VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Informační systém banky

Ročníkový projekt

2006

Jakub Horňák

Informační systém banky

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Mgr.Tomášem Masopustem.

Další informace mi poskytl Radim Burget.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……

Jméno Příjmení

Datum

Abstrakt

Tento Projekt se zabývá tvorbou dynamických webových stránek prostřednictvím Servletů a JavaServer Pages. Jak bylo mým úkolem prostudoval jsem tyto technologie a sepsal jejich použití a důležité rysy. Většinu popsaných principů, postupů a nástrojů, které jsem v této práci popsal, jsem použil při tvorbě informačního systému banky. Dokument obsahuje prostředky pro vývoj , návrh a způsob implementace tohoto projektu.

Klíčová slova

JavaServer Pages, Servlet, dynamické webové stránky, Apache Tomcat, Eclipse, MySql, Java, JDBC.

Abstract

This project pursues the creating of dynamic web pages through Servlets and JavaServer Pages. As was my objective I read through these technologies and wrote up their usage and their most important aspects. Almost all of the principles, processes and tools, which I described in this writing, I have used in the creating of the banks's information system. The document contains resources for developing, concept and method of implementation of this project.

Keywords

JavaServer Pages, Servlet, Java, dynamic web pages, Apache Tomcat, Eclipse, MySql, JDBC.

Obsah

1 Úvod

Díky velkému rozmachu komunikačních technologií dnes většina světa zná počítačovou síť Internet. Jen málo z nich ovšem ví kolik různých technologií, nástrojů a postupů se skrývá za každou zobrazenou stránkou. Jednou takovou technologií jsou dynamicky vytvářené webové stránky. Vznikly prostým vývojem a zvětšováním požadavků na webové stránky. Při použití statických stránek je potřeba i při nepatrném požadavku na změnu stránky načíst celou novou stránku. Jedna dynamická stránka oproti tomu může vypadat mnohokrát jinak a pořád půjde o tutéž stránku. Jde jen o změnu parametrů zaslaných dané stránce a tím změněný výstup dané stránky podle zpracování vstupních parametrů.

Dnes pro tvorbu těchto stránek můžeme použít více technologií např. ASP – Aktive server pages .NET od firmy Microsoft, PHP, CGI skripty, Java servlets, JSP – JavaServer Pages (dále jen JSP), atd.

Mým úkolem bylo prostudovat tvorbu dynamických webových stránek pomocí JSP a vytvořit informační systém banky pomocí této technologie.

Následující kapitola obsahuje úvod do problematiky servetů a tvorby JSP stránek.

Třetí kapitola obsahuje analýzu, návrh a implementaci projektu.

2 Technologie servletů a JSP

Dynamické stránky se zpravidla používají, pokud je jejich obsah založen na datech od uživatele nebo je odvozen z dat uložených v databazích. V této kapitole jsou popsány dvě technologie, které se dají využít pro tvorbu dynamických stránek. Nejde o popis všech funkcí, ale o seznámení s podstatou tvorby dynamických stránek za použití těchto prostředků. První část Vás seznámí s technologií servletů, které se dají a používají při tvorbě JSP stránek. Druhá část obsahuje mechanizmy tvorby JSP stránek. Jsou zde i informace o nutném softwaru pro provoz těchto technologií. V obou částech jsou zpravidla popisovány funkce a mechanizmy použité v projektu.

2.1 Servlety

V této podkapitole se seznámíme se strukturou servletu, s principy jejich fungování a budeme se zabývat technikami, které je možné při vývoji servletu použít, i tím, jaké možnosti nám servlety nabízí.

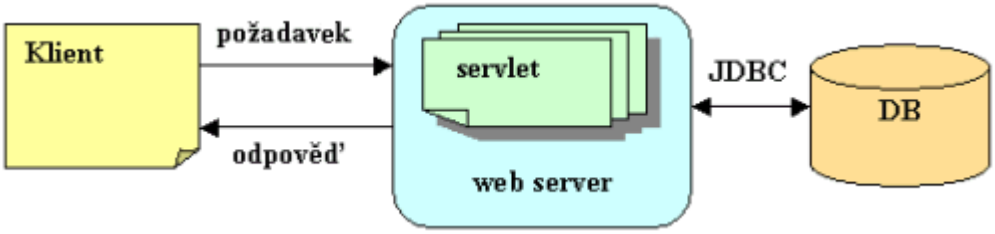
Servlety jsou vlastně programy, napsané v jazyce Java. To znamená, že vyžadují "java-enabled-server" a samozřejmě JVM (Java Virtual Machine), což je abstraktní platforma se svými vlastními instrukcemi. Java je interpretovaný jazyk, což znamená, že tyto instrukce jsou až při zavolaní programu překládány do instrukcí daného procesoru. To je zajištěno přes JVM, která má různé instalace pro různé platformy.

Většinou se používají servlety pracující nad internetovým protokolem HTTP. Odpovědí těchto servetů je potom převážně HTML kód. Tyto programy se používají ke generování dynamických webových stránek, k flexibilnější interakci s uživatelem a k provádění různých akcí na straně serveru. Například k pracování s databází (čtení, změna, zápis), posílání e-mailů, posílání cookies, zpracování formulářů atd. Tyto servlety se dají použít při tvorbě JSP stránek a zjednodušit tím vývoj těchto stránek.

2.1.1 Vlastnosti a výhody servletů

Interakce klient / server

Instance servletu je aplikace běžící na serveru, která zprostředkovává komunikaci mezi webovým prohlížečem či jiným HTTP klientem a serverem. Servlety implementují princip požadavek/odpověď, typický pro architekturu klient-server. Java Servlet API definuje standardní rozhraní pro obsluhu těchto požadavků a odpovědí mezi klientem a serverem. Následující obrázek zjednodušeně ilustruje procesy mezi klientem a serverem.



Obr. č. 1 – Interakce klient / server u servletu

1. Klient pošle požadavek na server.

2. Server přepošle požadavek na konkrétní servlet.

3. Servlet vytvoří odpověď a předá ji serveru.

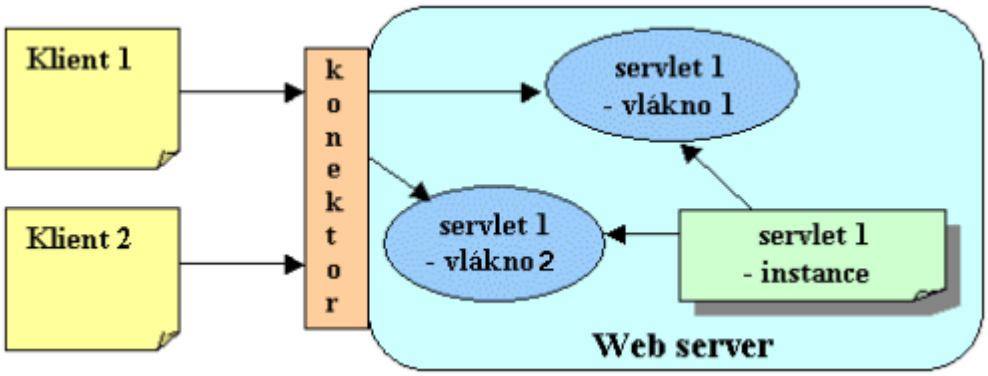
Odpověď je vytvořená dynamicky a její obsah závisí na zaslaném požadavku klientem.

4. Server pošle klientovi odpověď ve formě HTML kódu.

Úlohou serveru je řídit spouštění a inicializaci servletu, ukončovat a likvidovat servlet z paměti. Standardně je v paměti serveru jen jedna instance konkrétního servletu, kde zůstává i po dobu nečinnosti. Záleží na nastavení serveru, kdy bude uvolněná z paměti.

Přijde-li více požadavků na jeden servlet, server vytvoří pro každý požadavek nové vlákno. To znamená, že servlety jsou multithreadové.

Následující obrázek zjednodušeně ilustruje základní interakci mezi klientem a servetem v tomto případě:



Obr. č. 2 – Ilustrace multithreadu u servletů

1. Na počátku při prvním požadavku je servlet načtený serverem. Všechny proměnné instance jsou inicializované. Servlet zůstává aktivní po celou dobu životnosti.
2. Dva klienti žádají o službu stejný servlet. Server proto vytvoří dvě nezávislé vlákna, přičemž každé vlákno má přístup k instanci servletu, jeho metodám a proměnným.
3. Každé vlákno obsluhuje vlastní požadavky a odezvy, které vrací příslušnému klientovi.

Vlastnosti servletů

- **Portabilita a platformová nezávislost** - tuto vlastnost mají servlety díky jazyku Java. Java Servlet API totiž definuje standardní rozhraní mezi servletem a web serverem.
- **Perzistence a výkonnost** - servlet je zkompilovaný a natažený do paměti jen jednou a poté volaný při každém požadavku od klienta. To znamená, že servlet méně vytěžuje systémové zdroje, databázové připojení a podobně. Např. u CGI se vkládá do paměti při každém požadavku.
- **Založené na Jave** – díky tomu, že jsou servlety založené na jazyku Java, mají k dispozici všechny výhody tohoto jazyka. Objektovost, modularitu, bezpečnost a podobně.

V této práci se budu zabývat servlety podle specifikace 2.2, které jsou součástí Java Enterprise Edition (J2EE). **[1]**

2.1.2 Spuštění a životní cyklus servletu

Instanci servletu lze vytvořit vyvoláním URL odpovídajícího servletu přes HTML formulář nebo přímo z prohlížeče.

HTML formulář

Jde o jednoduchý a spolehlivý způsob jak dostat potřebné informace od uživatele. Jsou k dispozici různé prvky HTML formulářů. Např. tlačítka, editovací okénka, přepínače a seznamy. Každý z těchto prvků má svoje jméno a hodnotu.

Při definici formulářů ještě musíme u značky FORM nastavit ACTION na URL servletu či JSP stránky, kterou chceme spustit s parametry zadanými v tomto formuláři. Dalším důležitým parametrem formuláře je METHOD. Pro metodu GET mu můžeme (ale nemusíme) nastavit METHOD="GET". Pro metodu POST je nutné explicitně zadat METHOD="POST".

Použijeme-li metodu GET, pak se parametry v zakódovaném řetězci přidají na konec URL, která je nastavená v ACTION. Mezi tuto URL a zakódovaný řetězec se vloží znak '*?*'. Za tímto znakem následují parametry formuláře oddělené znakem '&'.

Pokud použijeme metodu POST, pak se parametry v zakódovaném řetězci odesílají na samostatném řádku (až po záhlavích požadavku a prázdné řádce) a tedy nejsou vidět z adresové řádky prohlížeče.

Spuštění a inicializace servletu

Životní cyklus servletu je zabezpečovaný prostřednictvím metod *init(), service(), doGet(), doPost() a destroy(),* které jsou obsaženy v rozhraní Servlet. Servlet je spuštěný a načtený do paměti až když je zavolaný. Lze také nastavit, aby se určité servlety inicializovali startu či restartu serveru.

V obou případech se yavolá metoda *init()*, která se postará o inicializaci servletu. Může např. Vytvořit spojení s databází, nastavit počáteční parametry atd. Tato metoda může být vynechána, v tom případě se zavolá metoda *init()* nadřazené třídy. Běžná forma této metody akceptuje object třídy *ServletConfig*, která zpřístupňuje inicializační parametry pro daný servlet. Také nám umožní přistupovat k objektu třídy *ServletContext,* ve kterém jsou informace o prostředí, ve kterém běží. Pro vstupní parametry můžeme použít soubor *web.xml* popsaný v podkapitole 3.3.5.

Průběh a ukončení servletu

Když je servlet inicializovaný, můžeme začít obsluhovat daný požadavek. Každý požadavek je definován jako objekt třídy *ServletRequest* a příslušná odpověď objektem *ServletResponse* z příslušného Java Servlet API. Při práci nad protokolem HTTP jde o *HttpServletRequest* a *HttpServletResponse*.

Objekt *HttpServletRequest* obsahuje jak informace o prostředí, tak všechny vstupní data zadané např. ve formuláři.Tento objekt obsahuje metody na přístup k těmto informacím např. *getParameter(), getAttribute(), getCookies(), getRequestURL()* atd.

Objekt *HttpServletResponse* je dynamicky vytvořená odpoveď poslaná klientovi. Odpovědí servletu může být HTML stránka, čistý text, XML dokument, obrázek atd. Odpovědí může být i přesměrování na jiné URL, servlet, JSP a podobně.

Při vytvoření nového vlákna se zavolá metoda *service()* , ktereá překontroluje typ požadavku : GET, POST ,DELETE, atd. A podle toho zavolá příslušnou metodu doGet(), doPost(), doDelete(), atd. Pokud bychom chtěli implementovat stejné akce pro různé typy požadavků (např. POST a GET), pak je lepší použít volaní metody *doGet()* z metody *doPost()* (nebo naopak). Důvodů je zde několik, jedním z nich je, že později můžeme jednoduše dodat implementace metod pro další typy požadavků (např. *doDelete()*). Dalším důvodem je, že pokud máme definovanou metodu *doGet()*, pak máme i automatickou podporu pro požadavky typu HEAD. Máme tedy zajištěno, že se vykoná metoda *doGet()*, ale uživateli se vrátí jen hlavička stránky se stavovými kódy a záhlavími odezev a ne tělo stránky. To může snížit zátěž serveru (např. při kontrole slepých odkazů). Další výhodou použití *doGet()* je, že standardní metoda *service()* bude automaticky odpovídat na podmíněné požadavky

GET, tedy požadavky, které umožňují prohlížečům načítat stránky z vyrovnávací paměti, pokud jsou tyto stránky aktuální.

2.1.3 Sledování sezení

Vždy, když si klient vyžádá novou stránku, otevře se nové samostatné spojení k webovému serveru. To platí pro HTTP protokol, čili si neuchovává žádné informace o stavu klienta. Neexistuje tedy žádná vestavěná podpora pro udržení souvisejících informací.

Toto se dá vyřešit pomocí tzv. Session tracking mechanizmu, který je podporovaný přímo Servlet API a to rozhraním *javax.servlet.http.HttpSession.* Výhoda toho mechanizmu oproti jiným (např. state tracking) je, že programátor se v podstatě nemusí starat o cookies, přepisování URL atd. Toto rozhraní obsahuje metody na vytvoření, sledování a řízení sezení.

Při použití *HttpSession* se obvykle nejdříve vyhledá nebo případně založí objekt rozhraní *HttpSession*. K tomu slouží metoda rozhraní *HttpServletRequest getSession()*. Po zavolaní této metody systém vyextrahuje identifikátor uživatele z cookies nebo z připojených dat v URL, tento identifikátor použije jako klíč do tabulky vytvořených objektů *HttpSession*. Pokud tam objekt nenajde založí nový. Pomocí metody *isNew()* jde zjistit, jestli byl vytvořen nový objekt. Do objektu *HttpSession* lze ukládat i jiné objekty definované názvem typu *String.* Pro práci s atributy a s vlastním sezením má toto rozhraní tyto metody :

- *setAttribute(String, Object)* – vloží do *HttpSession* objekt pod zvoleným názvem, je-li již název v *HttpSession* přepíše daný objekt novým
- *Object getAttribute(String)* – tato metoda vrací objekt, který je nutno přetypovat na typ objektu, který jsme do *HttpSession* vložili.
- *removeAttribute(String)* – odstraní z *HttpSession* objekt zvoleného názvu
- enumeration *getAttributeNames()* – vrací názvy všech atributů sezení
- string geoid() – vrací jednoznačný identifikátor sezení
- long *getCreationTime()* – vrací čas vytvoření sezení v milisekundách
- int *getMaxInactiveInterval()* – vrací čas v sekundách po který může být objekt sezení neaktivní, než se stane neplatným. Záporná hodnota udává časovou neomezenost.
- *setMaxInactiveInteval()* – nastavuje předchozí položku
- *invalidate()* – ruší platnost objektu sezení a uvolňuje všechny objekty s ním sdružené

2.1.4 Přesměrování požadavku

Budeme-li potřebovat přesměrovat na jiný dokument (např. stránku JSP, HTML nebo servlet) můžeme použít rozhraní *RequestDispatcher*. Toto rozhraní lze použít i pro začlenění nějakého vnějšího obsahu do servletu.

Nejdříve potřebujeme získat instanci rozhraní *RequestDispatcher* pomocí metody *ServletContext getRequestDispatcher*. Pro zaregistrované servlety či JSP stránky lze použít *getNamedDispatcher*.

Po získání instance můžeme bud' plně předat řízení cílové stránce metodou *forward* nebo můžeme zahrnout výstup cílové stránky do servletu metodou *include* a tím ponechat řízení u daného servletu. Parametry těchto metod jsou typu *HttpServletRequest* a *HttpServletResponse*.

2.2 JSP stránky

Jak již jsem zmínil, jsou JSP stránky jednou z možností, jak vytvářet dynamické webové stránky. JSP stránky jsou HTML stránky, do kterých se pomoci speciálních značek tzv. skriptovacích značek vkládá kód jazyku Java. Tento kód tvoří dynamický obsah webových stránek. Je více možností jak tvořit JSP stránky, například použití standardních programů pro tvorbu webových stránek (homesite, macromedia flash aj.) a dopsání kódu Javy nebo přímo psát stránky třeba pomoci IDE pro vývoj webových aplikací (Eclipse, JDeveloper aj.).

Použití JSP stránek je vhodné pokud převládá statický obsah stránek v opačném případě je lepší používat servlety. Třetí možností je používat obojí naráz.

2.2.1 Jak JSP stránky fungují?

Zdrojový kód stránky JSP je přeložený na servlet a následně zkompilovaný. Výsledkem je servlet, který generuje HTML. To je poslané klientovi, jako odpověď na jeho požadavek. Detailní postup je zde :

1. Uživatel požádá o webovou stránku, která byla vytvořená jako JSP. Klient vytvoří požadavek (request) a směřuje ho přes síť na server..
2. Požadavek je nasměrovaný na příslušný web server.

3. Web server zjistí, že požadovaný soubor je speciální, protože má koncovku "jsp". Proto přesměruje JSP soubor do JSP Servlet stroje.
4. V případe, že tento soubor byl volaný poprvé, JSP stroj ho překontroluje a pokračuje bodem 5. V opačném případě pokračuje bodem 7.
5. Zde se vygeneruje speciální servlet, vytvořený na základě souboru JSP. Všechno statické HTML je uložené v *out.println()* příkazech.
6. Zdrojový kód servletu je zkompilovaný a je vytvořený ".class" soubor. Např. ze souboru *index.jsp* bude vytvořený soubor *index_jsp.class*.
7. Je vytvořená instance servletu a zavolají sa metody init() a service() (kapitola 2.1.2).
8. Výstupem ze servletu je HTML, které je poslané přes síť.
9. Uživatelovi se zobrazí výsledky. Při zobrazení zdrojového kódu jde vidět pouze výsledný statický kód bez dynamického obsahu.

Použití JSP stránek

Instalace stránek JSP na server probíhá tak, že je nakopírujeme do libovolného adresáře, kam bychom mohli uložit statické HTML stránky. Jejich jednoduchost spočívá i v tom, že je nemusíme kompilovat a ani pro ně nemusíme nastavovat *CLASSPATH*. Jediné, co musí být nastaveno je webový server, který musí mít přístup k adresáři, ve kterém jsou stránky JSP uloženy a také musí mít přístup ke kompilátoru Javy. Více informací naleznete v podkapitole 3.3.1. **[2]**

2.2.2 Skriptovací značky

Skriptovací značky umožňují specifikovat kód v Jave, který se stane součástí výsledného servletu. Existují čtyři druhy skriptovacích značek.

- **Výrazy**

  Používají se tehdy, když potřebujeme vložit nějaký výstup přímo do stránky. Většinou se jedná o hodnotu nějaké proměnné. Výraz se v čase požadavku vyhodnotí a převede na řetězec a poté vloží na stránku. Má následující syntaxi:

<%= výraz v Javě %>

Např. `<%=request.getRemoteAddr()%>` vrátí na stránku jednoznačné ID session.

- **Skriplety**

  Výrazy ovšem nestačí, často je potřeba napsat složitější kód v Javě přímo do JSP stránky. K tomu máme skriplety. Skriplety, podobně jako výrazy JSP, mají přístup k tzv. implicitním objektům (podkapitola 2.2.3). Je možné prokládat HTML kód kolika skriplety jen chceme. Kód napsaný uvnitř těchto značek se vloží do metody *_jspService()*. Mají následující syntaxi :

<% kód v Javě %>

- **Deklarace**

  Umožňuje definovat metody, proměnné nebo pole, které se vloží do těla servletu. Oproti skripletům se vkládají do servletu mimo metodu *_jspService()*. Deklarace nevytváří žádný výstup, ale běžně se používá ve spojení s výrazy a skriplety. Mají následující syntaxi :

<%! kód v Javě %>

- **JSP komentáře**

  Mají v podstatě stejný význam jako HTML komentáře, až na to, že se neposílají klientovi. Při zobrazení zdrojového kódu nepůjde nic uvnitř značek vidět. Mají tuto syntaxi :

<%-- kód v Javě--%>

2.2.3 Implicitní objekty

JSP dává k dipozici devět implicitně vytvořených proměnných tzv. implicitní objekty. Jde o proměnné, které jsou spravovány serverem, není je třeba inicializovat a starat se o ně. Kód deklarace leží vně metody *_jspService()*, čili nejsou v deklaracích dotupné. Tyto proměnné jsou :

- **request (požadavek)**

  Tato proměnná je sdružená s požadavkem. Zpřístupňuje parametry požadavku, typ požadavku a záhlaví HTTP – *HttpServletRequest*. Jestliže je protokol jiný může být *request* typu *SerletRequest.*

- **response (odpověď)**

  Tato proměnná je naopak sdružená s odpovědí klientovi.

- **pageContext (kontext stránky)**

  Tato proměnná poskytuje jednotný přístup k mnoha atributům stránky a místo na uložení dat. Většinou sdílených dat. Je sdružená s aktuální stránkou.
- **session (sezení)**

  Proměnná je typu *HttpSession* a je svázaná s požadavkem. Je vytvářená automaticky. Jedinou výjimkou je nastavení parametru session v direktive page na false (podkapitola 2.2.4).
- **application (aplikace)**

  Stránky JSP mohou do proměnné vkládat perzistentní data. Na to se používají dvě metody *setAttribute()* a *getAttribute()*. Tato proměnná je sdílené mezi všemi servlety.
- **out (výstup)**

  Toto je objekt typu *PrintWriter* používaný pro odesílaní odpovědí klientovi.
- **config (konfigurace)**

  Tato proměnná obsahuje objekt typu *ServletConfig* pro aktuální stránku.
- **page (stránka)**

  Byla vytvořená pro případ, že bude možno použít i jiný jazyk než Javu..
- **exception (vyjímka)**

  Tento objekt je typu *Throwable* a obsahuje základní metody pro práci s výjimkami, které mohou vzniknout na stránce JSP. **[3]**

2.2.4 Direktivy

Ovlivňují celkovou strukturu servletu. Existují tři typy direktiv :

- **page**

  Importuje třídy, nastavuje typ obsahu stránky, kódování stránky, povoluje nebo zakazuje vytváření session atd. Syntaxe direktivy:

  <%@ direktiva atribut_l="hodnota_l" ... atribut_n="hodnota_n" %>

- **include**

  Direktiva include umožňuje, aby vývojář vložil obsah jednoho souboru do jiného. Často se takto vkládají různé navigační prvky, tabulky, hlavičky nebo paty, které jsou společné pro více stránek. Tato direktiva se zapisuje následovně, přičemž má jen jeden parametr:

  <%@ include file = "relativeURL" %>

- **taglib**

  Často se Tag Lib = Tag Library = knihovna uživatelských značek, které se mhou použít na stránce. Syntax je takový :

  <%@ taglib uri = "tag_library_URI" prefix = "tag_prefix" %>

  V podstatě umožňují vývojářům skrýt komplexní kód na stránce serveru před web designery. Atribut *uri* je nastaven na relativní nebo absolutní URL vedoucí k souboru TLD. Atributem prefix specifikujeme předponu, která se bude používat na stránce JSP před jménem vlastní značky:

  <tag_prefix:jméno značky />

3 Informační systém banky

Po domluvě s vedoucím projektu Mgr. Tomášem Masopustem, jsme se dohodli na IS – informačním systému internetového bankovnictví jako zadání projektu. Banku a aplikaci jsem nazval ***Student's Bank***.

3.1 Specifikace požadavků

Účelem této podkapitoly je podat úplnou specifikaci projektu podle dohody s vedoucím projektu. Jedná se o očekávané vlastnosti, funkce a rozsahu IS banky.

3.1.1 Kontext produktu

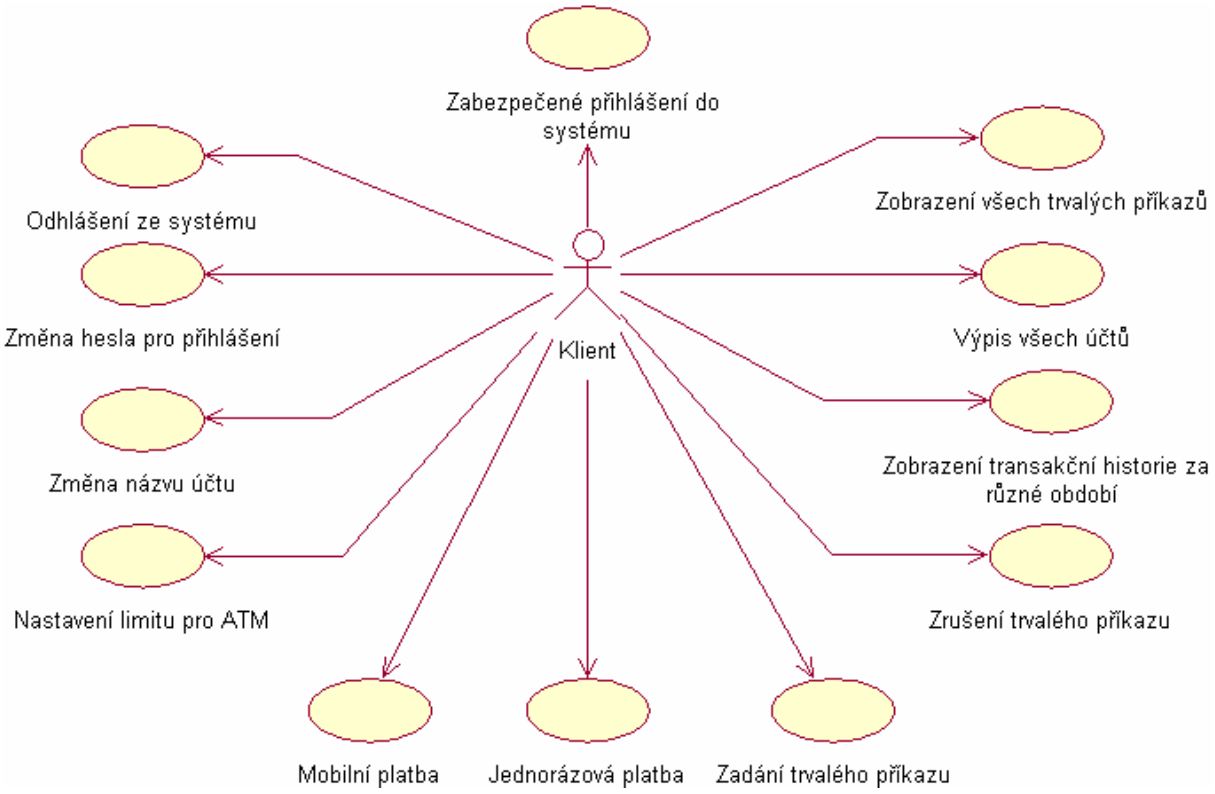
Pro implementaci a následné používání systému bude nutný databázový server a WWW server s podporou JSP stránek a servletů. Pro přístup do systému postačí libovolný prohlížeč HTML stránek.

3.1.2 Přehled funkcí

- ✓ Zabezpečené přihlašování do systému
- ✓ Odhlášení ze systému
- ✓ Výpis všech účtů
- ✓ Zobrazení transakční historie za různé období
- ✓ Zadání trvalého příkazu
- ✓ Zobrazení trvalých příkazů
- ✓ Zrušení trvalého příkazu
- ✓ Jednorázová platba
- ✓ Mobilní platba
- ✓ Změna hesla pro přihlášení
- ✓ Nastavení limitu pro výběr z ATM
- ✓ Změna názvu účtu

3.1.3 Uživatelé systému

U internetového bankovnictví stačí pouze klient jako jediný uživatel systému. Bude moci využívat všechny výše uvedené funkce.

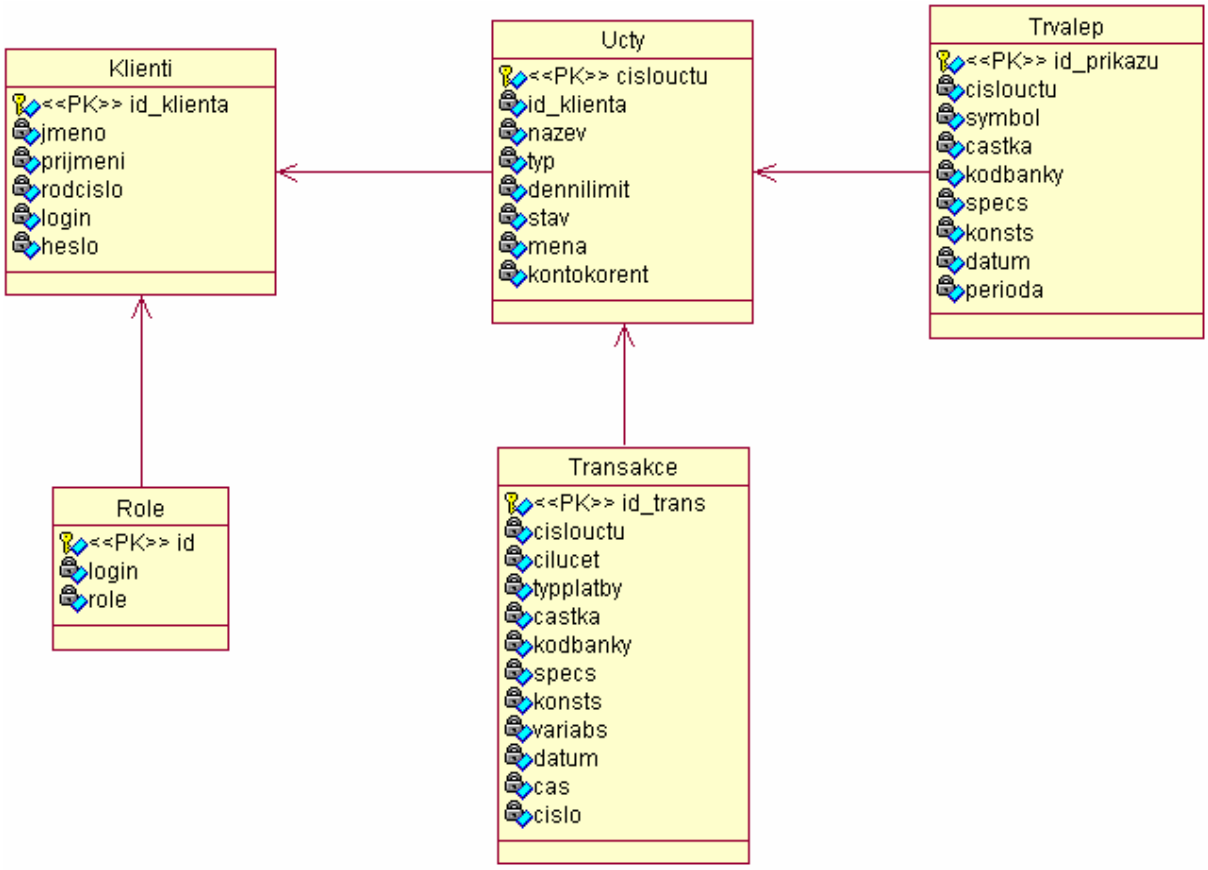


Obr. č.3 - Use case diagram klienta

3.2 Návrh databáze

Podle specifikace požadavků a podle způsobu přihlášení jsem zavedl tabulky:

- Klienti – uchovává informace o klientovi včetně jeho loginu a hesla
- Role – obsahuje loginy a jim přiřazené role.
- Účty – obsahuje účty všech klientů s informacemi o stavu, názvu, měně účtu atd.
- Transakce – uchovává informace o transakcích na účtech klientů se všemi atributy transakcí.
- Trvalep – obsahuje informace o trvalých příkazech vztažených na jednotlivé účty.



Obr. č.4 -Databáze z pohledu UML.

3.3 Implementace

3.3.1 Vývojové prostředí

Kontejner

Jako aplikační server jsem si zvolil Open Source **Apache Jakarta Tomcat** Servlet/JSP container. Tomcat verze 5.0.28, která implementuje servlety specifikace 2.4 a JSP stránky specifikace 2.0. volně stažitelný na stránce http://jakarta.apache.org/tomcat. Instalace nebyla nijak složitá, bylo potřeba nastavit v proměnném prostředí *TOMCAT_HOME* na instalaci Tomcatu. Server běží implicitně na portu 8080. Po vložení loginu a hesla do konfiguračního souboru *$TOMCAT_HOME\conf\server.xml* jsem byl schopen na stránce http://localhost: 8080/manager/list vidět všechny aplikace v adresáři Tomcatu /Webapps a spouštět si je. Všechny aplikace takto uložené jsou přístupné přes URL http://localhost: 8080/Název aplikace.

Grafické vývojové prostředí

Pro tvorbu webové aplikace pomocí JSP stránek společně se servlety jsem použil Open Source program **Eclipse SDK Verze: 3.1.2** volně stažitelný ze stránky http://www.eclipse.org/downloads/index.php. Jde o grafické vývojové prostředí vyvinuté firmou IBM. Ovšem dnes se jedná o samostatnou Open Source organizaci. Pro překlad servetů a JSP stránek jsem nainstaloval **Java 2 Platform Standard Edition Development Kit 5.0.** V programu Eclipse se mi po jistých potížích podařilo správně nastavit cestu ke všem potřebným ".jar" souborům pro překlad JSP stránek.

Plugin pro Eclipse

Pro napojení Eclipsu na Tomcat jsem nainstaloval Open Source Sysdeo Elipse Tomcat Launcher Plugin volně stažitelný ze stránky http://www.sysdeo.com/sysdeo/eclipse/tomcatplugin. Po nastavení *TOMCAT_HOME* a zvolení verze Tomcatu, jsem měl možnost vytvořit přímo projekt Tomcat 5.x. Díky tomuto nastavení jsem při změnách do zdrojového kódu nemusel restartovat Tomcat. Což by mi pokaždé zabralo asi 20s. Také chyby při kompilaci se zobrazovali přesnější v konzoli Eclipsu než na chybové stránce Tomcatu, čímž se mi usnadnilo lazení projektu.

Databázový server

Databázový server jsem zvolil Open Source MySql server v.4.1 volně stažitelný ze stránky http://www.mysql.com/downloads/mysql.html. V tomto serveru jsem vytvořil navrhnutou databázi (podkapitola 3.2) a použil ji k odzkoušení projektu.

3.3.2 Datbázové rozhraní

JDBC (Java Database Conectivity) je součástí J2EE (podobně jako servlety). Jde o rozhraní na práci s databázovými systémy. Použití tohoto rozhraní však přináší určité nevýhody, které je možné kompenzovat použitím tzv. connection poolingu s využitím JNDI (Java Naming and Directory Interface) ve spojitosti s databázemi. K použití potřebujeme implementovat pomocí metody *import java.sql.** pro používání níže popsaných metod. Dále je potřeba registrovat JDBC ovladač pomocí metody *Class.forName(String).* Načteme URL, aby šlo jednoznačně identifikovat datový zdroj a šlo s ním navázat spojení. K tomu slouží metoda *getConnection(URL, login, heslo).* Jde o login a heslo k připojení do databáze. Instance třídy *Statement* poté slouží k posílání příkazů databázi, pomocí metody *executeQuery(),* kde jsou zpracovány a vrací odpovídající hodnoty, které se ukládají do třídy *ResultSet.* Metoda *executeUpdate()* slouží k aktualizaci databáze, tedy příkazy do databáze jako UPDATE, INSERT, DELETE atd. Obě tyto metody mají parametr SQL dotaz do databáze.

3.3.3 Autentifikace a autorizace

Použil jsem rozhraní Tomcatu ***org.apache.catalina.realm.JDBCRealm***, který zpřístupňuje autentifikační informace uložené v relační databázi dostupné přes JDBC ovladač. Nejprve je potřeba nastavit Realm v konfiguračním souboru Tomcatu (conf/server.xml). Do tohoto souboru je potřeba vložit ovladač databáze, URL, heslo a login do databáze, název databáze, názvy tabulek a sloupců obsahující login, heslo a roli uživatele. Jsou také nutné zásahy do souboru *web.xml* (podkapitola 3.3.5). Pro způsob autentifikace jsem zvolil metodu FORM, což znamená, že stránka pro přihlášení a stránka pro nezdařené přihlášení bude mnou vytvořená a nebude to jen male okno pro zadání loginu a hesla, jak je to např. při přihlašování do *WIS*u. **[4]**

3.3.4 Servlety aplikace Student's Bank (/sb)

Aplikace obsahuje osm servetů. Každý z nich jsem popsal a namapoval v souboru *web.xml* (podkapitola 3.3.5). Následuje seznam jednotlivých servletů společně s jejich funkcemi :

- **Bonus.java**

  Tento servlet jsem pomocí *jsp:include* vložil do úvodní stránky projektu. Zjistí datum, roční období a kdo má svátek. Vypíše pozdrav se zjištěnými daty.

- **Kontrolajedplatby.java / Kontrolamobplatby.java / Kontrolatrvplatby.java**

  Tyto servlety kontrolují vyplnění jednotlivých formulářů pro jednorázový příkaz k úhradě, mobilní platbu a trvalý příkaz.

  Ve všech se kontroluje správně zadaná částka pomocí pokusu převodu na typ *double.* Proběhne kontrola cílového účtu. Je-li zvolen kód banky Student's Bank, prohledává se databáze účtů. Při zvolení jiného kódu banky není tato kontrola možná. Zjišťuje se je-li dostatek peněz na zvoleném účtu pro danou transakci.

  Při kontrole trvalého příkazu se kontroluje zvolené datum první platby, zda-li zvolené datum existuje (např. 31.02.yyyy neexistuje) a zda-li není zvolené datum menší než aktuální.

  Výsledky těchto kontrol se posílají příslušným JSP stránkám.

- **Smaztp.java**

  Tento servlet smaže vybraný trvalý příkaz z tabulky trvalých příkazů.

- **Proved.java / Provedmob.java / Provedtrv.java**

  Tyto servlety jsou spuštěny při správném zadání všech hodnot do formulářů jednorázové platby, mobilní platby a trvalého příkazu, a po od souhlasení potvrzovacího formuláře. Pro jednorázovou platbu a mobilní platbu se ještě jednou zkontroluje dostatek peněz na účtu, zapíše se nový stav účtu do databáze a zapíše se také transakce do tabulky transakcí. Je-li cílový účet při jednorázové platbě ve Student's Bank připočte se částka na daný účet a zapíše se tato transakce i pro tento účet do tabulky transakcí.

  U trvalého příkazu se za zapíše do tabulky trvalých příkazů.

- **Zmenan.java / Zmenalimitu.java**

  Zajistí kontrolu a případnou změnu názvu účtu zapsáním do tabulky účtů. / Zajistí kontrolu a případnou změnu denního limitu účtu zapsáním do tabulky účtů.

- **Zmenah.java**

  Zajistí kontrolu a případnou změnu hesla účtu zapsáním do tabulky klientů.

3.3.5 Soubor web.xml aplikace Student's Bank

Soubor *web.xml* je konfiguračním souborem pro servlety dané aplikace. Základní element je *<servlet>* ve kterém jsou všechna nastavení pro danou třídu servletu. Použitím *<servlet-name>* určíme jméno (tzv. alias), pod jakým bude servlet vystupovat. Nastavením *<servlet-class>* říkáme s jakou třídou má být servlet spojen. Můžeme také namapovat servlet na vlastní URL pomocí elementu *<servlet-mapping>* a jeho dvou subelementů *<servlet-name>* a *<url-pattern>*. Jméno servletu je zaregistrované jméno. To znamená, že nejprve musíme mít zaregistrovaný servlet, až potom k němu můžeme vytvořit vlastní URL. Do elementu *<url-pattern>* vložíme požadované URL. Obsah tohoto elementu musí začínat lomítkem „/“. Element *<init-param>*, se svými podelementy *<param-name>* a *<param-value>*, definuje vstupní parametr, který se předává servletu. Tyto parametry se čtou metodou *getlnitParameter()* objektu třídy *ServletConfig* předaného metodě servletu *init()*.

V tomto souboru jsem mimo těchto elementů ještě nastavil subelement *<welcome-file>* elementu *<welcome-file-list>* na *index.jsp*. Jde o nastavení uvítací stránky. Dále jsem nastavil všechny parametry elementu *<security-constrain>* tak, že všechny soubory v adresáři aplikace jsou v chráněné zóně a do které mají přístup jen přihlášení uživatelé s rolí „klient”. V elementu *<login-config>* jsem nastavil způsob autentifikace na *FORM* pro použítí vlastní přihlašovací stránky a stránky při špatném přihlášení. Tyto stránky jsem nastavil na *login.jsp* a *error.jsp*. Posledním nastavením v subelementu *<session-timeout>* elementu *<session-config>* na 30. Jde o zajištění, že po 30 sekundách neaktivního sezení se sezení prohlásí za neplatné. **[5]**

3.3.6 JSP soubory aplikace Student's Bank

Všechny JSP stránky kromě login.jsp a error.jsp obsahují kaskádové styly a používají soubor stylů /css/default.css.

- **login.jsp**

  Jde o stránku s formulářem pro autentizaci klienta. Klient po zadání loginu a hesla je verifikován a přesměrován na úvodní stránku *index.jsp*. Pokud verifikace nebyla úspěšná je přesměrován na stránku *error.jsp*. Pro přístup do aplikace můžete použít přihlašovací jméno 'hoho' a přihlašovací heslo 'hohohoho'.

- **error.jsp**

  Stránka se zobrazí při neúspěšné verifikaci klienta. Obsahuje zpětný odkaz na *login.jsp*.

- **výpisuctu.jsp**

  Na této stránce se do tabulky vypíšou všechny účty klienta nalezené v databázi společně s některými atributy účtu. Číslo účtu funguje jako odkaz na stránku *ucty.jsp*. Každého účtu jsou odkazy na zobrazení trvalých příkazů(*trvprikazy.jsp*) a na zobrazení transakční historie účtu za poslední měsíc(*transakce.jsp*). Tato stránka také zobrazí výsledek změny názvu účtu viz *Zmenan.java.*

- **ucty.jsp**

  Stránka zobrazí do tabulky detailní informace o účtu. Dále obsahuje odkaz na výpis trvalých příkazů(*trvprikazy.jsp),* formulář pro pro změnu jména účtu, který volá servlet *Zmenan.java a* obsahuje formulář s výběrem časového intervalu pro zobrazení transakcí daného účtu, který volá *transakce.jsp*.

- **platby.jsp**

  Jde o stránku na které se zobrazí submenu plateb(odkazy na *jedplatba.jsp, mobplatba.jsp* a *trvprikaz.jsp*).

- **jedplatba.jsp, mobplatba.jsp, trvprikaz.jsp**

  Tyto stránky obsahují formuláře pro jednorázový příkaz k úhradě, mobilní platbu a trvalý příkaz. Tyto formuláře volají servlety *Kontrolajedplatby.java, Kontrolamobplatby.java, Kontrolatrvplatby.java.* Automaticky formuláře obsahují seznam naplněný účty klienta.

- **jedplatbaver.jsp, mobplatbaver.jsp, trvprikazver.jsp**

  Uvedené stránky vypisují buď kontrolní formulář naplněný daty zadanými v předchozích stránkách nebo informace o chybách při vyplnění formuláře. Záleží na odpovědi příslušných servetů starajících se o kontrolu hodnot. Při bezchybném kontrole obsahují odkaz na servlet pro provedení požadavku, při chybě obsahují zpětný odkaz na stránku formuláře.

- **trvprvyber.jsp, transvyber.jsp**

  Tyto stránky obsahují formuláře pro zobrazení trvalých příkazů, transakcí účtu. V obou stránkách je ve formuláři seznam s účty klienta. Na stránce *transvyber.jsp* je navíc ve formuláři seznam určující dobu za kterou mají být transakce zobrazeny. Akce formulářů jsou *trvprikazy.jsp* a *transakce.jsp.*

- **trvprikazy.jsp, transakce.jsp**

  Uvedené stránky obsahují tabulku všech trvalých příkazů účtu, transakce účtu za zvolenou dobu. Stránka *trvprikazy.jsp* navíc možnost smazání trvalého příkazu.

- **prikazend.jsp**

  Stránky zobrazí zpávu o úspěšně provedené platbě či založení trvalého příkazu. Závisí na vstupních parametrech stránky.

- **detaily.jsp, kontakt.jsp**

  Stránka *detaily.jsp* zobrazí informace o klientovi a formulář s možností změny hesla. Stránka *kontakt.jsp* zobrazí informační a kontaktní údaje na autora.

4 Závěr

Před tvorbou projektu jsem musel nastudovat jazyk Java a tvorbu JSP stránek. Na internetu je většina těchto dokumentů anglicky, což mi nedělalo větší potíže. Jelikož jsem se s těmito technologiemi setkal poprvé, trvalo mi pár týdnů než se mi podařilo obsáhnout potřebné vědomosti a sehnat a nastavit software pro jejich tvorbu. Použité softwarové prostředí mi přišlo velice užitečné a nakonec jsem byl rád, za čas strávený nastavováním a spojováním těchto programů.

Výsledný projekt považuji za velmi úspěšnou prototypovou implementaci zadaného informačního systému. Funkčnost systému jsem odzkoušel na vytvořeném vzorku dat podle kapitoly 3.2.

Tato technologie mi přijde celkem jednoduchá a velice výhodná pro tvorbu dynamických webových stránek. Ovšem předpoklad pro úspěšného programátora této technologie je dobrá znalost anglického jazyka.

Při seznamování s touto technologií jsem narazil na technologie, které jsem nepovažoval za nutné použít, ale při rozšiřování projektu, by bylo vhodné jejich použití. Např. framework Strus, řešení perzistence dat pomocí Hibernace, autentifikace pomocí JAASRealm což je realm využívající technologie JAAS (Java Authentication & Authorization Service).

Literatura

[1] Branický Marek: Java Servlets [seriál online], 3. listopadu 2004. Dostupné z URL <http://interval.cz/serialy/java-servlets/> (květen 2006).

[2] Branický Marek: JavaServer Pages pro všechny [seriál online], 19. února 2003. Dostupné z URL < http://interval.cz/serialy/javaserver-pages-pro-vsechny/> (květen 2006).

[3] Bc. Maixner David: Tvorba aplikací v J2EE [diplomová práce], 2004. Masarykova univerzita, fakulta informatiky v Brně. Dokument dostupný z URL <http://www.slunecnice.cz/product/Tvorba-aplikaci-J2EE/download.html> (květen 2006).

[4] The Apache Jakarta Tomcat 5 Servlet/JSP Container [online dokumentace], 2003. Apache Software Foundation. Dostupné z URL <http://tomcat.apache.org/tomcat-5.0-doc/index.html> (květen 2006)

[5] Marty Hall: Tutorial on Servlets and JSP [online], 1999. Dostupné z URL <http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/> (květen 2006)