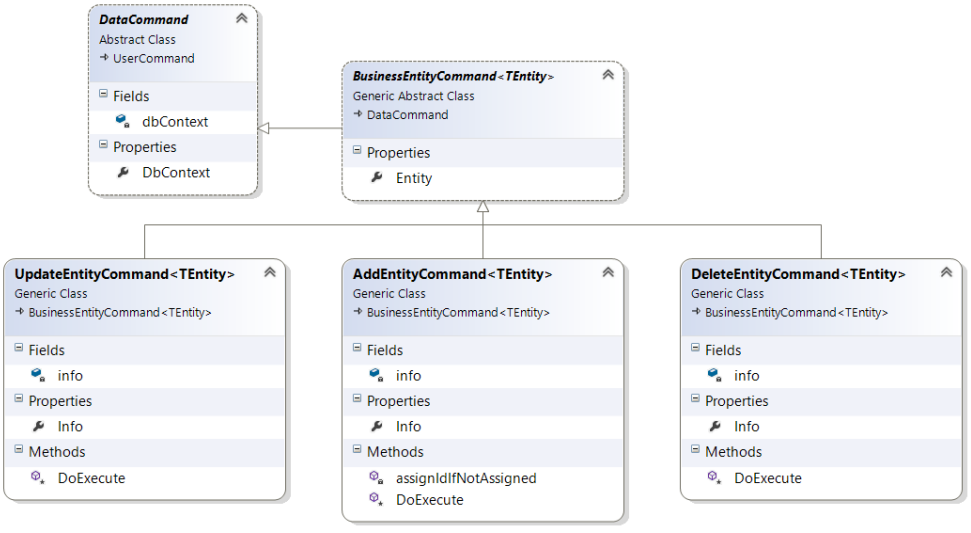
Implementace GetEntities už je komplikovanější. Tato implementace zaprvé řeší, aby vrácený seznam obsahoval pouze entity, které nejsou smazané. Provádí tedy kontrolu, zda má entita nastavená booleovskou hodnotu Deleted na false a zároveň umožňuje provádět filtrování dat pomocí delegátu Func<TEntity, bool>. Výsledný seznam, pak vrací v podobě interface IEnumerable<>. Class diagram dotazů je na obrázku 4.1.

### 4.5.2 Příkazy

Oproti dotazům implementace příkazů obsahuje pouze tři základní běžné operace známé z CRUD a to vytvořit, upravit a smazat. AddEntityCommand slouží pro vytváření nových entit. DeleteEntityCommand je určen pro jejich smazání, tedy pouze pro změnu booleovské hodnoty. A UpdateEntityCommand pro jejich aktualizaci. Na obrázku 4.2 je vidět class diagram příkazů.




Obrázek 4.1: Class diagram dotazů




Obrázek 4.2: Class diagram příkazů

## 4.6 Entity Framework

Pro přístup k datům aplikace implementované v ASP.NET MVC se nabízí mnoho možností. Patří mezi ně například LINQ, nHibernate a ADO.NET Entity Framework. Aplikace Šoupátka pro přístup k datům využívá poslední jmenovaný Entity Framework a to ve verzi 6.2.

Tento přístup patří v současnosti i mezi doporučovaný přístup společností Microsoft [14]. Entity Framework patří mezi ORM frameworky. Objektově relační frameworky zajišťují propojení mezi objektově orientovanými jazyky a relační databází. Entity Framework ve verzi 6.2 podporuje tři možnosti jak tyto objekty na databázi napojit. Mezi podporované způsoby patří takzvané přístupy první kód, první databáze anebo první model. Běžněji se ale čtenář setká s anglickými názvy code-first, database-first a model-first. V nově plánované verzi Entity Frameworku 7 bylo ale rozhodnuto, že dále bude podporovaný pouze způsob kód první. Pro účely aplikace Šoupátka nelze další způsoby z logických důvodů použít, a proto je postavena právě na tomto způsobu.

### 4.6.1 Kód první

Nenechte se zmást názvem kód první. Tento název může být lehce matoucí a ve skutečnosti je možno tento způsob využít i ve chvíli, kdy už máme vytvořenou databázi [15]. Ve skutečnosti je názvem pouze myšleno, že pro přístup k databázi je použito tříd a tedy kódu.

```
1  public class TestEntity
2  {
3      [Key]
4      public Guid Id { get; set; }
5
6      [Display(Name = "Int", Order = 0)]
7      public int Int { get; set; }
8
9      [Display(Name = "Bool", Order = 1)]
10     public bool Bool { get; set; }
11
12     [Display(Name = "Decimal", Order = 2)]
13     public decimal? Decimal { get; set; }
14
15     [Display(Name = "DateTime", Order = 3)]
16     public DateTime? DateTime { get; set; }
17 }
```

Jak lze vidět tento přístup pro definici databázového modelu využívá jednoduchý přístup, kdy je výsledná podoba databázových tabulek definovaná pomocí

jednotlivých vlastností třídy a pomocí atributů nad těmito vlastnostmi. Tato definice je v případě architektury MVC nazývána modelem. Definice pomocí modelu a atributů je dostačující ve většině případů a je možné pomocí ní vytvořit komplikované databázové schéma.

Entity Framework ve verzi 6 umožňuje tímto způsobem definovat nad databází

- Název schématu
- Název databázové tabulky
- Název sloupce
- Klíče tabulky
- Omezující podmínky (constraint)
- Typy sloupců
- Databázové indexy (od verze Entity Framework 6.1)

a mnoho dalších nastavení. V některých případech není tato definice dostačující, protože využívá konvence definované Entity Frameworkem, které nemusí být ve všech případech uspokojující. Je tak nutné rozšířit definici pomocí Fluent API. Šoupátka v současnosti nepodporují definice pomocí Fluent API, ale pouze pomocí atributů.

### 4.6.2 Databázové migrace

Databázové migrace patří mezi jednu z potřebných funkcionalit pro navrženou architekturu aplikace a slouží k sjednocení definovaného modelu a databáze. Toto sjednocení je prováděno porovnáním modelu aplikace s metadaty v databázi.

Samotná migrace se skládá z několika kroků. Nejdříve je nutné provést zmíněné porovnání databáze a modelu. To je možné provést příkazem add-migration. Tento příkaz vytvoří kód migrace, který popisuje změny, které je nutné provést nad databází. Jako druhý krok je třeba tyto změny aplikovat na databázi. To je provedeno pomocí dalšího příkazu update-database. Po provedení těchto kroků je tak databázové schéma stejné jako model aplikace.

## 4.7 Autentifikace

Jednou ze základních součástí webových aplikací je možnost autentifikace, tj. možnost zaregistrovat se a následně se přihlásit do aplikace. Šoupátka jsou postavena na Visual Studio 2013 projektové šabloně pro ASP.NET MVC 5. Tato šablona již obsahuje připravenou možnost autentifikace.

### 4.7.1 ASP.NET Identity

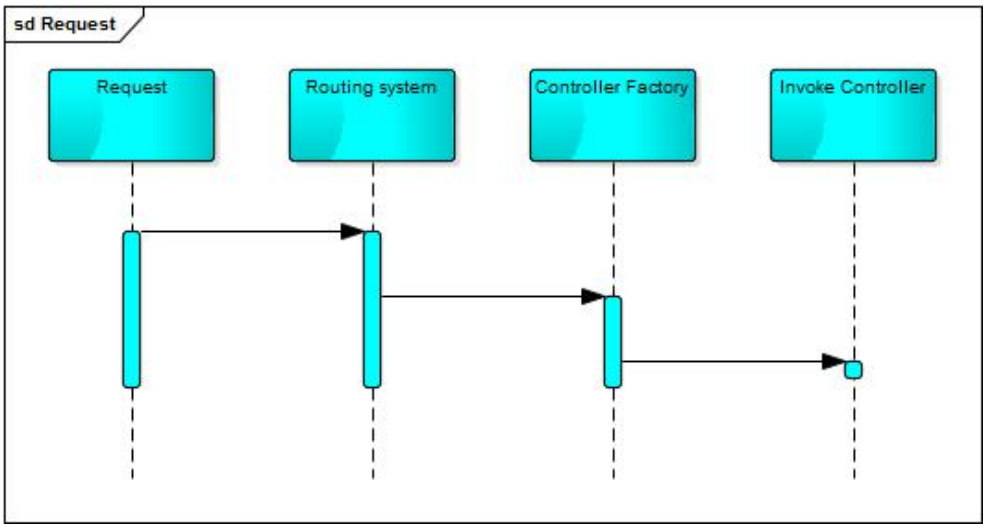
Tato šablona je postavena na ASP.NET Identity. ASP.NET Identity je nástupce ASP.NET Membership a ASP.NET Simple Membership. V průběhu vývoje webových aplikací, díky vzniku sociálních sítí jako je Facebook či Twitter, vznikl požadavek na možnost přihlášení do webové aplikace pomocí identity vytvořené v jedné z těchto sociálních sítí. ASP.NET Identity vznikla právě s cílem umožnit uživateli přihlášení pomocí jednoho z těchto účtů. To ale nebyl jediný cíl, proč Identity vznikla. Mezi další cíle patří například sjednocení přihlašování v jednotlivých ASP.NET aplikacích, lepší možnost provádění unit testů, poskytování rolí a také splnění dalších požadavků na moderní webové aplikace [17].

ASP.NET Identity je postavena na OWIN, díky čemuž je možné aplikaci používat na jakémkoliv serveru podporujícím OWIN. OWIN je zkratka pro Open Web Interface for .NET. OWIN definuje interface mezi .NET webovou aplikaci a webový server. Cílem OWIN interface je oddělit server od aplikace a umožnit tak jednoduchý vývoj modulů pro webové aplikace v .NET [18].

#### 4.7.1.1 Uživatelský profil

Aplikace Šoupátka využívá možnost jednoduše pomocí ASP.NET Identity vytvořit pro registrované uživatele profil. Do tohoto profilu si aplikace ukládá informace o uživateli, jako je jeho jméno, příjmení, profilový obrázek, seznam notifikací, seznam Aplikací, do kterých je uživatel registrován a v jaké části aplikace a v jaké Aplikaci se naposledy nacházel. Neschází ani informace o zvoleném jazyku aplikace.

Uživatel je tak díky všem těmto informacím při přihlášení automaticky přesměrován do části aplikace, ve které se naposledy nacházel. Stejně tak jako je mu nastaven uložený jazyk aplikace.




Obrázek 4.3: Průběh požadavku na ASP.NET MVC aplikaci

## 4.8 Sestavení aplikace

### 4.8.1 Routing a jmenné konvence

Při sestavování aplikace je třeba myslet na výslednou URL, přes kterou budou uživatelé do svých aplikací přistupovat.

V ASP.NET MVC je rozhodování o tom, která URL patří kterému řadiči, řešeno pomocí routingu a pomocí ControllerFactory 4.8.2.

Routing je jednoduchá definice, která říká, jak by měla URL vypadat, pokud chceme přistupovat k specifickému řadiči. Celý proces od přijmutí požadavku do ASP.NET MVC webové aplikace tak lze znázornit jako na obrázku 4.3.

Základní routovací pravidlo v ASP.NET MVC:

```
routes.MapRoute(
name: "Default",
url: "{controller}/{action}/{id}",
defaults: new {
controller = "Home",
action = "Index",
id = UrlParameter.Optional
},
namespaces: new[] {
"Soupatka.Web.Desktop.Controllers"
}
);
```

Takové pravidlo ale není v případě Šoupátek možné. Za prvé úplně schází definice, pro jakou aplikaci je řadič určen, a za druhé je třeba myslet na konflikty mezi názvy jednotlivých modelů. Jednoduchým příkladem je situace, kdy se dva uživatelé rozhodnou pojmenovat svojí aplikaci stejným názvem, například Hello world, a v té pak vytvoří stejně pojmenovaný řadič. V takovém případě by v aplikaci vzniknul konflikt mezi jednotlivými URL.

Prvním řešením takové situace by bylo vyžadovat unikátní název pro aplikaci. Při zakládání aplikace by tak proběhla kontrola, zda již stejně pojmenovanou aplikaci nevlastní jiný uživatel. Toto řešení ale není vhodné například v případě, kdy aplikaci chce vyzkoušet nový uživatel. Při zadání názvu aplikace Hello world zjistí, že je název Hello world již zabraný a musí tak použít první volný Hello world 1254. Proto je tato situace v Šoupátkách řešena pomocí jednoduché jmenné konvence a vytvořením vlastního routovacího pravidla.

```
1  public void RegisterArea(AreaRegistrationContext arc)
2  {
3  arc.MapRoute(
4  name: "Application_default",
5  url: @"Application/{urlhash}/
6  {controller}/{action}/{id}",
7  defaults: new {
8  controller = "Home",
9  action = "Index",
10 id = UrlParameter.Optional
11 }
12 );
13 }
```

Toto pravidlo rozšiřuje základní pravidlo o definici area a urlhash. Areas jsou jeden z možných způsobů jak v ASP.NET MVC řešit rozdělení projektu obsahujícího velké množství řadičů. V aplikaci Šoupátka jsou areas využity k oddělení Návrháře a Aplikace. Toto routovací pravidlo je tedy použito pouze tehdy, pokud se uživatel nachází v area Aplikace. Urlhash je vygenerovaný unikátní název, tvořený z e-mailu uživatele, který aplikaci vytvořil, a ze zadaného názvu. Díky této definici si tedy dva různí uživatelé mohou vytvořit aplikaci se stejným názvem a stejným modelem.

### 4.8.2 Controller factory

Po nastavení routingu je, jak je zobrazeno na obrázku 4.3, třeba implementovat vlastní ControllerFactory. ControllerFactory je součást MVC, která pomocí hodnot získaných z routingu a názvu vytvoří instanci požadovaného řadiče.

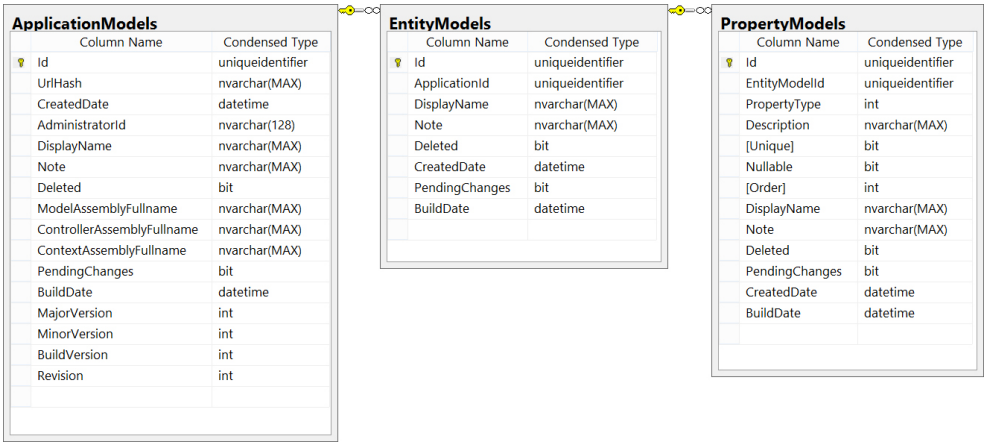
Defaultní implementace se na základě přijatého názvu řadiče a hodnot z routingu pokusí najít typ řadiče a poté za pomoci aktivátoru inicializovat novou instanci. Protože Šoupátka kompilují vlastní assembly, nedokáže defaultní implementace tyto řadiče najít. Proto je třeba v implementaci Šoupátek toto rozhodování upravit. Tuto úpravu stačí provést pouze v případě, kdy se uživatel nachází v Aplikaci. V ostatních případech stačí použít defaultní implementaci ControllerFactory.

### 4.8.3 Application builder

Při návrhu aplikace vznikl jednoduchý doménový model 3.9 zobrazující základní funkcionalitu application builder. V modelu jsou dvě hlavní funkce. První funkcí je Build. Tato funkce by měla vygenerovat výsledné assembly, které budou obsahovat zdrojový kód jednotlivých aplikací vytvořených uživateli. Druhou funkcí je RegisterToApplication, která se stará o zaregistrování těchto assembly do aplikace. Zaregistrováním je myšleno přidání DLL do aplikační složky bin. Z pohledu MVC architektury je tedy třeba vytvořit assembly s modely, pohledy a řadiči. O generování těchto assembly se v aplikaci stará AssemblyBuilder. AssemblyBuilder využívá třídy CSharpCodeProvider. Tuto třídu je možné v jazyce C# použít ke zkompilování zdrojového kódu do assembly. Tato třída se nachází v assembly System.dll. CSharpCodeProvideru je možné nastavovat několik parametrů, které jsou pro výslednou aplikaci zásadní.

- GenerateExecutable: Boolean hodnota určující, zda se má vygenerovat spustitelný program nebo DLL
- GenerateInMemory: Boolean hodnota která určí, zda se má vygenerovat výsledný soubor pouze v paměti nebo na disk
- OutputAssembly: Nastaví název a případně i cestu k výslednému souboru
- ReferencedAssemblies: Vazby na další potřebné assembly

Pro aplikaci je tedy třeba nastavit GenerateExecutable na false. Tím vznikne výsledná DLL, kterou je následně možné v aplikaci registrovat. Stejně tak je třeba nastavit GenerateInMemory na false. Tím se vygeneruje assembly do složky TEMP. Z této složky je pak možné DLL vzít a přesunout do složky bin, která zajistí registraci assembly v aplikaci, a je jí tak možné ve webové aplikaci používat. Dále je možné nastavit pro výslednou DLL název a v neposlední řadě reference na další assembly. Po nastavení veškerých potřebných parametrů je už pouze třeba CSharpCodeProvideru předat v datovém typu string zdrojový kód, který bude do assembly zkompilován.




Obrázek 4.4: Databázové schéma pro uchování definice aplikace

### 4.8.4 Generování zdrojového kódu do stringu

Pro generování zdrojového kódu je možné využít několik přístupů. První a nejjednodušší možností je vytvořit výsledný zdrojový kód přímo na místě použití za pomoci StringBuilderu. Toto řešení je sice, co se týče doby implementace, nejrychlejší, ale jeho velkou nevýhodou je nepřehlednost.

Další variantou je využití přístupu, který používá například Visual Studio pro Scaffolding. Tento způsob spočívá ve vytvoření t4 šablon kódu a vygenerování výsledného kódu z těchto šablon. Tyto šablony, které Visual Studio interně používá pro scaffolding, je možné stáhnout ze stránek SideWaffle `http://sidewaffle.com/`

Další uvažovanou možností pro generování zdrojového kódu je nová možnost v ASP.NET MVC od verze 3 využít Razor view engine. Tento engine využívá odlišnou syntaxi od předchozího ASPX view engine. Nahrazuje například všudypřítomné znaky <% a %> (tolik podobné znakům <# a #> používaných v t4 šablonách) za @ a přispívá tak k zjednodušení a zpřehlednění výsledného kódu [19] Jeho nevýhodou je naopak potřeba využití speciálního <text> elementu při přechodu ze zdrojového kódu do zápisu HTML. Stejně tak jako obecná nevýhoda využití view engine pro generování zdrojového kódu. View engine, jak už název vypovídá, je totiž primárně určen pro generování views neboli pohledů a tedy pro generování HTML. To se projeví jako problém například u C# kódu využívajícího generických atributů, které jsou mylně považovány za HTML elementy.

Aplikace Šoupátka přesto využívá právě tento nejnovější ASP.NET Razor view engine ke generování výsledného kódu a to z důvodu jednoduché syntaxe, lepší podpory ze strany textových prohlížečů a snadného generování na základě vytvořeného databázového modelu.

#### 4.8.4.1 Modely

Generování modelu probíhá na základě databázové tabulky EntityModel. Aby řešení bylo co nejjednodušší, jsou všechny modely vytvořené uživatelem spojeny do jednoho společného stringu. Nevýhodou tohoto řešení je to, že je nutné při každé změně nad modelem znovu generovat celou aplikaci.

Všechny uživatelem vytvořené modely dědí od jedné společné třídy. Touto třídou je SoupatkaEntity. Ukázkový kód je záměrně zkrácen.

```
1  public class SoupatkaEntity : Entity<Guid>, IComparable
2  {
3      [Required]
4      [Display(Name = "DisplayName", Order = 0)]
5      [DefaultColumn(IsDefaultColumn = true)]
6      public virtual string DisplayName { get; set; }
7
8      [DataType(DataType.MultilineText)]
9      [Display(Name = "Note", Order = 0)]
10     public virtual string Note { get; set; }
11
12     [Display(Name = "Deleted", Description = "Deleted")]
13     public virtual bool Deleted { get; set; }
14 }
```

Tato třída dále dědí od generické třídy Entity.

```
1  [Serializable]
2  public abstract class Entity<TId>
3  {
4      [Key]
5      public virtual TId Id { get; set; }
6  }
```

Všechny modely Aplikace tak obsahují Id, DisplayName, Deleted a Note. Tyto vlastnosti vycházejí z návrhu aplikace.

Id slouží jako unikátní rozlišení jednotlivých instancí modelů. Na základě Id je generován v databázi primární klíč. Tato vlastnost modelu tak také slouží jako routovací hodnota pro akce Edit a Detail.

Další vlastností je textová hodnota DisplayName. Každý model v aplikaci musí mít kromě unikátního Id také název. Vytvořené modely je tak možné snadno odlišit v seznamovém pohledu.

Deleted slouží k možnosti smazat jednotlivé záznamy. Záznamem může být v tomto případě jak ApplicationModel, EntityModel, PropertyModel tak i jednotlivé záznamy vytvořené uživateli. Pokud uživatel v aplikaci smaže některý takovýto záznam, není ve skutečnosti smazán, ale je pouze přenastavena boolovská hodnota na true. V grafickém rozhraní aplikace se tak tento záznam

přestane zobrazovat, zatímco v databázi se stále nachází. Toto řešení umožňuje administrátorovi aplikace tak na vyžádání některý ze záznamů obnovit. V současnosti pro tuto akci neexistuje GUI a obnovu je potřeba provést přímo v databázi.

Note slouží k uchování interních informací aplikace.

### Podporované typy

Při generování modelu je třeba se rozhodnout, jaké vlastnosti bude moct uživatel pro model definovat, a tedy jaká data bude koncový uživatel moci zadávat. Aplikace v současnosti podporuje pouze základní datové typy:

- Číslo (int)
- Text (string)
- Datum a čas (datetime)
- Peníze (decimal)
- Ano a ne (bool)

O možnostech a plánech dalšího rozšíření se více dočtete v sekci 6.2.

### Model metadata provider

Protože každý model v aplikaci obsahuje hodnoty Id, Deleted a Note, které by pro uživatele v aplikaci neměly být vidět, je nutné tuto situaci vyřešit. Z tohoto důvodu je v aplikaci implementován vlastní model metadata provider. Model metadata provider slouží k získání metadat o modelu a jejich vlastnostech. Metadata jsou jak informace nastavené pomocí atributů tak další informace, které pomocí atributů není možné nastavit. Mezi metadata patří například Required, PropertyName, Order, DisplayName, ShowForDisplay, ShowForEdit a právě ShowForEdit a ShowForDisplay je potřeba pro Id, Deleted a Note změnit na false. V aplikaci Šoupátka tuto funkcionalitu zajišťuje SoupatkaModelMetadataProvider.

### Automatické migrace

Jednou z vlastností Entity Frameworku jsou i databázové migrace 4.6.2. Pro účely aplikace Šoupátka jsou využity takzvané automatické migrace. Toto nastavení zajistí, že je při přístupu k databázi porovnán současný model aplikace s uloženými metadaty v databázi. Pokud je nalezen rozdíl mezi současným modelem a těmito metadaty, je vytvořena DbMigrace, která provede nad databází SQL skripty. Tyto skripty zajistí, aby se současný model aplikace shodoval s databází. Na rozdíl od klasických migrací tak není třeba ručně vytvářet novou migraci a tu ručně spustit. Pro aplikaci Šoupátka to znamená, že

se aplikace nemusí starat o to, jak databáze vypadá. Vše je vyřešeno pomocí zapnutých automatických migraci.

#### 4.8.4.2 Řadiče

Generování business logiky obsažené v řadičích aplikace vyžaduje vytvořit složitou šablonu. Dalším problémem generování řadičů je nutnost při změně business logiky nutnost znovu vytvořit všechny již vytvořené assembly. Tento problém je v Šoupátkách vyřešen pomocí ApplicationEntityControlleru.

**ApplicationEntityController**

ApplicationEntityController slouží v aplikaci jako rodičovská třída všech řadičů. Každý řadič z Aplikace tedy dědí od ApplicationEntityControlleru. Toto řešení zároveň zjednoduší logiku obsaženou v řadičích na minimum. ApplicationEntityController tak obsahuje veškerou business logiku a definici všech potřebných akcí.

#### 4.8.4.3 Pohledy

Při generování pohledů se nabízejí dvě možnosti. První možností je stejně jako u generování modelů a řadičů využít view engine. Generovat pohledy pomocí syntaxe Razor view enginu by ale znamenalo potřebu escapovat veškeré znaky @, které by měly ve výsledném pohledu být. Druhým a jednodušším řešením je vytvoření společných pohledů pro všechny řadiče. Tyto pohledy pak stačí pouze umístit do společné složky Shared a všechny vytvořené řadiče je při hledání vhodného pohledu automaticky najdou. Rozdíly mezi jednotlivými pohledy je pak možné řešit pomocí HTML helperů. Výhodou tohoto způsobu navíc je, podobně jako u použití společného ApplicationEntityControlleru, možnost upravit výsledné pohledy pro všechny vytvořené aplikace, a to bez nutnosti znovu generovat pro všechny již vytvořené aplikace nové assembly.

Protože každá uživatelem vytvořená Aplikace má svůj vlastní DbContext, a tedy vlastní modely, je třeba dále řešit, s jakými metadaty v databázi má být který model porovnán. Entity framework naštěstí nově ve verzi 6 podporuje možnost nastavit contextKey. Díky tomuto nastavení je možné pro každý DbContext nastavit jméno, pod kterým jsou pak metadata o současném modelu uložena do databáze. Jednoduše tak stačí nastavit každému DbContextu unikátní contextKey.

Automatické migrace sebou ale přináší i své nevýhody. Mezi ty hlavní patří problém s přejmenováním modelů a případně i jejich vlastností. Entity Framework v takovém případě neví, jakým způsobem databázi upravit a automatická migrace tak skončí s výjimkou Automatic migration was not applied because it would result in data loss. Tento problém Šoupátka řeší obecně dvěma způsoby. První způsob je jednoduše zakázat změny, které by

vedly k této chybě. Druhým způsobem je ošetření změn tak, že se ve skutečnosti aplikace pouze tváří že došlo k přejmenování, ale ve skutečnosti je u modelu změněn pouze takzvaný displayname atribut. Tento atribut určuje, jak se model či případně jeho vlastnost zobrazí v aplikaci a databázový název zůstává pořád stejný jako při vytvoření.

## 4.9 Použité knihovny

### 4.9.1 ASP.NET MVC SiteMapProvider

Pro generování menu a drobečkové navigace Šoupátek je použit MVC SiteMapProvider [20]. MVC SiteMapProvider je implementace ASP.NET SiteMap provideru vytvořená speciálně pro účely MVC. MVC SiteMapProvider je poskytován formou NuGet balíčku a obsahuje velké množství nastavení. Toto nastavení je možno provádět buď pomocí interního DI kontejneru, nebo pomocí externího DI kontejneru. Při konfiguraci pomocí interního kontejneru je veškeré nastavení prováděno pomocí web.configu aplikace. Na druhou stranu je možné nastavení externího kontejneru používat společně s libovolným DI kontejnerem. Využití externího kontejneru poskytuje možnost pokročilého nastavování MVC SiteMapProvideru. Pro účely aplikace Šoupátek je dostačující interní DI a tedy jednodušší nastavení.

Mvc SiteMapProvider využívá nastavení mapy aplikace pomocí XML souboru Mvc.sitemap. V této sitemapě je možné nastavit rozložení stránek webové aplikace. Pomocí tohoto souboru je pro jednotlivé stránky aplikace možné vytvořit uzel, který popisuje stránku. Tomuto uzlu je možné nastavit název, pod jakým se v aplikaci zobrazí, jeho viditelnost 4.9.1.1, routovací hodnoty a mnoho dalšího nastavení.

#### 4.9.1.1 Visibility provider

Visibility provider je součást MVC SiteMapProvideru, která se stará o zobrazení jednotlivých uzlů z sitemapy v aplikaci.

Pomocí visibility provideru je tak možné nastavit, jaké uzly z sitemapy se mají zobrazit v menu a jaké uzly se mají zobrazit v drobečkové navigaci.

Aplikace Šoupátka využívají implementaci FilteredVisibilityProvider. Tato implementace na rozdíl od základní implementace umožňuje specifikovat zobrazení uzlů pro jednotlivé grafické prvky. Je tak možné si například nastavit, že jeden uzel není v menu vidět, ale je vidět v drobečkové navigaci. Toto nastavení se provádí jednoduše přímo v sitemapě za pomoci vlastnosti uzlu visibility. Je-li tedy například potřeba zobrazit uzel pouze v drobečkové navigaci, stačí nastavit visibility="SiteMapPathHelper,!*". Toto nastavení říká: V případě, že je použit HTML helper SiteMapPathHelper (ten se stará právě o generování drobečkové navigace) zobraz tento uzel v ostatních, ale tento uzel nezobrazuj.

#### 4.9.1.2 Dynamic node provider

Součástí zadání diplomové práce je požadavek na dynamické generování položek v menu. V předchozích částech této diplomové práce je popsáno řešení vytváření řadičů a jejich pohledů. Pomocí změny URL je tedy možné k těmto

stránkám aplikace přistoupit a dále s nimi pracovat. Toto řešení ale není použitelné a je potřeba tyto stránky zobrazit v menu Aplikace. Tento problém je vyřešen pomocí dynamic node provideru. Dynamic node provider je součást MVC SiteMapProvideru, která umožňuje do výsledného XML souboru přidávat dynamicky uzly, například na základě informací z databáze. Nastavení dynamic node provideru je velmi jednoduché a skládá se ze dvou kroků. Prvním krokem je přidat (na místo, kam chceme nové uzly umístit) do sitemapy uzel a tomuto uzlu nastavit vlastnost dynamicNodeProvider na plný název třídy, která se bude o dynamické vytváření uzlů starat. Druhým krokem je implementace této třídy. Implementace spočívá v přepsání metody GetDynamicNodeCollection tak, aby vracela seznam uzlů, které do aplikace chceme přidat. Díky tomuto řešení tak uživatel v menu aplikace vidí všechny již dříve vytvořené řadiče a jejich akce.

### 4.9.2 Automapper

Další knihovnou, na které je aplikace postavena, je Automapper. Automapper je knihovna umožňující snadné mapování entitního modelu na pohledový model. Pomocí této knihovny je možné snadno mapovat jednotlivé vlastnosti entity na pohledový model, který bude použit v pohledu. Jinak řečeno je možné snadno udělat projekci databázové tabulky. Tato knihovna navíc umožňuje provádět i komplikované mapování.

## 4.10 Řešené problémy

### 4.10.1 Ztráta session

Jedním z hlavních problémů při implementaci zvoleného řešení aplikace byla ztráta session. K této ztrátě docházelo z důvodu změny adresáře bin v aplikaci. Tedy pokaždé, když některý z uživatelů provedl sestavení aplikace, byly veškeré informace uložené v session ztraceny. Proto bylo ukládání potřebných informací přesunuto z session do databáze. Veškeré potřebné informace jsou tak uloženy v databázi, kde nedochází k jejich ztrátě. Stejně tak toto řešení poskytuje uchování informací při odhlášení a novém přihlášení do aplikace. Aplikace si tak například pamatuje, zda se uživatel naposledy nacházel v Aplikaci nebo Návrháři, a je při přihlášení automaticky přesměrován do jedné z těchto částí.

### 4.10.2 Cache menu

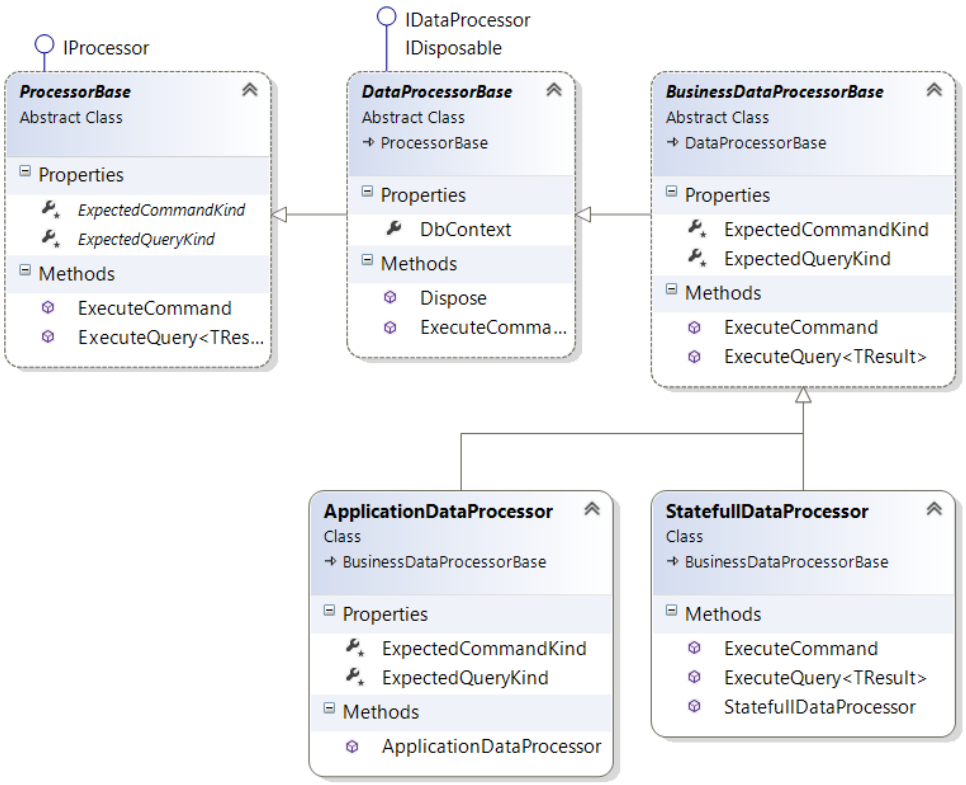
Dalším problémem, který nastal při implementaci aplikace, je nastavení cache u MVC SiteMapProvideru. Tento provider například v případě vytvoření nového modelu v aplikaci stále zobrazoval stejné položky v menu. Naštěstí je tento problém v MVC SiteMapProvideru možné řešit velmi snadno pomocí atributu, který pokud je umístěn nad některou z akcí, případně nad celý řadič, zajistí, že je cache smazána a dojde tedy k obnovení položek v menu.

### 4.10.3 Propojení navržené architektury s autentifikací

Na obrázku 3.7 je vidět výsledná navržená architektura. Nevýhodou tohoto řešení je nutnost vyřešit propojení architektury s autentifikací 4.7. Implementace Kernelu totiž vyžaduje, aby jednotlivé příkazy pracovaly s modely, které dědí od SoupatkaEntity. Implementace modelu ApplicationUser a další modely ze základní implementace autentifikace tento požadavek ale nesplňují. Ani není možné jednoduše předělat třídy tak, aby dědily od SoupatkaEntity. Třída totiž již dědí od jiné třídy, kterou je IdentityUser. Tento problém, tak musel být bohužel vyřešen přímým přístupem k modelům, které jsou součástí autentifikace. Aplikace tak v případě potřeby některého z těchto modelů přistupuje přímo k DbContextu bez využití implementovaných příkazů a dotazů.

### 4.10.4 Databázový kontext

Dalším problémem je rozlišit, nad kterým databázovým kontextem provádět dotazy. V případě, že se uživatel nachází v Návrháři, existuje pouze jeden kontext a všechny dotazy jsou prováděny nad tímto kontextem. V případě, kdy se uživatel nachází v části Aplikace, existují pro něj dva databázové kontexty. První je kontext, který se nachází i v Návrháři a slouží k získávání informací




Obrázek 4.5: Rozšířený diagram procesoru

o uživatelích, jejich aplikacích a dalším informacím. Druhý je dynamicky vytvořený databázový kontext v zkompilované assembly. Ten slouží k uchovávání jednotlivých modelů vytvořených uživatelem v Návrháři. Aby bylo možné provádět dotazy a příkazy buď nad jedním kontextem, nebo nad druhým, byla architektura rozšířena o ApplicationDataProcessor 4.5, který obsahuje Aplikační kontext. Dále pak byl BaseKernel rozšířen o logiku, kdy na základě typu příkazu respektive dotazu použije jeden nebo druhý procesor. Tedy jeden nebo druhý databázový kontext.

Kapitola 5

# Testování

## 5.1 Unit testy

Unit testy se zaměřují na nejmenší části aplikace. Pomocí unit testů je možné otestovat jednotlivé metody, třídy a jejich funkčnost. Při implementaci aplikace Šoupátka byly unit testy využity hlavně k možnosti průběžně kontrolovat funkcionalitu scaffolderu a jejich šablon. Dále pak byly použity k otestování ApplicationBuilderu, který z vygenerovaného kódu kompiluje výsledné assembly. Unit testy slouží v Šoupátkách také k otestování HTML helperů aplikace.

### 5.1.1 Scaffoldery

Hlavní část, na kterou se unit testy zaměřily, jsou scaffoldery. Tedy na součást aplikace, která z aplikačního modelu vygeneruje kód pro výsledné assembly. Otestování spočívá ve vytvoření ukázkových modelů, řadičů a databázových kontextů a v porovnání těchto souborů s výsledky od jednotlivých scaffolderů. K tomuto účelu vznikla v aplikaci pomocná statická třída pro připravení aplikačního modelu PrepareModels. Tato třída se postará o vygenerování testovacího aplikačního modelu, z kterého je následně kód pomocí scaffolderů generován.

### 5.1.2 HTML helpery

Jednou z testovaných komponent jsou HTML helpery. HTML helper je součást, která se stará o generování složitějšího HTML.

Při složitější logice je velmi jednoduché udělat chybu a zapomenout ukončovací HTML element. Proto je v aplikaci Šoupátka pro generování složitější logiky využíváno třídy tag builder. Třída tag builder umožňuje snadno definovat generované HTML a pomáhá tak psát čistější a hlavně bezchybnější kód. Přesto i tato součást byla podrobena testům a to speciálně disposable HTML

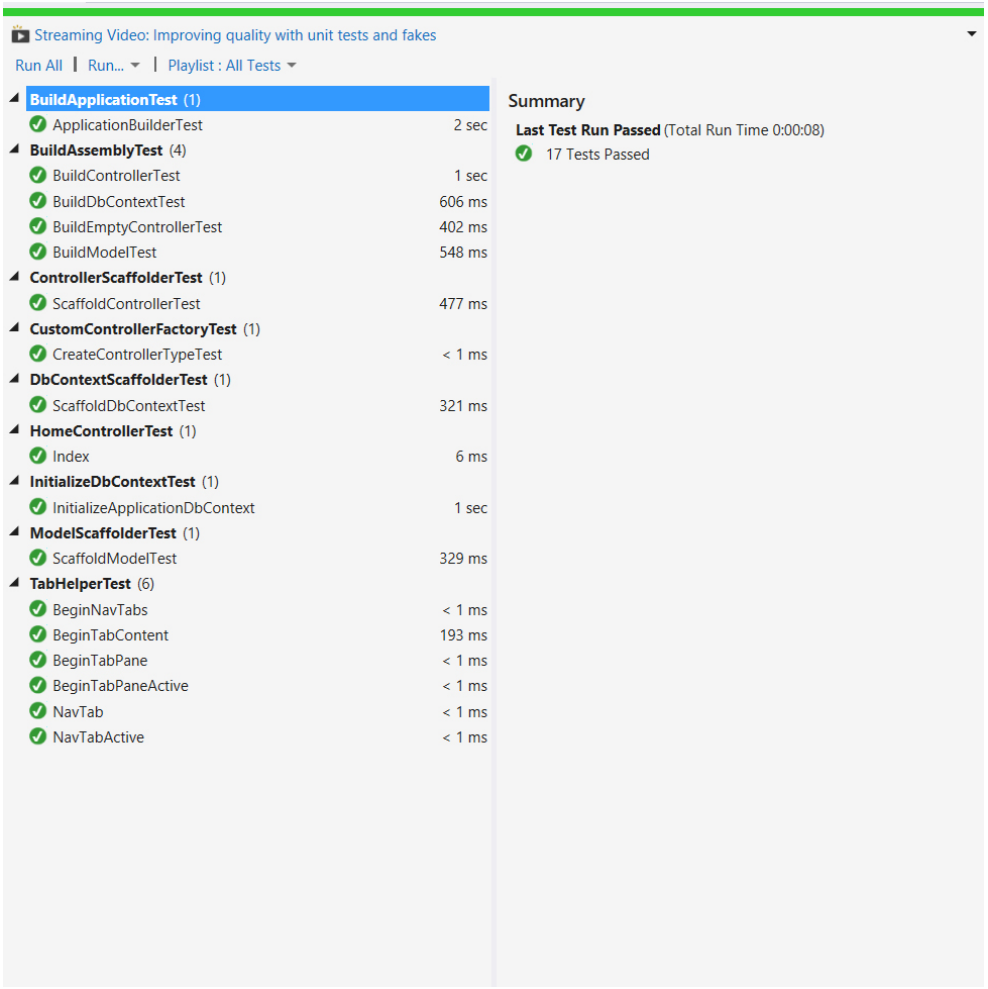
helpery, které umožňují provádět vnořování jednotlivých HTML helperů dohromady.

### 5.1.3 Disposable HTML helpery

Otestování disposable HTML helperů není jednoduché. Na rozdíl od běžných HTML helperů zapisují výsledné HTML přímo do streamu ViewContextu. Pro tyto účely tak byla implementována pomocná třída, která se stará o vytvoření vlastního falešného ViewContextu. Tento ViewContext má pomocí moq frameworku nahrazen stream za vlastní instanci memory streamu z kterého je pak možné získat výsledné HTML.

### 5.1.4 Výsledky unit testů

Celkově bylo provedeno základních 17 unit testů. Jejich výsledky můžete vidět na obrázku 5.1.




Obrázek 5.1: Unit testy

## 5.2 Uživatelské testování

Pomocí automatických testů je dobře pokryt kód aplikace. Další testování se tak zaměřilo na uživatelské testování grafického rozhraní.

### 5.2.1 Testovací scénáře:

- Registrace a přihlášení.
- Vytvořte ukázkovou aplikaci s několika modely a zkuste ve vzniklé aplikaci vytvořit ukázkové záznamy.
- Pozvěte uživatele do aplikace.
- Zkuste přijmout pozvánku do aplikace od jiného uživatele.

- Změna jazyku aplikace.

Po otestování základní funkcionality pomocí scénářů, bylo uživatelům umožněno provádět další libovolné testování aplikace.

### 5.2.2 Odhalené a následně opravené chyby

- Po smazání všech uživatelů aplikace již není možné žádného uživatele přidat.
- Možnost vytvořit dvě aplikace se stejným názvem pod jedním uživatelem. Sestavení aplikace následně končí chybou.
- Možnost vytvořit do jedné aplikace dva modely se stejným názvem. Sestavení aplikace následně končí chybou.
- Možnost vytvořit do jednoho modelu dvě vlastnosti se stejným názvem. Sestavení aplikace následně končí chybou.
- Smazání vlastnosti modelu nejde, vlastnost se v modelu pořád zobrazuje, ale v Aplikaci již správně není.
- Přepínání mezi Návrhářem a Aplikací funguje někdy až při druhém kliknutí.
- Nezobrazení datepickeru pro volbu data v Aplikaci.
- Scházející překlady.
- Části aplikace neobsahují texty ale pouze lorep ipsum.
- Nefunkční tlačítko Nastavení v menu.
- V Internet Exploreru dochází k občasnému pádu aplikace při úpravě grafického prvku pro přidávání a odebírání hodnot ze seznamu.
- V Internetu Exploreru nedojde při přepnutí jazyku ihned k aktualizaci HTML selectu.

## 5.3 Vyhodnocení testů

Aplikace byla otestována z pohledu uživatele a zároveň byl kód aplikace dostatečně pokryt unit testy. Uživatelské testy ukázaly, že aplikace obsahuje velké množství chyb a to převážně z důvodu scházející validace. Přesto, že aplikace jistě obsahuje množství dalších chyb, všechny nalezené chyby byly odstraněny.

Kapitola 6

# Dokumentace

Součástí implementace je také dokumentace. Dokumentace aplikace se zaměřuje hlavně na část aplikace, která sestavuje z uživatelem vytvořených modelů samotnou Aplikaci. Dále pak má za cíl popis Kernelu, jemuž se v této práci věnovala kapitola 3.7.1. Dokumentace Šoupátek je postavena na Doxygenu [21].

## 6.1 Doxygen

Doxygen umožňuje jednoduše zapisovat strukturované komentáře přímo do zdrojového kódu aplikace. Z těchto komentářů je pak možné jednoduše vygenerovat pomocí nástroje Doxywizard dokumentaci aplikace a to v několika formátech. Na přiloženém CD je dokumentace vygenerována do formátu Latex a hlavně HTML, který umožňuje velmi snadné pohybování se mezi jednotlivými komentáři aplikace.

Odlišení komentářů pro Doxygen od běžných komentářů je pomocí tří lomítek:

```
1 /// <summary>
2 /// View for LogOn users
3 /// </summary>
4 /// <returns>View (/Account/Logon)</returns>
5 public ActionResult LogOn()
6 {
7 return View();
8 }
```

## 6.2 Diagramy

Aplikace je dále zdokumentována pomocí diagramů. Diagramy použité v této diplomové práci jsou často pouze zjednodušenou verzí skutečných diagramů. Kompletní diagramy se nacházejí na přiloženém CD k diplomové práci. Konkrétní umístění na CD naleznete v kapitole C. Jednotlivé diagramy jsou vytvořeny v aplikaci Enterprise Architect a je tak nutná instalace tohoto nástroje pro jejich zobrazení. V projektu Soupatka.Kernel je také možné najít kompletní diagramy tříd aplikace. Diagramy jsou vygenerované přímo pomocí Visual Studia 2013 a pro jejich zobrazení je nutná verze Ultimate.

# Závěr

## Přínos práce

Na první pohled se může zdát, že aplikace Šoupátka je jen další běžnou webovou aplikací. Opak je však pravdou! Tato aplikace totiž není jen další běžnou implementací v technologii ASP.NET MVC. Šoupátka ukazují jeden z možných způsobů, jak v této technologii za pomoci EF umožnit uživateli za běhu systému definovat vlastní databázové tabulky. Splnění tohoto požadavku je v současné době velmi komplikované. EF sám o sobě nepodporuje možnost dynamicky vytvářet entity a je tak třeba hledat způsob, jak tento požadavek vyřešit. Práce ukázala dvě možná řešení, jak takovýto požadavek splnit a zároveň navrhnula vlastní neověřené řešení. Cílem této diplomové práce tak je ukázat a ověřit tento způsob. Dalším cílem bylo implementovat aplikaci, která bude tento způsob využívat. Tato diplomová práce tak může sloužit jako základní stavební kámen pro webové programátory, kteří se v dané technologii setkali s podobnými požadavky.

Dalším přínosem práce je navrhnuté jádro aplikace. Toto jádro je navrženo tak, aby bylo velmi snadno použitelné i pro další webové aplikace psané za pomoci technologie ASP.NET MVC. Celé jádro je odděleno do vlastního projektu Soupatka.Kernel a je tak možné toto jádro využít pro další aplikace pouze přidáním reference na tento projekt. Umožňuje tak snadnou znovupoužitelnost implementovaného kódu.

Práce může být dále zajímavá i pro programátory, kteří řeší implementaci lokalizace v technologii ASP.NET. Popisuje jak lokalizovat jednotlivé části aplikace, ať už lokalizaci jednotlivých pohledů, položek v menu, tak například i lokalizaci enumerátorů v aplikaci. Na základě těchto informací je tak možné implementovat lokalizaci do vlastní aplikace.

Poslední přínos práce je samotná vzniklá aplikace. Nejde totiž o běžnou aplikaci, která by sloužila pouze k jednomu účelu, a její možnosti využití jsou obrovské. A to i přesto, že je v aplikaci velké množství prostoru na vylepšení 6.2.

# Zhodnocení výsledků

Při vytváření zadání této diplomové práce byla jedním z hlavních problému nejistota, zda je vůbec možné splnit požadavek na dynamické vytváření modelů za běhu aplikace. Přesto, že již v době zadání existovala základní představa o možnosti řešení tohoto požadavku, bylo třeba řešení tohoto požadavku ověřit. Jednotlivé části řešení totiž samy o sobě požadují pokročilé znalosti jednotlivých součástí na kterých je řešení postaveno a i když proběhl návrh aplikace, je spousta funkcionalit, jejichž implementace nebyla navržena a v době návrhu ani nebylo možné takto do detailů návrh provést. Spousta součástí aplikace je dokonce postavena na funkcích, které v použitých knihovnách v době zadání diplomové práce ani neexistovaly. Například nastavení ContextKey pro DbContext je součástí Entity Frameworku od verze 6. Přes všechny tyto nejistoty se ale povedlo vytvořit na základě navržené architektury funkční aplikaci. Tato aplikace navíc nemusela ustoupit ani jednomu z požadavků anebo pozměnit některou část navrženého konceptu funkcionality aplikace. Implementace provedla jak potvrzení koncepce, tak ukázala spoustu dalších možností jak logiku aplikace rozšířit o další funkcionalitu. Proto může být výsledek této práce považován za více než úspěšný.

# Splnění požadavků

## Přidávání modelů, pohledů a řadičů z grafického prostředí aplikace

Bylo vytvořeno grafické prostředí s možností přidávat jednotlivé modely. Na základě modelů jsou již dále vygenerovány pohledy a řadiče automaticky, a není tak pro ně potřeba vlastní GUI.

## Možnost změn vlastností modelů

Kromě vytvoření nových modelů je možné provádět i jednoduché úpravy. Mezi ty podporované patří přidávání nových vlastností, smazání vlastnosti a případně změna názvu vlastnosti. Není možné provádět změny vlastností mezi jednotlivými typy.

## Vhodné grafické komponenty pro úpravu údajů

Aplikace podporuje jen několik základních datových typů. Pro jejich nastavení stačí ve většině případů běžné HTML input elementy. Dále pak obsahuje Datepicker pro možnost volby data a checkbox pro hodnoty Ano/Ne.

### Možnost konfigurace rozložení grafických komponent v pohledech

Uživatel může snadno nastavit pořadí jednotlivých grafických komponent v pohledech a rozhodnout se, zda chce komponentu umístit na jeden z tabů nebo do hlavní části pohledu.

### Dynamické generování položek v menu

Na základě vytvořených modelů je v Aplikaci zobrazeno menu obsahující akce, které je možné nad modely provádět. Tímto požadavkem se zabývala část práce o MVC SiteMapProvider 4.9.1.

### Správa aplikací

Správa aplikací umožňuje aplikace vytvářet, upravovat, smazat. Dále pak umožňuje změnit uživatele aplikace a upravovat jejich modely.

### Správa modelů

Modely je možné, jak již bylo řečeno, z grafického prostředí aplikace vytvářet, upravovat a smazat.

### Automatická aktualizace databáze

Na základě vytvořených modelů je automaticky provedena migrace databáze na verzi odpovídající modelu 4.8.4.1.

### Verzování vytvořených assemblies

Při každém sestavení nové aplikace je možné nastavit verzi další assembly. Defaultní implementace zvedá při každém sestavení Aplikace část Build version z verze assembly o jedna.

### Rychlé přepínání mezi aplikací a modelem

Do grafického prostředí bylo přidáno tlačítko pro snadné přepínání mezi Návrhářem a Aplikací 4.2.

### Možnost pozvat uživatele do aplikace

Při vytváření aplikace je možné nastavit e-maily uživatelů, na které přijde pozvánka do nově vytvořené aplikace. Pro nastavený e-mail se následně v aplikaci zobrazí uživateli notifikace o pozvání do aplikace, kterou může přijmout 4.3.

### Lokalizace do českého a anglického jazyku

Jak Aplikace, tak Návrhář umožňují přepínat mezi českým a anglickým jazykem. Toto nastavení se navíc ukládá do uživatelského profilu, a proto je poslední známé nastavení jazyku obnoveno při novém přihlášení do aplikace 4.4.

### Zhodnocení požadavků

Všechny požadavky na aplikaci vyplývající jak ze zadání, tak z analýzy, byly do jednoho splněny. Aplikaci je tedy i z tohoto pohledu možné považovat za kompletní.

# Další postup

Implementovaná aplikace Šoupátka je určena pro širokou veřejnost. Mezi nejběžnější očekávané zákazníky patří malé firmy s nízkým počtem zaměstnanců, které nechtějí investovat do vývoje vlastního systému. Stejně tak jsou Šoupátka vhodným řešením v případě, kdy se předpokládá využití aplikace pouze na omezené období. Počítá se s tím, že samotná aplikace by měla být poskytována za stanovený měsíční poplatek. V současnosti ale není rozhodnuto, jak vysoká by tato částka měla být a počítá se s nasazením základní verze aplikace, která bude poskytována zcela zdarma. Na základě uživatelských reakcí pak bude stanovený další postup. Bohužel k dnešnímu dni není aplikace nasazená a není tak možné si jí vyzkoušet. Před samotným nasazením se počítá ještě s dalšími změnami, které aplikaci vylepší a zpřístupní jí tak širší skupině zájemců.

# Možnosti dalšího rozšíření

Přesto, že aplikace již splňuje všechny požadavky ze zadání diplomové práce, je v ní několik míst, kde je ji možno vylepšit a případně se počítá s jejím rozšířením. Hlavním takovým místem je přední část aplikace.

## Podporované typy

Aplikace v současnosti podporuje pouze několik základních datových typů. Tyto typy dokážou pokrýt drtivou většinu scénářů, ale zvolená bootstrap šablona poskytuje velké množství dalších grafických komponent pro práci s daty. Jedním z takových příkladů je GUI pro nastavení času nebo možnost zvolit barvu. Toto rozšíření by navíc poskytovalo aplikaci mnoho dalších možností, jako například umožnit uživateli změnu barev v aplikaci a to jak například barvu menu tak i komplikovanější nastavení jako je změna barvy jednotlivých záznamu v seznamovém pohledu.

## Vazby

V relačních databázích je možné mezi jednotlivými tabulkami definovat vazby. Entity Framework umožňuje tyto vazby jednoduše definovat přímo do modelu. Jedním z možných rozšíření je tak vytvořit grafické rozhraní pro definici těchto vazeb a dále pak rozšířit šablonu pro generování kódu o tuto definici. Jednotlivě vytvořené modely by tak mezi sebou mohly mít libovolné vazby, což by umožnilo aplikaci rozšířit o další funkcionalitu.

## Správa uživatelů

Současná správa uživatelů je velmi jednoduchá. Umožňuje pouze přidat či odebrat uživatele do aplikace. I přesto, že je již zadní část na tuto možnost

připravena, není možné například rozlišit, kdo je aplikační administrátor a kdo je jen její uživatel. Všichni uživatelé tak mohou do aplikace libovolně zasahovat a upravovat ji. S tím souvisí i možnost nastavení práv v části Aplikace.

### Práva

Budoucí řešení počítá s možností nastavení práv pro jednotlivé uživatele. Ať už formou přiřazení role, tak případně s možností nastavit práva jednotlivým uživatelům. V budoucnosti tak může existovat například uživatelská role, která nebude moci jednotlivé entity upravovat, ale bude si je moci pouze prohlížet.

### Uživatelský profil

Jednou z předních částí aplikace, která musela být implementována, je uživatelský profil. Color Admin ve verzi 1.5, na kterém je celá aplikace postavená, neobsahoval grafický návrh uživatelského profilu a byl tak vytvořen jednoduchý vlastní profil. V době psaní této diplomové práce byl již vydán Color Admin 1.7. Tato verze rozšiřuje implementaci o několik funkcionalit mezi kterými nechybí právě uživatelský profil.

### Menu

Pro snadné rozlišení polohy uživatele je v aplikaci upraveno menu tak, aby v případě, kdy se uživatel nachází v Návrháři, mělo tmavou barvu, zatímco v Aplikaci je menu světlé. Stejně tak jako u uživatelského profilu bohužel zvolená bootstrap šablona neobsahovala možnost změnit tímto způsobem barvu. Bylo tak třeba upravit aplikaci tak, aby tuto změnu umožnila. Od verze 1.7 COLOR ADMIN byla šablona rozšířena o možnost zvolit si právě takové světlé menu, které je oproti implementaci v Šoupátkách graficky povedenější. Proto jednou z budoucích změn je změna vlastní implementace za tuto.

### Lokalizace

Posledním příkladem součásti, která v použité verzi šablony není podporována, je lokalizace. Přesněji schází grafické rozhraní pro změnu jazyku. I tuto součást tedy bylo potřeba implementovat a úplně stejně, jako v případě menu a uživatelského profilu, je v budoucnu počítáno s náhradou i této součásti.

### Notifikace

Součástí implementace aplikace je i část s notifikacemi. Zde jsou v současnosti zobrazovány pouze informace o pozvánkách do aplikace a o tom, který uživatel kterou pozvánku do aplikace přijmul. Uživatel tak má přehled o historii jednotlivých akcí v aplikaci. Tato část aplikace by měla být v budoucnu rozšířena

o další notifikace. A to například o notifikaci o chybě v aplikaci, uživatelských akcích a hlavně o notifikace ze změnových 6.2 a časových akcí 6.2.

### Změnové akce

Dalším možným rozšířením do budoucnosti jsou takzvané změnové akce. Tyto akce by měly uživateli umožnit si nastavit možnost reagovat na změny nad jednotlivými entitami. Reagovat na změny by mělo být možné několika způsoby. První ze způsobů je zobrazit určitému uživateli notifikaci o změně, případně mu zaslat e-mail. Druhou možností je nastavit při změně entity automatické provedení další změny, nebo třeba provedení speciální validace. Jednoduchým příkladem je možnost vytvořit si v aplikaci upozornění na smazání záznamu. Ve chvíli, kdy kupříkladu nějaký uživatel smaže záznam, jiný nastavený uživatel dostane o této akci zprávu a může na ní reagovat.

### Časové akce

Podobně jako změnové akce by měly fungovat i časové akce. Ty ale na rozdíl od změnových akcí nereagují na změny entit, ale pouze na nastavení času. Uživatel by si tak mohl nastavit čas, kdy chce být informován o hodnotách v určité entitě a ve stanovený čas, by mu stejně jako u změnových akcí, tato informace přišla na e-mail, případně se zobrazila formou notifikace. Stejně tak by si uživatel mohl nastavit provedení změny záznamu v nastavený čas.

Aplikace navíc již v současnosti podporuje v modelu uchovávání informace typu Datum a čas. Místo nastavení pevného času by akce mohly být prováděny i na základě tohoto nastaveného času.

# Literatura


[1] *GenerateExecutable.* MICROSOFT. Msdn [online]. [cit. 2015-05-02]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.codedom.compiler.compilerparameters.generateexecutable%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396

[2] *GenerateInMemory.* MICROSOFT. Msdn [online]. [cit. 2015-05-02]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.codedom.compiler.compilerparameters.generateinmemory%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396

[3] *OutputAssembly.* MICROSOFT. Msdn [online]. [cit. 2015-05-02]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.codedom.compiler.compilerparameters.outputassembly%28v=vs.110%29.aspx

[4] *ReferencedAssemblies.* MICROSOFT. Msdn [online]. [cit. 2015-05-02]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.codedom.compiler.compilerparameters.referencedassemblies%28v=vs.110%29.aspx

[5] *Unknown Class Dynamically Generated at runtime.* Code Project [online]. [cit. 2015-05-02]. Dostupné z: http://www.codeproject.com/Articles/839640/Unknown-Class-Dynamically-Generated-at-runtime-for

[6] *MagicDbModelBuilder.* Github [online]. [cit. 2015-05-02]. Dostupné z: https://github.com/maxbeaudoin/MagicDbModelBuilder

[7] ERICH GAMMA, Richard Helm. *Design patterns CD: elements of reusable object-oriented software*. cd-rom udgave. s.l.: Addison-Wesley, 1998. ISBN 978-020-1634-983.

[8] *CQRS.* Martin FOWLER [online]. [cit. 2015-05-02]. Dostupné z: http://martinfowler.com/bliki/CQRS.html

[9] *Color Admin.* Wrapbootstrap [online]. [cit. 2015-05-02]. Dostupné z: https://wrapbootstrap.com/theme/color-admin-admin-template-front-end-WB0N89JMK

[10] *Resources in .Resx File Format.* MICROSOFT. [online]. [cit. 2015-05-02]. Dostupné z: https://wrapbootstrap.com/theme/color-admin-admin-template-front-end-WB0N89JMK

[11] *Table plug-in for jQuery.* DataTable [online]. [cit. 2015-05-02]. Dostupné z: https://www.datatables.net/

[12] *ImgAreaSelect.* Odyniec [online]. [cit. 2015-05-02]. Dostupné z: http://odyniec.net/projects/imgareaselect/

[13] *Font Awesome.* Fortawesome [online]. [cit. 2015-05-02]. Dostupné z: http://fortawesome.github.io/Font-Awesome/

[14] *ASP.NET Data Access Options.* MICROSOFT. [online]. [cit. 2015-05-02]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms178359%28v=vs.110%29.aspx

[15] *EF7 - What Does "Code First Only" Really Mean.* MICROSOFT. [online]. [cit. 2015-05-02]. Dostupné z: http://blogs.msdn.com/b/adonet/archive/2014/10/21/ef7-what-does-code-first-only-really-mean.aspx

[16] *Custom Resource Provider.* Code Project [online]. [cit. 2015-05-02]. Dostupné z: http://www.codeproject.com/Articles/33875/Custom-Resource-Provider//

[17] *Introduction to ASP.NET Identity.* MICROSOFT. ASP.NET [online]. [cit. 2015-05-02]. Dostupné z: http://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity

[18] *Open Web Interface for .NET.* OWIN [online]. [cit. 2015-05-02]. Dostupné z: http://owin.org/

[19] *ASPX View Engine VS Razor View Engine.* C# Corner. [online]. [cit. 2015-05-02]. Dostupné z:http://www.c-sharpcorner.com/UploadFile/ff2f08/aspx-view-engine-vs-razor-view-engine/

[20] *MvcSiteMapProvider.* Github [online]. [cit. 2015-05-02]. Dostupné z: https://github.com/maartenba/MvcSiteMapProvider

[21] *Generate documentation from source code.* Doxygen [online]. [cit. 2015-05-02]. Dostupné z: http://www.stack.nl/~dimitri/doxygen/

Příloha **A**

# Seznam použitých zkratek

**GUI** Graphical user interface

**XML** Extensible markup language

**MVC** Model-view-controller

**ORM** Object-relational mapping

**API** Application Programming Interface

**CRUD** Create, Read, Update, Delete

**CQRS** Command Query Responsibility Segregation

**OWIN** Open Web Interface for .NET

**SQL** Structured Query Language

**UML** Unified Modeling Language

**CSS** Cascading Style Sheets

**HTML** HyperText Markup Language

**MS** Microsoft

**PDF** Portable Document Format

**DI** Dependency Injection

**URL** Uniform Resource Locator

**EF** ADO.NET Entity Framework

**ASP** Active Server Pages

**LINQ** Language Integrated Query

Příloha B

# Uživatelská příručka

## B.1 Instalace aplikace

Pro spuštění aplikace na vlastním stroji jsou doporučené následující nástroje:

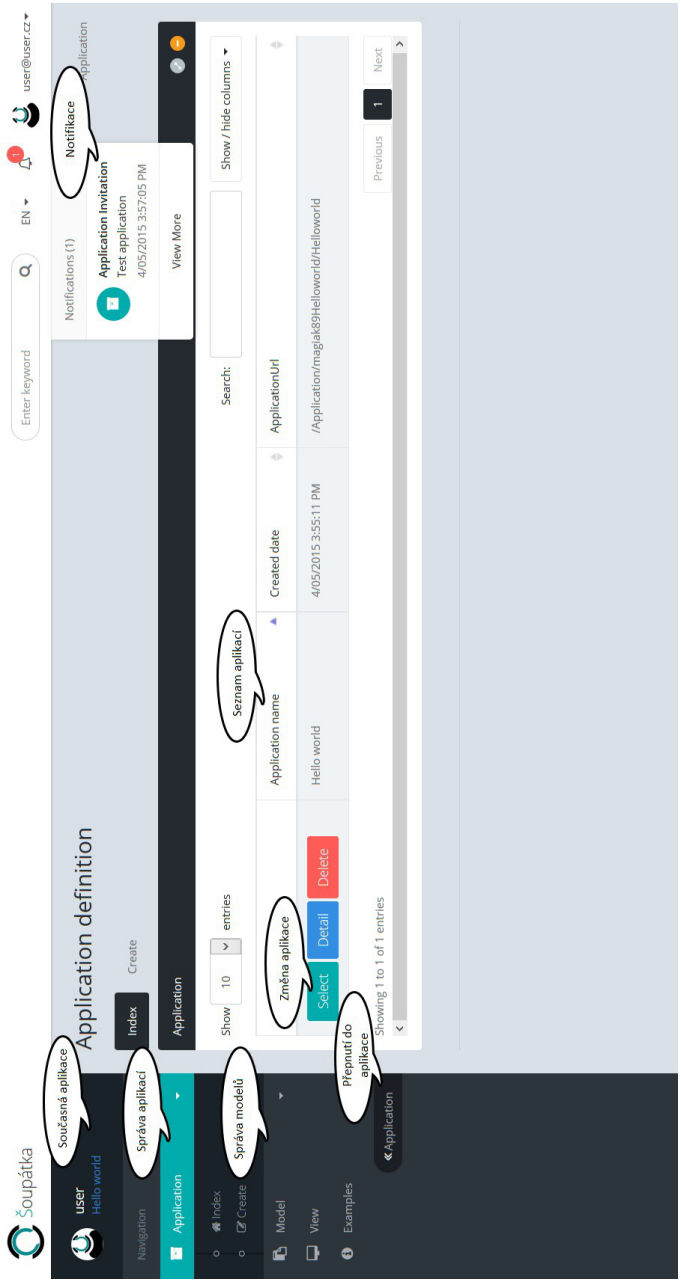
- Microsoft Visual Studio 2013 Ultimate
- .NET 4.5
- Microsoft SQL Server 2012 Express

Microsoft Visual Studio 2013 ve verzi Ultimate není nutností. Tato verze je pouze doporučená v případě kdy chce mít uživatel možnost vidět diagramy aplikace a výsledky zátěžových testů, které jsou součástí pouze verze Ultimate.

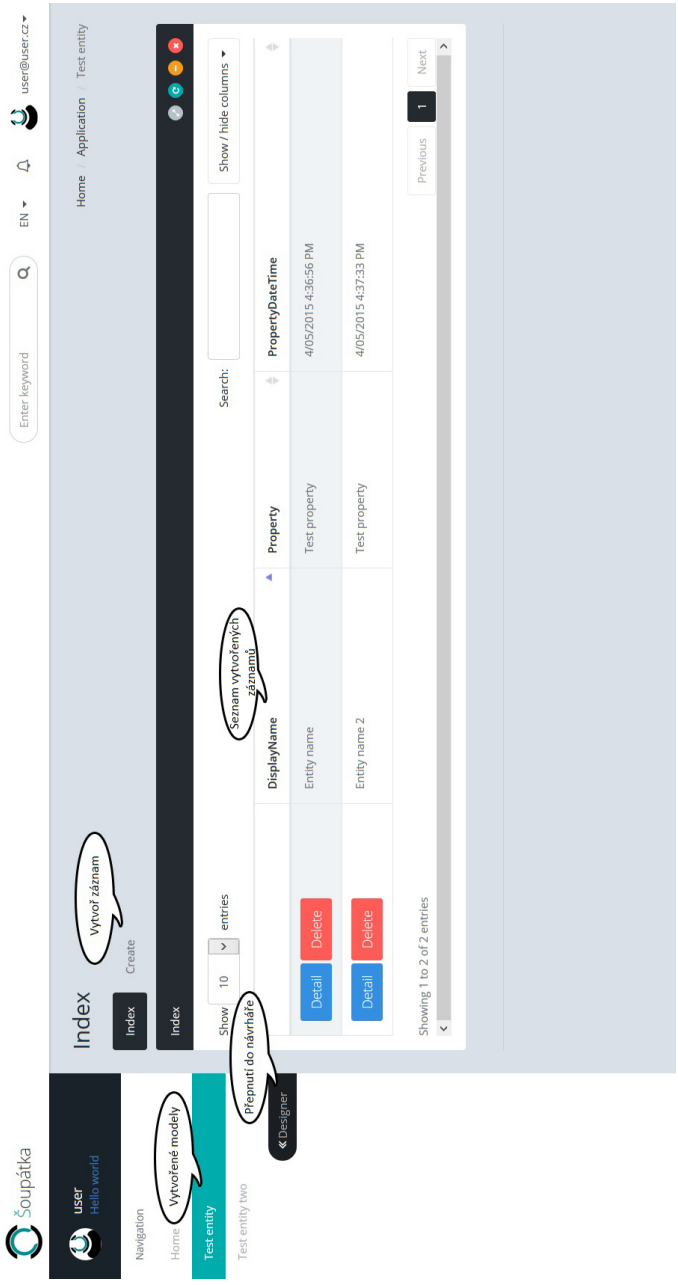
Stejně tak by aplikace měla bez problémů fungovat i na starší verzi Microsoft SQL Serveru. V případě specifické konfigurace SQL Serveru je ale nutné změnit nastavený connection string v souboru web.config. Současné nastavení počítá s instalací SQL Serveru jako defaultní instance.

Instalace .NET 4.5 je součástí instalace Visual Studia 2013. Případně je možné ji stáhnout ze stránek microsoft.com

Veškeré další potřebné nástroje, na kterých je aplikace postavena, jsou dostupné formou Nuget balíčků. Příslušný seznam potřebných balíčků pro jednotlivé projekty je možné najít v packages.config. Při prvním spuštění aplikace by ale mělo dojít k instalaci těchto balíčků automaticky.




Obrázek B.1: Návrhář




Obrázek B.2: Aplikace

Příloha C

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `dokumentace` ..... dokumentace implementace
  - `html` ..... dokumentace ve formátu HTML
  - `latex` ..... dokumentace ve formátu LaTeX
  - `diagramy` ..... diagramy aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Lukas_Kmoch_2015.pdf` ..... text práce ve formátu PDF
  - `DP_Lukas_Kmoch_2015.ps` ..... text práce ve formátu PS