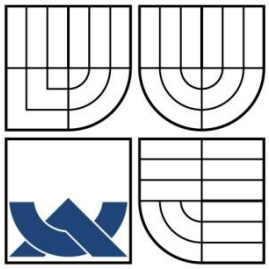
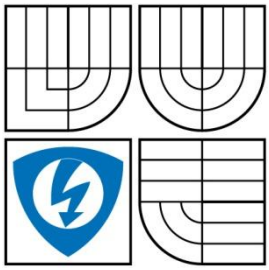


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

# PLÁNOVAČ IPTV VYSÍLÁNÍ S VYUŽITÍM TOMCAT A FRAMEWORKŮ STRIPES A HIBERNATE

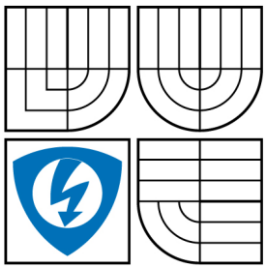
IPTV TRANSMISSION PLANNER USING TOMCAT AND STRIPES AND HIBERNATE

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — PAVEL VAJSAR
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADIM BURGET
SUPERVISOR

BRNO 2008



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**Fakulta elektrotechniky a komunikačních technologií**

**Ústav telekomunikací**

# Bakalářská práce

bakalářský studijní obor
**Teleinformatika**

***Student:*** Vajsar Pavel ***ID:*** 74901
***Ročník:*** 3 ***Akademický rok:*** 2007/2008

**NÁZEV TÉMATU:**

## Plánovač IPTV vysílání s využitím tomcat a frameworků stripes a hibernate

**POKYNY PRO VYPRACOVÁNÍ:**

Seznamte se s tomcat, subversion a rámci stripes a hibernate. Proveďte analýzu aplikace pro plánování IPTV vysílání. Analýzu detailně popište pomocí UML diagramu užití a UML ER-diagramu. V prostředí jazyka JAVA implementujte. Blíže se věnujte otázkám bezpečnosti, spouštění naplánovaných událostí a uploadování souborů vetší velikosti.

**DOPORUČENÁ LITERATURA:**

[1] HORTON, I., Beginning Java 2, Wrox, 2002
[2] BAUER C., King, G., Java Persistence with Hibernate, Manning Publications, 2006
[3] CHOOPRA, V., BAKORE, A., AEVES, J. GALBRAIHT, B., Professional Apache Tomcat 5, Wrox, 2004

***Termín zadání:*** 11.2.2008 ***Termín odevzdání:*** 4.6.2008

***Vedoucí práce:*** Ing. Radim Burget

**prof. Ing. Kamil Vrba, CSc.**
*předseda oborové rady*

**UPOZORNĚNÍ:**

# LICENČNÍ SMLOUVA
# POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

**1. Pan/paní**

Jméno a příjmení: Pavel Vajsar

Bytem: Kondrac 112, 258 01 Vlašim

Narozen/a (datum a místo): 20.03.1986, Benešov

(dále jen „autor“)

a

**2. Vysoké učení technické v Brně**

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

prof. Ing. Kamil Vrba, CSc.

(dále jen „nabyvatel“)

## Čl. 1

## Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

| | |
|---|---|
| Název VŠKP: | Plánovač IPTV vysílání s využitím tomcat a frameworků stripes a hibernate |
| Vedoucí/ školitel VŠKP: | Ing. Radim Burget |
| Ústav: | Ústav telekomunikací |
| Datum obhajoby VŠKP: | ……... |

VŠKP odevzdal autor nabyvateli v*:

- □ tištěné formě – počet exemplářů 1
- □ elektronické formě – počet exemplářů 1

*hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

## Článek 2
## Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - □ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

## Článek 3
## Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 4.6.2008

…….. ……

Nabyvatel Autor

## Anotace

Bakalářská práce se zabývá návrhem webové aplikace pro plánování IPTV vysílání – IPTV klient. Konkrétní vysílání pomocí protokolů RTP a RTCP již není její součástí. Prostřednictvím aplikace lze po přihlášení nahrávat na server pořady (video soubory), spravovat uživatele aplikace, vytvářet feedbacky a nové role. První část bakalářské práce se zabývá analytickým návrhem aplikace – model databáze, diagram případů užití a diagram aktivit. V další části se věnuje použitým softwarovým prostředkům, návrhem uživatelského rozhraní a výměně dat mezi aplikací a databází. Dalšími oblastmi, které jsou podrobněji rozebrány a popsány jsou bezpečnost aplikace, rozlišení rolí uživatelů a upload souborů větší velikosti na server.

## Klíčová slova

Tomcat, Java, Stripes, Hibernate, UML, Java, Entity Relationship Diagram

## Abstract

Bachelor´s thesis deals with concept of web application for planning IPTV broadcast - IPTV client. The specific broadcast with protocols RTP and RTCP is not included.
After logging on you are able to record programs (video files), manage users, create feedbacks and create new roles as well with this application. First part of this work considers with analytical concept of application - the database model, diagram of cases of use and activity diagram. The next part deals with using software resources, user interface design and data exchange between the application and the database. Other areas, which are better described in this project, are for example safety of application, role separation and upload files with bigger size on the server.

## Keywords

Tomcat, Java, Stripes, Hibernate, UML, Java, Entity Relationship Diagram

## Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Plánovač IPTV vysílání s využitím tomcat a frameworků stripes a hibernace jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení §11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplívajících z ustanovení §152 trestního zákona č. 140/1961 Sb.

V Brně dne 4.6.2008 ……..

podpis autora

## Poděkování

Děkuji vedoucímu bakalářské práce Ing. Radimu Burgetovi, za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne 4.6.2008 ……..

podpis autora

# OBSAH

# Úvod

Bakalářská práce se zabývá návrhem webové aplikace pro plánování IPTV vysílání. Tvorba celé aplikace je rozdělena mezi několik studentů. Úkolem této práce je věnovat se návrhu objektového modelu pomocí jazyka UML, návrhu uživatelského rozhraní, správě uživatelů a jejich rolí. Bližší popis je věnován zabezpečení aplikace a uploadu souboru větší velikosti. Objednávání dostupných pořadů a plánování jejich spuštění není již součástí práce. Tato oblast přísluší kolegovi, který se má touto otázkou zabývat, jelikož práce byla rozdělena mezi více studentů.

První kapitola bakalářské práce je věnována struktuře IPTV vysílání. Službám, které nabízí a službám které by v budoucnu nabízet mohla. Dále je zde nastíněna technická realizace IPTV a stručný popis protokolů, které jsou nedílnou součástí technologie. Druhá kapitola je již věnována zadání bakalářské práce a je zde uveden popis softwarových prostředků pro implementaci aplikace, jako je popis platformy Java a jejích předností, použité vývojové prostředí a nástroj pro spolupráci více lidí na jednom projektu. Dále je blíže popsán webový server Tomcat, na kterém je aplikace provozována. Další část práce obsahuje grafický popis aplikace z pohledu případů užití a aktivit. Tento návrh je realizován pomocí jazyka UML (Unified Modeling Language), který je právě pro tyto účely vytvořený. Dále je uveden model databáze obsahující jednotlivé entity (tabulky) a sloupce, které jsou pro správnou funkci aplikace nezbytné. Pro lepší orientaci při čtení následující části bakalářské práce je ve čtvrté kapitole uvedena adresářová struktura aplikace s popisem důležitých adresářů a souborů.

Po spíše teoretickém pojetí práce je od páté kapitoly popsané konkrétní praktické řešení, které je rozděleno do několika dílčích kroků (několika kapitol). Přímo pátá kapitola se věnuje návrhu uživatelského rozhraní aplikace, popisu validace dat s využitím frameworku stripes a popisuje, co znamená ActionBean a k čemu je využit. Dalším krokem při realizaci aplikace bylo vytvoření vrstvy model a manager. Vrstva model obsahuje perzistentní objekty, které jsou využity pro práci s daty mezi aplikací a databází. Ve vrstvě manager jsou poté definovány metody pro výměnu dat mezi databází a perzistentními objekty. Problematice těchto dvou vrstev je věnována sedmá kapitola. Závěr bakalářské práce se detailněji zabývá problematikou spojenou se zabezpečením aplikace a nahráváním souboru větší velikosti na server.

# 1 IPTV

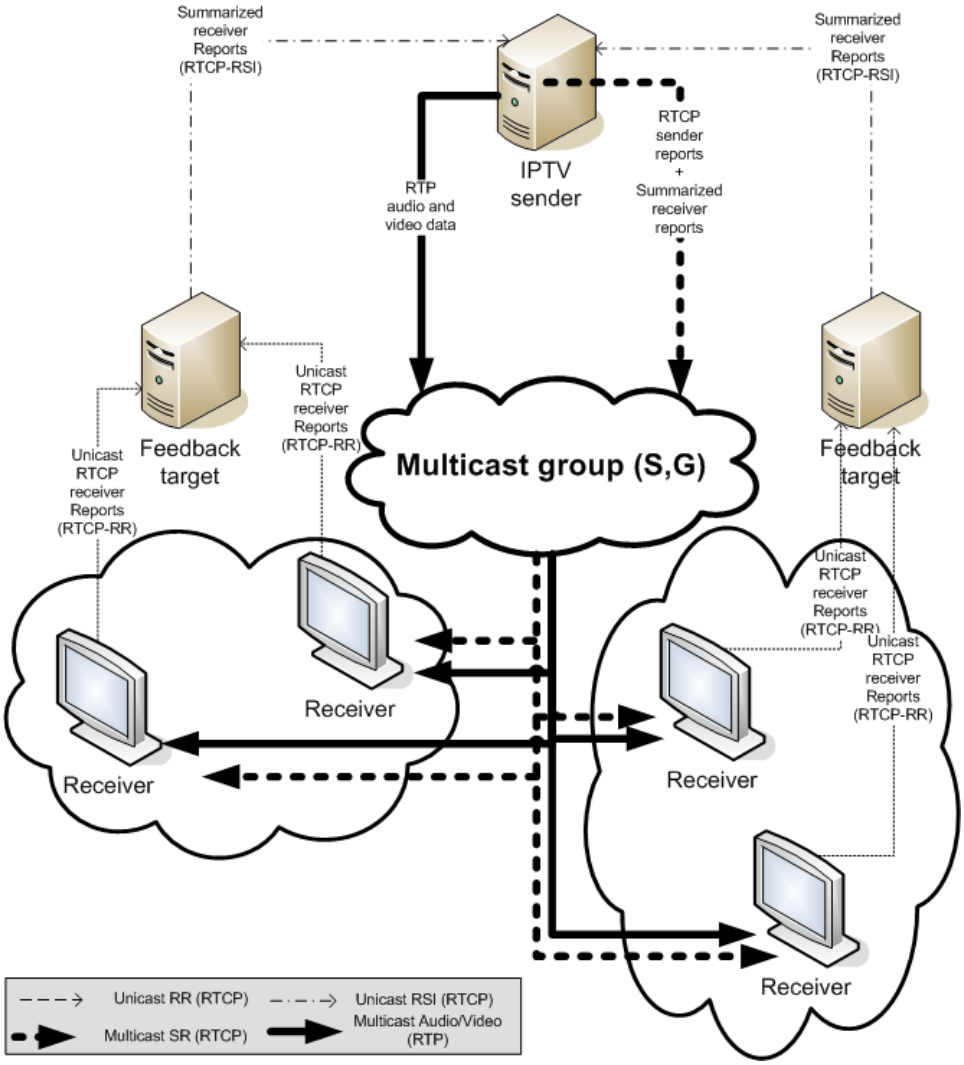
IPTV je televizní vysílání šířené prostřednictvím privátní IP sítě. Jedná se o streamovaný tok videa v digitální podobě, proto je pro IPTV vyžadována vysokorychlostní přípojka. Vzhledem k tomu, že přenos IPTV probíhá po síti, tak existuje mezi vysílačem a příjemcem služby vzájemná konektivita. To dává prostor pro rozšiřování služeb oproti klasickému pozemnímu vysílání. Mezi tyto služby lze zařadit elektronického programového průvodce (EPG), video na přání (VoD), televizní archiv (VCR), sdílení fotografií a videa (My Own TV). Nabídka služeb se postupem času bude rozšiřovat. Již v současné době se mluví o televizním bankovnictví, přijímaní emailů a různých formách textové komunikace (chat) [1]. Na rozdíl od klasického televizního vysílaní neputují k přijímací straně všechny dostupné programy, ale pouze jeden (přijímací strana vybere, který chce sledovat). V případě změny programu musí přijímací část vyslat požadavek na změnu programu vysílací části.

Vlastní přenos IPTV vysílání je realizován pomocí protokolů RTP (Real Time Transport Protocol) a RTCP (RTP Control Protocol ). RTP protokol slouží pro přenos dat v reálném čase a klade důraz na co nejmenší zpoždění. Je založen na UDP protokolu, což je protokol zajišťující nespolehlivý přenos dat, tudíž nedochází ke zpoždění v důsledku potvrzovaní a kontroly dat. Protokol RTP zjistí typ dat, připraví data pro přenos a odešle, už ale nezaručuje správné pořadí přijatých paketů ani to, zda byly přijaty všechny pakety [2]. Spojení mezi vysílačem a přijímačem je založené na páru portů. Jednou cestou se posílají RTP pakety, což jsou multimediální data a zbývající druhou cestou putují RTCP pakety. RTCP pakety u přenosu zajišťují kontrolu přenesených dat a komunikaci mezi účastníky přenosu. RTCP pakety vysílají oba účastnící přenosu, jsou to kontrolní a řídící pakety přenosu. Účastníci si prostřednictvím RTCP sdělují např. nedoručená data, poškozená data, množství přenesených dat. Protokol RTCP umožňuje i řízení rychlosti přenosu v případě zahlcení příjemce.

IPTV vysílání může být vysíláno na multicastovou (skupina příjemců), nebo unicastovou adresu (jeden konkrétní příjemce). Výhodnější z hlediska zatížení vysílače, je rozesílat IPTV vysílání na multicastovou adresu. Vysílač IPTV vyšle data pouze jednou a pak sama přenosová síť na základě multicastové adresy stanic určí, kam budou data rozeslána. Konkrétní rozesílaní dat pomocí multicastingu řeší protokol IGMP [3]. V případě vysílání na unicastovou adresu vysílač vysílá ke každé stanici zvlášť a tím dochází k jeho velkému zatížení.

Jedno z konkrétních řešení IPTV je vidět na obr. 1.1. Zdroj vysílání (IPTV sender) rozesílá vysílání multicastové skupině (RTP protokol) spolu s řídícími informacemi (RTCP protokol). Feedback targety zde slouží jako zpětná vazba od příjemců (receivers) směrem k vysílači

IPTV (sender) a zrychlují tím komunikaci v rámci sítě. Jestliže by tyto feedback targety v síti neexistovaly, tak by se každý příjemce IPTV vysílání musel obracet s RTCP daty přímo na vysílač. Tím by se v síti výrazně zvýšilo zpoždění a zatížení IPTV vysílače. Fedback target existuje pro jednotlivé skupiny uživatelů, od kterých přebírá tzv. RTCP Reports prostřednictvím unicast kanálu a na jejich základě zjišťuje různé parametry sítě. Zjištěné parametry jsou předány IPTV vysílači, který podle výsledků těchto parametrů může přizpůsobit vysílání (změnit kvalitu, změnit šířku pásma pro přenášená data). Více o této struktuře a její implementaci se lze dozvědět na stránkách fakultního výzkumného projektu [4].




*Obr. 1.1 Struktura IPTV – Stupňovitá struktura (převzato se souhlasem autora)*

# 2 Použité nástroje pro tvorbu a běh aplikace

Prvním krokem při tvorbě aplikace bylo seznámit se s použitými nástroji pro vývoj, tyto nástroje získat z internetu, nainstalovat a případně nakonfigurovat. Vzhledem k tomu, že jsou všechny nástroje, které jsou pro vývoj aplikace potřebné, šířeny zdarma, nebyl proto žádný problém s licencemi ani autorskými právy.

## 2.1 Java

Jelikož je aplikace tvořena s využitím frameworků stripes a hibernate, které právě z jazyka Java vycházejí, je tedy Java základním kamenem a téměř nejdůležitějším nástrojem.

Java existuje v několika distribucích – pouze jako běhová (JRE), která slouží pro běh aplikací napsaných v Javě a dále jako vývojová (JDK), která obsahuje běhové i vývojové prostředí. V našem případě bude samozřejmě potřeba vývojová distribuce – JDK, kterou lze stáhnout z webových stránek společnosti SUN Microsystems, která stojí za vývojem Javy.

Java je moderní objektově orientovaný programovací jazyk. Má představovat nástupce jazyka C/C++. Oproti jazyku C/C++ je srozumitelnější, lépe naučitelná a odstraňuje některé jeho nedostatky, jako je například práce s ukazateli, které Java nepodporuje. Rozdílů mezi Javou a C++ je mnoho a některé z nich jsou uvedeny zde [5]. Java je jazyk interpretovaný. Při překladu nedochází k vytvoření strojového kódu ale tzv. bajtkódu, označovaného jako mezikód. Pro běh bajtkódu je potřeba virtuální stroj Javy - Java Vitrual Machine (JVM). Mezi největší výhody jazyka patří jeho přenositelnost mezi platformami (MS Windows, Linux, Mac OS, Solaris atd.). Jednou napsaný a přeložený program v Javě lze spustit na každé z těchto platforem. Samotná Java platforma se skládá ze tří částí.

- ***Java virtual Machine (JVM)*** – zajišťuje běh přeložené aplikace
- ***Debugger , překladač a nástroj pro generování dokumentace***
- ***Java Core API*** – knihovna tříd Javy

Důležité nástroje platformy Java se nacházejí ve složce /bin a lze mezi ně zařadit:

- **java** (běhové prostředí Javy)
- **javac** (překladač, nejjednodušší použití je uvést javac + název souboru)
- **jdb** (debugger neboli program na ladění aplikace)
- **jar** (správce archivu, obsahuje přeložené soubory aplikace)

Informace a dalších nástrojích lze získat z oficiální dokumentace k Javě, která je svým obsahem rozsáhlá [6]. Dále je možné spoustu informací nalézt na internetových stránkách zabývajících se programováním v jazyce Java.

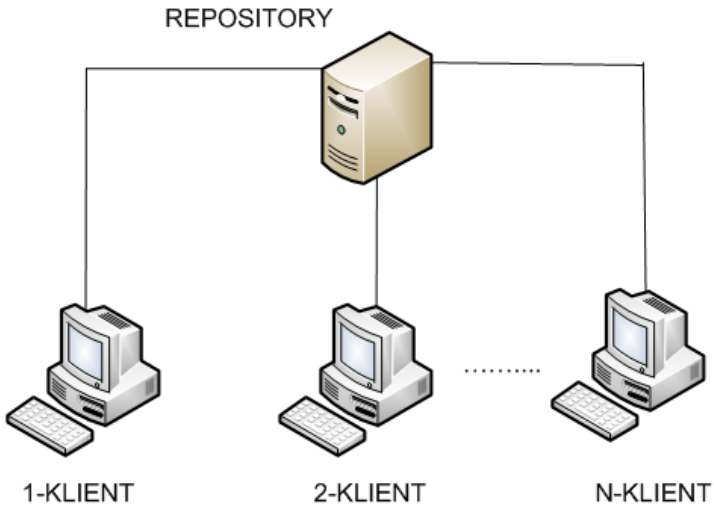
## 2.2 Eclipse a Subversion

Po instalaci Javy můžeme přejít k dalšímu nástroji a to je Eclipse [7]. Eclipse, je vývojové prostředí primárně určené pro Javu, ale po doinstalování rozšiřujících pluginů v něm lze psát i aplikace např. v jazyce C.

Důležitým pluginem pro prostředí Eclipse a naší práci je Subclipse [8]. Po instalaci Subclipse získáme rozšíření prostřednictvím kterého se lze připojit ke vzdálenému systému Subversion [9], zkráceně SVN. Subversion je nástroj pro správu zdrojových kódů aplikace při spolupráci více vývojářů současně. Nahrazuje a zároveň vylepšuje systém CVS [10] (Concurrent Version System), který je předchůdcem SVN. Oba tyto systémy patří do skupiny SCM (Software configuration management), což jsou systémy řešící spolupráci více lidí na jednom projektu. SVN je šířeno pod bezplatnou licencí i pro komerční využití. Tato licence se ale nevztahuje na doplňující knihovny od jiných distributorů.

Zdrojové kódy projektu jsou uloženy v tzv. Repository, což je místo v síti, ke kterému se lze vzdáleně připojit obr. 2.1. V Repository jsou uchovávány všechny pracovní kopie (všechny změny v projektu). Každá změna provedená v projektu se nazývá revize (revision) a udává pořadové číslo každé změny. Pracovní kopie umožňují použití starší verze v případě výskytu vážného problému v poslední verzi projektu. Nejběžnější pracovní postup s SVN je ten, že si klient stáhne aktuální verzi projetu (checkout), provede v této kopii změny a nahraje je zpět do Reposiry (commit). Uživatel dále může svůj projekt aktualizovat (Update). Tím se mu do jeho pracovní kopie nahrají změny, které byly uskutečněny od posledního stažení projektu. Dále SVN umožňuje

získání informace a rozdílu mezi pracovní kopií a repository (diff). Je též možné z pracovní kopie do repository (z repository do pracovní kopie) posílat pouze balík změn. SVN uloží pouze informace o změnách (changeset).




*Obr. 2.1 Základní struktura systému SVN*

## 2.3 Apache Tomcat

Apache Tomcat je webový server podporující technologie Java Servlet a Java Server Pages. Bývá označován jako servlet container. Je vydán pod licencí Apache Software License a na jeho vývoji se podíleli vývojáři s celého světa. V současné době je k dispozici verze 6.0.14, která podporuje Java Servlets 2.5 a Java Server Pages 2.1. Apache Tomcat existuje ve verzi pro operační systém Microsoft Windows i pro operační systémy Unixového typu. Před vlastní instalací Apache Tomcat je nutné nejprve nainstalovat podporu Javy (JDK). Postup instalace je poté pro každý systém odlišný. Adresářová struktura a způsob použití Apache Tomcatu zůstali shodné pro oba operační systémy. Ve struktuře Apache Tomcat má každý adresář a podadresář konkrétní význam. Při implicitním nastavení je služba Apache Tomcat dostupná na localhost, což je adresa 127.0.0.1 a na portu 8080. Pro přístup ke službě je dobré zadávat IP adresu spolu s číslem portu (http://127.0.0.1:8080/). Jestliže chceme využít spojení se šifrovaným přenosem dat, je nutné na serveru tuto službu aktivovat a nastavit. Implicitní port pro přístup ke službě pomocí šifrovaného spojení je 8443 (https://127.0.0.1:8443/). Následuje popis nejdůležitějších adresářů Apache Tomcatu.

**/bin** – obsahuje soubory pro spuštění a zastavení Apache Tomcatu formát těchto souborů závisí na použitém operačním systému. Ve Windows jsou to soubory s příponou *.exe* a v Unixu spustitelné skripty s příponou *.sh*

**/conf -** zde jsou umístěny soubory pro konfiguraci Apache Tomcatu z nichž nejdůležitější je *server.xml*, který slouží pro základní konfiguraci serveru. Dále adresář obsahuje soubory pro konfiguraci servlet containeru a jednotlivých uživatelů.

**/lib –** obsahuje *.jar* knihovny serveru

**/logs –** log soubory serveru. Lze v nich nalézt dostatek informací o serveru a případných chybách vzniklých při startu, nebo za běhu serveru.

**/webapps –** adresář obsahující jednotlivé webové aplikace běžící na Apache Tomcat serveru. Každý adresář webové aplikace musí mít přesně definovanou strukturu tzv. Standard Directory Layout [11]. V tomto adresáři je umístěna aplikace, která je tvořena v rámci bakalářské práce.

# 3 Objektový model aplikace

Při tvorbě objektového modelu aplikace se vycházelo z toho, co všechno by aplikace měla nabízet. Představuje grafický pohled na aplikaci srozumitelný pro laika. Pro programátora pak šablonu, podle které bude aplikaci tvořit. Pro tvorbu objektového modelu byl využit jazyk UML. V našem případě byl z dostupných UML diagramů použit diagram případů užití a diagram aktivit. Více informací o jazyku UML a dostupných diagramech se lze dozvědět na oficiálních webových stránkách projektu [12].

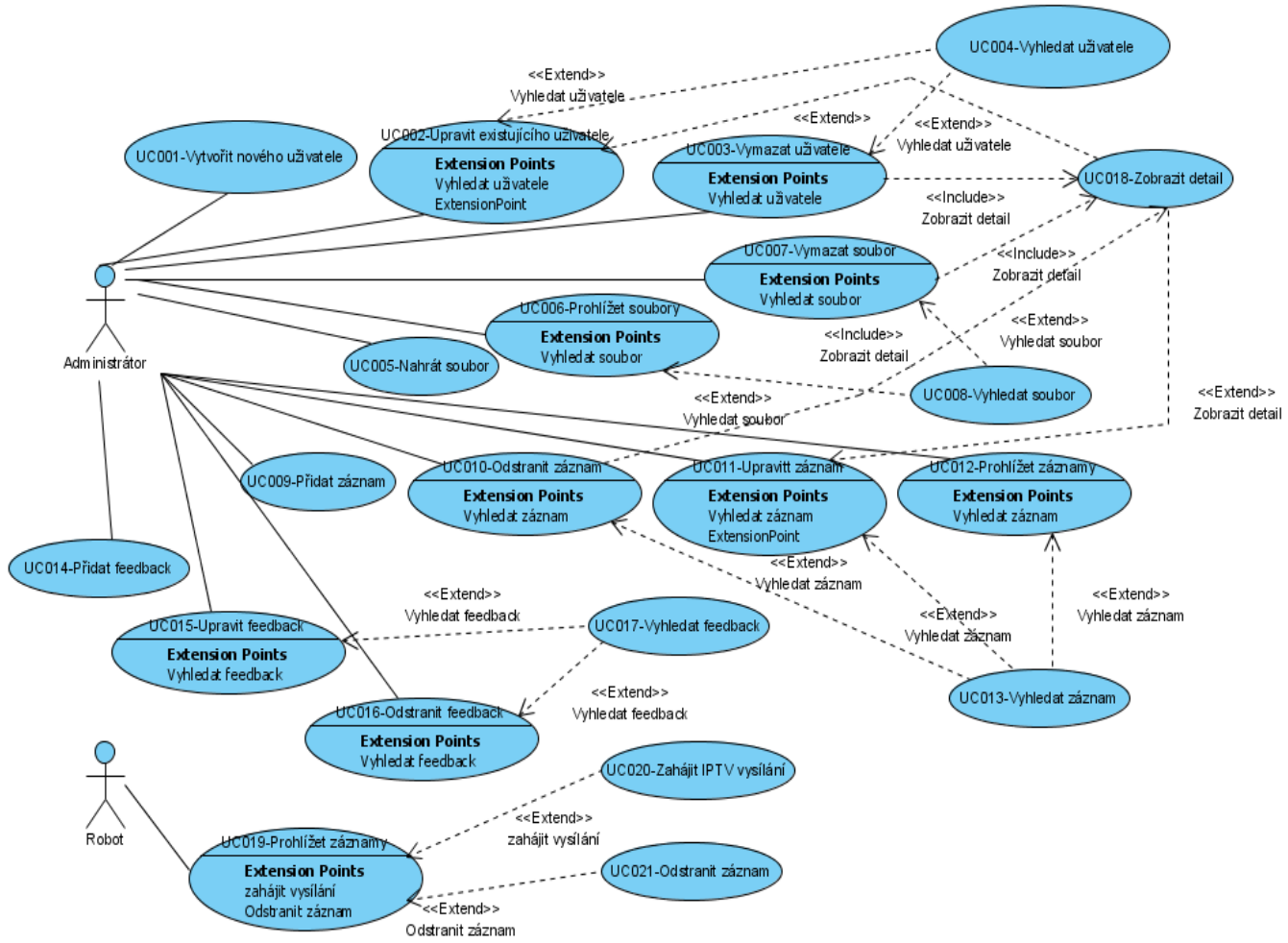
## 3.1 Diagram případů užití (Use case diagram)

Use case diagramy, neboli diagramy případů užití zachycují rozsáhlost aplikace, její obecnou funkci, jednotlivé uživatele, kteří budou s aplikací pracovat. A vymezuje těmto uživatelům (actorům) přesně definovanou funkci v systému. Pod slovem actor si lze představit buď fyzickou osobu, která bude se systémem pracovat, nebo abstraktní stroj, který bude vykonávat předem definovanou činnost. Diagram případů užití vztahující se k aplikaci pro plánování IPTV je patrný na obrázku 3.1.

Základním prvkem diagramu jsou dvě role (administrátor a robot), ke kterým se vztahují veškeré případy užití. Administrátor zde představuje fyzickou osobu se všemi dostupnými právy. Pro lepší přehlednost lze případy užití vztahující se k administrátorům rozdělit do několika skupin – uživatelé, soubory, záznamy a feedbacks. V těchto skupinách jsou dále případy užití specifikující možné operace v dané skupině. Znamená to tedy, že například ve skupině uživatelé bude možné provádět tyto operace – vytvořit nového uživatele, upravit existujícího uživatele, vymazat uživatele, vyhledat uživatele a zobrazit detail (případ užití zobrazit detail se vztahuje k více skupinám). Pro jednoznačnost je každý případ užití označen *UCXXX*, kde UC je zkratkou Use case a XXX představuje pořadové číslo případu. Role robot nepředstavuje fyzickou osobu, ale programový kód, který se stará o správu objednaných vysílání (záznamů). V definovaném časovém intervalu prochází databázi se záznamy. V případě, že nalezne záznam, který by měl být spuštěn, tak tento záznam převede do stavu ke spuštění. V případě, že nalezne záznam, který již není aktuální a byl odvysílán, tak jej odstraní z databáze.

Vazby mezi případy vyhledat uživatele, vyhledat soubor, vyhledat záznam, zobrazit detail atd. jsou znázorněny přerušovanou čárou. Jedná se o speciální relace mezi případy. V diagramu jsou takovéto relace dvou typů – extend a include. Relace extend říká, že klientský případ může být o

dodavatelský (rozšiřující) případ užití rozšířen, ale není to podmínkou. Relace include říká, že klientský případ musí být o dodavatelský (rozšiřující) případ užití rozšířen. Jako příklad lze uvést to, že před upravením uživatele se může, ale nemusí použít vyhledávání, které specifikuje parametry uživatele, kterého chceme upravit (relace extend). Jestliže má být uživatel vymazán, je vždy před touto operací nutné zobrazit jeho detail, aby došlo k vymazání správného uživatele (relace include).




*Obr.3.1 Diagram případů užití*

## 3.2 Diagram aktivit (Activity diagram)

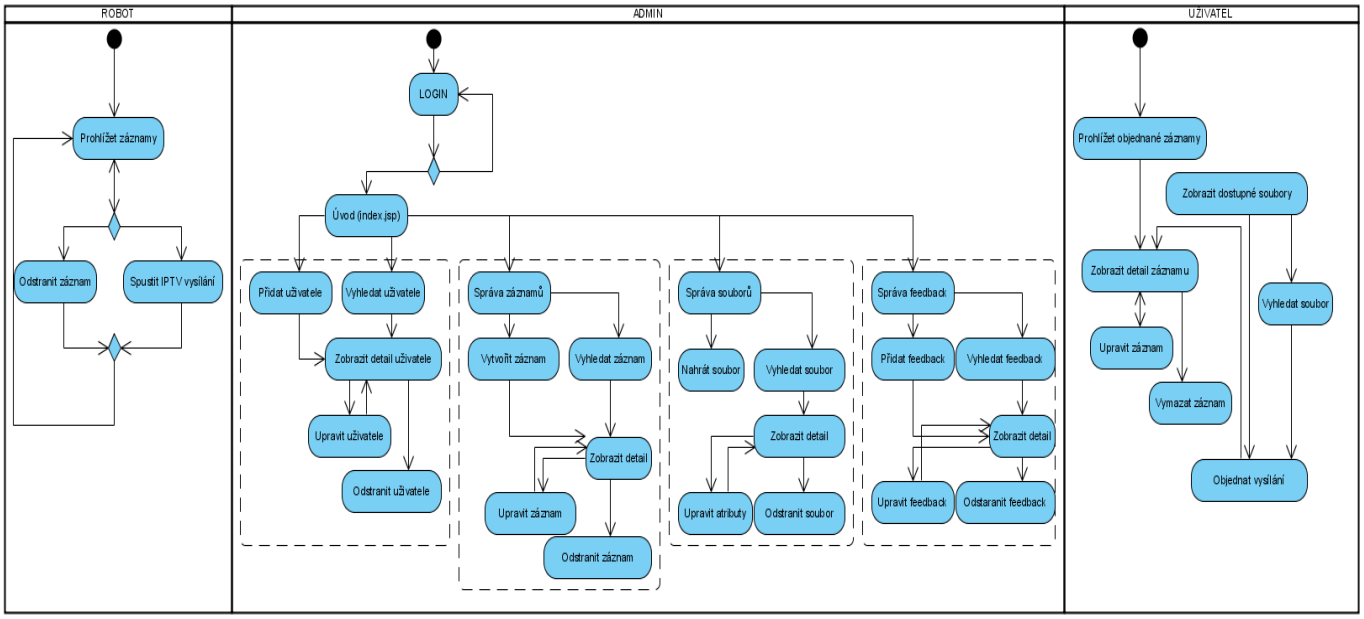
Diagramem aktivit je znázorněno dynamické chování aplikace. Zobrazuje předávání informací mezi jednotlivými operacemi a chování aplikace jako celku. Diagram je dále členěn podle

toho kdo s aplikací bude pracovat, protože každý typ uživatele má v aplikaci své specifické aktivity viz obr. 3.2.

Aktivity uživatele robot lze popsat tak, že bude prohlížet záznamy v databázi. Načtený záznam poté předá do podmínky (značka čtyřúhelníku), kde se rozhodne, zda má být záznam převeden do stavu ke spuštění nebo být vymazán. Jestliže nelze vykonat ani jednu z těchto podmínek, nebo již byla podmínka vykonána, tak se vrací zpět k prohledávání záznamů.

V případě administrátora se musí jako první provést přihlášení do aplikace. Jestliže je přihlášení neúspěšné, tak se načte opět obrazovka pro opravu přístupových informací. V opačném případě se uživatel dostane na úvodní stránku, z které lze za pomoci menu přistupovat do dostupných sekcí aplikace. Jak probíhá předávání dat mezi jednotlivými aktivitami v případě úspěšně přihlášeného uživatele je patrné z diagramu.

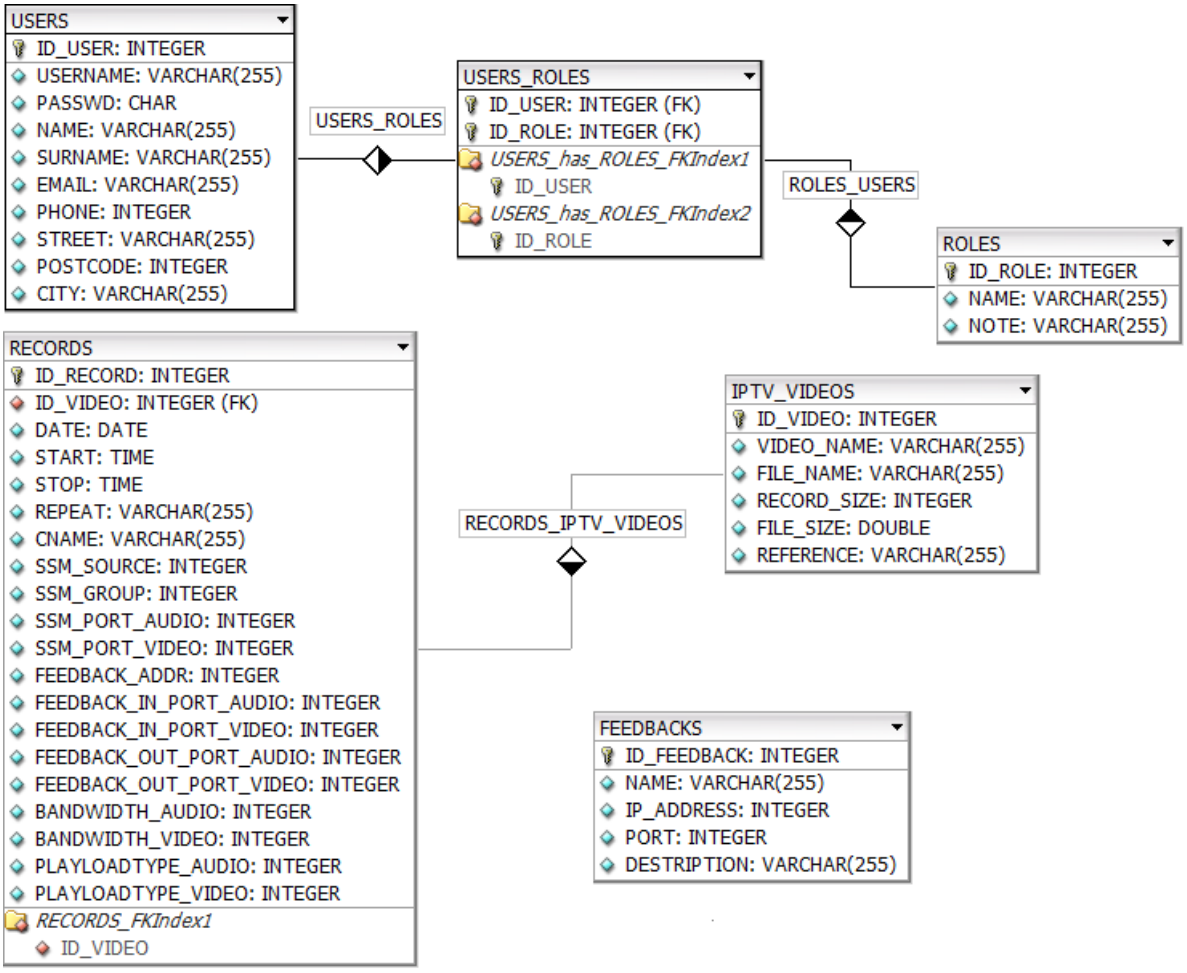
Jestliže bude aplikaci využívat běžný uživatel, tak po přístupu má možnost prohlížet dostupné soubory v těchto souborech poté vyhledávat na základě zadaných parametrů a případně objednat vysílání. Dále má možnost prohlížet již objednané vysílání a to podle potřeby upravovat, nebo zrušit.




*Obr. 3.2 Diagram aktivit*

# 4 Model databáze (Entity Relationship Diagram)

Většina aplikací využívá ke své funkčnosti datové úložiště v podobě databáze. Mezi nejrozšířenější datové úložiště patří bezesporu MySQL [13]. Entity Relationship Diagramy slouží k vytvoření modelu databáze a relací mezi jednotlivými entitami. Před implementací a vytvořením databáze je vhodné tento model nejprve navrhnout. Návrh by měl vycházet z účelu databáze a jejím použití. Základní stavební kameny při návrhu jsou entita a relace. Entita představuje tabulku v databázi. Obsahuje název a seznam sloupců v tabulce. U každého sloupce bývá uveden datový typ a další nepovinné parametry, které blíže specifikují jeho možnosti. Jestli-že mezi entitami existuje závislost, tak jsou tyto tabulky spojeny relací, která určuje vztah mezi entitami. Existují tři typy relací, které se označují jako 1:1, 1:N a M:N.




*Obr. 4.1 Model databáze*

V modelu databáze každá entita shromažďuje data vzájemně spolu související a tvořící jeden celek. Pro vyjádření vzájemného vztahu mezi těmito celky slouží již zmíněné relace. Model databáze obsahuje tyto entity:

- *USERS* – shromažďuje informace o jednotlivých uživatelích (name, surname, username, password atd.).
- *ROLES* – dostupné role v aplikaci. Je zde uveden název role (name) a poznámka vztahující se k roli, která ji dále zpřesňuje (note).
- *USERS_ROLES* – tato entita je výsledkem relace M:N mezi *USERS* a *ROLES*, podrobněji popsána bude dále.
- *IPTV_VIDEOS* – obsahuje informace o nahraném souboru na server prostřednictvím aplikace. Po nahrání souboru dojde k vytvoření záznamu, který obsahuje jméno souboru (*VIDEO_NAME* – využito jako název pořadu při prohlížení uživatelem), konkrétní název nahraného souboru (*FILE_NAME*), délku video záznamu (*RECORD_SIZE*), velikost nahraného souboru (*FILE_SIZE*) a cestu k souboru na serveru (*REFERENCE*). Tato cesta se vztahuje k domovské složce příslušného webového serveru.
- *RECORDS* – neboli záznamy, které bude vytvářet běžný uživatel aplikace při objednání vysílání z nabídky, která je obsažena v *IPTV_VIDEOS*. Důležitými údaji pro vytvoření záznamu je informace o vybraném pořadu (cizí klíč – *ID_VIDEO*), datum (*DATE*), začátek vysílání (*START*), konec vysílání (*STOP*) a případně periodicita vysílaní (REPEAT) daného pořadu. Dále jsou zde uvedeny potřebné parametry pro konkrétní vysílání pomocí protokolů RTP a RTCP.

V případě modelu databáze pro aplikaci IPTV client, byla použita relace M:N mezi entitami *USERS* a *ROLES*. Znamená to, že jeden uživatel bude moci disponovat více rolemi a jedna role může být přiřazena více uživatelům. Spojení konkrétního uživatele s konkrétní rolí je v tzv. „mezientitě“ nazvané *USERS_ROLES*. Tato entita je součástí relace M:N a obsahuje dva sloupce. Záznam v této entitě obsahuje ID uživatele (sloupec *ID_USER*) a ID přiřazené role (sloupec *ID_ROLE*). Další použitou relací je 1:N mezi entitami *RECORDS* a *IPTV_VIDEOS*. Zde je vazba použita z důvodu, že jedno video může být obsaženo ve více záznamech, ale jednomu záznamu odpovídá pouze jedno video.

# 5 Adresářová struktura

Pro lepší pochopení dalšího výkladu a orientaci v aplikaci bude uvedena adresářová struktura celé aplikace. Jak již bylo napsáno u popisu webového serveru Tomcat, tak složka ve které bude projekt uložen, nese název webapps. Adresářová struktura vypadá takto:

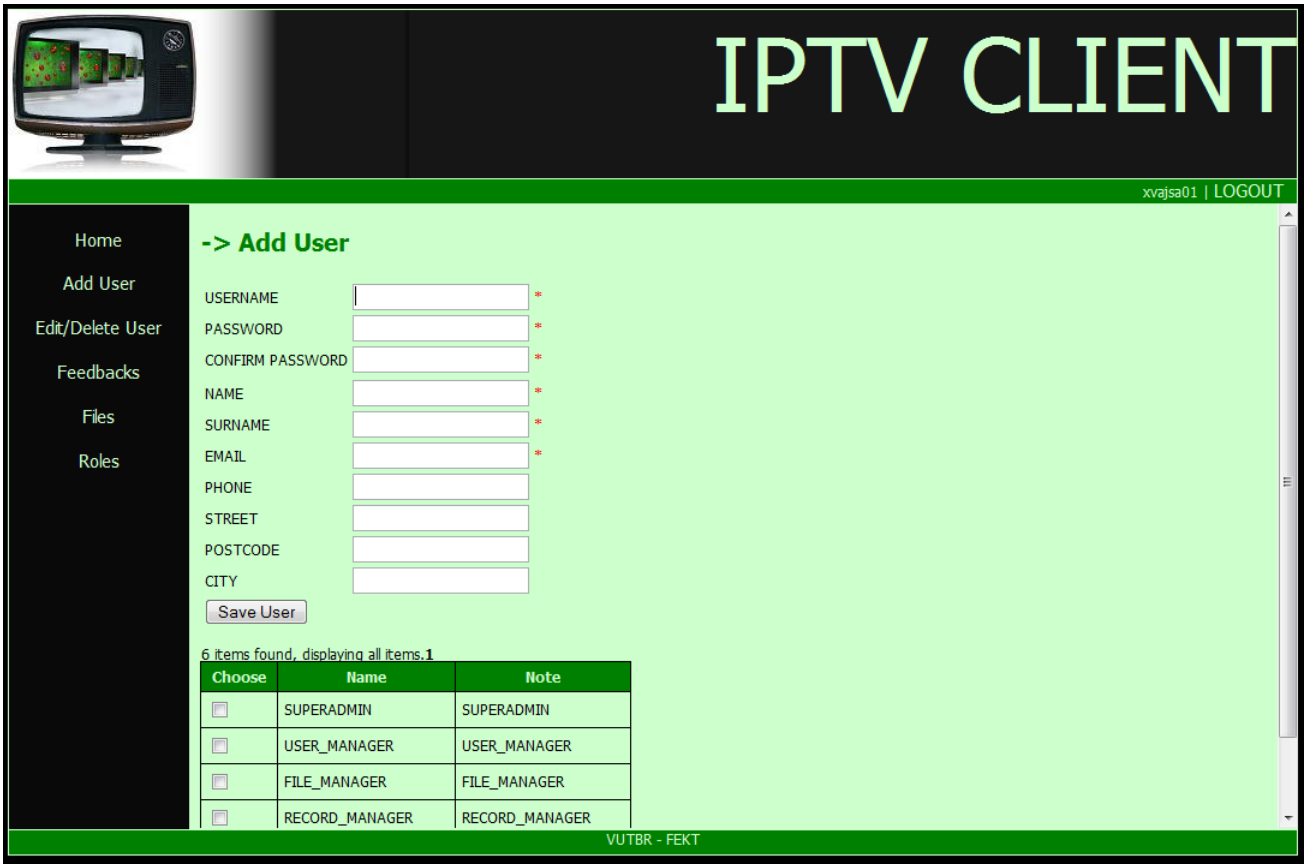
```
+IPTVclient\
  +src\
  |--+hibernate.properties
  |--+hibernate.cfg.xml
  |--+stripesResources.properties
  |--+log4j.properties
  |--+cz\
  |----+vutbr\
  |------+xvajsa01\
  |--------+model\  (třídy tvořící model databáze)
  |--------+manager\(metody pro výměnu dat mezi aplikací a databází)
  |--+net\
  |----+sourceforge\
  |------+stripes\
  |--------+iptvclient\   (ActionBeany aplikace)
  |--+test\
  |----+cz\
  |------+vutbr\
  |--------+xvajsa01\
  |----------+manager\ (třídy testující správnou funkci metod z manageru)
  +IPTV\(jsp stránky aplikace)
  |--+img\
  |--+style\  (definice stylu uživatelského rozhraní)
  +WEB-INF\
  |--+web.xml (konfigurační soubor projektu)
  |--+upload\
  |--+lib\
```

Adresář src\ obsahuje 2 konfigurační soubory frameworku hibernate, jeden frameworku stripes a soubor log4j.properties pro konfiguraci log souborů Javy a dále všechny zdrojové kódy napsané v Javě (třídy, ActionBeany, testovací třídy). Adresář IPTV\ obsahuje všechny jsp stránky, které definují uživatelskou část aplikace. Právě tyto stránky využívají již zmíněné ActionBeany a třídy z adresáře src\. V podadresáři img\ jsou obrázky vyskytující se v aplikaci a v podadresáři style\ CSS styly definující vzhled aplikace. Součástí každého projektu musí být adresář WEB-INF\, který obsahuje soubor web.xml. Dále se zde nachází podadresář lib\, ve kterém jsou uloženy veškeré jar knihovny nezbytné pro běh aplikace (součástí jsou i jar knihovny frameworků stripes, hibernate a JUnit). Do adresáře upload\ se ukládají soubory nahrané na server.

# 6 Návrh uživatelského rozhraní

Cílem bylo navržení jednoduchého a přehledného uživatelského rozhraní, ve kterém bude lehké se orientovat, je zároveň prvním krokem při tvorbě aplikace. Ukázka výsledného vzhledu je patrná z obrázku 6.1. Pro tvorbu uživatelského rozhraní byl použit značkovací jazyk HTML používaný pro webové prezentace spolu s kaskádovými styly definující rozložení stránky, styl písma a použité barvy.

Hlavním prvkem stránky je hlavička, ve které je uveden nadpis – IPTV CLIENT, která je statická a v celé aplikaci neměnná. Pod hlavičkou je umístěn tzv. informační panel, na kterém je zobrazeno uživatelské jméno právě přihlášeného uživatele a za oddělovačem odkaz na bezpečné odhlášení ze systému. Pod informačním panelem je zobrazeno menu, prostřednictvím kterého se lze přepínat do různých sekcí aplikace. Vlastní obsah stránky je uveden vedle sloupce s menu a představuje nejdůležitější část. V případě, že je obsah větší než jemu vyhrazený prostor, je automaticky zobrazena posuvná lišta. Celé uživatelské rozhraní je poté zakončeno zápatím lemujícím spodní okraj stránky.




*Obr.6.1 Uživatelské rozhraní aplikace IPTV klient*

Jak již bylo řečeno, celá stránka se skládá z několika bloků (hlavička, informační panel, menu, obsah a zápatí), kde každý z těchto bloků je reprezentován HTML tagem umístěním na stránce:

```
<div id="hlavička">
      obsah hlavičky
</div>
```

Každému takovému bloku odpovídá nastavení jeho stylu pomocí kaskádových stylů CSS. Styly jednotlivých bloků jsou definovány v externím souboru *StylePage.css* a proto na každé stránce, kde mají být tyto styly použity, musí být uveden odkaz na tento externí soubor. Přiřazení stylu bloku se provádí na základě hodnoty id v definici tagu div v souboru *StylePage.css*.

```
#hlavička {
      Font-family: Tahoma;
      .
      .
}
```

Jelikož se jednotlivé části stránky budou v každé sekci opakovat, tak by bylo neefektivní je vždy psát znovu. Proto části jako je hlavička, informační panel a menu byly vloženy do souboru *header.jsp* a zápatí do souboru *footer.jsp*, které jsou vždy na nově vytvořenou stránku importovány. Struktura stránky poté vypadá takto:

```
<%@ include file="header.jsp" %>
Obsah stránky
<%@ include file="footer.jsp" %>
```

### 6.1 Uživatelské rozhraní, framework stripes a ActionBean

Stripes [14] je framework určený pro rychlou tvorbu webových aplikací založených na jazyce Java a nabízí velké množství možností. Od ostatních frameworků podobného typu, jako je například Struct, Spring atd., se odlišuje jednodušším použitím a minimální konfigurací.

Práci s Frameworkem stripes lze rozdělit na část klientskou a na část, která se odehrává na straně serveru. Jako příklad lze uvést formulář pro vytvoření nového uživatele. Na straně klienta vytvoříme formulář pomocí tagů stripes a nastavíme, že po odeslání formuláře se mají předat zadané informace příslušnému ActionBeanu. Ve chvíli kdy tyto informace předáme, spustí webový

server kód, který není běžnému uživateli dostupný. Kód běží na serveru a převezme odeslané informace z formuláře, tyto informace zpracuje a vrátí klientské části aplikace pouze výsledek. Operace, které proběhly při zpracovávání informací z formuláře nejsou uživateli dostupné. Tyto bloky kódů pro zpracování informací z klientské části se nazývají ActionBeany. Jedná se v podstatě o soubor s příponou *.java* reprezentující třídu s tím, že tato třída musí implementovat rozhraní *ActionBean*. Dále musí být v názvu souboru reprezentující třídu (ActionBean) slovní spojení ActionBean – např. *AddUserActionBean.java.*

Důležitou vlastností frameworku stripes na straně serveru je validace zadaných hodnot do odesílaného formuláře. Po odeslání formuláře se v příslušném ActionBeanu zkontroluje, zda odeslané hodnoty splňují určitá pravidla, např. uživatelské jméno bylo zadáno, zadané heslo bylo delší než 5 znaků, pole pro telefonní číslo obsahuje pouze číselné hodnoty atd.. Tímto ověřením lze předejít dalším problémům při zpracování přijatých informací v ActionBeanu.

Jako příklad klientské části lze uvést formulář pro nahrání souboru na server. Formulář obsahuje textové pole pro zadání názvu záznamu (*RECORD NAME*) a textové pole pro zadání cesty k souboru, který má být na server nahrán. Obě tyto položky jsou povinné pro úspěšné uložení souboru na serveru. Po stisku tlačítka pro odeslání dojde k předání zadaných informací souboru *FileUploadActionBean.java* (pozn. Jedná se o soubor uvedený v tagu action v hlavičce formuláře. Zde je název tohoto ActionBeanu uveden ve tvaru FileUpload.action).

```
<stripes:form action="/iptvclient/FileUpload.action" focus="">
      ${actionBean.resolution}
      RECORD NAME:<stripes:text name="video.videoName" />
      FILE:<stripes:file name="file" />
<stripes:submit name="saveFile" value="Save File"></stripes:submit>
</stripes:form>
```

Jestliže byl formulář odeslán, spustí se soubor s názvem FileUploadActionBean.java, který nejdříve ověří, zda byly vyplněny obě položky ve formuláři. Jestliže jedna z položek nebyla zadána, tak se vrátí zpět k uživateli chybové hlášení a ke zpracování formuláře (odeslání souboru) nedojde. Jestliže byly odeslané informace z formuláře v pořádku, dojde tak k provedení těla metody *saveFile()*. Tato metoda musí mít stejný název jako tlačítko pro odeslání formuláře a před její signaturou je uvedena anotace @DefaultHandler. Tato metoda vrátí výsledek zpracování jsp stránce, ze které byl příslušný FileUploadActionBean.java spušten.

```
public class FileUploadActionBean implements ActionBean {
      private ActionBeanContext context;
      @ValidateNestedProperties( {
      @Validate(field = "videoName", required = true)})
      public Video video;
      @Validate(required = true)
      private FileBean file;

      @DefaultHandler
      public Resolution saveFile() {
            .
            return new ForwardResolution("/IPTV/UploadFile.jsp");
      }
```

Jak je patrné z příkladu, tak validace dat pomocí frameworku stripes je velice jednoduchá a nabízí hodně možností. Například ověření zadané emailové adresy, ke kterému je potřeba pouze jeden řádek kódu, který dále ověří, zda je adresa opravdu zadána a zda její délka nepřesahuje 50 znaků:

```
@Validate(converter=EmailTypeConverter.class,required=true,maxlength=50)
```

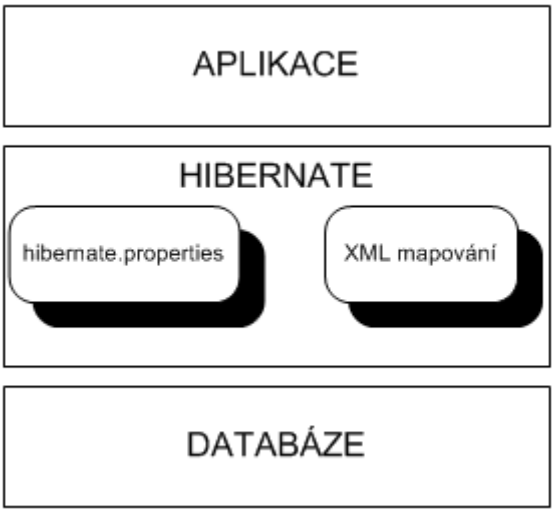
Na rozdíl o jiných programovacích jazyků, jako je například PHP, které též slouží pro vývoj webových aplikací, není potřeba psát žádné složité bloky kódů pro ověřování zadaných hodnot. Validace vstupních dat je při návrhu webové aplikace velice důležitá z důvodu zachování integrity dat. Framework stripes je proto výborný nástroj, jak toto provést a zároveň si ulehčit práci.

# 7 Model databáze a framework hibernate

Model databáze, který bude použit v aplikaci, byl již popsán. Dalším krokem bude na databázovém serveru MySQL vytvořit potřebné entity popsané v tomto modelu, realizovat relace mezi těmito entitami a dále realizovat výměnu dat mezi aplikací a databází. K těmto dílčím krokům bude použit právě framework hibernate. V následující části bude uvedeno co je framework hibernate a k čemu je vhodné ho použít. Dále bude ukázáno jeho využití v aplikaci.

## 7.1 Hibernate

Hibernate [15] je stejně jako stripes framework vytvořený v Javě, který lze použít při tvorbě aplikace, která bude spolupracovat s databází. Hibernate slouží k tzv. objektově-relačnímu mapování dat (ORM). Znamená to tedy, že se používá pro převod dat z objektového modelu do modelu relačního a naopak. Hibernate stojí mezi aplikací a databází, zajišťuje jejich vzájemnou komunikaci a předávání dat ve vhodné podobě, viz obr. 7.1




*Obr.7.1 Struktura frameworku hibernate*

Soubor hibernate.properties obsahuje konfiguraci spojení s databází (adresa databázového serveru, uživatelské jméno, heslo, název databáze, atd.). Tento soubor je umístěn ve složce src/ spolu se souborem hibernate.cfg.xml, který nese informace o tom, které třídy budou mapovány jako perzistentní objekty. Místo mapování perzistentních objektů pomocí XML souborů byly v našem

případě použity anotace [15]. Jsou aplikovány přímo ve třídě, které má představovat perzistentní objekt. Anotace jsou modernější a jednodušší podobou XML mapování.

### 7.2 Entity jako perzistentní objekty

Aby třída mohla představovat perzistentní objekt, tak musí implementovat rozhraní *Serializable*. Objekt je převeden na posloupnost bytů a v případě potřeby obnoven a znovu použit. Není závislý na životním cyklu programu a po jeho vytvoření je stále dostupný. Do těchto objektů se načítají data z databáze a jsou dostupná pro aplikaci, proto jejich nezávislost na životním cyklu programu je důležitá. Data jsou uložená v perzistentním objektu do té doby než jsou přepsána jinými daty. Toto je výhodné i z hlediska vytížení databázového serveru. Při opakujících se požadavcích o stále stejná data nedochází k načítání z databáze, ale z perzistentního objektu.

Smyslem frameworku hibernate je právě ke každé entitě v databázi vytvořit takovýto perzistentní objekt. Ten musí implementovat rozhraní *Serializable* a za použití anotací se nastaví, že se jedná o objekt vztahující se na entitu v databázi.

```
@Entity
@Table(uniqueConstraints = @UniqueConstraint(columnNames = {"id"} ))
public class User implements Serializable {
private static final long serialVersionUID = -4814719013497551615L;
      private Integer id;
      private String username;
      private String passwd;
      private String name;
      private String surname;
      private String email;
      private Integer phone;
      private String street;
      private Integer postcode;
      private String city;
      private List<Roles> roles = new ArrayList<Roles>();

      @Id @GeneratedValue(strategy = GenerationType.AUTO)
      public Integer getId() {
            return id;  }
```

```
public void setId(Integer idUser) {
      this.id = idUser; }

@ManyToMany(cascade = {CascadeType.PERSIST})
@JoinTable(name = "USER_ROLE")
public List<Roles> getRoles() {
      return roles;}

public void setRoles(List<Roles> roles) {
      this.roles = roles;}

<zbývající get a set metody>}
```

Ve výpisu kódu je uvedena třída vztahující se k entitě USER v databázi. Jednotlivé anotace předchází znak @. Před hlavičkou třídy jsou uvedeny anotace, které říkají, že se jedná o entitu a dále, že primární klíč entity je sloupec s názvem *id*. Následuje hlavička třídy, ve které je uvedeno, že třída implementuje rozhraní *Serializable* a bude tudíž využita jako perzistentní objekt. Jednotlivé deklarované vlastnosti třídy (proměnné) představují jednotlivé sloupce shodných názvů v entitě *USER*. Následuje vytvoření *get* a *set* metod pro jednotlivé vlastnosti třídy nad kterými lze dále specifikovat vlastnosti sloupce v databázi pomocí anotací. Například to, že sloupec *id* se nebude nastavovat ručně, ale při vložení záznamu do databáze se jeho hodnota zvětší o jedna oproti poslednímu vloženému záznamu. Má nastavenu vlastnost auto-increment.

Práce s takovým to objektem poté probíhá tak, že jednotlivé jeho vlastnosti se nastaví např. na hodnoty získané z formuláře. Poté se zavolá metoda pro uložení tohoto objektu do databáze, která jako vstupní parametr má právě tento objekt. Perzistentní objekty celé aplikace se nachází v adresáři /src/cz/vutbr/xvajsa01/model/. Jednotlivé objekty jsou reprezentovány třídami, z nichž každá třída je uložena v samostatném souboru. Třídy reprezentující vrstvu model aplikace jsou tyto:

```
|-+model\
 |-+User.java
 |-+Roles.java
 |-+Video.java
 |-+Record.java
 |-+FeedBack.java
```

### 7.3 Perzistentní objekty a relace mezi entitami

Relace mezi entitami v databázovém modelu (1:1, 1:N, M:N), který byl již popsán se pomocí hibernate řeší na úrovni perzistentních objektů. Hibernate tyto relace řeší velice elegantně a práce s nimi je jednoduchá. Pro tyto relace se používají též anotace a v závislosti na typu relace lze použít jednu z těchto anotací - @OneToOne (relace 1:1), @OneToMany (relace 1:N), @ManyToMany (relace M:N).

V předchozí ukázce kódu je uveden výpis objektu vztahující se na entitu uživatelů. Z ER diagramu uvedeného na obrázku 4.1 je patrné, že entita uživatelů je v relaci M:N s entitou rolí. V hibernate se tato relace vytvoří tím, že v objektu uživatelů se přidá kontejner typu rolí:

```
private List<Roles> roles = new ArrayList<Roles>();
```

dále se pro tento kontejner vytvoří *get* a *set* metody a k metodě *get* pro získání tohoto kontejneru uvedeme anotace:

```
@ManyToMany(cascade = {CascadeType.PERSIST})
@JoinTable(name = "USER_ROLE")
```

Tím vytvoříme relaci M:N mezi uživateli a rolemi. Anotace @JoinTable specifikuje tzv. mezientitu, která je pro tuto vazbu nezbytná a hibernate jí na základě této anotace automaticky vytvoří. Jestliže poté budeme chtít získat uživatele z databáze, tak v spolu s uloženými informaci o uživateli získáme i jemu odpovídající role, které budou obsaženy v kontejneru rolí v objektu uživatele. Není potřeba vytvářet žádné spojené dotazy, nebo načítat data postupně z jednotlivých entit. Další zajímavostí je atribut cascade v anotaci @ManyToMany. Tím je docíleno to, že jestliže vymažeme uživatele z databáze, tak se automaticky vymažou i všechny záznamy z entity USERS_ROLES, které danému uživateli náleží.

### 7.4 Vrstva manager

Vrstva manager stojí nad vrstvou model, což jsou perzistentní objekty představující jednotlivé entity v databázi, které jsou popsány výše. Tato vrstva má za úkol výměnu dat mezi aplikací (vrstvou model) a databází. Znamená to, že se stará o ukládání hodnot perzistentních

objektů do databáze a naopak v případě potřeby získávání data z databáze a předávání tyto data perzistentním objektům. Třídy, které tvoří vrstvu manager, se nacházejí v adresáři /src/cz/vutbr/xvajsa01/manager/, kde každé třídě z vrstvy model odpovídá jedna třída z vrstvy manager, která obsahuje metody pro operace s daty a v názvu třídy obsahuje slovo manager (odlišení od tříd modelu):

```
|-+manager\
  |-+UserManager.java
  |-+RolesManager.java
  |-+VideoManager.java
  |-+RecordManager.java
  |-+FeedBackManager.java
  |-+MySecurityManager.java
```

Třída MySecurityManager.java je využita pro řešení bezpečnosti aplikace a podrobněji bude popsána dále. Metody obsažené ve třídách jsou si podobné, jako příklad uvedu výpis metod z třídy UserManager.java. Operaci, kterou metoda provádí lze odvodit z názvu metody.

```
|-+manager\
  |-+UserManager.java
    |-+public void userSave(User user)
    |-+public void userUpdate(User user)
    |-+public User getUserById(int id)
    |-+public User getUserByUsername(String username)
    |-+public List<User> getUserAll()
    |-+public List<User> searchUser()
    |-+public void deleteUser(User user)
```

### 7.4.1 Testování metod z vrstvy manager pomocí JUnit

JUnit je framework vytvoření pro tvorbu testů již napsaného kódu v jazyce Java. Znamená to tedy, že po napsání bloku kódu lze poté tento blok pomocí JUnit otestovat zda plní svou funkci podle předpokladů. Písmeno J na začátku jména frameworku značí, že se jedná o testovací framework určený pro programovací jazyk Java. Existují testovací frameworky i pro další programovací jazyky např. CPPUnit (pro C++) atd.

JUnit je v našem případě použit pro testování metod ve vrstvě manager. Každé třídě z adresáře manager/ odpovídá jedna testovací třída umístěna v adresáři IPTVclient/test/cz/vutbr/xvajsa01/manager/. Na následující ukázce kódu je patrný způsob testování metod z manageru. Dále je ukázána prakticky práce s perzistentními objekty, která byla popsána výše. Výpis kódu testuje metody z UserManager a jako perzistentní objekt využívá třídu User a Roles z vrstvy model.

```
public class UserManagerTest {
private UserManager uMng = new UserManager();
      private RolesManager rMng = new RolesManager();

      @Before
      public void init() {
            HbnSessionUtil.beginTransaction();}

      @After
      public void destroy() {
            HbnSessionUtil.resolveTransaction();
            HbnSessionUtil.closeSession();}
      @Test
      public void createAndStoreUser() {
            User user = new User();
            User user2, user3 = new User();
            Integer id;
            user.setUsername("username");
            user.setPasswd("passwd");
            user.setName("name");
            user.setSurname("surname");
            user.setEmail("email");
            user.setPhone(123);
            user.setStreet("street");
            user.setPostcode(123);
            user.setCity("city");
            user.setRoles(rMng.getRolesAll());

            uMng.userSave(user);
            user.setUsername("NEWusername");
            uMng.userUpdate(user);
            id = user.getId();
            user2 = uMng.getUserById(id);
            user3 = uMng.getUserByUsername("NEWusername");
```

```
Assert.assertEquals(user, user2);
Assert.assertEquals(user2, user3);
uMng.deleteUser(id);}}
```

Nejdříve je vytvořen nový perzistentní objekt typu User, kterému jsou přiřazeny hodnoty a všechny dostupné role v entitě ROLES. Dále je tento objekt uložen do databáze. Poté je objektu změněno uživatelské jméno na NEWusername a následuje update záznamu v databázi. Tento objekt je načten z databáze nejdříve pomocí id (objekt user2) a poté podle nově vytvořeného uživatelské jména (objekt user3). Rozhodující částí testování jsou poté příkazy Assert.assertEquals(user, user2) a Assert.assertEquals(user2, user3), které ověří, zda se objekty získané z databáze shodují s těmi, které byly do databáze ukládány. V případě, že se shodují, bude test vyhodnocen jako úspěšný. V opačném případě jako neúspěšný. Posledním krokem je vymazání vytvořeného záznamu z databáze.

JUnit testy jsou výhodné z hlediska rychlosti a efektivnosti programování. Uvedeným testem jsme ověřili správnou funkci vrstvy model a manager. Lze tedy na těchto vrstvách dále stavět. Případné další chyby budou s velkou pravděpodobností mimo tyto vrstvy a jejich nalezení je poté jednodušší.

# 8 Zabezpečení aplikace

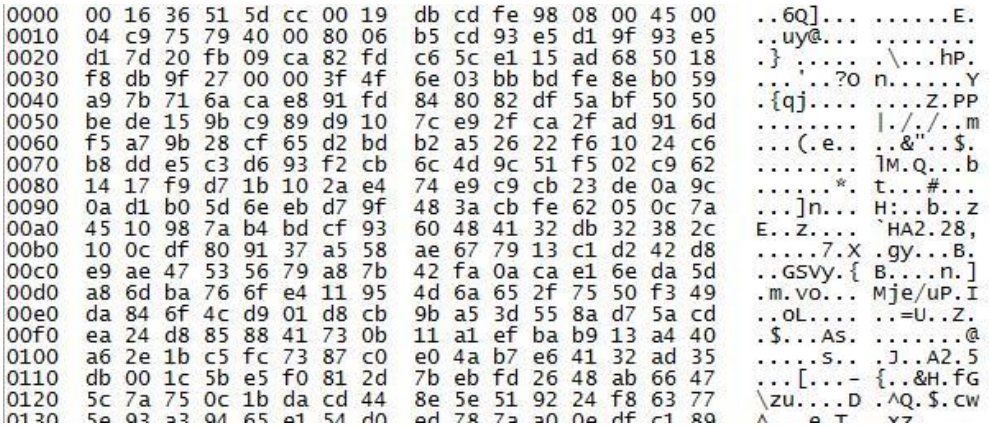
Zabezpečení aplikace je řešeno na několika úrovních a snaží se zamezit neoprávněnému přístupu do administrační části aplikace a tím získání citlivých informací z databáze jako jsou například osobní informace uživatelů a zvláště pak jejich hesla, která většina z nich používá pro přístup i k ostatním službám na internetu. Dále jsou řešeny role jednotlivých uživatelů v systému, aby každý uživatel mohl provádět pouze operace, které mu přísluší a na které má přidělené právo.

## 8.1 Komunikace mezi klientem a webovým serverem

Zabezpečení na nejvyšší úrovni, které není řešeno programově v aplikaci, ale přímo na webovém serveru řeší šifrovanou komunikaci mezi klientem a webovým serverem, na kterém je aplikace provozována. Výměna dat s webovým serverem probíhá šifrovaně, a tudíž při zachycení komunikace síťovým analyzátorem dojde k zachycení šifrované podoby přenášených dat. Výměna dat probíhá pomocí https. Jedná se o běžný protokol http a nadstavbou SSL, která má za úkol šifrovat komunikaci mezi klientem a webovým serverem. Přenos v šifrované podobě je důležitý zejména při přihlašování klienta do aplikace. Klient do formuláře ve webovém prohlížeči zadá své přístupové jméno a heslo a poté data odešle k ověření na server. V případě, že by komunikace nebyla šifrována, tak pomocí síťového analyzátoru lze komunikaci zachytit a získat heslo i uživatelské jméno uživatele. Na obrázku 8.1 je patrná komunikace pouze pomocí http. Ze zachyceného paketu lze zjistit, že uživatelské jméno je username a heslo passwd. Paket přenesený pomocí https obsahuje pouze náhodné znaky, kterým rozumí komunikující strany na základě vygenerovaných klíčů, obrázek 8.2.

```
0000  00 19 db cd fe 98 00 16  36 51 5d cc 08 00 45 00   ........ 6Q]...E.
0010  00 e6 2b 85 40 00 80 06  03 a5 93 e5 d1 7d 93 e5   ..+.@... .....}..
0020  d1 9f 07 28 1f 90 60 95  06 5b 16 78 82 e8 50 18   ...(..`. .[.x..P.
0030  ff 08 d7 ed 00 00 43 6f  6e 74 65 6e 74 2d 54 79   ......Co ntent-Ty
0040  70 65 3a 20 61 70 70 6c  69 63 61 74 69 6f 6e 2f   pe: appl ication/
0050  78 2d 77 77 77 2d 66 6f  72 6d 2d 75 72 6c 65 6e   x-www-fo rm-urlen
0060  63 6f 64 65 64 0d 0a 43  6f 6e 74 65 6e 74 2d 4c   coded..C ontent-L
0070  65 6e 67 74 68 3a 20 31  31 38 0d 0a 0d 0a 75 73   ength: 1 18....us
0080  65 72 6e 61 6d 65 4c 6f  67 69 6e 3d 75 73 65 72   ernameLo gin=user
0090  6e 61 6d 65 26 70 61 73  73 77 6f 72 64 3d 70 61   name&pas sword=pa
00a0  73 73 77 64 26 6c 6f 67  69 6e 3d 4c 6f 67 69 6e   sswd&log in=Login
00b0  26 5f 73 6f 75 72 63 65  50 61 67 65 3d 25 32 46   &_source Page=%2F
00c0  49 50 54 56 25 32 46 4c  6f 67 69 6e 2e 6a 73 70   IPTV%2FL ogin.jsp
00d0  26 5f 5f 66 70 3d 53 56  38 52 4c 44 6a 73 32 30   &__fp=SV 8RLDjs20
00e0  4b 38 79 25 32 42 59 4b  59 32 75 76 39 77 25 33   K8y%2BYK Y2uv9w%3
```

*Obr. 8.1 Přenášená data v nešifrované podobě*




*Obr. 8.2 Přenášená data v šifrované podobě*

Pro nastavení služby je zapotřebí nejdříve vygenerovat certifikát. To se provádí spuštěním nástroje keytool s potřebnými parametry. Tento nástroj je součástí instalace Java platformy. Pro vygenerování certifikátu je zapotřebí nástroj keytool spustit s těmito parametry:

```
keytool -genkey -alias tomcat -keyalg RSA
```

Po provedení příkazu dojde k vygenerování souboru .keystore, který je následně zkopírován do složky webového serveru. Dalším krokem je aktivovat tuto službu a serveru sdělit kde se nachází soubor .keystore. Toto se provádí v konfiguračním souboru serveru server.xml v adresáři conf/. Konfigurační část, která umožní šifrovanou komunikaci mezi webovým serverem a klientem je tato:

```
<Connector
       port="8443" minSpareThreads="5" maxSpareThreads="75"
       enableLookups="true" disableUploadTimeout="true"
       acceptCount="100"  maxThreads="200"
       scheme="https" secure="true" SSLEnabled="true"
       keystoreFile="C:/Tomcat 6.0/.keystore" keystorePass="xvajsa01"
       clientAuth="false" sslProtocol="TLS"/>
```

Z konfigurace je patrné, že služba naslouchá na portu 8443, maximální počet klientů využívajících současně tuto službu je 200. Soubor certifikátu je uložen a adresáři serveru Tomcat a heslo nastavené při vytváření certifikátu je xvajsa01.

### 8.2 Veřejná a chráněná část aplikace

Aplikace je rozdělena na část veřejnou kam můžou přistupovat uživatelé bez nutnosti zadávání uživatelského jména a hesla a na část chráněnou kde je toto vyžadováno. O rozlišení těchto dvou částí se stará tzv. security filtr. Je umístěn v adresáři net/sourceforge/stripes/iptvclient/ a představuje ho soubor SecurityFilter.java. Je to třída napsaná v programovacím jazyce Java, ve které jsou definovány veřejně přístupné stránky. Jestliže klient bude chtít navštívit stránku, která v security filtru není definována jako veřejná, tak automaticky dojde k přesměrování na stránku pro zadání uživatelského jména a hesla. Aby filtr takto fungoval, tak je zapotřebí, aby se webový server o něm dozvěděl a aplikoval ho na potřebné stránky. Proto se informace o security filtru musí uvést do konfiguračního souboru web.xml. Nejdříve nastavíme jméno filtru a třídu, která filtr představuje:

```
<filter>
    <description>Provides login security for IPTV</description>
    <filter-name>IPTVSecurityFilter</filter-name>
    <filter-class>
          net.sourceforge.stripes.iptvclient.SecurityFilter
    </filter-class>
</filter>
```

Poté nastavíme, na jaké soubory bude filtr aplikován. V našem případě jsme ho aplikovali na všechny ActionBeany a jsp stránky aplikace:

```
<filter-mapping>
    <filter-name>IPTVSecurityFilter</filter-name>
    <url-pattern>/IPTV/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<filter-mapping>
    <filter-name>IPTVSecurityFilter</filter-name>
    <url-pattern>/iptvclient/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

### 8.3 Autentizace uživatele

Autentizace uživatele slouží pro jeho identifikaci v aplikaci. Jestliže proběhne v pořádku, tak daný uživatel získá přístup do chráněné části aplikace. Ověřování identity uživatele se provádí na základě zadaného uživatelského jména a hesla v souboru Login.jsp. Na tuto stránku se lze dostat kliknutím na odkaz LOGIN v informačním panelu stránky, nebo přesměrováním, které realizoval security filtr z důvodu pokusu o vstup do chráněné části aplikace. Po potvrzení zadaných přístupových informací dojde ke spuštění souboru LoginActionBean.java, který ověří správnost zadaných hodnot s hodnotami v databázi. Jestliže ověření proběhne v pořádku, tak je uživatel zapsán do kontextu ActionBeanu a je pro něj vytvořeno sezení. Security filtr mu poté umožní přístup na všechny dostupné jsp stránky a využití dostupných ActionBeanů.

### 8.4 Ochrana hesel v databázi

Ve většině případů uživatel používá jedno nebo několik málo hesel pro přístup k více různým službám na internetu, proto ochrana proti odcizení a následnému zneužití je důležitá. Mezi informace o uživateli uložené v databázi patří i jeho emailová adresa. Jestliže tedy uživatel použije pro přístup do aplikace heslo shodné s heslem, které používá pro přístup k emailovému účtu, tak v případě získání těchto informací z databáze bude útočník schopen získat přístup i k emailové schránce uživatele.

Zabezpečení hesel v databázi se provádí tzv. hash funkcí. Tato funkce ze vstupního řetězce libovolné délky vytvoří výstupní řetězec konstantní délky – otisk. Z tohoto otisku by u kvalitní hash funkce nemělo být možné získat původní řetězec. Při ověření správnosti zadaného hesla dochází k porovnání otisku zadaného hesla a otisku hesla uloženého v databázi. Dalším požadavkem na kvalitní hash funkci je, že pro dva různé vstupní řetězce by neměl být shodný otisk. Jinak řečeno – hash funkce by neměla obsahovat kolize. Tyto požadavky splňují hash funkce ze skupiny SHA-2 (SHA-256, SHA-384, SHA-512 a SHA-224).

Dále existují tabulky obsahující slova z běžného jazyka a zároveň jejich otisk, tzv. rainbow tables. Při získání dat z databáze je poté možno porovnat otisky s daty v těchto tabulkách a dohledat tak textovou podobu hesla. Proto je vhodné spojit zadané heslo s dalším řetězcem a až poté vytvořit otisk. Ve vytvořené aplikaci je toto řešeno tak, že před uložením hesla do databáze dojde k jeho zdvojení a přidání doplňujícího řetězce:

```
String sha256 = passwd + passwd + "e38RgH2@";
```

Jestliže si uživatel při vytváření účtu zvolí heslo např. jen slovo heslo tak řetězec, ze kterého se bude vytvářet otisk je - heslohesloe38RgH2@, což je řetězec, který rainbow tables s velkou pravděpodobností obsahovat nebudou a jeho dohledání pomocí porovnání otisků bude náročné, protože obsahuje náhodné znaky, číslice, velká a malá písmena a speciální znaky.

O spojení hesel, přidání řetězce a vytvoření otisku se stará metoda encrypt definována ve třídě Hash.java v adresáři /net/sourceforgre/stripes/iptvclient/, které se jako vstupní parametr předá heslo v textové podobě. Metoda vrací otisk hesla vytvoření hash funkcí SHA-256.

### 8.5 Rozlišení rolí

Po přihlášení do aplikace má uživatel přiřazeny určité role, na základě kterých může vykonávat některé věci uvnitř aplikace a některé ne. Role jsou spojeny s jeho činností v aplikaci. Jsou dostupné tyto role:

- SUPERADMIN (neomezené možnosti)
- USER_MANAGER (práce s uživateli)
- FILE_MANAGER (práce se soubory)
- RECORD_MANAGER (práce se záznamy)
- FEEDBACK_MANAGER (práce s feedbacky)
- ROLE_MANAGER (práce s rolemi)

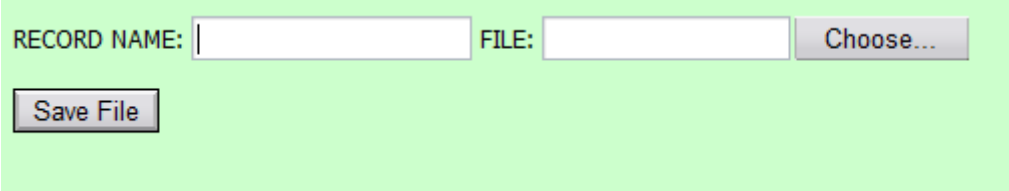
I po úspěšném přihlášení uživatele do aplikace je proto dále nutné před každým jeho úkonem ověřit, zda disponuje rolí, která je pro příslušnou operaci vyžadována. Ověření probíhá na straně serveru před konkrétním provedením požadované operace a provádí se pomocí anotace Secure uvedené nad blokem kódu, který operaci provádí. Jako příklad lze uvést metodu save(), která má za úkol uložit nového uživatele. Definici metody předchází anotace, která ověří, zda uživatel, který chce vytvořit nového uživatele disponuje rolí SUPERADMIN, nebo rolí USER_MANAGER, které jsou vyžadovány pro provedení operace. Jestliže ano, tak následuje provedení metody a uložení nově vytvořeného uživatele do databáze. V opačném případě se metoda neprovede a dojde k vypsání informace o nedostatečném oprávnění pro vykonání této operace.

```
@DefaultHandler
@Secure(roles="SUPERADMIN,USER_MANAGER")
public Resolution save() throws NoSuchAlgorithmException {
      // metoda obsahující kód pro uložení uživatele
}
```

Anotace Secure předá zadané role souboru MySecurityManager.java, ve kterém dojede k ověření, jestli uživatel má přiřazenou alespoň jednu z požadovaných rolí. Jestliže ano, vrátí hodnotu true a kód pokračuje dále za anotací. Jestliže uživatel nedisponuje ani jednou z uvedených rolí, tak MySecurityManager vrátí hodnotu false. Tím dojde k ukončení prováděné operace a následuje přesměrování na stránku s chybovým hlášením (IPTVunauthorized.jsp).

# 9 Upload souborů větší velikosti

Důležitou funkcí IPTV klienta je administrace multimediálních souborů a to především jejich upload na server prostřednictvím webového rozhraní. Jestliže po přihlášení uživatel disponuje příslušnými právy (SUPERADMIN nebo FILE_MANAGER) pro nahrávání souborů, tak má možnost v menu vybrat položku s názvem *Upload File*, tím se dostane na stránku *UploadFile.jsp*, na které je umístěno textové pole pro název záznamu a tlačítko pro procházení souborů na lokálním disku a jeho následné vybrání, viz obrázek 9.1.




*Obr. 9.1 Upload souboru*

Po vybrání souboru se v poli *FILE*, zobrazí úplná cesta k vybranému souboru na lokálním disku. Samotné nahrání souboru na server se provede až po stisku tlačítka Save File, kdy se spustí příslušný ActionBean - *FileUploadActionBean.java.*

Zde se nejprve ověří, zda uživatel disponuje potřebnými právy a zda byl zadán název záznamu a vybrán soubor. Jestliže nebyla jedna z těchto podmínek splněna, tak dojede k výpisu chybové zprávy s žádostí o nápravu. V opačném případě se provede nahrání souboru na server a do databáze se uloží příslušné informace o souboru – název záznamu, název souboru, velikost souboru a odkaz na soubor.

Nejdůležitější částí *ActionBeanu* je vlastní upload souboru na server. Nejprve je provedena deklarace proměnné *file* typu *FileBean*. Za použití dostupných metod v typu *FileBean* se lze o nahrávaném souboru dozvědět další informace, jako je např. název, velikost atd. Dále je potřeba deklarovat proměnnou *newFile* typu *File*, které se pomocí konstruktoru předá cesta pro uložení souboru na serveru + název souboru pod jakým se má nahrávaný soubor uložit. Nakonec se zavolá metoda *save()* typu *FileBean* a jako vstupní parametr se ji předá proměnná *newFile*, tím dojde nahrání souboru na server do požadovaného adresáře a s požadovaným názvem souboru.

```
private FileBean file;
File newFile = new File("webapps/IPTVclient/WEB-INF/upload/" +
file.getFileName());
file.save(newFile);
```

Dalším problémem, který může nastat, je maximální dovolená velikost nahrávaného souboru. Velikost výchozí hodnoty bývá nastavena řádově na jednotky až desítky MB a proto pro upload souborů větší velikosti nedostatečná. Jelikož je v našem případě použit jako výchozí programovací jazyk Java a framework stripes, provádí se konfigurace maximální hodnoty uploadovaného souboru v konfiguračním souboru *web.xml*, který se nachází v adrešáři WEB-INF/ umístěné v projektu. Soubor *web.xml* musí obsahovat následující blok kódu, kde *X* představuje maximální možnou velikost souboru a písmeno *m* značí jednotku. Při tvorbě aplikace byla hodnota *X* nastavena na 1000 a hodnota *m* na MB. Maximální hodnota nahrávaného souboru je tedy 1000MB. Hodnota byla zvolena v závislosti na tom, že se bude pracovat s multimediálními soubory, které mohou dosahovat vyšší velikosti.

```
<init-param>
<param-name>FileUpload.MaximumPostSize</param-name>
      <param-value>Xm</param-value>
</init-param>
```

Při použití jiných nástrojů pro vývoj webových aplikací, jako je například PHP se nastavení maximální hodnoty uploadovaného souboru liší a je nutné seznámit se s dokumentací příslušného nástroje.

# 10 Závěr

Cílem bakalářské práce bylo navrhnout webovou aplikaci pro podporu plánování IPTV vysílání. Úplným začátkem bylo seznámení se s problematikou a možnostmi IPTV a poté si vytvořit představu o celkové koncepci aplikace a tuto představu interpretovat pomocí jazyka UML. Poté následovalo studium všech potřebných nástrojů pro implementaci aplikace. Zejména studium Javy a rámců stripes a hibernate. Právě tyto nástroje zajistily bakalářské práci moderní přístup k tvorbě webových aplikací a i určitý způsob inovativnosti. Jsou to totiž nástroje ne zcela běžně rozšířené. Při prvním použití sice vyžadují mnohem více znalosti z oblasti programování v jazyce Java, ale po získání potřebných informací práci velice urychlí. Rozdělením do jednotlivých vrstev plnících jednotlivé služby je poté možno aplikaci bez velkých problémů rozšířit o nové funkce a služby. V aplikacích pracujících s databází bývá hodně času věnováno psaním dotazů do databáze, následným zpracováním vrácených dat a ukládáním dat. Za použití frameworku hibernate a perzistentních objektů je toho minimalizováno a práce s daty a databází je velice elegantní a hlavně rychlá a jednoduchá. Při tvorbě webových aplikací je všeobecně z velké části využíván programovací jazyk PHP, ve kterém bývá zpravidla vývoj stejné aplikace časově náročnější a obsahující více potřebného kódu. V případě, že bychom si v PHP chtěli ulehčit práci při práci s daty a databází, lze použít framework cakePHP, který je k tomu určen.

Další výhodou je, že získané znalosti z programování v Javě lze uplatnit i v případě tvorby programů určených pro operační systémy typu MS Windows, Unix/GNU Linux i Mac OS. Java je nezávislá na platformě operačního sytému, takže jednou napsaný kód bude fungovat stejně ve všech uvedených operačních systémech. Jak je patrné, tak Javu lze využít pro programování ve více oblastech, což je motivace k získávání dalších znalostí a zkušeností v Javě.

# 11 Seznam použité literatury


**[1]** DigiZone.cz. *IPTV v České republice : Co všechno nabízí IPTV* [online]. DigiZone.cz, 2005-2007 [cit. 2007-12-17]. Dostupný z WWW:< http://iptv.digizone.cz/co-vsechno-nabizi-iptv/>

**[2]** Ing. HUJKA, Petr. *Real - Time Transport Protocol a aplikační rozhraní RTP Java Media Framework* [online]. 2003 , 27.5.2003 [cit. 2007-12-17]. Dostupný z WWW: <http://www.elektrorevue.cz/clanky/03018/index.html>

**[3]** MOLNAR, Karol. *BHWS : IGMP, Observer*. [s.l.] : [s.n.], 2004. 5 s

**[4]** MULTICAST IPTV : RESEARCH GROUP [online]. Brno : FEEC, University Of Technology, [2007] [cit. 2008-04-02]. Dostupný z WWW: <http://adela.utko.feec.vutbr.cz/projects/homepage.html>.

**[5]** Stickfish s.r.o.. *Úvaha o rozdílech mezi Javou a C++* [online]. 1999-2007 , 9.5. [cit. 2007-12-17]. Dostupný z WWW: <http://www.abclinuxu.cz/blog/Republic_of_Mordor/2007/5/9/179646>

**[6]** Sun Microsystems, Inc.. *Sun Developer Network* [online]. 1994-2007 [cit. 2007-12-17]. Dostupný z WWW: <http://java.sun.com/reference/docs/>

**[7]** The Eclipse Foundation. *Eclipse - an open development platform* [online]. 2007 [cit. 2007-12-17]. Dostupný z WWW: <http://www.eclipse.org/>

**[8]** CollabNet. *Subclipse* [online]. 2006 [cit. 2007-12-17]. Dostupný z WWW: <http://subclipse.tigris.org/>

**[9]** CollabNet. *Projects : Subversion License* [online]. 2006 [cit. 2007-12-17]. Dostupný z WWW: <http://subversion.tigris.org/license-1.html>

**[10]** *CVS* [online]. 2007 , 17.5.2007 [cit. 2007-12-17]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/CVS>

**[11]** The Apache Software Foundation. *Application Developer\'s Guide : Deployment* [online]. 1999-2006 [cit. 2007-12-17]. Dostupný z WWW: < http://tomcat.apache.org/tomcat-6.0-doc/appdev/deployment.html>

[**12**] Unified Modeling Language : UML® Resource Page [online]. c1997-2008 , January 15, 2008 [cit. 2008-05-08]. Dostupný z WWW: <http://www.uml.org/>

[**13**] MySQL [online]. MySQL AB., c1995-2008, January 15, 2008 [cit. 2008-05-08]. Dostupný z WWW: http://www.mysql.com/

**[14]** STRIPES Home [online]. 2.6.2. 2007 , Nov 26, 2007 [cit. 2008-05-20]. Dostupný z WWW: <http://www.stripesframework.org/display/stripes/Home>.

**[15]** Hibernate : Java Persistence with Hibernate [online]. 2006 [cit. 2008-05-20]. Dostupný z WWW: <http://hibernate.org/>.

**[16]** PITNER, Tomáš. *JAVA : začínáme programovat*. Monika Samcová; Libor Samec. Praha : Grada Publishing, spol. s.r.o., 2004. 224 s

**[17]** ZAKHOUR, Sharon, et al. Java 6 Výukový kurz. Jakub Mikulaštík. [s.l.] : [s.n.], 2007. 534 s.