VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

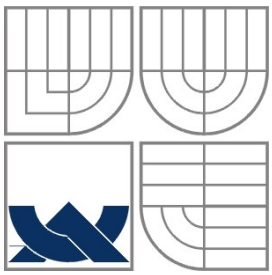
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATICKÝ DESIGN WEBOVÝCH APLIKACÍ

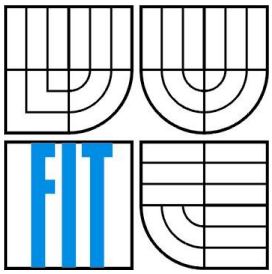
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Radek Žilka
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATICKÝ DESIGN WEBOVÝCH APLIKACÍ

AUTOMATIC DESIGN OF WEB APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Radek Žilka
AUTHOR

VEDOUCÍ PRÁCE — Ing. Ruttkay Ladislav
SUPERVISOR

BRNO 2009

Abstrakt

Tato práce se zabývá návrhem a implementací softwaru, který slouží jako generátor programového kódu pro vývoj webových informačních systémů vyvíjených pomocí technologie ASP.NET 2.0. Tyto systémy využívají jako svůj datový zdroj především SQL databázi. Zdrojové kódy vygenerované aplikace jsou základem pro další vývoj ve Visual Studio 2008. Práce se také zabývá zkoumáním podobných existujících generátorů.

Klíčová slova

Internetová aplikace, ASP.NET, Microsoft Visual Studio, generátor kódu, šablony, databáze, SQL, návrhové vzory, informační systém, C#, Microsoft .NET

Abstract

This project is focused on software design and implementation. This software represents a generator of web information system source code developed in ASP.NET 2.0. The information system requires the SQL database. The source code generated by this application is the base of the next development in Visual Studio 2008. This project inspects similar generators too.

Keywords

Web application, ASP.NET, Microsoft Visual Studio, code generator, templates, database, SQL, design patterns, information system, C#, Microsoft .NET

Citace

Žilka Radek: Automatický design webových aplikací. Brno, 2009, diplomová práce, FIT VUT v Brně.

Automatický design webových aplikací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Ladislava Ruttkay.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radek Žilka
20.05.2009

Poděkování

Děkuji mému vedoucímu panu Ing. Ladislavu Ruttkayovi za přípravu a pomoc při realizaci tohoto projektu a za jeho přátelský přístup v závěru studia na Fakultě informačních technologií, VUT v Brně.

Obsah

1 Úvod

V poslední době se mnoho aplikací přesouvá z klientských počítačů na internetové stránky. Není potřeba nic instalovat a uživatel pak může využívat služby poskytované různými portály z jakéhokoliv počítače připojeného k internetu. Velké množství informací, které weby poskytují jsou zpravidla uložené v databázích a není to případ pouze rozsáhlých portálů. Programátoři a vývojáři, kteří vytvářejí tyto aplikace stále postupují velmi podobným způsobem při řešení každého nového projektu. Pro úsporu času by bylo výhodné do určité míry tuto činnost zautomatizovat.

Úkolem této práce je zanalyzovat, navrhnout a implementovat internetovou aplikaci pro automatické generování webových portálů. Přesněji řečeno, jedná se o generátor, který vytváří aplikaci podle předem navržené databáze. V tomto diplomovém projektu jsem se zaměřil na databázi Microsoft SQL a technologii ASP.NET. Aplikace se připojí k databázi a programátor si pak zvolí, tabulky pro generování objektů a ASP.NET stránek, které poskytnou grafické uživatelské rozhraní pro základní operace s daty, jakými jsou prohlížení, vkládání, mazání a editace.

V prvních kapitolách se seznámíme s technologií ASP.NET a dalšími prostředky, které jsou v této práci použity. Najdete zde i část zabývající se architekturou softwarových systémů, která je důležitá nejen pro vytvářenou aplikaci tohoto projektu, ale především pro generovaná řešení na základě databáze, která budou výstupem naprogramované aplikace. V závěru teoretického rozboru jsou popsány některé existující aplikace.

V analýze se specifikují základní funkce systému, které jsou zachyceny v diagramu použití. Vzhledem k tomu, že se jedná o internetovou aplikaci, zmíním se i o bezpečnosti. Kapitola návrhu rozděluje aplikaci do tří vrstev kvůli snadnějšímu budoucímu vývoji a rozšíření. Strukturu jednotlivých vrstev zachycují diagramy tříd, které jsou jednou z nejdůležitějších částí návrhu.

Testování aplikace je provedeno nad jednoduchou databází, která obsahuje tabulky s běžnými datovými typy hodnot a vazbami mezi tabulkami, které jsou reprezentovány cizími klíči. Tím je ukázána univerzálnost této aplikace. Vzhledem k tomu, že se na tomto projektu podílí pouze jediný autor, není aplikace tolik rozsáhlá a univerzální jako komerční software. Před samotným závěrem jsou popsána i rozšíření, která by mohla být velkým přínosem a námětem pro další studentské práce.

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků, zhodnocení z pohledu dalšího vývoje projektu a náměty vycházející ze zkušeností autora s řešeným projektem.

Tato práce nenavazuje na semestrální projekt, byla vytvořena úplně od začátku jako diplomový projekt.

2 Teoretický rozbor

Velký rozmach internetových aplikací s sebou přináší prosazování stále většího množství programovacích jazyků v této doméně. Jedni vsází na zcela interpretované jazyky, jako např. PHP a na jiných místech můžeme vidět plně typované a předkompilované jazyky jakým je Java nebo C#. S každou technologií je svázáno i určité vývojové prostředí, které většina programátorů zná, a které také do jisté míry ovlivňuje rozhodování v čem aplikace vytvářet. K vývojovým prostředím lze získat zásuvné moduly, doplňky třetích stran. Robustní softwarové balíky jsou často náročné na výkon a elegantní prostředí, které oslňují svoji rychlostí a jednoduchostí neumí zdaleka všechno, co by programátor a tvůrce potřeboval. Není neobvyklé vygenerovat z diagramu tříd kód v daném jazyce, i když o kvalitě se dá někdy spekulovat. Ještě možná lehčí je z entitně-relačního diagramu vygenerovat SQL dotazy pro vznik nové databáze. To může být začátek pro novou aplikaci, které předchází návrh struktury systému, což je činnost velmi těžko automatizovatelná. Od návrhu struktury lze postupovat strojově podle modelu nové aplikace do té doby, dokud se nezačne implementovat její logika a specifická funkční část. Úkolem tohoto projektu je zautomatizovat vytváření programového kódu na základě návrhu struktury.

V následujících podkapitolách se můžete dozvědět, jak k tomuto problému přistupují různá vývojová prostředí pracující s odlišnými technologiemi.

2.1 Technologie pro vytváření webových aplikací

Na trhu webových aplikací se setkáte s technologií Java EE, JSP a JSF stránkami. Jejich velkou konkurencí je aplikace běžící nad platformou .NET, resp. ASP.NET, která není až tak závislá na programovacím jazyku, ve kterém se aplikace vytváří. Úplně jiným způsobem se vytváří webové aplikace pomocí programovacího jazyka PHP, který se lze snadno naučit, ale programátor musí více dbát na určitá pravidla, protože jazyk není typovaný a tudíž se mnoho chyb může vyskytnout až za běhu aplikace.

2.1.1 PHP

PHP je programovací jazyk určený především pro programování dynamických internetových stránek. Jedná se o jazyk interpretovaný, jeho skripty se začleňují přímo do struktury jazyka HTML a jsou prováděny na straně serveru. PHP je nezávislý na platformě. Obsahuje rozsáhlé knihovny funkcí pro zpracování textu, grafiky, práci se soubory, přístup k několika databázovým serverům, podporu celé řady internetových protokolů (HTTP, SMTP, SNMP, FTP, IMAP, POP3, LDAP, …).

PHP se stalo velmi oblíbeným nástrojem především díky jednoduchosti použití a tomu, že kombinuje vlastnosti více programovacích jazyků a nechává tak vývojáři částečnou svobodu

v syntaxi. V kombinaci s databázovým serverem MySQL a webovým serverem Apache je často využíván k tvorbě webových aplikací. S verzí PHP 5 se výrazně zlepšil přístup k objektově orientovanému programování. V další verzi PHP 6 by se měla objektovost ještě zlepšit.

Protože PHP je určeno k programování na straně serveru, je potřeba, aby byl spuštěn webový server s podporou PHP. V dnešní době je jeden z nejpoužívanějších již zmíněný Apache, provozovaný obvykle na platformě UNIX. Nelze opominout ani IIS (Internet Information Server) pro operační systémy Microsoft Windows. Nejen pro testovací účely je vhodné využít kompletní softwarové balíky, které obsahují internetový server, PHP a MySQL. Programátor se tak nemusí zabývat zbytečným nastavováním jednotlivých částí, ale zároveň se připravuje o znalost, co všechno server umožňuje a v budoucnu se tomu pravděpodobně nevyhne.

2.1.2 Java

Java Platform, Enterprise Edition (neboli Java EE, dříve označovaná jako Java 2 Enterprise Edition nebo J2EE) je součást platformy Java určená pro vývoj a provoz podnikových aplikací a informačních systémů. Základem pro platformu Java EE je platforma Java SE, nad ní jsou definovány součásti tvořící Java EE.

Jednotlivé součásti platformy Java EE jsou definovány pomocí dílčích specifikací, které jsou vytvářeny ve spolupráci více firem v rámci tzv. Java Community Process (JCP). Vlastní Java EE je poté definována zastřešující specifikací opět vyvíjenou v rámci JCP.

Součástí platformy Java EE jsou především specifikace pro:

- vývoj webových aplikací – Java Servlets, Java Server Pages (JSP),
- vývoj sdílené business logiky – Enterprise Java Beans (EJB), Spring,
- přístup k legacy systémům[i] – Java Connector Architecture (JCA), Hibernate,
- přístup ke zprávovému middleware[ii] – Java Messaging Services (JMS),
- podpora technologií Web Services.

Aplikace pro platformu Java EE jsou vyvíjeny na základě API a dalších fragmentů definovaných v jednotlivých specifikacích. Běhovým prostředím pro tyto aplikace je poté tzv. Aplikační Server (dále AS). Tyto AS jsou dodávány různými dodavateli, aplikace by teoreticky měla být provozovatelná na kterémkoliv AS kteréhokoliv dodavatele implementujícím příslušnou verzi specifikace – koncept přenositelnosti. Většina AS však doplňuje některé vlastnosti nad rámec specifikace a aplikace využívající těchto vlastností poté nejsou přenositelné. [1]

i Legacy systém je starší, někdy i zastaralý systém, který je ale stále nutné provozovat z různých důvodů

ii Middleware je software, který slouží jako konverzní nebo překladatelská vrstva.

2.1.3 ASP.NET, C#

Především je vhodné zmínit, co je to .NET Framework. Je to zastřešující název pro soubor technologií v softwarových produktech, které tvoří celou platformu, která je dostupná nejen pro Web, Windows i Pocket PC. Základní komponentou je Microsoft .NET Framework, prostředí potřebné pro běh aplikací nabízející jak spouštěcí rozhraní, tak potřebné knihovny. Pro běh aplikací vyvíjených pro .NET lze použít i framework Mono. Ten je produktem nezávislé open source iniciativy, implementující .NET runtime pro operační systémy unixového typu (Linux, MacOS X). Dále lze na unixových platformách využít DotGNU. Platforma .NET nepředepisuje použití žádného programovacího jazyka. Bez ohledu na to, v čem byla aplikace původně napsána, se vždy přeloží do mezijazyka Common Intermediate Language. Nejpoužívanější programovací jazyky pro vývoj .NET aplikací jsou C#, Visual Basic .NET a Delphi. C# je programovací jazyk podobný jazykům C nebo Java. VB.NET je pokračovatelem jazyka Visual Basic.

Ačkoliv název ASP.NET je odvozen od starší technologie pro vývoj webů ASP, obě technologie jsou velmi odlišné. ASP.NET je založen na CLR (Common Language Runtime), který je sdílen všemi aplikacemi postavenými na .NET Frameworku. Programátoři tak mohou realizovat své projekty v jakémkoliv jazyce podporujícím CLR, např. Visual Basic.NET, JScript.NET, C#, Managed C++, ale i mutace Perlu, Pythonu a další. Aplikace založené na ASP.NET jsou také rychlejší, neboť jsou předkompilovány do jednoho či několika málo DLL souborů, na rozdíl od ryze skriptovacích jazyků, kde jsou stránky při každém přístupu znovu a znovu parsovány.

ASP.NET ulehčuje programátorům přechod od programování klasických aplikací pro Windows do prostředí webu: stránky jsou poskládány z objektů, ovládacích prvků (Controls), které jsou protějškem ovládacích prvků ve Windows. Při tvorbě webových stránek je tedy možné používat ovládací prvky jako tlačítko (Button), nápis (Label) a další. Těmto prvkům lze přiřazovat určité vlastnosti, zachytávat na nich události, atd. Tak, jako se ovládací prvky pro Windows samy kreslí do formulářů na obrazovku, webové ovládací prvky produkují HTML kód, který tvoří část výsledné stránky poslané do klientova prohlížeče.

Ačkoliv webový protokol HTTP je sám o sobě bezstavový, událostmi řízené programování zachování stavu (uchování kontextu mezi jednotlivými požadavky) vyžaduje. ASP.NET tento problém řeší kombinací HTML a JavaScriptu pomocí dvou základních technik:

- ViewState uchovává informace mezi opakovaným odesíláním formuláře na server v zakódovaném tvaru ve skrytých formulářových polích. Jeho výhodou je, že využívá pouze HTML a nevyžaduje žádnou speciální podporu na straně serveru ani klienta. Nevýhodou je, že se mezi serverem a klientem přenáší větší objem dat, zejména je-li ViewState využíváno nesprávně.
- Session State oproti tomu ukládá veškeré informace na straně serveru a předává (typicky jako cookie nebo součást URL) pouze jednoznačný identifikátor. To sice zmenšuje objem

přenášených dat, ale klade vyšší nároky na výkon serveru. Pokud se sessions používají nesprávně, může být server náchylný i k Denial of Service[i] útokům. [2]

2.1.4 Databázové systémy

Oracle je systém řízení báze dat (Oracle database management system – DBMS), moderní multiplatformní databázový systém s velice pokročilými možnostmi zpracování dat, vysokým výkonem a snadnou škálovatelností.

Tento systém podporuje nejen standardní relační dotazovací jazyk SQL podle normy SQL92, ale také proprietární firemní rozšíření Oracle (např. pro hierarchické dotazy), imperativní programovací jazyk PL/SQL rozšiřující možnosti vlastního SQL (v tomto jazyce je možné tvořit uložené procedury, uživatelské funkce, programové balíky a triggery), dále podporuje objektové databáze a databáze uložené v hierarchickém modelu dat (XML databáze, jazyk XSQL). Taktéž obsahuje širokou paletu nástrojů pro podporu snadného nasazení na gridových sítích pro podporu Grid Computing[ii]. [3]

Systém řízení báze dat MS SQL Server je natolik robustní a spolehlivý, že tvoří databázovou vrstvu aplikací podnikových informačních systémů strategického významu. Jeho nasazení jako databáze pro interaktivní internetové aplikace je také běžné.

MS SQL Server je dodáván s užitečnými aplikacemi s grafickým uživatelským rozhraním usnadňujícím vývojáři jeho práci a tím i život. V jádru je to samozřejmě klientská konzolová aplikace pro práci s databázemi pomocí SQL příkazu, jako u kteréhokoli jiného databázového serveru. Aplikace pro správu databází umožňují správcům všechny úkoly administrace serveru - správu rolí, uživatelů a jejich přístupových práv, pravidla pro zálohování a údržbu dat atd. Také pro MSSQL server není problémem zpracovat vnořené dotazy, triggery, uložené procedury, nebo transakce a podporuje T-SQL (konkurence PL/SQL). [4]

MySQL je databázový systém, vytvořený švédskou firmou MySQL AB. Pro svou snadnou implementovatelnost (lze jej instalovat na Linux, MS Windows, ale i další operační systémy), výkon a především díky tomu, že se jedná o volně šiřitelný software, má vysoký podíl na v současné době používaných databázích. Velmi oblíbená a často nasazovaná je kombinace MySQL, PHP a Apache jako základní software webového serveru.

MySQL bylo od počátku optimalizováno především na rychlost, a to i za cenu některých zjednodušení: má jen jednoduché způsoby zálohování, a až donedávna nepodporovalo pohledy, triggery, a uložené procedury. Tyto vlastnosti jsou doplňovány teprve v posledních letech, kdy začaly

---

i Denial of Service je technika útoku na internetové služby nebo stránky, při níž dochází k přehlcení požadavky a pádu nebo minimálně nefunkčnosti a nedostupnosti pro ostatní uživatele.

ii Grid v pravém slova smyslu je dnes definován jako infrastruktura umožňující sdílet kapacity a funkce, integrovat služby a prostředky v rámci organizací a mezi nimi, umožňující aktivní spolupráci v distribuovaném multiorganizačním prostředí.

nejčastějším uživatelům produktu – programátorům webových stránek – již poněkud scházet. Naštěstí i tato databáze už umí pracovat transakcemi, různými znakovými sadami, vnořenými dotazy, procedurami, triggery, s pohledy i s metadaty.

2.2 Architektura software

Softwarová architektura určuje strukturu systému, jeho části a vztahy mezi nimi. Způsob jakým bude aplikace napsána je velmi důležitý. Vhodný výběr usnadní vývoj, testování, údržbu, vytvoření dokumentace i znovupoužitelnost. Nad krátkými skripty se tvůrce ani nepozastavuje nad nějakou složitější architekturou. Maximálně se rozhodne pro programovací jazyk a ten už mu i trošku napoví, jestli použít strukturované, objektové či jiné metody. U složitějších aplikací je třeba se zamyslet. Protože tato práce má zjednodušit programátorům vytváření databázové webové aplikace, je velmi pravděpodobné, že takovou aplikací bude např. informační systém. Ten v sobě bude pracovat s daty o svých službách či produktech, zákaznících a třeba i vztahy mezi nimi. Výsledkem bude projekt, který bude rozsáhlý a vytváření i správa si vyžádá nemalou režii.

Osvědčený způsob jak řešit problémy je snaha rozdělit je na menší a věnovat se jim po částech. Podle toho se vytvořili i softwarové architektury. Existuje mnoho druhů jako např.:

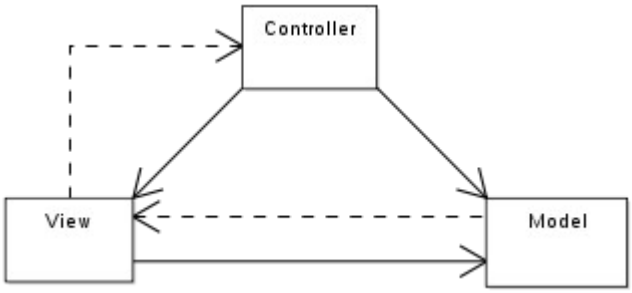
- Klient-server – vhodná pro systémy komunikující po síti.
- Datacentrická – využívá co nejvíce databázového systému a všech procedur a funkcí, které poskytuje.
- Distribuovaná – pro složité a náročné výpočty na více strojích.
- Orientovaná na služby (SOA) – umožňuje spojování aplikací napříč sítěmi prostřednictvím společného komunikačního protokolu.
- Architektura orientovaná na vyhledávání – která je důležitá pro databázové řídící systémy.

V této práci je třeba zauvažovat nad strukturou aplikace, která má být výsledkem tohoto projektu a zároveň strukturou automaticky generovaných webových aplikací, které potom budou výstupem vzniklého projektu. V poslední době se využívá především vícevrstvá architektura a Model-View-Controller, kterou se řídí např. vývojové prostředí NetBeans pro vytváření webových aplikací.

2.2.1 Model-View-Controller

Model-view-controller (MVC) je jak návrhovým vzorem tak i vzorem pro softwarové inženýrství. Izoluje obchodní logiku od uživatelského rozhraní a zároveň ovládá chování celé aplikace, podle toho, jak uživatel reaguje.

- Model – představuje data, se kterými aplikace pracuje a určuje, jak s nimi pracovat.
- View – koresponduje s grafickým uživatelským rozhraním, čímž jsou u internetové aplikace zpravidla webové stránky.
- Controller – řídí komunikaci mezi uživatelem a modelem. Má vlastně na starosti způsob chování aplikace podle akcí uživatele.



Obrázek 1: MVC architektura

MVC může být realizováno různým způsobem, funkce je ale stejná:

MVC funguje následovně:

1. Uživatel zareaguje (stiskne tlačítko, pohne myší, apod.).
2. Controller zachytí vstupní událost z rozhraní a zavolá obslužnou metodu a případně přistoupí k modelu, aby změnil jeho stav.
3. Model změní data podle svých pravidel.
4. Uživatelské rozhraní zobrazí změněná data popř. jiná, podle stavu aplikace.
5. Uživatelské rozhraní čeká na další akci.

MVC je také návrhový vzor, který je implementován v mnoha jazycích, takže i pro ASP.NET existuje např. ASP.NET MVC nebo ASP.NET Monorail. [5]

2.2.1 Třívrstvá architektura

Vícevrstvá architektura rozděluje systém na více samostatných vrstev (modulů), které jsou pak závislé jedna na druhé podle hierarchie. Nejčastěji používaná je třívrstvá architektura. Jedná se o architekturu, ve které jsou tři nezávislé části:

- grafické uživatelské rozhraní,
- obchodní logika,
- data s přístupovými metodami.

Tyto části mohou být vyvíjeny na různých platformách jinými specialisty, čímž lze dosáhnout vyššího výkonu a kvality.

Třívrstvá architektura je zároveň také softwarovým návrhovým vzorem.

Na rozdíl od klasického modulárního přístupu s dobře definovaným rozhraním tato architektura umožňuje upravit či nahradit jakoukoliv z vrstev nezávisle na požadavcích a technologických změnách. Například při změně struktury databáze není nutné upravovat vrstvu obchodní logiky nebo prezentační vrstvu. Pokud zůstalo zachováno rozhraní, výsledky veřejných metod jsou stejné. Vrstva uživatelského rozhraní běží na počítači uživatele, na kterém se zobrazují data patřičným způsobem. Obchodní logika a systém řízení báze dat s daty mohou být na úplně jiném místě třeba na vzdáleném serveru.

Co tedy poskytují jednotlivé vrstvy:

- **Prezentační vrstva** je nejsvrchnější v celé aplikaci. Zobrazuje informace, které se vztahují ke službám, které aplikace nabízí. Komunikuje s nižší aplikační vrstvou prostřednictvím voláním jejích metod.
- **Aplikační vrstva** tvoří prostředníka mezi vrstvou prezentační a vrstvou datovou. Obsahuje tzv. business logiku aplikace, která definuje, jak se má zacházet s daty. Tato vrstva je jakýsi mozek celé aplikace. Předává ve svých metodách data prezentační vrstvě a ukládá popř. načítá data voláním metod datové vrstvy.
- **Datová vrstva** obsahuje funkce pro přístup k informacím v datovém úložišti, na které se připojuje a ukládá do něj své informace.

V porovnání s MVC architekturou v aplikaci navržené jako třívrstvou architekturu nikdy nekomunikuje klientská část přímo s daty. To vždy zprostředkují prostřední vrstvy. Konceptuálně je třívrstvá architektura lineární oproti MVC, která je trojúhelníková, protože View zašle reakci uživatele Controlleru, ten aktivuje Model a View dostane data přímo od Modelu.

Použití třívrstvé architektury ve webových aplikacích je běžné, vytvoří se následující části:

1. Webové stránky zobrazující informace poskytované systémem.
2. Prostřední vrstva, která zajišťuje chování aplikace, obchodní logiku.
3. Databáze a operace pro přístup a manipulaci s daty.

Poslední vrstva se nedá vždy úplně rozdělit. V poslední době je snaha přejít na plně objektové programování a to i u přístupu k datům, která jsou často uložena v relačních databázích. Programátoři vytvářejí databázové objekty, které se shodují se záznamy v tabulkách a navíc poskytují rozšířené operace nad daty. V takovém případě nelze jednoduše upravit či vyměnit celou spodní vrstvu, ale lze např. přejít na jiný typ databáze, protože lze přistupovat k datům přes abstraktní datové objekty.

2.3 Návrhové vzory

Návrhový vzor (anglicky design pattern) představuje obecné řešení problému, které se využívá při návrhu programů. Návrhový vzor není knihovnou nebo částí zdrojového kódu, která by se dala přímo vložit do našeho programu. Jedná se o popis řešení problému nebo šablonu, která může být použita v různých situacích. Objektově orientované návrhové vzory typicky ukazují vztahy a interakce mezi

třídami a objekty, aniž by určovaly implementaci konkrétní třídy. Algoritmy nejsou považovány za návrhové vzory, protože řeší konkrétní problémy a nikoliv problémy návrhu.

Návrhové vzory nepocházejí ze softwarového inženýrství – jsou zcela běžné v každodenním životě. K asi nejznámějším a nejstarším příkladům patří architektura. Gotickou katedrálu poznáte už zdaleka právě proto, že tehdejší architekti a jejich stavební hutě používali stejných návrhových vzorů.

Návrhové vzory v softwarovém inženýrství se dají rozdělit do tří skupin:

- Creational Patterns (vytvářející).
- Structural Patterns (strukturální).
- Behavioral Patterns (chování).

V následující tabulce můžete vidět přehled návrhových vzorů podle kategorií a v dalších podkapitolách jsou popsány ty nejpoužívanější. [6]

Creational Patterns	Structural Patterns	Behavioral Patterns
Factory Method Pattern	Adapter Pattern	Chain Of Responsibility Pattern
Abstract Factory Method Pattern	Bridge Pattern	Command Pattern
Builder Pattern	Composite Pattern	Interpreter Pattern
Prototype Pattern	Decorator Pattern	Iterator Pattern
Singleton Pattern	Facade Pattern	Mediator Pattern
	Flyweight Pattern	Memento Pattern
	Proxy Pattern	Observer Pattern
		State Pattern
		Strategy Pattern
		Template Pattern
		Visitor Pattern

Tabulka 1: Návrhové vzory

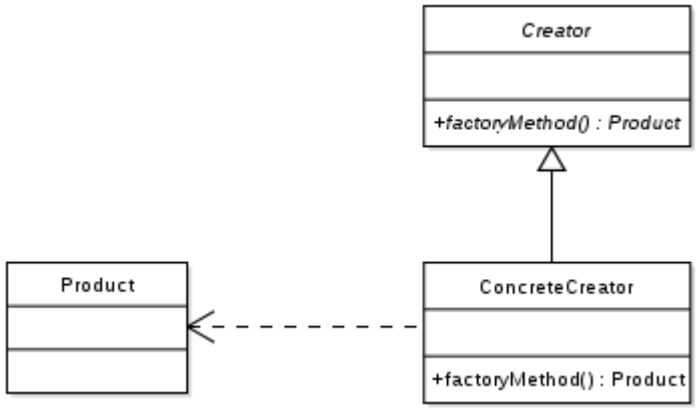
2.3.1 Vytvářecí návrhové vzory

Vytvářecí návrhové vzory (Creational Patterns) řeší problémy související s vytvářením objektů v systému. Snahou těchto návrhových vzorů je popsat postup výběru třídy nového objektu a zajištění správného počtu těchto objektů. Většinou se jedná o dynamická rozhodnutí učiněná za běhu programu.

Factory Pattern řeší otázku, jakým způsobem rozhodnout v průběhu programu o vytvoření instance konkrétní třídy. Předpokladem je existence několika tříd, které mají obvykle společného předka, ale poskytují různé služby nad různými daty. Factory method pattern potom dovoluje vybrat v průběhu programu vytvoření instance některé z těchto tříd.

Je nutné rozhodnout o principu výběru třídy, na jejímž základě bude objekt vytvořen. Je definován objekt, který se stará o způsob vytváření instancí podřízených tříd (implementuje factory metodu vytvářející produkty). Třída vytvářejícího objektu je definována jako abstraktní nebo jako konkrétní (přímo implementující factory method).

Máme obecnou třídu Product, která představuje rozhraní implementované třídami Concrete Creator. Na místě, kde se vzor používá je pouze reference na objekt Creator. Od něho se vyžádá instance jedné třídy typu ConcreteCreator, ale přímo tato třída se neurčuje. Creator na základě věcné logiky a předaných parametrů rozhodne, která třída bude vybrána pro vytvoření instance. Tuto instanci potom vrací jako výsledek, a v místě, kde se vzor používá se nemusí implementovat logika výběru třídy a v dané chvíli se ani neví, jakou instanci ConcreteCreator získal.



Obrázek 2: Factory Pattern

Tento návrhový vzor lze použít např. u aplikace, kde obecně chceme připojení k databázi, ale předem nevíme, o jaký typ databázového serveru se bude jednat.

Abstract Factory Pattern řeší problém, jak vytvořit na základě rozhodnutí v běhu programu instanci třídy, která dále vytváří instanci souvisejících nebo závislých tříd.

Abstract Factory představuje o něco vyšší stupeň abstrakce než předchozí Factory Method Patttern. Problém je, že v určitých případech máme více tříd, které vytvářejí instance konkrétních produktů. Každá takové třída vytváří různé typy produktů, jež spolu ovšem nějakým způsobem souvisí. Například odpovídají stejnému uživatelskému rozhraní. Návrhový vzor řeší jak umožnit vybrat za běhu programu jednu z těchto „vytvářejících“ tříd, aniž bychom museli dělat úpravy v celém systému.

Je nutné vytvořit abstraktní třídu pro konkrétní produkty obdobně jako v předešlém návrhovém vzoru. Místo abstraktní třídy lze využít i rozhraní. Navíc je nutné určit další rozhraní, které bude implementováno konkrétními třídami vytvářející jednotlivé produkty.

Prototype Pattern řeší otázku, jak vytvořit kopii existujícího objektu místo vytváření nové instance dané třídy. Používá se, aby se zamezilo zbytečnému vytváření podtříd v aplikaci a nebo aby

se zbytečně nevytvářela nová instance klasickou cestou pomocí klíčového slova *new*, které je časově i výkonnostně náročnější.

Signleton je návrhový vzor, který se používá v aplikacích, kde požadujeme pouze jednu instanci konkrétní třídy a používáme ji, jako by to byla globální proměnná. Třída, která implementuje tento návrhový vzor, si uchovává informaci, jestli už byla instance vytvořena a pokud je potřeba vrací právě tuto instanci. V jazyce C# je možné implementovat třídu způsobem, jaký můžete vidět na následujícím obrázku.

```
using System;
public class Singleton
{
  private static Singleton
instance;

  private Singleton() {}

  public static Singleton
Instance
  {
    get
    {
      if (instance == null)
      {
        instance = new
Singleton();
      }
      return instance;
    }
  }
}
```

Obrázek 3: Implementace návrhového vzoru Singleton v jazyce C#

2.3.2 Strukturální návrhové vzory

Strukturální návrhové vzory (Structural Patterns) představují skupinu návrhových vzorů zaměřujících se na možnosti uspořádání jednotlivých tříd nebo komponent v systému. Snahou je zpřehlednit systém a využít možností strukturalizace kódu.

Adapter Pattern – jak již název tohoto návrhového vzoru napovídá, jedná se o přizpůsobení určité třídy, aby ji bylo možné využívat i jiným požadovaným způsobem. Problémem je zajištění konverze rozhraní jedné třídy na rozhraní jiné třídy.

Základní situací, s kterou se setkáváme při přechodu na nové verze systémů, je zajištění jejich zpětné kompatibility. Například se rozhodneme využívat nové třídy, které poskytuje knihovna Swing a existuje mnoho stávajících klientů, kteří očekávají komunikaci podle standardů knihovny AWT (starší verze grafické knihovny). Abychom zajistili zpětnou kompatibilitu a zároveň mohli využívat

nové technologie, musíme vytvořit mechanismus zpracování požadavků od stávajících klientů naší aplikace. Tento problém lze řešit právě využitím tohoto návrhového vzoru.

Na obrázku (Obrázek 4) je třída Adapter, která zajišťuje funkcionalitu systému, ale neimplementuje požadované rozhraní. Rozhraní, které očekává třída Client, je definované v abstraktní třídě Target. Tato třída slouží jako předek pro konkrétní třídu Adapter, která má za úkol transformovat volání od klienta na příslušné metody tříd typu Adaptee.

```
using System;

class MainApp
{
  static void Main()
  {
    Target target = new Adapter();
    target.Request();
  }
}

class Target
{
  public virtual void Request()
  {
    Console.WriteLine("Called Target
Request()");
  }
}

class Adapter : Target
{
  private Adaptee _adaptee = new Adaptee();

  public override void Request()
  {
    _adaptee.SpecificRequest();
  }
}

class Adaptee
{
  public void SpecificRequest()
  {
    Console.WriteLine("Called SpecRequest()");
  }
}
```

Obrázek 4: Implementace návrhového vzoru Adapter v jazyce C#

Existují dvě možná řešení tohoto návrhového vzoru v jazyce Java. První je založen na odvození nové třídy od staré nevyhovující a přidání metod, které zajistí implementaci požadovaného rozhraní. Toto řešení je nazýváno třídní (class) adapter a využívá dědičnosti. Naproti tomu objektový (object) adapter využívá principu vložení reference na třídy Adaptee do Adapteru a delegování požadavků na

metody Adaptee objektu. V takovém případě může jeden Adapter zajišťovat komunikaci s více objekty.

Na jednoduchém příkladě s ovocem a zeleninou lze znázornit použití tohoto návrhového vzoru.

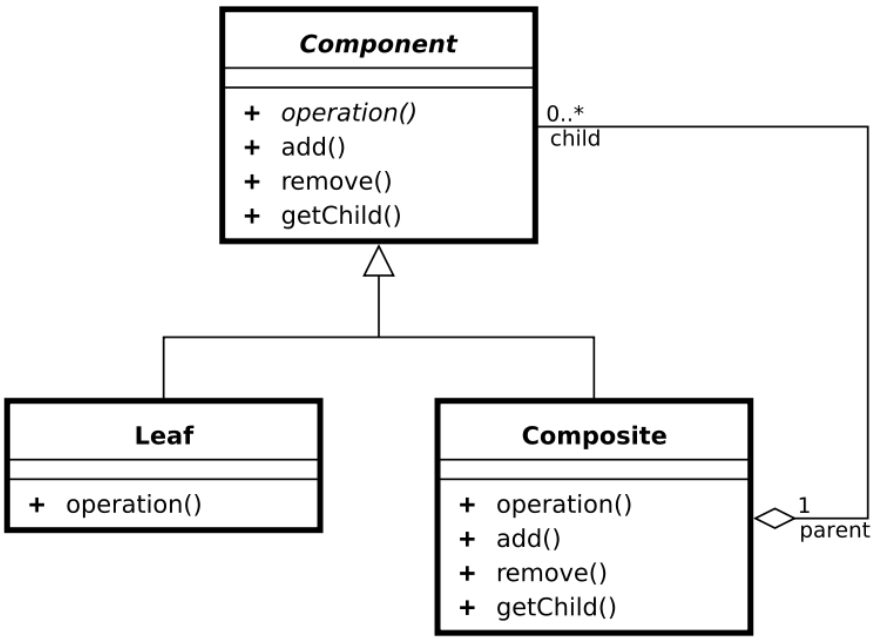
Maloobchod je zvyklý přijímat informace o stavu množství zásob ve skladech velkoobchodu. Ten mění logistickou strategii a přesouvá skladování ovoce na jednotlivé pěstitele. Z tohoto důvodu získává přehled o možných zásobách jednotlivých druhů ovoce a zeleniny přímo od pěstitele. Maloobchody by se mohly dotazovat také přímo pěstitelů, ale musely by se přizpůsobit jejich rozhraní. Druhou možností je využití rozhraní velkoobchodů. Velkoobchody potom slouží jako třídy Adapter, protože delegují požadavky od maloobchodů na jednotlivé pěstitele. Jedná se o objektový adaptér, protože si lze lehce představit situaci, kdy jeden obchod deleguje požadavek na více pěstitelů.

Výsledkem je, že není nutné měnit klientskou část aplikace, která využívá adaptovanou třídu. Stávající klienti budou využívat existující rozhraní představované třídou Adapter. Noví klienti rozhraní, které je implementováno ve třídě Adaptee.

Dalším používaným vzorem je **Composite Pattern**. Tento vzor představuje řešení, jak uspořádat jednoduché objekty a z nich složené (kompozitní) objekty. Snahou vzoru je, aby k oběma typům objektů (jednoduchým a složeným) bylo možné přistoupit jednotným způsobem.

Před dalším vysvětlením tohoto návrhového vzoru je nutné vymezit, co znamená pojem složený (v Design Pattern terminologii Composite) objekt. Tento objekt obsahuje kolekci jiných objektů, z nich každý může být buď jednoduchý objekt, nebo opět složený objekt. Jednoduchý objekt neobsahuje referenci na žádné jiné objekty.

Použitím vzoru Composite lze uspořádat jednoduché objekty a kompozitní objekty do stromové hierarchické struktury. Ta má jeden výchozí uzel (kořen), který obsahuje reference na potomky tj. jednoduché objekty (listy) nebo složené objekty (větve). Každá větev se může dále větvit, nebo obsahuje již jednoduché koncové objekty. Composite pattern řeší situaci, jak uspořádat tuto strukturu, aby bylo možné k větvím i listům přistupovat jednotným způsobem.



Obrázek 5: Diagram tříd návrhového vzoru Composite

Obecně tento návrhový vzor předpokládá, že existuje jednotné rozhraní pro jednoduché objekty i kompozitní objekty, které je v diagramu představováno třídou Component. Tato třída definuje metody pro manipulaci s potomky (add, remove, getChild) a také věcné metody (operation). Základní výhodou tohoto vzoru je, že klient se nemusí starat, jestli získaný objekt představuje větev (Composite) nebo list (Leaf), protože jsou děděny ze stejné třídy. Jestliže klientský objekt zavolá metodu Operation na listovém objektu, je přímo vykonána. U větve, ve většině případů, je tato metoda vyvolána u všech potomků a mohou být provedeny další operace, jako je například sečtení získaných výsledků a použití získaného součtu jako návratové hodnoty.

Někteří autoři doporučují mít dvě rozdílná rozhraní pro větve a pro listy, čímž je vyřešen problém, jestli pracujeme s koncovým objektem nebo se složeným.

2.3.3 Návrhové vzory chování

Návrhové vzory se zaměřením na chování (Behavioral Patterns) mohou být založeny na třídách nebo objektech. U tříd využívají při návrhu řešení především principu dědičnosti. V druhém přístupu je řešena spolupráce mezi objekty a skupinami objektů, která zajišťuje dosažení požadovaného výsledku.

Iterator Pattern řeší problém, jak se pohybovat mezi prvky, které jsou sekvenčně uspořádány, bez znalosti implementace jednotlivých prvků posloupnosti. Iterator je jeden z nejjednodušších, ale nejvíce používaných návrhových vzorů. Vysvětluje možnost způsob sekvenční pohybu v seznamu objektů. Pro tento pohyb není nutné znát detailní implementaci dat v seznamu, jelikož jeho průchod není zajišťován přímo tímto seznamem.

Iterator je rozhraní, které definuje metody pro průchod seznamem. Mezi metody může patřit přechod na první záznam, na další záznam v sekvenci, kontrola, jestli existuje další záznam nebo předání odkazu na aktivní záznam. Protože toto rozhraní definujeme sami, je možné použít i více sofistikované metody. Rozhraní Iterator je implementováno třídou, např. Concrete Iterator, která zajišťuje průchodnost kolekcí. Při vytváření Concrete Iterator je předán odkaz na seznam, jímž se bude procházet. Klient pak vytváří pouze tento seznam a naplní ho daty. Potom požádá o vytvoření objektu Concrete Iterator, který mu umožní procházet seznam a získávat jednotlivé položky.

Observer Pattern se zabývá otázkou definování závislosti jednoho objektu k více objektům. Umožnění šíření události, která nastala v jednom objektu, ke všem závislým objektům.

Observer je možné použít v situaci, kdy je definována závislost jednoho objektu na druhém. Závislost, ve smyslu tohoto návrhového vzoru, představuje propagaci změny nezávislého objektu závislým objektům (pozorovatelům). Nezávislý objekt musí informovat závislé objekty o událostech, které je mohou ovlivnit. Je nutné zajistit existenci způsobu dynamické modifikace seznamu podřízených objektů. Snahou je oddělit nezávislý nadřízený objekt od logiky informování závislých objektů, aby tyto mohly být informovány bez znalosti jejich vnitřní struktury.

```
public class ObservableImpl:IObservable {
   protected Hashtable _observerContainer=new Hashtable();
   public void Register(IObserver anObserver){
      _observerContainer.Add(anObserver,anObserver);
   }//Register

   //remove the observer
   public void UnRegister(IObserver anObserver){
      _observerContainer.Remove(anObserver);
   }//UnRegister

   public void NotifyObservers(object anObject) {
      foreach(IObserver anObserver in
_observerContainer.Keys) {
        anObserver.Notify(anObject);
      }//foreach
   }//NotifyObservers

}//ObservableImpl
```

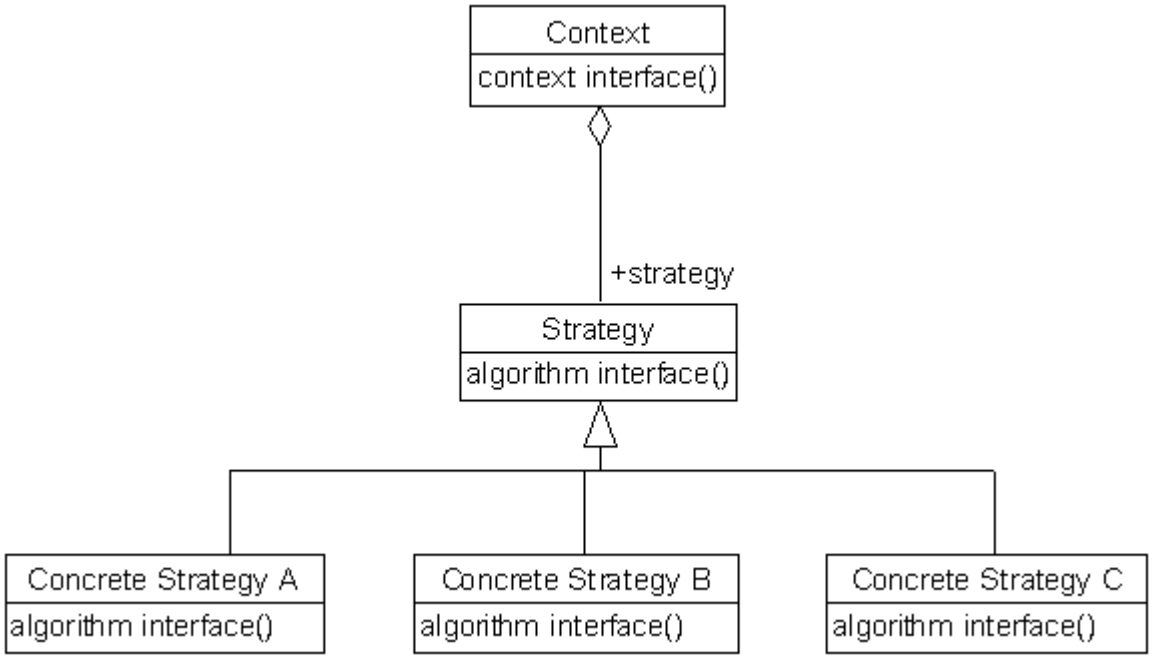
Obrázek 6: Implementace návrh. vzoru Observer v jazyce C#

Strategy Pattern řeší problém tam, kde potřebujeme určit rodinu algoritmů, zapouzdřit každý z nich do samostatného objektu a umožnit jejich záměnu. Řeší problém změny algoritmu nezávisle na klientovi, který jej využívá. Základní podmínkou návrhového vzoru je předpoklad, že existuje více podobných řešení stejného problému. Tento návrhový vzor uvažuje o skupině algoritmů, které řeší stejnou funkčnost rozdílným způsobem. Jestliže budou tyto algoritmy implementovány jako

monolitický celek plný podmínek, snižuje se možnost jejich opětovného využití a dynamické změny algoritmu dle zvolené strategie.

Context je objekt, s kterým komunikuje klient. Uchovává referenci na určitý Concrete Strategy objekt. Výběr správného objektu z množiny možných Concrete Strategy objektů, kterým má být zpracován požadavek od klienta, je možný buď na straně klienta, nebo tento výběr provádí Context. Při první možnosti, která je obvyklejší, je Concrete Strategy předáván Contextu jako parametr. Tím klient sám určuje strategii, jež bude uplatněna. V některých případech může rozhodnout i Context o nejlepší volbě strategie pro klienta.

Strategy představuje společné rozhraní, které musí být implementováno všemi použitými algoritmy. Context využívá tohoto rozhraní pro volání algoritmu definovaného v Concrete Strategy objektech, které představují rozdílnou implementaci stejného problému.



Obrázek 7: Diagram tříd návrhového vzoru Strategy

Obvyklý model chování tohoto návrhového vzoru, začíná určením strategie. Klient předá Contextu v parametru zvolený objekt Concrete Strategy. Klient komunikuje pouze s Contextem, který zprostředkovává volání mezi klientem a zvolenou strategií. Context může předat Concrete Strategy všechna data, která jsou nutná k výpočtu pomocí algoritmu. Druhou variantou je předání Concrete Strategy odkazu na Context. V tomto případě je vytvořeno ještě jedno rozhraní Contextu, které využívají Concrete Strategy objekty pro získání potřebných dat.

2.4 UML

UML, Unified Modeling Language je v softwarovém inženýrství grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů. UML nabízí standardní způsob zápisu jak návrhů systému včetně konceptuálních prvků jako jsou obchodní procesy a systémové funkce, tak konkrétních prvků jako jsou příkazy programovacího jazyka, databázová schémata a znovupoužitelné programové komponenty.

UML podporuje objektově orientovaný přístup k analýze, návrhu a popisu programových systémů. UML neobsahuje způsob, jak se má používat, ani neobsahuje metodiky, jak analyzovat, specifikovat či navrhovat programové systémy.

Podrobná dokumentace a specifikace je v dnešní době nedílnou součástí vývoje každého většího projektu. Součástí vypracování této dokumentace a specifikace je sběr dat a požadavků na systém, což vede k lepšímu a hlavně správnému pochopení zadání a účelu, ke kterému bude daný systém použit. Modelovacích technik UML lze využít i při analýzách systémů bez dostupné dokumentace, nebo systémů které dokumentaci nemají vůbec. Tato analýza slouží k lepšímu pochopení jejich činnosti a také k nalezení a specifikaci jejich výhod a nedostatků. Ve vypracovaném modelu pak lze nalézt možnosti optimalizace, rozšíření, způsoby odstranění chyb, nebo se dá modelovaný systém srovnávat s podobnými systémy.

Vytvořené modely systémů mohou být využity jednak při jejich implementaci, pří řízení projektů, nebo při práci ve skupině, ale i při jejich prezentaci konečnému zákazníkovi.

Jazyk UML rozeznává pět základních pohledů na systém.

1. Pohled případů užití, které vyjadřují základní požadavky kladené na systém. Veškeré další pohledy se pohybují v mantinelech vymezených pohledem případů užití.
2. Logický pohled se zabývá zejména pojmy z problémové domény zadavatele a jejich vzájemnými statickými vztahy.
3. Procesní pohled se soustřeďuje na chování systému, které musí splňovat požadavky a omezení z případů užití, jež jsou kladeny na průběh procesů. Jedná se o procesně orientovaný doplněk logického pohledu.
4. Implementační pohled zachycuje fyzické rozdělení aplikace na samostatné komponenty a jejich závislosti.
5. Pohled nasazení mapuje komponenty na fyzické zařízení v cílovém prostředí.

Ke svému znázornění používá prvky, vztahy a diagramy. Prvky jsou určitým typem abstrakce:

- strukturní (třída, případ použití, komponenta),
- chování (interakce, stav),
- seskupování (modul, balíček, podsystém).

Vztahy, které existují mezi prvky:

- závislost,
- asociace,
- generalizace,
- realizace.

Diagramy pak zachycují prvky a jejich vztahy. V UML existují diagramy tříd, objektů, případů použití, interakce (sekvence a spolupráce), aktivit, komponent, nasazení a stavový diagram.

V této práci je použity především diagram případů použití a diagram tříd. V kapitole Demo aplikace, která je součástí testování můžete najít popis databáze v entitě-relačním diagramu, který ale není součástí UML 2.0.

2.5 Podobné existující aplikace

Ve vyhledávačích najdete spoustu generátorů stránek ať už pro PHP nebo ASP.NET. PHP se přímo nepojí s určitým vývojovým prostředí, ale při práci s ASP.NET je velkým usnadněním Microsoft Visual Studio (dále jen Visual Studio). Více se o něm dozvíte v analýze, která se zabývá strukturou jeho projektů. S Visual Studiem se ovšem většina existujících generovaných aplikací nepřátelí stoprocentně. Projekty jdou samozřejmě vložit a po určitých nastaveních spustit, ale jejich úprava, kterou je nutno udělat ve většině případů, už není tak jednoduchá. ASP stránky pak často postrádají tzv. *designer* soubory, které používá grafický editor stránek. Navíc není oddělená datová, logická a prezentační část, což není vhodné pro větší projekty. Také jsou tyto aplikace většinou desktopové, je třeba je nainstalovat a nelze je použít bez instalace, ale pokud jsou jejich výsledky kvalitní a vývojář hodlá program používat, pak je to v pořádku.

Světlou výjimkou je Free Rod (*http://www.rodtherobot.com/FreeRod.asp*) generátor, který je bohužel jenom v beta verzi a dokáže zpracovat pouze jednu tabulku. Navíc vygeneruje pouze ASP soubory, čímž spojí logiku, zobrazení i datové operace dohromady.

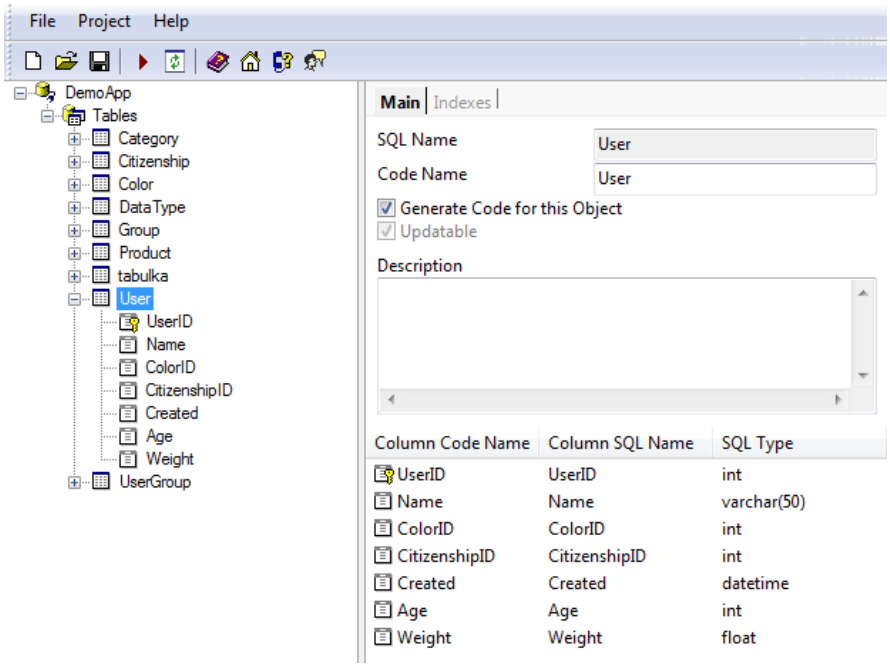
ASPRunner.NET desktopová aplikace, podporuje řadu databází (MSSQL, MySQL, Oracle, MS Access), umožňuje tvorbu struktury stránek Master-Detail, umožňuje vyhledávání, vícejazyčnost, zabezpečený přístup pomocí přihlášení na stránky, stylování, definování událostí u komponent, které se nachází na stránce. Jedná se o velice propracovaný nástroj, který ovšem není zadarmo. Chybí mu ale designer soubory pro grafickou editaci webových formulářů ve Visual Studiu. Používá pro větší textové vstupy FCKeditor, což je WYSIWYG editor.

2.5.1 RapTier

RapTier je aplikace pro operační systém Windows. Je zdarma pro použití na maximálně 15 tabulek. Podle internetových stránek soudím, že se jedná jedinou aplikaci tohoto druhu od společnosti

SharpPower Corp. Zde také uvádějí, že jejich generátor vytváří aplikaci ve dvou vrstvách a to datové a prezentační.

První dialogové okno, které se objeví při založení nové projektu, zobrazí přehled všech dostupných ovladačů pro připojení k různým typům databáze. Ostatní testované aplikace nabízí pouze typ nikoliv konkrétní ovladač. Po připojení lze definovat programovací jazyk (C#, VisualBasic), vývojové prostředí (poslední nabízené je Visual Studio 2003) a název projektu a jmenných prostorů. Výjimečný je tento program tím, že nabízí generování nejen ASP.NET stránek, ale i vytvoření aplikace pro Windows. Další nastavení umožňuje zvolit jména sloupců a tabulek v kódu a jejich názvy, které uvidí uživatelé na stránkách (Obrázek 8). To je v podstatě všechno, co daná aplikace nabízí. Při vygenerování se pokouší vytvořit virtuální adresář na webovém serveru.



Obrázek 8: RapTier – nastavení datového objektu

Výsledkem je Solution, které obsahuje 2-3 projekty. Jeden znázorňuje datovou vrstvu, druhý prezentační a logickou pro webovou aplikaci a třetí prezentační a logickou pro aplikaci Windows. Datová vrstva vůbec neřeší, jestli mohou být hodnoty NULL nebo nikoliv, pouze ukládá data tak, jak jsou jí předložena. Pro aplikační logiku jsou pravděpodobně určeny soubory s behind kódem od ASP stránek. Editace veškerých dat je možná pouze textovými poli a to jenom přímo v tabulce, neexistuje např. vkládání na nové stránce. Aplikace neřeší přihlašování ani oddělení veřejně dostupných stránek a části pro správu webu. Bohužel jsem nepřišel na to, jak se pracuje s cizími klíči. Nikde nebyla ani zmínka o tom, že databáze nějaké má nebo že by se provázání tabulek dalo vytvořit manuálně. Na webových stránkách produktu (*http://www.sharppower.com/*) je ale zmínka, že program generuje i dokumentaci k databázi, kde jsou zachyceny i vazby mezi tabulkami. Nevím proč tedy nejsou tyto informace využity. Vygenerovaná aplikace obsahuje soubory se zdroji dat, ve kterých jsou uloženy jazykové proměnné, jako jsou názvy tabulek a sloupců, ale tento soubor je prázdný.

Vytvořená část s Windows aplikací se dá použít v grafickém editoru Visual Studia, ale u webové aplikace se mi nepodařilo otevřít projektu ve Visual Studio 2008, takže jsem to nemohl otestovat.

2.5.2 ASP.NET Maker

V době kdy jsem testoval ASP.NET Maker byl ve verzi 3. V čase dokončování této práce již ve verzi 7. Vývojáři tohoto softwaru zřejmě postupují vesmírnou rychlostí, protože postup o 4 verze během 6 měsíců je až neuvěřitelný. Nicméně už v mnou testované verzi byl tento program asi jako jeden z nejlepších na trhu. ASP.NET Maker pochází od firmy, jejíž skutečný název jsem na stránkách tohoto produktu nenašel. Nicméně má tato společnost ve svém sortimentu také generátory nejen pro ASP.NET, ale i pro ASP a PHP. Ke všem technologiím nabízí minimálně tři různé technologie:

- **Report Maker** – nabízí přehledy dat z tabulek i jejich detaily. Tato aplikace umí exportovat do HTML/Excel/Word formátů, ale také generovat grafy (sloupcové, koláčové, čárové), které jsou velmi používané především při prezentaci statistických údajů. Tyto přehledy mají jednu velkou nevýhodu, nelze v nich totiž vyhledávat.
- **XMLMaker** – tento program vytvoří ASP.NET stránky, které produkují XML výstup. Ten lze použít v dalších aplikacích založených např. na Ajax, RSS nebo Flash, které umí s XML pracovat. Uživatel si může definovat formát XML, aby co nejvíce vyhovoval ostatním programům. Tento nástroj je vhodný pro začátečníky i pokročilé uživatele, kteří potřebují podrobnější natavení.
- **Maker** – tento nástroj je nejvíce podobný tomu, co by mělo být v budoucnu rozšířeným výsledkem tohoto projektu. Jedná se o generátor aplikace, která dokáže data nejen zobrazovat jako již zmíněný Report Maker, ale také vkládat, mazat, editovat a také v nich vyhledávat.

Všechny tyto aplikace mají určité společné funkce. Lze v nich data třídit a řadit, vytvářet vlastní pohledy. Uživatel může použít svoje šablony pro generování XML nebo ASP/HTML. Obrovskou výhodou je synchronizace (aktualizace) s databází. Otázkou ale zůstává, jak se program zachová, pokud si uživatel něco upraví ve vygenerovaných souborech. Pro technologii ASP.NET dokáže produkovat zdrojové kódy pro jazyk C# i VisualBasic. Poskytuje připojení k databázi Microsoft Access nebo k jakékoliv jinému datovému zdroji, který podporuje ADO.NET jakými jsou Microsoft SQL Server, MySQL Server, Oracle apod.

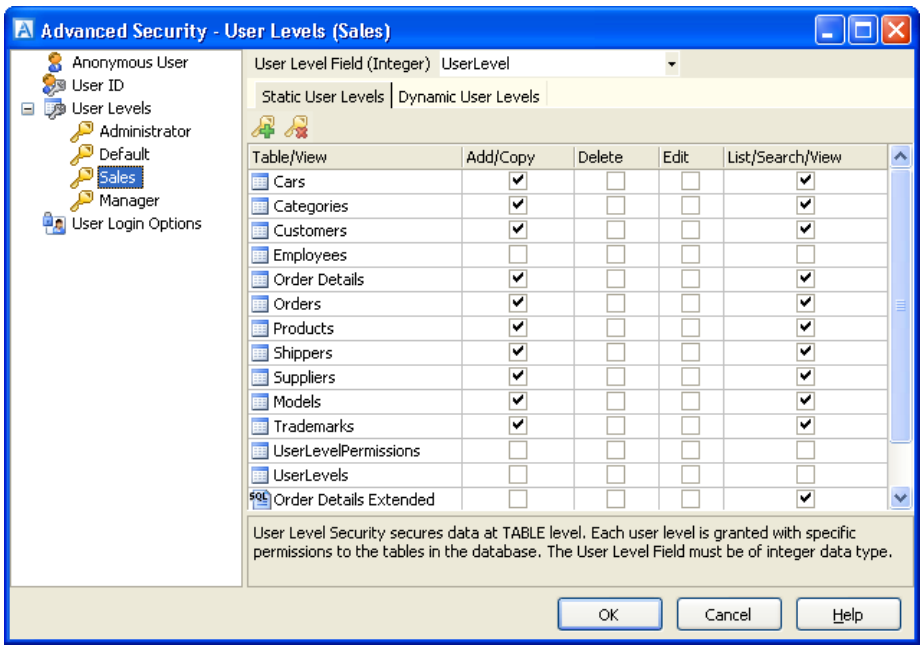
K hlavním charakteristikám programu ASP.NET Maker patří:

- všechny odkazy se odkazují na existující stránky,
- možnost zapnutí nebo vypnutí zobrazení přehledů dat, detailů, editačních stránek, přidávání, mazání a vyhledávání,
- volba přesměrování po provedených akcích (smazání, vložení, změna),

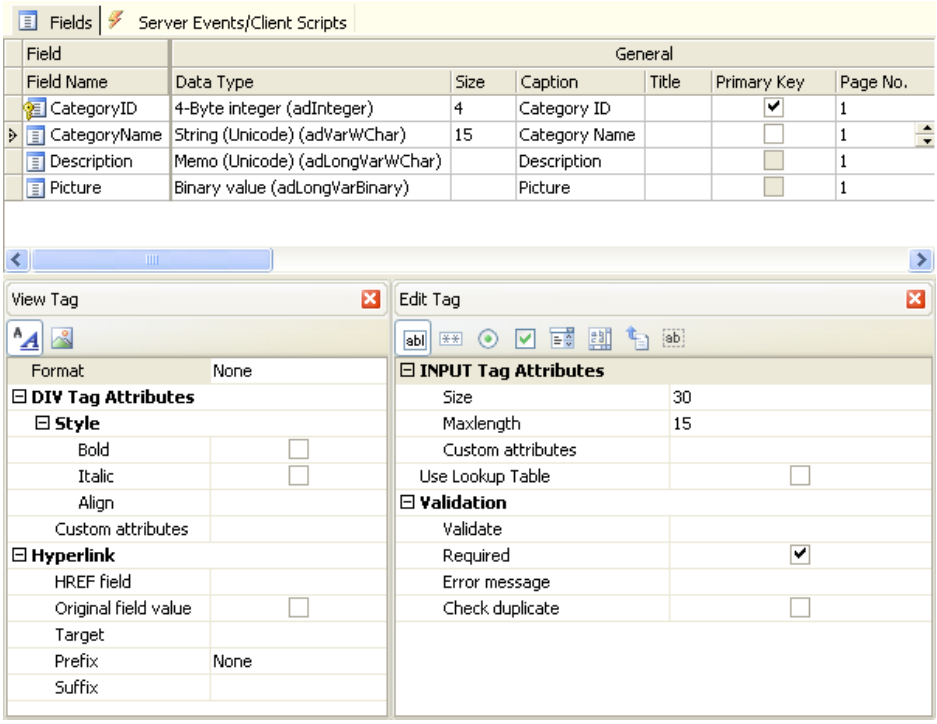
- editace přímo v tabulce nebo na zvláštní stránce,
- volitelné upozornění před vkládáním a změnou záznamů,
- stránkování s nastavením počtu záznamů u přehledů i v detailech,
- nastavení formátu zobrazení a editace pro každý sloupec v tabulce,
- kontrola vstupů na straně serveru nebo pomocí javascriptu na straně klienta,
- volby pro vyhledávání (rychlé, rozšířené) se zvýrazněním vyhledaných výsledků,
- zabezpečený přístup ke stránkám (viz níže) a registrace nových uživatelů,
- nastavení HTML, znakové sady, písma, kaskádových stylů a témat náhledů,
- editor menu,
- automatické vytvoření virtuálního adresáře v IIS serveru,
- použití agregačních funkcí (součty, průměry a počty záznamů),
- vlastní pohledy tvořené v interním editoru SQL dotazů,
- export do tisknutelné podoby,
- nahrávání souborů do složky na serveru nebo databáze,
- dynamické načítání tabulek až v okamžiku, kdy jsou potřeba,
- podpora složený klíčů v databázi,
- našeptávač textových polí,
- zaznamenávání změn, zasílání emailů o změnách,
- CAPTCHA[i] systém.

Zabezpečení přístupu k datům si zaslouží kratičký rozbor. V systému, který je vygenerovaný pomocí ASP.NET Maker je možné nastavit heslo administrátora, který má přístup ke všemu. Toto jméno a heslo je uvedeno přímo v kódu aplikace. Lze ale také využít existující tabulku uživatelů, ve které je uloženo jméno a heslo. To je tzv. statický způsob zabezpečení. Za dynamický je považovaný takový, kde se dají nastavit úrovně zabezpečení (levels) a podle nich povolit uživatelům přístup k určitým stránkám a přidělit oprávnění akcí, které mohou uživatelé provádět.

i CAPTCHA je Turingův test, který se na webu používá ve snaze automaticky odlišit skutečné uživatele od robotů. Je to akronym pro „ automated public Turing test to tell computers and humans apart“



Obrázek 9: ASP.NET Maker – Statické nastavení přístupu



Obrázek 10: ASP.NET Maker 3 – nastavení sloupců

Pokud uživatel programu ASP.NET Maker nepotřebuje žádné speciální nastavení, je použití aplikace poměrně jednoduché a intuitivní. Při pokročilejším nastavení a provázání tabulek je potřeba si s programem pohrát trošku déle, protože poskytuje skutečně mnoho voleb a ne všechny jsou jasné na první pohled. Aplikace ale dokáže ukládat rozpracovanou práci a případně ji znovu otevřít, aktualizovat podle upravené struktury databáze a změnit různá nastavení.

Ve srovnání se generátorem, který má být výsledkem tohoto projektu, je ASP.NET Maker klientskou Windows aplikací, takže je třeba ji instalovat na systém Windows. Tato aplikace

nerozpozná cizí klíče a ani podle nich nevytvoří žádné vazby, vše je třeba udělat ručně. Alespoň jsou k tomu k dispozici grafické nástroje. Výsledkem jsou pouze soubory, podle jejichž pojmenování jsem usoudil, že je tam snaha o jakési oddělení jednotlivých vrstev podle pojmenování některých souborů tříd, ale proč si autoři nedali alespoň trochu práce s rozložením do jmenných prostorů. Ani samotné pojmenování není na první pohled viditelné. Soubor i třída mají jména např. Tabulkabll a Tabulkadll, z čehož jsem usoudil, že se bude jednat o business logiku a datovou vrstvu. Pokud bych si ale nepřečetl několik tutoriálů, ve kterých je označení podobné, ale zvýrazněné alespoň velkými písmeny, nepřišel bych na to. Vygenerovaný kód navíc obsahuje textové řetězce přímo v ASP stránkách, takže u vícejazyčných rozsáhlejších portálů je výsledek této aplikace ne zcela použitelný. Vyprodukované soubory nejsou určené pro žádné vývojové rozhraní, takže neobsahují soubory projektu pro Microsoft Visual Studio a tím pádem neumožňují využití jeho design módu pro další vývoj aplikace.

Celkově na mne program poměrně zapůsobil a byl bych rád, kdyby se tento projekt ve velké míře tímto softwarem inspiroval. Program je zdarma pouze ve zkušební verzi. Více se o této aplikaci můžete dozvědět na internetových stránkách *http://www.hkvstore.com/aspnetmaker/*.

2.5.1 Three Tier Code Generator For ASP.NET

Tento generátor není tak rozsáhlý jako ASP.NET Maker a nemá takové zázemí, ale je k dispozici zdarma včetně zdrojových kódů. Sám o sobě je totiž také ASP.NET aplikací napsanou v jazyce VisualBasic, kterou je nutno spustit na webovém serveru. S tím přichází hned první svízel při jeho použití. Při stažení zdrojových kódů se mi nepodařilo aplikaci spustit, takže následující analýza vychází z demonstračního projektu, který je k dispozici na internetových stránkách této aplikace.

Demonstrační projekt obsahuje dvě části. Jednou je implementace v jazyce C# a v druhé jsou zdrojové kódy ve VisualBasicu. I když obsahuje tento projekt soubor s příponou SLN, nepodařilo se mi jej otevřít ve Visual Studiu. Přesto jsem vytvořil analýzu toho, co se píše na stránkách a co je obsaženo ve demo aplikaci.

Jak již sám název napovídá, výstupem toho softwaru je ASP.NET aplikace, jejíž kód je navržený pomocí třívrstvé architektury.

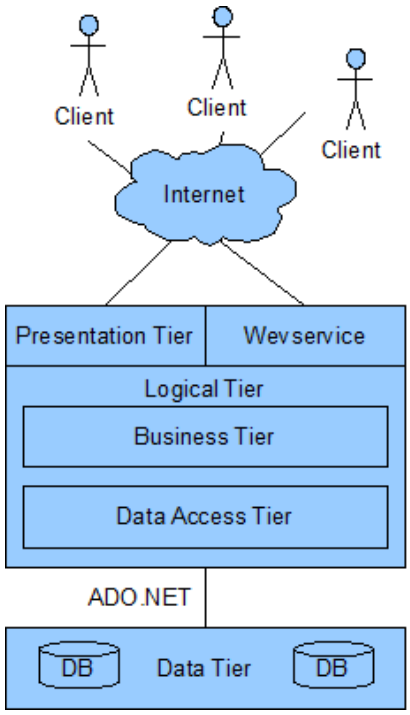
Mezí základní charakteristiky tohoto software patří:

- generování veškerého kódu ve třívrstvé architektuře pro Visual Studio,
- vytvoření šablonových a chybových vzorů,
- generování SQL procedur a skriptů na vytvoření, přidání, aktualizaci a prohlížení záznamů tabulek,
- generování kódů pro C# i VB.NET,
- editace záznamů přímo v tabulce nebo na samostatných stránkách,
- umožňuje použití kaskádových stylů,

- využívá Microsoft Data Access Application Block, který zjedodušuje práci s volanými procedurami a textovými SQL dotazy a také určuje parametry detailu záznamů.

Autor na stránkách hovoří o třech vrstvách, ale popisuje jich pět (Obrázek 11).

- Datová vrstva (Data Tier) je zodpovědná za získávání, ukládání informací v databázi. Zahrnuje uložené procedury, které zvyšují výkon a aplikace je více transparentní.
- Logická vrstva (Logical Tier) je jádro systému, které spojuje prezentační a datovou vrstvu. Tato vrstva určuje, co se bude dít se získanými informacemi.
- Business vrstva (Business Tier) je součástí logické vrstvy a počítá různé agregační hodnoty, součty. Tato vrstva se nezajímá o prezentaci dat ani o způsob jejich ukládání.
- Vrstva přístupu k datům (Data Access Tier) vystupuje v aplikaci jako rozhraní k datové vrstvě. Obsahuje informace o připojení k databázi a získává nebo ukládá do ní data.
- Prezentační vrstva je zodpovědná za komunikaci s uživateli, využívá objekty businness vrstvy.



Obrázek 11: Three Tier Code Generator For ASP.NET – model vrstev

Každá entita v této aplikaci má svůj objekt v patřičné vrstvě. Tyto objekty dědí vlastnosti a metody obecných objektů v dané vrstvě. Při návrhu business a datových vrstev se zpravidla používají dvě alternativy:

- vytvoření jedné třídy, která obstará všem objektům přístup k datům,
- vytvoření datového objektu pro každou entitu, která potřebuje přístup k databázi.

Tato aplikace používá obojí. Microsoft Data Access Application Block for .NET vykonává databázové úkoly a datové objekty mohou mít vlastní uživatelem definované metody, které pomohou při získávání informací, jenž nejsou poskytovány pomocí Application bloku.

V prezentační vrstvě jsou všechny stránky děděny od bázové třídy, která zachytává a definuje šablony a obsahuje společnou hlavičku a zápatí.

Po prozkoumání struktury demonstrační aplikace jsem nikde nenašel designer soubory, takže nepředpokládám, že by se s projektem dalo pohodlně pracovat ve Visual Studiu v design módu. Aplikace ale na rozdíl od předchozího programu generuje i soubory s datovými zdroji, takže je vhodné ji použít i pro vícejazyčné weby. Neobjevil jsem ale nastavení každého sloupce a je těžké dělat podrobnější analýzu, když se mi nepodařilo aplikaci spustit. Více se o této aplikace dozvíte na *http://www.codeproject.com/KB/aspnet/ThreeTierCodeGenerator.aspx*.

2.5.1 Java, NetBeans a JDeveloper

Vývojové prostředí NetBeans, které je primárně určeno pro programovací jazyk Java, nabízí pomoc při vývoji webových aplikací různé průvodce. Dokáže na základě databáze automaticky vygenerovat třídy, které znázorňují jednotlivé tabulky a pomáhají přistupovat k záznamům v databázi jako k objektům. Tím je zajištěna datová vrstva. Logiku má každá aplikace jinou, takže tu si programátor musí vytvořit sám a pro prezentační vrstvu lze použít např. JSF Pages, které vytvoří náhledy na tabulky s tlačítky pro jednotlivé operace a editační formuláře. To vše lze také vygenerovat pomocí průvodce.

Stejně jako NetBeans i JDeveloper nabízí vytvoření datového modelu pomocí Enterprise JavaBeans a grafické uživatelské rozhraní v JavaServer Faces Page. Výhodou tohoto přístupu je, že není třeba nic instalovat a základ pro správu aplikace je vytvořen velmi rychle a jednoduše. Na takto vygenerovaných stránkách ovšem nemůžeme očekávat propojení mezi tabulkami, které jsou svázány cizími klíči, to už je zase v rukou vývojáře.

3 Analýza

Z analyzovaných existujících generátorů ASP.NET aplikací je zřejmé, že v době provádění tohoto malého průzkumu neexistovali snadno dostupné generátory, které by se daly jednoduše použít ve Visual Studiu, aby programátor mohl nadále využít všech nástrojů – především grafických, které toto vývojové prostředí poskytuje. Hlavním cílem navrhované aplikace bude pokořit v tomto směru konkurenční programy. Mohlo by to být trošku na úkor širokému výběru databází. Vzhledem k tomu, že MS SQL je domácí databáze k vývojovému prostředí Visual Studio, určitě bude zahrnuta podpora této databáze. Velmi rozšířenou je v internetových aplikacích taky MySQL databáze. Dva typy databází by mohly ukázat univerzálnost tohoto projektu.

Program na generování webových aplikací z databáze se může tvářit jako jednoduchý průvodce, který pouze vyhledá databázi, zpracuje uživatelem zadaná kritéria a vygeneruje kód. Bude se ale jednat o webovou aplikaci, která může být do budoucna dostupná více uživatelům a poskytovat jim další služby a možnosti. I zde je třeba rozdělit aplikaci do více částí viz dále.

3.1 Použitá architektura a návrhové vzory

Jak už bylo zmíněno v jedné z předchozích kapitol, třívrstvá architektura rozděluje aplikaci do logických vrstev, kde svrchnější vrstva je přímo závislá pouze na sousední nižší vrstvě a využívá jejich služeb. Stejně jako v modulárním programování platí, že je snadnější rozdělit větší celky na menší a lépe se zaměřit na dílčí problémy k vyřešení celku. Architektura .NET poskytuje především objektové nástroje, takže se v každé vrstvě nachází třídy a jejich metody, které se zabývají různými úkoly.

U velmi malých aplikací lze vrstvu aplikace vytvořit třídou. Tím by vznikly např. pouze tři třídy (prezentační, logická a datová) a ty by se staraly o svoje operace. V takovém případě není namístě mluvit o vícevrstvé architektuře.

Dalším velmi častým řešením je oddělení do jmenných prostorů, které seskupují třídy s podobnou funkcionalitou. Vývojové prostředí Visual Studio nabízí pro svoji práci projekty, které se mohou přirovnat k jmenným prostorům. Mají ale tu výhodu, že se dají zvlášť testovat, upravovat optimální spuštění apod. I přes tato oddělená nastavení lze projekty mezi sebou provázat, aby byly zřejmé závislosti. Projekty se pak zapouzdří do tzv. Solution (řešení).

Nejnižší datová vrstva zpravidla využívá dostupné ovladače k databázi a získává z ní data, aby z nich mohla vytvářet objekty. Ideálním projektem pro nejnižší vrstvu je *knihovna*. Její objekty a jejich metody využívá prostřední logická vrstva, která dohlíží, aby se s objekty pracovalo takovým způsobem, aby plnily správné funkce modelovaného systému. I v tomto případě je vhodným řešením *knihovna*. Poslední prezentační vrstvou může být webová aplikace, aplikace pro systémy Windows

nebo třeba Linux a není problém prezentovat data ve strohé textové podobě a zobrazovat je konzolovou aplikací.

Aplikace ASP.NET i bez explicitního rozdělení struktury částečně oddělují zobrazení od logiky a to pomocí aspx souborů a tzv. behind kódu, který je implementován v konkrétním jazyce (C#, VB, …). Behind kód může měnit nejenom zobrazení stránek, ale i jejich přesměrování a provádět libovolné funkce daného programovacího jazyka. Toto oddělení a spoustu dalších předprogramovaných tříd, které lze použít v aspx stránkách, najdeme v tutoriálech na internetu, které ukazují, jak rychle a jednoduše *naklikat* aplikaci, která pracuje s databází, obsahuje editaci i zobrazení a to včetně řazení a stránkování. V praxi jsou takové postupy použitelné jenom pro malé aplikace. Základní chybou, která by se u větších projektů projevila při pokročilejších změnách je např. přístup z aspx souboru přes SqlDataSource přímo do databáze. V této i generované aplikaci se s takovými technikami nesetkáte.

V teoretickém rozboru je kapitola o návrhových vzorech. Není důvod se jim v tomto projektu vyhýbat. Vzniklý software bude pracovat s databází a pokud se nejedná o velmi rozsáhlou aplikaci, která by měla zapotřebí připojovat se současně k více databázím, je vhodné použít návrhový vzor Singleton právě pro připojení k databázi a toto spojení používat v celé aplikaci, aby se nevytvářelo zbytečně další. Když už se zmiňuji o databázích, nabízí se návrhový vzor Factory, kterým lze zajistit použití stejného rozhraní pro různé typy databázových systémů. K dalším možným použití lze dospět během návrhu aplikace, který bude rozebrán v jedné z následujících kapitol.

3.2 Visual Studio a struktura webových aplikací

Pro vytváření .NET aplikací vydala společnost Microsoft vývojové prostředí Visual Studio .NET. Existují i jiná vývojová prostředí, např. PrimalCode .NET IDE. Vzhledem k tomu, že platforma .NET se zrodila právě u výše zmíněné společnosti je pravděpodobné, že i jejich vývojové prostředí bude nejlepší volbou pro tvorbu ASP.NET aplikací.

Aplikace vytvořená ve Visual Studiu je zapouzdřená v tzv. Solution (řešení). To samo o sobě obsahuje jeden binární (nazev_solution.suo) a jeden XML soubor s nastavením (nazev_solution.sln). Samotné solution neobsahuje nic speciálního ale slouží jako obálka pro menší celky, kterými jsou projekty. Ty mohou být mezi sebou různě závislé a navzájem si poskytují třídy a jejich metody, které jsou při vývoji logicky oddělené. Ve vrstvové architektuře je vhodné pro každou vrstvu použít jeden projekt. Typů projektů má Visual Studio 2008 desítky, jsou závislé na programovacím jazyku na účelu aplikace a dalších faktorech. Zde se budu zabývat pouze několika z nich pro programovací jazyk C#.

Snad v každé aplikaci se dá použít projekt zvaný Class Library. Jak již název napovídá jedná se o knihovnu tříd. V ní by měly být seskupeny části z podobné domény nebo s podobnou funkčností. Tento typ projektu lze použít např. pro datovou nebo aplikační vrstvu generovaných aplikací.

V těchto vrstvách nepotřebujeme žádné zobrazení a můžeme jejich třídy použít i pro další projekty, které budou libovolně prezentovat získaná data. Projekt typu Library obsahuje minimálně následující soubory a složky:

- XML soubor (nazev_projektu.csproj) s nastavením projektu a všemi jeho položkami. Soubory, které jsou ve stejném adresáři a nejsou zapsány v tomto souboru jsou ve Visual Studiu ignorovány a ani se nezobrazí.
- Složky pro sestavené soubory a vytvořené dynamické knihovny (obj, bin).
- Složka Properties se souborem AssemblyInfo.cs, který obsahuje údaje o sestavení projektu (assembly[i]) .
- Soubor s minimálně jednou třídou, jinak by nemělo smysl projekt zakládat.

Pro prezentační vrstvu se hodí projekty typu Console Appliaction (konzolová aplikace), Windows Forms Application (pro instalaci na klientské počítače) a ASP.NET Web Application (internetová aplikace). Konzolovou aplikaci jsem použil pouze pro zkoušení a testování při práci s datovou vrstvou, abych si stručným výpisem ověřil správnou funkčnost. Aplikaci pro Windows jsem si poté vytvořil pro zobrazení metadat, která byla rozsáhlejší a špatně se v nich orientovalo v prostém textovém režimu. Generátor webových portálů i jím vygenerované nové aplikace budou používat pro svoji práci projekt typu ASP.NET Web Application. Ten ve svém základu obsahuje více položek než knihovna tříd. Kromě souboru s informacemi o sestavení, souboru s nastavením projektu a složek obj a bin navíc obsahuje:

- Web.config – jak již název napovídá obsahuje konfiguraci daného webu. Skrývá v sobě informace, které ovlivňují načítání, zabezpečení (přihlašování), nastavení session, jazyk aplikace i nastavení kompilace. Může také obsahovat uživatelská nastavení, které mohou řídit funkce aplikace.
- App_Data – složka, do které ukládají data, které nebudou přímo přístupné přes http(s) protokol, takže se k nim uživatel nedostane pouhým zadáním cesty do url v prohlížeči.
- Alespoň jednu ASP stránku (nazev_stranky.aspx), ve které se definuje její vzhled, ale může obsahovat i metody, které napomáhají k správné prezentaci dat.
- Soubor designu k ASP stránce (nazev_stranky.aspx.designer.cs), do kterého ukládá Visual Studio objekty, které jsou použité na stránce. Soubor obsahuje kód ve zvoleném programovacím jazyce (C#). Uložené objekty lze pak použít v behind kódu.
- Tzv. behind kód soubor (nazev_stranky.aspx.cs) ovlivňuje logiku zobrazení stránky. Může obsahovat přesměrování a např. předzpracování dat před zobrazením apod. Soubor je ve zvoleném programovacím jazyce (C#).

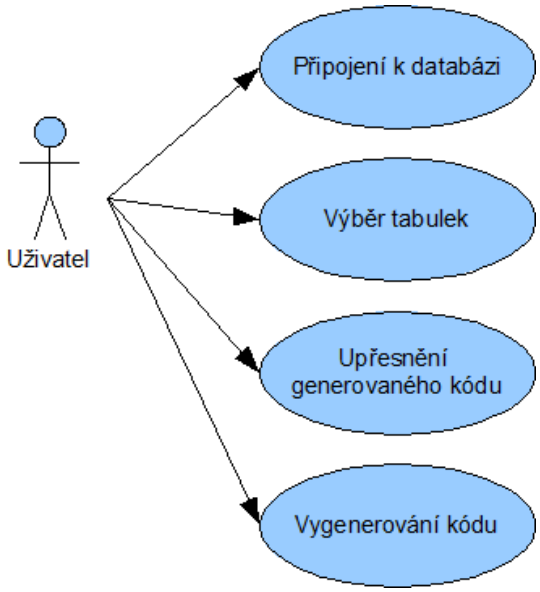
i Assembly (česky sestavení) je balíček obsahující metody, pole, vlastnosti, rozhraní, obrázky a mnoho dalšího. Všechny tyto entity existují v assembly v IL (Intermediate Language) formátu a jsou zkonvertovány do strojového kódu až za běhu pomocí CLR (Common Language Runtime). Assembly může existovat ve dvou podobách: Portable Executable (EXE soubor), Dynamic Link Library (DLL soubor).

U menších aplikací se můžete setkat s tím, že prezentaci dat zajišťují pouze ASP stránky a aplikační logiku a práci s daty obstarává právě behind code. Pro jednodušší aplikaci bez dalšího vývoje je to ale dostačující.

Výše popsané soubory jsou základní soubory prázdných projektů, samozřejmě k nim lze přidávat soubory s vlastnostmi, nastaveními, jazykovými konstantami a jinými datovými zdroji apod. Klíčovým nastavením je připojovací řetězec k databázi (ConnectionString), který nebude uložen v souboru Web.config, jak bývá zvykem, ale bude u datové vrstvy, protože ta s databází pracuje. Ostatní vrstvy v podstatě netuší, odkud data pochází.

3.3 Základní požadavky na generátor

Vzhledem k tomu, že tento projekt se zaměřuje především na kvalitní a snadno použitelný vygenerovaný kód, je samotný generátor velmi jednoduchou aplikací. Jeho funkční požadavky shrnuje následující diagram případů použití (Obrázek 12).



Obrázek 12: Diagram případů použití – generátor kódu

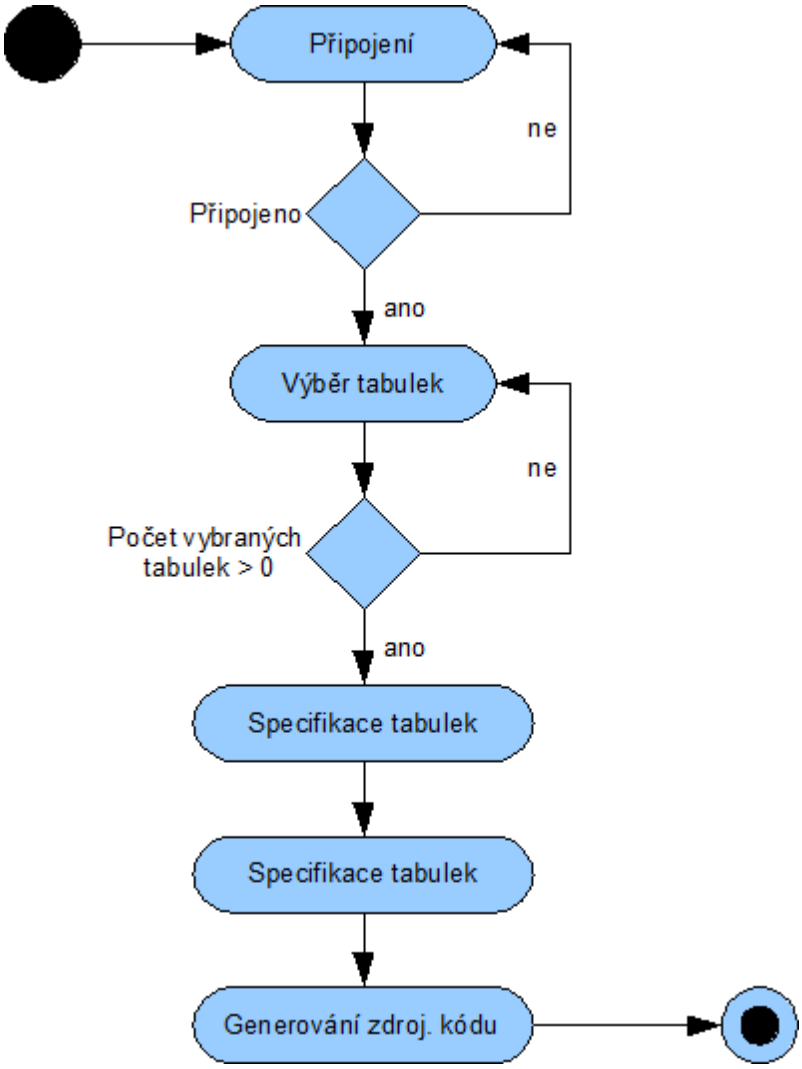
Časová posloupnost případů je pravděpodobně snadno odhadnutelná, ale pro úplnost ji znázorňuje diagram aktivit (Obrázek 13). Uživatel se nejprve připojí k databázi, zvolí tabulky, pro které chce vygenerovat zdrojové kódy. Upřesnění pro nejsložitější část pro uživatele bude obsahovat možnosti pro:

- uživatelem definované názvy sloupců a tabulek, které se budou zobrazovat,
- definice, jestli bude možné tabulku editovat nebo jenom prohlížet její data,
- návaznost na tabulky provázané cizími klíči (alespoň pro vazbu 1:N),
- způsob zobrazení hodnot jednotlivých sloupců v tabulce.

Na generátor je kladen mnohem větší důraz u nefunkčních požadavků a těmi jsou:

- podpora více typů databází,
- schopnost generovat datové objekty se základními typy jako jsou čísla, texty a logické hodnoty,
- generovat názvy objektů a vlastností podle názvu tabulek a sloupců,
- vytvořit *Solution* pro Visual Studio 2008,
- zachovat *designer* soubory, aby bylo možné projekt upravit v *design módu,*
- rozdělit *Solution* do projektů podle vrstev ve třívrstvé architektuře,
- generovat prezentační vrstvu jako ASP.NET stránky s použitím v *Master pages* a kaskádových stylů,
- navrhnout generátor tak, aby se jeho prezentační vrstva dala implementovat jako aplikace pro Windows,
- vytvořit projekt tak, aby se výstupní generovaný kód mohl změnit bez zásahu do aplikace, např. pomocí šablon kódu,
- výstupem bude jediný komprimovaný soubor, uvnitř kterého bude vytvořené *Solution.*

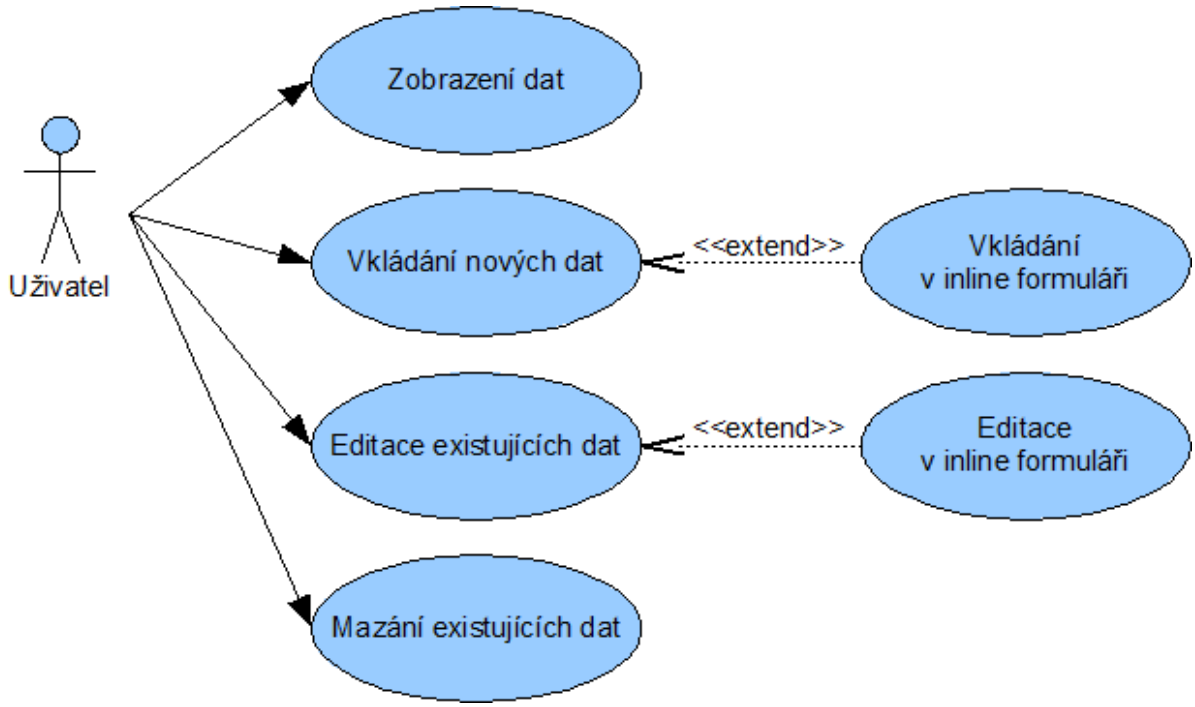
Podrobnější specifikace vygenerovaného kódu bude uvedena v následující kapitole.



Obrázek 13: Diagram aktivit – generátor kódu

3.4 Požadavky na kód generované aplikace

Funkční požadavky generované aplikace jsou shodné se základními operacemi, které se dají s databází provádět. Jedná se o vkládání, editaci, mazání a zobrazení dat. Vzhledem k tomu, že ASP.NET umožňuje velmi jednoduše vkládat záznamy přímo v okně tabulky, jsou přidány i rozšiřující případy pro tzv. inline vkládání/editaci (Obrázek 14).



Obrázek 14: Diagram případů použití – generovaná aplikace

Vytvořené *Solution* bude základem pro nový informační systém, programátor se v něm musí velmi rychle zorientovat, proto jedním z hlavních požadavků je použití standardních vizuálních prvků v prezentační vrstvě. Datová vrstva bude objektově co nejvěrněji kopírovat strukturu tabulek v databázi, aby bylo na první pohled zřejmé s jakými daty se pracuje. Obsah aplikační vrstvy si autor nového systému vytvoří sám. Budou pouze vygenerovány základní řídící třídy pro práci s datovými objekty. Není nutné ošetřovat ve vzniklém kódu všechny výjimky, protože autor do jisté míry bude stejně upřesňovat formáty vstupních dat. Stejně tak není potřeba kontrolovat, jestli se názvy sloupců neshodují s některými klíčovými slovy programovacího jazyka, ve kterém se aplikace vygeneruje. V případě, že nastane kolize pojmenování, je na programátorovi, aby změnil pojmenování. Pokud bude aplikace přehledná, nebude to činit žádné problémy.

Další funkční požadavky:

- zobrazení, vkládání a editace textových hodnot, čísel, logických hodnot, volitelně i binárních dat,
- zobrazení dat jako pouhý text, hypertextový odkaz nebo obrázek,
- editace dat pomocí textového pole (TextBox), zaškrtávacího políčka (CheckBox) nebo možnost volby ze seznamu definovaných hodnot (DropDownList),
- možnost vybírat hodnoty cizího klíče ze seznamu (DropDownList),
- rozdělení webové aplikace na veřejnou a privátní část.

Hlavními nefunkčními požadavky jsou:

- kompatibilita s vývojovým prostředí Visual Studio 2008,
- možnost editace projektu i v Design módu.

3.5 Bezpečnost

Pokud bude webová aplikace v provozu přímo na internetu a ne jenom v lokální síti, měla by být v provozu na zabezpečeném protokolu https, protože uživatel bude zasílat citlivé přihlašovací údaje ke vzdálené databázi. Ideální by bylo použít server s důvěryhodným certifikátem ověřený certifikační autoritou. Výstupní soubor s celým řešení by pak neměl obsahovat heslo ani přihlašovací jméno k databázi. Přihlašování k aplikaci jako takové není potřeba, aplikace by mohla být součástí většího portálu podobných nástrojů a ten by zabezpečil autentizaci uživatele.

Na vygenerovanou aplikaci žádné speciální bezpečnostní požadavky nejsou. V tomto směru je ponechána bezpečnost na tvůrci nového systému. Velmi jednoduše lze např. zapsat do souboru web.config nastavení autorizace a vytvořit přihlašovací stránku, takže by neměl být v budoucnu problém rozšířit generátor tak, aby i nová aplikace obsahovala přihlášení do privátní části generovaného systému.

Zabezpečení aplikační vrstvy je ponecháno na tvůrci nové aplikace, ale vygenerovaná datová vrstva musí mít aspoň základní zabezpečení. Nesmí povolit útoky na databázi a to především pomocí techniky SQL injection, takže je nutné na nejnižší úrovni ošetřit vstupní data.

4 Návrh a implementace

Po vytyčení cílů a základních požadavků je třeba se zamyslet nad způsobem realizace. Generátor jako takový může mít jednoduché rozhraní, které působí jako průvodce nastaveními a ve výsledku vytvoří vygenerovanou aplikaci. Před tvorbou generátoru jsem si nejprve zkusil jednoduchou ukázkovou aplikaci, která by měla být výsledkem. Samozřejmě jsem používal základní prvky, které ale využívají vymožeností ASP.NET i Visual Studia, myslel jsem na to, že aplikace se bude automaticky generovat a kód by tudíž neměl být odlišný v podobných situacích a případech. Zároveň jsem dbal na to, aby programátor neměl svázané ruce jednoúčelovým elementem. V tomto případě mluvím především o objektech třídy ListView, která poskytuje největší možnosti a přitom je jednoduše dostupná z nástrojové lišty Visual Studia. Na následující tabulce můžete vidět podporované funkce i podobných prvků.

Třída	Stránkování	Seskupování dat	Flexibilní rozvržení	Editace, mazání	Vkládání	Řazení
ListView	Ano	Ano	Ano	Ano	Ano	Ano
GridView	Ano	Ne	Ne	Ano	Ne	Ano
DataList	Ne	Ano	Ano	Ne	Ne	Ne
Repeater	Ne	Ne	Ano	Ne	Ne	Ne

Tabulka 2: Podporované funkce ASP prvků pro zobrazení seznamů

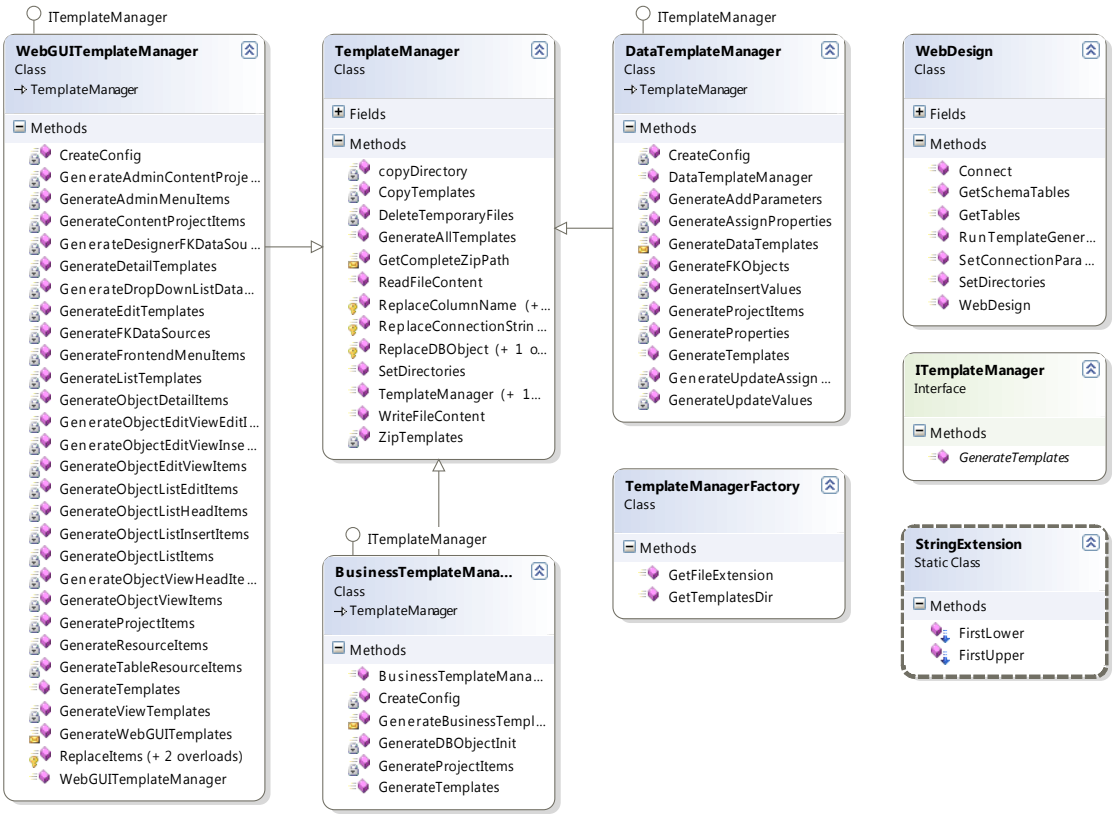
První ukázková aplikace pracovala pouze s celými čísly a textovými řetězci, které mohly být editovány přes textové pole. Dvě tabulky, kde byla uložena data, neobsahovaly cizí klíče a jejich celočíselný primární klíč, který měl vlastnost AUTO_INCREMENT, tvořil identifikační číslo objektu. Tato vlastnost je v aplikaci stále. Každá tabulka musí mít právě takový primární klíč. Vývoj software se odehrával postupně jak ukazuje diagram aktivit generátoru kódu (obrázek 13). Jakmile bylo potřeba získat z datové vrstvy nějaká data, která nebyla schopná poskytnou, rozšířila se datová a aplikační vrstva, aby se informace dostaly až k uživateli prostřednictvím internetových stránek. První verze generátoru dokázala vytvořit jednoduchou aplikaci s několika základními typy bez cizích klíčů. Následovalo rozšíření o další typy, zobrazovací prvky na stránkách a práce s cizími klíči. Tato rozšíření se implementovala a testovala v dalších iteracích.

Vývoj generátoru začal od uživatelského rozhraní, které se podobá konkurenčním nástrojům. Nejprve se tedy připojí k databázi. K tomu je potřeba mít patřičný ovladač. Připojení k MS SQL Server proběhne ve Visual Studio v pořádku, pro připojení k MySQL jsem použil MySQL ODBC Connector verze 5.0.9. Až v průběhu řešení jsem zjistil, že všechna metadata o tabulkách a jejích klíčích nelze získat pomocí existujících metod ADO.NET, takže je třeba sestavit složité dotazy, které se bohužel mohou v novějších verzích měnit.

Postup vývoje začal tím, že jsem si nejdříve vytvořil jednoduchou databázovou aplikaci, která měla všechny typy vazeb mezi tabulkami a obsahovala většinu používaných typů hodnot.

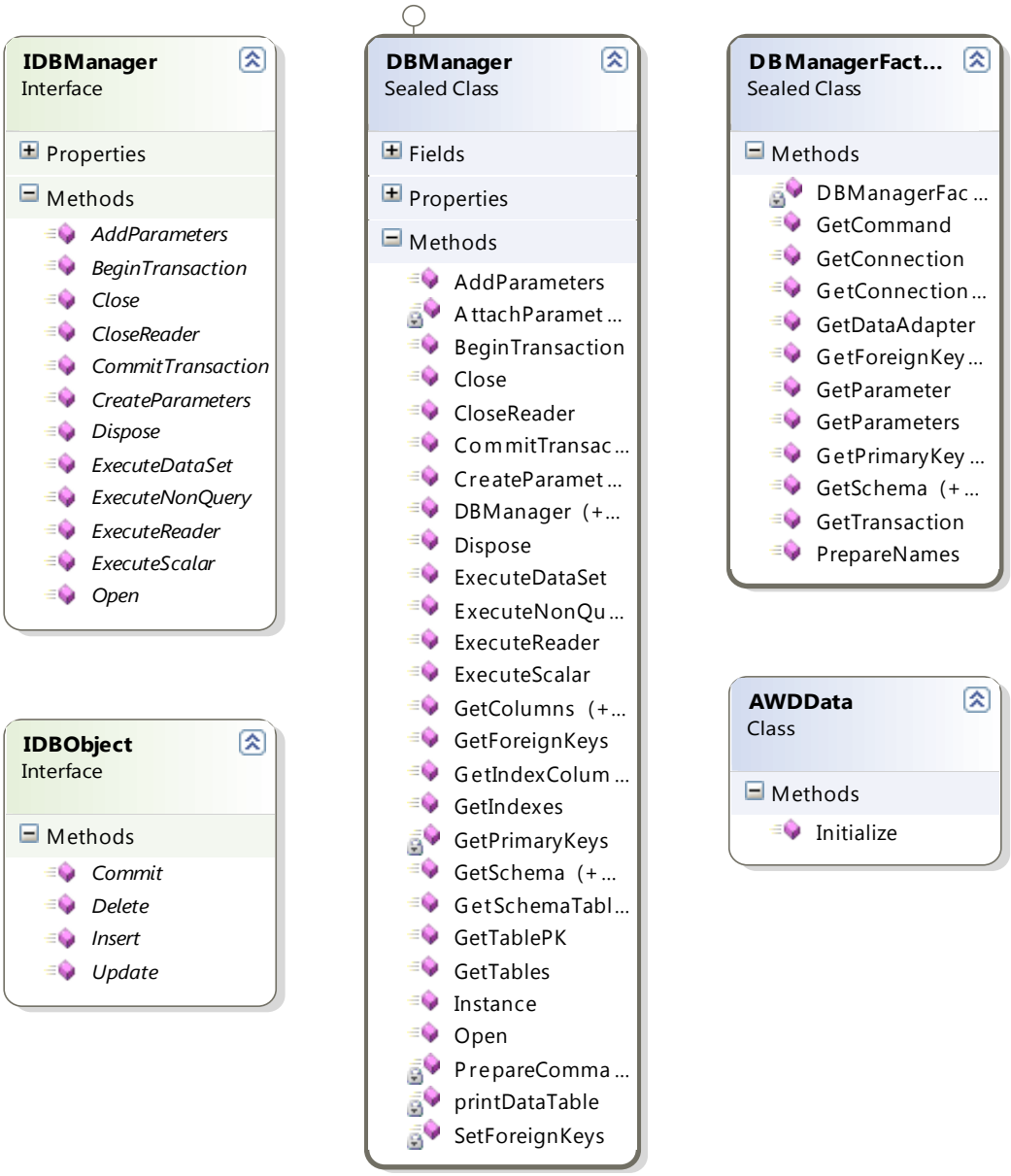
4.1 Struktura projektu

Generátor je navržen podle třívrstvé architektury. V nejnižší datové vrstvě jsou nejdůležitějšími třídamy třídy *DBManagerFactory* a *DBManager*. Jsou navržené podle návrhového vzoru Factory. Třída DBManagerFactory svými metodami podle parametrů vrací objekty pro práci s konkrétním typem databáze. Třída DBManager tuto třídu využívá a systém pouze nastaví typ databáze a pak s ní pracuje stejnými metodami bez ohledu na její typ.



Obrázek 15: Diagram tříd – aplikační vrstva (AutoWebDesign)

Aplikační vrstva se stará o zpracování dat a generování nového projektu. V podstatě se zapojuje až po všech nastaveních při stisknutí tlačítka *generovat*. Tato vrstva funguje na bázi šablon. Generátor je připravený na vytváření aplikací pro různé programovací jazyky. Podle toho jsou šablony odděleny v různých podadresářích. Většina generovaného kódu pro konkrétní programovací jazyk je umístěna ve složce, kde existuje prázdné řešení a šablony. Více se o nich dozvíte v následující podkapitole. Při vygenerování všech souborů nového projektu se aplikační vrstva postará o komprimaci celého vytvořeného řešení a poskytne výsledný soubor prezentační vrstvě.



Obrázek 16: Diagram tříd – datová vrstva (DataLayer)

Prezentační vrstva je z hlediska návrhu nezajímavá a není ničím výjimečná. Použitím Master Page je zachován jednotný vzhled a využitím kaskádových stylů lze změnit vzhled webových stránek. Lze přidat další část prezentační vrstvy, kterou bude aplikace windows spustitelná na lokálním počítači bez použití webového serveru.

Prezentační vrstva obsahuje:

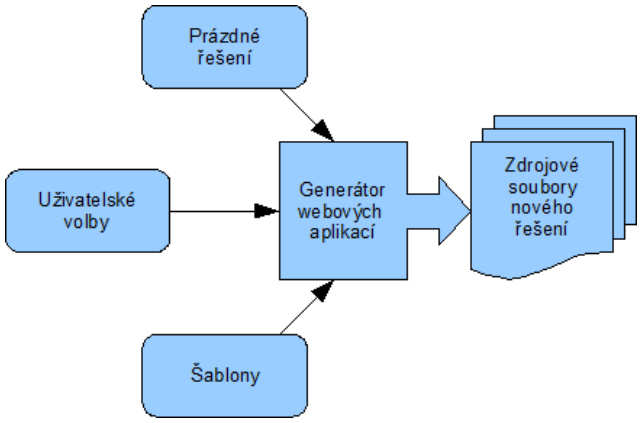
- přihlašovací formulář,
- výběr tabulek,
- nastavení jednotlivých tabulek.

Nejsložitější je generování stránky pro zobrazení nastavení jednotlivých souborů. To se totiž neděje prostřednictvím ASP kódu, ale vše se odehrává v kódu na pozadí, který do stránky vkládá HTML tagy a jejich obsah. Je to řešení poněkud krkolomné, ale zdálo se ze začátku nejjednodušší.

4.1.1 Šablony

Jak již bylo zmíněno, aplikační vrstva pracuje se šablonami, které jsou pro každý programovací jazyk ještě rozděleny na:

- prázdné řešení
- a šablony.



Obrázek 17: Princip generátoru webových stránek

Při generování nové aplikace se prázdné řešení kopíruje do pracovního adresáře a stává se jejím základem. Datová vrstva tohoto řešení obsahuje pouze bázové třídy a rozhraní, Singleton pro připojení k databázi a další soubory (např. konfigurace) společné pro libovolné generované aplikace. Stejně tak projekt aplikační vrstvy, ten v sobě ukrývá pouze hlavní Controller pro inicializaci datových objektů. Nejsložitější je prezentační vrstva, ta obsahuje Master Pages pro veřejnou i administrační část a Resources pro jazykové konstanty. Soubory prázdného řešení se z nové aplikace již nebudou nikde kopírovat. Veškeré změny, které probíhají uvnitř těchto souborů, spočívají v nahrazení tzv. proměnných šablony, po těchto operacích se vše uloží znovu do těchto souborů.

Další částí jsou šablony. Jsou oddělené od prázdného řešení. Na jejich základě vznikají nové soubory, které se pak vkládají do prázdného řešení. Šablony jsou také rozděleny do tří vrstev a obsahují mnohem více souborů a podsložek než prázdné řešení, protože jsou v nich i soubory pro podsekce generovaných objektů (např. privátní proměnné, přiřazení ve funkcích). Datová vrstva obsahuje šablonu pro datové objekty, které jsou téměř shodné s názvy zdrojových tabulek z databáze. Sloupce tabulky jsou pak členskými proměnnými těchto objektů. Programátor se snadno zorientuje a ví s čím pracuje. Toto řešení má však i své nevýhody. Pokud se zvolí názvy tabulek nebo sloupců, které jsou klíčovými slovy, nemusí vygenerovaný kód fungovat. Obrázek *Šablona datového objektu*

vám poskytne náhled do jedné z mnoha šablon. Jejich proměnné používají jako uvozovky znak mříže #. Tyto proměnné mohou být nahrazeny dvěma způsoby:

- z řetězce zapsaného přímo v kódu generátoru,
- nahrazením další šablonou (obsahem dalšího souboru).

Při nahrazení další šablony se můžou její proměnné nahradit zase další šablonou. Obecně se tento proces může zanořovat, ale v této aplikaci se používá maximálně jedno zanoření. Operace jsou stále přehledné a zároveň jsou šablony stále variabilní. Šablony v této vrstvě jsou závislé na datových typech sloupců resp. na jejich ekvivalentních typech v daném programovacím jazyce.

Aplikační vrstva šablon je poměrně jednoduchá. Obsahuje Controller pro každý datový objekt. Controllery se starají o načítání a ukládání dat z datové vrstvy do datových objektů (nikoliv do databáze). Svými metodami pak poskytují data prezentační vrstvě.

```
namespace DataLayer
{
  public class #DBObject# : DBObject, IDBObject
  {
    protected static String tableName;
    protected static String oidColumnName;
    protected static Hashtable cache;

#properties#
#FKObjects#

    public #DBObject#()
    {
      Init();
    }

    public static void Init()
    {
      tableName = "#TableName#";
      oidColumnName = "#DBObjectID#";
      if (cache == null)
      {
        cache = new Hashtable();
      }
      SetConnection();
    }

    public static #DBObject# Get#DBObject#(long oid)
    {
      return Get#DBObject#(oid, true);
    }
...
```

Obrázek 18: Šablona datového objektu

Nejsložitější jsou šablony prezentační vrstvy, protože právě té se týká nejvíce nastavení v poslední fázi před generováním vytvářené aplikace. Při generování této vrstvy musí být v šablonách soubory, které umožní podle uživatelského nastavení provést následující operace:

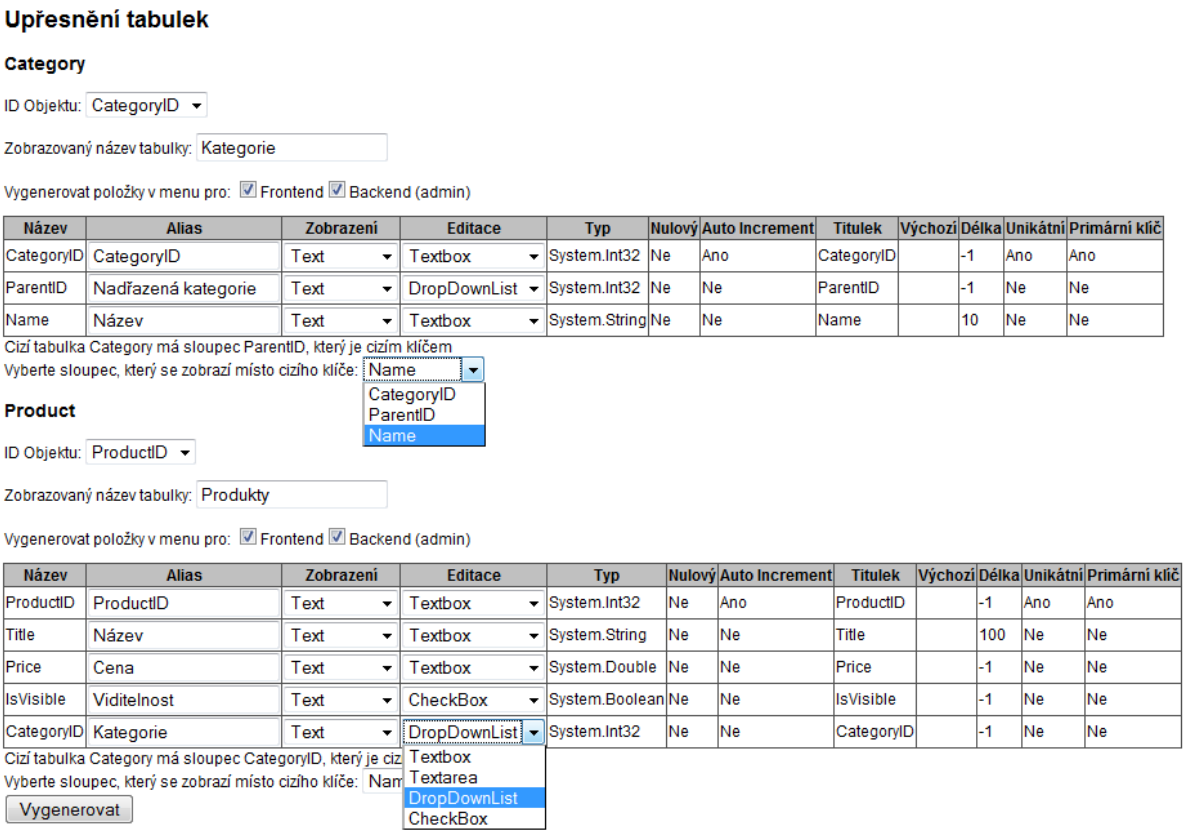
- vytvoření menu pro přístup k tabulkám na veřejné i administrační sekci,
- zobrazení souhrnných přehledů dat ve veřejné části,
- zobrazení detailních informací o datech ve veřejné části,
- zobrazení v administrační sekci,
- editační formuláře pro správu dat,
- generování jazykových konstant.

Stránky pro zobrazení i editaci dat jsou závislé nejen na datových typech, ale také na volbě způsobu zobrazení případně editace. Data z databáze lze prezentovat např. jako text nebo obrázek a v editační části lze data zadat přes seznamy nebo pomocí textových polí, které obsahují validátory pro konkrétní datový typ. Součástí šablon prezentační vrstvy jsou i zdroje dat (Resources), které umožní programátorovi upravit všechny textové řetězce, které se zobrazují uživateli a nejsou získány z databáze za běhu aplikace. Konstanty jsou logicky pojmenovány stejně jako datové třídy a jejich vlastnosti.

Aby nebylo potřeba vytvářet šablony pro všechny datové typy a všechny způsoby zobrazení, systém zjišťuje, jestli existuje šablona pro daný typ a zobrazení, pokud ne, snaží se získat obecnou šablonu pro zobrazení a pokud ani tu nenajde, získá obecnou šablonu. Toto se týká zobrazení konkrétních hodnot dat z databáze, které se zobrazují ve editačních formulářích, detailech i souhrnech. Má-li systém k dispozici např. šablonu pro textová data, která poskytuje validátor pro datum, použije ho. Pokud takovou šablonu nenajde, snaží se najít obecné textové pole, které už samozřejmě validátor obsahovat nebude a pokud neexistuje ani šablona pro textové pole, použije šablonu obecnou, která v tomto případ bude stejně textové pole, protože je to základní vstupní prvek pro vkládání dat.

Systém šablon má tu výhodu, že lze měnit novou generovanou aplikaci bez změny v generátoru. Při změně šablon je nejvhodnější použít stávající šablony a postupně je upravovat k obrazu svému.

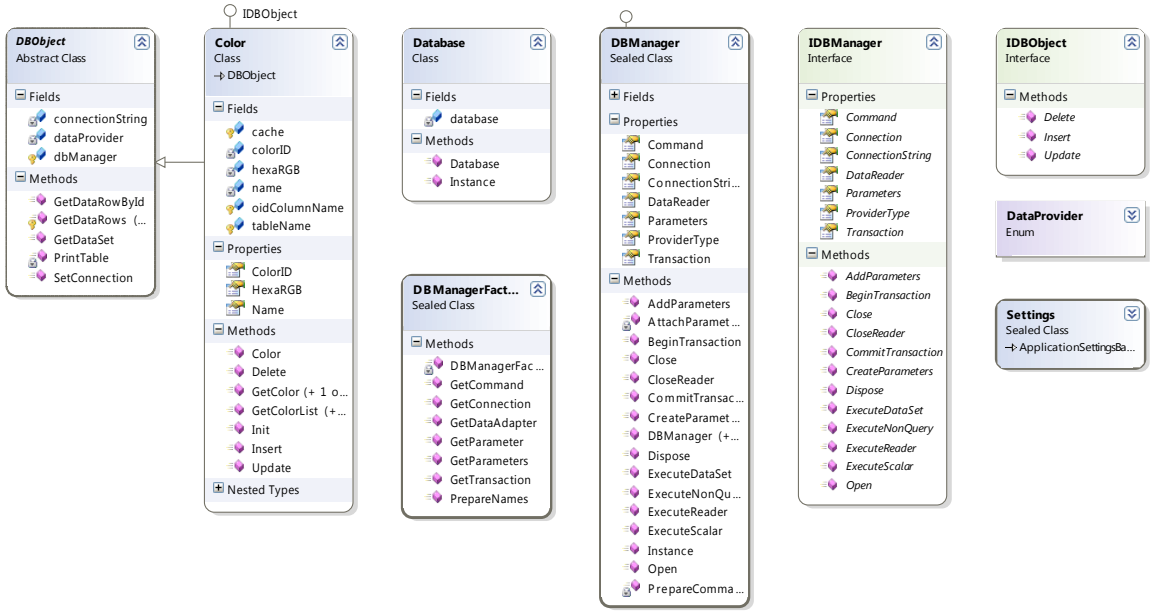
Všechny vrstvy obsahují také konfigurační soubory, ve kterých je třeba zajistit, aby se v nich vyskytovaly všechny použité soubory v projektu, protože jinak nebude možné s vygenerovanými zdrojovými kódy pohodlně pracovat v prostředí Visual Studio.



Obrázek 19: Generátor webových aplikací – uživatelské rozhraní aplikace

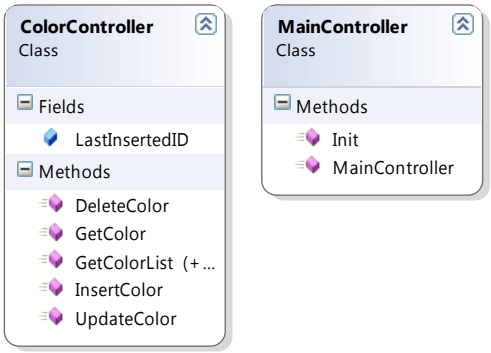
4.2 Nový webový portál

Generovaný portál má třívrstvou architekturu, která byla částečně popsána v kapitole o šablonách, protože z nich se nová aplikace generuje. Obrázek 20 zachycuje kompletní diagram tříd datové vrstvy nové aplikace v téměř nejjednodušší podobě. Byl vygenerován z jedné tabulky s názvem *Color*. Všechny třídy a metody, kromě třídy *Color*, se budou vyskytovat ve všech nově vygenerovaných aplikací.



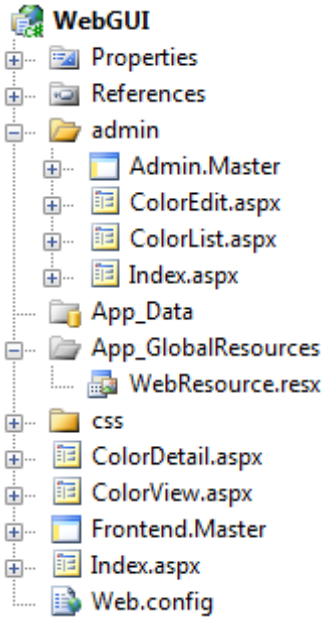
Obrázek 20: Digram tříd jednoduché nové aplikace

Na dalším obrázku (Obrázek 21) můžete vidět aplikační vrstvu, která má vždy třídu *MainController* a ostatní jsou generované podle tabulek.



Obrázek 21: Diagram tříd nové aplikace – aplikační vrstva

Pro prezentaci prezentační vrstvy jsem zvolil pouze adresářovou strukturu webového projektu, který zobrazuje data. Ve všech aplikacích se objevují Master Pages pro veřejnou i administrační část, úvodní soubory (index.aspx), dále pak datové zdroje (Resources), kaskádové styly (css) a soubor s konfigurací. Samozřejmě nemluvím o souborech, které jsou vždy součástí všech projektů Visual Studio. Ostatní čtyři soubory (tedy i třídy) které obsahují slovo *Color* jsou generované právě pro tabulku *Color*.



Obrázek 22: Adresářová struktura nové aplikace – prezentační vrstva

5 Problémy při implementaci

Při řešení toho projektu jsem narazil na řadu problémů, které jsem bohužel zjistil až při implementaci a v některých případech bylo už pozdě měnit návrh a předělávat celou aplikaci. Při návrhu jsem vycházel z předpokladu, že ADO.NET poskytuje třídy a metody, které fungují nad všemi podporovanými databázemi stejně a bude triviální připojit se a získat veškeré informace o tabulkách. Narazil jsem v případě připojení k MySQL přes ODBC konektor. Při načtením tabulek do objektu typu DataSet se ztratí cizí klíče, takže by uživatel musel všechna propojení definovat znovu a mohlo by tak dojít i k nekonzistenci dat. Řešením je získání těchto dat přímo ze systémových tabulek databázového systému pomocí složitého SQL dotazu. Tato metoda s sebou nese riziko v případě, že se struktura systémových tabulek změní.

Další nedostatkem použitého konektoru pro MySQL je předávání parametrů do SQL dotazu pomocí metody *OleDbCommand().Parameters.Add()*. Parametry jsou ignorovány. Je to chyba, která je u tohoto konektoru známá, ale přesto se s ním nic neudělalo. Bylo nutné přistoupit k vlastnímu sestavování dotazů a tím pádem i k řešení, které nedovolí útok pomocí sql injection. V šablonách a tím pádem i ve zdrojových kódech vygenerovaných aplikací ale stále zůstaly zakomentované části, které používají nefungující metody. To je pro případ, až bude konektor fungovat správně. Jedná se o změnu na dvou místech, takže oprava bude jednoduchá.

Oba zmiňované problémy bylo nutné vyřešit i když s nimi původní návrh nepočítal, ale bez nich by aplikace nefungovala.

S jazykovými zdroji dat jsem počítal hned od začátku. Internet je celosvětová síť a mnoho portálů je vícejazyčných. Platforma .NET pro to poskytuje skutečně jednoduché řešení, kterým jsou zdroje dat (Resources). Pro přehlednost měl být každý zdroj oddělen podle datových objektů, se kterými pracuje. Takže pokud by existovaly v systému tři tabulky, existovali by kromě společných zdrojů také tři zdroje ke každé tabulce. Datové zdroje se ovšem zapisují do binárního soubor celého řešení (.suo) a jeho struktura není dostatečně jasná na to, aby se dal soubor změnit a přidat do něj další zdroje. Řešením je tedy jeden společný zdroj, který je obsažen už v *prázdném řešení*. Do tohoto zdroje se pak přidává XML kód s jazykovými konstantami, které jsou použity v ASP stránkách nového portálu.

6 Testování a rozšíření

Při vývoji rozsáhlejšího softwaru bez přesné specifikace nikdy není zcela jasné, kdy je projekt kompletně hotový. Stále je možné něco vylepšovat a zdokonalovat. Když už ne po funkční stránce, tak lze optimalizovat algoritmy uvnitř zdrojových souborů a vylepšovat vnitřní strukturu nebo třeba komunikaci. V praxi je často důležité, aby taková rozšíření viděl zákazník, takže v případě webových portálů se to musí projevit v prezentační vrstvě.

Generátor a jím vytvářené aplikace jsou teprve v první fázi, obsahují základní nezbytnou funkcionalitu a je kladen důraz na to, aby se ve zdrojových kódech dobře orientoval programátor. Tato práce už se ale nezabývá ergonomií z pohledu návštěvníků vygenerovaných aplikací. K lepší ovladatelnosti vygenerovaných aplikací by přispěla následující vylepšení:

- **Třídění** – Běžnou součástí např. internetových obchodů je třídění produktů podle určitých kritérií. Pro třídění sloupců dané tabulky by řešení nemělo být příliš obtížné. Realizovatelné je i v přehledech se sloupci z cizích tabulek.
- **Kalendář** – Současná verze generátoru dokáže vygenerovat vstupní políčka s validátory pro datové typy System.Date. Uživatel musí napsat datum (čas) ve správném formátu, pro většinu uživatelů je mnohem pohodlnější vybrat datum z vizuálního kalendáře, který může být implementován JavaScriptem. Vhodným kandidátem je např. Basic Date Picker (*http://www.basicdatepicker.com*), který je bohužel dostupný jenom za poplatek, ale existují i jiné nástroje. Realizace by neměla zasahovat do samotného generátoru ale pouze do šablon.
- **Zapamatování parametrů zobrazení** – Je velice nepříjemné pokud si uživatel vybere v internetovém obchodě produkty podle jeho požadovaných parametrů a potom si je seřadí podle ceny a při návratu po vložení do košíku zjistí, že musí filtr pro zobrazení nastavovat znovu. Toto vylepšení určitě přispěje k ergonomii nových aplikací.
- **WYSIWYG Textarea** – Ve webových redakčních systémech se už běžně vyskytují editory, které se podobají aplikacím jako je Micrososft Word nebo OpenOffice.org Writer. Jejich internetoví konkurenti neumí úplně všechno, ale dokáží formátovat text, vkládat obrázky, některé i nahrávat obrázky na server, používat styly apod. Formátovat texty lze např. pomocí různých značkovacích jazyků, které používají např. wikicitáty, ale proč učit uživatele něco nového. Řešení se nabízí hned dvě TinyMCE nebo FCKeditor. První uživatelsky přívětivější a pohodlněji se s ním pracuje, na druhou stranu druhý má zdarma i souborový manager, který umožní nahrávat i obrázky na server. Obě tyto verze je nutné zakomponovat do šablon.
- **Validátor na délku textového pole** – V šablonách pro nový projekt už jsou validátory na datové typy a na hodnoty, které nemohou být prázdné. Chybí tam ale validátor, který by

ošetřil délku vstupu pro sloupce s omezenou délkou. Prozatím je to na programátorovi nové aplikace.

- **Obrázky v základních šablonách** – Je všeobecně známo, že malý obrázek je někdy lepší než sáhodlouhé vysvětlování. Základní šablony neobsahují žádné obrázky, ale bylo by vhodné je použít např. u navigačních prvků nebo tlačítek. Obrázky musí být srozumitelné a dostatečně malé, aby daly prostor na stránce hodnotnému obsahu.
- **Vyhledávání** – V dnešní době nemá portál bez vyhledávání šanci uspět na trhu. Uživatelé jsou netrpěliví a chtějí najít své informace rychle a co nejjednodušší cestou. Není nic jednoduššího než zadat klíčové slovo a spoléhat na to, že systém má kvalitní vyhledávací mechanismus, který vrátí relevantní výsledky. Další verze této aplikace by mohla poskytovat alespoň jednoduché vyhledávání, které by hledalo části slov ve vybraných sloupcích všech tabulek.
- **Agregační funkce** – Peníze hýbou světem a narazíme na ně snad všude. Když to nejsou peníze, tak se setkáme s jinými číselnými hodnotami, které udávají cenu. Pokud procházíme takové přehledy dříve nebo později nás bude zajímat, jaký je součet nebo počet zobrazených hodnot a další agregační funkce, které se používají při statistikách. Databáze tyto funkce podporují, takže lze o tyto informace generovanou aplikaci obohatit.

Všechna výše uvedená rozšíření se týkají především nové aplikace. Snadno ovladatelný a hlavně srozumitelný by měl být i samotný generátor. I pro něj existuje mnoho rozšíření:

- **Jazykové verze** – Vzhledem k tomu, že se jedná o internetovou aplikaci a může tak být dostupný celému světu, měl by do budoucna umožňovat přepínání jazykových verzí.
- **Uživatelské šablony** – Generátor umožňuje využití šablon právě proto, aby si je uživatelé mohli vytvořit sami a v případě opakovaného používání budou mít ušetřenou další práci a nový projekt se přiblíží více jejich představám. Dalším vylepšením je tedy použití uživatelských šablon, které si uživatel nahraje server, aby z nich mohl vygenerovat nový portál.
- **Více cizích sloupců** – Existuje-li např. v internetovém obchodě nějaký produkt, je zpravidla zařazen do kategorie. Pokud chceme vidět detail produktu, budeme předpokládat i nějaké informace o jeho kategorií – nejen její číslo. V této verzi dokáže generátor vytvářet aplikace, které místo ID záznamu v databázi pracují se zástupným sloupcem cizí tabulky, což je uživatelsky mnohem přívětivější. Bylo by ale zajímavé, pokud by si mohl uživatel zvolit víc než jenom jeden zástupný sloupec. To se především týká veřejné části nového portálu.
- **Volba editace** – Generátor vytváří část pro správu obsahu s tzv. inline formuláři i formuláři na oddělených stránkách. To proto, aby programátor jenom smazal to nepotřebné a ponechal, co se mu líbí. Je to rychlejší než trápit vývojáře s vytvářením

chybějící varianty. Úplně nejrychlejší by ale bylo, kdyby si toto nastavení mohl provést ještě před vygenerováním.

- **Všechny datové typy** – Největší výzvou do budoucna je podpora všech typů dat, které databáze umožňují ukládat. Bohužel nejsou zatím podporované binární datové typy, které se hodí především pro ukládání souborů libovolného formátu. Prozatím se to dá řešit ukládáním souborů na disk a zapsáním cesty do tabulky jako text.
- **Více typů zobrazení/editace** – Zobrazení dat lze provádět několika způsoby. Ve výsledné generované aplikaci lze zatím použít jenom několik základních. S binárními daty souvisí jejich upload a download. Doplněním těchto funkcí by byl velký přínos.

O rozšíření a využití by se dalo napsat mnoho stran. Na závěr této kapitoly můžete vidět jenom seznam několika z nich:

- využití AJAXu pro lepší ovládání,
- kompatibilita pro různá vývojová prostředí,
- podpora více databází a programovacích jazyků,
- optimalizace pro rychlejší práci s databází,
- podpora pohledů a uložených procedur.

6.1 Ladění a testování

Testování aplikace, která není jednoúčelová není jednosměrné. Generátor se v prvním kroku připojuje na více databází. Každá z nich má trošku jiné datové typy i když ty se automaticky mapují na datové typy konkrétního programovacího jazyka pomocí ADO.NET. Ne vždy je ale pomoc dostačující. S každou novou volbou, jak editovat data z databáze přichází testování přes všechny typy a všechny podporované databáze. Univerzální aplikace musí být velmi dobře ošetřeny před neočekávanými vstupy. Navíc se musí testovat generátor i vygenerovaná aplikace. Následující testy byly provedeny pro databáze MS SQL i MySQL.

Název testu	Připojení k databázi.
Vstup	Prázdné hodnoty, nevalidní hodnoty, neexistující hodnoty parametrů pro přijení
Popis	Test připojení k databázi.
Požadovaný výstup	Musí ohlásit chybu (anglicky nebo česky), pokud se nepodaří připojit.

Název testu	Test datových typů databáze.
Vstup	Tabulka se všemi typy sloupců.
Požadovaný výstup	Nesmí povolit nepodporované typy.

Tabulka 3: Testy pro generátor webových aplikací

Název testu	Testování základní operací s databází .
Vstup	Tabulka se všemi podporovanými typy s možnou hodnotou NULL. Vstupní prvky jsou textová pole.
Popis	Testování vkládání, editace, smazání, zobrazení: • s prázdnými hodnotami, • s nevalidními hodnotami, • se správnými hodnotami.
Požadovaný výstup	Nesmí se projevit žádná chyba při zobrazení. Správné uložení v databázi, pokud jsou hodnoty validní.

Název testu	Testování základní operací s databází bez prázdných hodnot.
Popis	Stejný jako test Testování základní operací s databází ale s tabulkami, jejichž sloupce neumožňují uložit hodnotu NULL.

Tabulka 4: Testy pro vygenerovanou aplikaci s tabulkami bez cizích klíčů

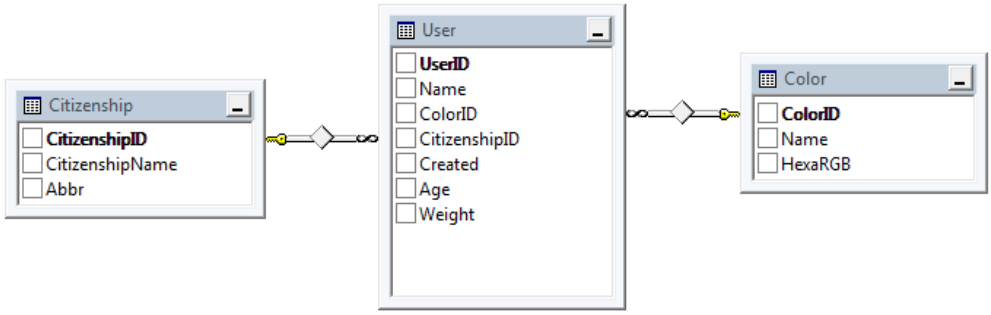
Název testu	Test cizího klíče na stejnou tabulku.
Vstup	Tabulka, která obsahuje cizí klíč, který odkazuje na tutéž tabulku.
Popis	Kontrola vkládání a editace a mazání položek z dané tabulky.
Požadovaný výstup	Aplikace nesmí skončit chybou. Hodnoty jsou správně uložené v databázi, pokud jsou validní. Vlastnosti objektů musí být správně zvolené názvy, aby se neshodovali s názvy tříd.

Název testu	Vkládání neexistujícího cizího klíče.
Vstup	Tabulky s cizími klíči, u kterých se kontroluje existence klíče.
Popis	Vkládání hodnoty cizího klíče takové, která neexistuje.
Požadovaný výstup	Výjimka kvůli neexistenci hodnot při vkládání. (Kontrola je na programátorovi nové aplikace.)

Tabulka 5: Testy pro vygenerovanou aplikaci s tabulkami s cizími klíči

6.1.1 Demo aplikace

Testovací aplikace byla vytvořena velmi jednoduchá. Posloužila především pro porovnání časové náročnosti mezi tvorbou webového portálu pomocí generátoru internetových aplikací a manuálním programování všech částí. Vytvoření webového portálu bez generátoru je časově náročné. Určitá třída bude u všech nových portálů stejná. Tím se myslí bázová třída datových objektů a třídy pro přístup k databázi, hlavní Controller v aplikační vrstvě a Master Pages v prezentační vrstvě. Ostatní



Obrázek 23: Struktura databáze demo aplikace

úkony jsou lineárně závislé na počtu tabulek a pak ještě lineárně závislé na počtu jejich sloupců. V následujících tabulkách jsou zaznamenány časy, jak dlouho by trvalo vytváření částí nového projektu. Časy jsou pouze orientační, protože nemá smysl vytvářet detailní analýzu. Pomocí generátoru je totiž projekt hotový během méně než deseti minut.

Operace	Čas
Vytvoření Solution, projektů a závislostí, konfigurace	5 min
Datová vrstva	
DBManager, DBManagerFactory	6 hod
DBObject	30 min
Aplikační vrstva	
MainController	5 min
Prezentační vrstva	
Master Pages, úvodní stránky	15 min
Zdroje dat (Resources)	5 min
Celkem (T_i)	*7 hod*

Tabulka 6: Délka vytváření nezávislých částí aplikace

Ze získaných časů lze snadno získat vztah, jehož výsledkem bude celkový čas vytváření webové aplikace závislý na počtu tabulek a průměrného počtu jejich sloupců.

T_i – celková doba vytváření nezávislých částí

T_t – celková doba vytváření částí závislých na počtu tabulek

T_c – celková doba vytváření částí závislých na počtu sloupců v tabulce

n – počet tabulek

C_n – počet sloupců v tabulce

T – celková doba vytváření aplikace

$$T = T_i + \sum_{1}^{n} (T_t + C_n \cdot T_c)$$

Vztah 1: Výpočet celkového času vytvoření aplikace

Následující tabulka zachycuje časy různých částí implementace přepočtených na tabulku, resp. sloupec.

Operace	Čas pro jednu tabulku	Čas pro jeden sloupec
Datová vrstva		
Datová třída	2 hod	30 min
Logická vrstva		
Controller datové třídy	30 min	10 min
Prezentační vrstva		
Vytvoření souhrnného zobrazení	1 hod	30 min
Zobrazení detailu	1 hod	15 min
Zobrazení souhrnu v admin. části	2 hod	45 min
Editační formulář datového objektu	2 hod	15 min
Doplnění zdrojů dat (Resources)	10 min	10 min
Celkem	T_t = *8 hod 40 min*	T_c = *2 hod 35 min*

Tabulka 7: Délka vytváření částí aplikace závislých na počtu tabulek a jejich sloupců

V tabulce *Přehled tabulek v testovací aplikaci* vidíte počet sloupců v testovací aplikaci a časy vytváření všech sloupců.

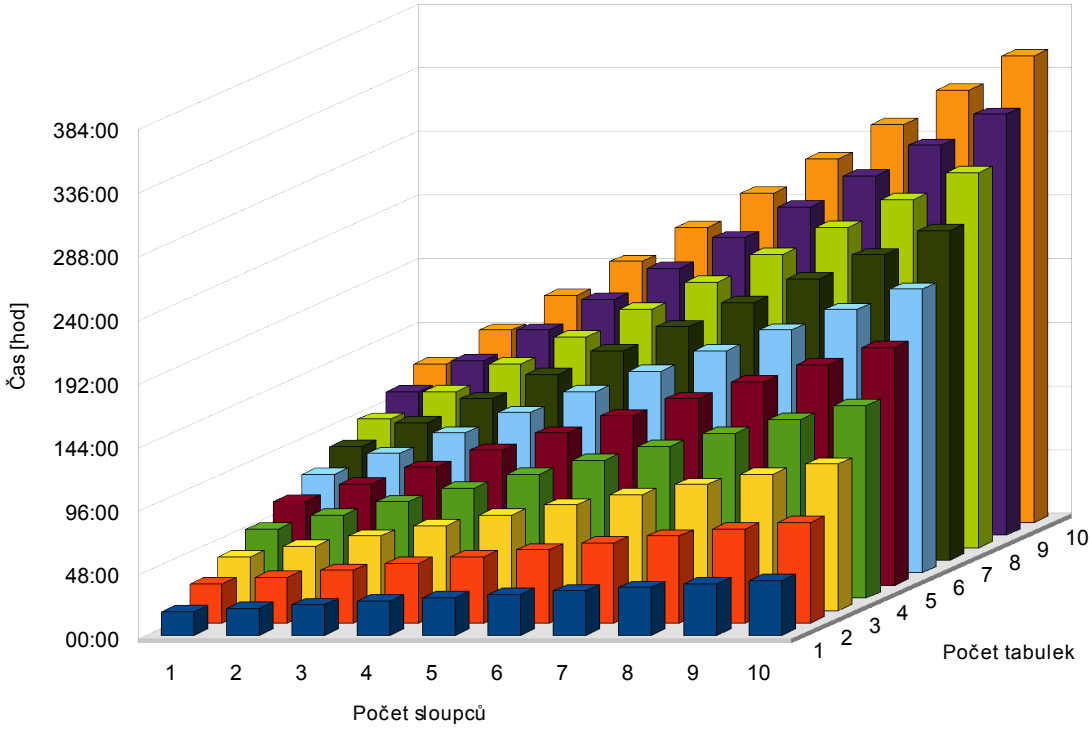
Tabulka	Počet sloupců	Čas pro sloupce v tabulce - $C_n T_c$
Citizen	3	7 hod 40 min
Product	4	10 hod 15 min
Color	3	7 hod 40 mi
User	8	20 hod 30 min
Celkem	18	36 hod 5 min

Tabulka 8: Přehled tabulek v testovací aplikaci

Po dosazení hodnot do vzorce lze spočítat, že taková aplikace bude trvat přibližně více než jeden týden práce i s testováním. Předpokladem je práce na projektu 8 hodin denně. Tento čas je subjektivní, protože záleží na schopnostech programátora, toto je s největší pravděpodobností čas nezkušeného programátora.

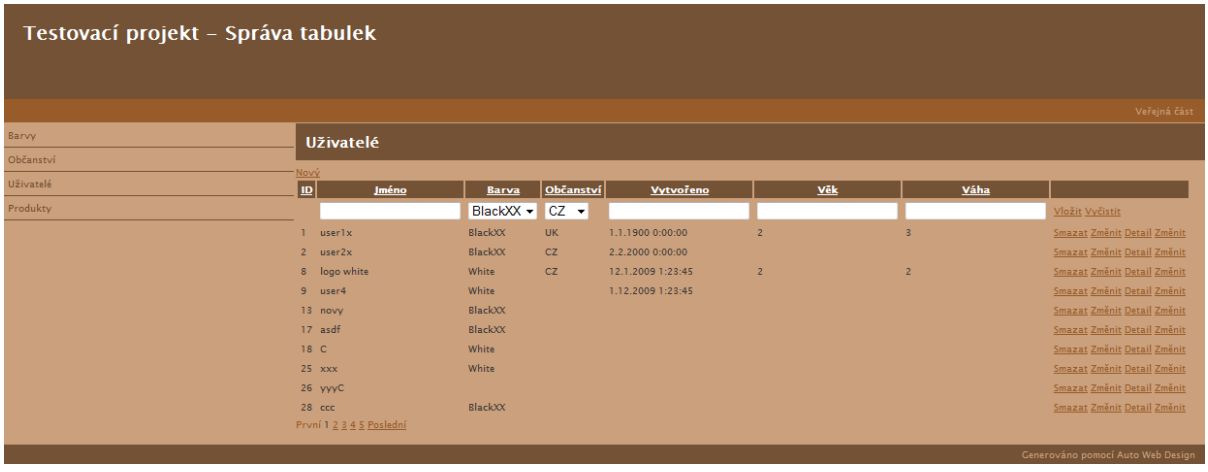
$$T \simeq 7 + 3 \cdot 8{,}6 + 36 \simeq 7 + 25{,}8 + 36 \simeq 58{,}8$$

Pro lepší znázornění můžete vidět výsledky v grafu (Obrázek 24). Zachycuje časy tvorby nové aplikace pro 1–10 tabulek s průměrným počtem sloupců 1–10.



Obrázek 24: Čas vytváření webové aplikace

Závěrem této kapitoly je, že generátor webových portálů, který je cílem toho projektu, ušetří čas desítek hodin a ve většině případů i dnů. Je bohužel zatím jenom v první verzi a je třeba ho vyladit a rozšířit o další funkce.



Obrázek 25: Uživatelské rozhraní vygenerované aplikace

6.1.2 Požadavky pro spuštění aplikací

Samotný generátor i generovaná aplikace mají téměř stejné požadavky pro běh systému. Vybavení nutné ke spuštění je dáno prostředím, ve kterém byly aplikace spouštěny a testovány. Testování bylo provedeno za následujících podmínek:

- Framework .NET v3.5 a vyšší,
- .NET Framework Data Provider for SQL Server ,
- MySQL 5.x,
- vzdálený nebo lokální Microsoft SQL Server 2008,
- vzdálený nebo lokální MySQL Server 5.0,
- MySQL Connector/ODBC 5.1,
- pro vývoj nové aplikace je potřeba použít Microsoft Visual Studio 2008.

Aplikace by měla umět i připojení ke starším typům databází, protože nepoužívá žádné speciální funkce a vymoženosti, které byly novinkami posledních verzí databázových systémů.

7 Závěr

Cílem tohoto diplomového projektu bylo prostudovat existující generátory webových aplikací. Na základě toho pak zanalyzovat a navrhnout vlastní generátor webových aplikací. Našel jsem na internetu hned několik podobných programů, které však nesplňovaly plnou kompatibilitu s vybraným vývojovým prostředím a tudíž vzniklý software, který je výsledkem toho projektu, je v tomto směru výjimečný. Je to i na úkor některých funkčních i nefunkčních požadavků, ale nelze tento generátor srovnávat s konkurenčními programy, které jsou již v pokročilejších verzích.

Velká výzva pro mne byla technologie ASP.NET, se kterou jsem doposud nepracoval déle než několik minut a v kombinaci s programovacím jazykem C# poskytuje skutečně velmi univerzální nástroj pro vytváření webových aplikací. Vzhledem k tomu, že je tento projekt mým prvním skutečným projektem v této kombinaci, jsou bohužel některé implementace ne zcela optimální. Rozsah projektu umožnil použití nejednoho návrhového vzoru. Proces implementace a testování se skládá z několika iterací, ve kterých se postupně přidávala nová funkcionalita.

Velkým ponaučením je použití knihoven třetích stran, u kterých není dostatečná dokumentace a v praxi by bylo nutné vědět, jakou funkcionalitu knihovny skutečně poskytují. V případě špatného předpokladu by se vytvořil nesprávný odhad (finanční i časový) a tím by se prodražil vývoj aplikace kvůli jejím změnám v průběhu implementace. V analýze je tedy nutné více zkoumat použité nástroje.

Již ve fázi analýzy jsem počítal s tím, že se projekt bude postupně rozšiřovat i v této první verzi. Pro generátor aplikací není ale vhodné vytvořit pouze minimální model generované aplikace a ten potom vylepšovat, ale také maximalistický prototyp, který obsahuje veškerou funkcionalitu s ohledem na praktické využití, aby bylo zřejmé, co je skutečně potřeba a kde nesmí generátor vytvářenou aplikaci omezovat.

V kapitole Testování a rozšíření jsem uvedl mnoho způsobů, jak na tento projekt mohou navázat další studenti a vylepšit ho pro další uživatele, aby tento generátor webových aplikací mohl konkurovat již existujícím programům. V kapitole Problémy při implementaci jsou zmíněny chyby, kterých jsem se dopustil nejen při programování ale už i v analýze. Na jejich základě by mohl být vytvořen nový nebo upravený návrh a podle něj by se pak mohla vytvořit aplikace s rozsáhlejší funkčností.

Literatura

[1] Java EE. *Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-03-29]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/J2EE>.

[2] ASP.NET. *Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-03-29]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/ASP.NET>.

[3] Oracle. *Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-03-29]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Oracle>.

[4] JAKEL, M. Sblížení s Microsoft SQL Server. *Interval.cz* [online]. 2002 [cit. 2009-03-27]. Dostupný z WWW: <http://interval.cz/clanky/sblizeni-s-microsoft-sql-server/>.

[5] MVC. *Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-03-29]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/MVC>.

[6] vzor. *Wikipedie, otevřená encyklopedie* [online]. 2009 [cit. 2009-03-29]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/N%C3%A1vrhov%C3%BD_vzor>.

[7] DVOŘÁK, M. Návrhové vzory (design paterns). *Objektová analýza, návrh a programování* [online]. 2003 [cit. 2009-03-27]. Dostupný z WWW: <http://objekty.vse.cz/Objekty/Vzory>. UML Applied: Object Oriented Analysis and Design. *Ariadne Training* [online]. 2005, edition 2 [cit. 2009-03-27].

[8] .NET Assemblies: pohled pod pokličku. *VBNET.CZ* [online]. 2007, roč. 2007 [cit. 2009-03-27]. Dostupný z WWW: <http://www.vbnet.cz/clanek--58-net_assemblies_pohled_pod_poklicku.aspx>.

[9] OMG Unified Modeling Language. *The Object Management Group* [online]. 2007 [cit. 2009-03-27]. Dostupný z WWW: <http://www.omg.org/docs/formal/07-11-02.pdf>.

[10] MySQL. 2007. MySQL 5.0 Reference Manual. MySQL Documentation. [Online] 2007. http://dev.mysql.com/doc/refman/5.0/en/select.html.

[11] KAČMÁŘ, D. *Programujeme .NET aplikace ve Visual Studiu .NET*. Computer Press, Praha 2001. ISBN 80-7226-569-5.

[12] PAGE-JONES, M. *Základy objektově orientovaného návrhu v UML*. [s.l.] : Grada Publishing, Praha, 2001. 368 s. ISBN 80-247-0210-X.

[13] PROSISE, Jeff. *Programování v Microsoft .NET : Webové aplikace v C#, ASP.NET a .NET Framework*. [s.l.] : Computer Press, Praha, 2003. 736 s. ISBN 80-7226-879-1.

[14] Schmuller, J. *Myslíme v jazyku UML – knihovna programátora.* [s.l.] : Grada Publishing, Praha, 2001. 360 s. ISBN 80-247-0029-8.

Seznam příloh

Použitý software

Microsoft Visual Studio 2008

Microsoft SQL Server 2005

MySQL Server 5.1

MySQL Connector NET 5.0.9

PSPad editor

OpenOffice.org