

Vývoj webové aplikace v ASP.NET

A development of application in ASP.NET

Radim Kolečkář

Bakalářská práce

2005



Univerzita Tomáše Bati ve Zlíně
Fakulta technologická

Univerzita Tomáše Bati ve Zlíně
Fakulta technologická
Institut řízení procesů a aplikované informatiky
akademický rok: 2004/2005

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Radim KOLEČKÁŘ**
Studijní program: **B 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Zpracování interaktivní databázové aplikace, která bude sloužit jakožto monografický katalog děl hudebního skladatele Vincenta Maška.**

Zásady pro vypracování:

Na základě předložené analýzy je třeba sestavit databázovou aplikaci, která bude sloužit jednak pro zadávání dat do katalogu a jednak jako interaktivní katalog, kde bude možné vyhledávat, třídit a procházet jednotlivé vložené záznamy. Výsledná aplikace musí být schopná pracovat mimo jiné s hudebními ukázkami ve formátu MIDI, které je třeba v rámci aplikace přehrávat. Pro formát MIDI (poměrně komplikovaný binární standardizovaný formát) je třeba sestavit algoritmus, který se pokusí o převedení MIDI ukázek na dva nejpoužívanější standardy pro kódování hudebního zápisu (RISM a SHK). Tyto číselné kódy se následně používají pro třídění a vyhledávání v notovém materiálu. Výsledná aplikace bude distribuována na CD jako interaktivní digitální katalog.

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Vieira, Robert: SQL Server 2000 Programujeme profesionálně

Liberty, Jesse: Programming ASP.NET

Avery, James: Microsoft ASP.NET Konfigurace a nastavení

Viescas, John: Mistrovství v Microsoft Access 2000

Huber, Davis Miles: The Midi Manual

www.asp.net

Vedoucí bakalářské práce:

Mgr. Martin Kuchař

Institut řízení procesů a aplikované informatiky

Datum zadání bakalářské práce:

21. února 2005

Termín odevzdání bakalářské práce:

10. června 2005

Ve Zlíně dne 17. února 2005



prof. Ing. Josef Šimoník, CSc.
děkan



prof. Ing. Vladimír Vašek, CSc.
ředitel institutu

ABSTRAKT

Abstrakt česky

Tato bakalářská práce se zabývá realizací projektu webového katalogu hudebních děl Vinzenze Maschka. Práce v teoretickém oddílu popisuje jednotlivé části v projektu použité třívrstvé architektury, metody návrhu a vývoje webové aplikace a zmiňuje evoluci užívaných technologií s důrazem na nejmodernější prostředí ASP.NET, které bylo přímo pro webové aplikace vyvinuto. V závěru teoretického oddílu práce seznamuje čtenáře se standardem MIDI, jeho strukturou a strukturou alfanumerických kódů SHK a RISM A II používaných pro přepis notového záznamu. Praktický oddíl se zaměřuje na popis vzniku aplikace s ukázkou nejdůležitějších částí a s popisem algoritmu, který byl v rámci práce sestaven pro převod MIDI souboru do zmíněných hudebních kódů.

Abstrakt ve světovém jazyce

This bachelor thesis deals with the realization of the project of a web catalogue containing musical pieces of Vinzenz Maschek. The thesis, in its theoretical part, describes particular parts of three-layer architecture used in the project, methods of the proposal and the process of web application. It also mentions the evolution of used technologies underlining the most modern environment ASP.NET, which has been created especially for web applications. At the end of the theoretical section, the dissertation acquaints the reader with the MIDI standard, its structure and the structure of alphanumeric codes SHK and RISM A II used for the transcription of the notation. Practical section aims at the description of creation of the application, gives examples of the most important parts and describes the algorithm that has been created within the dissertation for the transfer of MIDI files into the mentioned musical codes.

Diese Diplomarbeit verlegt sich auf eine Realisation des Webkatalog-Projekts der Musikwerke von Vinzenz Maschek. Die Arbeit beschreibt im theoretischen Abschnitt einzelne Teile dreischichtiger Architektur, die im Projekt benutzt wird, weiter Methoden des Konzepts und die Entwicklung der Webapplikation und erwähnt die Evolution benutzter Technologien mit Nachdruck auf modernste Umgebung ASP.NET, die gerade für Webapplikationen entwickelt war. Am Ende des theoretischen Abschnitts die Arbeit informiert über den MIDI-Standard, seine Struktur und die Struktur der alphanumerischen Codes SHK und RISM A II, die für Transkription der Notierung gebraucht werden. Praktischer Abschnitt konzentriert sich auf die Beschreibung der Applikationentstehung. Es gibt hier auch der Muster der wichtigsten Teile und die Beschreibung des Algorithmus, der im Rahmen der Arbeit für Transkription der MIDI-Datei in die Musikkodes konstruiert war.

Poděkování:

Děkuji za užitečné rady v oblasti vývoje webových aplikací a jejich implementaci vedoucímu bakalářské práce, panu Mgr. Martinu Kuchařovi.

Prohlašuji, že jsem na celé bakalářské práci pracoval samostatně a použitou literaturu jsem citoval.

Ve Zlíně, 30.8.2005

.....

jméno bakaláře

OBSAH

OBSAH	7
ÚVOD	10
I. TEORETICKÁ ČÁST	12
1 WEBOVÁ APLIKACE	13
1.1 CO JE TO WEBOVÁ APLIKACE?	13
1.2 VÝHODY A NEVÝHODY WEBOVÉ APLIKACE	13
1.2.1 Výhody	13
1.2.2 Nevýhody	14
2 DATOVÁ VRSTVA	15
2.1 IMPLEMENTACE DATOVÉ VRSTVY	16
2.1.1 Co je to SQL?	16
2.1.2 Historie SQL	16
2.1.3 SQL servery	17
3 APLIKAČNÍ VRSTVA	18
3.1 CGI SKRIPTY	18
3.1.1 Co je to CGI skript	18
3.1.2 Jak funguje CGI skript	18
3.1.3 Výhody CGI skriptu	19
3.1.4 Nevýhody CGI skriptu	19
3.2 SERVLETY	19
3.2.1 Co jsou to servlety?	19
3.2.2 Jak servlety fungují?	20
3.2.3 Výhody servletů oproti CGI	20
3.3 PHP	21
3.3.1 Co je to PHP a jeho historie	21
3.3.2 Jak funguje PHP	21
3.3.3 Výhody PHP	22
3.3.4 Nevýhody PHP	22
3.4 JSP	23
3.4.1 Co je to JSP?	23
3.4.2 Jak funguje JSP?	23
3.4.3 Výhody JSP	23
3.5 ASP	24
3.5.1 Proč vzniklo ASP?	24
3.5.2 Co je to ASP a jak funguje?	24
3.6 ASP.NET A JEHO VÝVOJ	25
3.6.1 Co je to .NET?	25
3.6.2 .NET FRAMEWORK	25
3.6.3 Přejít od ASP k ASP.NET	26
3.6.4 Proč zvolit ASP.NET?	26
4 PREZENTAČNÍ VRSTVA	28

5	PROCES VÝVOJE WEBOVÉ APLIKACE.....	29
5.1	SBĚR A SPECIFIKACE POŽADAVKŮ.....	29
5.2	ANALÝZA POŽADAVKŮ	31
5.2.1	Skupiny uživatelů.....	31
5.2.2	UseCase diagramy	31
5.2.3	Datový slovník.....	32
5.2.4	Mapa webu.....	32
5.2.5	Základní fragmenty grafického designu aplikace.....	32
5.2.6	Ověřovací testy a ostatní požadavky	33
5.3	NÁVRH DATOVÉ STRUKTURY	34
5.3.1	Vstupy a výstupy pro návrh datové struktury	35
5.4	NÁVRH HYPERTEXTOVÉHO MODELU	35
5.4.1	Hrubý návrh.....	35
5.4.2	Detailní návrh	36
5.5	IMPLEMENTACE A TESTOVÁNÍ.....	37
6	OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ A NÁSTROJE PRO NÁVRH OOP APLIKACÍ.....	38
6.1	CO JE TO VLASTNĚ OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ ?	38
6.2	FRAMEWORK.....	39
6.2.1	Co je to FRAMEWORK?	39
6.2.2	K čemu slouží FRAMEWORK?.....	39
6.2.3	Struktura FRAMEWORKU	40
6.3	RELAČNÍ MODEL DATABÁZE	40
6.3.1	Historie.....	40
6.3.2	K čemu slouží relační model?	40
6.3.3	ER diagram.....	41
6.4	UML – UNIFIED MODELING LANGUAGE	41
6.4.1	Co je to UML?	41
6.4.2	Historie UML	42
6.4.3	Pohled na systém	42
6.4.4	Notace pro případy užití	44
7	MIDI.....	46
7.1	CO JE TO MIDI.....	46
7.2	PRINCIP MIDI.....	46
7.3	STRUKTURA MIDI SOUBORU (STANDARD MIDI FORMAT).....	47
7.3.1	Hlavička MIDI.....	47
7.3.2	Události MIDI a Deltačas	49
7.3.3	Meta – události.....	50
7.4	PŘÍKLAD SOUBORU MIDI.....	52
8	SHK A RISM AII.....	54
8.1	SHK.....	54

8.1.1	Popis kódu.....	54
8.2	RISM AII	56
8.2.1	Popis kódu.....	57
II.	PRAKTICKÁ ČÁST.....	60
9	NÁVRH APLIKACE	61
9.1	ZVOLENÍ TYPU APLIKACE	61
9.2	ZVOLENÍ VÝVOJOVÝCH PROSTŘEDKŮ	61
9.3	KONFIGURACE POČÍTAČE NA KTERÉM MÁ BÝT APLIKACE SPUŠTĚNA	62
9.3.1	Minimální hardwarové požadavky (server)	62
9.3.2	Minimální hardwarové požadavky (klient)	62
9.3.3	Softwarové požadavky (server).....	62
9.3.4	Softwarové požadavky (klient)	63
10	USE CASE DIAGRAM APLIKACE	64
11	SEKVENČNÍ DIAGRAM APLIKACE	66
12	ER DIAGRAMY DATABÁZE APLIKACE.....	68
13	FRAMEWORK APLIKACE	80
14	NEJDŮLEŽITĚJŠÍ ČÁSTI KÓDU APLIKACE	81
15	ALGORITMUS PŘEVODU MIDI SOUBORU NA KÓDY SHK A RISM AII.....	87
15.1	POPIS PROBLÉMU	87
15.2	GENEROVÁNÍ KÓDU	87
15.3	UKÁZKY Z TŘÍDY PRO GENEROVÁNÍ – IMPLEMENTACE V C#	87
15.3.1	Zjištění délky noty – deltačas	87
15.3.2	Nalezení Meta Eventu.....	88
15.3.3	Určení předznamenaní incipitu	88
15.3.4	Určení taktu incipitu	89
16	POPIS APLIKACE A JEJÍHO OVLÁDÁNÍ	90
16.1	POPIS APLIKACE	90
16.2	PŘIHLÁŠENÍ DO APLIKACE	90
16.3	VYTVOŘENÍ/EDITACE ZÁZNAMU	92
	ZÁVĚR	95
	SEZNAM POUŽITÉ LITERATURY.....	96
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	98
	SEZNAM OBRÁZKŮ.....	99
	SEZNAM TABULEK	100
	SEZNAM PŘÍLOH.....	101

ÚVOD

Při výběru tématu bakalářské jsem měl zájem o zvolení problematiky, která by nabízela praktickou využitelnost a byla zároveň spojena s moderním vývojovým prostředím. Jakmile se taková možnost naskytla v podobě vývoje webové aplikace sloužící ke katalogizaci hudebního materiálu, neváhal jsem a toto téma zvolil. Tématický katalog děl Vinzenze Maschka je grantovým projektem Ministerstva kultury ČR (dotace poskytnutá odborem umění a knihoven Ministerstva kultury ČR na projekt z oblasti vědy a výzkumu, název projektu Vinzenz Maschek, ID: DA01P01OUK009, řešitel Jiří Mikuláš, České muzeum hudby). Úkolem zmíněného grantového projektu je monografické zpracování děl význačného českého hudebního skladatele, který byl Mozartovým současníkem. Webová katalogizační aplikace vyvinutá v rámci této bakalářské práce slouží pro uspořádání informací z grantového projektu, jejich prezentaci široké veřejnosti prostřednictvím internetové sítě a v neposlední řadě také pro vyhledávání, které díky počítačovému zpracování ušetří badatelům spoustu neefektivní práce na sběru a třídění materiálu.

Vzhledem k tomu, že projekt má všechny parametry vývoje klasického softwarového produktu, bylo třeba věnovat značnou pozornost návrhu a tento neustále porovnávat s uživatelskými požadavky. Ve fázi návrhu bylo používáno moderních návrhových metod pro objektově orientované programování (UML). Osvědčilo se také použití „testovací aplikace“, která jednak umožnila začít před dokončením aplikace finální se zadáváním dat a navíc si při jejím použití uživatelé dodefinovaly poměrně přesně požadavky, které pak mohly být hned od počátku zahrnuty do návrhu aplikace konečné. Na vývoj „testovací aplikace“ bylo použito prostředí ADP, které je součástí Microsoft Access. Konečná aplikace je pak vyvinuta v prostředí ASP.NET 2.0 s psaným kódem v C#, postaveném na verzi .NET FRAMEWORK 2.0. Pro databázovou část byl použit MS SQL SERVER 2000.

Co se samotného vývoje týče, tak nejnáročnější částí bylo naprogramování aplikačního frameworku. Vzhledem k tomu, že se dá předpokládat nárůst katalogizačních projektů z této oblasti, bylo naší snahou vyvinout v rámci práce univerzální framework použitelný pro případné další webové aplikace katalogizačního charakteru. Problematický se ukázal také automatický převod struktury MIDI souboru na alfanumerické hudební kódy SHK a RISM AII, který je popsán v samostatné kapitole.

Práce je ukázkou interdisciplinární spolupráce mezi tak vzdálenými obory, jako je muzikologie na straně jedné a informační technologie na straně druhé. Vzhledem k celkovému vývoji moderních technologií se zdá, že potřeba takovéto spolupráce bude i nadále narůstat.

I. TEORETICKÁ ČÁST

1 WEBOVÁ APLIKACE

1.1 Co je to webová aplikace?

Webová aplikace má několik zvláštností, které ji od klasických aplikací odlišují. Pokusím se tyto charakteristiky stručně popsat.

Webové aplikace se vyznačují především způsobem komunikace s uživatelem, která probíhá přes webový prohlížeč. Uživatel může být prakticky kdekoli na světě a pokud má přístup k internetu a zařízení na němž se může připojit, tak splňuje veškeré podmínky pro použití webové aplikace. S velkým rozmachem internetu, který dnes svět zažívá, se tak námi vytvořená webová aplikace může stát přístupná velkému počtu uživatelů.

Další charakteristickou vlastností webových aplikací je jejich vnitřní stavba, která vychází z modelu klient/server. Dnešní moderní webové aplikace jsou často postaveny na tzv. třívrstvé architektuře. Tato architektura má, jak napovídá její název, tři vrstvy a to je:

- Vrstva databázová
- Vrstva aplikační
- Vrstva prezentační

Jednotlivým vrstvám se ještě budeme věnovat podrobněji.

1.2 Výhody a nevýhody webové aplikace

1.2.1 Výhody

- Široká dostupnost webové aplikace
- Možnost pružně reagovat na potřeby zákazníka
- Centralizace dat na jednom serveru
- Jednoduchá správa aplikace

1.2.2 Nevýhody

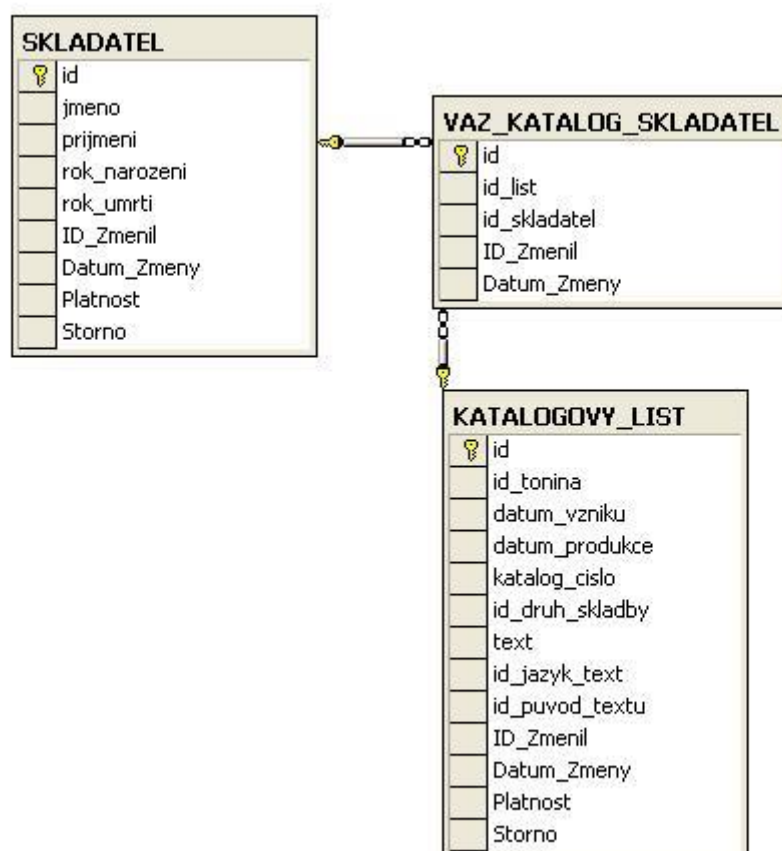
- Bezpečnost – hackerské útoky, únik a ztráta dat atd.
- Závislost na výpadcích a kvalitě sítě
- Závislost na rychlosti připojení k internetu

2 DATOVÁ VRSTVA

Na nejnižší úrovni systému se nachází datová vrstva, která má na starosti fyzické uložení informací ze systému a jejich vztahů v systému. V praxi stále nejpoužívanější databázovou technologií je relační databáze. V těchto databázích jsou data uložena jako řádky v tabulkách (entitách), přičemž sloupce představují atributy těchto entit.

Vazby mezi entitami jsou realizovány spojením přes hodnotu mezi identifikátorem (primárním klíčem) jedné entity a identifikátorem (cizím klíčem) navázané entity. Tímto způsobem je možné modelovat vazby do kardinality 1:n.

Pro umožnění obecnější vazby m:n se pak zavádí speciální vazební entita, která je navázána mezi dvěma jinými entitami. Tento databázový model (entit a vazeb s jejich kardinalitami) byl využit i v této implementaci, jeho konkrétní fyzickou podobu přehledně modeluje následující ER diagram. Tento diagram je jen ukázka, podrobnou strukturou databáze aplikace se bude zabývat kapitola Popis nejdůležitějších objektů v aplikaci.



Obr. 1 Příklad ER - diagramu

2.1 Implementace datové vrstvy

2.1.1 Co je to SQL?

SQL je anglická zkratka pro Structured Query Language, což je programovací jazyk, jímž se pomocí klienta ptáme SQL serveru a ten následně při správně položeném dotazu odpoví. Odpověď je formulována většinou do tabulky o jednom, více či dokonce žádném řádku (to pokud dotazu nevyhoví ani jedna položka). Z tabulky pak můžeme pomocí skriptů získat potřebná data a dále je používat. [1]

2.1.2 Historie SQL

S jazykem **SQL** se můžeme setkat již v roce 1974. V té době se však jmenoval **Sequel**. Jako první byl použit v **Systému R** vyvinutého v kalifornské laboratoři IBM. Od té doby se jazyk rozšířil a byl používán v ostatních systémech.

Postupem času však vznikaly další „verze“ jazyka a bylo potřeba, aby byl standardizován. K tomu došlo v roce 1986, kdy jej přijala standardizační skupina **ANSI** (a v roce 1987 **ISO**). Standardem byl uznán „dialekt“ firmy IBM. V literatuře se můžeme setkat také s označením **SQL86**. Později bylo potřeba rozšířit definičního jazyka pro možnost integritního omezení. Výsledná zpráva byla zveřejněna v roce 1989 organizací ISO. Tomuto rozšíření se říká **SQL89**. Poslední přijatý standard byl v roce 1992 (ANSI) a je označován jako **SQL92**.

První významnější produkty, kde byl jazyk SQL zabudován, byly od IBM: **DB2**, **SQL/DS** (oba vycházejí ze Systému R) a **QMF**. S nástupem počítačů třídy PC začaly i ostatní výrobci více uvažovat o SQL. Asi nejznámější je firma Ashton-Tate se svou **dBASE IV** (i když implementace SQL nebyla úplná). Jako další jmenujme **INFORMIX-SQL**, **INGRES** pro PC, **ORACLE**, **SQLBase**, **XDB II**, **XQL**. Postupem času ze začal jazyk SQL přenášet i na systém **UNIX** a od jedinouživatelských úloh se přecházelo k serverům založených na SQL. Jmenujme aspoň **SŘBD ORACLE**, **INFORMIX**, **PROGRESS** a **INGRES**. Dále např. **SQL Server** od firmy Sybase či Microsoft.

Mezi standardy patří i **X/OPEN SQL** vycházející z ANSI, ale je přihlédnuto k některým rysům systému UNIX. Ačkoliv jako standard byl uznán dialekt firmy IBM, ustanovila si svůj

vlastní standard pod názvem **SAA-SQL**. SQL je také částí federálního standardu pro zpracování informací (**FIPS**). [2]

2.1.3 SQL servery

Databázi si můžeme představit jako místo, kam se ukládají všechny potřebné údaje. Přístup k údajům uloženým v databázi obstarává program, kterému se říká *SŘBD -- Systém Řízení Báze Dat*. [3] Tento poněkud krkolomný název vznikl přeložením původního anglického termínu DBMS -- DataBase Management System, více se však používá název **SQL server**.

SQL serverů existuje celá řada. Jmenujme alespoň Informix a Oracle pro high-end prostředí s velkými nároky na výkon, PostgreSQL a MySQL z řady volně šiřitelných a přesto výkonných, a MiniSQL z řady low-end serverů. Na platformě Linuxu si v poslední době probíjí své místo na slunci MySQL server, zvláště proto, že implementuje spoustu funkcí "velkých" serverů s dostatečnou rychlostí. Jedinou jeho nevýhodou je to, že nepodporuje transakce (transakce je jeden či více SQL příkazů provedených jedním uživatelem, které končí úspěšnou změnou v databázi nebo uvedením zpět do výchozího stavu, přičemž pro ostatní uživatele musí být tato změna patrná až po úspěšném vykonání celého sledu instrukcí). Na platformě Windows je pro své vlastnosti velmi často využíván **Microsoft SQL Server 2000**, který jsme také použili při vývoji této bakalářské práce.

3 APLIKAČNÍ VRSTVA

Na prostřední úrovni se nachází aplikační vrstva. Jejím úkolem je přijímat požadavky od vrstvy uživatelského rozhraní a ve spolupráci s databázovou vrstvou získat a zpracovat potřebné informace z databáze a výsledek předat zpět vrstvě uživatelského rozhraní. Popíšeme si nejběžnější možnosti implementace aplikační vrstvy ve webových systémech.

3.1 CGI skripty

3.1.1 Co je to CGI skript

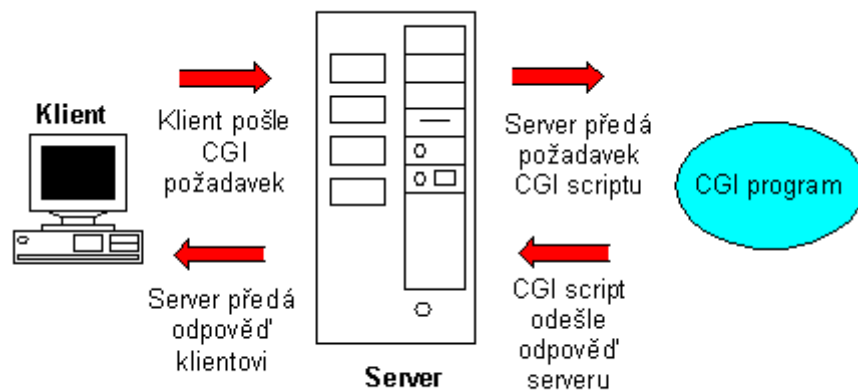
CGI skript je libovolný skript nebo program. Jedinou podmínkou je jeho schopnost komunikovat s WWW serverem. Program musí umět data přebírat od WWW serveru a musí umět předat serveru zpět výsledek. WWW server musí dokázat tento program nebo skript spustit. Jako CGI může pracovat program tj. spustitelný soubor vzniklý překladem zdrojového textu napsaného např. v jazyce C nebo skript (procedura - tj. sekvence příkazů) interpretovaný např. programem Perl, Shellem v Unixu apod.

CGI skripty jsou často vytvářeny ve skriptovacím jazyce Perl nebo v některém shellu OS Unix. Z programovacích jazyků se používá např. jazyk C, C++, Visual Basic. Výběr jazyka závisí pouze na autorovi CGI skriptu.

Pravidla komunikace (rozhraní) mezi CGI skriptem a WWW serverem definuje norma CGI - Common Gateway Interface dostupná na <http://hoohoo.ncsa.uiuc.edu/cgi>. Díky existenci normy pro CGI rozhraní je možné používat CGI skripty ve spojení s většinou existujících serverů a způsob použití je ve všech případech stejný. [4]

3.1.2 Jak funguje CGI skript

CGI skripty přebírají data zadaná uživatelem, zpracovávají je a jako výsledek vytvářejí většinou html stránky. Tyto dynamicky vytvořené stránky pak WWW server posílá zpět klientovi. [4] Pomocí CGI může server provádět výpočty, vyhledávat data v databázi nebo např. odesílat objednávky. Výsledkem skriptu může být nejen html stránka vytvořená podle požadavku uživatele, ale i obrázek v gif formátu, nebo jiná data, která je možno zobrazit prohlížečem.



Obr. 2. Komunikace mezi klientem, serverem a CGI skriptem

3.1.3 Výhody CGI skriptu

- Lehká práce s textem (nahrazování)
- Jednoduchý zápis do souborů
- Malý kód

3.1.4 Nevýhody CGI skriptu

- Malá podpora databází
- Nepřehledný kód

3.2 Servlety

3.2.1 Co jsou to servlety?

Jedná se o programy, napsané v jazyce Java. Tyto programy se používají ke generování dynamických webových stránek, k flexibilnější interakci s uživatelem a k provádění různých

akcí na straně serveru. Dynamické stránky se zpravidla používají, pokud je jejich obsah založen na datech od uživatele nebo je odvozen z dat uložených v databázích.

3.2.2 Jak servlety fungují?

Instance servletu je aplikace běžící na serveru, která zprostředkovává komunikaci mezi webovým prohlížečem či jiným http klientem a serverem. Tato aplikace může dále například obsluhovat požadavky klienta na práci s databází. Servlety mohou provádět tyto akce:

- Přečíst data odeslaná uživatelem (z HTML formuláře, Java appletu...)
- Vyhledat další informace o požadavku (podrobnosti o prohlížeči, cookies,...)
- Vytvořit výsledky (výpočet odezvy, komunikace s databází, ...)
- Zformátovat výsledky (tvorba HTML výstupu)
- Nastavit parametry odezvy http (typ dokumentu, cookies,...)
- Odeslat dokument zpět klientovi

3.2.3 Výhody servletů oproti CGI

V CGI se pro každý požadavek spouští nový proces. Tedy pro každý požadavek se kód CGI zavede do paměti znovu. S tím spojená režie může být někdy vyšší než u samotného běhu programu. U servletů stále běží virtuální stroj Java, který jednotlivé požadavky obsluhuje vlákny. V paměti je pouze jediná instance daného servletu. U CGI se po ukončení obsluhy požadavku ukončí i program. Tím se ztratí i vše, co bylo vypočítáno, což je další nevýhodou, pokud např. chceme ukládat data, která budou dostupná i později bez toho, abychom je znovu počítali. Oproti tomu servlety zůstávají v paměti dokonce i po dokončení odezvy. Servlety jsou přenositelnější. Díky tomu, že jsou napsány v Javě, mohou běžet bez jakýchkoliv změn v kódu na libovolné platformě a libovolném webovém serveru, který servlety podporuje.

3.3 PHP

3.3.1 Co je to PHP a jeho historie

PHP je skriptovací jazyk pro tvorbu dynamického webu a jeho počátky spadají do roku 1994. Tehdy se pan Rasmus Lerdorf rozhodl vytvořit jednoduchý systém pro počítání přístupu ke svým stránkám; bylo to napsáno v PERLu. Za nějakou dobu byl systém přepsán do jazyka C, protože perlůvský kód nepřiměřeně zatěžoval server. Sada těchto skriptů byla ještě později téhož roku vydána pod názvem "Personal Home Page Tools", zkráceně PHP. Později měl název "Personal Home Page Construction Kit".

U toho však nezůstalo. V polovině roku 1995 se systém PHP spojil s jiným programem stejného autora, a to sice s nástrojem "Form Interpreter" neboli zkráceně FI. Tak vzniklo PHP/FI 2.0, systém, který si postupně získal celosvětovou proslulost a byl velmi rozšířen.

Koncem roku 1998 byla již k dispozici verze PHP 3.0, která byla mnohem rychlejší (a vybavenější) než "dvojka" a která byla k dispozici rovněž pod operačními systémy Windows. Počet webů které používaly PHP se zvyšoval, až dosáhl cca 150 000. "Čtyřkové" verze PHP, které jsou k dispozici nyní, přidávají do jazyka mnoho nových funkcí a rovněž přinášejí přepracované a tudíž podstatně rychlejší jádro Zend, další vývoj dopředu přineslo PHP 5.0. [5]

3.3.2 Jak funguje PHP

PHP je technologie běžící na serveru. Typický PHP skript obsahuje jednak kusy normálního HTML kódu, a jednak kusy programového kódu. Když webový server obdrží požadavek na zpracování takového skriptu, vezme:

- Kusy HTML kódu tak, jak jsou
- Části PHP programového kódu provede
- Výsledek zkombinuje a odešle prohlížeči

Tato filozofie fungování je nesmírně účinná. Server totiž může provést jednu nebo dokonce několik operací a výsledek poslat do prohlížeče jako obyčejnou HTML stránku.

3.3.3 Výhody PHP

- PHP je relativně jednoduché na pochopení
- PHP má syntaxi velmi podobnou jazyku C a je tedy většině vývojářů dost blízký
- PHP podporuje širokou řadu souvisejících technologií, formátů a standardů
- Je to otevřený projekt s rozsáhlou podporou komunity
- Dají se najít kvanta již hotového kódu k okamžitému použití nebo funkční PHP aplikace. Podstatná část z hotového kódu je šířena pod svobodnou licencí a dá se použít ve vlastních projektech
- PHP si dobře rozumí s webovým serverem Apache
- PHP snadno komunikuje s databázemi, jako je MySQL, PostgreSQL a řada dalších
- PHP je multiplatformní a lze jej provozovat s většinou webových serverů a na většině dnes existujících operačních systémů
- PHP podporuje mnoho existujících poskytovatelů webhostingových služeb

3.3.4 Nevýhody PHP

- PHP je interpretovaný, ne kompilovaný jazyk
- Kdokoli má přímý přístup k serveru, může nahlédnout do vašich PHP skriptů
- Podpora objektového programování není v PHP na moc dobré úrovni. V PHP 5 by se to ale mělo zlepšit
- Protože je PHP aktivně vyvíjen, v budoucích verzích jazyka se mohou některé funkce změnit nebo se mohou chovat jinak než dosud

3.4 JSP

3.4.1 Co je to JSP?

Java Server Pages (JSP) nám umožňují směřovat statické HTML stránky s dynamicky vytvořeným obsahem. Často chceme mít převážnou část stránky statickou a doplňovat jen několik málo údajů z nějakých dynamických zdrojů. Pak je zbytečné používat pro generování celé stránky servlet. JSP umožňuje vytvářet jak statickou část, tak dynamickou v jednom programu. Tento program se pak skládá z normálního HTML kódu a z částí, které definují dynamiku stránky pomocí speciálních značek podobných HTML značkám.

3.4.2 Jak funguje JSP?

Z programu JSP se při prvním zavolání vytvoří servlet, který se zkompile a spustí. Při dalším volání se pokaždé volá již zkompileovaná verze, takže zpomalení nastává jen u prvního volání. Avšak při změně zdrojového kódu JSP se tvorba a překlad servletu uskuteční znovu. Proto je vhodné, aby programátor stránek využívajících JSP provedl sám první zavolání všech JSP, aby došlo k jejich přeložení do servletu a následné kompilaci.

3.4.3 Výhody JSP

- JSP má oproti konkurenční technologii ASP (Active Server Pages) od společnosti Microsoft dvě výhody. Za prvé, dynamická část stránek je psána v jazyce Java, který je přenositelný a programy v něm napsané lze opětovně použít jako komponenty jiných programů - ASP používá jazyk VBScript. Za druhé, JSP lze přenést i na jiné webové servery a operační systémy. Nemusí jít jen o IIS (Internet Information Server od firmy Microsoft) a Windows.
- Porovnáme-li JSP se servlety, tak ty mají tu nevýhodu, že i poměrně jednoduché konstrukce HTML kódu mohou po zapsání do kódu servletu vypadat složitě a je pak lehké v tomto kódu přehlédnout nějakou chybu. Je tedy přehlednější oddělit statickou část od dynamické, což JSP umožňují.

3.5 ASP

3.5.1 Proč vzniklo ASP?

Tak jak se Internet vyvíjel, rostly požadavky na internetové stránky. Stránky se stávaly více rozsáhlými, a postupně tak vznikala například potřeba fulltextového hledání, zabezpečení, monitorování pohybu na stránkách, počítačla a obecně statistiky serveru (stránek). ASP tak vzniklo jako reakce na UNIXovou platformu (CGI, Perl, PHP). Domovské prostředí ASP je Windows, zprvu to byla Windows NT, nyní můžeme ASP provozovat na platformách Windows 95/98/NT/2000/Me/XP Profi/2003/Vista. Jak jsme se již zmínili, jedná se o konkurenci UNIXu. Můžeme říci, že ASP, resp. server, na kterém běží ASP, je mnohem více náročný na výkon v porovnání s PHP serverem. Na druhou stranu ASP podporovalo například session (možnost monitorování pohybu uživatele na stránce) již na svém počátku, ale PHP jej umí až od verze 4.0. [6]

3.5.2 Co je to ASP a jak funguje?

ASP tedy *active server pages* je jakési prostředí, které nám umožňuje vkládat do běžné HTML stránky speciální tagy a pomocí nich tak vykonávat funkce, které by pomocí HTML nebyly jinak možné. Rozdíl mezi ASP a například JavaScriptem je ten, že ASP je *server-side script language*, tedy skript prováděný na serveru, zatímco JavaScript je *client-side script language* – skript spuštěný na vašem PC. V praxi to vypadá následovně: pokud si zobrazíte v prohlížeči stránku napsanou za použití ASP, nepoznáte vlastně nic. Bude se vám zdát jako běžná HTML stránka. Na klienta, tedy na vaše PC, putuje již spuštěný ASP soubor, nepřesně jej můžeme nazvat zkompilevaným souborem. Ten se vytváří na základě akce uživatele (např. je odeslán požadavek, že chceme vyhledat PSČ města). ASP skript vyhodnotí dotaz, v našem případě prohledá databázi PSČ, vygeneruje HTML stránku a tu nám zašle, zatímco JavaScript vykonává skripty přímo u uživatele na počítači. Bystrý čtenář jistě pozná první výhodu ASP – je *cross-browser compatible*, nezáleží tedy na prohlížeči. Z tohoto rozdílu je zcela patrné také použití Javascriptu, který používáme pouze například pro efekty, zpestření stránek atd. [6]

3.6 ASP.NET a jeho vývoj

3.6.1 Co je to .NET?

.NET je nová architektura, která by měla od základu změnit a hlavně zjednodušit vývoj software. Podle názvu se může zdát, že se jedná o architekturu určenou pro programování webových aplikací. To však není tak docela pravda, neboť pod platformou .NET lze vytvářet všechny druhy aplikací včetně klasických desktopových.

.NET je novou platformou a technologií, pod kterou lze spouštět produkty, u nichž nelze vždy zaručit kompatibilitu se současnými technologiemi. Pro vývoj těchto produktů nabízí podporu 27 programovacích jazyků, které sdílejí rozsáhlý soubor tříd poskytující základní služby. Ústředním jazykem .NET platformy se stal jazyk C#, který byl vytvořen spojením nejlepších vlastností z ostatních jazyků, hlavně z jazyků Java, C++ a Visual Basic.

Aplikace vytvořené pomocí .NET, na rozdíl od současných aplikací, již nejsou překládány přímo do kódu pro daný procesor, ale jsou překládány do mezijazyka Microsoft Intermediate Language (MSIL), který běží ve stroji nazývaném Common Language Runtime (CLR).

Hlavní a nepostradatelnou součástí architektury je její jádro .NET Framework, okolo kterého se vše točí. Podívejme se na něj tedy trochu blíže.

3.6.2 .NET FRAMEWORK

Při vytváření .NET Framework se jeho tvůrci snažili o následující cíle (uvedu jen ty nejdůležitější):

- vytvořit odpovídající objektově orientované prostředí, nezávislé na tom, zda je kód programu uložen a vykonáván na lokálním počítači, nebo zda je kód stahovaný z internetu je vykonáván na lokálním počítači, nebo zda je vykonáván vzdáleně
- vytvořit takové běhové prostředí, které minimalizuje rozmístění softwaru (tj. aby všechny části programu byly pokud možno na stejném místě) a nedocházelo ke konfliktům mezi jednotlivými verzemi programů
- vytvořit běhové prostředí, které zaručí bezpečné vykonání kódu
- přiblížit navzájem vývoj aplikací pro Windows a aplikací pro Web

.NET Framework se skládá ze dvou hlavních částí. První je tzv. common language runtime (CLR), ta má na starosti mnoho věcí, jako je správa paměti, vykonávání procesů, vykonávání kódu, kontrolu bezpečnosti kódu a mnoho dalších věcí. Druhou nedílnou součástí je .NET Framework class library. Jedná se o plně objektově orientovanou kolekci prvků, které mohou být použity při programování všech druhů aplikací od jednoduchých konzolových aplikací až po aplikace s grafickým uživatelským rozhraním známým z Windows, či aplikací vytvářených pomocí ASP.NET, jako jsou Webové služby apod.

3.6.3 Přechod od ASP k ASP.NET

Technologie ASP už dnes pro vývoje moderních webových stránek nestačí. Obsahuje několik základních objektů umožňujících základní operace, ale pokročilejší možnosti, mezi které patří například práce se SMTP či generování obrázků na straně serveru, musely být zprostředkovány pomocí cizích ActiveX prvků. Většina uživatelů byla tedy dána na milost a nemilost svého poskytovatele hostingu. ASP.NET je na tom jinak. Využívá úplně jiných systémů a bez váhání ho můžeme označit za první technologii, s kterou se dá na serveru *skutečně* programovat. [7]

3.6.4 Proč zvolit ASP.NET?

Je třeba si uvědomit, že ASP.NET pracuje úplně jinak než ASP. Klasické ASP vždy vykonávalo bloky od shora dolů a kód `<% %>` ve výkonných blocích byl posílán na výstup okamžitě po jeho nalezení překladačem ASP.

ASP.NET stránky jsou však kompilovány do IL kódu a později do nativního kódu Just-in-time kompilátory (zajišťuje CLR) a následovně vykonávány plnou rychlostí. [7]

Z výše uvedených informací je patrné, že nástupce ASP ASP.NET přináší výrazné novinky:

- Potenciální vyšší výkon. Jelikož kód není interpretován, nýbrž přeložen a následovně vykonáván plnou rychlostí.
- Jazykovou nezávislost. Pro vytváření kódu na aspx stránkách si můžeme zvolit kterýkoliv jiný jazyk na platformě NET.

- Podporu, správu stavu a bezpečnost aplikace.
- Konfiguraci ASP.NET.
- V neposlední řadě vyšší spolehlivost aplikací.

4 PREZENTAČNÍ VRSTVA

Hlavní výhodou oddělení aplikační a prezentační vrstvy je zejména ve snadnější údržbě a modifikaci celé aplikace. Pokud budeme chtít např. provést změny v prezentaci a vizualizaci dat klientovi, nemusíme zasahovat přímo do aplikace, ale stačí upravit prezentační vrstvu.

Další výhodou oddělení obou vrstev je známa jako princip tenkého klienta. Tenký klient je koncové zařízení, které buď ve formě softwaru nebo specializovaného jednoúčelového hardwaru slouží k zobrazování aplikace, která je instalována, zpřístupněná, spravovaná a vykonávána výhradně na centrálním serveru, takže kromě vstupu z myši, klávesnice a obrazovky klienta se žádná další data nepřenášejí.

Každý z připojených uživatelů má vlastní uživatelské prostředí. Server přijímá a zpracovává všechny stisky kláves a kliknutí myši od připojeného klienta a směruje výstup obrazovky k příslušnému klientovi. Klientská zařízení, tak mají okamžitý přístup k firemním aplikacím přes server, bez nutnosti instalace těchto aplikací a údržby na každém jednotlivém koncovém zařízení a to kdykoliv, kdekoliv, odkudkoliv a z jakéhokoliv koncového zařízení.

V prezentační vrstvě je tedy webová aplikace tvořena HTML kódem, zobrazovaným a interpretovaným klientem – webovým prohlížečem. Kód HTML stránek je dynamicky generován aplikační vrstvou, vygenerované stránky odeslané klientovi jsou již statické jen se sporadickým využitím aktivního skriptování na straně klienta pomocí jazyka JavaScript.

5 PROCES VÝVOJE WEBOVÉ APLIKACE

Běžný a osvědčený proces vývoje webové aplikace se skládá z následujících částí:

- sběr a specifikace požadavků
- analýza
- návrh
- implementace
- testování
- nasazení a údržba

5.1 Sběr a specifikace požadavků

V této fázi si musíme vytvořit základní představu o budované aplikaci. Zdrojem informací se mohou stát především rozhovory se zákazníkem. Velice užitečné pro nás mohou být již existující dokumenty nejrůznějšího typu.

Velké množství zákazníků již má alespoň částečnou představu o tom, co vlastně chce. Zkušenější společnosti připravují kvalitní specifikaci požadavků jako podklad pro výběrové řízení na dodavatele aplikace. Rozhodnutí o pořízení nové webové aplikace nebo prezentace nepřichází ve většině případů náhle, ale předchází mu určité kroky, například proces schvalování, zda společnost novou prezentaci nebo aplikaci potřebuje. Součástí těchto kroků bývá vytvoření základní funkční specifikace, která nám může velice pomoci. [8]

Typy informací a oblastí, kterými bychom se měli při sběru požadavků zabývat:

- **Funkční požadavky:** Tvoří základní stavební kámen specifikace. Jedná se o funkce, které musí systém realizovat.
- **Identifikace uživatelů:** Nezbytné je identifikovat jednotlivé typy uživatelů, kteří s naší aplikací budou pracovat a rozeznat skupiny uživatelů. Tento úkol bývá součástí tvorby UseCase diagramů.
- **Požadavky na personalizaci:** Musíme si uvědomit, zda obsah a služby aplikace budou pro všechny uživatele totožné, nebo je požadována personalizace aplikace (přístupová práva a podobně).

- **Požadavky na data:** Musíme si ujasnit, jaká data mají být aplikací zpracovávána a prezentována. Zde se jedná pouze o základní koncept. Podrobněji se datům věnujeme při vytváření datového modelu aplikace.
- **Ostatní požadavky:** Jedná se především o klasické charakteristiky jakosti informačních systémů, které můžeme nalézt v normě ISO/IEC 9126.
 - o *Použitelnost:* Srozumitelnost, snadná obsluhovatelnost a atraktivnost aplikace (prezentace).
 - o *Výkonnost:* Většinou stanovovaná jako počet požadavků respektive přístupů do aplikace za časovou jednotku (například aplikace musí obsloužit 100 požadavků za sekundu).
 - o *Dostupnost:* Většinou stanovená v procentech jako čas, kdy aplikace funguje bez problémů (často kolem 99 %).
 - o *Škálovatelnost:* Možnost zvyšovat výkon aplikace (například využitím techniky content switch).
 - o *Bezpečnost:* Zabýváme se ochranou osobních údajů, silou ověřovacích mechanismů, ukládáním hesel, šifrováním přenosu.
 - o *Rozšiřitelnost:* Možnost přidávat další funkce aplikace, úroveň modularity. Zde záleží především na architektuře a technologii, na které je aplikace postavena.

5.2 Analýza požadavků

Pokud jsme skončili se sběrem požadavků, musíme tyto často nepřehledné a v různých formách posbírané fragmenty nějakým způsobem sjednotit a formalizovat.

5.2.1 Skupiny uživatelů

Měli bychom identifikovat potenciální uživatele a roztrdit je do skupin. Pro každou skupinu bychom měli sepsat následující údaje:

- Jednoduchý slovní popis skupiny
- Informace, které chceme o uživatelích této skupiny ukládat
- Nadřízená a podřízená skupina, pokud existuje hierarchie mezi skupinami uživatelů
- Relevantní případy užití z UseCase diagramů
- Data, ke kterým bude mít uživatel přístup na čtení, a data, která bude moci uživatel dané skupiny spravovat

5.2.2 UseCase diagramy

UseCase diagramy jsou jedním z nejužitečnějších nástrojů pro úvodní analýzu softwarové aplikace. S jejich použitím se můžeme setkat především v UML, které je převzalo z metodologie OMT Jamese Rumbaughy.

Tvorba UseCase diagramů a specifikace jednotlivých skupin uživatelů se navzájem doplňují. Participant z UseCase modelu představuje skupinu uživatelů a každá skupina uživatelů má své relevantní případy užití (*UseCases*). Pro každý případ užití musíme specifikovat především jeho účel, dále vstupní a výstupní podmínky a průběh případu užití.

Jako klasický příklad případu užití můžeme označit přihlášení administrátora do systému. Vstupní podmínkou zde je existence registrace příslušného uživatele, výstupní podmínkou úspěšné přihlášení a načtení uživatelského prostředí relevantního pro danou skupinu uživatelů. V popisu průběhu tohoto UseCase můžeme například uvést, že je nutné, aby uživatel zadal přihlašovací údaje, které jsou potom ověřovány proti databázi, a že na základě skupiny je uživatel přesměrován na výchozí stránku administračního rozhraní. [9]

5.2.3 Datový slovník

Při vytváření datového slovníku musíme na základě sebraných požadavků identifikovat základní objekty, respektive entity, které v systému vystupují a které budeme chtít ukládat v databázi. Pro každý z takto identifikovaných objektů musíme specifikovat:

- jméno
- stručný popis
- základní a na první pohled zřejmé atributy
- zřejmé vztahy s jinými objekty
- příklady takového objektu

5.2.4 Mapa webu

Ve chvíli, kdy vytváříme mapu webu, musíme znát jednotlivé uživatelské skupiny, případy užití a datový slovník.

Výstupem je jedna či více map vytvářené aplikace nebo webu, dle počtu uživatelských skupin a požadavků na personalizaci aplikace.

Pro každou mapu bychom měli uvést:

- Jméno
- Stručný popis
- Cílové uživatelské skupiny, pro které je tento pohled relevantní
- Případy užití, které tato mapa implementuje
- Oblasti dané mapy, pokud je rozsáhlejší

5.2.5 Základní fragmenty grafického designu aplikace

Požadavky na grafický design jsou někdy stanoveny zákazníkem. Především u větších firem se musí brát ohled na *corporate identity*, kterou má daná společnost často již vytvořenou a jejíž součástí je provedení základních grafických prvků, barevná schémata, logo a další.

Jindy naopak začínáme od začátku, ale ve všech případech musí být shromážděny požadavky na následující fragmenty grafického uživatelského rozhraní:

- layout prezentace
- logo a umístění loga na webu
- menu, typ menu, umístění menu, úrovně zanoření menu
- barevné schéma
- fonty pro nadpisy a běžné texty
- styly zobrazování seznamů, tabulek a dalších objektů

Nedílnou součástí požadavků na grafický design je kompatibilita aplikace s nejpoužívanějšími webovými prohlížeči.

5.2.6 Ověřovací testy a ostatní požadavky

Ověřovací testy se týkají především požadavků, které nepatří mezi takzvané funkční požadavky. Při stanovení ověřovacích testů se jedná o přesné určení hodnot, kterých musí dosáhnout naše aplikace v jednotlivých kritériích a jednotlivých kategoriích požadavků. Dále musíme určit způsob, kterým bude splnění těchto požadavků ověřeno.

Zajímají nás následující údaje:

- **Výkonnost:** Musíme specifikovat, jakým způsobem budou probíhat výkonnostní testy a jaké hodnoty jsou od naší aplikace požadovány (například vystavení 100 stránek za sekundu ve veřejné části aplikace).
- **Dostupnost:** Stanovíme dostupnost, kterou bude muset aplikace splňovat (například aplikace bude z 98 % plně funkční). Případně se stanoví sankce za nedodržení těchto podmínek.
- **Škálovatelnost:** Měli bychom být schopni stanovit, jak může být zvýšen výkon aplikace v případě potřeby a zda je to vůbec možné.
- **Bezpečnost:** Musí se naplánovat způsob, kterým budou probíhat testy bezpečnosti aplikace, a jaké výsledky jsou pro zákazníka přijatelné.

- **Udržitelnost a rozšiřitelnost:** Musíme specifikovat, zda bude možné nadále rozšiřovat aplikaci o další funkčnost a jak by byl celý proces časově a finančně náročný.

5.3 Návrh datové struktury

K navržení vhodné datové struktury potřebujeme mít kvalitně provedenou a ukončenou fázi sběru a analýzy požadavků a samozřejmě také určité zkušenosti s tvorbou webových aplikací. [9] Při návrhu datové struktury můžeme být postaveni před dva zcela odlišné úkoly:

- Navrhnout novou datovou strukturu
- Postavit webovou aplikaci nad existujícím datovým zdrojem

Často musíme překonat nedostatky existujícího datového modelu, při jehož vzniku jsme ne vždy tušili, že se jednou stane nosným pro webovou aplikaci. V podstatě se nám nabízí dva způsoby řešení této situace:

- **Navrhnout novou datovou strukturu** a provést převod dat z existující do nově navržené datové struktury, což nemusí být triviální záležitost, zejména pokud se převod dat musí provést za chodu původní aplikace. V tomto případě získáme především možnost přizpůsobit datovou strukturu potřebám nově vznikající webové aplikace.
- **Postavit aplikaci na původní datové struktuře** bez jakýchkoli změn, případně tuto strukturu pouze rozšířit na základě požadavků webové aplikace. Jedná se v podstatě o jediné možné řešení, pokud nad původní datovou strukturou funguje jiná aplikace a naše webová aplikace je pouze rozšířením původního systému. V tomto případě můžeme fázi datového modelování přeskočit. Samozřejmě musíme promyslet a navrhnout, jaké části z původní datové struktury bude naše aplikace využívat.

5.3.1 Vstupy a výstupy pro návrh datové struktury

Základním vstupem pro návrh datové struktury by se měla stát především specifikace a analýza uživatelských požadavků. Konkrétně se jedná o:

- datový slovník
- funkční a uživatelské požadavky
- mapa budovaného webu
- již existující datové zdroje

Výstupem návrhu datové struktury je **datové schéma**. Běžně se využívá ER diagramů.

5.4 Návrh hypertextového modelu

Pokud máme vytvořený datový model, můžeme začít s návrhem hypertextového modelu. Nás nyní nebude zajímat, jak se takový diagram zakresluje, ale jak se postupuje při tvorbě hypertextového modelu jako celku.

Základním předpokladem pro vytvoření modelu je existence následujících podkladů:

- datové schéma
- mapa webu
- uživatelské a funkční požadavky

Celý proces můžeme rozdělit na hrubý a detailní návrh.

5.4.1 Hrubý návrh

Při tvorbě hypertextového modelu musíme zpočátku rozčlenit aplikaci do takzvaných **pohledů**, které představují hlavní součásti aplikace (typicky například veřejná část webu nebo administrační rozhraní).

Jednotlivé pohledy na webovou aplikaci musíme rozčlenit do **oblastí** (area). Oblast představuje základní složku (modul) budované webové aplikace, která se skládá z jednotlivých typů stránek respektive obrazovek. Při identifikaci oblastí vycházíme z funkčních požadavků a mapy webu. Často platí vztah, že oblast odpovídá jednomu případu

užití z UseCase diagramů. Například můžeme definovat oblast editace zboží, která odpovídá stejně pojmenovanému případu užití.

Pokud máme identifikované jednotlivé oblasti, můžeme přejít k definici viditelnosti a přístupnosti jednotlivých oblastí. Zajímá nás především, která oblast bude implicitní po vstupu do daného pohledu aplikace a které oblasti budou typu *landmark*, tedy stále přístupné z jakékoli jiné oblasti. Z těchto typů oblastí se obvykle generuje menu aplikace. Následuje identifikace oblastí, které nejsou označené ani jako default ani jako landmark a jsou tedy přístupné pouze prostřednictvím odkazů z jednotlivých stránek, nikoli z menu. [9]

5.4.2 Detailní návrh

Náš první úkol ve fázi detailního návrhu představuje identifikace stránek, ze kterých se budou skládat nalezené oblasti. Například oblast editace zboží se může skládat ze stránky se seznamem zboží pro danou kategorii a ze stránky s vkládacím, respektive editačním formulářem, případně dalších stránek.

Potom musíme definovat typ každé stránky a určit, zda je to:

- **Domovská stránka:** Její význam není třeba vysvětlovat. Jedná se o stránku, která se automaticky načte při zadání adresy příslušné prezentace. Takto označená může být samozřejmě pouze jedna stránka.
- **Implicitní stránka:** Implicitní stránka je obdobou domovské stránky pro příslušnou oblast. Je to tedy stránka, která se automaticky zobrazí, pokud vstoupíme do příslušné oblasti (sekce) webu.
- **Stále přístupná (landmark) stránka:** Stále přístupné stránky představují stránky, na které se můžeme dostat z jakékoli jiné stránky, která je obsažena ve stejné přímo nadřazené oblasti nebo pohledu na webovou prezentaci.
- **Klasická stránka:** Všechny ostatní stránky dosažitelné přes klasický odkaz z jiné stránky.

5.5 Implementace a testování

Návrhem samozřejmě vývoj aplikace nekončí. Následuje fáze implementace, které je věnována samostatná kapitola.

Testování je jedna z důležitých fází vývoje. Bohužel, již méně firem je ochotno na něj uvolnit adekvátní množství finančních prostředků. Důležité je vědět, že správné testování nepředstavuje pouze proklikávání aplikací. Měly by být sestaveny testovací scénáře a zátěžové testy.

6 OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ A NÁSTROJE PRO NÁVRH OOP APLIKACÍ

6.1 Co je to vlastně objektově orientované programování ?

Objektově orientované programování je jakási metodika vývoje softwaru, založena na následujících myšlenkách:

- *Objekty* – jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných objekty.
- *Abstrakce* – programátor, potažmo program, který vytváří, může abstrahovat od některých detailů práce jednotlivých objektů. Každý objekt pracuje jako černá skříňka, která dokáže provádět určené činnosti a komunikovat s okolím, aniž by vyžadovala znalost způsobu, kterým vnitřně pracuje.
- *Zapouzdření* – zaručuje, že objekt nemůže přímo přistupovat k „vnitřnostem“ jiných objektů, což by mohlo vést k nekonzistenci. Každý objekt navenek zpřístupňuje *rozhraní*, pomocí kterého (a nijak jinak) se s objektem pracuje.
- *Dědičnost* – objekty jsou organizovány stromovým způsobem, kdy objekty nějakého druhu mohou *dědit* z jiného druhu objektů, čímž přebírají jejich schopnosti, ke kterým pouze přidávají svoje vlastní rozšíření. Tato myšlenka se obvykle implementuje pomocí rozdělení objektů do *tříd*, přičemž každý objekt je instancí nějaké třídy. Každá třída pak může dědit od jiné třídy (v některých programovacích jazycích i z několika jiných tříd).
- *Polymorfismus* – odkazovaný objekt se chová podle toho, jaký je jeho skutečný typ. Pokud několik objektů poskytuje stejné rozhraní, pracuje se s nimi stejným způsobem, ale jejich konkrétní chování se liší. V praxi se tato vlastnost projevuje např. tak, že na místo, kde je očekávána instance nějaké třídy, můžeme dosadit i instanci libovolné její podtřídy (třídy, která přímo či nepřímo z této třídy dědí), která se může chovat jinak, než by se chovala instance rodičovské třídy, ovšem v rámci mantinelů, daných popisem rozhraní.

Základním principem OOP je snaha modelovat, při řešení úloh, svět v počítači pokud možno jedna ku jedné. V praktickém životě otevíráme dveře pořád stejně, bez ohledu na to, zda jsou dřevěné nebo laminované, zda mají kukátko, bezpečnostní vložku nebo řetízek navíc. Stejně tak se můžeme dívat na televizi, přepínat programy a docela dobře ji ovládat, přestože nevíme vůbec nic o principech jejího fungování. Analogicky při vývoji složitých informačních systémů mohou jednotliví vývojáři používat již vytvořené komponenty, podle potřeby si je trochu upravit nebo je používat jako stavebníci pro sestavování důmyslnějších a složitějších objektů. [10]

Existuje velké množství programovacích jazyků umožňujících objektově orientované programování, např. Smalltalk, Java, C++, Object Pascal, C#, Lisp, PHP, ...

6.2 FRAMEWORK

6.2.1 Co je to FRAMEWORK?

Mezi programátory se v poslední době slovo FRAMEWORK hodně rozšířilo, bohužel se pro tento programátorský termín těžko hledá přesná definice. Pokud budete hledat slovíčko framework v anglicko-českém slovníku, naleznete jeho český ekvivalent jako rámec, konstrukce či systém. Český překlad není daleko od pravdy. Framework je v podstatě jakási nádstavba nad API (Application Program Interface). Je to základ naší aplikace, ve kterém jsou definovány a implementovány nejrůznější námi vytvořené objekty, funkce, ovládací prvky a jiné části programu, které budeme využívat libovolně ve všech částech naší aplikace.

6.2.2 K čemu slouží FRAMEWORK?

Dobře naprogramovaný framework zajišťuje jeho přenositelnost. Na jednom frameworku tak můžeme postavit několik aplikací, které se sobě na první pohled nemusí podobat. Mohou být určeny k rozdílným účelům, ale všechny využívají objektů a funkcí jednoho frameworku. Nepřehlédnutelná je také doba, o kterou se urychlí vývoj jednotlivých aplikací. Nejčastější rutinní funkce jsou totiž už napsány a programátorovi tak odpadne velké množství práce.

6.2.3 Struktura FRAMEWORKU

Není přesné pravidlo, či šablona, podle které by se měl framework vytvářet. Nejčastěji se ale framework skládá z jednotlivých objektů, které se následně v aplikaci „vydělí“ a použijí.

6.3 Relační model databáze

6.3.1 Historie

V roce 1969 přišel doktor E. F. Codd (*A relational data model for large shared data banks*) se svou představou o databázi založené na matematickém aparátu relačních množin. V zásadě jde o to že stačí vzít množinu například osob a množinu adres a z nich vytvořit kartézský součin, představující možné vazby mezi jednotlivými množinami. Pak je možno považovat za relaci podmnožinu tohoto kartézského součinu odpovídající skutečnosti (tato konkrétní osoba bydlí na této konkrétní adrese), případně kartézský součin celý (formálně je relací libovolná podmnožina uvedeného kartézského součinu).

6.3.2 K čemu slouží relační model?

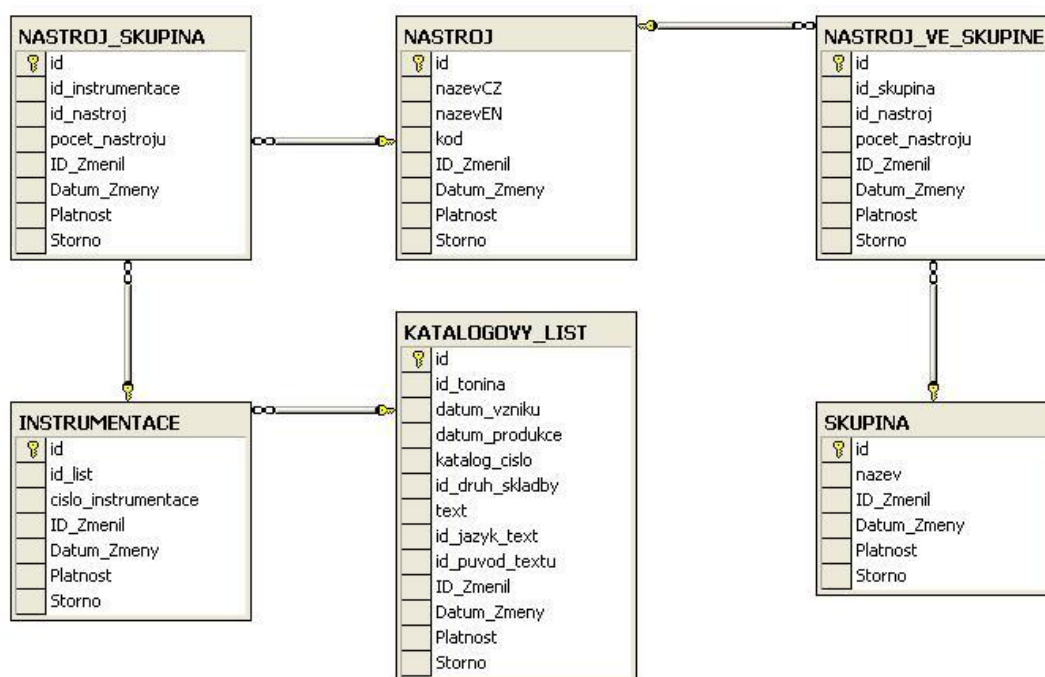
Relační databázový model sdružuje data do tzv. n-tic (tabulek). Tabulky tvoří základ relační databáze. Tabulka je struktura záznamů s pevně stanovenými položkami (sloupci). Každý sloupec má definován jednoznačný název, typ a rozsah. Záznam se stává řádkem tabulky. Pokud jsou v různých tabulkách sloupce stejného typu, pak tyto sloupce mohou vytvářet tzv. relace (vazby) mezi jednotlivými tabulkami. Tabulky se poté naplňují vlastním obsahem - konkrétními daty.

Kolekce více tabulek, jejich relací (vztahů mezi nimi), indexů (řazení podle vybraných sloupců) a dalších součástí tvoří relační databázi.

Relační model přináší celou řadu výhod, zejména mnohdy přirozenou reprezentaci zpracovávaných dat, možnost snadného definování a zpracování vazeb apod.

6.3.3 ER diagram

Pro přehledné zobrazení struktury relační databáze, jejich tabulek a vztahů mezi nimi, slouží tzv. ER diagramy (Entitně – relační diagramy).



Obr. 3. Příklad ER diagramu

6.4 UML – Unified modeling language

6.4.1 Co je to UML?

UML je jazyk, který umožňuje modelovat jednoduché i složité aplikace pomocí stejné formální syntaxe, a proto můžete výsledky své práce sdílet s ostatními návrháři. Unified Modeling Language (UML) má již ve svém názvu slovo unifikovaný, protože jedním z jeho cílů je sjednocení používaných výrazových prostředků. UML je jazykem s bohatou sémantikou a syntaxí, který usnadňuje návrh a vizualizaci různých typů aplikací. Navíc tvůrci jazyka UML nebyli natolik naivní ani sebevědomí, aby věřili, že standardními elementy jazyka UML pokryli kompletně veškeré požadavky návrhářů, a přímo do jazyka

zabudovali mechanismy rozšíření, které usnadňují řízenou deklaraci a definici nových elementů jazyka. [14]

6.4.2 Historie UML

Jazyk UML nevznikl ve vzduchoprázdnu, předcházelo mu myšlenkové úsilí mnoha znalců o vytvoření kvalitní metodiky. Následující stručný výčet metodik, které měly největší vliv na jazyk UML, si nečiní nárok na úplnost.

- Object modeling technique (Rumbaugh, Premerlani)
- Object oriented software development (G. Booch)
- Object oriented software engineering (I. Jacobson)
- Object oriented structured design notations (Wassermann, Pircher)

V roce 1996 pánové Jacobson, Booch a Rumbaugh, kteří již dříve významně přispěli k rozvoji OOP metodik, přešli k firmě Rational Corp., pro kterou vytvořili jazyk UML. V roce 1997 byl jazyk UML akceptován jako průmyslový standard a postupně začal vytlačovat ostatní analytické jazyky na pověstné smetiště dějin. V současné době se připravuje uvedení verze 2.0. V existujících CASE nástrojích se můžete většinou setkat s podporou UML verze 1.2, 1.3 nebo dokonce 1.4. [14]

6.4.3 Pohled na systém

Jaké hlavní aspekty modelované aplikace dovoluje UML zachytit? Model v UML se skládá z různých diagramů, jež představují průhledy na různé části sémantického základu navrhované aplikace. Sémantickým základem je souhrn specifikací aplikace, který vymezuje teritorium, v němž se může návrh pohybovat. Diagram ve vizuální formě vypráví právě jeden konkrétní "příběh" o aplikaci. Žádný dvourozměrný diagram nemůže zachytit komplexní aplikaci v celku, ale soustředí se vždy právě na jeden důležitý aspekt. Jazyk UML rozeznává pět základních pohledů na systém.

- *Pohled případů užití (diagramy UseCase).* V případech užití jsou vyjádřeny základní požadavky kladené na systém. Veškeré další pohledy se pohybují v mantinelech vymezených pohledem případů užití.

- *Logický pohled* se zabývá zejména pojmy z problémové domény zadavatele a jejich vzájemnými statickými vztahy.
- *Procesní pohled* se soustřeďuje na chování systému, které musí splňovat požadavky a omezení z případů užití, jež jsou kladeny na průběh procesů. Pregnantně řečeno, jedná se o procesně orientovaný doplněk logického pohledu.
- *Implementační pohled* zachycuje fyzické rozdělení aplikace na samostatné komponenty a jejich závislosti.
- *Pohled nasazení* mapuje komponenty na množinu fyzických výpočetních uzlů v cílovém prostředí.

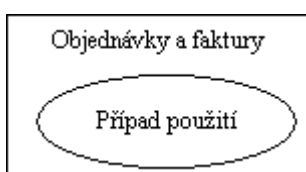
Pohledy jsou konkretizovány v následujících typech diagramů.

- Diagram tříd
- Diagram objektů
- Diagram komponent
- Diagram nasazení
- Sekvenční diagram
- Kolaborační diagram
- Stavový diagram
- Diagram aktivit

6.4.4 Notace pro případy užití

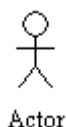
Případy užití jsou zachyceny ve vizuální a textové podobě. Diagramy případů užití (**UseCase diagrams**) nám poskytnou rychlou představu o jednotlivých funkcích systému, ale přesné postupy, rozšiřující a alternativní scénáře musejí být zachyceny v textové formě. V jazyce UML je kodifikován pouze diagram případů užití, strukturu dokumentu s textem případů užití si musíme navrhnout sami.

V UML diagramech zachycujeme případy užití, aktory a jejich vztahy. Případ užití je v UML diagramech zobrazen jako ovál s názvem případu.



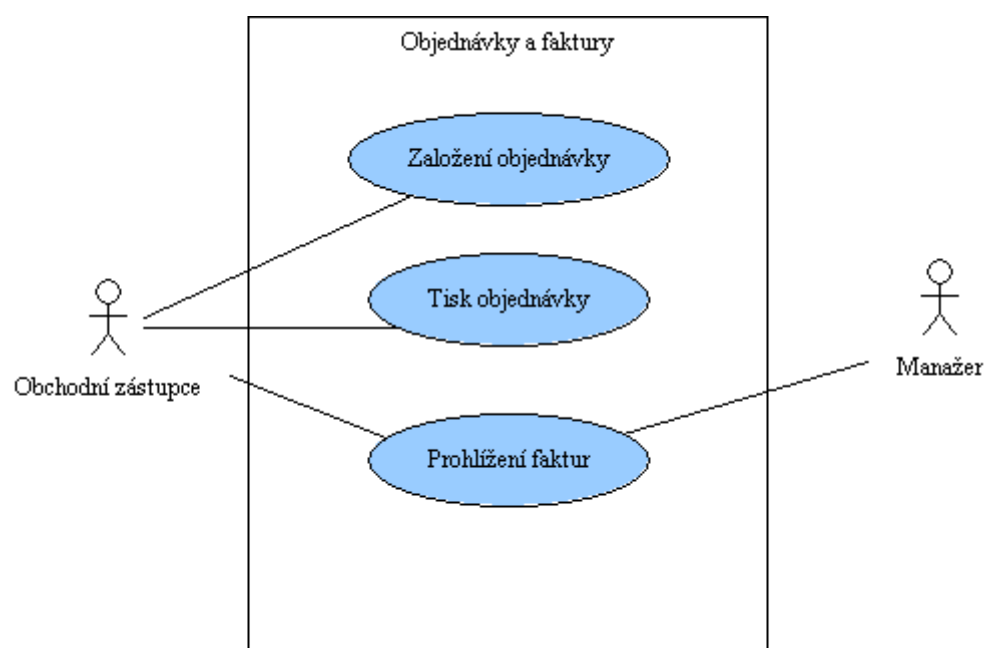
Obr. 4. Značka případu

Pro aktora jsou navrženy dva rovnocenné symboly. Prvním z nich je symbol postavy a druhým čtverec. Symbol postavy většina analytiků používá pro aktora zastupujícího osoby, symbol čtverce, na nějž je ještě aplikován stereotyp `<<system>>`, reprezentuje externí systémy. Oba symboly musejí vždy obsahovat název aktora.



Obr. 5. Značka aktora

Nejdůležitějším vztahem v diagramu je přiřazení případu užití k aktorovi. Přiřazení je vyjádřeno plnou nepřerušovanou čarou. U tohoto vztahu se předpokládá stereotyp `<<communication>>`, i když se v diagramu případů užití nezobrazuje. Jednoduchý diagram případů užití:



Obr. 6. Příklad užití UseCase diagramu

7 MIDI

7.1 Co je to MIDI

Digitální rozhraní hudebních nástrojů (neboli Musical Instrument Digital Interface) poskytuje uživatelům univerzální jazyk, který je společný jak hardwarovým zařízením, tak softwaru. MIDI můžeme rozdělit do tří samostatných celků: jazyk, také známý jako protokol; hardwarové rozhraní, které MIDI používá pro přenos a příjem informace; distribuční formáty, které MIDI používá pro přenos dat na jiné zařízení – příkladem může být Standard MIDI Format (SMF). Pro naši potřebu je nejdůležitější distribuční formát, ve kterém je uložený soubor MIDI, který budeme převádět na kódy SHK a RISM AII viz kap. 8.

7.2 Princip MIDI

MIDI je v zásadě jazyk, který popisuje hudbu v binární formě, kde každé binární slovo popisuje událost vyskytující se během hudebního výkonu. Když se nad tím zamyslíme, každý hudební nástroj vytváří nějaký zvuk, který ovládá hudebník. Právě hudebník určuje, kdy začne nástroj znít – houslista začne pohybovat smyčcem po strunách, basista brnkne na strunu, trumpetista vdechne vzduch do trumpetky a klávesista stiskne klávesu. Všechny tyto činnosti můžeme považovat za událost Note On. Zpráva MIDI sama o sobě neobsahuje zvuk, který by byl nástrojem zahrán, ale spíše popisuje událost, která vedla k vytvoření daného zvuku – jako například která nota bude zahrána, jak tvrdě nebo měkce, kdy má být nota zahrána atd. Jakmile cellista přestane pohybovat smyčcem po strunách, basista přestane brnkat na strunu, trumpetista vdechovat vzduch do trumpetky a klávesista uvolní klávesu, zvuk přestane znít. Popsaný jev lze v jazyce MIDI interpretovat jako událost Note Off. [12]

7.3 Struktura MIDI souboru (Standard MIDI Format)

7.3.1 Hlavička MIDI

Hlavička SMFF (Leader Chunk) je skupina bajtů, které MIDI používá k uložení nebo přenosu informace do nebo ze SMF souboru. Je to základní kámen souborů MIDI. Existují dva typy hlaviček: **MThd Chunk** – datová hlavička SMFF a **MTrk Chunk** – data stopy (track stopy). Obě mají podobnou strukturu, ale slouží k rozdílným účelům.

MThd (chunk)

V souboru MIDI se vždy nachází pouze jedna datová hlavička MThd a má na starosti identifikovat soubor jako formát Standard MIDI, určit počet obsažených stop a způsob dělení (time base nebo clock base). Protože se každá událost v souboru MIDI vyskytuje v daném čase, hlavička definuje určitou minimální dobu, která by měla mezi jednotlivými událostmi uplynout. Time base nebo clock base představuje základní hodnotu, která je aplikována na celý soubor. Jinak řečeno, jedná se o hodnotu časového měřítka (time grid), která slouží jako reference k zarovnávání všech událostí obsažených v souboru. MThd hlavička vždy obsahuje 6 datových bytů informací.

<Hlavička SMF> = <HThd> <délka> <formát> <počet stop> <rozlišení>

<HThd> určuje druh hlavičky, tedy hlavičku SMF

<délka> 32-bitová reprezentace čísla 6, jež udává délku hlavičky

<formát> viz formát SMF

<počet stop> udává počet stop, pro formát 0 je to vždy jedna, velikost je dva byty

<rozlišení> udává vztažnou hodnotu pro kódování *deltačasu*, obvykle určuje relativní čas jako počet tiků do jedné čtvrté hodnoty (noty).

Tab. 1. Příklad hlavičky SMF pro formát 0

4D 54 68 64	00 00 00 06	00 00	00 01	01 80
MThd	délka 6 bytů	formát 0	jedna stopa	96 tiků

Formát SMF

SMF formát slouží k jednotnému kódování hudebních i jiných dat, což dovoluje přenos těchto dat mezi nejrozličnějšími přístroji. Může mít tři hodnoty:

0 – soubor se skládá z jedné stopy, která může obsahovat více MIDI kanálů

1 – soubor se skládá z jedné i více současně hrajících stop.

2 – celá skladba je uložena do jednoho souboru, je určen pro patternově orientované sekvencery, kde každý pattern má obsahovat jednu multikanálovou stopu.

MTrk Chunk (Eventy MTrk)

Event MThd je obvykle následován sérií eventů MTrk. Vždy existuje nejméně tolik MTrk eventů, kolik je stop v souboru MIDI. Pokud by se například jednalo o soubor MIDI formátu 0 a obsahoval by tedy pouze jednu stopu, po eventu MThd by následoval pouze jeden event MTrk. Pokud by ovšem šlo o soubor MIDI formátu 1, který by obsahoval 16 stop, počet eventů by se zvýšil nejméně na 16.

<MTrk Chunk> = <MTrk><délka><data>

<HTrk> určuje druh hlavičky, tedy hlavičku stopy

<délka> délka stopy v bytech

<data> obsahuje události MIDI, události SyEx, nebo Meta-události. Pro naši potřebu jsou důležité Meta-události a události MIDI

7.3.2 Události MIDI a Deltačas

Obsahují hudební data stopy (zprávy) spolu s údajem o relativním časovém umístění, tzv. deltačas. Deltačas může být různě dlouhý, proto i počet bytů pro jeho popis je různý. Pro kódování se používá převod osmibitových skupin na sedmibitové a pak se do MSB doplní ve všech bytech 1, jen do posledního 0.

Příklad 1.3.2.1

Budeme kódovat deltačas o velikosti 2000 hexadecimálně.

2	0	0	0	hexadecimálně
0010	0000	0000	0000	binárně

Převod na sedmibitové skupiny:

00	1000000	binárně
0000000		

Do MSB 1, do posledního 0 a máme zakódovaný deltačas:

1100	0000	0000	0000	binárně
C	0	0	0	hexadecimálně

Jako nejvyšší vstupní číslo je dovoleno 0F FF FF FF (hexadecimálně), které se zakóduje jako FF FF FF 7F. Teoreticky je možné i větší číslo, ale v praxi to bohatě stačí, protože takto vysoká hodnota představuje vzdálenost mezi dvěma hudebními událostmi několik dní (záleží na rozlišení a tempu).

7.3.3 Meta – události

Meta-události se používají k uchovávání speciálních informací přímo nesouvisejících s MIDI. Všechny meta-události začínají vyhrazenou hodnotou status byte FF, která událost identifikuje jako meta-událost. Následuje byte, který se chová podobně jako druhý status byte, protože definuje povahu následující hodnoty. Po tomto identifikačním bytu následuje hodnota vyjadřující délku této události.

Pokud není určeno jinak, nejsou meta-události povinné. V jednom MTrk chunku se může vyskytovat více než jedna meta-událost. V tomto případě jsou jednotlivé meta-události odděleny pomocí hodnot deltatime, které určují, kdy má událost vejít v platnost. Následuje přehled nejdůležitějších meta-událostí.

Číslo sekvence (sequencer number)

- jedná se o volitelnou událost
- Společná identifikační hodnota: FF 00 02
- Počet datových bytů: 2 byty
- Znáznornění zprávy v hexadecimálním kódu: FF 00 02 XX XX, kde XX představují proměnné byty
- Představuje: V souboru MIDI formátu 0 nebo 1 je tato událost umístěna v prvním kousku MTrk chunku a představuje číslo sekvence. V celém souboru se může číslo sekvence vyskytovat pouze jednou.

Textová událost (copyright)

- Jedná se o nepovinnou meta-událost, která se obvykle nachází na začátku eventu MTrk
- Společná identifikace: FF 02
- Počet datových bytů: proměnný

- Znázornění zprávy v hexadecimálním kódu: FF 02 LL LL XX XX XX, kde LL představuje délku této události v bytech a XX zase text v 8bitovém kódu ASCII nebo 8bitovém binárním formátu.

Nástroj (instrument)

- Jedná se o nepovinnou meta-událost, která se většinou nachází na začátku eventu MTrk
- Společná identifikační hodnota: FF 04
- Znázornění zprávy v hexadecimálním kódu: FF 04 LL LL XX XX XX....., kde LL představuje délku této události v bajtech a XX zase text v 8bitovém kódu ASCII nebo 8bitovém binárním formátu. Délka je vyjádřena ve formátu proměnné délky, takže lze k jejímu znázornění použít neomezené množství bytů.
- Představuje: Jméno stopy. Může tedy představovat jméno nástroje, který je určen k přehrání události obsažených v této stopě, ale také něco zcela jiného.

Konec stopy (End of track)

- Jedná se o povinnou meta-událost, která by se měla nacházet na konci každé části MTrk v souboru MIDI.
- Společná identifikační hodnota: FF 2F 00
- Počet datových bytů: žádný.
- Znázornění zprávy v hexadecimálním kódu: FF 2F 00
- Představuje: Tato meta-událost se používá k označení konce části MTrk. U každé části MTrk (chunk) se nachází pouze jedna tato meta-událost

Tempo

- Jedná se o nepovinnou meta-událost a může se vyskytovat kdekoli v kousku MTrk. Pokud se v kousku MTrk tato událost nevyskytuje, je použito implicitní nastavení.

- Společná identifikační hodnota: FF 51 03
- Počet datových bytů: 3 bajty
- Znázornění zprávy v hexadecimálním kódu: FF 51 03 XX XX XX, kde XX představuje hodnotu v mikrosekundách za jednu čtvrt'ovou notu; jinak řečeno, jak dlouhá bude jedna čtvrt'ová nota.
- Představuje: Udává tempo přehrávání skladby v mikrosekundách. Například tempo 160 BPM by bylo převedeno na mikrosekundy (375 000 mikrosekund), potom do hexadecimálního kódu (05 B8 D8). Tato hodnota pak představuje trvání jedné čtvrt'ové noty v mikrosekundách. [12]

7.4 Příklad souboru MIDI

Tab. 2. Soubor MIDI

Příklad skladby v SMF				
Data				Komentář
4D	54	68	64	MThd
00	00	00	06	délka hlavičky
00	00	00	01	formát 0, jedna stopa
		01	80	96 tiků na jednu čtvrt'ovou
4D	54	72	6B	MTrk
00	00	00	3A	délka stopy - 58 bytů
00	FF	58	04	<i>Meta-event</i> : metrum 3/4, metronom
03	02	18	08	každou čtvrt'ovou, 8 dvaatřicetinek do jedné čtvrt'ové
00	FF	51	03	<i>Meta-event</i> : tempo, jedna čtvrt'ová
	07	A1	20	trvá 500 000 sec
D-čas	Data			Komentář
00	90	46	6E	zapnuta nota 70 rychlostí 110

81 40		46	00	vypnuta nota 70
00		2D	50	zapnuta nota 45 rychlostí 80
60		2D	00	vypnuta nota 45
00		26	0D	zapnuta nota 38 rychlostí 80
00		3E	0D	zapnuta nota 62 rychlostí 80
00		41	0D	zapnuta nota 65 rychlostí 80
00		45	0D	zapnuta nota 69 rychlostí 80
81 40		3E	00	vypnuta nota 62
00		41	00	vypnuta nota 65
00		45	00	vypnuta nota 69
00		26	00	vypnuta nota 38
60	FF	2F	00	<i>Meta-event</i> :konec stopy

Tabulka je rozdělena do třech částí. Na začátku je inicializační část s hlavičkou souboru a stopy. Stopa je dlouhá 58 bytů. Dále následují dva meta-eventy udávající metrum a tempo. Protože jsou již součástí stopy, je před každým údaj o deltačase. Následují data, kde je použit trvající stavový byte. Soubor je ukončen meta-eventem znamenajícím konec stopy.

8 SHK A RISM AII

Sbírky hudebnin bývaly buď součástí větších knihoven patřících církevním či vzdělávacím institucím a soukromým osobám nebo existovaly jako samostatné sbírky u různých provozovacích těles, ať už to byl venkovský kostelní kůr nebo operní dům. Potřeba a způsob jejich evidence se vyvíjely a měnily s dobou od starých rukopisných inventářů až po dnešní katalogy a digitální databáze. Nezbytnost systematické odborné evidence hudebních pramenů vyvstala spolu se vznikem novodobých věd, zvláště hudební historiografie. [15]

8.1 SHK

SHK kód je numerický kód, který využívá souborný hudební katalog (soupis hudebních pramenů uložených na území Čech a Moravy – dále SHK) za účelem evidence hudebních incipitů v České republice.

8.1.1 Popis kódu

1. údaj v kódu = počet předznamení (počet křížků a béček)

nejde o specifikaci tóniny – nerozhoduje zda jde o dur či moll

1 křížek = 18, 2 křížky = 28, ... 7 křížků = 78

1 béčko = 19, 2 béčka = 29, ... 7 béček = 79

2. údaj v kódu = metrum

$2/2 = 22$

$2/4 = 24$

$3/4 = 34$

$4/4 = 44$

$3/8 = 38$

$6/8 = 68 \dots$

3. údaj = 10 znaků notového incipitu – tóny a pauzy**Tóny:**

Neberou se v úvahu ligatury. Nebere se v úvahu předznamenání. Ke každé notě se píše posuvka bez ohledu na předznamenání.

C – H

C = 10, D=20, ... H=70

Křížek = 8

Béčko = 9

Cis =18, Dis=28, ... His=78

Ces=19, Des=29, ... B=79

Pauzy:

Celá = 01

Půlová =02

Půlová s tečkou = 03

Čtvrt'ová = 04

Osminová=08

Šestnáctinová=06

Všechny ostatní pauzy = 09 (tj. např. deset taktů, dva takty, dvaatřicetinová pauza aj.)

Příklad 2.2.1.1:



Obr. 7 SHK – Incipit č.1

38	22	18	20	48	30	30	18	48	30	30	28
----	----	----	----	----	----	----	----	----	----	----	----

Příklad 2.2.2.2:



Obr. 8 SHK – Incipit č.2

29	44	20	40	50	39	20	08	04	79	39	79
----	----	----	----	----	----	----	----	----	----	----	----

8.2 RISM AII

Na rozdíl od kódu SHK jde o věrný přepis incipitu do alfabeticko-numerického kódu. Nevyjadřuje se počet předznamenání (ani druh tóniny) a metrum. Tento kód byl vytvořen pro účel evidence notových incipitů v projektu *RISM - Répertoire Interational des Sources Musicales*. Soupis pramenů RISM si vytyčil za cíl kompletní soupis světových hudebních

pramenů. Do evidence přispívají pracovní skupiny z 32 zemí. Celkový počet elektronicky dostupných záznamů o hudebních pramenech z České republiky je nyní cca 16 959 titulů, celkově lze vyhledávat v elektronické databázi světových hudebních pramenů v počtu 504 505 titulů. Publikace RISM jsou rozděleny do řad A-alfabetická, B-systematická a C-seznam hudebních knihoven. Řada A se dělí ještě na A/I hudební tisky cca před rokem 1850 a A/II rukopisy po roce 1600. [15]

8.2.1 Popis kódu

Registr oktav

´ = jednočárková oktáva

'' = dvoučárkovaná oktáva

''' = tříčárkovaná oktáva

, = malá oktáva

, = velká oktáva

,,, = kontra oktáva

Rytmické hodnoty

0 = longa

9 = brevis

1 = celá

2 = půlová

4 = čtvrt'ová

8 = osminová

6 = šestnáctinová

3 = dvaatřicetinová

5 = čtyřiašedesátinová

7 = stodvacetiosminová

4. = čtvrt'ová s tečkou

8.. = osminová s dvěmi tečkami

Předznamení

x = křížek

xx = dvoukřížek

b = béčko

bb = dvě béčka

n = odrážka

Tóny

Název noty a značení si odpovídá - C,D,E,F,G,A,B, ...

Pauzy

8- = osminová pauza

2- = půlová pauza

= = jeden takt pauza

=1 = jeden takt pauza

=35 = 35 taktů pauza

kvintoly, trioly ...

Závorka a před závorkou typ vazby.

Např.

Kvintola: 8(3ABCDE;5) – pět dvaatřicetin v jedné osmině

Příklady 2.3.1.1:



Obr. 9 RISM – Incipit č.1

8xCDxFE4E8.E6xC8.xF6E8.E

Příklad 2.3.1.2:



Obr. 10 RISM – Incipit č.2

8D4.F6GbE8D6-4-4bB4.D6bE

II. PRAKTICKÁ ČÁST

9 NÁVRH APLIKACE

9.1 Zvolení typu aplikace

Hlavním úkolem celého projektu je uchovávání údajů o hudebních dílech Vinzenze Maschka. Je proto samozřejmé, že vytvořená aplikace musí spolupracovat s databází, ve které budou data uložena. Dále je důležité, aby bylo možné dynamicky a pružně reagovat na požadavky uživatele a provést v co nejkratší době potřebné změny. Aplikace by také měla být přístupná co největšímu okruhu badatelů a nebýt funkční jen lokálně na jednom počítači.

Přirozeně se tedy nabízelo vyvinout webovou aplikaci s třívrstevným aplikačním modelem, funkčně od sebe oddělující vrstvu uživatelského, aplikačního rozhraní a databáze.

9.2 Zvolení vývojových prostředků

Prostředků pro vývoj webové aplikace je celá řada. Naskýtá se možnost vybrat si z celé řady možných implementací od PHP přes JSP až po ASP. Mým záměrem bylo vytvořit aplikaci využívající nejmodernějších technologií a postavit ji na vlastním frameworku, který bude možné dále rozšířit a využít i u dalších projektů.

Využil jsem tedy nové verze technologie ASP.NET 2.0 a moderního programovacího jazyka C#, který si bere to nejlepší z jazyků C++ a Visual Basic. Zde jsou vývojové prostředky, které byly použity:

- Programovací prostředí Microsoft Visual Studio 2005 Beta
- ASP.NET 2.0
- .NET FRAMEWORK 2.0
- programovací jazyk C#

Programovací prostředí Microsoft Visual Studio 2005 Beta je možné využívat bez poplatků a je ho možné získat dvojím způsobem. Jednak žádostí o zaslání DVD na stránkách www.microsoft.cz, nebo přímo stažením celého prostředí na stejných internetových stránkách. Vše zdarma.

Další nutností je zvolení databáze, která bude uchovávat data. K nejpoužívanějším relačním databázím na platformě Windows pro webové aplikace patří MySQL a SQL Server. ASP.NET 2.0 využívá databáze Microsoft SQL Server. V prostředí Microsoft Visual Studio 2005 Beta je přímo implementována verze Microsoft SQL Server 2005, protože však tato verze ještě není zcela doladěna, přiklonil jsem se k použití osvědčené verze Microsoft SQL Server 2000. Bohužel MS SQL Server 2000 je komerční software, který stojí nemalé peníze. Alternativou k němu je MSDE 2000, který má stejný databázový engine, pouze je zde omezení počtu současně vykonávaných transakcí.

9.3 Konfigurace počítače na kterém má být aplikace spuštěna

9.3.1 Minimální hardwarové požadavky (server)

- Procesor Intel Pentium II 300MHz / Intel Pentium III 600MHz
- Paměť 96 MB / 128 MB RAM
- HDD - zabrané místo 210 MB, místo nutné pro instalaci 360 MB

9.3.2 Minimální hardwarové požadavky (klient)

- Procesor 233 MHz
- Paměť 64 MB RAM
- Barevný monitor s rozlišením min. 800x600

9.3.3 Softwarové požadavky (server)

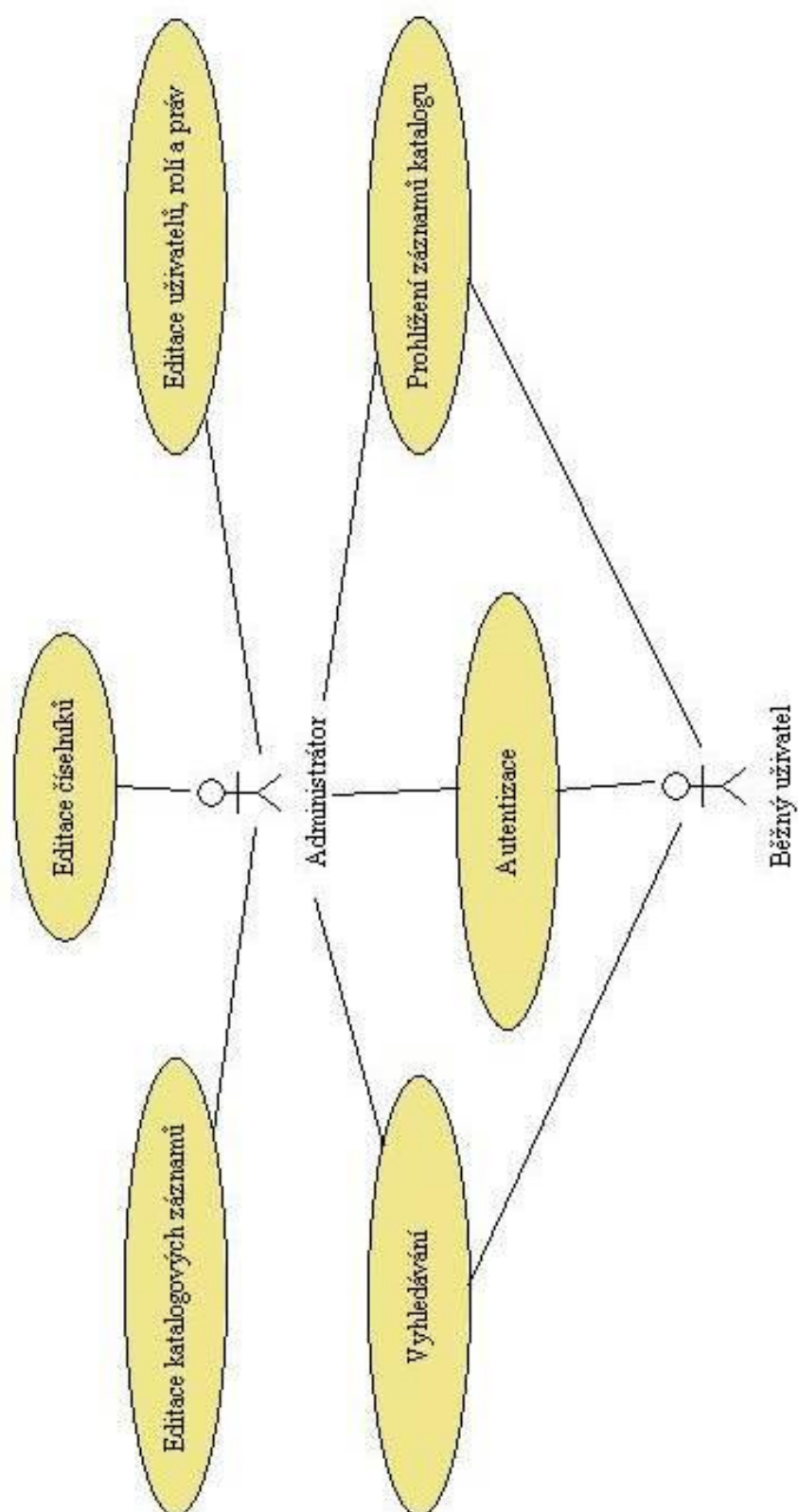
- Operační systém Windows
- Microsoft .NET FRAMEWORK SDK 2.0
- IIS 5.1 – Internetová informační služba
- MSDE 2000, nebo MS SQL Server 2000

9.3.4 Softwarové požadavky (klient)

- Připojení k internetu
- Internetový prohlížeč – IE 5.0, Mozilla, FireFox

10 USE CASE DIAGRAM APLIKACE

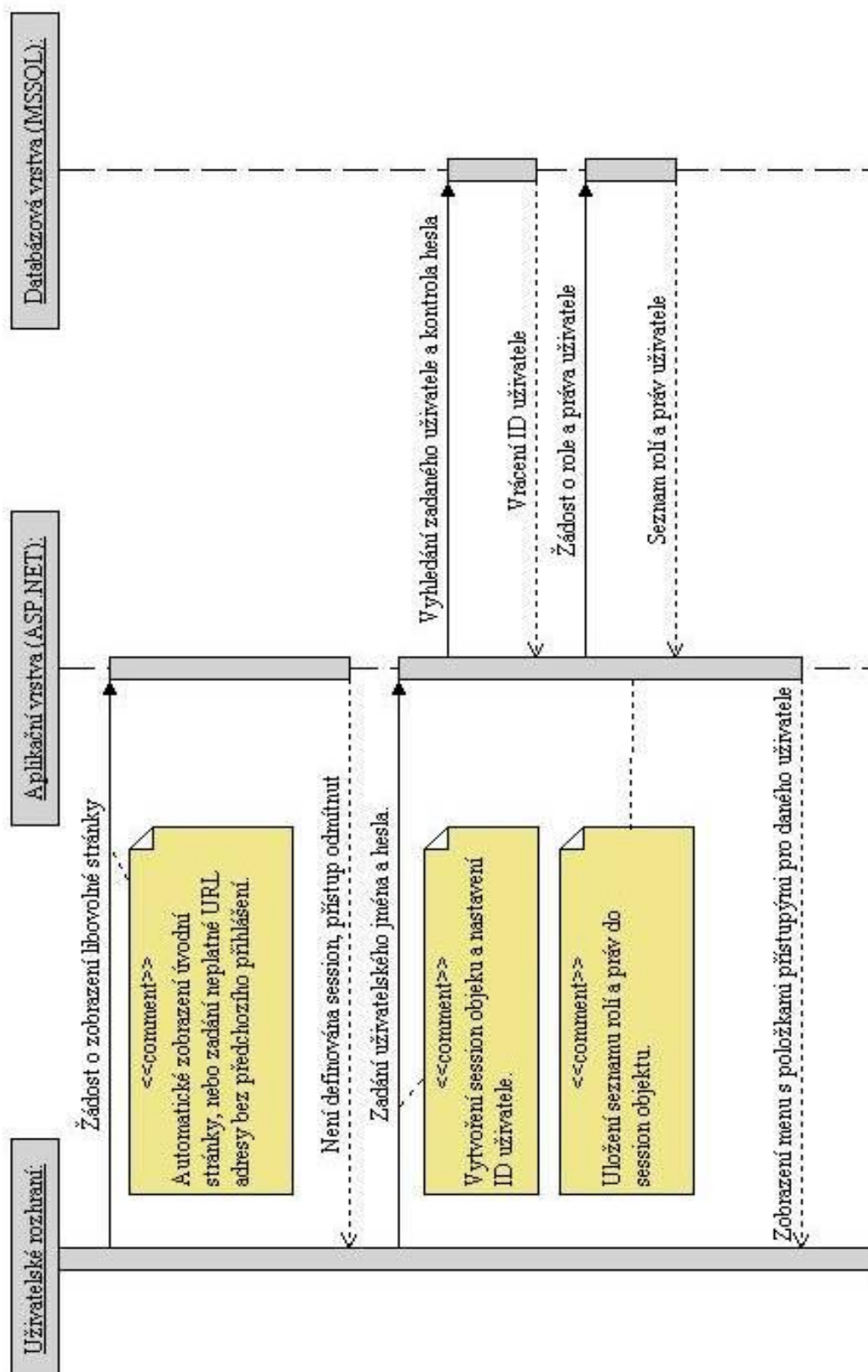
Požadavky kladené na systém je možné z pohledu uživatele přehledně znázornit UML diagramem typu UseCase. Naše aplikace jde popsat pomocí dvou aktorů – Administrátor a Běžný uživatel. Administrátor může v aplikaci provádět všechny činnosti, tzn. editovat číselníky, katalogové záznamy, přidávat a upravovat uživatele, jejich práva a role. Samozřejmě může provést svoji autentizaci (přihlášení) a vyhledávat v záznamech. Naproti tomu běžný uživatel může pouze prohlížet záznamy katalogu, vyhledávat v něm a autentizovat se.



Obr. 11. UseCase diagram

11 SEKVENČNÍ DIAGRAM APLIKACE

Následující diagram modeluje komunikaci mezi komponentami (vrstvami) uživatelského rozhraní a aplikační a databázové vrstvy, při provádění některých požadavků (pokus o zobrazení informací bez předchozí autorizace, autentizace uživatele podle účtu v databázi):

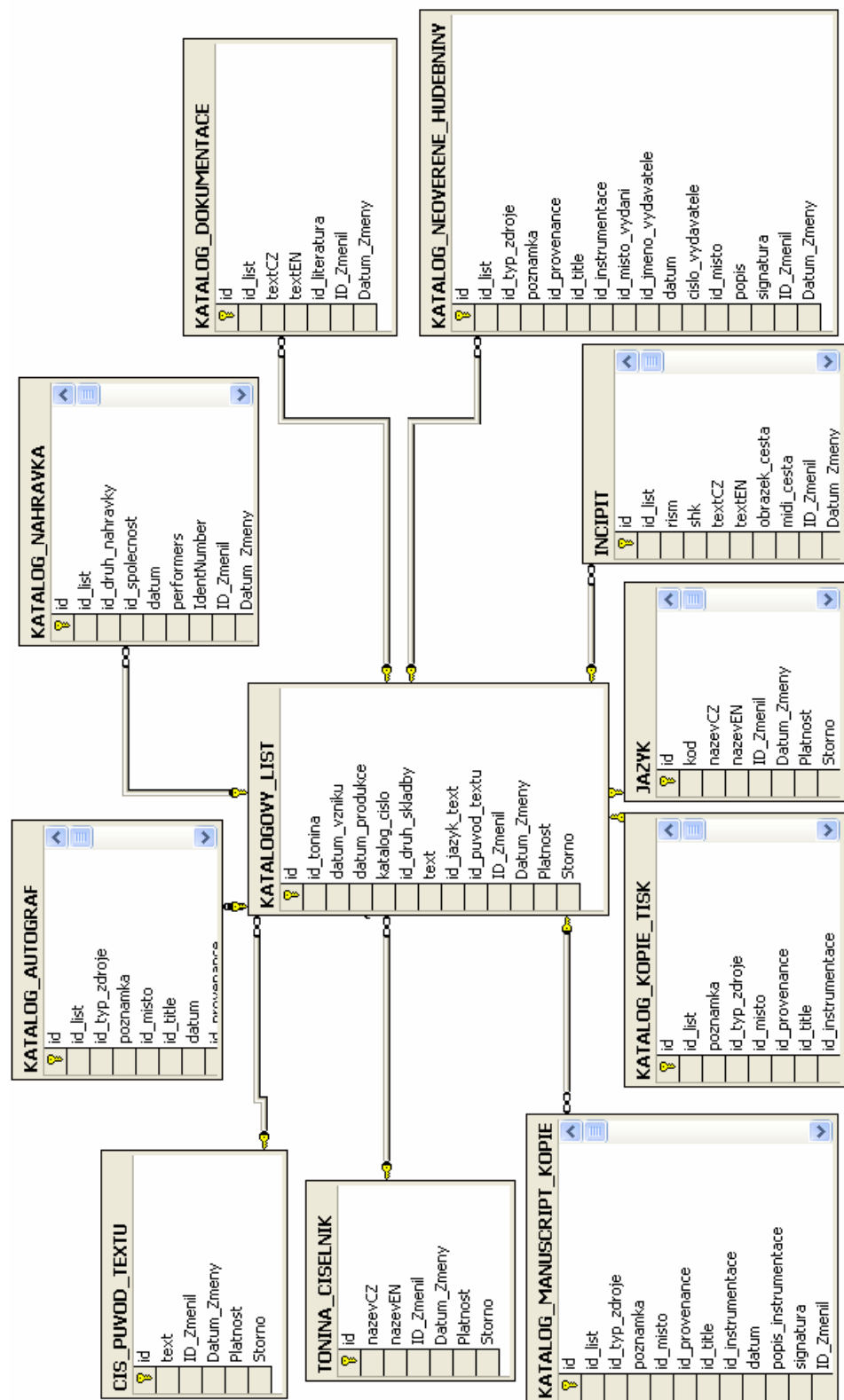


Obr. 12. Sekvenční diagram

12 ER DIAGRAMY DATABÁZE APLIKACE

V této podkapitole je popsána struktura celé databáze aplikace, stručný popis jednotlivých tabulek a jejich relační vztahy. Celá databáze je zde pro přehlednost rozdělena na menší části s vlastním názvem a ER diagramem, které jsou následně popsány.

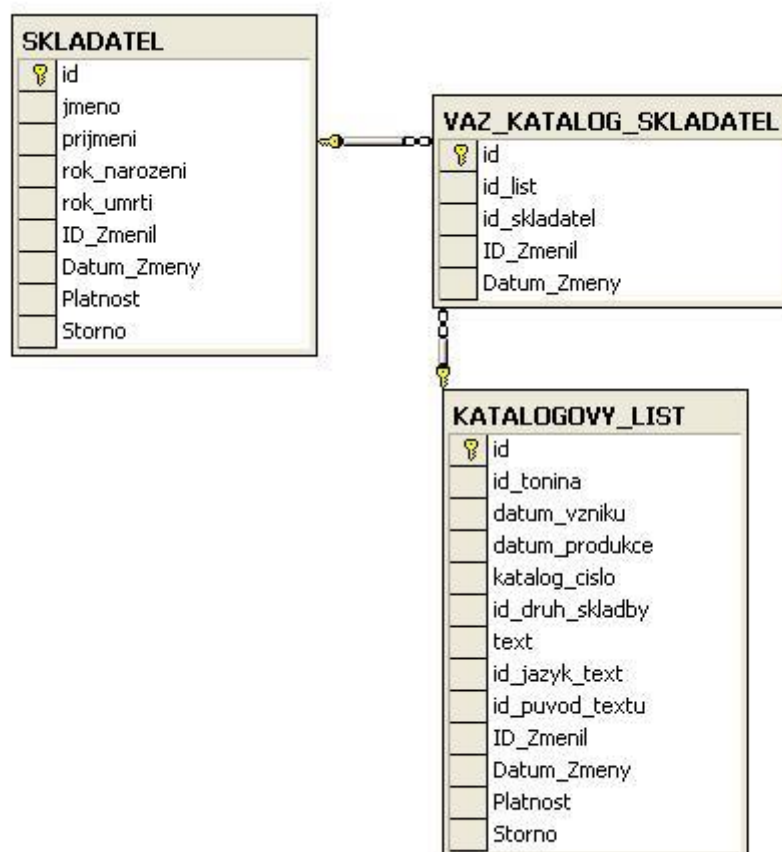
Katalogový list



Obr. 13. ER - Katalogový list

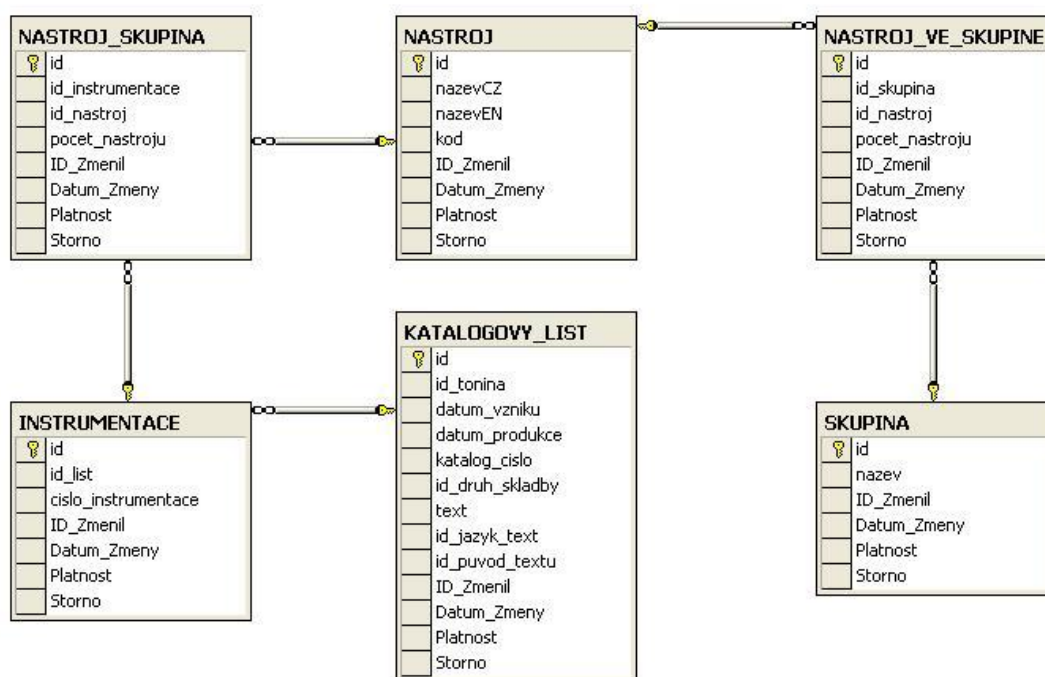
V tabulce katalogový_list jsou uloženy základní informace o každém jednom záznamu. Co řádek v tabulce katalogový_list to je jedno hudební dílo Vinzenze Maschka.

Vazba katalogového listu a číselníku skladatel



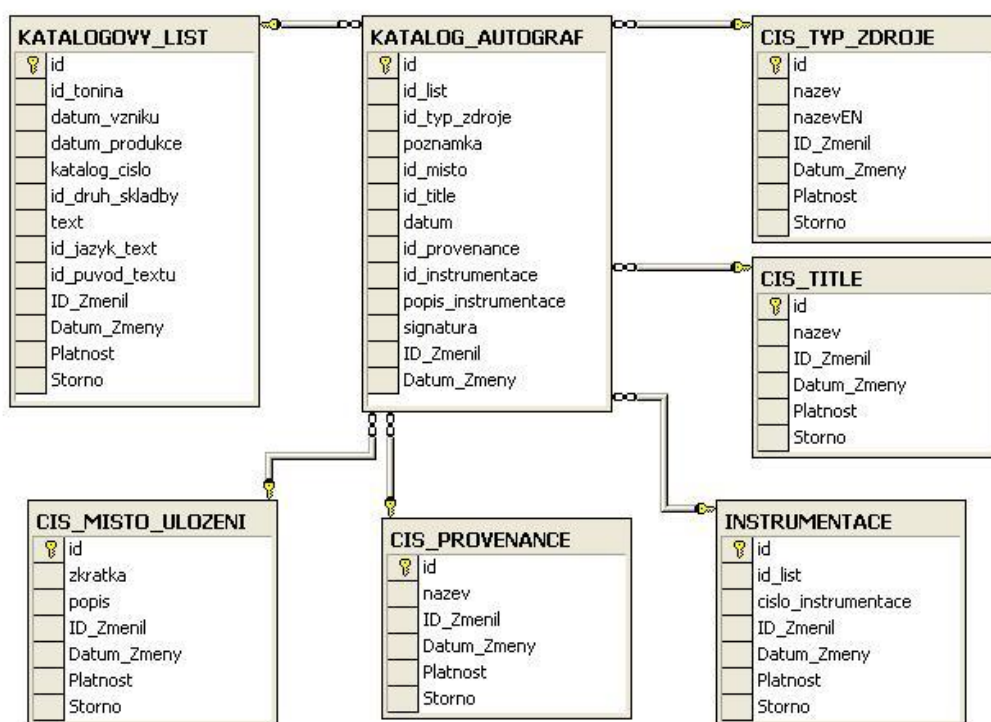
Obr. 14. ER - Vazba katalogový list a skladatel

Vinzenz Maschek byl znám pod více jmény a proto se u každé skladby může vyskytovat více jmen skladatele. Všechny jména Vinzenze Maschka pod, kterými byl autor znám, jsou uložena v číselníku SKLADATEL. Tento číselník je spojen s tabulkou KATALOGOVY_LIST přes „vazební tabulku“ VAZ_KATALOG_SKLADATEL.

Instrumentace

Obr. 15. ER-Instrumentace

Každá skladba může mít jednu, či více instrumentací. Instrumentace je jakýsi popis skladby z pohledu složení a počtu nástrojů, které ji reprodukuji. Každá instrumentace má tedy svoji skupinu nástrojů, která danou skladbu hraje. Na straně uživatele funguje vytváření instrumentace dvěma způsoby. V prvním způsobu vytváří uživatel instrumentaci zadáváním po jednotlivých nástrojích. V druhém způsobu si může uživatel vybrat z předem předdefinovaných skupin nástrojů (tabulky SKUPINA a NASTROJ_VE_SKUPINE), odkud se pak záznamy „přehrají“ do tabulek INSTRUMENTACE a NASTROJ_SKUPINA.

Autograf (rukopis)

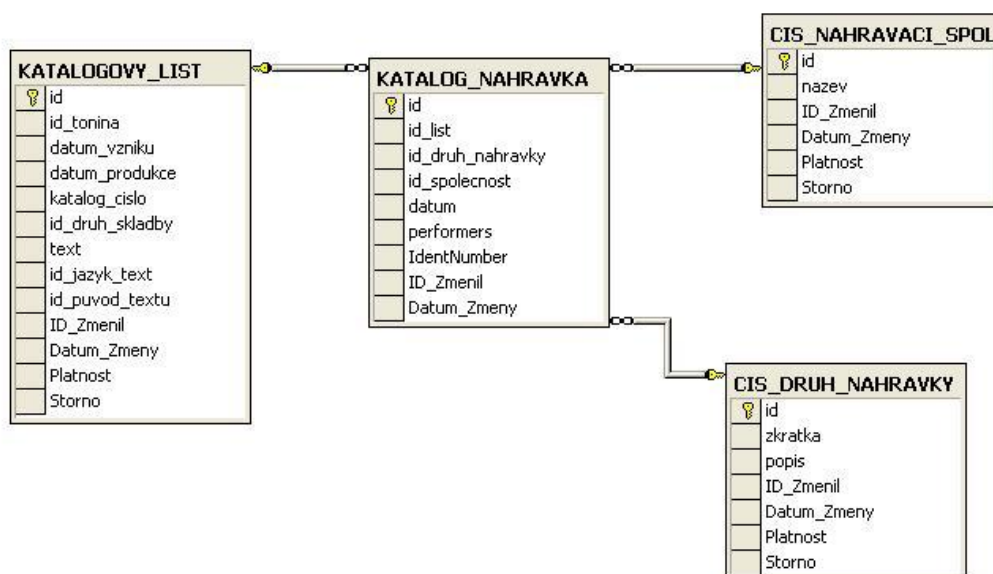
Obr. 16. ER-Autograf

Každá skladba má svůj autograf (rukopis), těchto rukopisů může být také více a proto je tabulka KATALOGOVY_LIST a KATALOG_AUTOGRAF spojena relací 1:n. Každý rukopis má pak svůj název, místo kde je právě teď uložen, oblast (provenanci) kde vznikl, zdroj a instrumentaci.

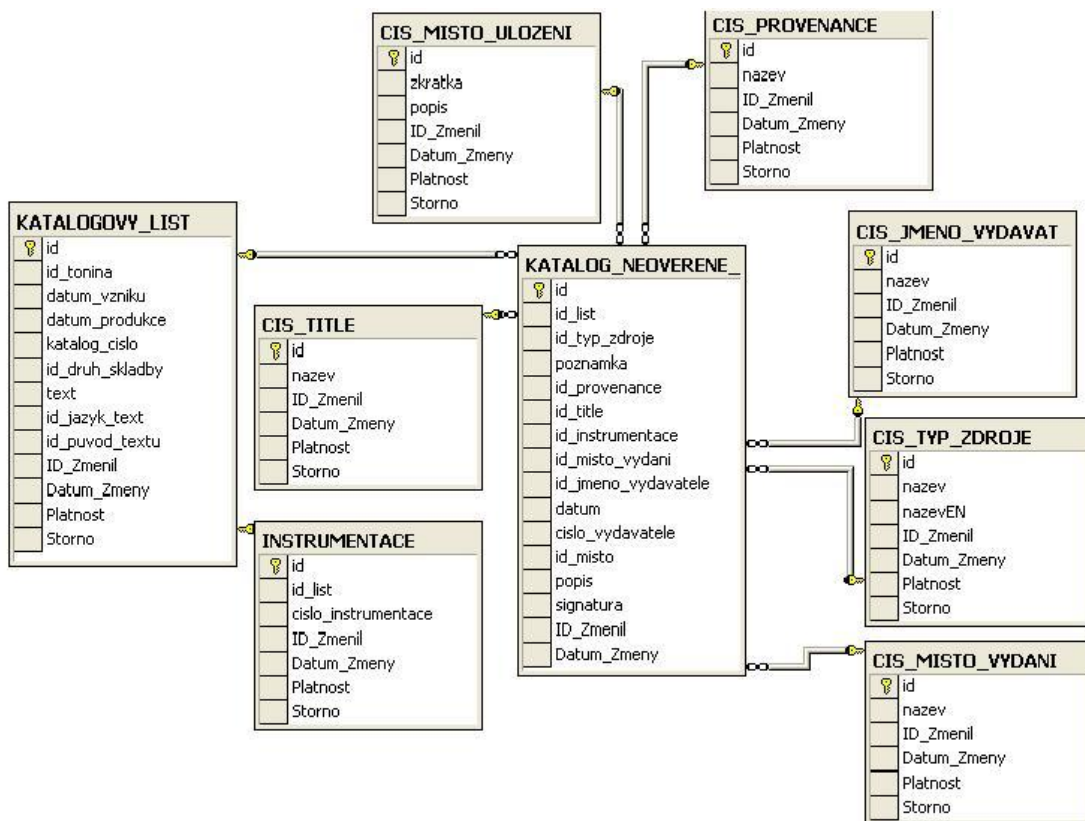
Dokumentace

Obr. 17. ER-Dokumentace

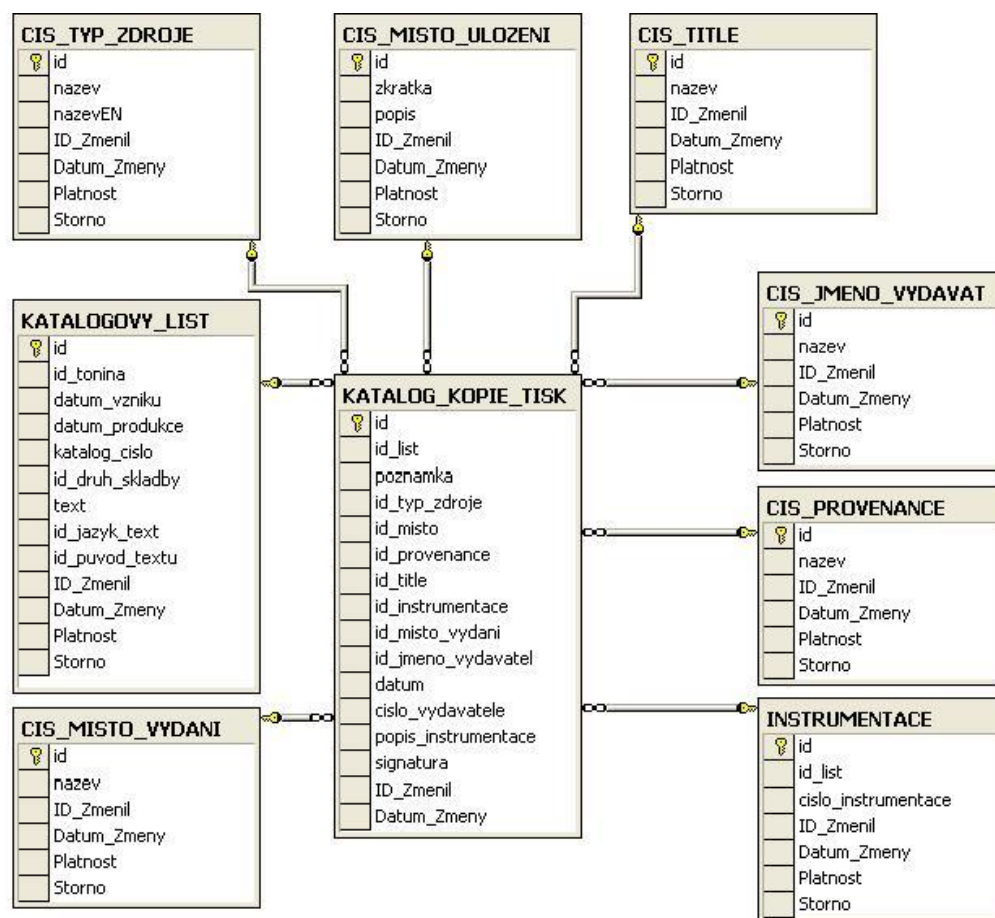
Každá skladba má svojí dokumentaci, či dokumentace. V tabulce KATALOG_DOKUMENTACE je uložený text dokumentace a literatura, ze které bylo čerpáno.

Nahrávka**Obr. 18. ER-Nahrávka**

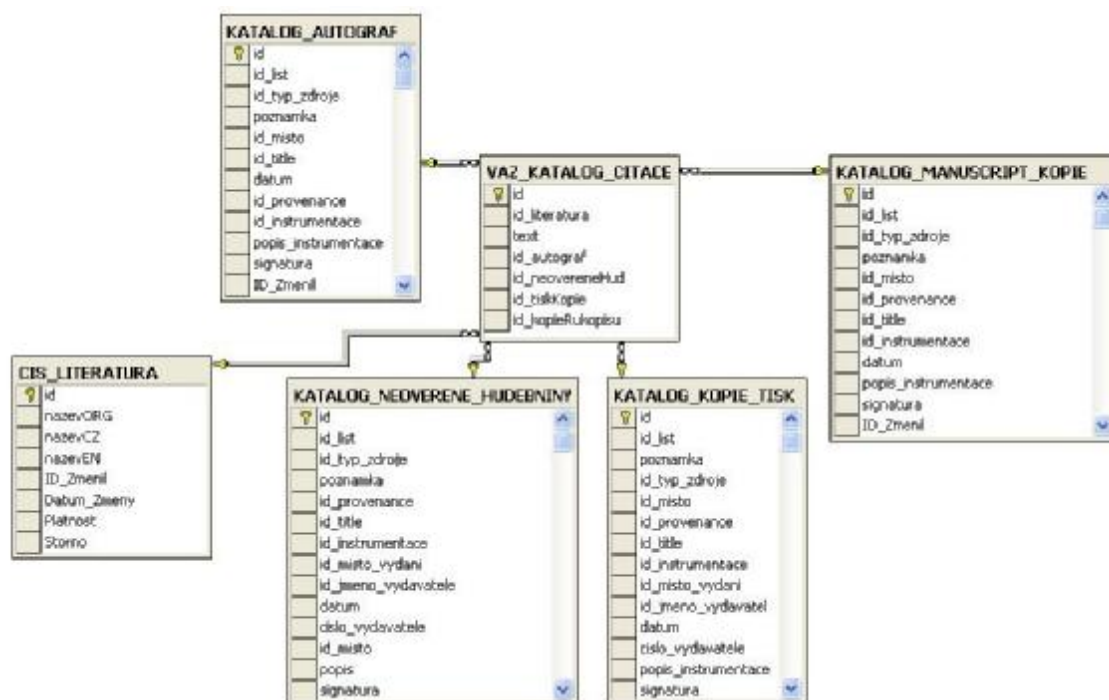
Pokud byla již někdy skladba nahrána, je tento údaj uložen v tabulce KATALOG_NAHRAVKA. Každá nahrávka má svůj druh (např. CD, kazeta, LP) a nahrávací společnost, kde byla nahrávka vytvořena.

Neověřené hudebniny

Obr. 19. ER-Neověřené hudebniny

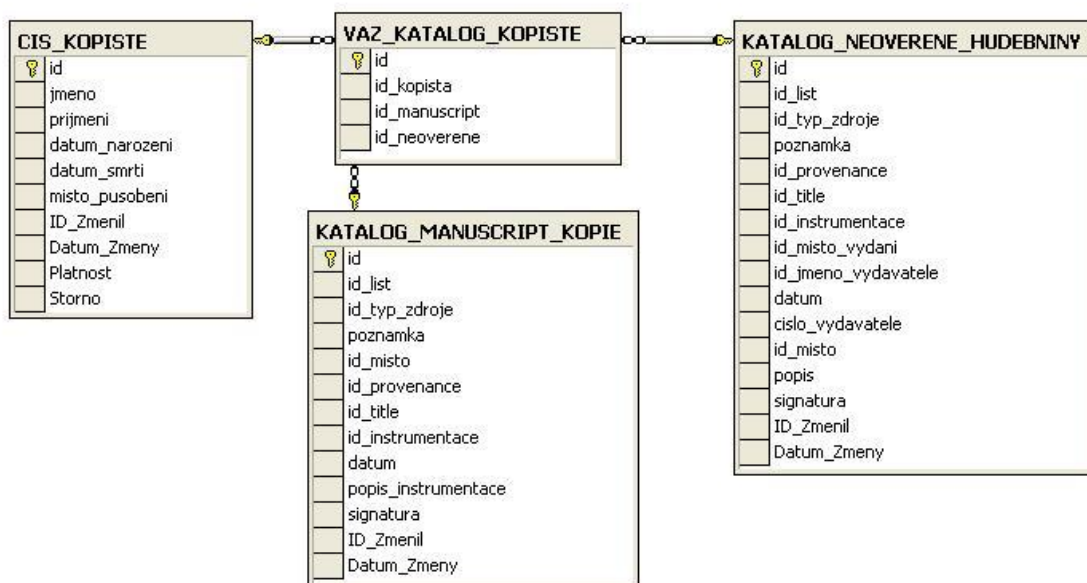
Tištěná kopie rukopisu*Obr. 20. ER-Tištěná kopie rukopisu*

Informace o tištěné kopii rukopisu jsou uloženy v tabulce KATALOG_KOPIE_TISK. Každá skladba může mít více těchto kopií, proto relace 1:n. Kopie má svoji instrumentaci, vydavatele, provenanci, místo vydání, místo uložení a svůj název a zdroj.

Citace

Obr. 21. ER-Citace

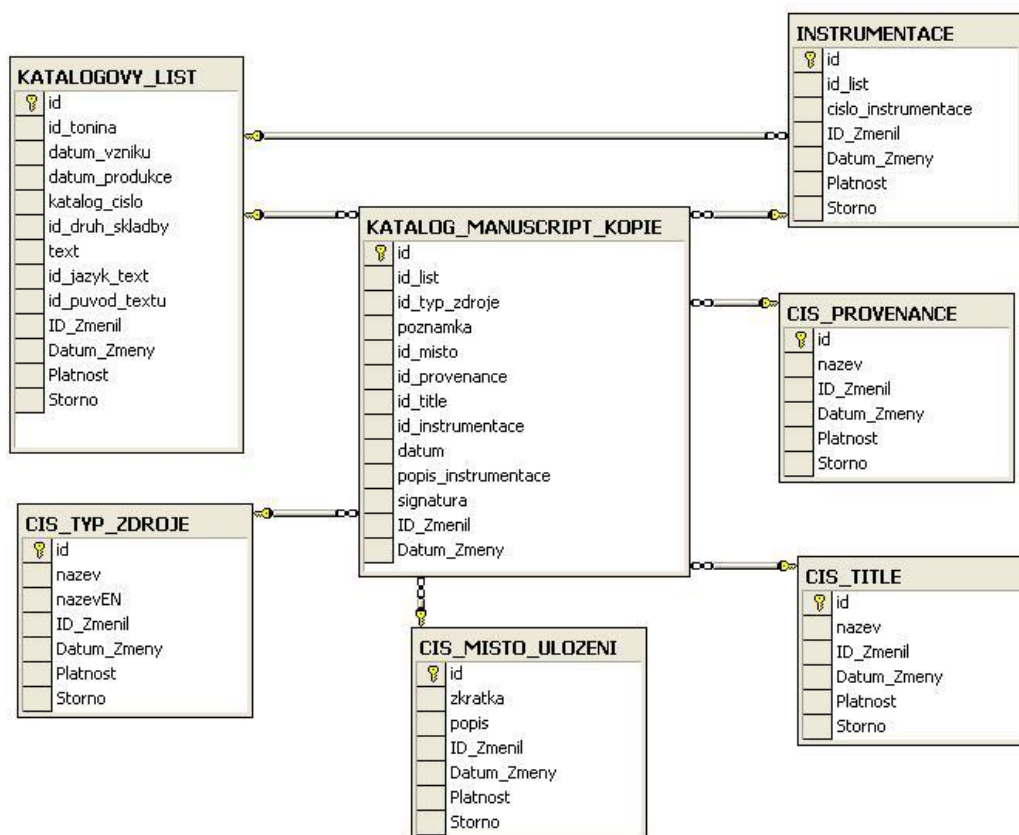
Citace z rukopisu skladby a jejich kopií má svůj text a zdroj literatury. Číselník literatury a jednotlivé tabulky jsou spojeny přes tabulku vazební VAZ_KATALOG_CITACE.

Kopisté

Obr. 22. ER-Kopisté

Jména kopistů, kteří vytvářeli ručně psané kopie jsou známa a uložena v tabulce CIS_KOPISTE. Tato tabulka je pak s ostatními spojena přes tabulku vazební VAZ_KATALOG_KOPISTE.

Manuskript kopie



Obr. 23. ER-Manuskript kopie

Kopie manuskript je kopie, psaná ručně. Údaje o ní jsou uloženy v tabulce KATALOG MANUSKRIP KOPIE.

13 FRAMEWORK APLIKACE

Vytvořený framework je komplexní systém pro vývoj webových informačních systémů založených na platformě ASP.NET 2.0 (Microsoft .NET Framework verze 2.0, v současnosti BETA 2). Je implementován jako sada tříd, které:

- Rozšiřují (dědí) standardní webové komponenty .NET (System.Web.UI.WebControls)
- Automatizují datové operace s formuláři (okno seznam-detail, změna editačního stavu, automatické naplnění komponent daty i uložení do databáze)
- Provádějí autentizaci a autorizaci uživatele, ověřování přístupových práv ke konkrétním objektům a operacím

Vlastní zdrojový kód frameworku je napsán v jazyce C#. Aplikace využívající framework ze tříd dědí.

Framework předpokládá spolupráci se standardní relační databází (např. MS SQL Server 2000, MSDE 2000). Vzhledem k automatizovaným datovým operacím musí použitý datový model splňovat několik nutných požadavků:

- Každá nevazební i vazební entita (tabulka) musí obsahovat jednoduchý primární klíč (nazvaný ID s automatickým generováním hodnoty). Díky tomu je možné adresovat jakýkoliv záznam z libovolné entity z databáze použitím právě jednoho klíče. Složené klíče ve vazebních entitách je možno realizovat jako cizí klíče s přidaným unique constraint.

Každá tabulka musí obsahovat další 4 povinné sloupce:

- ID_Zmenil – cizí klíč do tabulky uživatelů, identifikující uživatele, který záznam naposledy změnil
- Datum_Zmeny - datum a čas této poslední změny (implementovaný pro SQL Server jako GetDate())
- Platnost - 'A'/'N' udávající, zda je záznam validní/smazaný (pro nově přidané záznamy default 'A')
- Storno - 'A'/'N' udávající, zda je záznam stornovaný/validní (pro nově přidané záznamy default 'N')

14 NEJDŮLEŽITĚJŠÍ ČÁSTI KÓDU APLIKACE

V následujících odstavcích budou popsány některé části frameworku a příklady kódu.

Pomocný kód – fdAsp.Web.cs

Tato část kódu slouží k usnadnění psaní kódu. (např. NullToStr()), případně pro specializované činnosti, využívané další částmi systému (např. tvorba výrazů pro filtrování využité na seznamových oknech), získávání aktuálního času a jiné.

Následující příklad ukazuje třídu pro filtrování záznamů:

```
static class Filtering
{
    static public string StrLeft(string s, int len)
    {
        if (len <= 0)
            return "";
        else if (s.Length < len)
            return s;
        else
            return s.Substring(0, len);
    }

    static public string StrRight(string s, int len)
    {
        if (len <= 0)
            return "";
        else if (s.Length < len)
            return s;
        else
            return s.Substring(s.Length - len);
    }

    static public string ReplaceDecimal(string n)
    {
        return n.Replace(',', '.', '.');
    }

    static public string MSSQLDateTime(DateTime d)
    {
        return "\"" + d.ToString("yyyy-mm-dd hh:nn:ss") + "\"";
    }

    static public void AddFilterElement(ref string fil, string field,
    int datatype, string value)
    {
        string newfil = "";
        bool first = false, pack = false;
        string OP; // =, <>, <, >, <=, >=
        string val;
        string log; // AND, OR
        string s, t, v1, v2;
        int i1, i2, m;
        DateTime dt;
```

```
bool doSkip;

if ((value.Trim()).Length == 0) return;

if (fil != "")
{
    fil += " AND ";
    first = true;
}

value = value.Replace('*', '%');
s = value.ToUpper() + " AND";

// rozdeleni filtru na operandy a hodnoty
while (s != "")
// --- BEGIN --- of while
{
    i1 = s.IndexOf("AND");
    i2 = s.IndexOf("OR");

    log = "";
    if ((i1 > 0) && ((i2 <= 0) || (i1 < i2)))
    {
        log = "AND";
        t = (StrLeft(s, i1 - 1)).Trim();
        s = (StrRight(s, s.Length - i1 - 3)).Trim();
    }
    else if ((i2 > 0) && ((i1 <= 0) || (i2 < i1)))
    {
        log = "OR";
        t = (StrLeft(s, i2 - 1)).Trim();
        s = (StrRight(s, s.Length - i2 - 3)).Trim();
    }
    else
    {
        s = "";
        goto Skip;
    }

    v1 = (StrLeft(t + " ", 1)).Trim();
    v2 = (StrLeft(t + " ", 2)).Trim();
    m = t.Length;

    if (v2 == "<=")
    {
        OP = "<=";
        val = (StrRight(t, m - 2)).Trim();
    }
    else if (v2 == ">=")
    {
        OP = ">=";
        val = (StrRight(t, m - 2)).Trim();
    }
    else if ((v2 == "<>") || (v2 == "!="))
    {
        OP = "<>";
        val = (StrRight(t, m - 2)).Trim();
        goto Cont;
    }
    else if (v1 == "<")
    {
        OP = "<";
        val = (StrRight(t, m - 1)).Trim();
    }
}
```

```

else if (v1 == ">")
{
    OP = ">";
    val = (StrRight(t, m - 1)).Trim();
}
else if (v1 == "=")
{
    OP = "=";
    val = (StrRight(t, m - 1)).Trim();
}
else
{
    // default operator
    if (datatype == 0)
        OP = "LIKE";
    else
        OP = "=";
    val = t;
}
}

Cont:

doSkip = false;
switch (datatype)
{
    case 0: // varchar
        if ((val != "") && (val[val.Length-1] != '%'))
            val += "%";

        val = field + " " + OP + " '" + val + "'";
        break;

    case 1: // datetime
        if (DateTime.TryParse(val, out dt))
        {
            if ((OP == "=") && (dt == dt.Date))
            {
                // je specifikovan jeden den bez casu,
                tak z toho vykouzlime vyraz pro interval 24 hodin namisto konkretniho casu
                00:00.00
                val = "(" + field + ">=" +
                    MSSQLDateTime(dt) + " AND " + field + "<" + MSSQLDateTime(dt.AddDays(1))
                    + ")";
            }
            else
            {
                val = field + " " + OP + " " +
                    MSSQLDateTime(dt);
            }
        }
        else
        {
            doSkip = true;
            break;
        }

    case 2: // number
        val = field + " " + OP + " " +
            ReplaceDecimal(val);
        break;
}

if (doSkip) goto Skip;
if (s != "") val = "(" + val + ") " + log + " ";
if (newfil != "")
{
    val = "(" + val + ")";
}

```

```

        pack = true;
    }
    newfil += val;

Skip:
;

// --- END --- of while
}

if (pack) newfil = "(" + newfil + ")";
fil += newfil;
if (first) fil += ")";
}
}

```

Webová stránka - *fdAsp.Web.Page*

Představuje předka webové stránky v ASP.NET, který doplňuje funkčnost stránky např. o generování dynamického obsahu při prvním načtení, nebo automatické doplňování titulku stránky jednotlivých sekcí.

Další příklad ukazuje funkci, která vypisuje chybové hlášení pomocí javaskriptu, pokud se na stránce vyskytne chyba:

```

protected override void OnError(EventArgs e)
{
    String errorMsg = Page.Server.GetLastError().Message;
    Response.Write(String.Format(@"
<script language='javascript'>
    window.alert("Na stránce nastala chyba: \n {0}");
    history.back();
</script>", errorMsg.Replace('\n', '\r').Replace("\n",
"\n").Replace("\r", "\\r") ));
    Context.ClearError();
}

```

Seznam (GridView) a Detail (Panel)

Tyto dvě části dohromady implementují základní prvek uživatelského rozhraní systému, formulář seznam – detail. GridView zobrazuje seznam položek, jejichž detail si můžeme pomocí ovládacího prvku Panel zobrazit.

V příkladu se nachází funkce, která vytváří SQL příkaz pro vložení záznamu do databáze.:

```

public SqlCommand createInsertData(string table, string listControls,
bool ignore, string otherCommand, string otherValues)
{
    SqlCommand com = new SqlCommand();
    string command = "insert into " + table + "(" + otherCommand;
    string values = " values (" + otherValues;
    string field = "";
    for (int i = 0; i < Controls.Count; i++)

```

```

    {
        Control c = Controls[i];
        if ((
            c.Visible &&
            (c is IEditDataControl) && (
                (listControls.Length == 0) ||
                ((listControls.Length > 0) && (
                    ignore && (listControls.IndexOf(c.ID + "*") < 0))
                || // ignorovat vsechny controly uvedene v seznamu
                    (!ignore && (listControls.IndexOf(c.ID + "*") >=
0))) // pouzit pouze controly uvedene v seznamu
            ))
        {
            IEditDataControl edc = ((IEditDataControl)c);
            if (edc.DBValue != DBNull.Value)
            {
                field = edc.DataField;
                command += field + ", ";
                values += "@" + field + ", ";
                com.Parameters.Add("@ " + field, edc.DBType).Value
= edc.DBValue;
            }
        }

        if (Common.NullToStr(LinkField) != "")
        {
            command += LinkField + ", ";
            values += "@" + LinkField + ", ";
            com.Parameters.Add("@ " + LinkField, SqlDbType.Int).Value
= LinkID;
        }

        command += "Datum_Zmeny, ID_Zmenil";
        values += "GetDate(), " +
fdAsp.Web.MembershipProvider.UzivatelID.ToString();
        com.CommandText = command + ") " + values + ") select
scope_identity()";
        return com;
    }

```

Detailové tlačítko – fdAsp.Web.DetailButton.cs

Tento objekt dědí z třídy Button (tlačítko), přebírá tedy všechny jeho funkce, ale doplňuje své nové. Umožní po stisknutí tlačítka spuštění modálního okna, které obsahuje detailní informace o určitém záznamu. Umožňuje tedy zobrazení relace 1:n.

Příklad zobrazuje funkci, která do html kódu k vlastnostem tlačítka připsí atribut onclick a ten po stisknutí tlačítka zavolá javaskriptovou funkci pro otevření detailového okna s požadovaným obsahem.

```
protected override void AddAttributesToRender( HtmlTextWriter writer )
{
    base.AddAttributesToRender(writer);

    if (Enabled)
    {
        string url = DetailPage + "?LinkField=" + LinkField +
"&LinkID=" + linkID.ToString() + "&FormCaption=" +
Convert.ToBase64String(Encoding.UTF8.GetBytes(FormCaption.ToString()));

        writer.AddAttribute(HtmlTextWriterAttribute.Onclick,
"javascript:RunDetail('" + url + "', '" + FormCaption.ToString() + "')");
    }
}
```

15 ALGORITMUS PŘEVODU MIDI SOUBORU NA KÓDY SHK A RISM AII

15.1 Popis problému

Vytváření kódů SHK a RISM AII „ručně“ je časově náročné. Proto se hledal způsob, jak celý proces urychlit a zautomatizovat. Nejideálnější řešení tedy je, aby uživatel jen zadal hudební incipit ve tvaru MIDI souboru a aplikace sama kód SHK a RISM AII vygenerovala.

15.2 Generování kódu

Po načtení MIDI souboru aplikací, předá tato úplnou cestu k souboru na disku jako parametr pro konstruktor třídy, která následně MIDI soubor zpracuje. Obsah souboru MIDI, nebo jeho část, se v binární podobě načte do proměnné, kde je uchovávána pro potřeby generování kódu.

Jakmile je soubor, nebo jeho část, načten, začnou se postupně testovat jeho byty a hledat určité známé hodnoty. Prohledávání souboru je realizováno hned několika funkcemi, které mají každá jiný úkol. Funkce pro hledání not, pro nalezení pozice v souboru, pro zjištění předznamenání notového incipitu, pro určení délky noty atd. Jakmile je nashromážděn dostatek informací pro vygenerování kódu SHK, či RISM AII, zpracuje další funkce získaná data a předá je pro další využití jako textový řetězec. Ten je potom použit jako textový parametr ovládacího prvku ASP.NET a zobrazen uživateli. Uživatel pak může vygenerovaný kód podle svých dalších případných potřeb dál upravovat.

15.3 Ukázky z třídy pro generování – implementace v C#

15.3.1 Zjištění délky noty – deltačas

Funkce přebírá pole typu integer, které určuje pozici noty/pauzy, rozšířené o indikaci, zda se jedná o notu, či pauzu.

```

public int[] GetLenNote(int[] poz)
{
    int DelkaDeltaCasu = 0;
    int poziceDeltaCasu = 0;
    int j = 0;

    if (HistPoz != startPoz) DelkaDeltaCasu = poz[0] - (HistPoz +
2);
    else DelkaDeltaCasu = 1;
    poziceDeltaCasu = poz[0] - DelkaDeltaCasu;
    int[] deltacas = new int[DelkaDeltaCasu+1];

    for (int i = poziceDeltaCasu; i < poz[0]; i++)
    {
        deltacas[j] = IntV[i];
        j++;
    }

    deltacas[deltacas.Length - 1] = poz[1]; //prida udaj o tom,
jestli je to pauza, delka noty

    return deltacas;
}

```

15.3.2 Nalezení Meta Eventu

Funkce přebírá tři parametry:

- Pole typu int, což je načtený MIDI soubor převedený do hodnot typu integer
- Další dva parametry určují jednoznačně event, jehož pozici v souboru hledáme

```

// nalezeni pozice zadaneho Meta-Eventu v soboru /dva parametry
public int SearchMetaEvent(int[] Ix, int compr1, int compr2)
{
    for (int i = 0; i < Ix.Length-3; i++)
    {
        if (Ix[i] == compr1 && Ix[i+1] == compr2) return i+2;
    }

    return -1;
}

```

15.3.3 Určení předznamenání incipitu

```

public string GetPocPredSHK()
{
    int retKB = 0;

    int id = SearchMetaEvent(IntV,
KeySignature[0], KeySignature[1], KeySignature[2]);

    if (id != -1)
    {
        KB = Buffer[id+1];
        DurMol = Buffer[id+2];
    }
}

```

```
    }

    if (KB >= 1 && KB < 8) retKB = Krizky[KB - 1];
    else if (KB >= 249 ) retKB = Becka[(KB - 256)*(-1) - 1];
    else if (KB == 0) retKB = 0;

    return retKB.ToString();
}
```

15.3.4 Určení taktu incipitu

```
//-----vraci takt-----

public string GetTimeSignature()
{
    int id = SearchMetaEvent(IntV, TimeSignature[0],
TimeSignature[1], TimeSignature[2]);
    if (id != -1)
    {
        nominator = Buffer[id + 1];
        denominator = (int) Math.Pow((double)2, (double)Buffer[id
+ 2]);

    }

    return nominator.ToString() + denominator.ToString();
}
```

16 POPIS APLIKACE A JEJÍHO OVLÁDÁNÍ

16.1 Popis aplikace

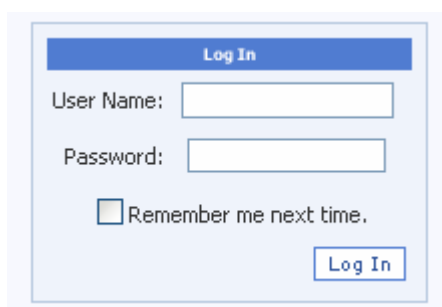
Tématický katalog děl Vinzenze Maschka zachovává pro přehlednost a snadnou orientaci členění do skupin podle typu skladby. Skupiny jsou označeny římskými číslicemi. Skladby jsou v každé skupině označeny arabskými číslicemi – využívá se toho, že čísla jsou neomezená a mohou se neustále v budoucnu přidávat.

Aplikace se skládá ze tří tří menších částí:

- Katalog – možnost editování a přidávání nových záznamů
- Číselníky – možnost editace číselníků
- Administrace – přidávání nových uživatelů, editace rolí a práv

16.2 Přihlášení do aplikace

Každý uživatel, který chce s aplikací pracovat, musí být správně přihlášen. Přihlášení se provádí v přihlašovacím formuláři, který je zobrazený na obrázku níže.



Obr. 24. Přihlašovací formulář

Uživatel zadá své uživatelské jméno a heslo, je zde také možnost zapamatování si uživatele pro další přihlášení. Po úspěšném přihlášení získá uživatel přístup na stránku s výběrem skupin skladeb.

I	Symfonie
II	Klavírní koncerty
III	Skladby pro klavír nebo cembalo s doprovodem dechových nástrojů
IV	Koncerty
V	Divertimenta
VI	Tance a pochody pro orchestr
VII	Smyčcové kvartety
VIII	Trio pro klavír housle [nebo flétnu] a violoncello
IX	Kvarteta pro klavír nebo cembalo, housle, violu a violoncello
X	Kvinteta pro klavír nebo cembalo, dvoje housle, violu a violoncello
XI	Sonáty pro klavír s doprovodem houslí
XII	Sonáty pro klavír nebo cembalo na dvě ruce
XIII	Sonáty pro klavír nebo cembalo na čtyři ruce
XIV	Variace pro cembalo nebo klavír
XVI	Ostatní klavírní kompozice
XVII	Tance a pochody pro klavír
XVIII	Skladby pro skleněnou harmoniku
XIX	Balety a pantomimy
XX	Chrámové instrumentální skladby
XXI	Mše
XXII	Requiem
XXIII	Litanie
XXIV	Stationes
XXV	Ostatní chrámové skladby
XXVI	Opery
XXVII	Kantáty
XXVIII	Árie
XXIX	Písně
XXX	Adaptace hudebně-dramatických děl pro klavír na dvě ruce
XXXI	Adaptace hudebně-dramatických děl pro smyčcové kvarteto
XXXII	Adaptace ostatních děl pro klavír
XXXIII	Adaptace duchovních skladeb

Obr. 25. Seznam druhů skladeb

Uživatel zvolí jednu ze skupin a může začít editovat a přidávat záznamy.

16.3 Vytvoření/Editace záznamu

	Druh skladby	Katalogové číslo	Tónina	Text	Původ textu	Datum složení	Datum produkce
->	XXV	1	C dur	Ad hanc solennitatem	Tractus communione festorum B. Mariae Virg. - Liber usualis	s. a.	
->	XXV	2	Es dur	Ad hoc festum properate	Ex Offertorio pro Dominica IV post Pascha	ante 1813	
->	XXV	3	C dur	Ad Mariam cum tribulatione	Ex Offertorio pro Dominica IV post Pascha	ante 1806	
->	XXV	4	C dur	Ad te Domine	Offertorium-Dominica X. post Pentecosten - Liber usualis	s. a.	
->	XXV	5	B dur	Adorare non cessabo	Ex Offertorio pro Dominica IV post Pascha	s. a.	
->	XXV	6	C dur	Ah amare fugio	Ex Offertorio pro Dominica IV post Pascha		
->	XXV	7	Es dur	Alleluja	Ex Offertorio pro Dominica IV post Pascha	ante 1829	
->	XXV	8	C dur	Ama sperando	Ex Offertorio pro Dominica IV post Pascha		
->	XXV	9	G dur	Angelis suis	Graduale-Dominica I in Quadragesima - Liber usualis	s. a.	
->	XXV	10	As dur	Angelis suis	Graduale-Dominica I in Quadragesima - Liber usualis	s. a.	
269 záznamů							
Nový Upravit Uložit Zpět Smazat							

Obr. 26. Seznam záznamů v databázi

V okně se seznamem vložených záznamů, může uživatel zvolit jeden se záznamů. S označeným může provádět tyto akce:

- Upravit
- Smazat

Pokud se rozhodne přidat nový záznam, musí uživatel postupovat takto:

- Zvolit volbu *nový* na panelu
- Po stisknutí tlačítka se zpřístupní vkládání jednotlivých položek záznamu

Nový		Upravit		Uložit		Zpět		Smazat	
Označení:		...	Číslo:		Tónina:		...	Datum vzniku:	
Datum prod.:			Jazyk:		...				
Text:									
Původ textu:									...
Skladatelé		Dokumentace		Reference		Incipit		Instrumentace	
Autograf		Tištěná kopie		Opis		Neověřené hudebniny		Nahrávka	

Obr. 27. Vytvoření nového záznamu

- Uživatel vloží všechna potřebná data a stiskne tlačítko uložit, tím se v tomto záznamu zpřístupní editace detailních informací o skladbě

Obr. 28. Vytvoření detailní informace o záznamu

- Uživatel zvolí jeden z detailů a pokračuje tak jako u hlavního záznamu

Obr. 29. Okno detailu

Pokud má uživatel práva administrátora, může v katalogu vytvářet nové uživatele, editovat číselníky, role a práva. Vše je založeno na stejném principu, jako výše popsany průběh přidávání nového záznamu, nebo jeho editace.

ZÁVĚR

V bakalářské práci jsem se snažil přistoupit k vývoji webové aplikace v prostředí ASP.NET se všemi náležitostmi, které by měl standardní vývoj dle moderních metod obsahovat. Musel jsem se tedy nejprve detailně obeznámit s nástroji pro návrh a vývoj webových aplikací a tyto posléze aplikovat na zadání práce. Přestože na začátku vývoje se použití těchto nástrojů jevílo jakožto časově neefektivní, průběh rozrůstání projektu jednoznačně potvrdil, že čím větší péče je věnována aplikaci ve fázi návrhu, tím snazší jsou veškeré pozdější zásahy a celková udržitelnost životnosti aplikace. Podobně lze hodnotit také použití vývojového frameworku, který mi vedoucí práce kvůli následnému snazšímu vývoji a případné budoucí použitelnosti pro práci v ASP.NET doporučil vyvinout.

Na aplikaci byl kladen nejen požadavek manipulace s textovými daty, ale také požadavek pracovat s notovými ukázkami ve standardizované podobě MIDI formátu. Bylo tedy třeba nejen pochopit strukturu tohoto standardu, ale navíc se pokusit o sestavení algoritmu, který by dokázal převést tento digitální formát do alfanumerických kódů SHK a RISM AII, které se používají pro evidenci a vyhledávání notových ukázek. Lze konstatovat, že se podařilo dosáhnout automatického generování obou kódů, přestože se v průběhu vývoje několikrát zdálo, že to nebude proveditelné.

Bakalářská práce mi umožnila seznámit se s širokou problematikou procesu vývoje aplikací a jejím výsledkem je webový katalog, který bude nadále prakticky používán pro vědecké účely. Projekt by měl být zveřejněn do konce roku 2005 na adrese www.maschek.cz

SEZNAM POUŽITÉ LITERATURY

- [1] *ROOT.CZ: Slovníček pojmů* [online]. [cit. 2005-08-20]. Dostupný z WWW:
<<http://www.root.cz/slovnicek/>>
- [2] *NaWEBka: Historie jazyka SQL* [online]. [cit. 2005-08-20]. Dostupný z WWW:
<<http://www.rydval.cz/phprs/view.php?cislocclanku=2005123125>>
- [3] *KOSEK.CZ: Jak pracují databáze na webu. Co je to databáze.* [online]. [cit. 2005-08-25]. Dostupný z WWW: < <http://www.kosek.cz/clanky/iweb/12.html>>
- [4] *CPRESS.CZ: CGI skripty.* [online]. [cit. 2005-08-25]. Dostupný z WWW:
<<http://www.cpress.cz/knihy/tcp-ip-bezp/CD-dalsi/CD-cgi/cgi.htm>>
- [5] Petr Zajíc: *Historie a budoucnost.* [online]. *Linuxsoft.cz: PHP 2004* – [cit. 2005-08-27]. Dostupný z WWW: < http://www.linuxsoft.cz/article.php?id_article=171>
- [6] Jaroslav Lhoták: *ASP – cesta k dynamickému Internetu.* [online]. *Živě.CZ: ASP 2001* – [cit. 2005-08-27]. Dostupný z WWW:
<<http://www.zive.cz/h/Programovani/AR.asp?ARI=29033>>
- [7] *Živě.CZ: Měníme stránky ASP na stránky ASP.NET.* [online]. [cit. 2005-08-27]. Dostupný z WWW: < <http://www.zive.cz/h/Programovani/AR.asp?ARI=125126>>
- [8] Zelenka, Petr: *Proces vývoje webové aplikace(Specifikace požadavků).* [online]. *Interval.cz: WebML 2003* - [cit. 2005-08-28]. Dostupný z WWW:
< <http://interval.cz/clanek.asp?article=3677>>
- [9] Zelenka, Petr: *Proces vývoje webové aplikace(Návrh aplikace).* [online]. *Interval.cz: WebML 2003* - [cit. 2005-08-28]. Dostupný z WWW:
< <http://interval.cz/clanek.asp?article=3748>>
- [10] *Wikipedie: Objektově orientované programování.* [online] [cit. 2005-08-28]. Dostupný z WWW:
<http://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD>

- [11] *Wikipedie: Databáze*. [online] [cit. 2005-08-28]. Dostupný z WWW: <<http://cs.wikipedia.org/wiki/Datab%C3%A1ze>>
- [12] GUÉRIN, Robert. *Velká kniha MIDI. Standardy, hardware, software*. Brno: Computer Press, 2004. 340 s. ISBN 80-722-6985-2
- [13] SHARP, John, JAGGER, Jon. *Microsoft Visual C# .NET krok za krokem*. Praha: Mobil Media a.s., 2002. 651 s. ISBN 80-86593-27-4
- [14] Stein, René: *Návrh aplikace v jazyce UML* [online] 2003 – [cit. 2005-08-28] Dostupný z WWW: <<http://interval.cz/clanek.asp?article=2783>>
- [15] Knihzdar.cz: *Tematické hudební katalogy a databáze hudebních pramenů* [online]. [cit. 2005-08-28]. Dostupný z WWW: <http://www.knihzdar.cz/kzv/2004-03/romportlova_temhudkat.htm>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ADP	Access Database	Projekt aplikace Access. Projects
SMF	Standard Midi File	Formát MIDI souboru.
MSDE	Microsoft Database Engine	Alternativa pro MS SQL Server.
CLR	Common Language Runtime	Část .NET FRAMEWORKU.
API	Aplication Programming Interface	Rozhraní pro programování aplikací.

SEZNAM OBRÁZKŮ

Obr. 1 Příklad ER - diagramu	15
Obr. 2. Komunikace mezi klientem, serverem a CGI skriptem.....	19
Obr. 3. Příklad ER diagramu.....	41
Obr. 4. Značka případu.....	44
Obr. 5. Značka aktora.....	44
Obr. 6. Příklad užití UseCase diagramu.....	45
Obr. 7 SHK – Incipit č.1.....	56
Obr. 8 SHK – Incipit č.2.....	56
Obr. 9 RISM – Incipit č.1	59
Obr. 10 RISM – Incipit č.2	59
Obr. 11. UseCase diagram.....	65
Obr. 12. Sekvenční diagram.....	67
Obr. 13. ER - Katalogový list	69
Obr. 14. ER - Vazba katalogový list a skladatel	70
Obr. 15. ER-Instrumentace.....	71
Obr. 16. ER-Autograf.....	72
Obr. 17. ER-Dokumentace	73
Obr. 18. ER-Nahrávka.....	74
Obr. 19. ER-Neověřené hudebniny	75
Obr. 20. ER-Tištěná kopie rukopisu	76
Obr. 21. ER-Citace.....	77
Obr. 22. ER-Kopisté.....	78
Obr. 23. ER-Manuskript kopie	79
Obr. 24. Přihlašovací formulář	90
Obr. 25. Seznam druhů skladeb	91
Obr. 26. Seznam záznamů v databázi.....	92
Obr. 27. Vytvoření nového záznamu.....	92
Obr. 28. Vytvoření detailní informace o záznamu.....	93
Obr. 29. Okno detailu	93

SEZNAM TABULEK

Tab. 1. Příklad hlavičky SMF pro formát 0	48
Tab. 2. Soubor MIDI.....	52

SEZNAM PŘÍLOH

- P I CD – ROM - Tématický katalog Vinzenze Maschka (projekt ADP, webová aplikace, skripty struktury databáze)