Obsah

1. Úvod ... 3
1.1. Ciele diplomovej práce ... 4
1.2. Členenie diplomovej práce ... 4
1.3. Rozdelenie práce na projekte ... 5
1.4. Označenia, skratky a terminológia ... 6
2. Ajax ... 8
2.1. Bližší pohľad na technológiu Ajax ... 9
2.2. Architektúra Ajaxu ... 9
2.3. Ajax ako „asynchrónny“ ... 10
2.4. Možnosti tvorby interaktívneho webu ... 11
2.5. Možné spôsoby implementácie Ajaxu ... 12
2.6. XMLHttpRequest ... 13
2.7. Dynamické generovanie HTML dokumentu. ... 18
2.8. Kedy použiť Ajax ... 19
2.9. Obmedzenia Ajaxu a omyly pri jeho použití ... 22
2.10. Bezpečnosť v Ajaxe ... 26
2.11. Podpora Ajax technológií v novodobých prehliadačoch. ... 26
3. Problematika tvorby rozvrhu ... 27
3.1. Všeobecná problematika tvorby rozvrhu ... 27
3.2. Odlišnosť problematiky tvorby rozvrhu na FMFI ... 28
3.3. Prínos navrhovaného systému tvorby rozvrhov ... 29
4. Analýza a špecifikácia ... 31
4.1. Popis procesu tvorby rozvrhu na fakulte ... 31
4.2. Analýza problému ... 32
4.3. Špecifikácia ... 33
5. Návrh riešenia ... 38
5.1. Prostredie ... 38
5.2. Dátový model ... 39

5.3. Rozdelenie aplikačného okna 44
5.4. Menu 45
5.5. Činnosti na zadávanie vstupných údajov 45
5.6. Činnosť Pedagógovia 46
5.7. Činnosť Predmety 50
5.8. Činnosť Miestnosť 56
5.9. Činnosť Rozvrhári 56
5.10. Činnosť na nasadzovanie rozvrhu 57
5.11. Činnosť Reporty 62
5.12. Validácia a verifikácia integrity dát 62
5.13. Bezpečnosť 63
5.14. Ukážka implementácie 64
6. Záver 74
7. Použitá literatúra a zdroje 75
7.1. Zoznam obrázkov 77

1. Úvod

Ako budú vyzerať webové aplikácie (stránky) o pár rokov? Ťažká otázka, na ktorú odpovie sám čas. S určitosťou však vieme povedať, že web nebude taký, aký ho poznáme dnes, takisto ako dnes nie je nie je taký, aký sme ho poznali pred piatimi rokmi. Informačné technológie patria v súčasnosti k najviac sa rozvíjajúcim odvetviam. Na trh prichádza mnoho nových technológií a postupov. Niektoré sa „ujmú“, iné zapadnú do zabudnutia, či už preto, že ich vytlačia iné technológie s lepším postupom alebo predajným marketingom, alebo sú príliš zložité a preto sa s nimi používatelia „nezžijú“.

Jednou z takýchto nových technológií je aj technológia Ajax, ktorá maže hranice rozdielov medzi webovými aplikáciami a „klasickými“ desktopovými aplikáciami. Webové aplikácie využívajúce technológiu Ajax prinášajú používateľovi komfort práce porovnateľný s desktopovou aplikáciou. Načítavanie časti stránok na pozadí, automatické odosielanie formulárov, automatická zmena obsahu zobrazeného dokumentu podľa požiadaviek používateľa. Pre ilustráciu spomeniem projekt Writley alebo ThinkFree – kompletný kancelársky balík v internetovom prehliadači. [1]

Samotný Ajax nie je technológia v pravom slova zmysle, skôr ide o techniku tvorby webových aplikácií s využitím kombinácie už existujúcich technológií (X)HTML, CSS, JavaScriptu, DOM, XML. Isto nám napadne otázka, prečo sa to nepoužívalo už skôr. Odpoveďou je možno ošúchaná odpoveď, ale *nebola na to vhodná doba.*[12] Podpora jednotlivých technológií bola v minulosti na strane prehliadačov rôzna, nehovoriac už o ich vzájomnej nekompatibilite, čo spôsobilo, že tieto technológie nebolo možné naplno využiť. V poslednom čase a s nástupom najnovších verzií internetových prehliadačov sa situácia zlepšila a tak môžu vývojári internetových aplikácií vytvárať lepšie a používateľsky príťažlivejšie webové aplikácie.

1.1. Ciele diplomovej práce

Prvým cieľom diplomovej práce je oboznámiť čitateľa s technológiou Ajax, s jej využitím pri tvorbe dynamických webových aplikácii. Popísať súčasné možnosti tvorby webových aplikácii a porovnať ich s technológiou Ajax. Ukázať spôsob použitia XMLHttpRequestu ako hlavného nástroja Ajaxu. Poukázať na možnosti, kedy je vhodné technológiu Ajax použiť a naopak, v ktorých prípadoch sa jej použitiu vyvarovať. Čitateľ by mal po prečítaní tejto diplomovej práce vedieť v prípade potreby teoreticky aj prakticky používať technológiu Ajax.

Druhým cieľom diplomovej práce je na komplexnej webovej aplikácii prakticky demonštrovať použitie technológie Ajax. Uvítal som preto požiadavku od pedagogických pracovníkov fakulty FMFI UK, ktorí sa už dlhšie zamýšľajú nad aspoň čiastočnou náhradou manuálneho procesu tvorby rozvrhu za elektronický. Dostal som možnosť prakticky vyskúšať svoje teoretické poznatky ohľadom tvorby softvérového produktu. Mojím cieľom je navrhnúť počítačový program, ktorý by pomohol pedagogickým pracovníkom s tvorbou fakultného rozvrhu s dôrazom na praktické použitie technológie Ajax. Popísať problematiku tvorby fakultného rozvrhu, vytvoriť analýzu súčasného spôsobu tvorby rozvrhu, špecifikovať požiadavky, a podrobne popísať návrh riešenia. Vytvorený návrh systému tvorby rozvrhu by mal pedagogickým pracovníkom ukázať spôsob tvorby fakultného rozvrhu elektronickou cestou.

1.2. Členenie diplomovej práce

Prvá kapitola diplomovej práce čitateľa oboznámi s hlavnou témou tejto práce, s jej cieľom a predpokladaným prínosom, členením práce a s logickým rozdelením práce na jednotlivé časti.

Druhá kapitola podrobne popisuje technológiu Ajax. Zaoberá sa možnosťami tvorby interaktívnych webových aplikácií, pri ktorých jedným z riešení je aj použitie technológie Ajax. Popisuje jeho architektúrou, spôsob implementácie a bližšie sa teoreticky a prakticky venuje popisu XMLHttpRequestu, ako hlavnému stavebnému kameňu Ajaxu. Poukazuje na správne i nesprávne spôsoby použitia technológie Ajax. V závere druhej kapitoly je vymenovaná podpora v dnešných prehliadačoch.

Tretia kapitola sa zaoberá problematikou tvorby rozvrhu. Porovnáva spôsob tvorby štandardného rozvrhu a špecifické požiadavky rozvrhu na fakulte FMFI UK.

Opisuje nevýhody dostupných štandardných riešení a poukazuje na výhody navrhovaného riešenia.

Štvrtá kapitola rozoberá súčasný spôsob tvorby rozvrhu, zaoberá sa analýzou a špecifikáciou navrhovaného systému.

V piatej kapitole je popísaný podrobný návrh riešenia fakultného rozvrhového systému. V závere kapitoly je ukážka implementácie s dôrazom na ukážku funkčného spôsobu realizácie s využitím technológie Ajax.

V závere práce sú zhrnuté dosiahnuté výsledky a celkové zhodnotenie diplomovej práce.

1.3. Rozdelenie práce na projekte

Prácu na diplomovej práci som rozdelil do nasledujúcich základných častí:

- Zistenie požiadaviek na rozvrhový systém
- Zistenie súčasného stavu rozvrhového systému fakulty
- Oboznámenie sa so spôsobom tvorby rozvrhu na fakulte, návšteva kompetentných osôb, zaoberajúcich sa tvorbou fakultného rozvrhu.
- Analýza spôsobu tvorby rozvrhu, a porovnanie tvorby rozvrhu pre rôzne cieľové skupiny.
- Popis systému tvorby rozvrhov na fakulte
- Analýza a špecifikácia navrhovaného fakultného rozvrhového systému
- Zvolenie vhodného databázového systému a navrhnutie databázovej štruktúry
- Vytvorenie návrhu fakultného rozvrhového systému
- Čiastočná implementácia s dôrazom na overenie postupov vytvárania Ajax aplikácie.
- Zhodnotenie, poukázanie na možnosti zlepšenia a rozšírenia

1.4. Označenia, skratky a terminológia

Informačné technológie sú v súčasnosti jedno s najrýchlejšie sa rozvíjajúcich odvetví a používaná terminológia je preberaná zväčša z anglického jazyka. V slovenskom jazyku táto terminológia nie je ešte dostatočne zaužívaná a zjednotená. Doslovné preklady anglickej terminológie mnohokrát zvádzajú čitateľa a plne nevystihujú podstatu pojmov. Preto niektoré z termínov budem uvádzať v anglickom jazyku aj vďaka predpokladu, že čitateľov má aspoň čiastočné znalosti informačných technológií a nebude problém pochopiť takto podaný obsah.

Ajax (Asynchronous JavaScript and XML) – technika tvorby webových stránok využívajúca viacero dostupných technológií ako CSS, DOM, JavaScript, XML, XMLHttpRequest podporovaných súčasnými prehliadačmi.

CSS (Cascading Style Sheets) – Kaskádové štýly. Kolekcia metód pre grafickú úpravu webových stránok. Zobrazovanému prvku webovej stránky je priradený určitý grafický štýl. Atribúty štýlov sú definované v hierarchickej štruktúre po vrstvách a ak niektorý štýl nemá definovaný určitý atribút, preberá tento atribút od svojho nadradeného štýlu.

DOM (Document Object Model) – Objektový model dokumentu. Aplikačné programové rozhranie, ktoré definuje všeobecný štandard pre prístup k akémukoľvek platnému HTML dokumentu, alebo správne štruktúrovanému XML dokumentu. Toto rozhranie je nezávislé na použitom programovacom jazyku.

Desktopový program – Program bežiaci lokálne na počítači.

HTML – Kódovací jazyk používaný k vytváraniu formátovaných dokumentov na webe. Html kód je textového charakteru a používa k formátovaniu značky, tzv. návestia (tagy). Klientsky program (internetový prehliadač) Html kód interpretuje a prevádza do vizuálnej podoby.

Http Request – Požiadavka, ktorú vyšle klient (internetový prehliadač) smerom k webserveru. Webserver na základe tejto požiadavky vráti klientovi požadovaný obsah.

JavaScript – Skriptovací jazyk, vytvorený ako odnož jazyka Java firmami Sun Microsystems a Netscape. Ide o jednoducho použiteľný programovací prostriedok podporovaný internetovými prehliadačmi a používaný hlavne na obsluhu a vytváranie interaktívnych webových stránok.

Klient – Hierarchicky podriadená súčasť systému využívajúca služby vyššej úrovne. Vo vzťahu k webovým stránkam sa pod pojmom klient rozumie internetový prehliadač, ktorý komunikuje s nadriadeným webserverom.

Proprietárny – uzavretý (pred vonkajším svetom). Proprietárny program – program, ktorý je chránený autorskými právami a ku ktorému nie je zverejnený zdrojový kód. Používateľ tak nemá možnosť si overiť deklarovanú funkčnosť programu.

Session – 1. Doba behu programu. 2. Doba, po ktorú medzi sebou dva počítače (programy) udržujú spojenie a prenášajú informácie.

W3C (World Wide Web Consorcium) - Organizácia, ktorá definuje štandardy www. Jej členmi sú komerčné firmy, ktoré činnosť tejto organizácie financujú, no ako celok je organizácia od nich nezávislá.

Webserver – Počítač, pripojený do internetovej siete, poskytujúci službu www, t.j. zasiela používateľom na žiadosť ich programov (internetový prehliadač) webové stránky. Webserver musí byť pre korektnú funkcionalitu trvalo pripojený do siete.

XHTML – Rozšírenie HTML jazyka, ktoré plne podporuje obsah vo formáte XML.

XML (Extensible Markup Language) – Rozšírený značkovací jazyk. Všeobecný jazyk na popis dokumentov, ktorý na rozdiel od HTML okrem obsahovej informácie nesie aj informáciu, čo jednotlivé dáta obsahu znamenajú.

XMLHttpRequest – Rozšírenie funkcionality internetových prehliadačov, ktoré umožňuje skriptu na strane klienta vyslať na server Http požiadavku, a obdržať vrátený obsah vo formáte textu alebo HTML.

2. Ajax

Pri vyslovení slovíčka Ajax sa väčšine ľudí vynorí v mysli spojitosť so známym futbalovým klubom Ajax Amsterdam. Tí, čo ešte nezabudli na učivo dejepisu si spomenú na antického hrdinu a nakoniec, ľudia zaujímajúci sa o svet informačných technológií potvrdia, že o tom v poslednom čase počuť stále viac v spojitosti s takzvanou ďalšou generáciou webu, niekedy nazývanou aj web 2.0, alebo aj HTML 5.

V skutočnosti sa už s použitím technológie Ajax stretol každý, kto sa trošku aktívnejšie pohybuje v internetovom priestore. Ako príklad spomeniem Google Suggest (Google našeptávač) [2], Google email [3], Google maps [4], Mapy atlas.sk [5], ThinkFree, Writley [6] – online office a hádam najstaršia Ajax aplikácia Web Outlook Express. Všetky tieto vyššie spomenuté webové stránky majú spoločnú funkcionalitu a to načítavanie určitej časti dát na základe užívateľského podnetu bez toho, aby webová stránka bola opäť celá načítaná. Napríklad spomenuté Google suggest pri písaní vyhľadávacieho reťazca priebežne ponúkajú počet relevantných výrazov začínajúcich na už napísanú časť vyhľadávanej frázy. Google maps pri zmene aktuálnej pozície na mape priebežne na pozadí načítava chýbajúce časti mapy.

Obr. 1. Ukážka Google našeptávača

Z pohľadu užívateľa sú webové stránky využívajúce Ajax technológie interaktívnejšie, rýchlejšie, načítavajú len informácie, o ktoré má používateľ záujem. Potláčajú bezstavovosť klasických webových stránok a viac približujú webové aplikácie k desktopovým aplikáciám, čo sa týka interaktivity a komfortu ovládania.

2.1. Bližší pohľad na technológiu Ajax

Názov Ajax vznikol ako skratka z výrazu „Asynchronous JavaScript and XML“. Samotný Ajax nie je technológia v pravom slova zmysle, skôr ide o techniku tvorby webových aplikácií s využitím kombinácie (X)HTML, CSS, JavaScriptu, DOM, XML. Všetky uvedené technológie tu už nejaký čas existujú, ale vďaka nedostatočnej podpore na strane internetových prehliadačov v minulosti sa masívne začínajú používať až dnes.

Najčastejším spôsobom použitia Ajaxu je asynchrónna komunikácia webového prehliadača so serverom pomocou objektu XMLHttpRequest, avšak nie je to nutnou podmienkou. Ajax aplikácia, aj keď jej názov je zložený zo slov asynchrónny, JavaScript a XML, vôbec nemusí komunikovať asynchrónne, ako skriptovací jazyk nemusí byť použitý JavaScript a dáta sa nemusia prenášať pomocou XML.

Na rozdiel od „klasickej“ webovej aplikácie spoznáme Ajax aplikáciu podľa toho, že sa javí ako desktopový program bežiaci lokálne na vašom počítači. Aj keď sú zobrazené dáta na základe používateľových požiadaviek priebežne aktualizované, samotná stránka nie je kompletne opätovne načítavaná (refresh).

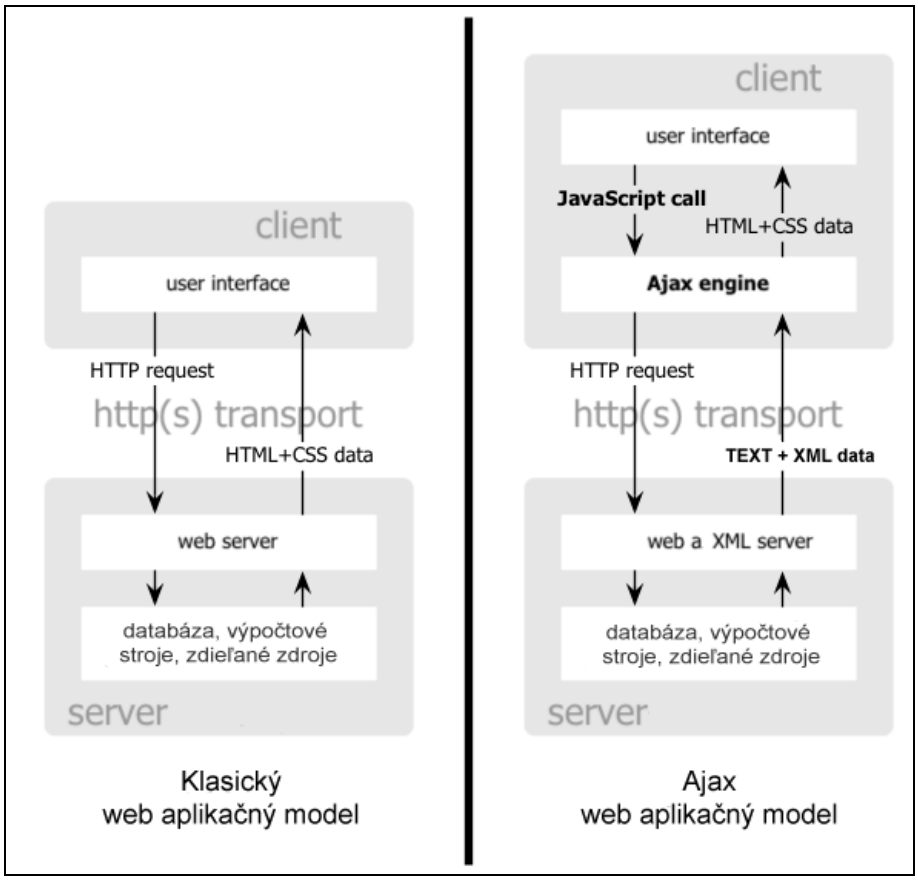
2.2. Architektúra Ajaxu

Najskôr popíšeme klasický web aplikačný model, ktorý potom porovnáme s Ajax web aplikačným modelom.

Klasický web aplikačný model pracuje nasledovne. V klientovi (internetovom prehliadači) zachytí používateľovu požiadavku používateľské rozhranie klienta a vyšle http požiadavku smerom na webserver. Daný webserver požiadavku spracuje, čiže vyberie požadované dáta z databáz, vykonáva nad nimi výpočty, komunikuje s ďalšími systémami a spracovanú požiadavku vráti späť klientovi v podobe HTML a/alebo CSS dát. Klient takto vrátenú HTML stránku zobrazí.

Ajax web aplikačný model obsahuje na strane klienta ešte jednu medzivrstvu, nazývanú tiež *Ajax engine*. Najčastejšie, ako som už spomínal vyššie, túto vrstvu predstavuje JavaScript. Požiadavku používateľa teda v prvom rade zachytí obslužný JavaScriptový kód. Ak ide o lokálnu požiadavku ako napr. kontrola používateľom vložených dát, vykoná ju. Ak je však požadovaná komunikácia so serverom, vyšle sa táto požiadavka pomocou XMLHttpRequestu na webový alebo XML server. Server túto požiadavku, podobne ako v prvom prípade, spracuje a vyšle späť ku klientovi dáta

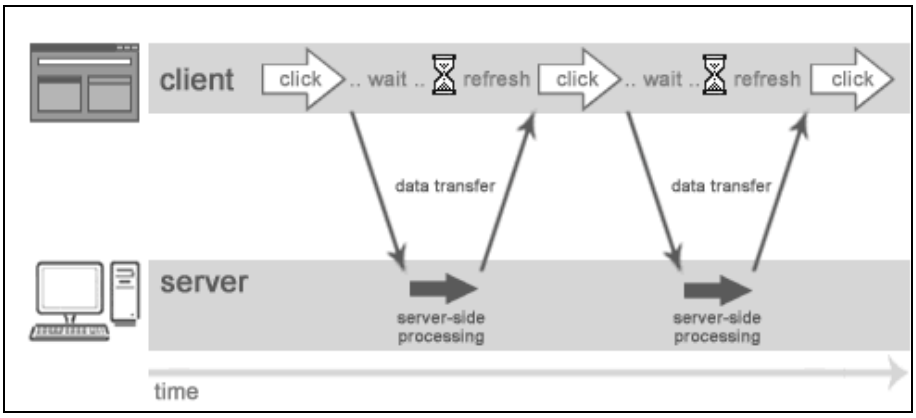
v podobe XML. Ajax engine tieto dáta zachytí, spracuje ich a pošle ich vo forme HTML alebo CSS na zobrazenie.



Obr. 2. Porovnanie klasického a Ajax web aplikačný modelu [9]

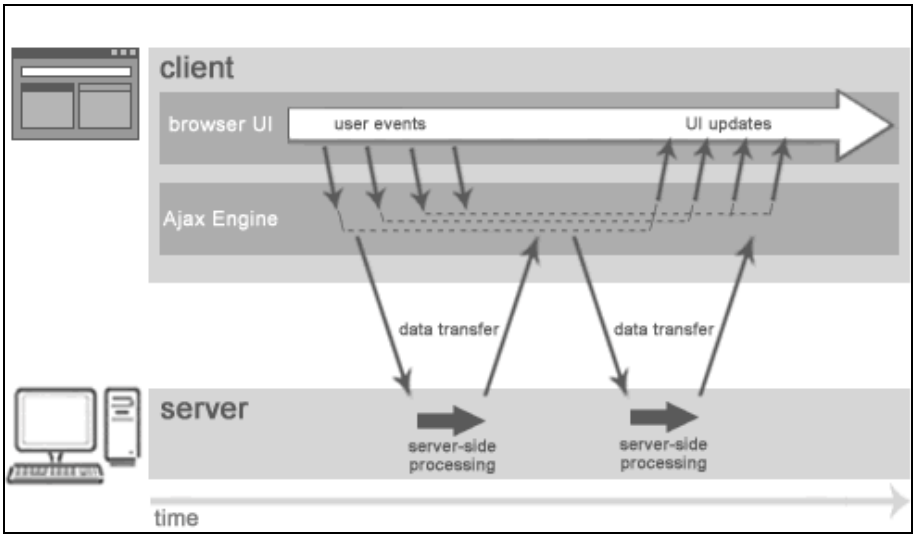
2.3. Ajax ako „asynchrónny“

V klasickom web aplikačnom modeli prebieha komunikácia synchrónne ako požiadavka/odpoveď. Klient (prehliadač) vždy inicializuje požiadavku a čaká na odpoveď servera. Po spracovaní požiadavky server vráti odpoveď prehliadaču, ktorý prijaté dáta zobrazí (prekreslí celú obrazovku). Z pozície používateľa potom prebieha komunikácia formou klik-čakaj-refresh (prekreslenie).



Obr. 3. Klasický web aplikačný model – synchrónna komunikácia [13]

Podstatná výhoda aplikácií postavených na Ajaxe je práve ich asynchrónna komunikácia klienta so serverom. Nahradením synchrónneho modelu požiadavka/odpoveď docielime v prvom rade to, že po požiadavke môže používateľ naďalej aktívne pracovať s obsahom webové stránky, zatiaľ čo sa jeho požiadavka spracováva na pozadí. Po spracovaní požiadavky sa zmení len príslušný obsah zobrazenej stránky. Používateľ môže zadávať ďalšie požiadavky, bez ohľadu na to, či už bola predchádzajúca požiadavka spracovaná.



Obr. 4. Ajax web aplikačný model – asynchrónna komunikácia [13]

2.4. Možnosti tvorby interaktívneho webu

V tejto časti si popíšeme rôzne spôsoby riešenia, akými môžeme postupovať pri umiestňovaní a spracovávaní dynamických prvokov na stránke.

Tradičné riešenie

Tradičné riešenie predstavuje to, že akcia je realizovaná bežným odkazom alebo formulárom. Po vykonaní akcie sa celá stránka načíta opäť. Nevýhody tohto riešenia sú zrejmé – pomalosť a dátová náročnosť pri nutnom nahradzovaní starého obsahu stránky za nový.

Spracovanie akcie v novo otvorenom okne

O niečo lepšie riešenie, v prípade že nepotrebujeme meniť obsah aktuálnej stránky. Treba si však uvedomiť, že veľa používateľov má vo svojich prehliadačoch blokované otváranie nových okien.

Umiestnenie dynamického komponentu do samostatného rámca

Elegantné riešenie u jednoduchých stránok, ktoré využívajú dynamické komponenty prevádzkované na inom serveri. Avšak použitie rámcov je v striktnom HTML zakázané, preto sa aj napriek svojej funkčnosti neodporúča.

Využitie skriptovania u klienta

Je to pravdepodobne najprácnejšie, ale zároveň prináša najväčší komfort práce pre používateľa. Realizovať sa dá viacerými spôsobmi:

- presmerovaním akcie do neviditeľného rámca, ktorý skriptom zmení pôvodnú stránku
- dynamickým vytvorením elementu `<script>` a jeho umiestnením do HTML dokumentu
- použitím objektu XMLHttpRequest, ktorý bol pre potreby komunikácie skriptov so serverom priamo navrhnutý.

Posledne spomínaný prístup pomocou XMLHttpRequest využíva nami popisovaná technológia Ajax.

2.5. Možné spôsoby implementácie Ajaxu

Ako už bolo povedané, pri implementácii Ajaxu nemusí byť nutne použitý JavaScript, ale napríklad aj Java alebo Flash. Ako uvidíme, každé ma svoje výhody aj nevýhody a predurčuje každého kandidáta na použitie v iných druhoch aplikácii.

Java (Java + XML)

Pri použití Javy sa typicky využíva odľahčená verzia Javy určená pre internetové prehliadače, ktorá zabezpečuje výpočty na strane klienta, vykonáva zmenu časti zobrazených dát a asynchrónnu komunikáciu so serverom, najčastejšie pomocou XML. Medzi výhody tohto použitia patrí robustnosť a spoľahlivosť Javy a jej objektovo orientovaný prístup. Je preto výhodné pre skúsených Java vývojárov.

Flash (ActionScript + SWF)

Flash na strane klienta využíva proprietárny *Flash engine* od spoločnosti Macromedia s ActionScript knižnicou pre výpočty na strane klienta. Komunikácia so serverom prebieha pomocou proprietárneho formátu SWF. Ako hlavné nevýhody tohto riešenia vidím práve proprietárnosť použitých formátov ako aj fakt, že Flash nie je štandardne obsiahnutý v prehliadačoch a používateľ si ho musí sám doinštalovať. Na druhú stranu vo Flashi sa dajú vytvárať veľmi pekné animácie, preto je obzvlášť určený pre grafických dizajnérov.

DHTML/JavaScript (JavaScript + XML)

Medzi výhody JavaScriptu patrí jeho primárne určenie pre webové aplikácie na strane klienta (prehliadača), a jeho automatická podpora v novodobých prehliadačoch. Je jednoduchý, ľahko implementovateľný do webových stránok, podporuje DOM model a s obľubou ho využívajú hlavne DHTML web vývojári. Je pre nich najprirodzenejším a najjednoduchším prostriedkom na začlenenie dynamických prvkov do webových stránok.

2.6. XMLHttpRequest

V tejto časti si ukážeme komunikáciu so serverom pomocou XMLHttpRequestu s použitím JavaScriptu. [10] [11]

Vytvorenie XMLHttpRequestu

V prvom rade je potrebne vytvoriť XMLHttpRequest objekt. V Internet Exploreri voláme metódu XMLHTTP ActiveXobjektu, v ostatných prehliadačoch (Mozilla, Safari...) je priamo implementovaná trieda XMLHttpRequest. Vytvorenie XMLHttpRequest objektu bude teda nasledovné:

```
if (window.XMLHttpRequest)
```

```
  { // Mozilla, Safari, ...
    http_request = new XMLHttpRequest();
  }
else if (window.ActiveXObject)
  { // IE
    http_request = new ActiveXObject("Microsoft.XMLHTTP");
  }
```

Niektoré verzie prehliadačov nepracujú korektne, ak sa ako odpoveď zo servera nevrátia dáta s hlavičkou mime-type: text/xml. Preto zavoláme extra metódu, ktorá prepíše hlavičku prijatých dát zo servera v prípade, ak by nebola typu text/xml.

```
http_request.overrideMimeType('text/xml');
```

V ďalšom kroku určíme parametre volaného servera:

```
http_request.open('GET', 'http://www.server.com/pokus.php', true);
```

Prvý parameter určuje metódu http požiadavky. V prípade že požadujeme od servera dáta, použijeme metódu GET, v prípade, že serveru len posielame informáciu, použijeme metódu POST. Prípadne môžeme použiť ďalšie metódy podľa štandardu W3C, pokiaľ je podporovaná prehliadačom (HEAD, PUT, DELETE, TRACE, CONNECT). Nesmieme zabudnúť písať názvy metód veľkými písmenami.

Druhý parameter je URL stránky, ktorú voláme. Z bezpečnostného hľadiska nie je povolené posielať dotaz na stránky z domén tretích strán.

Tretí parameter určuje, či má byť požiadavka asynchrónna. Ak je TRUE, vykonávaný JavaScript bude pokračovať ďalej, zatiaľ čo sa čaká na odpoveď servera.

Samozrejme, musíme nejakým spôsobom odchytiť výsledok volanej požiadavky. Docielime to priradením funkcie zaisťujúcej obsluhu pri zmene stavu požiadavky k parametru `onreadystatechange`. Môžeme použiť priamo definovanú funkciu,

```
http_request.onreadystatechange = spracujFunction;
```

alebo môžeme použiť techniku JavaScriptu definovania funkcií priebežne

```
http_request.onreadystatechange = function(){
    // spracovanie výsledku
};
```

Ďalej môžeme definovať hlavičky http požiadavky, napr.:

```
http_request.setRequestHeader('Content-Type', 'application/x-www-form-
urlencoded');
```

A nakoniec zavoláme metódu `send`, v ktorej môžeme definovať telo správy. Najčastejšie sa táto metóda volá s parametrom null, alebo s formulárovým `query` reťazcom.

```
http_request.send('name=jano&sort=asc&id=15');
http_request.send(null);
```

Spracovanie odpovede servera

Pri vytvorení požiadavky sme definovali funkciu `spracujFunction` ktorá musí odchytiť a spracovať všetky zmeny stavu http požiadavky.

V prvom rade musíme zistiť aktuálny stav Http požiadavky, ktorý sa môže nachádzať v nasledovných stavoch:

- 0 neinicializovaný (uninitialized)
- 1 načítavaný (loading)
- 2 načítaný (loaded)
- 3 spracovávaný (interactive)
- 4 kompletný (complete)

Ak stav = 4, znamená to, že odpoveď servera bola v poriadku obdržaná a môžeme ďalej pokračovať v spracovaní. Kód bude nasledovný:

```
if (http_request.readyState == 4) {
    // vsetko v poriadku, odpoved servera dorucena
} else {
    // caka sa na dorucenie odpovede
}
```

Ďalej musíme skontrolovať kód statusu odpovede serveru. Pre nás je dôležité aby tento kód statusu mal hodnotu 200 (OK). Samozrejme, odpoveď môže mať aj iný status, napr.: 404 (Not found), 500 (Internal server error) a pod.

```
if (http_request.status == 200) {
    // OK, mozeme pristupit k spracovaniu samotnych dat
} else {
    // problem, napr.:
    // 404 (Not Found)
    // 500 (Internal Server Error)
    alert("Chyba pri nacitavani: " + httpRequest.status + ":" +
httpRequest.statusText);
}
```

Praktický príklad

Z predchádzajúcich častí teraz vytvoríme jednoduchý príklad. JavaScriptový kód vytvorí požiadavku na HTML dokument `test.html`, obsahujúci text „TEST 123“ a obsah tohto súboru zobrazí v informačnom okne (alert).

```
<script type="text/javascript" language="javascript">

function makeRequest(url) {

    var http_request = false;

    if (window.ActiveXObject) {
        var xmlhttpRequest = new ActiveXObject("Microsoft.XMLHTTP");
    } else {
        var xmlhttpRequest = new XMLHttpRequest();
    }

    if (!http_request) {
        alert('Giving up :( Cannot create an XMLHTTP instance');
        return false;
    }
      http_request.onreadystatechange = function() {
      processRequest(http_request);
    };
    http_request.open('GET', url, true);
    http_request.send(null);

}

function processRequest (http_request) {

    if (http_request.readyState == 4) {
        if (http_request.status == 200) {
            alert(http_request.responseText);
        } else {
            alert('There was a problem with the request.');
        }
    }
}
</script>

<span
    style="cursor: pointer; text-decoration: underline"
    onclick="makeRequest('test.html')">
        Pokus o požiadavku
</span>
```

Popis:

- Používateľ klikne na odkaz „*Pokus o požiadavku*“
- To zavolá funkciu `makeRequest` s parametrom „test.html“. Funkcia vytvorí Http požiadavku na súbor s daným názvom v tom istom adresári.

- Pri zmene stavu (*onreadystatechange*) je zavolaná funkcia `processRequest`.
- `processRequest` skontroluje odpoveď servera a ak je všetko OK, zobrazí informačné okno (alert) s obsahom súboru „test.html“ („TEST 123“)

V prípade, že naša požiadavka nie je statický súbor, ale dynamicky generovaný (vo väčšine prípadov) je potrebné poslať hlavičku Cache-Control: no-cache, aby sme zamedzili ukladaniu súboru do vyrovnávacej pamäte (cache) prehliadača.

Takisto je dôležité, aby sme objekt `http_request` vytvárali ako lokálnu premennú v tele funkcie. V prípade že by bol vytvorený ako globálna premenná, každá nová požiadavka by prepísala tú existujúcu.

Spracovanie XML dokumentu

V predchádzajúcom príklade sme pracovali s vráteným obsahom ako s obyčajným textom (`http_request.responseText`). V prípade, že chceme pracovať s čistým xml, voláme metódu *responseXML* a vykonáme nasledovné úpravy kódu.

V prvom rade namiesto súboru text.html vytvoríme xml dokument test.xml s obsahom:

```
<?xml version="1.0" ?>
<root>
    I'm a test.
</root>
```

V kóde zmeníme volanie funkcie

```
...
onclick="makeRequest('test.xml')">
...
```

A vo funkcii `alertContents()` zmeníme spôsob volania informačného okna (Alert)

```
var xmldoc = http_request.responseXML;
var root_node = xmldoc.getElementsByTagName('root').item(0);
alert(root_node.firstChild.data);
```

Týmto spôsobom sme pomocou `responseXML` prevzali čistý XML dokument a použitím štandardných DOM metód aktívne pristupovali k obsahu. XML dokumentu.

2.7. Dynamické generovanie HTML dokumentu.

V predchádzajúcom príklade sme ukázali ako vytvoriť a obsluhovať http požiadavku. Vrátený výsledok sme však len zobrazili pomocou informačného okna (alert). Zatiaľ sme nemenili priamo obsah zobrazeného HTML dokumentu. Ukážeme to v nasledujúcom príklade.

Vytvoríme si stránku s výberovým poľom, v ktorom budú názvy html súborov. Po výbere súboru vo výberovom poli sa nám požadovaný obsah súboru zobrazí na konci zobrazeného dokumentu.

Vytvoríme `<select>` a pridáme parameter `onchange`, ktorý v prípade, že vyberieme nejakú položku z výberového poľa, zavolá našu obslužnú funkciu `makeRequest` z prvého príkladu. V dokumente si vytvoríme element `<div>` s `id="insertText"` do ktorého budeme načítavať zvolený súbor.

```
<select name="vyber" id="vyber" onchange=" makeRequest();">
   <option value="0">Vyberte súbor</option>
   <option value="subor1.htm">Subor jedna</option>
   <option value="subor2.htm">Subor dva</option>
</select>

<div id="insertText">
</div>
```

Vo funkcii `makeRequest` pridáme inicializáciu premennej url a upravíme volanie `httpRequest.open`

```
function makeRequest  ()
{
     var url = document.getElementById("vyber").value;

     if (url != 0)
     {
        if (window.ActiveXObject)
        { httpRequest = new ActiveXObject("Microsoft.XMLHTTP"); }
        else
        { httpRequest = new XMLHttpRequest(); }
        httpRequest.open("GET", url, true);
        httpRequest.onreadystatechange= function () {processRequest(); } ;
        httpRequest.send(null);
      }
      else
      {
        document.getElementById("insertText").innerHTML = "";

      }
}
```

Vo funkcii `processRequest` zmeníme časť, ktorá v prvom prípade zobrazovala informačne okno, tak aby obsah načítaného súboru vypísala do tela `<div>` elementu.

```
...
if(httpRequest.status == 200)
{
  var newText = document.getElementById("insertText");
  newText.innerHTML = httpRequest.responseText;
}
...
```

Príkazom `var newText = getElementById("insertText ");` sme premennej `newText` priradili objekt `insertText` (element) zo zobrazenej stránky a príkazom newText.innerHTML = httpRequest.responseText sme obsah tohto elementu prepísali obsahom načítanej webovej stránky. Využili sme tak vlastnosti DOM (Objektový model dokumentu).

Toľko k jednoduchým príkladom práce s XMLHttpRequestom. V príkladoch nebola vysvetľovaná syntax Html, JavaScriptu, a taktiež sa predpokladá znalosť DOM a práce s ním. Čitateľovi, ktorý by sa chcel viac dozvedieť o DOM a práci s ním, odporúčam oficiálnu dokumentáciu [7], prípadne seriál článkov na Žive [8].

2.8. Kedy použiť Ajax

V tejto kapitole načrtnem niektoré oblasti, v ktorých je použitie technológie Ajax pozitívnym prínosom.

Interaktívne formuláre

Editovanie informácii v prostredí webu býva niekedy veľmi nepraktické. Používateľ na začiatku určí, ktorú informáciu chce editovať, potom čaká na načítanie editovacieho formulára v novom okne, vyplní vstupné polia, stlačí tlačidlo *Potvrdiť* a čaká na načítanie pôvodnej webovej stránky. Následne skontroluje, či editované údaje zadal správne. V interaktívnej stránke si používateľ zvolí časť, ktorú chce editovať, automaticky sa mu otvorí editovací formulár v pôvodnej stránke a po potvrdení sa vykonané zmeny asynchrónne pošlú na spracovanie a výsledok sa zobrazí v pôvodnom okne.

Interaktívne výbery

Ideálne je použiť Ajax pri navzájom previazaných výberových poliach. Ako príklad uvediem výber nehnuteľnosti. Nehnuteľnosť môžeme vyberať podľa troch

kritérií: podľa lokality (Dúbravka, Rača, Petržalka...), podľa veľkosti (1,2,3 izby) a podľa typu (byt, rodinný dom). Je veľmi zdržujúce, keď používateľ zadá určitú kombináciu parametrov a po vyhľadaní zistí, že množina výsledkov je prázdna. Použitím Ajaxu pri interaktívnom zadaní niektorého z parametrov sa v ostaných výberových poliach zobrazia len tie hodnoty, pre ktoré existuje nejaká kombinácia.

Autosave

S použitím Ajaxu je možné vykonávať priebežné automatické ukladanie nejakého editovacieho prvku. Pre používateľa je to veľmi vítané a vyhne sa nepríjemnej situácii, keď napríklad pri písaní emailu vo webovom formulári príde o celý napísaný text pri prerušení spojenia.

Zobrazenie hierarchicky usporiadaných informácií

Jednoduchý prístup k dátam v stromovej štruktúre na začiatku zobrazí koreňové informácie stromu. Pri kliknutí na niektorú vetvu sa načíta ďalšia úroveň informácii prináležiaca danej vetve. Ak sa chce používateľ vrátiť o úroveň vyššie, opätovne sa musí načítať celá nadradená štruktúra stromu. Určitým vylepšením bolo na začiatku nahrať celý strom do pamäti a interaktívne ho, napríklad pomocou skriptov na strane klienta, postupne „rozbaľovať“. Nevýhoda takéhoto riešenia sa prejavila pri dátovo obsiahlych stromoch, kedy bolo nutné na začiatku čakať na načítanie celého stromu, aj keď niektoré informácie používateľ nepotreboval.

S použitím Ajaxu je možné na začiatku načítať len koreňovú časť stromu a podľa požiadaviek používateľa postupne načítavať jednotlivé časti podstromu. Pri kroku o úroveň vyššie už nie je nutné opätovne načítavať celú nadradenú vetvu v strome.

On-line komunikácia (Chat)

Pri on-line komunikácii medzi dvoma alebo viacerými používateľmi sa pri každom novom príspevku používateľa mení (pribúda) len malá časť zobrazenej informácie. Vytvorenie takejto on-line komunikácie založenej na statickom webe, kedy by sa pri každom novom príspevku musela načítať celá predošlá komunikácia, vyžaduje omnoho väčšiu prenosovú réžiu, než použitie dynamického načítania len novo pridaného obsahu.

Hlasovanie, hodnotenie

Hlasovanie alebo hodnotenie je z môjho pohľadu činnosť, ktorá by mala byť na webovej stránke jednoznačne riešená pomocou Ajaxu. Používateľ, ktorý je na webovej stránke nabádaný k hlasovaniu alebo hodnoteniu, o to viac, ak dobrovoľne, vyžaduje od tejto činnosti, aby ho čo najmenej obmedzila. Ak vie, že po kliknutí na hlasovací formulár sa mu bude stránka opätovne celá načítavať, radšej vôbec nezahlasuje. Pri použití Ajaxu sa po kliknutí požiadavka asynchrónne spracuje na pozadí a používateľa nijako neovplyvňuje. Keď je hlasovanie vyhodnotené, na mieste hlasovacieho formulára sa zobrazia napr. výsledky hlasovania. Všetko bez opätovného načítania stránky.

Zoraďovanie, filtrovanie

Zoraďovanie podľa mena, zoraďovanie podľa mena a dátumu, zapínanie a vypínanie filtrovanie. Všetko sú to činnosti, ktoré z pohľadu používateľa len interaktívne manipulujú s už načítanými dátami. Ak by sa každá takáto požiadavka mala riešiť opätovným načítaním dát zo servera, enormne by zvyšovala časovú náročnosť práce. Dynamické spracovanie takýchto požiadaviek na strane klienta tak zrýchľuje a zlepšuje komfort práce.

Predvyplňovanie textu

Pri vypisovaní tej istej textovej frázy alebo vkladaní predpokladaného textu môže byť Ajax užitočnou pomôckou, ktorá zlepší komfort pre používateľa. Príkladom môže byť už v úvodnej kapitole spomínaný Google našeptávač (Google Suggest), vyplňovanie email adresy na základe prvých písmen, alebo napríklad vyplnenie PSČ na základe uvedeného názvu obce, alebo naopak.

Interaktívne kontroly chýb

Ak používateľ vypĺňa komplikovaný formulár, výrazne napomáha, ak výsledky niektorých kontrol správnosti dát sú zistené a oznámené používateľovi ešte pred odoslaním formulára. Ide v prvom rade o kontroly, ktoré nepotrebujú komunikáciu zo serverom, ako napríklad kontrola správne zapísanej email adresy. V druhom rade ale môžeme Ajax použiť na overenie unikátnosti prihlasovacieho mena pri zakladaní nového používateľského účtu.

Zložité výpočty

JavaScript nie je určený na zložité matematické výpočty. V niektorých prípadoch môže byť vhodnejšie zložitý výpočet poslať pomocou Ajaxu na výkonný výpočtový server a na strane klienta už len zobraziť výsledok.

2.9. Obmedzenia Ajaxu a omyly pri jeho použití

Takisto, ako každá iná technológia, nech je akokoľvek dobrá a prínosná, dá sa použiť spôsobom, kedy jej použitie pôsobí skôr negatívnym dojmom. Nižšie popíšem zopár chybných spôsobov použitia Ajaxu, ktorým by sme sa mali vyhnúť.

Použitie Ajaxu kvôli Ajaxu

Typický neduh novej technológie, ktorá je inovatívna a prináša nové možnosti je jej použitie preto, aby bola použitá, nehľadiac na to, či je vôbec potrebné ju použiť. Špeciálne vývojári sa veľmi radi hrajú s novými technológiami, ale je potrebne si uvedomiť, že Ajax nie je hračka, ale nástroj, a preto ho treba používať s mierou a tam kde je naozaj potrebný a prinesie zlepšenie.

Znefunkčnenie tlačidla Spať (Back)

Tlačidlo Späť patrí k štandardným ovládacím prvkom internetových prehliadačov, avšak použitím Ajaxu sa môže jeho funkcia stať trochu zavádzajúca a problematická. Používateľ, ktorý dynamicky menil obsah stránky nemá možnosť elegantným spôsobom sa vrátiť o krok späť, ale musí začať pekne od začiatku. Je to spôsobené tým, že technológia JavaScriptu, najčastejšie použitá v Ajax aplikáciách, nevie dynamicky meniť funkcionalitu tlačidla Späť a udržať ho tak funkčné vzhľadom na zmeny vykonané na stránke.

Riešenie tohto problému je na vývojárovi aplikácie, ktorý musí sám zabezpečiť adekvátnym spôsobom možnosť vrátiť sa po jednotlivých krokoch späť, napríklad vlastným odkazom Späť v hlavičke stránky.

Signalizácia asynchrónnych operácií

Keďže Ajax je založený na asynchrónnom prístupe k informáciám, a teda nefunguje na klasickom princípe „kliknem a počkám na výsledok“, ale môžem klikať viackrát a výsledok sa mi zobrazuje priebežne, je potrebné dať užívateľovi nejakým spôsobom vedieť, že čakám na výsledok spracovania nejakej akcie. Opäť použijem ako

príklad Gmail, kde červený štvorec vpravo hore indikuje, že stránka čaká na výsledok nejakej asynchrónnej operácie. Žiaľ, je to daň za asynchrónnosť.

Problémy pri „off-line“ webových stránkach

Napriek tomu, že internetové pripojenie má čoraz viac ľudí, ešte stále je tu veľa používateľov, ktorí ho nemajú a pracujú off-line, prípadne niekedy, napr. pri archivácii, sa s dátami pracuje len v režime off-line. Pri dynamickom webe, kedy aplikácia komunikuje so serverom, treba riešiť off-line prístup iným spôsobom, t.j. zvláštnou verziou webovej stránky, ktorá je uspôsobená na off-line prezeranie.

Záložky (Taby)

Nová generácia internetových prehliadačov priniesla zaujímavé zrýchlenie načítavania webových stránok – záložky, keď si užívateľ pootvára naraz viacero stránok do jednotlivých záložiek, ktoré sa mu postupne na pozadí načítavajú. Pri použití dynamicky generovanej stránky niekedy nie je možne jednotlivé časti otvárať v samostatných oknách (záložkách), pokiaľ sú navzájom dynamicky previazané. Problém nastane pri pomalšom pripojení, keď používateľ musí vždy čakať na načítanie ďalšieho kroku.

Vzájomná kompatibilita

Ajax aplikácie sú vytvárané ako multiplatformové, ktoré musia bežať vo všetkých moderných webových prehliadačoch. Netreba zabudnúť otestovať vyvíjanú aplikáciu reálne vo viacerých prehliadačoch, vzhľadom na to, že aj keď má JavaScript nálepku „multiplatformový“, jeho implementácia v jednotlivých prehliadačoch je v niektorých maličkostiach odlišná, zvlášť u Microsoft IE.

Viacpoužívateľský prístup

Hlavne pri vyvíjaní intranetových aplikácií je dôležité mať na pamäti, že viacerí používatelia môžu v tom istom čase pracovať s rovnakou informáciou. Pokiaľ je informácia uložená v databáze, je potrebné zaistiť, že používateľ pracuje vždy s aktuálnou informáciou.

Príliš veľa kódu prehliadač spomaľuje

Technológia Ajax umožňuje vytvoriť web dynamickejší a pre používateľa zaujímavejší, čo však na druhú stranu znamená viac kódu na webovej stránke, a to

v konečnom dôsledku znamená spomalenie prehliadača. V minulosti bolo použitie napr. JavaScriptu v prehliadači veľkou záťažou pre procesor. V dnešnej dobe tomu už tak nie je, spracovaného kódu môže byť na stránke viac, bez toho, aby výrazne zaťažil procesor, ale v každom prípade treba vytvárať efektívny kód a myslieť na to, že pre používateľa, ktorý ma menej výkonný procesor a má naraz spustených viacero Ajaxových aplikácií, môže výpočtová náročnosť znemožniť serióznu prácu.

Dostupnosť JavaScriptu

Mnoho používateľov nemá JavaScript, alebo ho napríklad z dôvodu bezpečnosti má vypnutý. V takomto prípade, samozrejme Ajax aplikácia založená na JavaScripte nebude fungovať. Riešením je po detekcii prehliadača, ktorý nepodporuje JavaScript, poskytnúť inú verziu webovej stránky bez JavaScriptu.

Asynchrónne spracovanie

Vzhľadom na skutočnosť, že Ajax aplikácie sú asynchrónne, treba myslieť na to, že asynchrónne spracovávané požiadavky nemusia byť spracovávane v poradí, v akom boli počiatočne volané. Ak napríklad používateľ označí veľa zaškrtávacích políčok, z ktorých každé vyvolá asynchrónne spracovávanú požiadavku, môže sa stať, že stratí prehľad o závislostiach zmien na stránke. Ak navyše niektoré asynchrónne požiadavky vyvolajú potvrdzovacie okná (alerts) môže sa stať, že používateľ potvrdí inú vec, ako sa domýšľal.

Problém s externým odkazovaním

Ak používateľ už aktívne pracoval s obsahom zobrazenej stránky, má veľký problém poslať externú linku momentálne zobrazeného obsahu druhej osobe, prípadne uložiť odkaz do záložiek. Riešenie je opäť na strane vývojára, ktorý musí zobraziť externý odkaz na aktuálny obsah formou externého odkazu napr. v hlavičke stránky.

„Narastanie“ obsahu počas skrolovania

Niektoré stránky dynamicky generujú celý svoj obsah pomocou viacerých nezávislých častí. V prípade webovej stránky ktorá je dlhšia ako zobrazované okno môže pôsobiť veľmi rušivo, ak sa zrazu doplní obsah webovej stránky nad miestom, ktorý používateľ momentálne sleduje a sledovaná informácia v okne sa mu v okne posunie.

Vyhľadávacie roboty

Webové stránky, ktoré väčšinu svojho obsahu načítavajú dynamicky, predstavujú problém pre vyhľadávacie a indexovacie roboty (Google, Yahoo, Zoznam). Vyhľadávací robot sa nevie dopracovať k zložitejšie dynamicky načítanej informácii, a preto ju nemôže zaradiť do svojej databázy. Alebo opačný prípad, webová stránka obsahuje informáciu, ktorú robot aj vyhľadal, ale ktorá sa nezobrazuje hneď po načítaní stránky, ale až po akcii používateľa a používateľ, môže takúto prvotne načítanú stránku opustiť, pretože požadovanú informáciu nenašiel.

Metódy GET a POST

Väčšina Ajax aplikácii používa metódu GET pri komunikácii so serverom. W3C štandard však definuje použitie metódy GET len v prípade, keď od servera očakávame odpoveď. V opačnom prípade, keď serveru len posielame dáta, treba použiť metódu POST. Aj keď sa to vzhľadom na používateľa bude javiť rovnako, či sa použije GET alebo POST, je dobré dodržiavať štandard.

Zmeny a závislosti

V Ajax aplikáciách spravidla dynamicky meníme len časť obsahu stránky. Treba sa však na danú zmenu pozrieť aj s globálneho hľadiska a zmeniť tie časti, ktoré so zmeneným obsahom súvisia. Jednoduchý príklad: majme stránku o Jožkovi. Je dobré aby sa pri zmene mena v obsahu stránky z Jožka na Zuzku zmenil aj titulok okna z „Stránka o Jožkovi“ na titulok „Stránka o Zuzke“.

Problém pri hľadaní chyby

V klasických, serverovo orientovaných aplikáciách, kde klient zobrazuje výsledok nie je problém zobraziť zdrojový kód webovej stránky a nájsť prípadnú chybu. V klient-server aplikáciách je to omnoho ťažšie, lebo musíme naviac analyzovať dynamicky načítaný obsah.

Zložitosť

Webové stránky s použitím technológie Ajax sú vo väčšine prípadov zložitejšie na navrhovanie a implementáciu oproti klasickému statickému webu. Niekedy treba zvážiť efektivitu takto naviac vynaloženej námahy.

2.10. Bezpečnosť v Ajaxe

S použitím Ajaxu sú spojené aj určité bezpečnostné riziká. V svojej podstate nejde o nič nového, čo by otváralo pomyselné dvere počítača, tie sú tak či tak otvorené. Ajax, tým že využíva už zabehnuté technológie, neprináša žiadne nové riziká, ktoré by už neexistovali predtým. Umožňuje však zlomyseľnému programátorovi, aby škodil „elegantnejšie“. Je veľmi jednoduché napísať napríklad skript, ktorý bude sledovať znaky písané na klávesnici a priebežne ich odosielať na server k analýze. Koľkokrát sa vám stalo, že ste pri prihlasovaní do aplikácie začali písať heslo, ktoré používate niekde inde...

2.11. Podpora Ajax technológií v novodobých prehliadačoch.

Záverom by som zhodnotil, ako vyzerá v súčasnosti podpora Ajaxu v internetových prehliadačoch. Nasledujúci zoznam obsahuje internetové prehliadače podporujúce technológiu Ajax

- Apple Safari 1.2 a vyšší
- Konqueror
- Microsoft Internet Explorer 4.0 a vyšší
- Mozilla/Mozilla Firefox 1.0 a vyšší
- Netscape 7.1 a vyšší
- Opera 7.6 a vyšší

Tento zoznam vyzerá dostatočne optimisticky a pre tvorcov Ajax aplikácií predstavuje dostatočne širokú základňu umožňujúcu im vyvíjať bez väčších problémov s kompatibilitou.

3. Problematika tvorby rozvrhu

Jednou z dôležitých súčastí každej školskej organizácie je jej rozvrh, na ktorého kvalitu sú kladené vysoké požiadavky, nakoľko je jedným z faktorov, ktoré vplývajú na efektivitu štúdia. Štúdium, ako také je činnosť zameraná predovšetkým na duševnú prácu, preto by správna organizácia časových zdrojov mala zabezpečovať optimálne podmienky pre štúdium, ako študentov, tak aj pedagogických pracovníkov. A v neposlednom rade, premyslená organizácia štúdia šetrí čas, čo v dnešnej dobe znamená, že šetrí aj peniaze.

Pri tvorbe rozvrhových systémov, sa stretávame s viacerými problémami. V nasledujúcej kapitole zanalyzujeme všeobecnú problematiku tvorby základného modelu rozvrhu, s dôrazom na problematiku tvorby jednoduchého rozvrhu. V ďalšej kapitole poukážeme na odlišnosť fakultného rozvrhu od základného modelu.

3.1. Všeobecná problematika tvorby rozvrhu

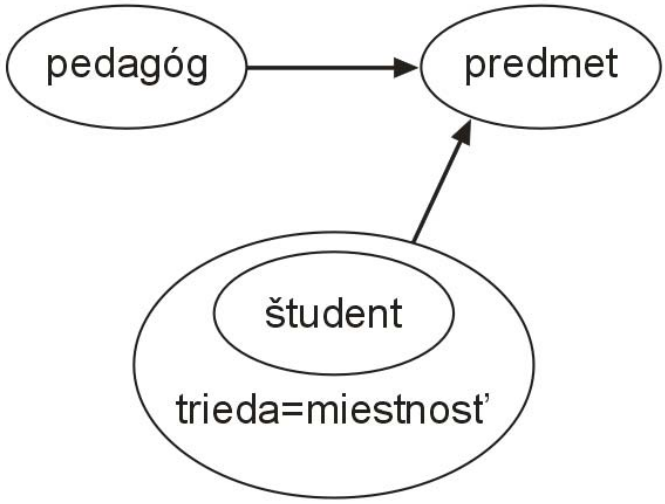
Pri základnom modeli rozvrhu (pre jednoduchšie pochopenie si predstavme stredoškolský rozvrh) pracujeme z nasledujúcimi dátovými entitami:

- miestnosti
- študenti, ktorí sú združení do vyššieho celku – skupiny, triedy
- pedagógovia
- predmety

V základnom modeli platia nasledujúce pravidlá:

- Študenti, resp. skupiny študentov sú zviazané s miestnosťami (triedami), kde prebieha celý vyučovací proces. Existuje malý počet výnimiek, ako výučba jazyka, telesnej výchovy, informatiky, ktoré sú ale viazané takisto na jednu, špecializovanú miestnosť.
- Pedagógovia sú zviazaní s daným predmetom.
- Vyučovací proces prebieha v týždňovej periodicite.
- Predpokladá sa ucelený proces výučby bez voľných hodín, s konštantným začiatkom prvej vyučovacej hodiny.

Zjednodušene povedané, pri základnom modeli rozvrhu je ku kombinácii študent – predmet priradený atribút pedagóg a čas.



Obr. 5. Obrázok jednoduchého modelu rozvrhu.

3.2. Odlišnosť problematiky tvorby rozvrhu na FMFI

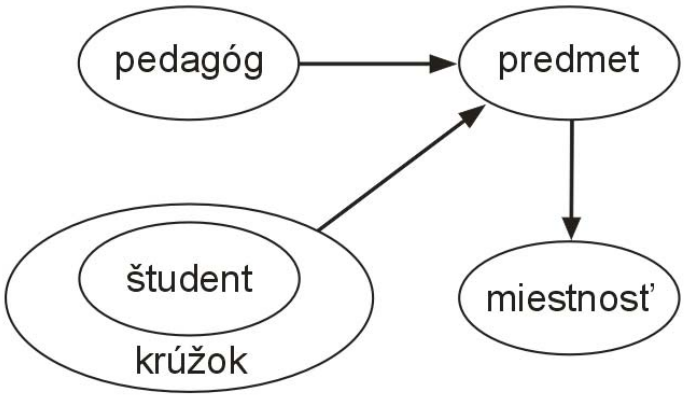
Pri rozšírenom fakultnom modeli rozvrhu pracujeme s nasledujúcimi dátovými entitami:

- miestnosti
- študenti, ktorí sú združení do vyššieho celku – skupiny, krúžku
- pedagógovia
- predmety

Od základného modelu rozvrhu sa navyše odlišuje v týchto aspektoch:

- dlhý vyučovací čas (8:00-20:00), navyše tento vyučovací čas nie je nutne spojitý a takisto vyučovanie nezačína vždy v tom istom čase
- dlhé časové rezervy medzi vyučovacími hodinami
- jednotliví študenti patria do jednotlivých študijných skupín (krúžkov). Vzhľadom na kreditový spôsob výučby, kedy si študent môže zapisovať ľubovoľné predmety sa stáva, že študent patrí prakticky do viacerých študijných skupín súčasne, resp. využíva časť rozvrhu rôznych študijných skupín.
- každá vyučovacia hodina môže byť v inej miestnosti (príp. budove)

- nie každá miestnosť vyhovuje každému predmetu (kapacita, vybavenie)
- skupín, predmetov, vyučujúcich, miestností je mnohonásobne viac, ako pri vyučovacom procese základných a stredných škôl



Obr. 6. Obrázok rozšíreného fakultného modelu rozvrhu

Na prvý pohľad vidíme, že fakultný model rozvrhu je omnoho zložitejší, obsahujúci mnoho výnimiek. Ďalej si treba uvedomiť, že nájsť riešenie hrubým výpočtom, by bolo z časového hľadiska veľmi náročné.

Nie je možné strojovo v rozumnom čase skontrolovať všetky riešenia. Samozrejme, existujú algoritmy, ktoré vedia nájsť riešenie rozvrhu, avšak fakultný rozvrh je natoľko špecifický a obsahuje veľmi veľa výnimiek a podmienok, že z praktického hľadiska je strojový návrh nepoužiteľný.

V minulosti boli pokusy generovať fakultný rozvrh pomocou softvéru, avšak ponúkané riešenia nevyhovovali hlavne preto, že nevedeli efektívne obsiahnuť všetky výnimky a dodatočné požiadavky na rozvrh. Výsledný rozvrh obsahoval veľa časových medzier, či už na strane pedagógov, alebo študijných skupín a aj tak v ňom neboli zahrnuté všetky špecifické podmienky. Z pohľadu efektivity práce a vynaložených prostriedkov bolo jednoduchšie vytvárať rozvrh ručne a preto sa na FMFI dosiaľ používa pri tvorbe rozvrhu spôsob bez použitia strojového návrhu.

3.3. Prínos navrhovaného systému tvorby rozvrhov

Cieľom navrhovaného systému tvorby rozvrhov je napomôcť pedagogickým pracovníkom pri tvorbe fakultného rozvrhu.

Chcem zdôrazniť, že vytvorený systém by nemal za úlohu sám navrhnúť rozvrh, nakoľko tvorba rozvrhov je na jednej strane zložitý algoritmický problém s veľkou časovou náročnosťou, na druhej strane nie je možné doň efektívne zakomponovať všetky požiadavky a ako som už vyššie uviedol, doterajšie pokusy neponúkli uspokojivé výsledky.

Navrhnutý systém, by mal spĺňať nasledovné požiadavky:

- Odstrániť problém „rozvrhárov“ „byť v jednom čase na jednom mieste“ použitím centrálnej databázy a prístupu k nej z používateľských pracovných staníc.
- Zabezpečiť integritu dát. Konkrétne problémy:
  - Vyučujúci učí súčasne na dvoch miestach v rovnakom čase
  - Študent má v rovnakom čase dva predmety
  - V miestnosti sa v rovnakom čase vyučujú dva predmety
- Zabezpečiť jednorazové zadávanie dát do systému, prípadne vytvoriť rozhranie na export/import do informačného systému fakulty.
- Umožniť detailný výpis informácií podľa zadaných kritérií ako napr: výpis pre konkrétnu miestnosť, vyučujúceho, študenta...
- Vypisovať rozvrhy študentov pre ročníky / krúžky / jednotlivcov

4. Analýza a špecifikácia

Súčasný stav rozvrhového systému na fakulte

Pri zisťovaní súčasného stavu tvorby rozvrhov som dospel k zisteniu, že aj v dnešnej, nazvime ju „modernej dobe počítačov“ sa tvorba rozvrhu na FMFI robí ručne.

Pri skúmaní dôvodov, prečo je tomu tak, som zistil, že v súčasnosti dostupný software na tvorbu rozvrhov neposkytuje dostatočne uspokojivé výsledky. Poskytuje príliš, technický alebo človeku neprirodzený výsledok. Naopak, rozvrh vytvorený človekom je lepšie použiteľný v reálnom živote. A tak prebieha tvorba rozvrhu na fakulte ručne.

4.1. Popis procesu tvorby rozvrhu na fakulte

Pre každú katedru je poverená jedna, prípadne dve osoby, ktoré sú zodpovedné za tvorbu rozvrhu na príslušnej katedre.

Samotný proces tvorby rozvrhu na fakulte prebieha v dvoch krokoch.

1. krok – fáza predprípravy

V tejto fáze predprípravy osoba poverená tvorbou rozvrhu si zosumarizuje nasledovné informácie:

- predmety, ktoré sa v nasledujúcom období (semestri) budú vyučovať na katedre
- počet študentov, navštevujúcich tieto predmety
- počet vyučujúcich

Z týchto údajov následne vypracuje harmonogram pridelenia jednotlivých skupín (predmetov) vyučujúcim, pričom treba brať ohľad na:

- kvalifikáciu vyučujúceho
- dĺžku týždenného pracovného úväzku

Ako poslednú časť prvej fázy sa jednotlivé predmety rozdelia do skupín:

- špeciálne predmety, ktoré majú vysoké prioritu pri určení miesta/času v rozvrhu

- veľké bloky (hlavné predmety povinného štúdia)
- malé bloky (výberové predmety)
- cvičenia

2. krok – fáza tvorby rozvrhu

V tejto fáze sa určia dni a čas , kedy sa bude vytvárať rozvrh pre celú fakultu. V tomto čase sa zhromaždia všetci zástupcovia katedier poverení tvorbou rozvrhu (rozvrhári) v jednej miestnosti kde sú veľké tabule na jednotlivé dni v týždni, rozdelené na miestnosti fakulty a čas vyučovacích hodín. Rozvrhárom sú k dispozícii aj (na začiatku prázdne) rozvrhy jednotlivých študijných skupín. Potom sa začnú postupne prideľovať jednotlivé predmety do časového harmonogramu podľa poradia: špeciálne predmety, veľké bloky, malé bloky, cvičenia. Rozvrhári pristupujú k rozvrhovým tabuliam a vkladajú do nich jednotlivé papieriky predstavujúce konkrétne predmety. Nasadený predmet súčasne zaznačia do rozvrhu príslušného krúžku/krúžkov. Medzitým si musia priebežne kontrolovať vzájomnú konzistenciu vložených údajov, aby sa nestalo, že napr. jeden pedagóg má nasadenú prednášku v rovnakom čase na viacerých miestach.

Každý vložený predmet treba ešte ručne zadať do elektronického systému, ktorý tlačí rozvrhy.

3. krok – fáza dolaďovania v praxi

Posledný, neformálny krok je dolaďovanie počas priebehu semestra, kde sa robia posledné úpravy a výmeny.

Tlač rozvrhov.

4.2. Analýza problému

Z predchádzajúcej kapitoly, ktorá popisuje tvorbu fakultného rozvrhu v môžeme vybadať niekoľko nevýhod súčasného stavu:

- Požaduje sa, aby boli všetky osoby zaoberajúce sa tvorbou rozvrhu „v tom istom čase na tom istom mieste“.
- Neumožňuje priebežnú automatickú „kontrolu integrity“ zadaných dát.
- Informácie treba zadávať ručne, aj napriek tomu, že by sa ich veľa dalo zautomatizovať.

- Neexistuje previazanosť s informačným systémom fakulty.
- V konečnom dôsledku treba ručne navrhnutý rozvrh prepísať do elektronickej podoby.
- Rozvrhy uverejnené na fakultnej stránke sú v tvare, ktorý je len veľmi ťažko dostupný našim nevidiacim študentom (prečítať si ho zvládnu len tí najzručnejší, čo mnohých ešte viac hendikepuje.

4.3. Špecifikácia

V nasledujúcej kapitole podrobne špecifikujeme, čo by mal fakultný systém na tvorbu rozvrhu obsahovať. Pre zjednodušenie budeme v nasledujúcich riadkoch používať pre pojem Fakultný rozvrhový systém skratku FRS.

FRS bude pracovať nad piatimi základnými skupinami objektov

- Pedagógovia
- Predmety
- Miestnosti
- Študenti združení do skupín, ďalej len Študijné skupiny
- Osoby navrhujúce rozvrh, ďalej len Rozvrhári

Pedagógovia

FRS bude umožňovať pridávať, editovať a mazať jednotlivých pedagógov fakulty. Informácia o každom jednotlivému pedagógovi bude obsahovať:

- Meno pedagóga
- Kontaktné údaje (telefón, email)
- Miestnosť pedagóga (kanceláriu)
- Voliteľné poznámky

V každom osobnom profile pedagóga budú viditeľné predmety, ktoré vyučuje, aj s časovým plánom výučby, ak mu už bol priradený.

Predmety

FRS bude umožňovať pridávať, editovať a mazať jednotlivé predmety. Informácia o každom predmete bude obsahovať:

- Názov predmetu

- Kód predmetu (napr.: 2-INF-101)
- Typ predmetu (malý/veľký, cvičenie/prednáška) podľa interných zvyklostí navrhovateľov fakultného rozvrhu
- Pedagóg, ktorý daný predmet vyučuje
- Miestnosť v ktorej bude výučba daného predmetu prebiehať
- Voliteľné poznámky (nutnosť projektora, potrebná veľká miestnosť)
- Rozvrhár, ktorý má daný predmet rozvrhnúť.

Každému predmetu bude navyše možné priradiť Študijné skupiny, u ktorých sa predpokladá, že študenti, patriaci do tejto študijnej skupiny, si budú zapisovať tento predmet. Pre konkrétnu študijnú skupinu bude navyše možnosť nastavenia, či sa má kontrolovať jej časové prekrývanie s inými predmetmi v rámci tej istej študijnej skupiny („príznak neprekrývania“)

Miestnosti

FRS bude umožňovať pridávať, editovať a mazať jednotlivé miestnosti. Informácia o každej miestnosti bude obsahovať:

- Názov miestnosti (číslo dverí)
- Blok (budovu) kde sa daná miestnosť nachádza
- Kapacitu (počet miest na sedenie)
- Voliteľné poznámky (špeciálne vybavenie)

Študijné skupiny

Pre študijné skupiny bude FRS umožňovať definovať pre každý predmet, či sa má u danej študijnej skupiny kontrolovať jej časové prekrývanie v rámci tej istej skupiny („príznak neprekrývania“). Toto definovanie môže byť zapracované priamo v činnosti definovania predmetov, a nie je potrebné mať zvlášť činnosť na definovanie Študijných skupín.

Rozvrhári

FRS bude umožňovať pridávať, editovať a mazať jednotlivých rozvrhárov. Informácia o každom rozvrhárovi bude obsahovať:

- Meno rozvrhára
- Kontaktné údaje (telefón, email)

- Predmety určené danému rozvrhárovi, s informáciou, či už je daný predmet priradený do rozvrhu

Činnosť: Prideľovanie rozvrhu

FRS bude v hlavnej činnosti určenej na prideľovanie rozvrhu zobrazovať vždy pre konkrétny predmet tri okná z časovým rozvrhom:

- Okno z časovým rozvrhom pre pedagóga
- Okno s časovým rozvrhom pre študijnú skupinu
- Okno s časovým rozvrhom pre miestnosť

Okno časového rozvrhu pre pedagóga a študijnú skupinu bude zviazané s aktuálne prideľovaným predmetom a bude needitovateľné.

Pre okno časového rozvrhu pre danú miestnosť bude možné v rámci prideľovaného predmetu dynamicky meniť jednotlivé miestnosti.

Vo všetkých oknách časového rozvrhu bude graficky zvýraznená obsadenosť jednotlivej časovej jednotky (hodiny) iným objektom.

Kontrola integrity dát

Kontrola integrity dát bude prebiehať na týchto úrovniach:

- Kontrola, či pedagóg v danom čase nevyučuje iný predmet
- Kontrola, či skupina v danom čase nemá nasadený iný predmet, pokiaľ má táto skupina pre daný predmet nastavený „príznak neprekrývania“
- Kontrola, či v danej miestnosti neprebieha v danom čase výučba iného predmetu.

Všetky tieto kontroly integrity dát budú mať len informatívny charakter, t.j. pri porušení integrity na to upozornia zadávateľa (rozvrhára) ale povolia nasadiť predmet, aj napriek porušeniu integrity.

Výstupné zostavy (reporty)

FRS bude umožňovať vytvárať tieto výsledné zostavy:

- Zostava pedagógov s predmetmi, ktoré vyučujú.
- Sumárna zostava pedagógov s počtom vyučovacích hodín (úväzky).
- Zostava nezaradených predmetov

- Zostava predmetov nepriradených rozvrhárom
- Zostava rozvrhárov s predmetmi, ktoré má nasadzovať.

Číselníky

FRS bude obsahovať číselníky, ktoré budú môcť byť editovateľné, no nemusí tak byť nutne v prostredí aplikácie. Bude postačovať ak ich editácia bude prístupná administrátorovi systémovými nástrojmi.

FRS bude nutne obsahovať nasledovné číselníky:

- Číselník: Typ predmetu (malý/veľký, cvičenie/prednáška)
- Číselník: Dni v týždni
- Číselník: Vyučovacie hodiny

Je predpoklad, že dané číselníky sa budú meniť len minimálne.

Role, prístupové práva

FRS bude definovať štyri základné role:

- Administrátor
- Rozvrhár
- Pedagóg
- Študent

V prvej verzii FRS budú mať tieto role len informatívny charakter a všetci prihlásení užívatelia budú vystupovať v roli administrátor. Po testovacej fáze v praxi sa dodatočne rozhodne, ktoré položky budú môcť prezerať/editovať ostatné role.

Predpokladá sa, že editáciu hlavných skupín objektov (Pedagógovia, Predmety...) bude môcť len administrátor. Rola Rozvrhár bude môcť len prideľovať jemu určené predmety. Rola Pedagóg bude môcť editovať svoj vlastný profil a časový harmonogram a rola Študent bude môcť pristupovať k niektorým vybraným zostavám.

Viacpoužívateľský prístup

FRS bude podporovať viacpoužívateľský prístup k aplikácii pomocou webového rozhrania. Vzájomná práca nad rovnakou podmnožinou dát bude v prvej verzii ošetrovaná formou kontroly integrity dát pri ukladaní zmien.

Ukladanie zmien bude riešené priebežne, po každej používateľskej akcii.

V prvej fáze nebude podporovaná možnosť verzionovania a vrátenia sa tak k predošlému stavu dát na strane aplikácie. V prípade potreby to bude zabezpečené na systémovej úrovni formou zálohy databázy.

5. Návrh riešenia

V nasledujúcej kapitole popíšem navrhované riešenie fakultného rozvrhového systému. Navrhnuté riešenie sa snaží v čo najlepším spôsobom zapracovať požiadavky špecifikované v predchádzajúcej kapitole (špecifikácii). Snažil som sa o jednoduchosť a zrozumiteľnosť, a takisto o čo najväčšie zapracovanie technológia Ajax do tohto riešenia.

5.1. Prostredie

Aplikácia fakultného rozvrhového systému bude prístupná pomocou internetového prehliadača. Vzhľadom na využitie technológie Ajax, s dôrazom na komunikáciu so serverom pomocou XMLHttpRequestu je potrebné, aby internetový prehliadač dané technológie podporoval. Podpora súčasných prehliadačov je bližšie spomenutá v predchádzajúcej kapitole. [kap. 2.11]

Obsluha používateľových požiadaviek bude prebiehať v nasledovných úrovniach:

- Základná obsluha (validácia vstupných dát) bude prebiehať pomocou JavaScriptu na strane klienta
- O asynchrónny prenos a dynamickú zmenu obsahu stránky sa bude starať Ajax engine, ktorý bude využívať XMLHttpRequest na asynchrónnu komunikáciu zo serverom a kombináciu JavaScriptu a DOM na dynamické prekresľovanie obsahu.
- Obsluha na strane servera bude realizovaná pomocou skriptovacieho jazyka, V tomto smere sa ako najvýhodnejšie riešenie javí jazyk PHP, vďaka jeho jednoduchosti, veľkej miere rozšírenia a v neposlednom rade, vzhľadom na akademickú pôdu, cene a otvorenému kódu.

Databáza

Na databázu nie sú kladené zložité požiadavky, vhodná je akákoľvek relačná databáza, pre ktorú bude existovať podpora v PHP. Vzhľadom na akademickú pôdu do užšieho výberu prichádzajú tieto najznámejšie OpenSource SQL databázy: [17]

- **MySQL** – Tento databázový server je obľúbený najmä v nasadení pre potreby webových aplikácii. Je veľmi rýchly, má dobrú podporu štandardov

jazyka SQL, ale v základnom móde mu chýbajú niektoré dôležité funkcie, ako napr.: transakcie, vnorené selekty, uložené procedúry a pohľady (view), kvôli čomu je diskvalifikovaný ako kandidát na zložitejšie aplikácie.

- **PostgreSQL** – Tento databázový server vyniká stabilitou, rýchlosťou, dobrou podporou a integráciou pokročilých funkcií, ktoré boli spomenuté ako nedostatok pri MySQL.
- **FirebirdSQL** – Databázový server, ktorý vznikol po dočasnom uvoľnení databázy Interbase, s ktorou je vysoko kompatibilný. Je to však nový databázový server (2004) preto nie je medzi používateľmi veľmi rozšírený.

Z uvedeného zoznamu SQL databáz navrhujem použiť databázu PostgreSQL.

Obslužné funkcie na komunikáciu s databázou budú v samostatnom module, aby v prípade požiadavky nasadenia programu na iný typ databázy, bolo možné tak urobiť s minimálnymi zmenami v programovacom kóde.

5.2. Dátový model

V systéme obsahuje štyri základné dátové objekty: pedagógov, predmety, miestnosti a rozvrhárov. Ďalej sú to jednoduché zoznamy (číselníky) predstavujúce dni v týždni, vyučovacie hodiny, študijné skupiny (krúžky) a typy predmetov (prednáška, cvičenie...). Dátový objekt predstavujúci samotný rozvrh bude tvoriť jedna referenčná tabuľka tvorená kombináciou identifikátorov záznamov v objektoch pedagóg, predmet a miestnosť a číselníkov dní a vyučujúcich hodín.

Každá z tabuliek obsahuje atribút ID, ktorý je zároveň jej primárnym kľúčom a slúži na presnú identifikáciu záznamu danej tabuľky.

V budúcnosti, pri zakomponovaní autentifikovaného viacpoužívateľského prístupu do aplikácie navrhujem rozšíriť každú tabuľku o atribúty *dátum*, *používateľ*, v ktorých bude uchovávané kedy a kto daný záznam vytvoril/upravil, prípadne ku každej tabuľke vytvoriť historickú tabuľku a pri každom *update* záznamu pôvodný záznam uložiť do historickej tabuľky.

Dátový objekt Pedagógovia

Dátový objekt *pedagógovia* bude riešený pomocou tabuľky `pedagog`. Tabuľka bude obsahovať atribúty *n_pedagog_id* (identifikátor záznamu tabuľky), *s_pedagog_name* (meno pedagóga), *s_pedagog_email* (email pedagóga),

s_pedagog_contact (kontaktné údaje o pedagógovi, napr. číslo kancelárie, telefón) a *s_pedagog_note* (voliteľná poznámka). Povinnými atribútmi sú *n_pedagog_id, s_pedagog_name, s_pedagog_email*.

pedagog	
n_pedagog_id	number
s_pedagog_name	varchar(200)
s_pedagog_email	varchar(200)
s_pedagog_contact	text
s_pedagog_note	text
primary_key (n_pedagog_id)	

Obr. 7. Tabuľka pedagog

Dátový objekt Predmet

Dátový objekt *predmety* bude riešený pomocou dvoch tabuliek `predmet` a `podpredmet`. Z toho dôvodu budeme doterajší pojem predmet ďalej rozlišovať na predmet, ktorý sa skladá z jednotlivých podpredmetov. (Napr.: predmet Matematická analýza I. obsahuje štyri podpredmety: prednášku, cvičenia pre 1i1, cvičenia pre 1i2 a cvičenia pre 1i3, z ktorých sa každý nasadzuje samostatne).

Tabuľka `predmet` bude obsahovať základné informácie o objekte predmet, konkrétne atribúty *n_predmet_id* (identifikátor záznamu tabuľky), *s_predmet_name* (názov predmetu), *s_predmet_code* (kód predmetu), *s_predmet_note* (voliteľná poznámka) a *n_rozvrhar_id* (identifikátor rozvrhára z tabuľky `rozvrhar`, ktorý daný predmet nasadzuje). Povinnými atribútmi sú *n_predmet_id, s_predmet_name, s_predmet_code*.

Tabuľka `podpredmet` bude predstavovať logické delenia predmetu na samostatné vyučovacie celky (prednáška, cvičenie...). Tabuľka bude obsahovať atribúty *n_podpredmet_id* (identifikátor), *s_skupina_pov* (povinné študijné skupiny), *s_skupina_vol* (voliteľné študijné skupiny), *n_podpredmet_hodiny* (počet vyučovacích hodín), *s_podpredmet_note* (voliteľná poznámka), *n_prtyp_id* (identifikátor typu predmetu z číselníka `prtyp`) a *n_predmet_id* (identifikátor predmetu z tabuľky `predmet`, do ktorého podpredmet patrí). Povinnými atribútmi sú *n_podpredmet_id, n_podpredmet_hodiny, n_prtyp_id*, *n_predmet_id.*

predmet	
n_predmet_id	number
s_predmet_name	varchar(200)
s_predmet_code	varchar(200)
s_predmet_note	text
n_rozvrhar_id	number
primary_key (n_predmet_id)	
foreign_key (n_rozvrhar_id, rozvrhar.n_rozvrhar_id)	

podpredmet	
n_podpredmet_id	number
s_skupina_pov	varchar(200)
s_skupina_vol	varchar(200)
n_podpredmet_hodiny	number
s_podpredmet_note	text
n_prtyp_id	number
n_predmet_id	number
primary_key (n_podpredmet_id)	
foreign_key (n_prtyp_id, prtyp.n_prtyp_id)	
foreign_key (n_rozvrhar_id, rozvrhar.n_rozvrhar_id)	

Obr. 8. Tabuľky predmet a podpredmet

Dátový objekt Miestnosti

Dátový objekt *miestnosti* bude riešený pomocou tabuľky `miestnost`. Tabuľka bude obsahovať atribúty *n_miestnost_id* (identifikátor záznamu tabuľky), *s_miestnost_name* (názov miestnosti), *s_miestnost_blok* (blok (budovu), v ktorej sa miestnosť nachádza), *s_miestnost_capacity* (kapacita miestnosti, počet miest na sedenie) a *s_miestnost_note* (voliteľná poznámka). Povinnými atribútmi sú *n_miestnost_id, s_miestnost_name, s_miestnost_blok*.

miestnost	
n_miestnost_id	number
s_miestnost_name	varchar(200)
s_miestnost_blok	varchar(200)
s_miestnost_capacity	varchar(200)
s_miestnost_note	text
primary_key (n_miestnost_id)	

Obr. 9. Tabuľka miestnost

Dátový objekt Rozvrhári

Dátový objekt *rozvrhári* bude riešený pomocou tabuľky `rozvrhar`. Tabuľka bude obsahovať atribúty *n_rozvrhar_id* (identifikátor záznamu tabuľky), *s_rozvrhar_name* (meno pedagóga), *s_rozvrhar_email* (email pedagóga), *s_rozvrhar_contact* (kontaktné údaje o pedagógovi, napr. číslo kancelárie, telefón) a *s_rozvrhar_note* (voliteľná poznámka). Povinnými atribútmi sú *n_rozvrhar_id, s_rozvrhar_name, s_rozvrhar_email*.

rozvrhar	
n_rozvrhar_id	number
s_rozvrhar_name	varchar(200)
s_rozvrhar_email	varchar(200)
s_rozvrhar_contact	text
s_rozvrhar_note	text
primary_key (n_rozvrhar_id)	

Obr. 10. Tabuľka rozvrhar

Číselníky

Číselníky predstavujú jednoduché zoznamy obsahujúce číselný identifikátor a textový reťazec. Databáza obsahuje nasledujúce číselníky:

- Tabuľka skupín: `skupina` – obsahujúca jednoduchý zoznam študijných skupín (1i1, 1i2, 1i3, 2i1, 2i2...) predstavujúca študijné skupiny (krúžky) vo forme ako je zaužívaná na fakulte. Prvé číslo predstavuje ročník, textový reťazec predstavuje študijný odbor a posledné číslo poradie študijnej skupiny.
- Tabuľka typov predmetov: `prtyp` – obsahujúca jednoduchý zoznam typov predmetov (veľká prednáška, cvičenie, výberovka...)

Číselník, predstavujúci dni v týždni vo svojom plnom tvare (Pondelok, Utorok, ...) ako aj v skrátenom tvare (Pon., Uto., Str., ...) nebude riešený formou tabuľky ale formou dátovej premennej (indexového poľa) v prostredí programu. Takisto číselník vyučovacích hodín (8:10, 9:00, 9:50, 10:00) bude riešený formou dátovej premennej (indexového poľa) v prostredí programu.

Referenčný objekt Rozvrh

Na obsiahnutie priradení pedagógov jednotlivým predmetom, priradení jednotlivých podpredmetov do daných miestností a ich zaradenie do časovej osi bola zvolená jediná referenčná tabuľka `rozvrh`. Tabuľka obsahuje atribúty *n_rozvrh_id* (identifikátor záznamu tabuľky), *n_den* (index dňa z číselníkového poľa `den`), *n_hodina* (index hodiny z číselníkového poľa `hodina`), *n_miestnost_id* (identifikátor miestnosti z tabuľky `miestnost`), *n_predmet_id* (identifikátor predmetu z tabuľky `predmet`), *n_podpredmet_id* (identifikátor podpredmetu z tabuľky `podpredmet`), *n_pedagog_id*

(identifikátor pedagóga, z tabuľky `pedagog`). Povinnými atribútmi sú *n_rozvrh_id, n_predmet_id, n_podpredmet_id, n_pedagog_id.*

rozvrh	
n_rozvrh_id	number
n_den	number
n_hodina	number
n_miestnost_id	number
n_predmet_id	number
n_podpredmet_id	number
n_pedagog_id	number
primary_key (n_prtyp_id)	
foreign_key (n_miestnost_id, miestnost.n_miestnost_id)	
foreign_key (n_predmet_id, predmet.n_predmet_id)	
foreign_key (n_podpredmet_id, podpredmet.n_podpredmet_id)	
foreign_key (n_pedagog_id, pedagog.n_pedagog_id)	

Obr. 11. Tabuľka rozvrh

Každý záznam tabuľky bude predstavovať nasadzovanie jednotlivej časti podpredmetu v kombinácii s pedagógom do rozvrhu. V prípade, že bude záznam obsahovať aj nenulové atribúty *n_den, n_hodina, n_miestnost_id*, znamená to, že podpredmet (resp. jeho časť) je už nasadený. Takto zvolená referenčná tabuľka `rozvrh` umožňuje zaznamenať všetky požadované kombinácie nasadenia. Bližšie to popíšem na príkladoch:

- Podpredmet Analýza (typ cvičenie) s priradeným pedagógom Hraško v dĺžke trvania výučby 1 hodina nie je nasadený v rozvrhu:

n_rozvrh_id	n_den	n_hodina	n_miestnost_id	n_predmet_id	n_podpredmet_id	n_pedagog_id
1				1 (analýza)	1 (cvičenie)	1 (hraško)

- Predchádzajúci prípad, ale už s nasadením do rozvrhu:

n_rozvrh_id	n_den	n_hodina	n_miestnost_id	n_predmet_id	n_podpredmet_id	n_pedagog_id
1	1 (pondelok)	1 (8:10)	1 (akvárko I.)	1 (analýza)	1 (cvičenie)	1 (hraško)

- Výučba v dĺžke dve hodiny:

n_rozvrh_id	n_den	n_hodina	n_miestnost_id	n_predmet_id	n_podpredmet_id	n_pedagog_id
1	1 (pondelok)	1 (8:10)	1 (akvárko I.)	1 (analýza)	1 (cvičenie)	1 (hraško)
2	1 (pondelok)	2 (9:00)	1 (akvárko I.)	1 (analýza)	1 (cvičenie)	1 (hraško)

-

- Podpredmet vyučujú dvaja pedagógovia súčasne:

n_rozvrh_id	n_den	n_hodina	n_miestnost_id	n_predmet_id	n_podpredmet_id	n_pedagog_id
1	1 (pondelok)	1 (8:10)	1 (akvárko I.)	1 (analýza)	1 (cvičenie)	1 (hraško)
2	1 (pondelok)	1 (8:10)	1 (akvárko I.)	1 (analýza)	1 (cvičenie)	2 (mrkvička)

- Podpredmet Analýza (typ cvičenie) v trvaní 3 vyučovacie hodiny, je rozdelený na dve časti. Prvú časť, v dĺžke 1 hodina, vyučujú v pondelok o 9:50 v miestnosti akvárko I. dvaja pedagógovia (hraško, mrkvička) súčasne. Druhú časť, v dĺžke 2 hodiny vyučuje jeden pedagóg (mrkvička) v utorok od 8:10 v miestnosti H6.

n_rozvrh_id	n_den	n_hodina	n_miestnost_id	n_predmet_id	n_podpredmet_id	n_pedagog_id
1	1 (pondelok)	3 (9:50)	1 (akvárko I.)	1 (analýza)	1 (cvičenie)	1 (hraško)
2	1 (pondelok)	3 (9:50)	1 (akvárko I.)	1 (analýza)	1 (cvičenie)	2 (mrkvička)
3	2 (utorok)	1 (8:10)	15 (H6)	1 (analýza)	1 (cvičenie)	2 (mrkvička)
4	2 (utorok)	2 (9:00)	15 (H6)	1 (analýza)	1 (cvičenie)	2 (mrkvička)

Jednoduchými selektami potom môžeme z takto navrhnutej tabuľky získavať rozvrhy pre daného pedagóga, predmet, podpredmet, miestnosť

5.3. Rozdelenie aplikačného okna

Aplikácia bude logicky rozdelená na menu a hlavnú časť. Menu bude zabezpečovať prístup k jednotlivým činnostiam:

- **Činnosť Rozvrh** – hlavná činnosť na nasadzovanie rozvrhu.
- **Činnosť Pedagógovia** – činnosť na správu a súhrnné informácie o pedagógoch.
- **Činnosť Predmety** – činnosť na správu a súhrnné informácie o predmetoch.
- **Činnosť Miestnosti** – činnosť na správu a súhrnné informácie o miestnostiach.
- **Činnosť Reporty** – činnosť na prezeranie a generovanie súrnych reportov.
- **Činnosť Rozvrhári** – činnosť na správu a súhrnné informácie o rozvrhároch.

Činnosť Pedagógovia, Predmety, Miestnosti budeme nazývať aj činnosti na zadávanie vstupných údajov.

Komponenty

Jednotlivé činnosti budú pozostávať z viacerých samostatných logických častí (komponentov). Obsah každého z týchto komponentov sa bude načítavať asynchrónne pomocou Ajaxu. Ak používateľova akcia bude vyžadovať zmenu obsahu niektorého komponentu, táto zmena bude vykonaná len na danom komponente. V prípade, že bude pri zmene potrebné načítať obsah komponentu, alebo jeho časti zo serveru, bude pomocou XMLHttpRequestu odoslaná požiadavka na server a vrátený obsah pomocou JavaScriptu spracovaný a následne vykonaná zmena obsahu komponentu. Obsah ostaných komponentov ostane nezmenený (okrem prípadu, keď zmena obsahu komponentu bude logicky vyžadovať zmenu iného komponentu).

Identifikácia komponentov

Identifikácia komponentov, ich podkomponentov a jednotlivých prvkov dokumentu bude zabezpečená pomocou identifikátorov. Komponenty budú tvorené párovým html tagom `<div id="identifikator_komponentu">` s jednoznačným logickým názvom (napr.: `id="pedagogZoznam"`). Ucelené zoznamy prvkov, ktoré budú zobrazené v dokumente a budú vyžadovať identifikáciu (napríklad zoznam pedagógov), budú identifikované pomocou automaticky generovaného unikátneho id príslušného html tagu. Prístup k takto identicky označeným komponentom a prvkom je potom realizovaný pomocou JavaScriptu a metód Objektového modelu dokumentu (DOM).

5.4. Menu

Aplikácia bude obsahovať spoločné menu na prístup k jednotlivým činnostiam.

Kliknutie na položku menu bude slúžiť na načítanie obsahu zvolenej činnosti do okna aplikácie. V tomto kroku zároveň nastane načítanie celého aplikačného okna (refresh). Všetky ostatné spracovania používateľových požiadaviek už budú prebiehať pomocou Ajaxu, teda bez nutnosti opätovne načítavať celú stránku dokumentu, ale len príslušný komponent.

5.5. Činnosti na zadávanie vstupných údajov

V aplikácii budú štyri činnosti na zadávanie vstupných údajov pre jednotlivé skupiny objektov: pedagógovia, predmety, miestnosti, rozvrhári. Činnosti budú mať nasledovné spoločné znaky:

- Obrazovka aplikácie bude logicky rozdelená na dva komponenty – na zoznam a detaily.
- V časti zoznam bude zoznam pedagógov, predmetov, miestností a rozvrhárov - podľa druhu činnosti.
- Po kliknutí na položku zoznamu sa v pravej časti zobrazia podrobné údaje o vybranom objekte.
- V činnostiach bude možné zadať nový objekt, editovať objekt, zmazať objekt.
- Pri novom a editovanom objekte bude po používateľovej akcii *uložiť* najskôr vykonaná validácia na správnosť vložených údajov, a samotné uloženie prebehne až po úspešnej validácii. V prípade, že zadané údaje nebudú validné, používateľovi bude umožnené údaje ďalej editovať.

5.6. Činnosť Pedagógovia

Činnosť *pedagóg* slúži na správu a súhrnné informácie o pedagógoch. Činnosť bude obsahovať nasledujúce komponenty:

- Zoznam pedagógov.
- Podrobnosti pedagóga.
- Priradené predmety (podpredmety)
- Týždenný časový harmonogram výučby pedagóga (rozvrh pedagóga).

Obr. 12. Rozloženie komponentov činnosti predmet

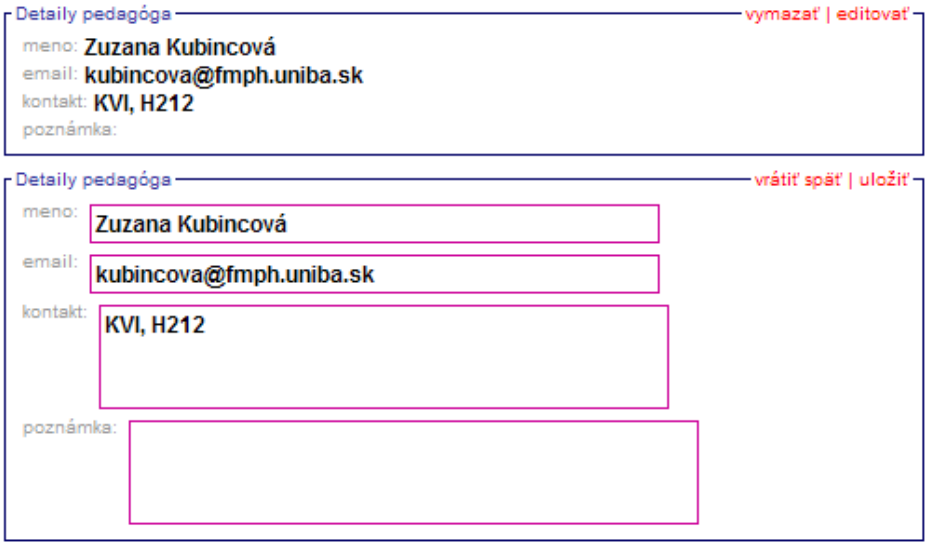
Komponent Zoznam pedagógov

Komponent *zoznam pedagógov* bude obsahovať zoznam všetkých pedagógov (tabuľka `pedagog`). Po kliknutí na niektorého pedagóga sa do ostaných komponentov načítajú podrobné informácie o danom pedagógovi. Komponent bude obsahovať tlačidlo *nový pedagóg* ktoré umožní vytvoriť nového pedagóga.

Obr. 13. Komponent Zoznam pedagógov

Komponent Podrobnosti pedagóga

Komponent *Podrobnosti pedagóga* môže existovať v dvoch stavoch. V stave *prezeranie* a v stave *editácia*.



Obr. 14. Komponent Podrobnosti pedagóga v stave prezeranie a editácia.

V stave *prezeranie* bude komponent obsahovať bližšie informácie o zvolenom pedagógovi, ako meno pedagóga, email, kontakt a poznámku. Ďalej bude obsahovať tlačidlá *vymazať a editovať*.

Po stlačení tlačidla *vymazať* sa používateľovi zobrazí potvrdzovacie okno s otázkou, či naozaj chce vymazať daného pedagóga. Po jeho potvrdení sa pomocou *Ajax engine* vytvorí požiadavka na vymazanie pedagóga z databázy. Obslužný program na strane servera vymaže daného pedagóga z databázy, ako aj jemu priradené predmety v rozvrhu. O úspešnom vymazaní pedagóga bude používateľ oboznámený informačným oknom. Zároveň sa daný pedagóg pomocou *Ajax engine* vymaže zo zoznamu pedagógov a všetky komponenty činnosti sa nastavia do svojej iniciálnej podoby.

Po stlačení tlačidla *editovať* sa komponent nastaví do stavu *editácia*.

V stave *editácia* bude komponent obsahovať editovateľné polia jednotlivých položiek. Ďalej bude obsahovať tlačidlá *vrátiť späť* a *uložiť*.

Pri stlačení tlačidla *vrátiť späť* sa komponent spätne nastaví do stavu *prezeranie*, bez uloženia vykonaných zmien.

Po stlačení tlačidla *uložiť* sa vykoná nasledovné:

- Pomocou JavaScriptu prebehne primárna validácia vložených dát [kap. 5.12], konkrétne validácia na neprázdnosť položiek meno pedagóga a správnosť zadania emailového reťazca.
- V prípade neúspešnej primárnej validácie sa o tom používateľovi zobrazí informačné okno s popisom chyby.
- V prípade úspešnej primárnej validácie sa pomocou *Ajax engine* sa vytvorí požiadavka na server s parametrami požadovanej zmeny.
- Na strane servera prebehne sekundárna validácia [kap. 5.12], konkrétne validácia na to, či sa pedagóg s daným menom už nenachádza v databáze.
- V prípade úspešnej sekundárnej validácie za požadované zmeny uložia do databázy.
- Server vráti výsledok spracovania požiadavky. *Ajax engine* vrátený výsledok požiadavky spracuje a zobrazí používateľovi formou informačného okna.
- Aktuálna položka v zozname pedagógov sa pomocou *Ajax Engine* zaktualizuje podľa vykonanej zmeny mena pedagóga.

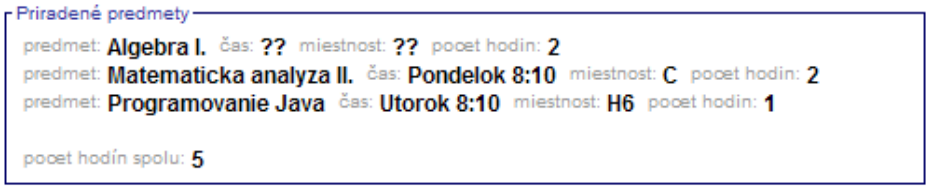
Špeciálnym prípadom je pridanie nového pedagóga (tlačidlo *nový pedagóg*), ktorý je analogický k *editácii pedagóga*. Rozdiel je v tom, že editované polia nie sú predvyplnené, a na strane servera neprebieha *update* záznamu ale vytvorenie nového.

Iniciálnou podobou komponentu *podrobnosti pedagóga* je zobrazenie podrobných informácii o pedagógovi v stave *prezeranie*, alebo je komponent prázdny, pokiaľ nie je vybraný žiadny predmet zo zoznamu.

Komponent Priradené predmety

Komponent *priradené predmety* zobrazuje zoznam predmetov priradených vybranému pedagógovi. Jednotlivé položky zoznamu budú obsahovať názov predmetu, počet vyučovacích hodín a v prípade, že je daný predmet už zaradený do rozvrhu, bude zobrazená informácie o priradenom čase a miestnosti. Pod zoznamom priradených predmetov pedagóga bude súhrnná informácia o počte vyučovacích hodín pedagóga (úväzku).

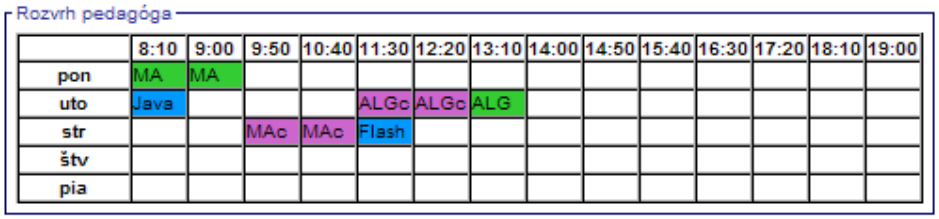
Komponent *priradené predmety* bude mať len informatívny charakter a nebude editovateľný. Jednotlivé priradenia predmetov pedagógom prebiehajú v činnosti *predmety*.



Obr. 15. Komponent Priradené predmety pedagóga

Komponent Rozvrh pedagóga

Komponent *rozvrh pedagóga* bude obsahovať grafickým spôsobom zobrazený týždenný rozvrh daného pedagóga. V rozvrhu budú graficky odlíšené jednotlivé typy predmetov. Zobrazený rozvrh bude mať informatívny charakter a nebude editovateľný. Nasadzovanie jednotlivých predmetov do rozvrhu prebieha v činnosti na nasadzovanie rozvrhu. [kap. 5.10]

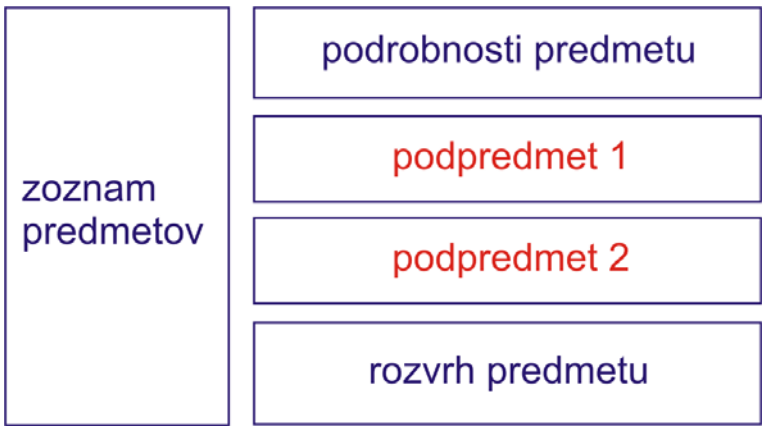


Obr. 16. Komponent Rozvrh pedagóga

5.7. Činnosť Predmety

Činnosť *predmet* slúži na správu a súhrnné informácie o predmetoch. Činnosť bude obsahovať nasledujúce komponenty:

- Zoznam predmetov.
- Podrobnosti predmetu.
- Komponenty jednotlivých podpredmetov daného predmetu.
- Týždenný časový harmonogram výučby predmetu (rozvrh predmetu).



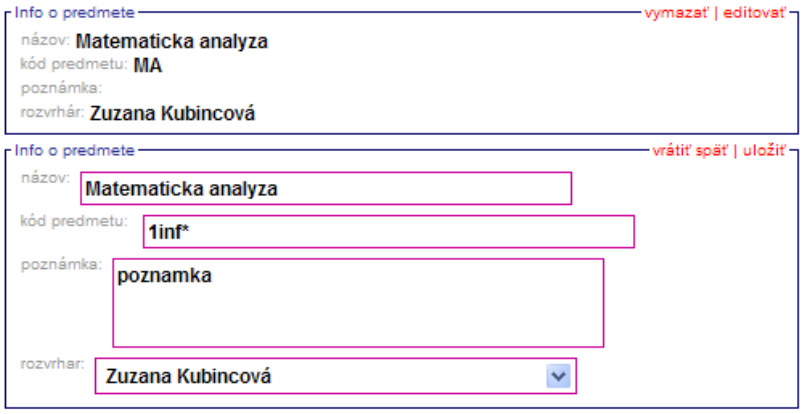
Obr. 17. Rozloženie komponentov činnosti predmet

Komponent Zoznam predmetov

Komponent *zoznam predmetov* bude obsahovať zoznam všetkých predmetov (tabuľka `predmet`). Po kliknutí na niektorý predmet sa do ostaných komponentov načítajú podrobné informácie o danom predmete. Komponent bude obsahovať tlačidlo *nový predmet* ktorý umožní vytvoriť nový predmet.

Komponent Podrobnosti predmetu

Komponent *Podrobnosti predmetu* môže existovať v dvoch stavoch. V stave *prezeranie* a v stave *editácia*.



Obr. 18. Komponent Podrobnosti predmetu v stave prezeranie a editácia.

V stave *prezeranie* bude komponent obsahovať bližšie informácie o zvolenom predmete, ako názov predmetu, kód predmetu, poznámku a rozvrhára, ktorý daný predmet nasadzuje. Ďalej bude obsahovať tlačidlá *vymazať predmet*, *editovať predmet a pridať nový podpredmet.*

Po stlačení tlačidla *vymazať predmet* sa používateľovi zobrazí potvrdzovacie okno s otázkou, či naozaj chce vymazať daný predmet. Po jeho potvrdení sa pomocou *Ajax engine* vytvorí požiadavka na vymazanie predmetu z databázy. Obslužný program na strane servera vymaže daný predmet z databázy, ako aj jemu priradené podpredmety a jednotlivé nasadenia podpredmetov v rozvrhu. O úspešnom vymazaní predmetu bude používateľ oboznámený informačným oknom. Zároveň sa daný predmet pomocou *Ajax engine* vymaže zo zoznamu predmetov a všetky komponenty činnosti sa nastavia do svojej iniciálnej podoby.

Po stlačení tlačidla *editovať predmet* sa komponent nastaví do stavu *editácia*.

V stave *editácia* bude komponent obsahovať editovateľné polia jednotlivých položiek. Ďalej bude obsahovať tlačidlá *vrátiť zmeny* a *uložiť zmeny*. Položka rozvrhár nebude editovateľná. Priradenie konkrétneho predmetu rozvrhárovi bude možné v činnosti *Rozvrhár*.

Pri stlačení tlačidla *vrátiť zmeny* sa komponent spätne nastaví do stavu *prezeranie*, bez uloženia vykonaných zmien.

Po stlačení tlačidla *uložiť zmeny* sa vykoná nasledovné:

- Pomocou JavaScriptu prebehne primárna validácia vložených dát [kap. 5.12], konkrétne validácia na neprázdnosť položiek názov predmetu a kód predmetu.

- V prípade neúspešnej primárnej validácie sa o tom používateľovi zobrazí informačné okno s popisom chyby.
- V prípade úspešnej primárnej validácie sa pomocou *Ajax engine* sa vytvorí požiadavka na server s parametrami požadovanej zmeny.
- Na strane servera prebehne sekundárna validácia [kap. 5.12], konkrétne validácia na to, či sa predmet s daným názvom alebo kódom už nenachádza v databáze.
- V prípade úspešnej sekundárnej validácie za požadované zmeny uložia do databázy.
- Server vráti výsledok spracovania požiadavky. *Ajax engine* vrátený výsledok požiadavky spracuje a zobrazí používateľovi formou informačného okna.
- Aktuálna položka v zozname predmetov sa pomocou *Ajax Engine* zaktualizuje podľa vykonanej zmeny názvu predmetu.

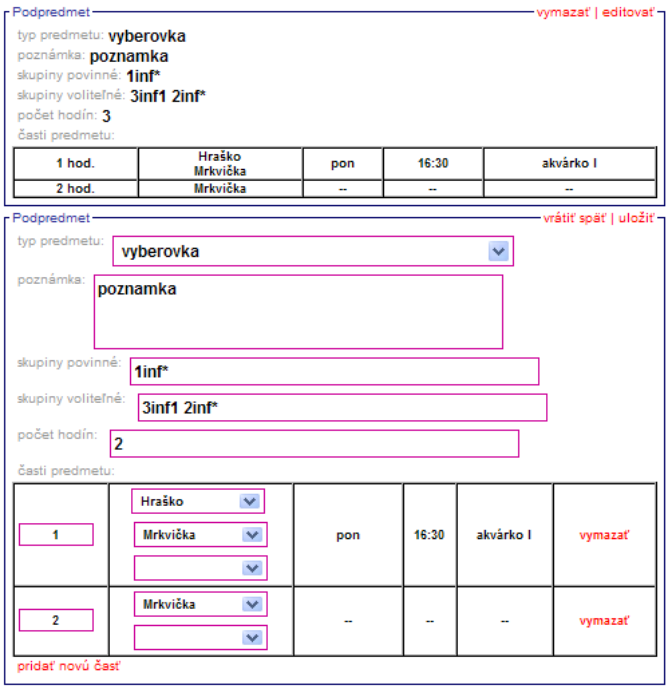
Špeciálnym prípadom je pridanie nového predmetu (tlačidlo *nový predmet*), ktorý je analogický k *editácii predmetu*. Rozdiel je v tom, že editované polia nie sú predvyplnené, a na strane servera neprebieha *update* záznamu ale vytvorenie nového.

Po stlačení tlačidla *pridať nový podpredmet* sa vytvorí nový komponent *Podpredmet* v stave *editácia.*

Iniciálnou podobou komponentu *Podrobnosti predmetu* je zobrazenie podrobných informácií o predmete v stave *prezeranie*, alebo je komponent prázdny, pokiaľ nie je vybraný žiadny predmet zo zoznamu.

Komponent Podpredmety

Komponent *Podpredmety* obsahuje podrobné informácie o podpredmetoch (tabuľka `podpredmet`) a môže existovať v dvoch stavoch. V stave *prezeranie* a v stave *editácia*.



Obr. 19. Komponent Podpredmet v stave prezeranie a editácia.

V stave *prezeranie* bude komponent obsahovať bližšie informácie o zvolenom podpredmete, ako typ podpredmetu, priradené študijné skupiny (povinné a voliteľné), počet hodín, poznámku a subkomponent *časti podpredmetu* v stave *prezeranie*. Ďalej bude obsahovať tlačidlá *vymazať podpredmet* a *editovať podpredmet.*

Po stlačení tlačidla *vymazať podpredmet* sa používateľovi zobrazí potvrdzovacie okno s otázkou, či naozaj chce vymazať daný podpredmet. Po jeho potvrdení sa pomocou *Ajax engine* vytvorí požiadavka na vymazanie podpredmetu z databázy. Obslužný program na strane servera vymaže daný podpredmet z databázy ako jednotlivé nasadenia podpredmetu v rozvrhu. O úspešnom vymazaní podpredmetu bude používateľ oboznámený informačným oknom. Zároveň sa daný komponent *podpredmet* pomocou *Ajax engine* vymaže zo zobrazeného dokumentu.

Po stlačení tlačidla *editovať podpredmet* sa komponent nastaví do stavu *editácia*.

V stave *editácia* bude komponent obsahovať bližšie informácie o zvolenom podpredmete v editovateľných poliach a subkomponent *nasadené hodiny* v stave editácia. Položky priradená študijné skupiny a poznámka budú v tvare editovateľných polí, a položka typ podpredmetu bude v tvare výberového poľa s obsahom číselníka `prtyp`. Ďalej bude obsahovať tlačidlá *vrátiť zmeny* a *uložiť zmeny*. Subkomponent *časti podpredmetu* bude bližšie popísaný v nasledujúcej kapitole.

Pri stlačení tlačidla *vrátiť zmeny* sa komponent spätne nastaví do stavu *prezeranie*, bez uloženia vykonaných zmien.

Po stlačení tlačidla *uložiť zmeny* sa vykoná nasledovné:

- Pomocou JavaScriptu prebehne primárna validácia vložených dát [kap. 5.12], konkrétne validácia na:
  - neprázdnosť položiek typ predmetu a počet hodín
  - súčet hodín v subkomponente *časti podpredmetu* musí byť rovný alebo väčší ako hodnota položky *počet hodín*.
  - Existencia zadaných študijných skupín v číselníku skupiny, pričom sa zohľadní možnosť zadať skupinu v „hviezdičkovej forme“ (1i*, 2m*…)
- V prípade neúspešnej primárnej validácie sa o tom používateľovi zobrazí informačné okno s popisom chyby.
- V prípade úspešnej primárnej validácie sa pomocou *Ajax engine* sa vytvorí požiadavka na server s parametrami požadovanej zmeny.
- Server vráti výsledok spracovania požiadavky. *Ajax engine* vrátený výsledok požiadavky spracuje a zobrazí používateľovi formou informačného okna.

Špeciálnym prípadom je pridanie nového podpredmetu (tlačidlo *pridať nový podpredmet*), ktorý je analogický k *editácii podpredmetu*. Rozdiel je v tom, že editované polia nie sú predvyplnené, a na strane servera neprebieha *update* záznamu ale vytvorenie nového.

Iniciálnou podobou tohto komponentu sú zobrazené podrobné informácie o podpredmete v stave *prezeranie*.

Subkomponent Časti podpredmetu

Subkomponent *Časti podpredmetu* je súčasťou komponentu *podpredmety* a zobrazuje stav nasadenia konkrétneho podpredmetu v rozvrhu. Môže sa nachádzať v dvoch stavoch, v stave *prezeranie* a v stave *editácia*.

Úlohou tohto subkomponentu je zabezpečenie funkcionality možnosti nasadenia podpredmetu v rôznych variáciách a to:

- Rozdelenie výučby podpredmetu s väčším počtom vyučovacích hodín na viacero častí a ich nezávislé nasadenie do rozvrhu.

- Umožnenie nasadenia konkrétneho podpredmetu do rozvrhu viackrát (alternatívne nasadenie)
- Umožnenie priradiť podpredmetu viacero pedagógov vyučujúcich súčasne.
- Umožnenie priradiť podpredmetu viacero pedagógov, z ktorých každý vyučuje len jeho časť.

Každá jednotlivá časť podpredmetu bude obsahovať nasledovné informácie:

- Dĺžka výučby vo vyučovacích hodinách.
- Pedagógov, vyučujúcich konkrétnu časť podpredmetu
- Informáciu o nasadení časti v rozvrhu, ak už bola nasadená

V stave *editácia* bude subkomponent obsahovať editovateľné položky dĺžka výučby a výberové polia pedagógov. Ďalej bude obsahovať tlačidlo *pridať novú časť* a pre každú časť tlačidlo *vymazať*.

Počet výberových polí pedagógov bude vždy o jedno väčší, ako je aktuálny počet priradených pedagógov. Výberové poliach budú prednastavené na hodnoty priradených pedagógov. Posledné výberové pole bude prednastavené na prázdny záznam. V prípade zmeny výberu posledného nulového záznamu, sa pomocou obslužného kódu (JavaScriptu) vykoná validácia na existenciu výberu toho istého pedagóga v iných výberových poliach a na koniec sa pridá nové výberové pole s prednastaveným prázdnym záznamom. V prípade, že sa vyskytnú dve prázdne výberové polia, obslužný kód jedno z nich vymaže zo zobrazeného dokumentu. Touto funkcionalitou sa zabezpečí, že bude možné jednej časti podpredmetu priradiť viacero pedagógov.

Pri stlačení tlačidla *uložiť* (v komponente Podpredmety) sa vykoná nasledujúca primárna validácia:

- Musí existovať aspoň jedna časť podpredmetu.
- Dĺžka výučby jednotlivých častí musí byť kladné číslo.
- Musí existovať aspoň jeden pedagóg pre každú časť.

Následne sa obsah subkomponentu *časti podpredmetu* uloží do databázy analogickým spôsobom ako v prípade vyššie spomínaných komponentov.

V praxi bude väčšina podpredmetov obsahovať len jednu časť podpredmetu s jedným priradeným pedagógom.

Komponent Rozvrh predmetu

Komponent *rozvrh predmetu* bude obsahovať grafickým spôsobom zobrazený týždenný rozvrh daného predmetu, resp. jednotlivých častí jeho podpredmetov. V rozvrhu budú graficky odlíšené jednotlivé typy podpredmetov.

Komponent *rozvrh predmetu* je podobný komponentu *rozvrh pedagóga* z činnosti *Pedagógovia* preto ho nebudem ďalej popisovať. [kap. 5.6]

5.8. Činnosť Miestnosť

Činnosť *Miestnosť* slúži na správu a súhrnné informácie o miestnostiach. Činnosť je podobná s činnosťou *Pedagógovia*, preto spomeniem len ich odlišnosti.

Činnosť obsahuje komponenty:

- Zoznam miestností - (tabuľka `miestnost`)
- Podrobnosti o miestnosti – obsahujúca položky názov miestnosti, blok, kapacita miestnosti a poznámka. Povinnou položkou je názov miestnosti.
- Týždenný časový harmonogram výučby v miestnosti (rozvrh miestnosti).

5.9. Činnosť Rozvrhári

Činnosť *Rozvrhári* slúži na správu a súhrnné informácie o rozvrhároch. Činnosť je podobná s činnosťou *Pedagógovia*, preto spomeniem len ich odlišnosti.

Činnosť obsahuje komponenty:

- Zoznam rozvrhárov - (tabuľka `rozvrhar`).
- Podrobnosti o rozvrhároch – obsahujúca položky meno rozvrhára, email, kontakt a poznámka. Povinnou položkou je meno rozvrhára a kontakt.
- Zoznam priradených predmetov.

Komponent Zoznam priradených predmetov

Komponent *Zoznam priradených predmetov* obsahuje zoznam predmetov, ktoré daný rozvrhár nasadzuje. Komponent môže existovať v dvoch stavoch. V stave *prezeranie* a v stave *editácia*.

V stave *prezeranie* bude komponent obsahovať needitovateľný zoznam priradených predmetov a tlačidlo *editovať*. Jednotlivé položky zoznamu budú obsahovať názov predmetu, a informáciu o počte nasadených časti podpredmetu daného

predmetu. Po kliknutí na meno predmetu v zozname bude používateľ presmerovaný do činnosti *predmety* na konkrétny predmet, kde môže daný predmet editovať, napr. priradiť častiam podpredmetov jednotlivých pedagógov.

Pod zoznamom priradených predmetov rozvrhára bude súhrnná informácia o počte nasadených časti podpredmetu daného predmetu.

Po stlačení tlačidla *editovať* sa komponent dostane do stavu *editácia*.

V stave *editácia* bude komponent obsahovať zoznam predmetov, kde každý predmet bude zobrazený formou výberového poľa a navyše jeden riadok zoznamu bude obsahovať pole s vybraným prázdnym záznamom. Ďalší postup pridávania a odoberania predmetov je analogický s postupom pridávania a odoberania pedagógov v subkomponente *Časti podpredmetov* v činnosti *Predmety* [kap. 5.7], preto ju nebudem podrobnejšie popisovať.

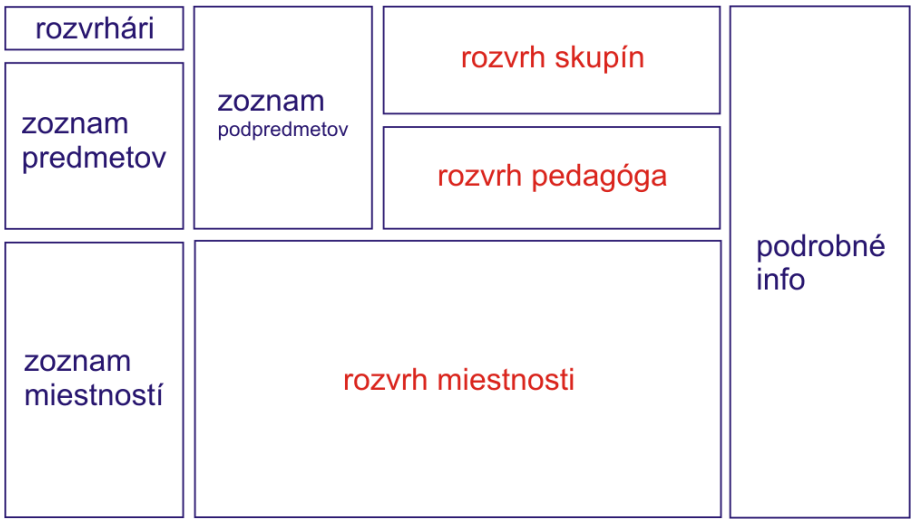
Aby sa zabránilo priradeniu jedného predmetu viacerým rozvrhárom, vo výberovom poli budú len nepriradené predmety.

5.10. Činnosť na nasadzovanie rozvrhu

V nasledujúcej kapitole bude popísaný návrh hlavnej činnosti navrhovaného fakultného rozvrhového systému – nasadzovanie rozvrhu.

Po kliknutí na položku rozvrhy v menu sa v hlavnej časti aplikácie zobrazia nasledujúce komponenty (v zátvorke uvádzame ich skrátený tvar, používaný ďalej v texte):

- Výber rozvrhára (rozvrhár)
- Zoznam predmetov (predmety)
- Zoznam podpredmetov (podpredmety)
- Zoznam miestností (miestnosti)
- Týždenný časový harmonogram skupín (rozvrh skupín)
- Týždenný časový harmonogram pedagóga (rozvrh pedagóga)
- Týždenný časový harmonogram miestnosti (rozvrh miestností)
- Podrobné informácie



Obr. 20. Rozloženie komponentov činnosti rozvrh

Komponent Výber rozvrhára

Komponent *výber rozvrhára* bude obsahovať výberové pole so zoznamom rozvrhárov. Po zmene vybranej hodnoty výberového poľa (zmene rozvrhára) sa do komponentu *predmety* načíta zoznam predmetov príslušného rozvrhára a ostatné komponenty sa nastavia do svojej iniciálnej podoby.

Iniciálnou podobou tohto komponentu je zoznam rozvrhárov bez konkrétneho určenia niektorej položky zoznamu.

Komponent Zoznam predmetov

Komponent *zoznam predmetov* bude obsahovať zoznam predmetov daného rozvrhára. Po výbere konkrétneho predmetu (kliknutí) sa vybraný predmet graficky zvýrazní a do komponentu *zoznam podpredmetov* sa načítajú podpredmety prináležiace danému predmetu. Komponenty *zoznam miestnosti*, *rozvrh skupín* a *rozvrh pedagóga* sa nastavia do svojej iniciálnej podoby. Do komponentu *rozvrh miestností* sa (neštandardne) načíta týždenný časový harmonogram nasadenia všetkých podpredmetov vybraného predmetu. Tento neštandardný obsah komponentu *rozvrh miestností* bude graficky odlíšený zmenou titulku komponentu.

Iniciálnou podobou tohto komponentu bude zoznam predmetov podľa zvoleného rozvrhára alebo prázdny zoznam.

Komponent Zoznam podpredmetov

Komponent *zoznam podpredmetov* bude obsahovať zoznam podpredmetov vybraného predmetu. O jednotlivých podpredmetoch budú zobrazené nasledovné informácie:

- typ podpredmetu
- vyučujúci pedagógovia
- priradené študijné skupiny
- počet hodín
- poznámku
- zoznam častí podpredmetu

Podpredmety, ktoré už sú v rozvrhu nasadené budú graficky odlíšené.

Po výbere konkrétneho podpredmetu (kliknutí) sa vybraný podpredmet graficky zvýrazní. Komponent *zoznam miestností* sa nastaví do svojej iniciálnej podoby. Do komponentu *rozvrh miestností* sa (neštandardne) načíta týždenný časový harmonogram nasadenia vybraného podpredmetu. Tento neštandardný obsah komponentu *rozvrh miestností* bude graficky odlíšený zmenou titulku komponentu. Do komponentu *rozvrh skupín* sa načíta týždenný časový harmonogram prináležiacich študijných skupín a do komponentu *rozvrh pedagóga* sa načíta týždenný časový harmonogram prináležiacich pedagógov.

Komponent *zoznam podpredmetov* bude obsahovať tlačidlo *zmena predmetu*, po ktorého stlačení (kliknutí) bude načítaná *činnosť predmet*, kde bude možná editácia podpredmetov daného predmetu.

Každá položka zoznamu častí podpredmetu bude obsahovať tlačidlo *zrušenie nasadenia* v prípade, že už je nasadená, alebo tlačidlo *nasadiť do rozvrhu*. Pri zrušení nasadenia bude používateľ vyzvaný k potvrdeniu svojho rozhodnutia. Nasadenie predmetu do rozvrhu bude popísané v nasledujúcich odsekoch.

Iniciálnou podobou tohto komponentu bude zoznam podpredmetov podľa zvoleného predmetu alebo prázdny zoznam.

Komponent Zoznam miestností

Komponent *zoznam miestností* bude obsahovať zoznam miestností (tabuľka `miestnost`) logicky zoradených podľa jednotlivých blokov. Po výbere konkrétnej

miestnosti (kliknutí) sa vybraná miestnosť graficky zvýrazní a do komponentu *rozvrh miestností* sa načíta prislúchajúci týždenný rozvrh vybranej miestnosti.

Iniciálnou podobou tohto komponentu bude zoznam miestností.

Komponent Rozvrh skupín

Komponent *rozvrh skupín* bude obsahovať týždenný časový harmonogram (rozvrh) študijných skupín, ktoré sú priradené vybranému podpredmetu. V prípade viacerých priradených študijných skupín bude umožnené pomocou tlačidiel (odkazov) zobraziť rozvrh pre všetky študijné skupiny súčasne (jednotlivé skupiny grafickým rozlíšené), alebo zobraziť rozvrh len pre jednu študijnú skupinu.

V prípade, že sa kurzor myši bude nachádzať nad nejakým nasadeným predmetom v rozvrhu, zobrazia sa v komponente *podrobné informácie* podrobnosti o danom predmete.

Iniciálnou podobou tohto komponentu bude týždenný časový harmonogram priradených študijných skupín daného podpredmetu alebo prázdny rozvrh.

Rozvrh pedagógov

Komponent *rozvrh pedagógov* bude obsahovať týždenný časový harmonogram (rozvrh) jednotlivých pedagógov, ktorí sú priradení vybranému podpredmetu. V prípade viacerých pedagógov bude umožnené pomocou tlačidiel (odkazov) zobraziť rozvrh pre všetkých pedagógov súčasne (grafickým rozlíšených), alebo zobraziť rozvrh len pre jedného pedagóga.

V prípade, že sa kurzor myši bude nachádzať nad nejakým nasadeným predmetom v rozvrhu, zobrazia sa v komponente *podrobné informácie* podrobnosti o danom predmete.

Iniciálnou podobou tohto komponentu bude týždenný časový harmonogram pedagógov vybraného podpredmetu alebo prázdny rozvrh.

Rozvrh miestností

Komponent *rozvrh miestností* bude obsahovať týždenný časový harmonogram (rozvrh) miestnosti vybranej v komponente miestnosti. Zobrazené nasadené predmety budú graficky rozlíšené podľa typu predmetu.

V dvoch špeciálnych prípadoch popísaných vyššie bude v komponente *rozvrh miestností* zobrazený rozvrh predmetov a rozvrh podpredmetov. Tieto špeciálne prípady

budú graficky odlíšené zmenou titulku komponentu. Zrušenie špeciálnych prípadov bude docielené kliknutím na nejakú miestnosť v komponente *miestnosti*.

V prípade, že sa kurzor myši bude nachádzať nad nejakým nasadeným predmetom v rozvrhu, zobrazia sa v komponente *podrobné informácie* podrobnosti o danom predmete.

Iniciálnou podobou tohto komponentu bude týždenný časový harmonogram vybranej miestnosti alebo prázdny rozvrh.

Nasadenie predmetu do rozvrhu

Pre nasadenie predmetu do rozvrhu bude potrebné:

- Mať v komponente *rozvrh miestnosti* vybraný rozvrh požadovanej miestnosti.
- Mať v komponente *podpredmety* vybraný konkrétny podpredmet, a v zozname jednotlivých nasadení stlačiť tlačidlo (odkaz) „nasadiť predmet“

Samotné nasadenie predmetu prebehne potom kliknutím do daného políčka rozvrhu v komponente *rozvrh miestností*. Následne prebehnú nasledujúce kroky:

- Pomocou *Ajax engine* sa vytvorí požiadavka na server s parametrami požadovaného nasadenia.
- Na strane servera prebehne verifikácia zachovania integrity dát. [kap. 5.12]
- Server vráti výsledok verifikácie, ktorý opäť spracuje *Ajax engine* (JavaScript).
- V prípade úspešného výsledku verifikácie sa používateľovi zobrazí potvrdzovacie okno, či chce požadovaný predmet naozaj uložiť.
- V prípade neúspešného výsledku verifikácie sa zobrazí potvrdzovacie okno, s informáciou o vzniknutej kolízii, a otázkou, či aj napriek tomu predmet nasadiť.
- V prípade potvrdenia sa serveru odošle požiadavka na definitívne nasadenie predmetu, v opačnom prípade sa komponent rozvrh miestností nastaví do svojej iniciálnej podoby.

- V prípade, že nastala chyba na strane servera pri definitívnom ukladaní predmetu, používateľ o tom dostane informáciu v podobe informačného okna (alert) s popisom chyby.

5.11. Činnosť Reporty

V tejto činnosti bude umožnené generovanie jednotlivých reportov. Každý report bude možné vygenerovať alebo vo formáte HTML, alebo vo formáte XML. Obslužný kód na generovanie reportov bude v samostatnom súbore, aby bolo čo najjednoduchšie pridať do zoznamu ďalší report.

Činnosť bude obsahovať na úvod tieto reporty

- Zostava pedagógov s predmetmi, ktoré vyučujú.
- Sumárna zostava pedagógov s počtom vyučovacích hodín (úväzky).
- Zostava nezaradených predmetov
- Zostava predmetov nepriradených rozvrhárom
- Zostava rozvrhárov s predmetmi, ktoré má nasadzovať.

Činnosť bude obsahovať výberové pole so zoznamom reportov, a dve tlačidlá: *generuj html* a *generuj XML*. Po stlačení tlačidla sa príslušný report vygeneruje.

5.12. Validácia a verifikácia integrity dát

V činnostiach na zadávanie vstupných údajov bude prebiehať kontrola vstupných údajov v dvoch krokoch:

- V prvom kroku pri používateľovej požiadavke na uloženie dát pri editácií záznamu alebo pri vytváraní nového záznamu sa ešte pred odoslaním požiadavky na server vykoná pomocou JavaScriptu validácia na neprázdnosť povinných vstupných údajov a v prípade emailovej adresy aj validácia na jej syntakticky správny zápis. V prípade zistenie nesprávne alebo nedostatočne vyplnených údajov sa požiadavka na zápis zmien neodošle na server, ale sa používateľovi zobrazí informačné okno (alert) a je umožnené vkladané údaje opraviť.
- Druhý krok validácie sa vykoná po odoslaní požiadavky na strane servera, kde sa overí, či editovaný záznam ešte stále existuje (mohol ho počas editácie vymazať iný používateľ) a v prípade novovytvoreného záznamu

prebehne kontrola, či sa daný záznam už nenachádza v databáze (napríklad v prípade pedagóga sa kontroluje, či pedagóg s daným menom už v databáze neexistuje). Neúspešný pokus o uloženie dát je užívateľovi oznámený formou informačného okna (alert).

V činnosti na tvorbu rozvrhu prebieha verifikácia na strane servera po odoslaní požiadavky na pridanie predmetu do rozvrhu. Vykoná sa nasledovná kontrola:

- Pedagóg v danom čase nevyučuje iný predmet.
- V pridelenej miestnosti neprebieha v danom čase iná výučba
- Ak predmet obsahuje záznam o povinnej študijnej skupine, vykoná sa kontrola, či v danom čase nemá táto skupina nasadený iný predmet.

V prípade zistenej kolízie sa používateľovi zobrazí potvrdzovacie okno, ktoré ho oboznámi so zistenou kolíziou a spýta sa ho, či aj napriek tomu chce tieto dáta uložiť. Ak užívateľ požiadavku potvrdí, údaje sa zapíšu do databázy.

5.13. Bezpečnosť

Bezpečnosť možno rozdeliť do troch kategórií:

- Bezpečný prístup k databáze
- Bezpečný spôsob komunikácie medzi klientom (internetovým prehliadačom) a databázou
- Autentifikácia prístupu jednotlivých používateľov.

K databáze sa pristupuje len zo strany servera pomocou skriptovacieho jazuka (PHP), ktorý v sebe obsahuje názov databázy a prístupové meno a heslo. Obsah tohto PHP skriptu nie je zvonku viditeľný a pri spustení vracia len vopred definovaný obsah.

Komunikácia medzi klientom a serverom prebieha pomocou Http protokolu. Je preto ľahko „odpočúvateľná“. V závislosti od implementácie ďalších verzií a podpory servera je možné v budúcnosti komunikáciu zabezpečiť pomocou bezpečného Https spojenia.

Ako bolo spomenuté v špecifikácii, [kap. 4.3] v prvej verzii programu nebudú implementované viacpoužívateľské prístupové práva. Návrh však do budúcnosti počíta s ich implementáciou, ktorá bude riešená prideľovaním práv na jednotlivé komponenty, prípadne ich stavy (prezeranie, editácia). Predávanie autentifikačných údajov môže byť

realizované pomocou predávanie session premenných, čo vylučuje neautentifikovaný prístup do systému.

5.14. Ukážka implementácie

V nasledujúcej kapitole popíšem ukážku časti implementácie, konkrétne činnosť *pedagógovia.* [kap. 5.6]

Ako skriptovací jazyk na strane klienta je použitý JavaScript, na strane servera je požitý jazyk PHP.

Kód programu je rozdelený do viacerých súborov:

- `index.php` – hlavný program
- `f_base.php` – spoločné základné funkcie
- `f_sql.php` – funkcie na komunikáciu s databázou
- `pedagog.php` – hlavný program pre činnosť pedagóg
- `pedagogZoznam.php`, `pedagogInfo`, `pedagogPredmet`, `pedagogRozvrh` – obslužné programy pre jednotlivé komponenty.
- `base.css` – definovanie CSS štýlov
- `xmlhttprequest.js` – funkcie na obsluhu XMLHttpRequestu.

Hlavný program index.php

Hlavný program `index.php` obsahuje základné členenie dokumentu na menu a hlavnú časť konkrétnej činnosti. Príslušný hlavný program činnosti, v našom prípade `pedagog.php` sa načíta na základe hodnoty parametra `cinnost=pedagog`, ktorý je vložený do url odkazu pri stlačení konkrétnej položky menu.

```
<body>
  ...
  <!-- horne spolocne menu -->
  <div id="menu">
    ...
    <a href="index.php?cinnost=pedagog">Pedagogovia</a>
    ...
  </div>

  <!-- hlavna cast cinnosti -->
  <div id="main">
      <? require($cinnost.'.php'); ?>
  </div>
  ...
</body>
```

Činnosť pedagógovia – pedagog.php

Obsah hlavnej časti činnosti *pedagógovia* je delený na jednotlivé komponenty. Komponenty sú v dokumente logicky usporiadané pomocou tagov , a každý takýto komponent ma pridelený jednoznačný identifikátor, aby bolo možné s ním pomocou metód DOM manipulovať.

```
<div id="leftBox">
  <!-- zoznam pedagogov  -->
  <div class="boxPopis">Zoznam pedagógov</div> <!-- titulok okna -->
  <div id="pedagogZoznam">
  </div>

  <script type="text/javascript">
    //naplnime komponent pedagogZoznam
    zoznamPedagog();
  </script>

</div>

<div id="rightBox">
  <!-- info o pedagogovi -->
  <div class="boxPopis">Info o Pedagógovi</div> <!-- titulok okna -->
  <div id="pedagogInfo">
  </div>

  <!-- priradene predmety -->
  <div class="boxPopis">Priradené predmety</div> <!-- titulok okna -->
  <div id="pedagogPredmet">
  </div>

  <!-- casovy harmonogram -->
  <div class="boxPopis">Èasový harmonogram</div> <!-- titulok okna -->
  <div id="pedagogRozvrh">
  </div>
</div>
```

Základná štruktúra dokumentu je načítaná formou statického html kódu. Obsah jednotlivých komponentov činností je už však načítavaný dynamicky pomocou *Ajax engine*.

Každý jednotlivý komponent môže existovať vo viacerých stavoch a pre každý stav sú definované JavaScriptové funkcie, ktoré manipulujú s obsahom jednotlivých komponentov, alebo ich častí.

Komponent *Zoznam pedagógov* (`pedagogZoznam`)

- stav: *view*
  - funkcia: *zoznamPedagog* na naplnenie zoznamu pedagógov
  - funkcia *viewPedagog* na načítanie podrobností o vybranom pedagógovi

- funkcia: *newPedagog* na pridanie nového pedagóga

Komponent *Podrobnosti Pedagóga* (`pedagogInfo`)

- stav *view*:
  - funkcia: *editPedagogInfo* na umožnenie editácie podrobností pedagóga
  - funkcia: *deletePedagog* na vymazanie pedagóga
- stav *edit*:
  - funkcia: *updatePedagogInfo* na uloženie zmien podrobností pedagóga
  - funkcia: *cancelPedagogInfo* na vrátenie editovaných zmien.
- stav *new*:
  - funkcia: *insertPedagogInfo* na vloženie nového pedagóga
  - funkcia: *cancelPedagogInfo* na vrátenie editovaných zmien.

Komponent *Priradené predmety* (`pedagogPredmet`)

- stav *view*:
  - funkcia: *viewPedagogPredmet* na zobrazenie priradených predmetov

Komponent *Rozvrh Pedagóga* (`pedagogRozvrh`)

- stav *view*:
  - funkcia: *viewPedagogRozvrh* na zobrazenie rozvrhu pedagóga

Jednoznačná identifikácia objektov v dokumente

V dokumente je potrebná jednoznačná identifikácia jednotlivých objektov. Identifikácia je zabezpečená pomocou atribútu *id* v jednotlivých objektoch (html tagoch).

Pre objekty ktoré sa v dokumente nachádzajú práve raz, ako napr. hlavné komponenty, je obsah atribútu *id* vopred daný.

Pre objekty, ktoré sa v dokumente nachádzajú viackrát, ako napríklad jednotlivé položky zoznamu pedagógov, je pre každú položku na strane servera pomocou PHP (funkcie *unique*) vygenerovaný unikátny obsah jej atribútu *id*. Obsah *id* je potom predávaný aj ako parameter obslužných funkcií (`viewPedagog(this.id,...)`) aby funkcia jednoznačne vedela, s akým objektom dokumentu pracuje.

```
// generovanie na strane servera
$obsah  .= '<div id="'.uniqid('ped',TRUE).'" '
        . 'class="polozka" '
```

```
        . 'onclick="viewPedagog(this.id,'.$riadok["n_pedagog_id"].');" '
...

// vygenerovane html
<div id="ped1254d47a57d57s57ds" clas="polozka"
onclick="viewPedagog(this.id,12);" ...
```

Manipulácia s objektmi

Manipulácia s jednotlivými objektmi prebieha pomocou volania metód DOM v kóde JavaScriptu. Najjednoduchším spôsobom zmeny obsahu objektu je použitie metódy `obj.innerHtml = "nejaky obsah",` ktorá daný objekt naplní priraďovaným obsahom. Pomocou metódy `innerHtml` v programe napĺňam celé bloky hlavných komponentov činnosti. Prvým príkazom do premennej miesto priradíme objekt s `id="pedagogInfo"` (komponent *detaily pedagóga*) a druhým príkazom naplníme tento objekt obsahom vráteným pomocou XMLHttpRequestu.

```
var miesto = document.getElementById("pedagogInfo");
miesto.innerHTML = xmlhttpRequest.responseText;
```

Ďalším spôsob manipulácie s objektmi si ukážeme na príklade dynamického vloženia nového mena pedagóga do komponentu *zoznamu pedagógov*.

```
var tempObj = document.createElement('div');
var tempObsah = '<div id="'+parameter["p_zoznam_id"]+'" '
              + 'class="polozka" '
              + 'onclick="viewPedagog(this.id,'+parameter["p_pedagog_id"]+');"
              + 'onmouseover="this.className = \'polozka select\';" '
              + 'onmouseout="this.className = \'polozka\';"'
              + '>'+decodeURIComponent(parameter["p_pedagog_name"])+'</div>'
             ;
tempObj.innerHTML = tempObsah;
var newObj = tempObj.firstChild;
var obj = document.getElementById('pedagogZoznam');
obj.appendChild(newObj);
```

Na začiatku si vytvoríme nový objekt `tempObj` – html tag `<div>`, a novú textovú premennú `tempObsah`, ktorá bude obsahovať novú položku *zoznamu pedagógov* s menom práve pridaného pedagóga. Pomocou ďalších dvoch príkazov vytvoríme nový objekt `newObj`, ktorý už bude obsahovať vkladanú položku zoznamu už nie ako textovú premennú, ale ako objekt `<div>` html dokumentu. V ďalšom príkaze do premennej obj priradíme komponent *zoznam pedagógov* a príkazom `obj.appendChild(newObj)` novú položku priradíme do dokumentu, ktorá sa tam automaticky zobrazí.

Predávanie parametrov

Predávanie parametrov je riešené tromi spôsobmi.

- Parametre hlavných funkcií volaných priamo z dokumentu sú predávané formou dvojice hodnôt, kde prvá obsahuje *id* aktuálnej položky zoznamu, a druhá obsahuje identifikátor zoznamu v databáze (*n_pedagog_id*). Takto sú napríklad volané funkcie `viewPedagog(zoznamID,pedagogID)` alebo `editPedagogInfo(zoznamID,pedagogID)`
- Parametre následných funkcií volaných z hlavných funkcií si predávajú parametre pomocou asociatívneho poľa. Treba podotknúť, že v JavaScripte sa pole ako parameter funkcie nepredáva priamo, ale len ako odkaz na premennú typu pole, a teda zmena prvku poľa v jednej funkcii ovplyvní jeho hodnotu vo všetkých funkciách, ktoré obsahujú dane pole ako parameter. Takýto spôsob je využitý napríklad v tele funkcie viewPedagog, keď funkcia posiela vstupné parametre ďalším funkciám vo forme asociatívneho poľa.

```
function viewPedagog(zoznamID,pedagogID)
{
  var parameter = new Array();
  parameter["p_zoznam_id"] = [zoznamID];
  parameter["p_pedagog_id"] = [pedagogID];

  viewPedagogInfo(parameter);
  viewPedagogPredmet(parameter);
  viewPedagogRozvrh(parameter);
}
```

- Pri asynchrónnom prenose pomocou XMLHttpRequestu prebieha výmena parametrov medzi prehliadačom a serverom v hlavičke http požiadavky. Funkcia `callRequest` vstupné parametre zaradí do hlavičky požiadavky. PHP skript na strane servera parametre hlavičky http požiadavky dekóduje, a pri vrátení odpovede spätne naplní hlavičku aj s prípadnými novými parametrami. Na strane prehliadača potom pristupuje JavaScript k jednotlivým hodnotám parametrov v hlavičke pomocou príkazu `xmlhttpRequest.getResponseHeader("p_zoznam_id");`

```
// naplnenie http hlavicky na strane prehliadaca
if (parameter) {
  for (var key in parameter) {
    xmlhttpRequest.setRequestHeader(key, parameter[key]);
  }
```

```
}

// získanie parametrov hlavicky na strane servera
$parameter = GetAllHeaders();

// naplnenie http hlavicky na strane servera
while (list($hlavicka, $data) = each($parameter)) {
  Header($hlavicka.": ".$data);
}
```

Asynchrónna komunikácia zo serverom

V tejto kapitole popíšeme vytvorenie a spracovanie XMLHttpRequestu, ktoré obsluhuje funkcia `callRequest`. Funkciu voláme s dvoma parametrami. Prvý parameter je vyššie spomínané asociatívne pole parametrov a druhý parameter `obsluha` ja názov obslužnej funkcie, ktorá obsluhuje výsledok požiadavky vrátený serverom. XMLHttpRequestu je bližšie popísaný úvodnej časti. [kap. 2.6]

```
01  // funkcia zavola xmlhttprequest na zaklade hodnot parametrov,
02  // obsluha = nazov obsluznej funkcie
03  function callRequest(parameter, obsluha)
04  {
05
06    // premenna url povinna
07    if ((!parameter["p_url"])) {
08      return
09    }
10
11    if (window.ActiveXObject)
12    { var xmlhttpRequest = new ActiveXObject("Microsoft.XMLHTTP"); }
13    else
14    { var xmlhttpRequest = new XMLHttpRequest(); }
15
16    if (!xmlhttpRequest) {
17      alert('ERROR:\nNepodarilo sa vytvorit XMLHTTPrequest');
18      return;
19    }
20
21    xmlhttpRequest.open('GET',parameter["p_url"]);
22    xmlhttpRequest.onreadystatechange = function() {
23      if (xmlhttpRequest.readyState == 4)
24      {
25        if(xmlhttpRequest.status == 200)
26        {
27          if (xmlhttpRequest.getResponseHeader("error")) {
28            alert(xmlhttpRequest.getResponseHeader("error"));
29            return;
30          }
31
32          if (!obsluha) {
33            // pokial nie je definovana obsluzna funkcia, zavolame innerHTML
34            if (parameter["p_object_id"]) {
35             var miesto = document.getElementById(parameter["p_object_id"]);
36              miesto.innerHTML = xmlhttpRequest.responseText;
```

```
37            }
38          }
39          else {
40            // ak je definovana obsluzna funkcia, zavolame ju
41            obsluha(xmlhttpRequest,parameter);
42          }
43        }
44        else
45        {
46          alert("ERROR: Chyba pri nacitani stanky"
47              + xmlhttpRequest.status + ": "
48              + xmlhttpRequest.statusText);
49          return;
50        }
51      }
52
53    };
54
55    if (parameter) {
56      for (var key in parameter) {
57        xmlhttpRequest.setRequestHeader(key, parameter[key]);
58      }
59    }
60    xmlhttpRequest.send(null);
61
62  // end function callRequest
63  }
```

Na riadku 06 overujeme, či je definovaný parameter *url.*

Na riadku 11 vytvárame objekt XMLHttpRequest.

Na riadku 21 mu priradíme metódu volania GET, url a nastavíme asynchrónne spracovanie parametrom *true*.

Na riadku 22 definujeme obslužnú funkciu, ktorá bude zavolaná vždy pri zmene stavu požiadavky.

Ak je požiadavka spracovaná a server vrátil výsledok, môžeme prejsť k jeho spracovaniu (riadok 26).

Na riadku 27 skontrolujeme, či sme na strane servera nenaplnili hlavičku parametrom *error*, ktorý signalizuje neúspešne spracovanie požiadavky (napríklad sa nepodarilo spojiť s databázou...).

Na riadku 32 skontrolujeme, či je definovaná obslužná funkcia. Ak nie, obsahom vrátenej požiadavky naplníme obsah komponentu s *id* definovaným v parametri *p_object_id.* Toto je základný spôsob napĺňania obsahu jednotlivých komponentov.

Ak je obslužná funkcia definovaná, zavoláme ju (riadok 41) a ako parameter jej prirad9me celý XMLHttpRequest pre ďalšie spracovanie.

Na riadku 46 obsluhujeme prípadné zlyhanie spracovania požiadavky.

Na riadku 56 hlavičku požiadavky vstupnými parametrami.

Na riadku 60 požiadavku odošleme na server.

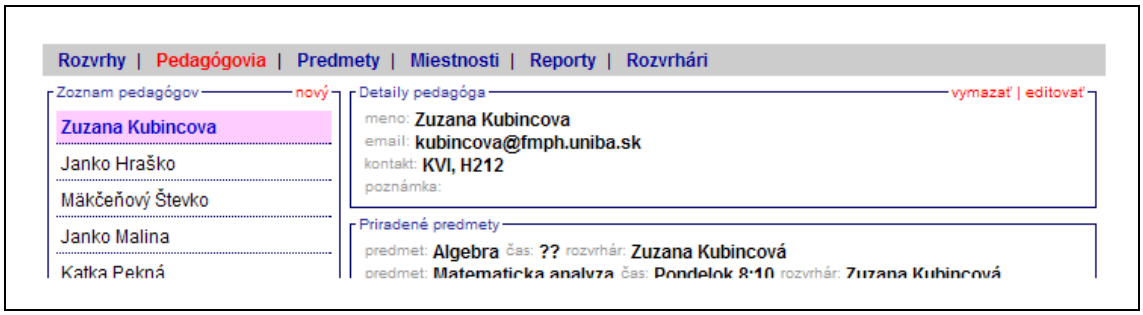
Popis funkcionality činnosti

V tejto kapitole popíšeme funkcionalitu činnosti. Nebudeme podrobne vysvetľovať jednotlivé príkazy funkcií, ale len ich hrubú ideu.

Po stlačení tlačidla *Pedagógovia* v menu aplikácie, sa načítajú prázdne komponenty *Zoznam pedagógov, Detaily pedagóga, Priradené predmety a Rozvrh pedagóga.* Po načítaní stránky sa zavolá funkcia *ZoznamPedagog*, ktorá naplní zoznam pedagógov z databázy.

Po kliknutí na konkrétneho pedagóga v zozname sa zavolá funkcia *viewPedagog,* ktorá naplní pole parametrov a zavolá funkcie *viewPedagogInfo, viewPedagogPredmet a viewPedagogRozvrh,* ktoré naplnia jednotlivé komponenty podrobnosťami o zvolenom pedagógovi.

Po kliknutí na tlačidlo *nový* sa zavolá funkcia *newPedagog*, ktorá naplní komponent *Detaily pedagóga* editovateľnými položkami na zadávanie nového pedagóga.



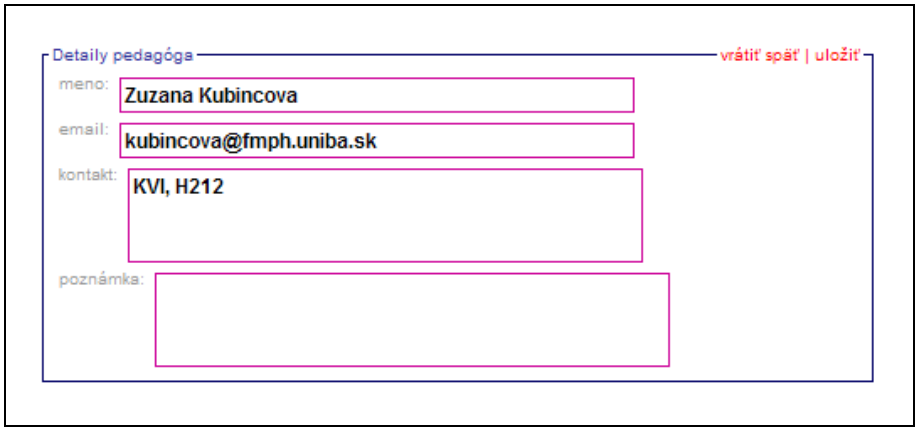
Obr. 21. Obrazovka činnosti Pedagógovia

V komponente *Detaily pedagóga* sa nachádzajú tlačidlá *zmazať* a *editovať*.

Po stlačení tlačidla *zmazať* je používateľ vyzvaný k potvrdeniu svojej požiadavky a následne je pedagóg pomocou funkcie *deletePedagog* z databázy vymazaný.

Po stlačení tlačidla editovať sa komponent *Detaily pedagóga* zmení pomocou funkcie *editPedagogInfo* do stavu *editácia*. V tomto stave je umožnené editovať jednotlivé položky. Z tohto stavu je možné tlačidlom *vrátiť zmeny* (funkcia

cancelPedagogInfo) načítať pôvodné hodnoty pedagóga bez uloženia vykonaných zmien, alebo pomocou tlačidla *uložiť* (funkcia *updatePedagogInfo*) vykonané zmeny uložiť.



Obr. 22. Obrazovka komponentu Detaily pedagóga v stave editácia

Zobrazenie obsahu komponentov Priradené predmety a Rozvrh pedagóga sa načítava pomocou funkcií *viewPedagogPredmet* a *viewPedagogRozvrh*.

Zvlášť by som ešte spomenul funkcie *insertPedagogInfo*, *updatePedagogInfo*, a *deletePedagogInfo* ktoré ako jediné využívajú volanie funkcie *callRequest* s vlastnou funkciou na obsluhu vrátenej http požiadavky. Podrobne popíšem jednu z nich: *updatePedagogInfo*.

```
01  // udpdate pedagoga v databaze
02  function updatePedagogInfo(zoznamID,pedagogID)
03  {
04    var parameter = new Array();
05    parameter["p_zoznam_id"] = [zoznamID];
06    parameter["p_pedagog_id"] = [pedagogID];
07    parameter["p_object_id"] = ["pedagogInfo"];
08    parameter["p_url"] = ["pedagogInfo.php"];
09    parameter["p_action"] = ["update"];
10
11    parameter["p_pedagog_name"] =
[encodeURIComponent(document.getElementsByName('pedagogInfo_name')[0].value)];
12    parameter["p_pedagog_email"] =
[encodeURIComponent(document.getElementsByName('pedagogInfo_email')[0].value)]
;
13    parameter["p_pedagog_contact"] =
[encodeURIComponent(document.getElementsByName('pedagogInfo_contact')[0].value
)];
14    parameter["p_pedagog_note"] =
[encodeURIComponent(document.getElementsByName('pedagogInfo_note')[0].value)];
15
16    callRequest(parameter,updatePedagogInfoObsluha);
17  }
```

Na začiatku naplníme pole parametrov.

- *p_zoznam_id* je *id* položky zvoleného pedagóga v zozname pedagógov.
- *p_pedagog_id* je identifikátor pedagóga v databáze (*n_pedagog_id*).
- *p_object_id* určuje, komponentom *pedagogInfo* (*detaily pedagóga*).
- *p_url* je názov obslužného PHP skriptu.
- *p_action* je druh vykonávanej akcie – *update* pedagóga v databáze.
- Parametre *p_pedagog_name, p_padagog_email, p_pedagog_contact, p_pedagog_note* obsahujú hodnoty položiek pedagóga, ktoré sme editovali.

Následne zavoláme funkciu *callRequest* s definovanou obslužnou funkciou `updatePedagogInfoObsluha`.

```
01  function updatePedagogInfoObsluha(result, parameter)
02  {
03    // updatneme meno v zozname pedagogov
04    var obj = document.getElementById(parameter["p_zoznam_id"]).firstChild;
05    obj.nodeValue = decodeURIComponent(parameter["p_pedagog_name"]);
06
07    // nacitam pegagogInfo
08    viewPedagogInfo(parameter);
09  }
```

Na riadku 04 do premennej `obj` priradíme objekt reprezentujúci obsah položky zvoleného pedagóga v *zozname pedagógov* (meno pedagóga) a na riadku 05 prepíšeme túto hodnotu nami zmenenou hodnotou pri editácii. Týmto sme docielili vizuálnu zmenu jednej položky zoznamu pedagógov bez potreby načítavať celý zoznam.

6. Záver

Vytvorenie diplomovej práce bolo pre mňa obohatením, zopakovaním teoretických postupov vytvárania odbornej práce a vyskúšaním jej praktickej realizácie. Takisto táto diplomová práca preverila praktickú aplikáciu teoretických poznatkov, ktoré som získal počas výučby na FMFI UK ohľadom analýzy, špecifikácie a návrhu softvérového projektu.

Podarilo sa mi s úspechom splniť ciele diplomovej práce vytýčené v úvode, a to podrobne popísať technológiu Ajax a navrhnúť konkrétny softvérový produkt využívajúci technológiu Ajax.

Vytvorený návrh riešenia webovej aplikácie na tvorbu fakultného rozvrhu chcem po obhajobe tejto diplomovej práce implementovať do reálnej podoby, ktorá by úspešne nahradila doterajší ručný spôsob tvorby fakultného rozvrhu.

Ukážka čiastočnej implementácie v záverečnej kapitole prakticky demonštruje využitie technológie Ajax v navrhovanej webovej aplikácii.

Verím, že výsledný softvérový produkt raz bude nasadený do praxe na FMFI a bude poskytovať komplexnejšiu a užívateľsky príjemnejšiu alternatívu k terajšiemu stavu.

Dosiahnuté praktické skúsenosti pri vytváraní tejto diplomovej práce sú pre mňa dobrou devízou do budúceho profesionálneho života.

7. Použitá literatúra a zdroje

1. LETKOVSKÝ, Alexander. *ThinkFree – kompletný kancelársky balík v internetovom prehliadači* [online]. 2006. Dostupné na internete: <http://www.itnews.sk/buxus_dev/generate_page.php?page_id=40697> [2006-05-01].
2. *Google suggest* [online]. Dostupné na internete: <http://labs.google.com/suggest>. [2006-05-01].
3. *Google email* [online]. Dostupné na internete: <http://mail.google.com>. [2006-05-01].
4. *Google maps* [online]. Dostupné na internete: <http://maps.google.com>. [2006-05-01].
5. *Mapy atlas.sk* [online]. Dostupné na internete: <http://mapy.atlas.sk>. [2006-05-01].
6. *Writley - online office* [online]. Dostupné na internete: <http://www.writely.com>. [2006-05-01].
7. *Document Object Model (DOM)* [online]. Dostupné na internete: <http://www.w3.org/DOM/>. [2006-05-01].
8. *HAVEL, Jakub. DOM – objektový model dokumentu* [online]. 2002. Dostupné na internete: <http://www.zive.cz/h/Programovani/AR.asp?ARI=103999>. [2006-05-01].
9. GARRETT, Jesse James. *Ajax: A New Approach to Web Applications* [online]. 2005. Dostupné na internete: <http://adaptivepath.com/publications/essays/archives/000385.php> [2006-05-01].
10. VRÁNA, Jakub. *AJAX* [online]. 2005. Dostupné na internete: <http://www.root.cz/clanky/ajax/> [2006-05-01].
11. ZRALÝ, Jíří. *AJAX - návod pro začátečníky* [online]. 2005. Dostupné na internete: <http://citron.blueboard.cz/clanek-239-ajax-navod-pro-zacatecniky.html> [2006-05-01].
12. KOZO. *Svet AJAX-u má perspektívu* [online]. 2005. Dostupné na internete: <http://kozo.bloguje.cz/204216_item.php> [2006-05-01].

13. COACH, K. Wei. *AJAX: Asynchronous Java + XML?* [online]. Dostupné na internete: <http://www.developer.com/design/article.php/10925_3526681_1> [2006-05-01].
14. Neznámy autor. *Ajax: Places To Use Ajax?* [online]. 2005. Dostupné na internete: <http://swik.net/Ajax/Places+To+Use+Ajax> [2006-05-01].
15. JANOVSKÝ, Dušan. *CSS – Kaskádové styly* [online]. 2004. Dostupné na internete: <http://www.jakpsatweb.cz/css/> [2006-05-01].
16. KOSEK, Jiří. *Tvorba interaktívnich internetových aplikací*. Grada, 1999.
17. OLŠAVSKÝ, Marek. *Malý přehled databází* [online]. 2004. Dostupné na internete: < http://www.linuxsoft.cz/article.php?id_article=304> [2006-05-01]

7.1. Zoznam obrázkov

Obr. 1. Ukážka Google našeptávača 8

Obr. 2. Porovnanie klasického a Ajax web aplikačný modelu [9] 10

Obr. 3. Klasický web aplikačný model – synchrónna komunikácia [13] 11

Obr. 4. Ajax web aplikačný model – asynchrónna komunikácia [13] 11

Obr. 5. Obrázok jednoduchého modelu rozvrhu. 28

Obr. 6. Obrázok rozšíreného fakultného modelu rozvrhu 29

Obr. 7. Tabuľka pedagog 40

Obr. 8. Tabuľky predmet a podpredmet 41

Obr. 9. Tabuľka miestnost 41

Obr. 10. Tabuľka rozvrhar 42

Obr. 11. Tabuľka rozvrh 43

Obr. 12. Rozloženie komponentov činnosti predmet 46

Obr. 13. Komponent Zoznam pedagógov 47

Obr. 14. Komponent Podrobnosti pedagóga v stave prezeranie a editácia. 47

Obr. 15. Komponent Priradené predmety pedagóga 49

Obr. 16. Komponent Rozvrh pedagóga 49

Obr. 17. Rozloženie komponentov činnosti predmet 50

Obr. 18. Komponent Podrobnosti predmetu v stave prezeranie a editácia. 51

Obr. 19. Komponent Podpredmet v stave prezeranie a editácia. 53

Obr. 20. Rozloženie komponentov činnosti rozvrh 58

Obr. 21. Obrazovka činnosti Pedagógovia 71

Obr. 22. Obrazovka komponentu Detaily pedagóga v stave editácia 72