VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

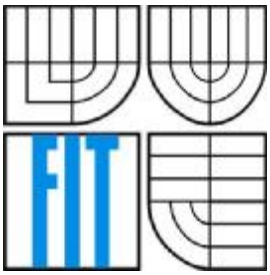
WEBOVÁ APLIKACE PRO VYHODNOCENÍ LETECKÝCH TRAS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ROMAN LANGER
AUTHOR

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE PRO VYHODNOCENÍ LETECKÝCH TRAS

WEB APPLICATION FOR THE PLANE ROUTE PLANNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN LANGER

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LADISLAV RUTTKAY

BRNO 2009

Abstrakt

Cílem bakalářské práce je implementovat webovou aplikaci pro vyhodnocení leteckých tras. Webová aplikace poskytne uživateli letecké trasy seřazené podle ceny nebo času s přestupy v reálném čase podle požadavků uživatele. Konkrétní lety získává systém z webové služby. Aplikace je implementována v jazyce C# s pomocí .Net Framework s návrhovou metodologií objektově orientovaného programování.

Abstract

The goal of the bachelor's thesis is an implementation of web application for the plane route planning. The web application gives users plane routes sorted by time or price of the journey in real time according to requirements of user. Application gets particulars routes from web service. The solution is implemented in C# with .Net Framework with object-oriented programming.

Klíčová slova

Webová aplikace, webová služba, letecké trasy, algoritmus, OOP, C#, .Net Framework, SQL

Keywords

Web application, web service, plane routing, algorithm, OOP, C#, .Net Framework, SQL

Citace

Langer Roman : Webová aplikácia pre vyhodnotenie leteckých trás, bakalárska práca, Brno, FIT VUT v Brně, 2009

Webová aplikácia pre vyhodnotenie leteckých trás

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Ladislava Ruttkaya.
Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Roman Langer
18.5.2009

Poďakovanie

Týmto by som sa chcel veľmi pekne poďakovať hlavne svojmu odbornému vedúcemu celej práce Ing. Ladislavovi Ruttkayovi za technickú podporu a pomoc pri zostavovaní celého projektu a tejto odbornej dokumentácie.

Obsah

1 Úvod

Táto dokumentácia popisuje návrh a algoritmy, ktoré som použil na riešenie mojej bakalárskej práce, ktorá sa nazýva Webová aplikácia pre vyhodnotenie leteckých trás, ktorá je implementovaná v jazykoch C# a ASP.NET.

Podmienkou vhodnej leteckej prepravy v dnešnej dobe je kvalitný systém na vyhľadanie efektívneho spojenia. Dobré určenie leteckej trasy podľa jeho špecifického výberu totiž zaručí užívateľovi najefektívnejší spôsob prepravy z jedného miesta na iné. K efektívnej preprave osôb takisto významne prispieva aj vhodné zadanie všetkých požadovaných informácií o doprave. Za týmto účelom sa využívajú systémy, ktoré by užívateľovi pomohli vybrať si najvýhodnejšie letecké spojenie. Tento systém by mal vyhľadať spojenie podľa užívateľových predstáv, to znamená, že by mal od užívateľa dostať čo najviac informácií o jeho požiadavkách na let, medzi ktoré patria dátum a čas odletu a príletu, cena, počet možných prestupov, minimálny a maximálny čas na prestup a výber trasy buď podľa ceny, alebo dĺžky letu s prestupmi.

Cieľom tohto projektu je uľahčiť prácu užívateľom a poskytnúť im komplexné riešenie na báze informačného systému, prístupného pomocou webového rozhrania, ktoré systematicky dokáže vyhodnotiť tieto užívateľove požiadavky v reálnom čase. Výsledky zobrazuje v poradí podľa toho, ako si užívateľ zažiadal (buď podľa ceny, alebo podľa trvania letu).

Prvá kapitola opisuje analýzu projektu. Sú tu popísané nástroje, ktoré sú využité v implementácii. Je tu popísané objektovo orientované programovanie, jazyk UML a iné nástroje.

V druhej kapitole sa popisujú nástroje, ktoré sa používajú na navrhovanie programov a sú nimi ER diagram a Use case diagram. Táto kapitola taktiež obsahuje popisy algoritmov, ktoré sa využívajú na vyhľadanie najkratšej (najideálnejšej) trasy a algoritmus, pomocou ktorého zobrazujeme hľadané výsledky na našom webovom portáli.

V tretej kapitole je načrtnuté GUI webového portálu a implementácia systému, popisuje sa tu testovanie a dôvody testovania. Ďalej nasleduje popis možného rozšírenia, ako systém vylepšiť. Poslednou kapitolou je záver, kde opisujem môj prínos a zhodnotenie systému.

2 Analýza projektu

V tejto kapitole sú popísané všetky implementačné prostriedky, ktoré sú využité v projekte.

2.1 Objektovo orientované programovanie

Objektovo orientované programovanie (**OOP**) je druh programovania, pri ktorom sa používajú štruktúry, zvané objekty, ktoré môžu existovať ako samostatné programové entity. Tieto objekty majú svoje vlastnosti, metódy a funkcie, pomocou ktorých objekt vykonáva určité činnosti, na ktoré bol naprogramovaný. Existuje niekoľko softwarových konceptov, ktoré sa považujú za ústredné body práce s objektmi [1]:

- Zapúzdrenie – zoskupenie súvisiacich ideí do jednej jednotky, na ktorú je možné sa neskôr odkázať jedným názvom.
- Skrývanie informácií a implementácií – je využitie zapúzdrenia k obmedzeniu externej viditeľnosti určitých informácií alebo implementačných rozhodnutí, ktoré sú pre štruktúru zapúzdrenia interné.
- Zachovanie stavu.
- Identita objektov – vlastnosť, podľa ktorej je možné každý objekt identifikovať a pracovať s ním ako so samostatnou softwarovou entitou.
- Správa – prostriedok, ktorým odosielajúci objekt obj1 prenáša cieľovému objektu obj2 požiadavku, aby objekt obj2 použil jednu zo svojich metód.
- Trieda – je šablóna, podľa ktorej sa vytvárajú objekty. Každý objekt má rovnakú štruktúru a chovanie ako trieda, ktorej je inštanciou.
- Dedičnosť (napr. triedy U z triedy T) je nástroj, pomocou ktorého si trieda U implicitne definuje všetky atribúty a operácie triedy T, akoby boli definované priamo v triede U. T sa označuje za nadradenú triedu U. U je potom podriadená trieda T.
- Mnohotvarosť (polymorfizmus) – je prostriedok, pomocou ktorého môže byť jeden názov operácie alebo atribútu definovaný na základe viac ako jednej triedy a môže nadobúdať rôznych implementácií v každej z týchto tried. Prostredníctvom mnohotvarosti môže atribút alebo premenná ukazovať (obsahovať identifikátor) na objekty rôznych tried v rôznych okamihoch
- Všeobecnosť – je taká konštrukcia triedy T, ktorá umožňuje dodanie jednej alebo viacerých jej interne používaných tried až za behu aplikácie.

2.2 Jazyk UML

UML (Unified Modeling Language) je v softwarovom inžinierstve grafický jazyk pre vizualizáciu, špecifikáciu, navrhovanie a dokumentáciu programových systémov. UML ponúka štandardný spôsob zápisu návrhu systémov vrátane konceptuálnych prvkov, ako sú businnes procesy a systémové funkcie, tak aj konkrétnych prvkov, ako sú príkazy programovacieho jazyka, databázové schémy a znovu použiteľné programové komponenty. Tento jazyk podporuje objektovo orientovaný prístup k analýze, návrhu a popisu programových systémov.

UML operuje s pojmom *pohľad* (*view*). Pohľad systému je projekcia systému na jeden z jeho relevantných aspektov. Takáto projekcia sa zameriava na príslušný aspekt a ignoruje ostatné. Je vhodné vytvárať niekoľko rôznych pohľadov na ten istý systém. Pohľady nad systémom sú potom modelované prostredníctvom vhodných nástrojov (modelov) poskytovaných UML. Môžeme rozlišovať tieto základné pohľady [2]:

- *Štrukturálny pohľad (structural view)* popisuje vrstvu medzi objektmi a triedami, ich asociácie a možné komunikačné kanály.
- *Pohľad chovania (behavior view)* popisuje, ako systémové komponenty (objekty) na seba vzájomne pôsobia, vzájomne sa ovplyvňujú, a charakterizuje reakcie na vonkajšie systémové operácie.
- *Dátový pohľad (data view)* popisuje stavy systémových komponentov (objekty) a ich väzby.
- *Pohľad rozhrania (interface view)* je zameraný na zapúzdrenie systémových častí a ich potenciálne použitie okolím systému.

Jazyk UML ponúka niekoľko základných diagramov:

- **štruktúrne diagramy:**
  - diagram tried (class diagram),
  - diagram komponentov (*component diagram*),
  - diagram zloženej štruktúry (*composite structure diagram*),
  - diagram nasadenia (*deployment diagram*),
  - diagram balíčkov (*package diagram*),
  - diagram objektov (*object diagram*), nazýva sa aj diagram inštancií.
- **diagramy chovania :**
  - diagram aktivít (*activity diagram*),
  - diagram prípadov použitia (*use case diagram*),
  - stavový diagram (*state machine diagram*).

- diagramy interakcie :
  - sekvenčný diagram (*sequence diagram*),
  - diagram komunikácie (*communication diagram*),
  - diagram prehľadu interakcií (*interaction overview diagram*),
  - diagram časovania (*timing diagram*).

2.3 Vývojové prostredie

Pri tvorbe rozsiahlejších aplikácií je vhodné zvoliť si vývojové prostredie, v ktorom sa aplikácia bude tvoriť. Pre túto aplikáciu je najvhodnejšie Microsoft Visual Studio. Tento vývojový prostriedok je hlavným integrovaným vývojovým prostredím firmy Microsoft. Visual Studio obsahuje debugger, editor kódu, ktorý sprehľadňuje kód vhodným formátovaním, a ďalšie utility pre návrh aplikácií.

2.4 Jazyk C#

Tento programovací jazyk je novým jazykom uvedeným spoločnosťou Microsoft vrátane .NET Framework a Visual Studiom. Ide o čisto objektovo orientovaný jazyk, ktorý sa stal interným vývojovým nástrojom platformy .NET. Firma Microsoft ho zaradila do tzv. rodiny C jazykov z dôvodu syntaktickej podobnosti s jazykom C++. V skutočnosti jazyk C# pripomína viac jazyk Java a v istých pohľadoch nesie niektoré myšlienky Visual Basicu.

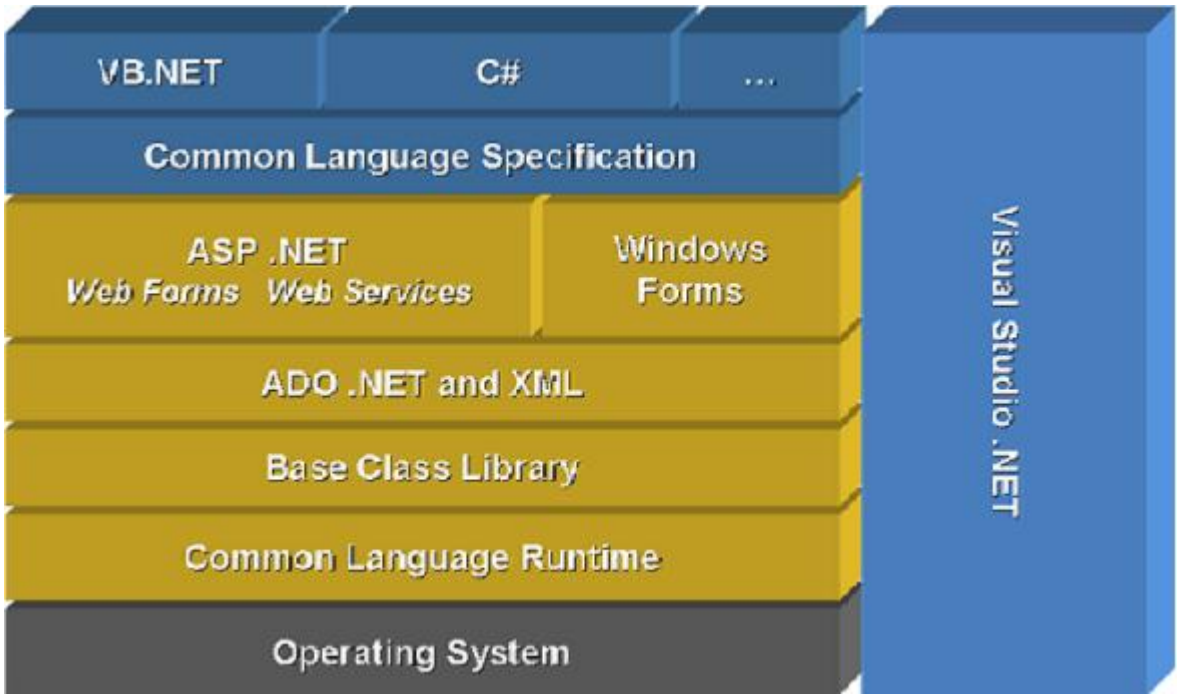
Základné charakteristiky tohto jazyka [3] :

- je čisto objektovo orientovaný,
- obsahuje natívnu podporu komponentového programovania,
- používa jednoduchú dedičnosť s možnosťou násobnej implementácie rozhrania,
- vedľa členských dát a metód pridáva vlastnosti a udalosti,
- správa pamäti je realizovaná prostredníctvom Garbage collector,
- podporuje spracovanie chýb pomocou výnimiek,
- zaisťuje typovú bezpečnosť a podporuje riadenie verzií,
- podporuje atribútové programovanie,
- zaisťuje hlbokú integráciu so stávajúcim kódom na binárnej aj zdrojovej úrovni.

2.5 .NET Framework

.Net framework je platforma na vytváranie a chod aplikácií. Jej zásadnými komponentami sú spoločný jazykový behový modul (Common Language Runtime - CLR) a knižnica tried .NET (Framework Class Library - FCL). CLR abstrahuje služby operačného systému a slúži ako vykonávacie jadro pre riadenie aplikácie – aplikácie, ktorej všetky akcie podliehajú schváleniu u CLR. FCL poskytuje objektovo orientované rozhranie API, do ktorého riadené aplikácie zapisujú [4].

Microsoft .NET Framework podporuje tzv. Common Type Specification (CTS). Týmto má programátor na výber programovací jazyk, ktorý mu najviac vyhovuje. Môže si napísať triedy v C# a vytvárať inštancie z nich vo VB.NET alebo naopak. Je možné si zvoliť niektorý z jazykov: C#, C++, VB, J#, JScript.



Obrázok 2.1: Architektúra .Net platformy [11]

2.5.1 ASP.NET (Active server pages)

ASP.NET je celkom novou vybudovanou technológiou bežiacou nad .NET Framework a využívajúca všetky vylepšenia, ktoré prináša. Ide o kvalitné prostredie, zvýši sa výkon a zníži sa riziko pádu. Nezanedbateľným prínosom je jednoduchosť, s akou je možné aplikácie vytvárať. ASP.NET má dva typy programových modulov, a to sú webové formuláre a webové služby.

Prostredie ASP.NET prináša mnoho nových myšlienok [3]:

- silná podpora vyrovnávacej pamäti,
- podpora a riadenie správy stavu aplikácie,
- podpora riadenia bezpečnosti webových aplikácií,
- konfigurácia ASP.NET,
- nasadenie a upgrade webových aplikácií.

2.5.2 ADO.NET (ActiveX Data Objects)

ADO.NET je od základu prepracovaná technológia ActiveX Data Objects, ktorá je veľmi často používaná v súčasnom asp. Zmeny zasiahli takmer každú časť, aj keď nájdeme veľa paralel medzi starou a novou verziou. Zmenil sa objektový model, pomenovanie jednotlivých častí a ich umiestnenie v menných priestoroch. Aplikácie sú čím ďalej, tým viac orientované do prostredia Internetu, ktoré je špecifické tzv. odpojenými riešeniami. Znamená to, že spojenie medzi klientom a serverom má charakter krátkych nepravidelných a nedeterminovateľných komunikácií. ADO.NET bolo navrhnuté od základu s víziou práce v týchto odpojených scenárov [4].

2.6 Kaskádové štýly (CSS)

CSS (*Cascading Style Sheets*) sú vytvorené na formátovanie internetových dokumentov (napr. fontov, farieb, okrajov). Hlavným dôvodom je oddeliť štruktúru (X)HTML od vzhľadu, ktorý sa navrhuje len pomocou štýlov a logika programu v dokumente je oddelená. CSS možno presunúť do externých súborov. Týmto sa zmenší dátová veľkosť a tým pádom je možné jedným súborom zmeniť štýl celej stránky. CSS zaručuje vo všetkých prehliadačoch rovnaké vykresľovanie [5].

2.7 Jazyk SQL

SQL (*Structured Query Language* - štrukturovaný dotazovací jazyk) je štandardizovaný dotazovací jazyk pre prácu s dátami v relačných databázach používaný najmä na správu. Taktiež podporuje procedurálne funkcie.

Základné syntaktické konštrukcie jazyka SQL sú nasledovné :

- **Jazyk definície dát (DDL – Data Definition Language)** Príkazy DDL sa používajú k vytváraniu, úpravám alebo odstraňovaniu databázových objektov (napr. tabuliek, pohľadov, schém, domén, triggerov a pod.). Sú to hlavne CREATE, ALTER a DROP.

- **Jazyk pre riadenie dát (DCL – Data Control Language)** Príkazy DCL umožňujú riadiť, kto bude mať prístup k určitým objektom databázy. V jazyku DCL môžeme povoľovať alebo obmedzovať prístup pomocou príkazov GRANT a REVOKE.
- **Jazyk pre manipuláciu s dátami (DML – Data Manipulation Language)** Príkazy DML sa používajú k prehliadaniu, pridávaniu, úpravám alebo odstraňovaniu dát uložených v databáze. Hlavnými kľúčovými slovami sú SELECT, INSERT, UPDATE a DELETE [6].

Ďalej sa používajú aj príkazy pre riadenie transakcií (START TRANSACTION, COMMIT, ROLLBACK).

2.7.1 SQL Server

SQL Server je všestranné, integrované a komplexné riešenie pre dáta, ktoré prispieva ku zvýšeniu výkonnosti užívateľov v celej organizácii poskytovaním bezpečnejšej, spoľahlivejšej a produktívnejšej platformy pre podnikové dáta a aplikácie [7].

2.8 Webové služby

Webová služba je aplikácia, ktorá [4]:

- Beží na webovom serveri.
- Vystavuje webové metódy volajúcim.
- Naslúcha požiadavkám HTTP predstavujúcom príkazy volajúcej webové služby
- Vykonáva webové metódy a vracia výsledky.

Základnými stavebnými kameňmi webových služieb sú nasledujúce operácie a pre ne navrhnuté protokoly [3]:

- **Publikovanie služby.** Akonáhle je služba naprogramovaná a pripravená k zverejneniu, treba ju na Internete uverejniť. Existujú dve špecifikácie slúžiace k publikovaniu webových služieb. Prvá sa nazýva UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION(UDDI). Slúži na publikovanie a vyhľadávanie webových služieb. Druhou špecifikáciou je SOAP DISCOVERY(DISCO). Slúži k vytváraniu zoznamov služieb dostupných na webovom serveri alebo serveroch.
- **Popis služby.** Akonáhle vieme, na ktorom serveri je služba dostupná, zaujíma nás, ako sa so službou komunikuje. Ekvivalentne v COM technológiách zisťujeme, aké rozhranie komponenta implementuje, ako sa jednotlivé metódy nazývajú a aké majú parametre. U webových služieb tomu nie je inak. Ďalšou špecifikáciou je WEB SERVICE DESCRIPTION LANGUAGE – WSDL. Ide o špecifikáciu definujúcu gramatiku XML

pre popis web služby. VSDL súbor obsahuje popis všetkých metód a parametrov, ktorými služba na konkrétnom serveri disponuje. Na ich základe sú nástroje ako Visual Studio.NET schopné generovať tzv. proxy triedy používané klientskymi aplikáciami pre komunikáciu s webovou službou.

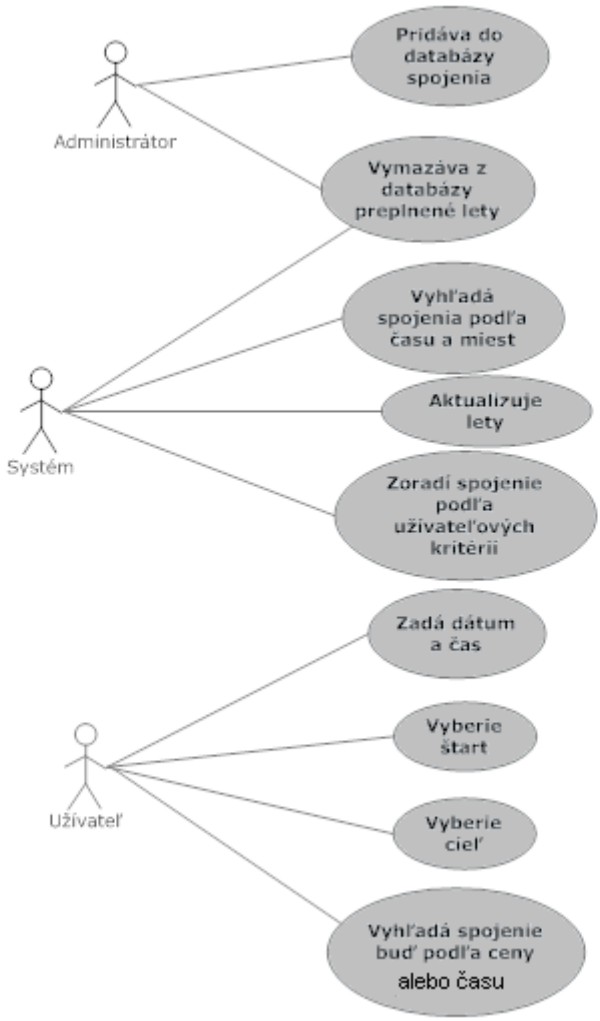
- **Prenosové protokoly.** Poslednou otázkou ostáva, ako vyvolať metódy služieb a predávať informácie na server a späť klientovi. Tu na tejto najnižšej úrovni sú používané už existujúce internetové protokoly ako HTTP, HTTPS alebo SMTP. Pomocou nich sú spracovávané požiadavky a odpovede. Výhodou je, že porty TCP/IP sú pre tieto protokoly otvorené vo všetkých aktívnych prvkoch a firewalloch. Pre samotné vyvolanie metód a odovzdávanie dát sa môže použiť aj SIMPLE OBJECT ACCES PROTOCOL(SOAP).

3 Návrh riešenia

Táto kapitola je zameraná na návrh riešenia pomocou vývojových prostriedkov a sú tu opísané algoritmy na riešenie úloh prehľadávania stavového priestoru.

3.1 Use case diagram

Use-Case diagram (diagram prípadov užitia) je dôležitou súčasťou pri tvorbe informačných systémov. Poskytuje nám rýchlu predstavu o všetkých funkciách systému a ich účastníkoch. Diagramy prípadov užitia modelujú hranice systému, jeho účastníkov, prípady užitia a interakcie medzi nimi a účastníkmi. V informačnom portáli na zobrazenie leteckej trasy sú účastníci užívateľ, resp. systém. Užívateľ má v ňom niekoľko možností. Všetky sú popísané v diagrame prípadov užitia, ktorý je znázornený na nasledovnom obrázku:



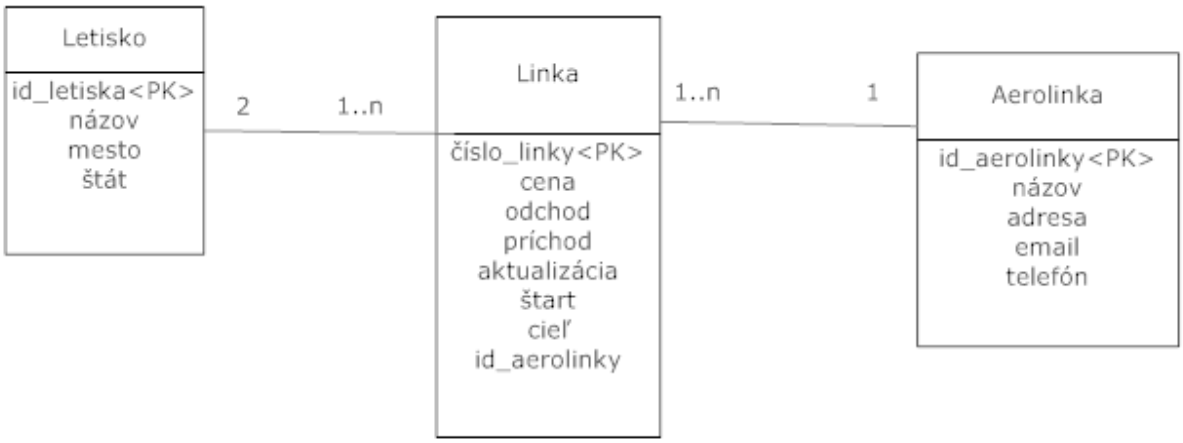
Obrázok 3.1: Use case diagram

Hlavným aktérom v use case diagrame je užívateľ, ktorý si vyberá dátum a čas letu, miesto odchodu a príchodu, taktiež si určí, či chce zobraziť výsledky podľa celkovej ceny alebo dĺžky letu.

Ďalším aktérom v use case diagrame je systém, ktorý vyhľadá spojenie podľa užívateľových kritérií. Taktiež sa stará o aktualizáciu systému a o vymazávanie preplnených letov, ktoré môže takisto vymazať aj administrátor. Administrátor pridáva jednotlivé lety do databázy.

3.2 ER diagram

Entity Relationship Diagram (ER) slúži k modelovaniu dát aplikačnej domény a ich vzťahov. ER diagram je sieťový model popisujúci návrh uložených dát v systéme na vyššiu úroveň abstrakcie. ER diagram je odlišný od diagramu dátových tokov (DFD), ktorý modeluje funkcie vykonávané navrhovaným systémom. ER diagram modeluje dáta, ktoré potrebujeme v systéme uchovávať, a vzťahy medzi týmito dátami. Návrh uloženia dát popisuje tento ER diagram:



Obrázok 3.2 ER Diagram

E-R diagram obsahuje nasledovné 4 entity:

- Letisko : táto tabuľka obsahuje atribúty id, názov, mesto a štát. Tieto atribúty najviac vystihujú každé letisko
- Linka : táto tabuľka obsahuje informácie o jednej linke z miesta na miesto a ako atribúty má číslo linky, cenu, vzdialenosť, odchod, príchod a čas aktualizácie
- Aerolinka : táto tabuľka vlastní údaje o prepravnej spoločnosti, a to id, názov, adresu, email a telefón

Každé spojenie sa môže skladať z niekoľkých liniek. Štart a cieľ je určený dvoma letiskami. Linka má jednoznačne určenú prepravnú spoločnosť a táto prepravná spoločnosť môže mať niekoľko liniek.

3.3 Algoritmy na prehľadanie stavového priestoru

3.3.1 Metóda rovnakých cien (UCS)

Metóda UCS (Uniform Cost Search) je algoritmus na prehľadávanie stavového priestoru, ktorý zvažuje skutočné ceny prechodov (kladné čísla) a skutočné ceny ciest (sú dané súčtom cien príslušných prechodov). Pre expanziu algoritmus vyberá zo zoznamu OPEN uzol s najmenším ohodnotením, tj. uzol s najnižšou cenou cesty.

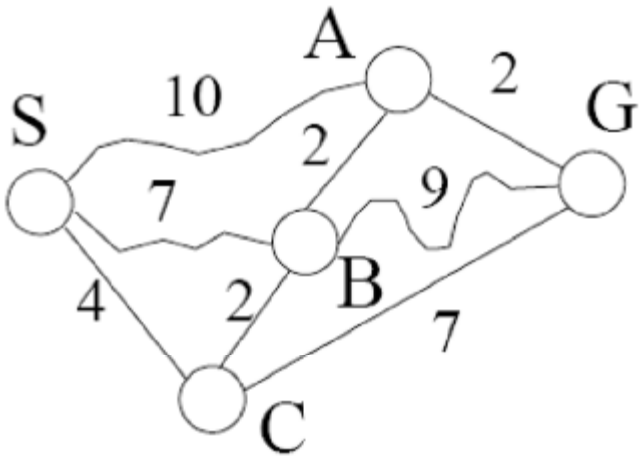
Základný algoritmus metódy UCS je nasledujúci [8] :

1. Zostroj zoznam OPEN (bude obsahovať všetky uzly určené k expanzii) a umiestni do neho počiatočný uzol vrátane jeho (nulového) ohodnotenia.
2. Ak je zoznam OPEN prázdny, úloha nemá riešenie a ukonči preto prehľadávanie ako neúspešné. Inak pokračuj.
3. Vyber zo zoznamu OPEN uzol s najnižším ohodnotením.
4. Ak je vybraný uzol uzlom cieľovým, ukonči prehľadávanie ako úspešné a vráť cestu od koreňového uzlu k uzlu cieľovému. Inak pokračuj.
5. Vybraný uzol expanduj, všetkých jeho následníkov vrátane ich ohodnotení umiestni do zoznamu OPEN a vráť sa na bod 2.

Ak je počet bezprostredných následníkov každého uzlu konečný, potom algoritmus UCS je úplný a optimálny, jeho časová i pamäťová náročnosť je exponenciálna - je daná výrazom $O(b^k)$, kde b je faktor vetvenia a k je koeficient získaný podielom ceny optimálneho riešenia C^* a najmenšieho prírastku ceny medzi dvoma uzlami $\Delta cmin$:

$$k = C^* / \Delta cmin$$

Príklad nájdenia najkratšej cesty:



Poradie expanzie	Zoznam OPEN
0	[((S),0)]
1	[((A,S),10),((B,S),7),**((C,S),4)**]
2	[((A,S),10),**((B,C,S),6)**,((G,C,S),11)]
3	[((G,C,S),11),**((A,B,C,S),8)**]
4	[**((G,A,B,C,S),10)**]

Obrázok 3.3 Metóda rovnakých cien [8]

3.3.2 Dijkstrov algoritmus

Dijkstrov algoritmus je jedným zo základných algoritmov teórie grafov. Jeho primárnym využitím je hľadanie najkratšej cesty v hranovo-ohodnotenom digrafe $G = (V, H, c)$. Tento graf pozostáva z množiny vrcholov V, množiny orientovaných hrán H a funkcie c, ktorá zobrazuje množinu hrán do množiny reálnych čísel. Teda pre ňu platí $H \rightarrow R$. Ďalším predpokladom je, aby $c(h) \geq 0$ pre všetky hrany h z množiny H. Jeho autorom je holandský matematik E. W. Dijkstra a typovo ide o algoritmus najkratšej cesty z jedného vrcholu (počiatok s) do ostatných, najčastejšie však do jedného konkrétneho (cieľ d).

Pre každý vrchol grafu G udržiava algoritmus tri symboly. Prvým je $t(v)$, ktorý zaznamenáva doteraz najkratšiu cestu z počiatku do vrcholu v. Druhým je $x(v)$, ktorý si pamätá predposledný vrchol

cesty *s – v*, teda vrchol, pomocou ktorého sa dostaneme do vrcholu *v* "najrýchlejšie". Tento symbol nie je priamo potrebný pre beh algoritmu, no má svoje využitie pri spätnej konštrukcii cesty z počiatku do cieľa (prípadne hociktorého iného vrcholu množiny *V*). Posledným symbolom je dvojstavová premenná *p(v)*, určujúca, či je doteraz najkratšia nájdená cesta *t* konečná alebo nie je. Ďalšou potrebnou charakteristikou bude riadiaci vrchol *r*.

Pred samotným behom je potrebné inicializovať horeuvedené symboly nasledovne:

- $t(v) = \infty$, pre všetky $v \neq s$; $t(s) = 0$,
- $x(v) = 0$, pre všetky v z V,
- $p(v) = false$, pre všetky $v \neq s$; $p(s) = true$,

a za riadiaci vrchol zvolíme počiatok, teda $r = s$.

Samotný algoritmus pozostáva z dvoch opakujúcich sa krokov. Prvý z nich kontroluje, či náhodou nie je riadiacim vrcholom vrchol *d*, teda cieľ. V takom prípade algoritmus končí a jeho výsledkom sú *t(d)*, dĺžka najkratšej *s – d* cesty a postupnosť vrcholov *s, ..., x(x(x(d))), x(x(d)), x(d), d*, čo je samotná cesta. Ak však podmienka, že riadiacim vrcholom je *d*, neplatí, vykonáme nasledujúci úkon: pre všetky hrany tvaru *(r, i)* z množiny *H*, pre ktoré platí, že $p(i) = false$ a $t(i) > t(r) + c(r, i)$ upravíme symbol *t(i)* na $t(r) + c(r, i)$ a značku *x(i)* nastavíme na *r*.

V druhom kroku nájdeme zo všetkých dočasne označených vrcholov *v* (platí pre ne $p(v) = false$) taký, ktorého *t(v)* je najmenšie. Tento vrchol prehlásime za nový riadiaci a jeho značku *p(v)* nastavíme na *true*, čo bude znamenať, že *t(v)* skutočne označuje najkratšiu cestu z vrcholu *s* do vrcholu *v*. Ak by nastala situácia, že vrcholov s minimálnym *t* je viac, môžeme vybrať ľubovoľný z nich. Ďalej pokračujeme vo výpočte prvým krokom.

Ak na konci výpočtu obsahuje *t(d)* nekonečno (a $x(d) = 0$), tak je zrejmé, že spojenie *s – d* neexistuje.

Vo všetkých behoch prvého kroku sa vykoná maximálne $|H| = m$ operácií, keďže každú hranu použijeme najviac raz. V druhom kroku stačí skontrolovať maximálne $|V| = n$ vrcholov. Výpočtová zložitosť Dijkstrovho algoritmu je teda $O(n^2)$. [9]

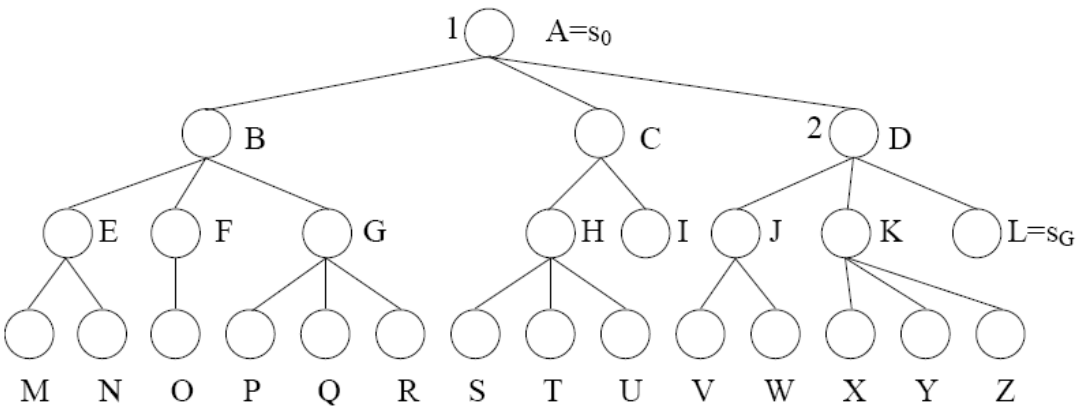
3.3.3 Metóda slepého prehľadávania do hĺbky (DFS)

Základný algoritmus metódy DFS (Depth First Search) je nasledujúci [8]:

1. Zostroj zásobník OPEN (bude obsahovať všetky uzly určené k expanzii) a umiestni doň počiatočný uzol.
2. Ak je zásobník OPEN prázdny, potom úloha nemá riešenie a preto ukonči prehľadávanie ako neúspešné. Inak pokračuj.
3. Vyber z vrcholu zásobníka OPEN prvý uzol.

4. Ak je vybraný uzol uzlom cieľovým, ukonči prehľadávanie ako úspešné a vráť cestu od koreňového uzlu k uzlu cieľovému (vracia sa postupnosť stavov, alebo operátorov). Inak pokračuj.
5. Vybraný uzol expanduj, všetkých jeho bezprostredných nasledovníkov umiestni do zásobníka OPEN a vráť sa na bod 2.

Algoritmus DFS nie je, na rozdiel od vyššie popísaného algoritmu UCS, ani úplný, ani optimálny. Časová náročnosť algoritmu DFS zostáva exponenciálna – teoreticky je daná výrazom $O(b^m)$, kde *b* je faktor vetvenia a *m* je maximálna prehľadávaná hĺbka stromu. Pamäťová náročnosť je lineárna a je daná výrazom $O(b^m)$; v prípade opakovania generovania už expandovaných stavov môže však hodnota *m* byť nekonečná.



Poradie expanzie	Zásobník OPEN
0	[[A,-]]
1	[[D,A,-],[C,A,-],[B,A,-]]
2	[[L,D,A,-],[K,D,A,-],[J,D,A,-],[C,A,-],[B,A,-]]
Riešenie:	[L,D,A,-]: $A \Rightarrow D \Rightarrow L$

Obrázok 3.4 Metóda slepého prehľadávania do hĺbky [8]

Metóda DFS v základnom prevedení však môže v jednoduchých prípadoch aj zlyhať, a to napr. v takom prípade, že generuje rovnaké uzly, a riešenie preto končí neúspechom. Z tohto uvedeného dôvodu je táto metóda prakticky nepoužiteľná, a je preto nutné prakticky vždy použiť jej modifikáciu – úpravu bodu 5 algoritmu DFS tak, aby boli eliminované uzly – predchodcovia generovaného uzlu a uzly, ktoré sú už v zásobníku OPEN uložené:

5. Vybraný uzol expanduj a umiestni do zásobníka OPEN všetky jeho bezprostredných následníkov, ktorí v tomto zásobníku ešte nie sú a ktorí nie sú ani predkami generovaného uzlu.

3.3.4 Metóda obmedzeného prehľadávania do hĺbky (DLS)

Tento algoritmus vynašli Patrick Doherty, Witold Lukaszewicz a Andrzej Szalas [10]. Je to obdoba algoritmu slepého prehľadávania (DFS). Ak riešime úlohu, pri ktorej dokážeme odhadnúť hĺbku riešenia, môžeme problém s opakujúcim sa generovaním rovnakých stavov vyriešiť obmedzením hĺbky prehľadávania.

Algoritmus DLS má potom tvar [8]:

1. Zostroj zásobník OPEN a umiestni doň počiatočný uzol s označením hĺbky (0).
2. Ak zásobník OPEN je prázdny, potom úloha nemá riešenie, a ukonči preto prehľadávanie ako neúspešné. Inak pokračuj.
3. Vyber z vrcholu zásobníka OPEN prvý uzol.
4. Ak je vybraný uzol uzlom cieľovým, ukonči prehľadávanie ako úspešné a vráť cestu od koreňového uzlu k uzlu cieľovému (vracia sa postupnosť stavov alebo operátorov). Inak pokračuj.
5. Ak je hĺbka vybraného uzlu menšia než zadaná maximálna hĺbka, tak tento uzol expanduj, všetkým jeho bezprostredným následníkom priraď hĺbku o jednu väčšiu než je hĺbka expandovaného uzlu a umiestni ich do zásobníku OPEN. Inak pokračuj.
6. Vráť sa na bod 2.

Algoritmus DLS nie je, rovnako ako algoritmus DFS, ani úplný, ani optimálny. Časová náročnosť algoritmu DLS zostáva exponenciálna - je daná výrazom $O(b^l)$, kde b je faktor vetvenia a l je maximálna povolená hĺbka prehľadávania, pamäťová náročnosť je lineárna a je daná výrazom O(bl).

3.3.5 Metóda postupného zanorovania do hĺbky (IDS)

Ak nie je možné stanoviť hĺbku riešenia a nemáme k dispozícii dostatok pamäte pre použitie metódy BFS, môžeme sa pokúsiť vyriešiť tento problém použitím metódy postupného zanorovania do hĺbky. Princíp tejto metódy spočíva v opakovanom použití metódy DLS s postupným zvyšovaním hĺbky prehľadávania.

Algoritmus je nasledujúci [8]:

1. Nastav hĺbku prehľadávania na hodnotu 1.
2. Použi metódu DLS. Ak skončí táto metóda úspechom (nájdením cesty), skonči tiež úspechom a vráť nájdenú cestu. Inak pokračuj.
3. Ak dosiahne hĺbka prehľadávania maximálnu hodnotu, skonči neúspechom. Inak inkrementuj hĺbku prehľadávania a vráť sa na bod 2.

Metóda postupného zanorovania do hĺbky sa zdá byť na prvý pohľad veľmi neefektívna, pretože pri každom ďalšom zanorení úplne opakuje prehľadávanie, ktoré už vykonala pri predchádzajúcej hĺbke. Počet expandovaných uzlov pre riešenie, ktoré je v hĺbke *d*, je dané vzťahom:

$$d.b + (d-1).b^2 + (d-2).b^3 + \ldots 1.b^d$$

(koreňový uzol sa expanduje *d*-krát, uzly v hĺbke 1 sa expandujú (*d*-1)-krát, až konečné uzly v hĺbke *d* sa expandujú iba raz), z neho je však zrejmé, že zložitosť metódy je stále $O(b^d)$, tzn., že metóda postupného zanorovania do hĺbky má rovnakú zložitosť ako metóda BFS. Pamäťová zložitosť metódy postupného zanorovania do hĺbky je daná vzťahom $O(b^d)$.

4 Implementácia

V tejto kapitole je uvedené grafické užívateľské rozhranie a popis implementácie webového portálu. Implementácia začala návrhom grafického užívateľského rozhrania, ktoré sa implementovalo pomocou kaskádových štýlov. Neskôr sa navrhlo uloženie dát podľa E-R diagramu. Nakoniec sa implementoval celý systém, ktorý postupne nadobúdal funkčnosť.

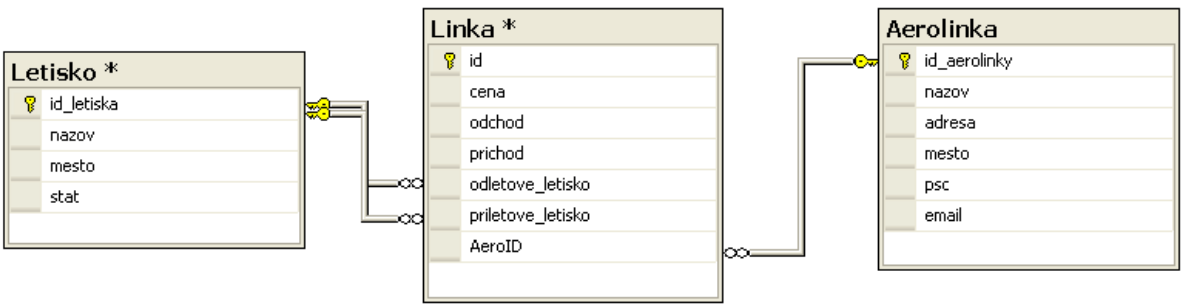
4.1 Trojvrstvová architektúra

Pri implementácii bola použitá typická trojvrstvová architektúra, čo znamená, že aplikácia sa skladá z troch na sebe nezávislých vrstiev. Takto sa vytvorí ľahko udržiavateľná a rozšíriteľná aplikácia. Tieto vrstvy sú nasledovné:

1. Dátová vrstva – triedy, ktoré získavajú dáta buď z databázy, alebo z webových služieb.
2. Aplikačná vrstva – triedy, ktoré obsahujú aplikačnú logiku, tvoria prepojenie medzi prezentačnou a dátovou vrstvou
3. Prezentačná vrstva – obsahuje triedy, ktoré vytvárajú webové alebo formulárové rozhranie pre komunikáciu s užívateľom

4.1.1 Dátová vrstva

V tejto vrstve sú vytvorené triedy na náplň objektu, ktorý obsahuje všetky lety. Jeden let je reprezentovaný ID, odletovým a príletovým letiskom, kódom týchto letísk, cenou letu, ID aerolinky a názvom, ktorý je sprostredkovateľom príslušného letu, a dátumom odletu a príletu. Tieto atribúty obsahuje objekt *Odlety*, ktorý je konštruovaný v triede *Odlety.cs*. Trieda *Vyberdat.cs* obsahuje metódy na náplň objektu, a to *Spoj* a *Service*. Metóda *Spoj* vyberie dáta z databázy, metóda *Service* tieto dáta získa z webových služieb. Na nasledujúcom obrázku je znázornený graf tejto databázy.



Obrázok 4.1. Graf databázy

Príklad pripojenia k databázy v zdrojovom kóde C#:

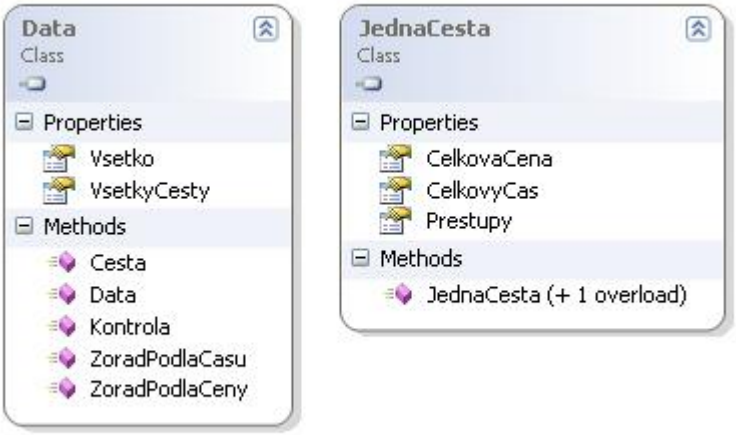
```
string connectionString = ConfigurationSettings.AppSettings["Pripojenie"];
SqlConnection cn = new SqlConnection(connectionString);

try
{
      string sql = "Select * from Linka";
      SqlCommand cmd = new SqlCommand(sql, cn);
      cn.Open();
      SqlDataReader reader = cmd.ExecuteReader();
      while (reader.Read())
      {
            Odlety let = new Odlety();
            let.ID = (Int32)reader["ID"];
            let.Cena = (Int32)reader["cena"];

            ...
      }
}
catch (SqlException)
{
      let.ID = null;
      let.Cena = null;
}

finally
{
      cn.Close();
}
```

Algoritmus 4.1. Pripojenie k databáze

Na predchádzajúcom príklade vidíme, že na pripojenie k databáze sa využíva trieda SqlConnection, ktorej objekty majú ako atribút reťazec pripojenia. V ňom je uvedená cesta k danej databáze. Objekty triedy SqlCommand sa využívajú na pripojenie k databáze, konkrétne cez reťazec pripojenia a požiadavku na databázu. Najprv prebehne test, či sa na danú databázu vôbec podarí pripojiť. Ak nie, prebehne ošetrenie pomocou výnimky. Výnimky sú bližšie popísané v kapitole 4.6. Dáta sa v systéme získavajú pomocou triedy SqlDataReader. Daný algoritmus číta údaje v cykle, ktorý skončí akonáhle objekt triedy SqlDataReader už neobsahuje žiadne dáta. Nakoniec sa spojenie s databázou uzavrie.

4.1.2 Aplikačná vrstva

Táto vrstva slúži na manipuláciu s dátami. V tomto projekte sú v tejto vrstve obsiahnuté triedy *JednaCesta.cs* a *Data.cs*, ktoré sú znázornené na nasledujúcom obrázku spolu s metódami, ktoré vytvárajú, a s vlastnosťami týchto tried:



Obrázok 4.2 Diagram tried v aplikačnej vrstve

Trieda *JednaCesta.cs* vytvorí objekt, ktorý obsahuje celkovú cenu letu, celkový čas letu a indexy, ktoré obsahujú čísla jednotlivých letov, z ktorých je spojenie vytvorené. Trieda *Data.cs* obsahuje metódy *Filter, Cesta, ZoradPodlaCeny, ZoradPodlaCasu a Kontrola.* Tieto metódy sú opísané v nasledujúcej podkapitole. Dáta z dátovej vrstvy sa získajú nasledovne:

```
VyberDat h = new VyberDat();
vsetko = new List<Odlety>();

if (o == 1)
    vsetko = h.Service();
else vsetko = h.Spoj();
```

Algoritmus 4.2. Získanie dát z dátovej vrstvy

Predchádzajúci úsek zdrojového kódu opisuje získanie dát do aplikačnej vrstvy. Najprv sa vytvorí inštancia triedy *VyberDat.cs* a objekt *vsetko* . Ak užívateľ požiada aktuálne lety, vyvolá sa metóda *Service*, ktorá načíta aktuálne lety z webovej služby. V opačnom prípade sa zavolá metóda *Spoj*, ktorá čerpá testovacie údaje z databázy.

4.1.3 Prezentačná vrstva

V prezentačnej vrstve sú triedy, ktoré tvoria užívateľské rozhranie. Stránka default.aspx obsahuje vyššie uvedené ovládacie prvky a kontroluje zadané hodnoty, vzhľad je určený v súbore *style.css*.
Parametre spojenia dostane ďalšia stránka z presmerovania z úvodnej stránky, kde to prebieha pomocou reťazca. Zobrazovacia stránka *default3.aspx* slúži na výpis spojení, ktoré vyhovujú užívateľovým požiadavkám. Táto stránka spracuje reťazec, v ktorom sú parametre spojenia pomocou

metódy *Request[“reťazec”]*. S týmito parametrami sa potom vyvolajú metódy v aplikačnej vrstve. Výsledky sú zobrazované do tabuľky, ktorej vzhľad je definovaný pomocou kaskádových štýlov. Ak nevyhovuje žiadne spojenie, vypíše sa jednoduchý text. V tejto vrstve sa volajú metódy z aplikačnej vrstvy na vyhodnotenie, prekontrolovanie a zoradenie podľa kritéria.

4.2 Získanie dát

Dátová vrstva obsahuje triedy *Odlety.cs* a *Vyberdat*.cs. V prvej je definovaný objekt *Odlety,* ktorý má ako atribúty všetky údaje o lete. Tento objekt sa napĺňa v triede *Vyberdat*.cs . Napĺňa sa podľa toho, aké tlačítko užívateľ klikne. Ak zadá, že chce vyhľadať podľa aktuálnych letov, tak sa zavolá v metóde *Service()* webová služba “*GetFlights*“ z triedy “WebováSlužbaSystémuBagr“, ktorá vráti kolekciu letov, a naplní sa tým objekt aktuálnymi spojeniami, ktoré táto webová služba poskytuje. Je potrebné pri implementácii pridať do projektu referenciu na danú webovú službu. Ak sa nepodarí spojiť sa s danou službou, metóda vráti prázdny objekt. Metódy pre prácu s webovými službami sú v mennom priestore System.Net. Táto služba však neposkytuje údaje o cene letu. Z toho dôvodu majú všetky spojenia rovnakú cenu 0 a nezobrazia sa zoradené.

Ak užívateľ klikne na tlačítko “Vyhľadať“, v takom prípade metóda Spoj() naplní objekt údajmi z databázy. Tabuľky v databáze boli vytvorené na základe ER diagramu uvedeného v kapitole 3. Pri nahrávaní dát z databázy musíme každý objekt explicitne pretypovať kvôli bezpečnosti. Pre prácu s databázou sa využívajú menné priestory System.Data a System.Data.SqlClient. Triedy SqlDataReader a OleDbDataReader ADO.NET poskytujú prúdový prístup k výsledkom databázovým požiadavkám. Tento prístup je rýchly a výkonný, je však používaný iba na čítanie dát a iba na vopred známe požiadavky. To znamená, že sa nemôžeme vrátiť a znovu načítať predchádzajúci záznam ani zmeniť výsledky a zapísať ich späť do databázy. Práve preto podporuje ADO.NET okrem prúdového prístupu aj množinový prístup k dátam. Tento prístup zachytí celú požiadavku do pamäte a taktiež podporuje spätný aj vopred známy prechod výsledkovou množinou. V databáze sú uložené lety s rôznou cenou letu, ktorá bola doplnená náhodne v rozmedzí 1000-9999.

V tomto projekte sa využila trieda DataReader. Objekty tejto triedy, na rozdiel od objektov triedy DataSet, preberajú iba tie dáta, ktoré sa skutočne používajú, a nespotrebováva sa zbytočne pamäť ukladaním všetkých záznamov. Ak sa však využijú všetky záznamy, tak je výhodnejšie použiť objekty DataSet [4].

4.2.1 Voľba znakov

Pri práci s databázou sa zohľadňovalo aj rozlišovanie malých a veľkých písmen. Celý program pracuje teda na báze case – insensitive. To znamená, že užívateľ si môže zvoliť rozličnú veľkosť písmen pri zadávaní. Užívateľ však musí zadať presný názov mesta, napr. London a nie Londýn. Ihneď po načítaní potrebných údajov od užívateľov sa tieto znaky v reťazcových dátach zmenia na

malé, taktiež pri čítaní údajov z databázy sa pracuje s dátami s malými znakmi. Pri vypisovaní dát užívateľovi na stránku sa prvé písmená názvu letiska zobrazia ako veľké.

4.2.2 Voľba dátových typov

Tento projekt pracuje so štandardnými dátovými typmi:

- Integer – pracuje s celými číslami
- String – pracuje s reťazcami
- DateTime – typ pracujúci s časom
- TimeSpan – časový rozdiel medzi príletom a odletom
- List<> – kolekcia dátových typov

Definované sú však aj vlastné dátové objekty. Prvým je *JednaCesta*, ktorá má ako atribúty indexy liniek, ktoré spojenie obsahuje, celkovú cenu a celkový čas. Projekt obsahuje aj objekt *Odlety*, ktorý obsahuje ako atribúty ID letu, odletové a príletové letisko, kódy týchto letísk, cenu letu, ID aerolinky a názov, ktorý je sprostredkovateľom príslušného letu a dátum odletu a príletu. Ďalším výnimočným objektom je objekt *vsetko*, ktorý je definovaný ako List objektov *Odlety*. Tento objekt obsahuje všetky lety, ktoré sú reprezentované vlastnosťami letu. Tento objekt sa využíva v dátovej vrstve pri čítaní údajov z databázy alebo z webovej služby. Objekt *VsetkyCesty* je definovaný ako List objektov typu *JednaCesta*. Tento objekt obsahuje všetky spojenia, ktoré vyhovujú užívateľovým požiadavkám. V projekte sú deklarované aj ďalšie objekty, tie však slúžia na pomocné účely.

4.3 Vyhľadanie spojenia

Aplikačná vrstva slúži na vyhodnotenie spojení, konkrétne v triede *Data.cs* sú metódy, ktoré postupne napĺňajú zoznam priaznivých spojení. Prvá metóda, ktorá sa zavolá z prezentačnej vrstvy, je metóda s názvom *Cesta* , ktorá má ako parametre štartové letisko, cieľové letisko, počet prestupov, priebežnú cestu, maximálnu cenu a dátum odletu. Metóda spustí ďalšiu metódu *Filter*, ktorá vráti všetky lety z konkrétneho letiska v požadovanom dátume, pričom vyberie iba také lety, ktoré nie sú v konkrétnej ceste už priebežne uložené. Táto metóda sa využíva iba v triede *Data.cs*, preto má modifikátor viditeľnosti súkromný (*private)*. *Cesta* však musí byť viditeľná aj v prezentačnej vrstve, preto je určená ako verejná (*public*). Po vyfiltrovaní možných letov z určeného letiska metóda zisťuje, či sa v týchto letoch nenachádza aj cieľové letisko. Ak áno, skontroluje spojenie, či vyhovuje požadovanému odchodu a cene, a vyhovujúce spojenia uloží s celkovou cenou a celkovým časom do objektu *VsetkyCesty*. Tento objekt je kolekcia objektov *JednaCesta*. Ďalej sa pokračuje v hľadaní ďalej a takisto prehľadáva ďalej aj lety z tohto letiska, pretože je možné dostať sa aj inou cestou do cieľa. Ak sa nepodarí dostať na cieľové letisko, pokračuje v prehľadávaní rekurzívne. To znamená, že skúša lety z ďalšieho letiska, na ktoré je možné sa dostať z prvého letiska a za štartovacie letisko určí

aktuálne letisko. Pritom kontroluje aj počet prestupov. Ak toto rekurzívne volanie presiahne hodnotu počtu možných prestupov, metóda ukončí prehľadávanie. Inak prehľadáva postupne všetky lety z možných letísk. Ak nevyhovuje žiadne spojenie, metóda vracia prázdny objekt. Toto je modifikovaná verzia algoritmov postupného zanorovania do hĺbky DLS a IDS. V tomto projekte však po nájdení cieľa sa algoritmus neukončí, pretože sa tieto spojenia ešte musia zoradiť. Kvôli tomu dôvodu nebol použitý Dijsktrov algoritmus, ktorý bol pri návrhu určený na implementáciu. Postupne však bolo zistené, že tento algoritmus končí pri nájdení najkratšej trasy a ďalšie spojenia by neuvažoval. Keďže užívateľ si môže vybrať zo všetkých možných spojení, algoritmus nájdenia najkratšej trasy by bol neúčinný.

Ďalej v tejto aplikačnej vrstve sú metódy na zoradenie podľa ceny alebo času v metódach *ZoradPodlaCeny* a *ZoradPodlaCasu*. Použila sa metóda ***bubblesort (bublinkové radenie)*** pri obidvoch radeniach. Tento algoritmus opakovane prechádza zoznam spojení, pričom porovnáva každé dva susedné prvky. Ak tieto prvky nie sú v správnom poradí, vymení ich. Toto porovnávanie sa deje do tej doby, pokiaľ zoznam nie je zoradený. Ak užívateľ chce spojenia zoradiť podľa ceny a chce ako zdroj aktuálne lety, tak sa mu tieto lety zobrazia implicitne kvôli tomu, že webová služba neposkytuje údaje o cene letu. To znamená, že každé spojenie má rovnakú cenu 0.

V aplikačnej vrstve je implementovaná ešte metóda *Kontrola,* ktorá kontroluje, či v zoznamoch spojeniach sa nenachádzajú spojenia, ktoré obsahujú za sebou nasledovné lety také, že odchod nasledujúceho letu je skorší ako príchod predchádzajúceho letu. Ak áno, tak toto spojenie je nevyhovujúce, vymaže sa zo zoznamu. Taktiež tu prebieha kontrola takých spojení, kde by cestujúci musel dlho čakať na letisku. Kvôli tomu je tu vykonaná taká kontrola, že sa príletu priráta čas, ktorý si užívateľ zadá ako maximálny prestupný čas. Ak tento čas je menší ako odlet ďalšieho letu, spojenie je vyhovujúce. V opačnom prípade to znamená, že cestujúci by musel dlho čakať na letisku. Tieto spojenia sa zo zoznamu odstránia.

4.4 Grafické užívateľské rozhranie

GUI (Graphical User Interface), v preklade grafické užívateľské rozhranie, je určené na komunikáciu s užívateľom. Pri návrhu GUI je dôležité, aby bolo navrhnuté tak, aby bola práca pre užívateľa čo najjednoduchšia a aby na prvý pohľad zaujalo užívateľa.

4.4.1 GUI vyhľadávajúcej stránky

Bolo navrhnuté je jednoduché GUI s odtieňom jemnej bledomodrej farby. Užívateľ si zadá, odkiaľ a kam chce letieť, dátum odletu, výšku maximálnej ceny a maximálny počet prestupov.

Obrázok 4.3 GUI úvodnej stránky

4.4.2 GUI stránky po vyhľadaní

Užívateľovi sa zobrazí po tom, ako si zvolí svoje požiadavky, jednoduchá stránka s výsledkami podľa jeho požiadaviek, kde má výsledné spojenia usporiadané v tabuľke. Spojenia sa mu zobrazia buď podľa celkovej ceny spojenia, alebo podľa celkového času spojenia.

Dátum	Odkiaľ/Kam	Odlet	Prílet	Aerolinka	Cena	ID linky	Čas na prestup
16.5.2009	Bratislava(BTS)	10:00:00		Ryanair	7607	622	
	London(STN)		11:15:00				00:45:00
16.5.2009	London(STN)	12:00:00		Ryanair	5636	695	
	Dublin(DUB)		13:15:00				01:00:00
16.5.2009	Dublin(DUB)	14:15:00		Ryanair	1405	653	
	Manchester(MAN)		15:10:00				

Dátum	Odkiaľ/Kam	Odlet	Prílet	Aerolinka	Cena	ID linky	Čas na prestup
15.5.2009	Bratislava(BTS)	10:00:00		Ryanair	8712	430	
	London(STN)		11:15:00				01:55:00
15.5.2009	London(LTN)	13:10:00		Ryanair	1711	506	
	Dublin(DUB)		14:20:00				01:35:00
15.5.2009	Dublin(DUB)	15:55:00		Ryanair	5662	472	
	Manchester(MAN)		16:50:00				

Obrázok 4.4 GUI vyhľadávajúcej stránky

4.5 Kontrola zadaných dát

Aby sa predišlo neočakávaným chybám pri spojení s databázou alebo webovou službou, kontrolujú sa užívateľove vložené dáta pri ich vložení do text boxu. Kontrola prebehla pomocou serverových ovládacích prvkov RequiredFieldValidator a RegularExpressionValidator, ktoré sú definované v mennom priestore System.Web.UI.WebControls.

- TextBoxy **odkiaľ** a **kam** musia byť vyplnené, užívateľ musí zadať odkiaľ a kam chce letieť, inak systém vypíše chybové hlásenie. Nachádza sa tu aj ovládací prvok "Vymeniť", ktorý vymení text v týchto políčkach.
- Ďalší TextBox **Dátum** sa kontroluje pomocou RegularExpressionValidator. Tento ovládací prvok využíva ku kontrole regulárne výrazy. Formát dátumu je v tvare dd.MM.yyyy, čo znamená, že prvé dve čísla značia deň, ďalšie dve mesiac a poslednými štyrmi číslicami je tvorený rok. Ak užívateľ nechá toto políčko nevyplnené, použije sa aktuálny čas. Dátum sa môže vybrať pomocou prvku Calendar.
- **Čas** sa takisto musí zapísať v správnom formáte, a to hh:MM:ss. Pri nevyplnenej hodnote sa použije čas polnoci, to znamená 00:00:00
- TextBox **Cena** využíva tiež RegularExpressionValidator. Tu sa kontroluje iba to, či užívateľ zadá číslo alebo nie. Ak políčko ostane nevyplnené, použije sa implicitná hodnota 999999999.

  Checkbox "Zoradiť podľa ceny" vyhodnocuje spojenia, pričom ak nie je zaškrtnuté, spojenia sa zoradia podľa ceny, v opačnom prípade podľa dĺžky trvania spojenia.
- Ak užívateľ nezadá počet prestupov, použije sa implicitná hodnota 3. Inak je užívateľ obmedzený na 0-9 prestupov. Užívateľ si taktiež môže zvoliť, že chce spojenia bez prestupov. V takom prípade zaškrtne "Spojenia bez prestupov".
- ListBox "Min čas na prestup (min)" je implicitne nastavený na hodnotu 60 minút a Prvok "Max čas na prestup (hod)" je nastavený na 24 hodín. Užívateľ si však môže zvoliť, koľko minút na prestup potrebuje.
- Pri kliknutí na tlačítko "Vyhľadať" sa vyhľadajú spojenia, ktoré sú aktuálne uložené v databáze, ak sa klikne na tlačítko "Vyhľadať podľa aktuálnych letov", vyvolá sa webová služba, ktorá načíta aktuálne lety.

Obrázok 4.5 GUI úvodnej stránky pri zadaní zlých údajov

4.6 Odchytávanie výnimiek

V každej aplikácií môžu nastať za behu rôzne chybové stavy. Snahou každého prekladača je eliminovať ich veľkú časť už pri preklade. Prispieva k tomu aj CLR. Aj napriek tomu zostáva určitá množina chýb, ktorú nie je možné dopredu detekovať. Sú to napríklad otvorenie neexistujúceho súboru, náhly nedostatok operačnej pamäte, delenie nulou alebo indexovanie mimo povolené hranice.

.NET Framework zareaguje na tieto chybové stavy tzv. vyvolaním výnimky. Aplikácia by mala byť na tieto chybové stavy dopredu pripravená. V C# sa výnimky ošetrujú kľúčovými slovami *try, catch, finally*. Princíp spočíva v uzatvorení kódu, ktorý môže vyvolať výnimku v bloku *try (skúsiť)*. Umiestnenie rutín spracovaných výnimiek je v bloku *catch (zachytiť)*. Účelom je determinovať, o aký typ výnimky išlo a vykonať potrebnú sekvenciu príkazov. Každá výnimka so sebou nesie určitú informáciu o dôvode udalosti vo forme objektu, ktorý je typu System.Exception. Každý blok *catch* musí uviesť, aký typ výnimky sa má spracovávať. Ak chceme zabezpečiť, aby bola časť kódu spracovaná za každých okolností, nezávisle na tom, či nastala výnimka alebo nie, musíme za posledný *catch* uviesť blok *finally*. Tento blok sa využíva napríklad pri uzatvorení súboru, sql pripojenia a podobne. V tomto projekte boli odchytávané výnimky typu:

- *ArgumentOutOfRangeException* – Argument je neplatný (mimo rozsah).
- *FormatException* – Pracuje sa zo zlým formátom objektu.
- *NullReferenceException* – Dochádza k dereferencii nulového odkazu

- *OutOfRangeException* – Neplatný index pola alebo iného indexovaného objektu
- *SqlException* – Chyba pri práci s databázou (nepodarilo sa pripojiť)
- *WebException* – Chyba pri práci s webovými službami (nepodarilo sa nadviazať pripojenie)

4.7 Testovanie

Jednou z najdôležitejších častí vývoja programu je dôkladné testovanie, ktoré by sa nemalo podceniť. Cieľom testovania je zistiť chyby a nedostatky implementovaného systému. Testovanie sa vykonáva zadávaním hraničných alebo zlých hodnôt. Prebiehalo s pomocou malej databázy, ktorá bola vytvorená za týmto účelom. Pri malom množstve spojení sa ľahšie ladilo. Microsoft Visual Studio obsahuje veľmi ľahko ovládateľný debugger. Pri testovaní sa prišlo na niekoľko nedostatkov, napr. vypisovanie tých istých spojení viac-krát. Inak testovanie prebehlo celkom pozitívne.

5 Možné rozšírenia

Vývoj informačných systémov ide v dnešnej dobe neustále dopredu, z čoho najprogresívnejší vývoj zaznamenávajú systémy navrhnuté ako webové aplikácie. Táto webová aplikácia bola implementovaná na základe analýzy a návrhu. Boli vyriešené všetky požiadavky zadávateľa projektu. Boli odhalené niektoré nedostatky, na ktoré sa prišlo až pri testovaní. V tejto kapitole sú opísané možné vylepšenia portálu.

Prvým možným vylepšením je optimalizácia algoritmu, ktorý vyhľadáva spojenia. Ak užívateľ zadá veľký počet prestupov, algoritmus sa pri veľkom počte letov mnohonásobne spomalí.

Ďalším možným vylepšením môže byť vytvorenie užívateľského rozhrania pre administrátora, kde by mohol jednotlivé lety editovať alebo pridávať prípadne vymazávať, napr. keď sa zistí, že daný let je preplnený.

Jedným z možných vylepšení sú aj efektívnejšie využívanie trojvrstvovej architektúry, lepšia práca s databázou alebo väčší dôraz na bezpečnosť z toho dôvodu, že údaje sa medzi stránkami prenášajú v nezašifrovanej forme a niekto by sa mohol do systému "nabúrať".

Webový portál by sa dal vylepšiť aj tak, že by webová služba poskytovala aj údaje o cene. Vtedy by táto aplikácia poskytovala aj presné údaje z databázy. Databáza by bola priebežne aktualizovaná. Užívateľ by si mohol vyberať reálne spojenia priamo z databázy. V tomto štádiu sú v databáze iba testovacie údaje.

6 Záver

Bakalárska práca opisuje tvorbu webovej aplikácie, ktorá vyhľadá letecké spojenie v reálnom čase a spĺňa užívateľove požiadavky. Tieto spojenia majú byť nakoniec vyhodnotené a zoradené podľa času alebo ceny spojenia. Systém by mal byť typu user friendly, to znamená, že by mal byť ľahko ovládateľný. Spojenia sa majú získať z webovej služby alebo z databázy.

Túto tému som si vybral, aby som zdokonalil svoje vedomosti v tvorbe webových aplikácií, a taktiež preto, lebo ma cestovanie zaujíma a chcel som podrobnejšie zistiť, ako sa vyhľadáva na rôznych webových portáloch.

Určovanie leteckých spojení bola zaujímavá úloha. Bola však aj časovo náročná, keďže som nemal dostatočné skúsenosti so spomínanými technológiami. Taktiež dlhšie trvalo skúmanie jednotlivých algoritmov a určenie toho správneho. Výsledný efekt však priniesol uspokojenie. Zistil som, čo všetko treba na tvorbu webovej aplikácie, aké problémy sa vyskytujú pri jej tvorbe a spôsob ich riešenia. Preštudoval som rôzne knižky a študijné opory a podrobnejšie som sa zoznámil s tvorbou viacvrstvových webových aplikácií v.NET 2.0. Spoznal som jazyk C#, ktorý ma veľmi zaujal. Chcel by som sa v týchto technológiách aj naďalej zlepšovať a zdokonaľovať.

Aplikácia bola implementovaná ako webové rozhranie, ktoré obsahuje užívateľské rozhranie pre jednoduché určenie spojenia pre jednoduchých používateľov. Požiadavky zadané zadávateľom prešli postupne cez analýzu, návrh, implementáciu systému a jeho následné testovanie. V týchto fázach sa našlo niekoľko nejasností a niekoľko optimalizácií. V súčasnej dobe je aplikácia pripravená na používanie. Chyby, ktoré neboli odhalené, sa môžu časom prejaviť. Ďalší vývoj samotnej aplikácie by mohol byť prispôsobený možným rozšíreniam, ktoré sú uvedené v predošlej kapitole.

Projekt bol vytvorený bez väčších predchádzajúcich skúseností a znalostí s návrhom a implementáciou podobnej aplikácie s použitím daných technológií. S tým samozrejme súvisí doba vývoja projektu, ktorá by mohla byť pri väčších skúsenostiach kratšia. Je preto vhodné dobré načasovanie a rozdelenie všetkých častí vývoja projektu.

Rozdiel oproti ostatným vyhľadávajúcich cestovných portáloch je napríklad v tom, že táto aplikácia poskytuje užívateľovi možnosť vybrať si, či chce spojenia načítať z databázy alebo z webovej služby. Taktiež si užívateľ môže určiť maximálny počet prestupov a maximálnu cenu za spojenie. Výhoda je aj v tom, že portál zoradí tieto spojenia podľa celkovej ceny alebo celkového času spojenia. Je to však na úkor rýchlosti vyhľadávania.

Literatúra

[1] Meilir, P. J.: *Základy objektově orientovaného návrhu v UML*. Grada Publishing, Praha, 2001. ISBN 80-40-247-0210-X

[2] *Unified Modeling Language – Wikipédia*[online] posledná zmena 28.4.2009[citované 29.4.2009] Dostupné na URL: http://sk.wikipedia.org/wiki/UML

[3] Kačmář, D.: *Programujeme .NET aplikace*. Computer Press, Praha, 2001. ISBN 80-7226-569-5

[4] Prosise, J.: *Programování v Microsoft .NET - Webové aplikace v .NET Framework., C# a ASP.NET*. Computer Press, Praha, 2003. ISBN 80-7226-879-1

[5] *Cascading Style Sheets* [online] posledná zmena 24.4.2009[citované 29.4.2009] Dostupné na URL: http://www.w3.org/Style/CSS/

[6] Scheldon R: SQL: A Beginner's Guide, Second Edition. California, 2003

[7] *Microsoft SQL Server 2005: Přehled produktu SQL Server 2005* [online] [citované 2.5.2009] Dostupné na URL:
http://www.microsoft.com/cze/windowsserversystem/sql/prodinfo/overview/default.mspx

[8] doc. Ing. František Vítězslav Zbořil, CSc., Ing. František Zbořil, Ph.D. : Študijná opora predmetu IZU – Základy umělé inteligence, Fakulta Informačných Technologií VUT v Brně , Verzia 3.2006

[9] *Dijkstrov algoritmus – Wikipédia* [online] posledná zmena 3.5.2009 [citované 3.5.2009] Dostupné na URL : http://sk.wikipedia.org/wiki/Dijkstrov_algoritmus

[10] *The DLS algorithm* [online] [citované 3.5.2009] Dostupné na URL : http://www.ida.liu.se/labs/kplab/projects/dls/

[11] Ing. Ladislav Ruttkay, prednášky predmetu IW5 – Programování v .NET a C#, Fakulta Informačných Technologií VUT v Brně

Zoznam príloh

Príloha 1. CD