JADRO ROZVRHOVACIEHO SYSTÉMU

BAKALÁRSKA PRÁCA

JURAJ MEŠŤÁNEK

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA INFORMATIKY

9.2.1 Informatika

Vedúci práce: RNDr. Mária Pastorová

BRATISLAVA 2008

Abstrakt

Autor:	Juraj Mešťánek
Názov bakalárskej práce:	Jadro rozvrhovacieho systému
Škola:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci bakalárskej práce:	RNDr. Mária Pastorová
Rozsah práce:	32 strán

Bratislava, jún 2008

Bakalárska práca sa zaoberá jadrom podporného rozvrhovacieho systému. Ide o webovskú aplikáciu postavenú na platforme Java, ktorej primárnym cieľom je poskytnúť nástroj na tvorbu a menežovanie rozvrhov v prostrediach, kde automatizovaná tvorba rozvrhov nie je možná. Plynulým spôsobom sa prechádza od otázok vnútornej logiky a návrhu systému, cez samotnú implementáciu s popisom technológií, až po otázky bezpečnosti a stability systému.

Kľúčové slová: podpora rozvrhovania, Spring framework, PostgreSQL

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Ďakujem mojej vedúcej RNDr. Marií Pastorovej za cenné rady pri návrhu systému. Zároveň sa chcem poďakovať spoluautorom Martinovi Madarasovi a Marošovi Bajtošovi, bez ktorých pomoci by táto práca nemohla vzniknúť.

Obsah

Zoznam obrázkov

Úvod

Znakom dnešnej doby je nedostatok času. Preto je veľmi dôležité hľadať spôsoby, ako čas organizovať čo najefektívnejšie. Táto problematika je úzko spätá s tvorbou rozvrhov. Od rozvrhu sa riadi celý časový harmonogram učiteľov aj študentov. Preto má požiadavka dobre spravených rozvrhov svoje opodstatnenie. Základným problémom pri tvorbe rozvrhov je veľké množstvo navzájom závislých dát. Udržiavať tieto dáta v prehľadnom a ľahko dostupnom stave je bez použitia informačných technológií komplikovaná úloha. Po vytvorení rozvrhu zároveň vzniká požiadavka na jeho umiestnenie v elektronickom formáte na internete, čo v prípade manuálneho vytvárania rozvrhov so sebou prináša prácu navyše. Preto prirodzeným riešením procesu rozvrhovania je jeho informatizácia.

Projekt SATKA je informačný systém umožňujúci tvorbu, aktualizáciu a údržbu študijných rozvrhov v súlade s meniacimi sa štruktúrami študijných programov a časovo-priestorových možností. Bol primárne navrhnutý s cieľom optimalizovať tvorbu fakultných rozvrhov v oblastiach časovej náročnosti, požiadavkách na personálne zdroje, minimalizácie kolízií, chýb a ľahkej aktualizácie a údržby. Tieto požiadavky realizuje v troch úrovniach:

- interaktívna tvorba rozvrhov umožňujúca užívateľom zobrazenie aktuálneho stavu, tvorbu rozvrhov a vzájomnú komunikáciu
- systém samostane sledujúci možné kolízie, časové a priestorové obmedzenia, politiku fair play, prioritu nasadzovaných predmetov
- databáza rozvrhov poskytujúca informácie pre rôzne platformy a umožňujúca jednoduchú aktualizáciu bez potreby zásadných štrukturálnych zmien.

Systém je predovšetkým určený do prostredia s veľkou vyťaženosťou jednotlivých miestností a náročnými časovými požiadavkami učiteľov a študentov. Za týchto okolností zvyčajne neexistuje priamočiare riešenie vytvorenia rozvrhu a mnohokrát sú nutné kompromisy. Z tohoto dôvodu úlohou systému nie je automatická tvorba rozvrhov na základe zvolených kritérií, čo by mohlo byť v daných podmienkach kontraproduktívne. Namiesto toho má systém slúžiť ako efektívny interaktívny nástroj na podporu tvorby rozvrhov, poskytujúci kompletnú flexibilitu a logiku narábania s dátami potrebnými na rozvrhovanie. Tohoto cieľu sa snaží dosiahnuť použitím najnovších trendov a technológií v oblasti tvorby sieťových aplikácií. Ďalšou nemenej

dôležitou výzvou, ktorú sa systém snaží riešiť, je poskytnutie rozhrania pre študentov na hľadanie rovrhov, tvorbu vlastných rozvrhov a export rozvrhov do viacerých otvorených formátov.

Cieľom tejto práce je poskytnúť náhľad do logiky, štruktúry a implementácie systému, pričom dôraz sa kladie na aspekty súvisiace s jadrom systému. V prvej kapitole sú podrobne popísané jednotlivé algoritmy a pravidlá, ktoré tvoria základ funkčnosti celého systému. Druhá kapitola sa zaoberá konkrétnou implementáciu daných algoritmov a zároveň poskytuje stručný náhľad do použitých technológií. Tretia kapitola upriamuje pozornosť na otázky bezpečnosti jadra.

Kapitola 1

Návrh systému

Správny návrh a špecifikácia systému značným spôsobom uľahčuje výber správnych technológii a redukuje čas potrebný na implementáciu. Na začiatok treba sformulovať požiadavky na systém a logické vzťahy v ňom, z ktorých sa dá potom vytvoriť databázový model. Po vytvorení modelu je potrebné určiť jednotlivé rozhrania, teda presný popis funkčnosti, ktorý systém bude poskytovať a stanoviť technológie, ktoré sa použijú na implementáciu rozhraní a vecí pridružených k nim. V tejto kapitole bude rozobratý databázový model a logika systému potrebná na rozvrhovanie.

1.1 Základné členenie systému

Z hľadiska funkčnosti sa dá systém rozdeliť na 2 logické celky:

1. Backend - logika systému
2. Frontend - prezentácia dát vonkajšiemu svetu

1.1.1 Backend

Backend je dátová a logická časť systému. Jej cieľom je uchovávať dáta, spracovávať požiadavky, generovať hrubé výstupy a kontrolovať vnútornú logiku programu.

Vnútorne sa delí na 2 časti:

1. databáza - správa a úložisko údajov
2. jadro - menežovanie systému a manipulácia s dátami.

1.1.2 Frontend

Frontend je časť systému, ktorá slúži ako rozhranie medzi užívateľom a Backendom. Jej cieľom je preposielať Backendu užívateľove požiadavky z grafického rozhrania, upravovať výstupy do zobraziteľnej podoby a následne ich v rozumnej forme zobrazovať užívateľovi.

Frontend sa skladá z dvoch častí:

1. prezentačná vrstva - upravuje vstupy a výstupy od užívateľa
2. browser - grafické zobrazenie údajov.

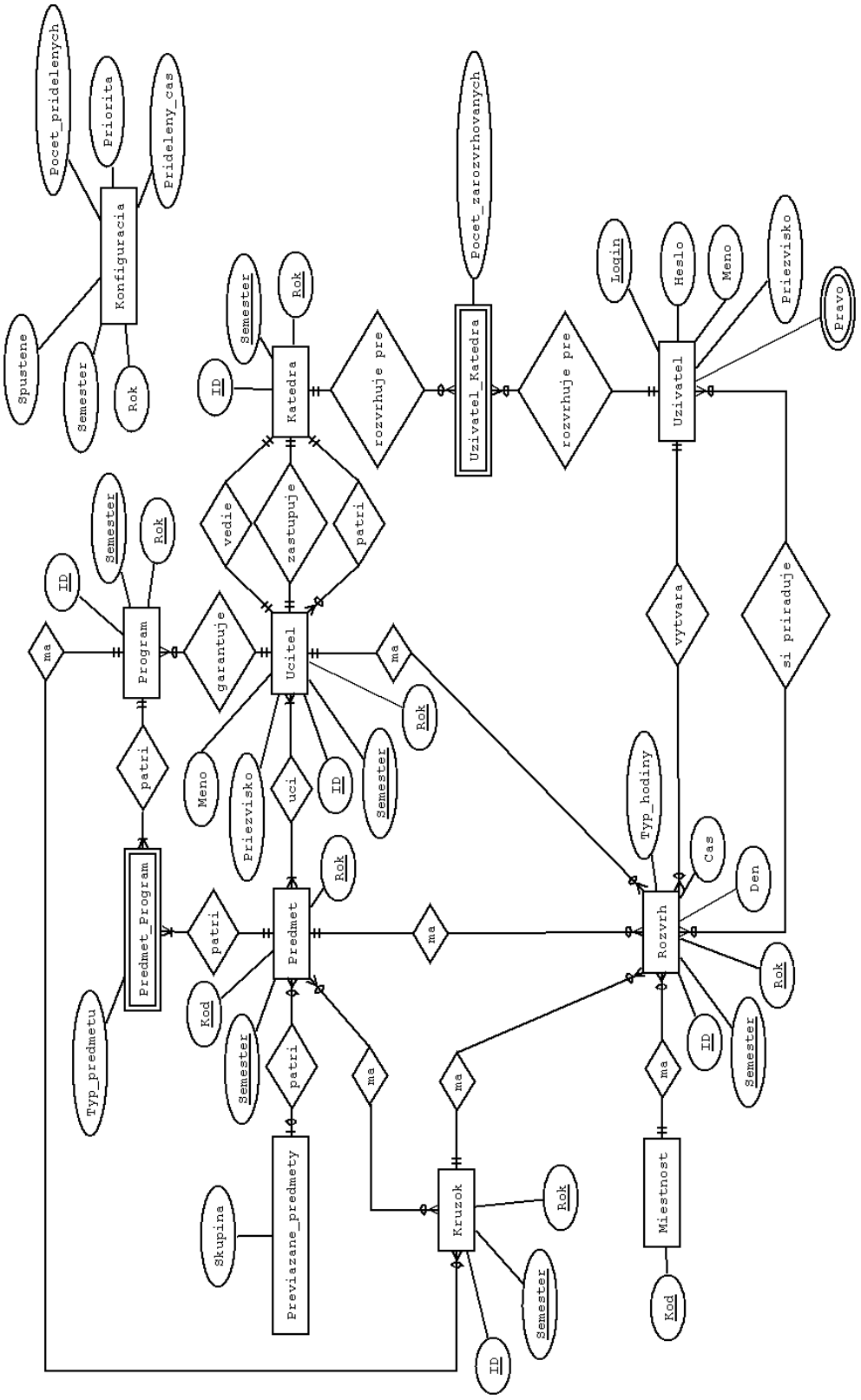
Podrobnejšie je táto časť systému SATKA popísana v práci Martina Madarasa (9).

1.2 Databázový model

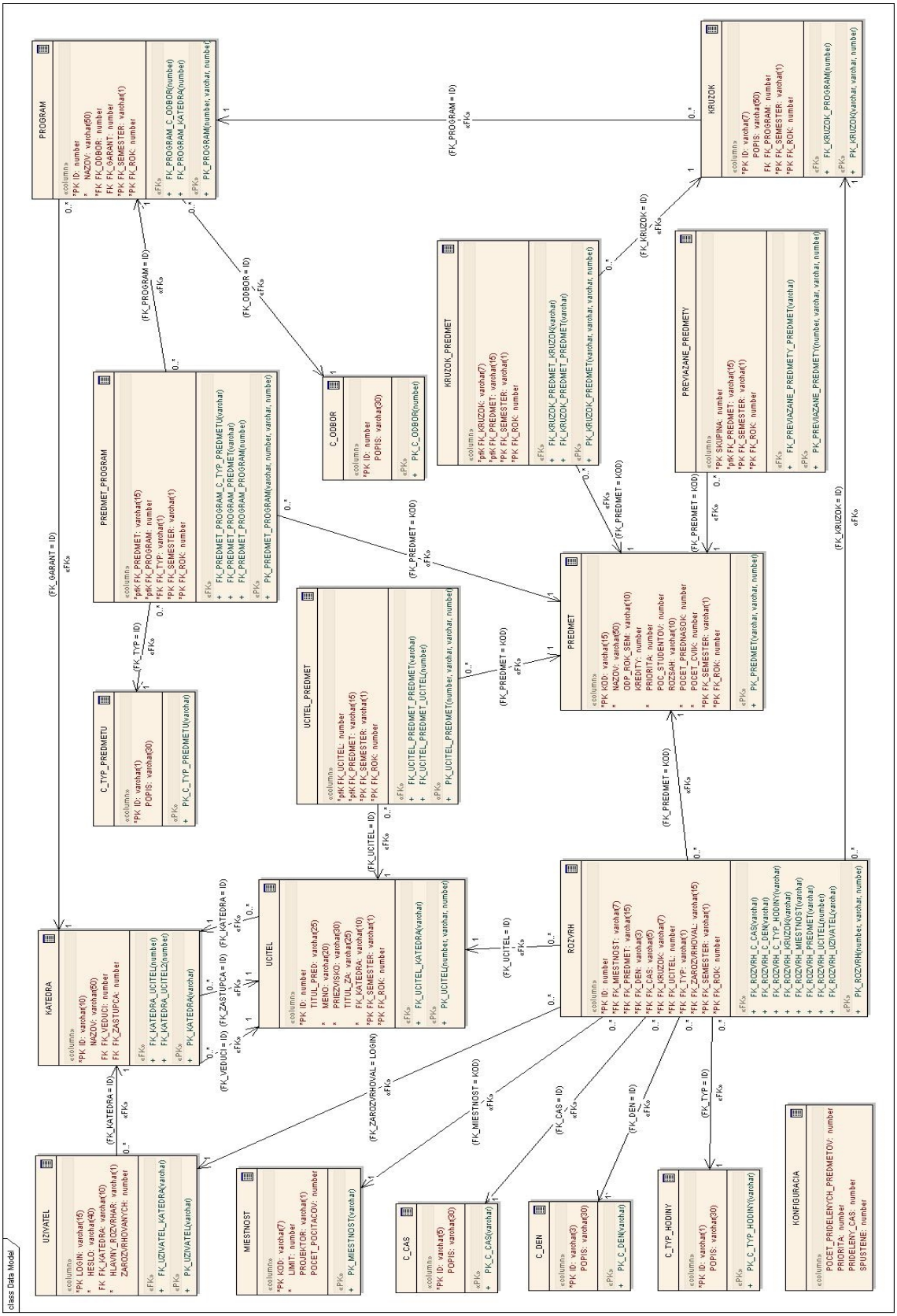
Základná otázka, ktorú bolo treba riešiť pri tvorbe databázového modelu, je štruktúra primárnych kľúčov. Systém musí byť schopný pracovať na úrovni jednotlivých období, ktoré sú v princípe od seba kompletne izolované. Vzhľadom na to, že značná časť entít má prirodzený primárny kľúč skratku a práca so systémom sa uskutočňuje vždy iba v jednom období, tak najefektívnejšie riešenie z hľadiska počtu volaní je použit ako primárny kľúč trojicu (id, semester, rok). Avšak nie všetky databázové entity sú závislé od obdobia. Z tohoto dôvodu sa dajú entity rozdeliť na 2 skupiny:

1. entity závislé od obdobia: Učiteľ, Krúžok, Študijný program, Katedra, Predmet, Rozvrh
2. entity nezávislé od obdobia: Užívateľ, Miestnosť.

Súvislosti medzi entitami názorne ilustrujú nasledujúce obrázky.



Obrázok 1.1: Entitno-relačný diagram



Obrázok 1.2: Databázový diagram

1.3 Typy užívateľov a práva

1.3.1 Štruktúra práv

Právo v systéme je reprezentované ako možnosť volať niektorú z metód systému. V návrhu sú prava zoskupené do skupín *Hosť, Užívateľ, Rozvrhár a Administrátor*, pričom práva sú zoradené od *Hosťa* podľa dôležitosti. Užívateľ môže byť členom viacerých skupín. V návrhu je zohľadnená možnosť prideľovania práv užívateľa nezávisle od skupín.

1.3.2 Hosť

Neprihlásení užívatelia majú prístup k rozvrhom aktuálneho obdobia a taktiež môžu prehliadať zverejnené tabuľky entít. Navyše majú možnosť registrácie nového účtu s právami *Užívateľ*.

1.3.3 Užívateľ

Táto skupina reprezentuje bežných užívateľov, ktorí si vytvoria v systéme svoj vlastný účet. Vytvorenie vlastného účtu sprístupňuje nasledovné funkcie:

- nastavenie personalizovaného vzhľadu
- prehliadanie rozvrhov zo starších období
- tvorbu vlastného rozvrhu vo zvolenom období
- export vlastných rozvrhov.

1.3.4 Rozvrhár

Skupina *Rozvrhári* je hlavná funkčná skupina celého systému. Táto skupina má kompletný read-only prístup k databáze a zároveň práva ako skupina *Užívatelia*. Každý užívateľ tejto skupiny má pridelených 0 alebo viac katedier z ktorých môže nasadzovať učiteľov. Užívateľov tejto skupiny vytvárajú *Administrátori*. Skupina môže v obmedzenej miere narábať s rozvrhom.

Činnosti, ktoré môžu Rozvrhári robiť s rozvrhom:

- pridávanie nových záznamov do tabuľky *Rozvrh*

Nasadzovať rozvrhy môže užívateľ len učiteľom, ktorí učia na katedre, ku ktorej má užívateľ prístupové práva. Zároveň je táto činnosť podmienená správnymi systémovými nastaveniami a bezkonfliktnosťou daného záznamu s ostatnými záznamami v tabuľke *Rozvrh* a *Predmet*(počet hodín). Zadávanie konfliktného záznamu je umožnené cez účet zo skupiny *Administrátori*.

- vymazanie a úprava záznamov z tabuľky *Rozvrh*

Užívateľ môže vymazať a upravovať iba tie záznamy, ku ktorým má vlastnícke práva.

1.3.5 Administrátor

Užívateľská skupina *Administrátori* má najvyššie povolené práva. Táto skupina môže narábať so všetkými dostupnými metódami. To znamená, že môže prakticky bez obmedzení manipulovať s databázou a nastavovať systémové tabuľky. Primárnym cieľom tejto skupiny je menežovať rozvrhovanie a robiť prípadné zmeny v rozvrhu, na ktoré skupina *Rozvrhári* nemá práva. V ideálnom prípade sa *Administrátor* stará iba o správne nastavenia rozvrhovania.

Administrátor má pri rozvrhovaní prístup k týmto nastaveniam:

- meniť aktuálny semester a rok systému
- nastavovať minimálnu prioritu predmetov na rozvrhovanie
- nastavovať čas, dokedy sa môže rozvrhovať
- spúšťať a zastavovať rozvrhovanie
- nastaviť maximálny počet hodín, ktoré môžu rozvrhári v danom sedení nasadiť
- resetovať rozvrhárom počet zarozvrhovaných predmetov pre katedry.

1.4 Core - jadro

Jadro plní nasledovné úlohy:

1. poskytovanie rozhraní pre Frontend
2. komunikácia s databázou
3. kontrola rozvrhovacej logiky
4. správa chýb
5. kontrola autentifikácie.

1.4.1 Rozvrhovacia logika

Ďalej sa bude pod pojmom *rozvrh* rozumieť jeden riadok tabuľky *Rozvrh*. V praxi to znamená, že pokiaľ sa nasadí viac hodín, ktoré reprezentujú jedno ucelené vyučovanie, tak v systéme sa vytvorí samostatný *rozvrh* pre každú z týchto hodín. Z hľadiska užívateľa a databázy dá postupné nasadenie viacerých hodín rovnakého vyučovania taký istý výsledok, ako nasadenie hodín po jednom.

Nasadenie rozvrhu

Pri nasadení rozvrhu musí byť splnená niektorá z nasledujúcich skupín podmienok:

1. - prihlásený užívateľ je *Administrátor*
   - je použité prebitie konfliktu v prípade, ak by daný *rozvrh* narušoval konzistentnosť
2. - prihlásený užívateľ je *Rozvrhár*
   - prihlásený užívateľ má nastavený rovnaký semester a rok ako je zadaný v systémových nastaveniach
   - rozvrhovanie je spustené
   - užívateľ má prístupové práva ku katedre učiteľa, ktorý je pre daný *rozvrh* nasadzovaný
   - užívateľ nepresiahol pridelený limit počtu zarozvrhovaných predmetov pre katedru nasadzovaného učiteľa
   - nasadzovaný predmet má aspoň takú prioritu, aká je nastavená v tabuľke *Nastavenia*
   - daný *rozvrh* nespôsobuje konflikt konzistentnosti.

Konflikty

Pri rozvrhovaní môžu nastať situácie, ktoré by mohli narušiť konzistentnosť rozvrhu. Všetky tieto situácie sú uvádzané vzhľadom na atribúty nasadzovaného *rozvrhu*. Medzi tieto situácie patrí nasledovné:

- krúžok už má nasadený iný *rozvrh* v tom istom čase
- učiteľ už má nasadený iný *rozvrh* v tom istom čase a v inej miestnosti
- už existuje iný nasadený učiteľ v tom istom čase a miestnosti
- už existuje *rozvrh* v tom istom čase a miestnosti ale s predmetom, ktorý nepatrí do rovnakej skupiny
- po zarozvrhovaní počet pridelených prednášok alebo cvík presiahne stanovený limit pre nasadzovaný predmet.

1.4.2 Úprava existujúcich rozvrhov

Úprava rozvrhov sa riadi podľa tých istých pravdiel ako ich vytváranie. Avšak existuje jeden podstatný rozdiel. Pri úprave sa musí brať do úvahy vlastník *rozvrhu*. Vlastník *rozvrhu* je užívateľ, ktorý *rozvrh* nasadil. Užívatelia zo skupiny *Administrátori* môžu upravovať všetky *rozvrhy* bez ohľadu na vlastníctvo. Užívatelia zo skupiny *Rozvrhári* môžu upravovať iba *rozvrhy*, ktoré sami nasadili. Pri kontrole pravidiel a konzistencie sa kontroluje databáza v stave bez upravovaného *rozvrhu*.

Výmena rozvrhov

V niektorých prípadoch je nutné, aby si užívatelia vedeli bezpečne vymeniť svoje *rozvrhy* v kontexte miestnosti a času bez toho, aby museli čokoľvek mazať. Na toto slúži funckia *Výmena rozvrhov*. Vstupom tejto funkcie je dvojica (*rozvrh1*, *rozvrh2*), pričom volajúci tejto funkcie musí vlastniť *rozvrh1*. Ak dvaja užívatelia použijú správne túto funkciu na rovnaké rozvrhy, tak dôjde k ich výmene, čo je ekvivalentné zámene času a miestnosti. Taktiež je možné si zamienať vlastné rozvrhy.

1.4.3 Vymazávanie rozvrhov

Pri vymazávaní rozvrhov musí byť splnená niektorá z nasledujúcich podmienok:

- prihlásený užívateľ je *Administrátor*
- zároveň sú splnené všetky nasledujúce podmienky:

1. prihlásený užívateľ je *Rozvrhár*
2. prihlásený užívateľ má nastavený rovnaký semester a rok ako je zadaný v systémových nastaveniach
3. *rozvrh*, ktorý sa ide vymazávať, je vlastnený daným užívateľom .

1.4.4 Vyhľadávanie rozvrhov

Vyhľadávanie rozvrhov je jedna z najdôležitejších funkčných súčastí celého systému. Predovšetkým je potrebné, aby bola zabezpečená dostatočná možnosť výberu kritérií, čo dokáže významným spôsobom urýchliť proces tvorby rozvrhov.

Vyhľadávanie rozvrhov v systéme sa dá robiť na základe nasledovných kritérií:

- zoznam časových intervalov
- semester
- rok
- zoznam študijných programov, v ktorých majú byť predmety
- miestnosť
- predmet
- krúžok
- učiteľ
- typ vyučovacej hodiny
- vlastník *rozvrhu*.

1.4.5 Vyhľadávanie voľných miestností

Voľná miestnosť je v systéme jednoznačne chápana ako pätica (miestnosť, hodina, deň, semeser, rok), pre ktorú neexistuje v tabuľke *Rozvrh* žiadny záznam.

Kritéria selektujúce vlastnosti *voľných miestností*:

- minimálny počet ľudí
- minimálny počet počítačov
- projektor.

Kritéria selektujúce časové vlastnosti *voľných miestností*:

- zoznam časových intervalov
- semester
- rok.

Kritéria zabezpečujúce odstránenie *voľných miestností* v čase, v ktorom by po nasadení rozvrhu vznikol konflikt:

- učiteľ
- krúžok.

Kapitola 2

Technológie backendu a implementácia

Táto kapitola sa zaoberá prechodom od návrhu k implemetácii. Všetky technológie jadra sú postavené na platforme *J2EE*. Ide o štandardnú Javu rožírenú o sadu vývojárskych knižníc. Tieto knižnice slúžia na tvorbu moderných business aplikácií určených predovšetkým pre sieťové prostredie.

Implementácia aplikácie bola uskutočnená vo vývojovom prostredí *Oracle JDeveloper*. Ide o voľne dostupný a technicky veľmi vyspelý software na tvorbu *J2EE* aplikácií. Poskytuje širokú sadu nástrojov na pohodlnú prácu s kódom v rôznych formátoch a taktiež nástroje na tvorbu diagramov.

2.1 Skratky

V tejto kapitole sa používajú nasledovné skratky:

- AJAX - Asynchronous JavaScripting And XML
- AOP - Aspect-oriented Programming
- AS - aplikačný server
- DB - databáza
- DAO - Data Access Object
- HTTP - Hypertext Transfer Protocol
- J2EE - Java 2 Enterpise Edition
- JDBC - Java Database Connectivity
- JSON - JavaScript Object Notation

- JSP - Java Server Pages
- OOP - Object-oriented Programming
- SQL - Structured Query Language.

2.2 Štruktúra aplikácie

Štruktúra aplikácie z hľadiska technológií sa dá rozdeliť na 3 nezávislé celky:

1. browser
2. *JBoss* aplikačný server
3. *PostgreSQL* databázový server.

Browser

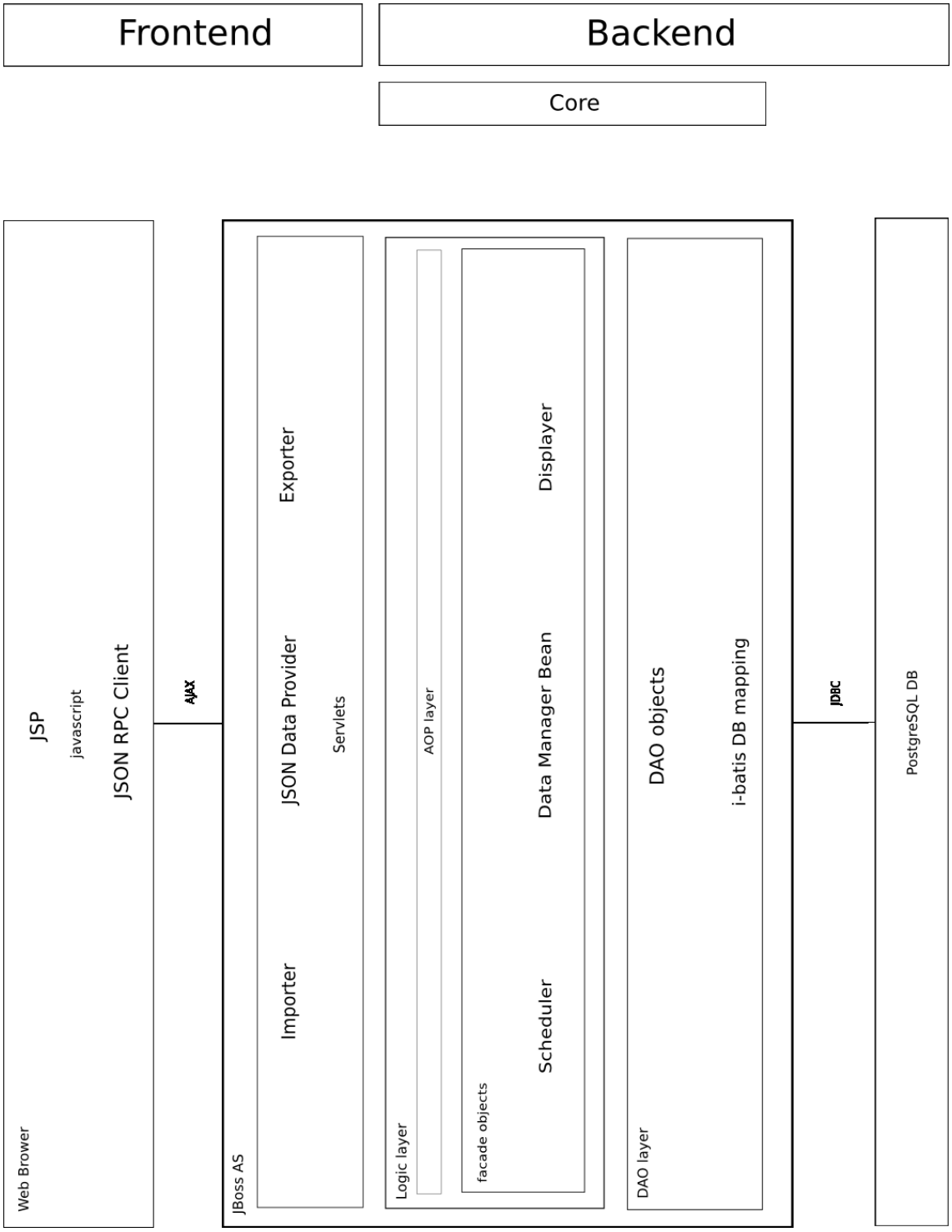
Technológia v browseri je postavená na báze JSP a JS. Volania medzi browserom a AS sa uskutočňujú pomocou technológie AJAX. Na strane browsera sa tieto volania uskutočňujú pomocou JSON RPC klienta, ktorý transformuje JS objekty do JSON notácie a následne ich posiela servletom bežiacim na AS. Po prijatí odpovede na konkrétnu správu dôjde k jej spracovaniu pomocou JS.

JBoss AS

AS pozostáva z prezentačnej vrstvy a jadra. Prezentačná vrstva pozostáva zo sady špecializovaných servletov, z ktorých každý plní inú sadu úloh. Ich hlavným poslaním je ale prijímať požiadavky od užívateľov, príslušným spôsobom ich spracovať alebo posunúť jadru a následne poskytnúť odpoveď. Úlohy jadra a jeho vrstvy budú podrobne popísané v nasledujúcich sekciách.

PostgreSQL databázový server

Posledná vrstva slúži ako zdroj a úložisko dát celého systému. K serveru sa pristupuje pomocou JDBC drivera za pomoci SQL dotazov.



Obrázok 2.1: Schéma štruktúry aplikácie

2.3 PostgreSQL

K systému sa dá stiahnuť voľne dostupný JDBC driver. Pomocou neho sa dokážu javovské aplikácie napojiť na *PostgreSQL* server a komunikovať s ním.

2.4 JBoss

Aplikácia SATKA je je spúšťaná pomocou aplikačného servera *JBoss* verzie 4.2.2. Ide o voľne dostupný open-sourcový aplikačný server so zabudovaným servletovým kontajnerom *Apache Tomcat* 5.5, ktorý beží na platforme Java EE 1.4 alebo 1.5 . *JBoss* podporuje tieto technológie: *Clustrovanie*, *Failover*, *Load balancig*, *Distribuované cachovanie*, *EJB3*, podpora *Aspect-Oriented Programming*, *Hibernate*, *J2EE-Web Services*, *JMS*, *JCS*, *JAAC*, *JSP/Servlet*, špecifikácia *EJB 2.1*, *RMI-IIOP*, *JTA*, *JDBC*, *SAAJ*, *JNDI*, *JAAS*, *JavaMail*, Deployment API, Management API. (7)

2.5 Spring Framework

Spring je open-sourcový aplikačný framework pre Javu. Dá sa povedať, že ide o určitú náhradu alebo dokonca vylepšenie klasického *Enterprise JavaBean*(J2EE) modelu. Jeho sila spočíva predovšetkým v snahe ponúknuť programátorom prostriedky na efektívnu tvorbu softwaru mimo bežne zaužívaných techník používaných v minulosti. *Spring* je vďaka tomu tiež známy ako framework, ktorý ako prvý ponúkol predtým novú funkcionalitu a zaviedol je do mainstreamových vývojárskych praktík. *Spring framework* sa dá považovať za kolekciu niekoľkých menších modulov. Väčšina z týchto modulov dokáže pracovať samostatne ale všetky dokopy vytvárajú jeden efektívny nástroj na tvorbu komplexných aplikácií. (3)

Spring framework pozostáva z týchto modulov:

- **Inversion of Control kontajner**: konfigurácia aplikačných komponentov a správa javovských objektov
- **Aspect-oriented programming framework**: pracovanie s funkcionalitou, ktorú nemožno implementovať do javovského OOP modelu bez robenia kompromisov
- **Data access framework**: narábanie so systémami na správu relačných databáz postavených na Jave s použitím *JDBC* a objektovo relačných mapovacích nástrojov
- **Transaction management framewrok**: harmonizácia rôznych nástrojov na správu transakcií a nastaviteľný menežment transakcií na javovských objektoch

- **Model-view controller framework**: framework postavený na HTTP a Servletoch poskytujúci funkcionalitu na ich rozšírenie a rôzne nastavenia
- **Remote Access framework**: nastaviteľný export a import javovských objektov cez počítačové siete pomocou technológií *RMI, CORBA* a protokolov postavených na HTTP vrátene webovských služieb (*SOAP*)
- **Authentication and authorization framework**: nastaviteľný menežment autentifikácie a autorizačných procesov podporujúci mnoho populárnych štandardov, protokolov, nástrojov a techník, celé zastrešené pod podprojektom *Acegi*
- **Remote Management framework**: nastiviteľné poskytovanie a menežment objektov na lokálnu alebo vzdialenú správu cez *JMX*
- **Messaging framework**: vylepšené posielanie správ pomocou *JMS* vzhľadom na štandardné *JMS* rozhrania
- **Testing framework**: podporné triedy na testovanie.

Princíp fungovania väčšiny týchto modulov bude objasnený v ďalších sekciách. V tejto časti už rozoberiem iba *Inversion of control kontajner* a štruktúru nastavovacích súborov.

2.5.1 Inversion of control

Jadrom celého *Spring frameworku* je *Inversion of control* kontajner, ktorý poskytuje nástroje na plnohodnotné menežovanie Java objektov. Jednou z najdôležitejších úloh kontajneru je správa životného cyklu objektov. To zahŕňa tieto aspekty:vytváranie objektov, nastavovanie objektov, volania inicializačných metód, predávanie vytvorených objektov iným objektom.

Objekty vytvorené pomocou *Springu* sa nazývajú beany. Jednotlivé beany sú zvyčajne nastavené pomocou konfiguračného XML súboru, ktorý obsahuje ich definície. Tieto definície obsahujú všetky informácie potrebné na vytvorenie objektu. Akonáhle sú objekty vytvorené bez akýchkoľvek chýb, tak sa stávajú dostupnými na použitie.

Objekty môžu byť spätne získané pomocou *Dependency lookup* alebo *Dependency injection*. *Dependency lookup* je pattern, kde žiadateľ chce získať od kontajnerového objektu objekt na základe špecifického názvu alebo špecifického typu. *Dependency injection* je pattern, kde kontajner posúva objekty na základe ich názvu iným objektom a to buď pomocou konštruktorov, atribútov alebo factory metód.(8)

2.5.2 Konfiguračné súbory

Nasledujúci zoznam súborov sú všetky konfiguračné súbory, ktoré sa v systéme vyskytujú a obsahujú nastavenia jednotlivých objektov *Springu*.

Konfiguračné súbory:

- **Satka-applicationContext.xml**: obsahuje chrbticové nastavenia *Springu*, napojenie na DB, definície takmer všetkých bean, nastavenia AOP objektov
- **Satka-applicationContext-Security.xml**: kompletné nastavenia modulu *Acegi Security*
- **Satka-applicationContext-Logging.xml**: nastavenie názvu logovacej beany
- **SatkaCore-log4j.xml**: nastavenie logovacích súborov a ich správania
- **Satka-sqlMapConfig.xml**: nastavenie prístupov k jednotlivým mapám modulu *Ibatis*
- **SQL mapy**: predpripravené SQL mapy modulu *iBatis* v balíku sk.uniba.fmph.satka.core.dao.maps .

2.6 Výnimky jadra

Všetky výnimky v systéme sú potomkami 2 základných tried a to tried *SatkaException*, ktorá je potomkom *Exception* a *SatkaRuntimeException*, ktorá je potomkom triedy *RuntimeException*. Obidve tieto triedy majú zadefinovaný atribút *errorId*, ktorým sa v potomkoch týchto tried označuje typ chyby. V prípade vyvolania niektorej výnimky sa výnimka nezachytáva v jadre, ale posiela sa do Frontendu, kde sa spracuje.

Potomkovia triedy *SatkaException*:

Ide o triedy *SatkaDataAccessException* a *UserException*. Tieto výnimky sú definované v rozhraniach ako výnimky hádzané metódami jadra. Ide vlastne o výnimky hádzané v chrbticovej štruktúre programu.

Potomkovia triedy *SatkaRuntimeException*:

Potomkami tejto triedy sú *SatkaAuthenticationException*,*InconsistencyException* a *AuthorizationException*. Všetky tieto výnimky sú hádzané v AOP vrstve.

2.7 DAO vrstva

DAO (Data Access Objects) vrstva je vrstva pokrývajúca rozhrania na pripojenie k databáze. Každý jednotlivý DAO objekt reprezentuje samostatnú jednotku na prístup a správu jednej databázovej entity, pričom k iným entitám prístup nemá. Táto vrstva zároveň zabezpečuje zachytenie a následnú interpretáciu databázových výnimiek.

2.7.1 iBatis

Implementácia DAO rozhraní je vykonaná pomocou špecializovaného SQL mapperu *iBatis*. Bol navrhnutý ako samostatný modul *Springu*, teda jeho nasadenie vyžaduje minimálne úsilie z hľadiska nastavenia prepojení. Modul *iBatis* umožňuje jednoduchú konverziu databázových dotazov na javovské objekty, čo značným spôsobom redukuje zdrojový kód. Opačne taktiež umožňuje jednoduché namapovanie atribútov javovských objektov do dotazu. Silou tohoto mapperu sú predovšetkým dynamické dotazy, pomocou ktorých sa dajú veľmi ľahko generovať aj zložité dotazy na filtre. Všetky mapovacie SQL dotazy sú umiestnené v externých XML mapách. To znamená, že zmena správania dotazu sa dá spraviť z vonku bez zásahu do implementácie.

2.7.2 Štruktúra DAO objektov

DAO objekt je objekt typu *SqlMapClientDaoSupport* ktorý implementuje jedno DAO rozhranie. Na správne fungovanie tohoto objektu musí byť pomocou *Springu* nastavený *DataSource* tj. v našom prípade pripojenie na *PostgreSQL* a taktiež atribút *sqlMapClient*, čo je objekt spravujúci konfiguračné mapy *iBatisu*. Na prístup k namapovaným dotazom v metódach rozhrania sa používa objekt *SqlMapClientTemplate*, ku ktorému sa dá dostať pomocou metódy *getSqlMapClientTemplate()*. Ten zabezpečuje zavolanie konkrétneho namapovaného dotazu a predanie parametrov volania.

Program obsahuje nasledujúce DAO rozhrania:
ConstantsDAO, SettingsDAO, UserDAO, TeacherDAO, RoomDAO, GroupDAO, SubjectDAO, StudyProgramDAO, ScheduleDAO, DepartmentDAO

2.7.3 Štruktúra iBatis máp

Jedna *iBatis* mapa tvorí ucelený celok nastavení a mapovaní. Mapy sú usporiadané podľa svojich názvov do takzvaných *namespaces*. V implementácií jedna mapa korešponduje s mapovaniami pre jednu entitu.

Mapa pozostáva z nasledujúcich prvkov, z ktorých každý musí mať svoj unikátny identifikátor v rámci *namespace*:

- *typeAlias* - aliasy javovských tried
- *resultMap* - mapovania výsledkov dotazov do javovských objektov
- *resultClass* - jednoduché triedy na vrátenie výsledkov (string, int)
- *parameterClass* - beana, z ktorej sa získavajú vstupy do dotazu. Špeciálne je možné použiť triedu *java.util.Map*
- *parameterMap* - alternatívne mapovania atribútov javovských objektov
- *statement,insert,delete,update,select* - mapovania dotazov, pričom ako parametre sa používajú predchádzajúce prvky.

2.7.4 Výnimky DAO vrstvy

Výnimky hádzané v DAO vrstve sú všetky potomkami *SatkaException*. Po zachytení výnimky sa určí typ a *errorId* problému a následne sa vytvorí nová výnimka, ktorá je preposielaná vyššie.

Hadzané výnimky v DAO vrstve sa delia na 2 druhy:

1. výnimky spôsobené databázovými volaniami
2. výnimky spôsobené nesprávnymi parametrami metód.

Výnimky spôsobené databázovými volaniami

Ide o výnimky typu *DataAccessException*, ktoré sú súčasťou *Springu*. Preposielaná výnimka je typu *SatkaDataAccessException*.

Výnimky *DataAccessException* sa v programe delia na 2 skupiny:

1. *DataIntegrityViolationException* - vzniká pri narušení integrity DB
   errorId = *DATA_INTEGRITY_VIOLATION*
2. Ostatné výnimky typu *DataAccessException*
   errorId = *GENERAL_DATABASE_EXCEPTION*.

Výnimky spôsobené nespravnymi parametrami metód

Ide o výnimky spôsobené samotným používateľom. Preposielaná výnimka je typu *UserException*.

Tieto výnimky môžu byť vyvolané nasledovnými spôsobmi:

1. zadanie neexistujúceho atribútu na vyhľadávanie entít
   errorId = *ATTRIBUTE_DOES_NOT_EXIST*
2. zavolanie metódy s nesprávnymi vlastnosťami parametrov
   errorId = *INCORRECT_OR_INCOMPLETE_ARGUMENTS*.

2.8 Logická vrstva

2.8.1 Facade objekty

Facade objekty sú špecializované objekty, ktoré zgrupujú rozhrania DAO objektov a prepájajú jednotlivé DAO objekty. Sú to najvyššie postavené objekty z hľadiska hierarchie jadra a poskytujú rozhrania na volania z *Frontendu*. Zároveň sa na nich vytvára AOP vrtsvta, ktorá zabezpečuje ďalšiu funkcionalitu. Každá z poskytovaných metód je z hľadiska transakcií atomická. Túto vlastnosť taktiež zabezpečuje AOP vrstva. Vďaka tomu dochádza v týchto objektoch k bezpečnému deleniu jednotlivých požiadaviek na viac menších.

V programe existujú 3 facade rozhrania a k nim pridružené implementácie:

1. DataManagerBean
2. DisplayerBean
3. SchedulerBean.

DataManagerBean

Rozhranie *DataManagerBean*, poskytuje prístup k metódam DAO objektov okrem metód rohrania *SchedulerDAO*. Ide vlastne o veľký wrapper DAO objektov. Špeciálna kontrola sa robí na metódach *UserDAO*, ktoré upravújú užívateľské preferencie, kde dochádza ku kontrole hesla.

DisplayerBean

Rozhranie *DisplayerBean* poskytuje niektoré metódy rozhraní *SchedulerDAO* a *RoomDAO*, menovite metódy na prezeranie a filtrovanie rozvrhov a vyhľadávanie voľných miestností v rozvrhu.

SchedulerBean

Rozhranie *SchedulerBean* poskytuje metódy rozhrania *SchedulerDAO*, ktoré nejakým spôsobom manipulujú s rozvrhom. Zároveň sa na implementáciu tohoto rozhrania napájajú všetky kontrolné objekty AOP vrstvy.

2.8.2 AOP vrstva

AOP - *aspect-oriented programming* je pomerne nová programovacia paradigma. Jej hlavným princípom je tzv. *cross-cutting*. Ide vlastne o vstúpenie do volania metódy pred tým, ako sa vykoná, bez zmeny jej implementácie. Tento princíp poskytuje iný pohľad na modularizáciu programov a striktné oddelenie čistej štruktúry aplikácie od jej kontrolných a servisných častí.

Implementácie *DataManagerBean* a *DisplayerBean* sú obalené do nasledovných AOP objektov:

1. ProxyLogger
2. TransactionController
3. SatkaServiceSecurity.

Implementácia *SchedulerBean* je obalená do nasledovných AOP objektov:

1. ProxyLogger
2. TransactionController
3. SatkaServiceSecurity
4. InputController
5. FairPlayController
6. ScheduleController
7. UserAdjuster.

2.8.3 Kontrola rozvrhovania

Kontrolu rozvrhovania zabezpečujú tieto AOP objekty:
InputController, FairPlayController, ScheduleController, UserAdjuster.

InputController

Úlohou tohoto objektu je kontrolovať viachodinové vstupy. Tieto vstupy sa musia líšiť iba v čase, a to po poradí a vždy iba o jednu hodinu. V prípade, že táto podmienka nie je splnená, tak je hodená výnimka typu *UserException* a *errorId INCORRECT_OR_INCOMPLETE_ARGUMENTS*

FairPlayController

FairPlayController kontroluje správnosť nastavení rozvrhovania a správne databázové vzťahy medzi rozvrhárom a nasadzovaným rozvrhom. V prvom kroku určí, o akú rozvrhovaciu metódu ide a následne na základe toho spustí niektoré z kontrolných metód, ktoré v prípade neúspešnej kontroly hádžu výnimku *AuthorizationException*.

Kontrolné metódy:

1. *checkRunning* - kontrola, či je rozvrhovanie spustené a či je ešte rozvrhovanie v rozvrhovacom čase
   errorId = *SCHEDULING_DISABLED*
2. *checkValidTime* - kontrola správnosti nastavenia obdobia.
   Rozvrhár musí mať obdobie nastavené podľa aktuálneho obdobia a nasadzovať rozvrh na aktuálne obdobie.
   errorId = *INCORRECT_USER_TIME_SETTINGS*
3. *checkSubjectPriority* - Predmet musí mať aspoň takú prioritu, aká je v nastaveniach rozvrhovania.
   errorId = *LOW_SUBJECT_PRIORITY*
4. *checkTeacherDepartment* - Rovrhár musí vlastniť práva ku katedre nasadzovaného učiteľa.
   errorId = *BAD_USER_DEPARTMENT*
5. *checkSchedulesCount* - Po nasadení predmetu nesmie počet nasadených hodín presiahnuť počet pridelených hodín daného predmetu
   errorId = *SCHEDULE_COUNT_EXCEEDED*
6. *checkScheduleOwner* - Nasadený rozvrh môže modifikovať iba jeho vlastník
   errorId = *BAD_SCHEDULE_OWNER*.

Pri volaní metód sa kontrolujú metódy nasledovným spôsobom:

- *createSchedules*(Nasadenie rozvrhov) - *checkRunning, checkValidTime, checkSubjectPriority, checkTeacherDepartment, checkSubjectCount*
- *setSchedules*(Úprava rozvrhov) - *checkRunning, checkValidTime, checkScheduleOwner, checkSubjectPriority, checkTeacherDepartment, checkSubjectCount*

- *deleteSchedules*(Mazanie rozvrhov) - *checkRunning, checkValidTime, checkScheduleOwner*
- *exchangeSchedules*(Výmena rozvrhov) - *checkRunning, checkScheduleOwner, checkValidTime*
- *deleteExchangeRequest*(Zmazanie požiadavky na výmenu) - *checkScheduleOwner*.

ScheduleController

Funkciou AOP objektu *ScheduleController* je kontrola konfliktov pri volaní metód *createSchedules* a *setSchedules*. Tieto kontroly sa riadia presne podľa pravidiel popísaných v podsekcií *Rozvrhovacia logika* kapitoly *Návrh systému*. Po prekontrolovaní všetkých podmienok sa v prípade existencie nekonzistencie všetky nasadené rozvrhy, ktoré sú v konfikte s práve nasadzovanými, pošlú ako parameter výnimky typu *InconsistencyException* s *errorId INCONSISTENT_DATA_INPUT*.

UserAdjuster

UserAdjuster modifikuje počet zarozvrhovaných hodín daného rozvrhára v danom sedení. K úprave dôjde len v prípade úspešného vytvorenia rozvrhov.

2.8.4 Transakcie

Správa transakcií je uskutočňovaná pomocou AOP objektu TransactionController, ktorý je typu *org.springframework.transaction.interceptor.TransactionInterceptor*. Ten má pomocou atribútu *transactionAttributes* nastavené správanie na jednotlivých metódach. Momentálna implementácia nastavuje tento atribút pre všetky metódy na *PROPAGATION_REQUIRED*. Toto nastavenie spôsobí rollback transakcie pri ľubovoľnej chybe počas behu danej metódy. Celá konfigurácia transakcií je nastavená v konfiguračnom súbore *Satka-applicatonContext.xml*.

2.8.5 Logovanie

Logovanie je dôležitá súčasť každého systému. V aplikácií je logovanie vyriešené pomocou špecializovaného AOP objektu *ProxyDebugLogger*. Ten má 2 atribúty, ktoré sa nastavujú v *Satka-applicationContext*: *logger* a *logAttributes*. Atribút *logger* je objekt typu *org.apache.log4j.Logger*. Jeho inštancia je vytvorená pomocou *Springu*, pričom jeho konfigurácia, ktorá zahŕňa umiestnenie, formátovanie a správanie logovacieho súboru, je umiestnená v konfiguračnom súbore *SatkaCore-log4j.xml*. Atribút *logAttributes* nastavuje špecifické potreby logovania pri jednotlivých metódach. Konkrétne ide o nastavenie *SKIP_ARGUMENTS*, ktoré sa používa pri volaniach s heslom alebo volaniach, kde sú parametre príliš veľké.

Volania jadra systému sú logované v 2 fázach:

1. zaloguje sa samotné volanie metódy s argumentami, ak sú povolené a následne sa spustí volaná metódy
2. v prípade vyskytnutia výnimky sa zaloguje vzniknutá výnimka aj s argumentami volania, ak sú povolené.

2.8.6 Acegi Security

Bezpečnosť a riadenie prihlasovania je v systéme riadené pomocou modulu *Acegi Security*. Ide o štandardný modul používaný aplikáciami postavenými na *Springu*. V systéme plní tento modul nasledovné úlohy:

- autentifikácia
- autorizácia volaní
- správa sedení.

Autentikácia

Autentifikácia v zmysle modulu *Acegi Security* spočíva v pridelení špeciálneho objektu danému vláknu na základe požiadavky z prihlasovacieho servletu, ktorý počúva na nastavenej adrese. Samotné rozhodovanie o autentifikácií prebieha v objekte typu *org.acegisecurity.providers.ProviderManager* (id = *SatkaAuthenticationManager*), ktorý menežuje objekty typu *org.acegisecurity.providers.AuthenticationProvider*. V systéme existuje jediný takýto objekt a to objekt typu *SatkaAuthenticationProvider*. V metóde *authenticate()* tohoto objektu sa sa najprv pristúpi do DB a skontroluje správny login a heslo. V prípade, že všetko prebehne úspešne, metóda vráti inštanciu objektu *SatkaUserAuthentication*, ktorý je potomkom triedy *User* a implementuje *Acegi* rozhranie *org.acegisecurity.Authentication*. Tento objekt obsahuje všetky údaje o danom užívateľovi a je pevne spojený s aktuálnym sedením. Následne je poslaný browseru cookie objekt, ktorým browser pri požiadavkách na server identifikuje svoje sedenie. V prípade neúspešnej autentifikácie je užívateľ presmerovaný na stránku s oznamom o neúspešnom prihlásení.

Autorizácia volaní

Autorizácia volaní prebieha na úrovni AOP vrstvy a je zabezpečená *Acegi* objektom typu *org.acegisecurity.intercept.method.aopalliance.MethodSecurityInterceptor* (id = *SatkaServiceSecurity*). Ten má nastavený objekt *SatkaAuthenticationManager*, pomocou ktorého získava informácie o aktuánom sedení, objekt *SatkaAccessDecisionManager* implementujúci rozhranie *AccessDecisionManager* a atribút *objectDefinitionSource*, ktorý nastavuje prístupové práva pre jednotlivé metódy. V prípade zavolania niektorej z metódy sa najprv identifikuje objekt *SatkaUserAuthentication* pridelený sedeniu, následne sa zistia parametre prislúchajúce danej metódy z atribútu *configDefinitionSource* a zavolá sa metóda *decide()* objektu *SatkaAccessDecisionManager*. Tá na základe práv užívateľa a parametrov metódy buď prejde, alebo hodí

výnimku typu *org.acegisecurity.AccessDeniedException*.

Správa sedení

Pre prístup k aktuálnemu sedeniu existuje v systéme objekt *CurrentUserHolder*. Ten má jedinú metódu *getCurrentUser*, ktorá pomocou statickej metódy objektu *org.acegisecurity.context.SecurityContextHolder* vráti objekt *SatkaUserAuthentication* pridelený aktuálnemu vláknu.

Kapitola 3

Bezpečnosť backendu

Bezpečnosť je jedným z kľúčových aspektov väčšiny moderných systémov. Pri návrhu bezpečnosti treba predovšetkým brať do úvahy kto všetko má mať prístup k danému systému a akým spôsobom môžu jednotliví užívatelia využívať funkcionalitu poskytovanú vonkajšiemu svetu. Zároveň treba poznať technológie, ktoré sú na implementáciu použité, zabrániť zneužitiu ich slabín a použiť ich silné stránky na optimalizáciu otázky bezpečnosti. Táto kapitola sa zaoberá otázkami bezpečnosti toku informácií na serveri.

3.1 Zabezpečenie servera

Pri nastavovaní servera *JBoss* treba zabezpečiť 2 prvky. Prvým z nich je konzola, pomocou ktorej sa dá spravovať celý server. Zabezpečenie tohoto prvku spočíva v nastavení prihlasovacieho mena a hesla. Ďalším prvkom je povolenie a nastavenie *https* protokolu v konfiguračných súboroch servera. Na tento krok je potrebné zaobstarať alebo vytvoriť dvojicu verejný/súkromný kľúč a v ideálnom prípade si nechať verejný kľúč certifikovať u certifikačnej autority za určitý poplatok. Certifikácia nie je nutná. Zahŕňa však možnosť napadnutia útokom *Man in the middle*.

3.2 SQL injection

SQL injection je technika, pri ktorej sa vstup DB dotazov modifikuje spôsobom, ktorý vyvolá neželaný výstup pôvodného dotazu. V prípade predpripravených dotazov ide o vsunutie kusu SQL kódu namiesto hodnoty. Tento útok môže mať neželané následky. Nielenže sa úžívateľ môže dostať k dátam ku ktorým by inak nemal prístup, ale zároveň môže spôsobiť vážne škody na dátach.

Zabránenie útoku:

Správu predpripravených dotazov zabezpečuje modul *iBatis*. V prípade reťazcov

a číselných hodnôt je modul proti SQL injection zabezpečný. Jediné riziko predstavuje priame nastavovanie kusu SQL kódu, čo sa v sytéme deje iba pri nastavovaní poradia zobrazenia. Tento problém je vyriešený použitím filtrovania vstupu a pomocou aliasov pri nastavovaní poradia.

3.3 Prístupové práva

Problematika typov účtov a implementácie prístupových práv už bola podrobnejšie popísana v predchádzajúcich kapitolách. Jednotlivé práva sú reprezentované ako prístup k volaniam metód jadra. To je podmienené korektným prihlásením užívateľa a teda znalosťou správneho prihlasovacieho mena a hesla a príslušnosťou užívateľa do jednej z užívateľských skupín.

3.4 Bezpečnosť hesiel

Bezpečnosť hesiel je jednou zo základných priorít, s ktorou si návrhár systému musí poradiť. Nie je dôležité mať heslá len bezpečne uložené, zároveň treba zabezpečit, aby žiadny človek, dokonca ani správca DB, nemohol zistiť, aké heslo používajú jednotliví užívatelia. To by mohlo totiž viesť k jeho zneužitiu pokiaľ by užívateľ používal to isté heslo aj na iných systémoch.

Bezpečnosť hesla v DB:

Prístup k DB je samozrejme podmienený znalosťou prihlasovacieho mena a hesla. V samotnej DB sú heslá uskladané pomocou fukcie *crypt*, ktorá sa nachádza v knižnici *pgcrypto*. Ide o jednu zo štandarných knížníc databázy *PostgreSQL*. Funkcia *crypt()* je špeciálna hašovacia funkcia na hašovanie hesiel. Jej vykonanie trvá veľmi dlho kvôli kompenzácií malej dĺžky hesiel. To čiastočne zabraňuje použitiu *brute force*. Zároveň sa pri jej realizácií používa takzvaný *salt*. Ide o nahodný zhluk znakov, ktorým sa docieli, že rovnaké heslá sú v DB reprezentované rôznymi reťazcami. To zabraňuje použitiu spätného získania hesla z predpripravených hašovacích tabuliek s heslami. Táto funkcia je taktiež adaptibilná voči rýchlosti počítačov. To znamená, že pokiaľ sa zvýši výkon počítača, tak je možné funkciu upraviť tak, aby bola ešte pomalšia.(4)

Bezpečnosť prístupu k heslám v DB:

Heslá v DB sú dôsledne izolované od systému. To znamená, že neexistuje žiadna metóda, ktorou by sa dalo získať hašované heslo uložené v DB. Jediný prístup k heslám je pomocou metódy *checkPassword*, ktorej parametrami sú login a heslo. Haš zadaného heslo sa v DB porovná s uloženým heslom a z DB sa vráti 't' ak bolo porovnanie úspešné, inak sa vráti 'f'.

Bezpečnosť zadaných hesiel v systéme:

Heslo po zadaní užívateľom putuje v systéme a je dočasne uložené v objekte sedenia pre potreby narábania s užívateľkými nastaveniami. Bezpečnostné riziko v tomto procese predstavujú formy logovania. Volania jednotlivých metód sa logujú aj s ich parametrami, čo by následne umožňovalo prezeranie hesiel v logu. Preto bola do logovacieho AOP objektu pridaná možnosť odstránenia logovania parametrov pri citlivých funkciách, ktorých názvy sú nastavené v konfiguračnom súbore *Satka-applicationContext.xml*. Po tejto korektúre sa v systéme nikde okrem DB neukladajú informácie o heslách užívateľov.

Záver

Ciele vytýčené v úvode boli splnené. Vznikla tak stabilná a bezpečná aplikácia, ktorá má všetky predpoklady byť reálne využívaná v praxi a zároveň pomôcť zefektívniť spôsob vytvárania a prezentácie rozvrhov. Implementácia prebehla bez výraznejších komplikácií a to predovšetkým vďaka špecifikácií. Voľba *Spring* frameworku výrazným spôsobom urýchlila vývoj celej aplikácie a zjednodušila implementáciu problematických častí systému.

Systém je funkčný a predstavuje solídny základ pre ďalšie rozširovanie. Potenciál je predovšetkým v možnosti robenia štatistík o jednotlivých predmetoch resp. v integrácií systému, ktorý túto funkčnosť podporuje a následnom prepojení s procesmi rozvrhovania.

Literatúra

[1] Eckel, B.:*Myslíme v jazyku Java - knihovna programátora*
Grada, 2000, ISBN 80-247-0027-1

[2] Eckel, B.:*Myslíme v jazyku Java - knihovna zkušeného programátora*
Grada, 2000, ISBN 80-247-9010-6

[3] http://www.springframework.net/ (2008-05-30)

[4] http://www.postgresql.org/docs/8.2/static/index.html (2008-05-30)

[5] http://www.jboss.org/jbossas/docs/ (2008-05-30)

[6] http://ibatisnet.sourceforge.net/DevGuide.html (2008-05-30)

[7] http://en.wikipedia.org/wiki/JBoss_application_server (2008-05-30)

[8] http://en.wikipedia.org/wiki/Spring_framework (2008-05-30)

[9] Madaras, M.:*Prezentačná vrstva webovej aplikácie na tvorbu rozvrhov na FMFI*
Bakalárska práca, FMFI UK, 2008

Dodatok A

Prílohy

K práci je priložené CD obsahujúce podporný rozvrhovací systém SATKA.