Sem vložte zadanie Vašej práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalárska práca

Informační systém pro Centrum epilepsie nemocnice Motol

Peter Tulala

Vedúci práce: Ing. Petr Ježdík, Ph.D.

10. mája 2015

Poďakovanie

V prvom rade by som chel poďakovať vedúcemu práce, Ing. Petrovi Ježdíkovi, Ph.D. za konštruktívnu diskusiu a ochotu pri riešení problémov, ktoré sa pri práci vyskytli. Ďalej by som rád poďakoval rodine a priateľom, ktorí ma podporovali počas celého štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 10. mája 2015 .....

Odkaz na túto prácu

Tulala, Peter. *Informační systém pro Centrum epilepsie nemocnice Motol*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Táto práca sa zoberá návrhom a implementáciou nového informačného systému spolu s užívateľským rozhraním pre správu epileptických pacientov. Nový systém umožňuje organizovať pacientov do štúdií, nastavovať prístupové práva pre užívateľov a štúdie a nahrávať súbory cez webové rozhranie. Súborom je následne možné nastavovať príznaky a vybrané typy súborov vizualizovať na interaktívnej časovej osi.

Klíčová slova databáza pacientov, vizualizácia dát, časová os, prístupové práva, správa súborov

Abstract

This thesis talks about design and implementation of a new information system with a user interface for management of epileptic patients. New system allows to organise patients into studies, set access rights for users and studies and file upload through web interface. Flags can be assigned to uploaded files. Some files can be visualised on an interactive timeline.

Keywords database of patients, data visualisation, timeline, access rights, file management

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Výskumná skupina ISARG[1] už niekoľko rokov zhromažďuje dáta o detských a dospelých pacientoch, zaradených do epileptochirurgického programu Centra pre epilepsiu nemocnice Motol. Tieto dáta sú bezprostredne využívané celou skupinou, ktorá zahrňuje výskumné týmy z ČVUT FEL, FN Motol a Miami Children Hospital. Vyhodnocovanie týchto dát slúži k výskumu nových epileptologických biomarkerov.

Pre výskum biomarkerov je potrebné efektívne organizovať a zdieľať dáta o pacientoch medzi pracoviskami. Objem dát počas doby fungovania skupiny ISARG narástol do takej miery, že súčasné riešenie tohto problému sa stalo nedostatočné.

Medzi požiadavky na nový systém patrí najmä vizualizácia vybraných typov súborov, organizácia pacientov do štúdií a nastavovanie prístupových práv pre jednotlivých pracovníkov alebo štúdie.

[1] Intracranial Signal & Research Group: `http://isarg.feld.cvut.cz/`

Kapitola 1

Cieľ práce

Cieľom tejto práce je analýza súčasného informačného systému pre správu databázy epileptických pacientov a následný návrh a implementácia nového systému, spolu s užívateľským rozhraním pre efektívne zdieľanie dát. Výstupom tejto práce je informačný systém, ktorý bol po dôslednom otestovaní nasadený na webový server skupiny ISARG. Medzi primárne požiadavky na nový systém patrí najmä:

1. Organizácia pacientov do štúdií.
2. Správa súborov cez webové rozhranie. K jednotlivým súborom sa dajú nastavovať príznaky a množinu týchto príznakov je možné upravovať.
3. Mechanizmus riadenia prístupových práv pre užívateľov, ako aj pre prístup k jednotlivým štúdiám.
4. Vizualizácia vybraných typov súborov na časovej osi. Môže ísť o obrázkové súbory, záznamy elektrofyziologických vyšetrení, apod.

Kapitola 2

Popis problému

2.1 Súčasný stav riešenia problematiky

V súčasnosti využíva skupina ISARG pre zdieľanie dát medzi pracoviskami jednoduchý FTP server, kam ukladá dáta pacientov. Tieto dáta sú tvorené prevažne súbormi výstupov z elektród, ktoré boli pacientom zavadené do mozgu (ECoG), záznamami z EEG, multimedálnymi súbormi, skriptami pre vyhodnocovanie dát, atď. Súbory sú organizované pomocou súborového systému do adresárov, kde každý pacient má svoje súbory uložené v samostatnom adresári.

2.2 Existujúce možnosti riešenia

2.2.1 Sieťové súborové úložisko

Z technického pohľadu je možné sieťové úložisko realizovať pomocou fyzického serveru, pripojeného do siete alebo pomocou špecializovaného súborového serveru NAS. K takémuto serveru sa dá pripájať viacerými spôsobmi – môže ísť o v súčasnosti používaný FTP protokol, SFTP, zdieľaný adresár cez SMB, WebDav, atď.

Ďalším spôsobom je využitie cloud súborového úložiska. Výhodou takéhoto riešenia je jednoduchá škálovateľnosť a takisto odpadá nutnosť prevádzkovať fyzický hardvér, s čím často súvisí aj jeho komplikovaná údržba. Pred použitím takéhoto riešenia je ale nutné zvážiť bezpečnostné otázky. Príkladom cloud úložiska je napríklad Amazon S3[2], Microsoft Azure[3] alebo Dropbox[4].

Aj keď ide o rôzne technológie, z pohľadu potrieb užívateľa neposkytuje žiadna z nich výrazné benefity, oproti súčasnému riešeniu. Kvôli potrebe rozšíriť systém o ďalšie funkcie je udržiavanie informácií o pacientoch iba pomocou sieťového súborového úložiska nedostatočné.

[2] Amazon Simple Storage Service (S3): `http://aws.amazon.com/s3/`
[3] Microsoft Azure Storage: `http://azure.microsoft.com/en-gb/services/storage/`
[4] Dropbox: `http://www.dropbox.com/`

2.2.2 Systémy na správu pacientov

Existuje množstvo systémov na správu pacientov, ktoré využívajú nemocnice a lekári pre udržiavanie elektronickej evidencie pacientov a ich elektronickej zdravotnej karty. Niektoré z takýchto systémov je možné používať zadarmo (napríklad systém GNU Health[5]), väčšinou však ide o proprietárne aplikácie. Pre nasadenie takéhoto systému by ho bolo nutné upraviť podľa požiadavok na nový systém (kapitola 3.1), čo by muselo byť prevedené odborníkom na daný systém, prípadne priamo spoločnosťou, ktorá daný software vyvíja. Vysoké vstupné náklady by boli pravdepodobne tiež prekážkou pre nasadenie takéhoto systému.-.

2.3 Zvolené riešenie

Ako bolo povedané v predchádzajúcej sekcii, existuje široké spektrum rôznych aplikácii pre zdieľanie dát a súborov. Takisto existujú aj aplikácie na správu pacientov zdravotníckeho zariadenia. Nový systém ale musí splňovať špecifické požiadavky, zadefinované v kapitole 3.1, kvôli ktorým sa stávajú existujúce riešenia nevhodné a je nutné navrhnúť a implementovať riešenie na mieru.

Motiváciou pre implementáciu nového systému na mieru je z časti aj fakt, že práca na tomto systéme začala už v rámci týmového softwarového projektu na predmete BI-SP2 na Fakulte informačných technológií. Táto práca je teda pokračovaním snahy o dokončenie tohto systému až do finálnej fázy a nasadenie systému do praxe.

V nasledujúcej časti budem diskutovať niektoré aspekty nového systému, ktoré si vyžadujú detailný popis.

2.3.1 Ukladanie súborov

Pri návrhu nového systému som vychádzal z faktu, že väčšina užívateľov je v súčasnosti zvyknutá nahrávať a organizovať súbory pomocou FTP serveru. V prípade, že by som týchto užívateľov pripravil o možnosť nahrávať súbory cez FTP protokol, skomplikovalo by to celý proces zavedenia nového systému do praxe. Takisto niektoré skripty a vizualizačné nástroje používané výskumnou skupinou komunikujú cez FTP protokol, čo by v konečnom dôsledku viedlo k nutnosti meniť nastavenia týchto nástrojov a upravovať používané skripty. Požiadavky na nový systém ale hovoria o webovom rozhraní, cez ktoré sa budú dať spravovať súbory, nastavovať práva a organizovať pacientov do štúdii.

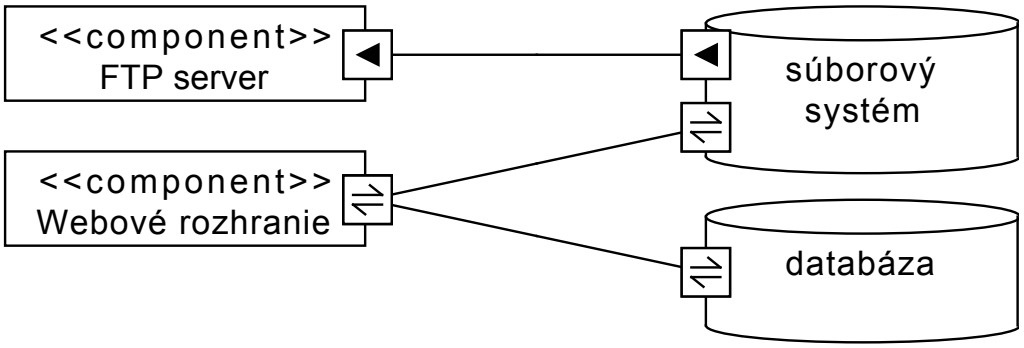
Po diskusii s vedúcim tejto práce sme teda zvolili riešenie, ktoré umožňuje skombinovať oba prístupy – teda ponechať konvenčný a rýchly spôsob nahrávania súborov cez FTP, rovnako ako využívať rozšírené funkcie, ktoré prináša webové rozhranie. Bolo teda potrebné navrhnúť spôsob, ako spravovať súbory pomocou pomocou dvoch rôznych rozhraní bez vzniku duplicít.

[5] GNU Health: `http://health.gnu.org/`

2.3.1.1 Naivné riešenie ukladania súborov

Ako naivné riešenie sa ponúka organizovať súbory do adresárov pomocou súborového systému, na ktorý by bol priamo namapovaný nejaký existujúci FTP server a zároveň by do neho mala prístup aj webová aplikácia. Medzi požiadavky na systém ale patrí aj možnosť ukladať príznaky pre jednotlivé súbory, čo žiadny bežný súborový systém neumožňuje. Tento problém by sa dal z časti vyriešiť tak, že by sa záznam o umiestnení súborov v súborovom systéme ukladal do databázy a príznaky sa nastavovali vzhľadom k tomuto záznamu. Takéto riešenie ale vedie k ukladaniu dvoch duplicitných záznamov o jednom súbore, čo by konečnom dôsledku mohlo viesť k nekonzistencií medzi dátami v súborovom systéme a databázou.

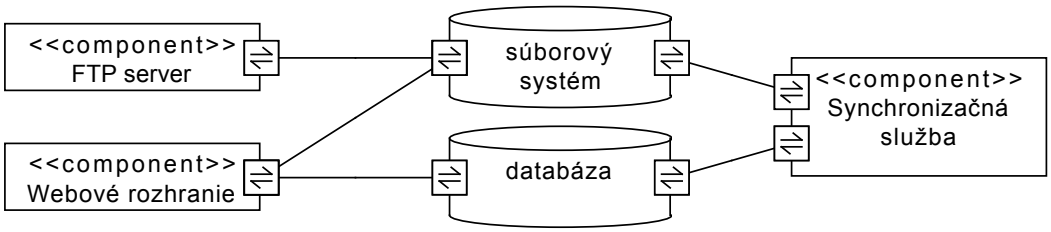
Najjednoduchším spôsobom ako zabrániť nekonzistencií dát by bolo umožniť pristupovať k súborom cez FTP rozhranie iba na čítanie tak, ako je znázornené na obrázku 2.1. Vytvárať, mazať alebo premiestňovať súbory by mohla iba webová aplikácia, ktorá by pri každej takejto operácii upravila záznamy v databáze.



Obr. 2.1: UML Composite Structure: Naivné riešenie ukladania súborov 1

Ak by sme ale chceli mať možnosť vytvárať a mazať súbory pomocou FTP protokolu, môžeme napríklad zaviesť nejakú formu synchronizačnej služby, ktorá by v pravidelných časových intervaloch vzniknuté nekonzistencie riešila (obrázok 2.2). Ak napríklad užívateľ zmaže súbor cez FTP, synchronizačná služba by sa postarala o to, aby sa zmazali aj príslušné záznamy o súbore z databázy.

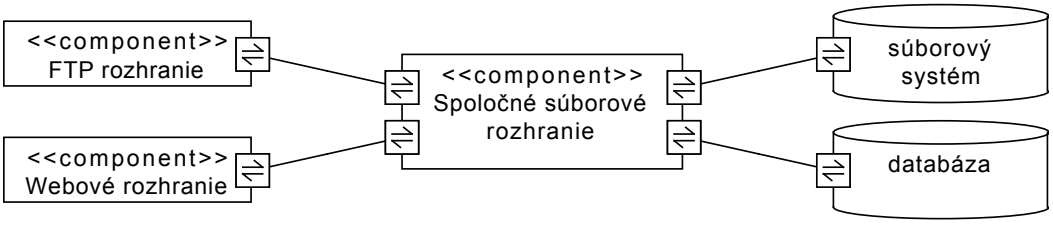
Je ale zrejmé, že takéto riešenie by nebolo veľmi spoľahlivé. Ak napríklad užívateľ presunie súbor pomocou FTP príkazu RNTO, z pohľadu synchronizačnej služby by išlo o dve samostatné operácie - vymazanie súboru a vytvorenie nového súboru. Tým by sa stratili príznaky, ktoré boli tomuto súboru pridelené pomocou webového rozhrania.



Obr. 2.2: UML Composite Structure: Naivné riešenie ukladania súborov 2

2.3.1.2 Zvolené riešenie ukladania súborov

Zvolené riešenie umožňuje spoľahlivý spôsob prístupu k súborom cez FTP aj webové rozhranie s možnosťou zápisu bez vzniku nekonzistencií medzi súborovým systémom a databázou. Podstatou zvoleného riešenia je jedno spoločné súborové rozhranie, ktoré predstavuje abstrakciu nad súborovým systémom a datababázou a bude poskytovať všetky operácie so súbormi. FTP aj webové rozhranie teda nebudú komunikovať s dátovým úložiskom priamo, ale iba cez toto súborové rozhranie.



Obr. 2.3: UML Composite Structure: Zvolené riešenie ukladania súborov

Nevýhodou zvoleného riešenia je, že nie je možné triviálnym spôsobom použiť nejaký existujúci FTP server a namapovať ho priamo na súborový systém. Je nutné vytvoriť vlastný alebo upraviť existujúci FTP server tak, aby dokázal komunikovať cez toto súborové rozhranie.

2.3.2 Zobrazenie súborov na časovej osi

Súbory nahrávané užívateľom systému pre jednotlivých pacientov sú rôznych typov. Môžu to byť záznamy z ECoG alebo EEG elektród, tabuľky, multimediálne súbory, textové dokumenty alebo rôzne skripty. Tieto súbory je potrebné umiestniť na interaktívnu časovú os, podľa dátumu kedy boli zaznamenané. V prípade niektorých typov súborov je časová informácia obsiahnutá priamo v súbore. Tento typ súborov sa na časovej osi zobrazí ako časový interval, počas ktorého bol pacient pozorovaný. Ostatné súbory sú na

časovú os pridávané podľa dátumu nahratia súboru do systému, čo ale nemusí zodpovedať dátumu vytvorenia takéhoto súboru. Preto bude užívateľ môcť meniť umiestnenie tohto súboru na časovej osi.

2.3.3 Správa pacientov

Pacientov je možné organizovať do štúdií, pričom platí, že jeden pacient môže byť zaradený do viacerých štúdií a zároveň jedna štúdia môže obsahovať viacero pacientov. Systém rozoznáva kód štúdie a kód pacienta zaradeného do danej štúdie. V systéme je možné pri každej štúdii nastaviť, ktorí užívatelia majú do nej prístup. Platí, že užívateľ môže pristupovať k pacientom a ich súborom, iba ak je takýto pacient zaradený do štúdie ku ktorej má nastavený prístup.

Kapitola 3

Analýza

3.1 Analýza požiadavok

Táto sekcia definuje požiadavky na nový systém. Požiadavky špecifikujú očakávané funkcie a charakteristiky systému, nezávisle od konkrétnej realizácie alebo vnútornej štruktúry daného systému. Existujú dva druhy požiadavok – funkčné a nefunkčné. Funkčné požiadavky zachycujú povahu interakcie medzi systémom a prostredím. Nefunkčné požiadavky obmedzujú typy riešení, nad ktorými sa dá uvažovať. [1, s. 15]

Po zadefinovaní požiadavok na nový systém sa funkčné požiadavky ďalej modelujú v jednotlivých prípadoch užitia.

3.1.1 Funkčné požiadavky

F.1 Kontrola prístupu.

F.1.1 Autentizácia.

F.1.1.1 Prihlásenie s možnosťou zapamätania identity užívateľa.

F.1.1.2 Odhlásenie užívateľa.

F.1.2 Autorizácia na základe nastavených práv pre konkrétneho užívateľa.

F.2 Správa užívateľov.

F.2.1 Aplikácia umožňuje nastaviť prístupové meno a heslo pre užívateľa.

F.2.2 Každému uživateľovi je možné nastaviť užívateľské práva. Práva sú organizované do stromovej štruktúry.

F.2.3 Užívateľovi je možné nastaviť osobné údaje (meno, priezvisko, rodné číslo, pohlavie), adresu (ulica, mesto, PSČ, krajina) a kontakt (telefónne číslo a email).

F.2.4 V prípadne pokusu o uloženie nepovolenej hodnoty vo formulári sa zobrazí chybová hláška a farebne sa zvýraznia polia s chybnou hodnotou.

F.2.5 Zoznam užívateľov je možné zobraziť v tabuľke, v ktorej sa dá vyhľadávať a usporiadavať podľa zvolených kritérií. V tabuľke je možné nastaviť počet záznamov na stránku a v prípade, že je celkový počet záznamov väčší ako táto hodnota, je možné v tabuľke stránkovať.

F.2.6 Operácie s užívateľmi.

F.2.6.1 Vytvorenie užívateľa.

F.2.6.2 Úprava užívateľa.

F.2.6.3 Zmazanie užívateľa. Zmazanie užívateľa je nutné potvrdiť, aby sa zabránilo náhodnému zmazaniu užívateľa.

F.3 Správa štúdií

F.3.1 Aplikácia umožňuje nastaviť názov, kód a skupinu príslušnej štúdie.

F.3.2 Zoznam štúdií je možné zobraziť v tabuľke, v ktorej sa dá vyhľadávať a usporiadavať podľa zvolených kritérií. V tabuľke je možné nastaviť počet záznamov na stránku a v prípade, že je celkový počet záznamov väčší ako táto hodnota, je možné v tabuľke stránkovať.

F.3.3 K štúdiám je možné nastavovať oprávnenia pre užívateľov systému. Užívateľ, ktorý nemá právo pristupovať ku všetkým štúdiám vidí iba pacientov a ich súbory, ktoré boli zaradené do štúdie, ku ktorej má oprávnenie.

F.3.4 Operácie so štúdiami.

F.3.4.1 Vytvorenie štúdie.

F.3.4.2 Úprava štúdie.

F.3.4.3 Zmazanie štúdie. Zmazanie štúdie je nutné potvrdiť, aby sa zabránilo náhodnému zmazaniu štúdie.

F.4 Správa príznakov

F.4.1 Aplikácia umožňuje nastaviť názov každého príznaku.

F.4.2 Zoznam príznakov je možné zobraziť v tabuľke, v ktorej sa dá vyhľadávať a usporiadavať podľa zvolených kritérií. V tabuľke je možné nastaviť počet záznamov na stránku a v prípade, že je celkový počet záznamov väčší ako táto hodnota, je možné v tabuľke stránkovať.

F.4.3 Operácie s príznakmi.

F.4.3.1 Vytvorenie príznaku.

F.4.3.2 Úprava príznaku.

F.4.3.3 Zmazanie príznaku. Zmazanie príznaku je nutné potvrdiť, aby sa zabránilo náhodnému zmazaniu príznaku.

F.5 Správa pacientov

F.5.1 Pacientovi je možné nastaviť osobné údaje (meno, priezvisko, rodné číslo, pohlavie), adresu (ulica, mesto, PSČ, krajina) a kontakt (telefónne číslo a email).

F.5.2 Zoznam pacientov je možné zobraziť v tabuľke, v ktorej sa dá vyhľadávať a usporiadavať podľa zvolených kritérií. V tabuľke je možné nastaviť počet záznamov na stránku a v prípade, že je celkový počet záznamov väčší ako táto hodnota, je možné v tabuľke stránkovať.

F.5.3 Operácie s pracientami.

F.5.3.1 Vytvorenie pacienta.

F.5.3.2 Úprava pacienta.

F.5.3.3 Zmazanie pacienta. Zmazanie pacienta je nutné potvrdiť, aby sa zabránilo náhodnému zmazaniu pacienta.

F.5.4 Pacienta je možné priradiť do štúdie. Užívateľ, ktorý nemá právo prehliadať všetky štúdie vidí iba pacientov zaradených do štúdie, ku ktorej má oprávnenie pristupovať.

F.5.5 Pacientovi je možné nahrávať súbory.

F.5.5.1 Štandardné súborové operácie cez webové rozhranie.

F.5.5.2 Štandardné súborové operácie cez FTP rozhranie.

F.5.5.3 Aplikácia umožňuje nastavovať príznaky k súborom cez webové rozhranie.

F.5.5.4 Pri každom súbore sa zaznamenáva čas vloženia súboru a MIME typ.

F.5.6 Zobrazenie súborov na časovej osi.

F.5.6.1 Automatické vytváranie náhľadov k obrázkovým súborom.

F.5.6.2 Súbory typu EDF a STS/SIG sa na časovej osi zobrazujú ako časový interval, ktorý je získaný z tohto súboru. Súbory typu DCM sa na časovej osi zobrazujú v čase, ktorý bol získaný z tohto súboru. Ostatné súbory sa na časovej osi zobrazujú v čase, v ktorom boli nahraté do systému.

F.5.6.3 Umiestnenie súborov na časovej osi môže užívateľ meniť.

3.1.2 Nefunkčné požiadavky

N.1 Aplikácia je správne nakonfigurovaná a nasadená na server podľa užívateľskej príručky.

N.2 Front-end aplikácie je dostupný cez webový prehliadač alebo pomocou FTP klienta na zariadení so sieťovým prístupom k serveru (napr. pomocou siete Internet).

N.3 Užívateľ požíva moderný webový prehliadač v štandardnom nastavení, t.j. prehliadač musí prejsť Acid3 testom [2].

N.4 Užívateľ používa FTP klienta, ktorý korektne implementuje RFC 765 [3], RFC 959 [4] a RFC 2228 [5].

N.5 Serverová časť beží na operačnom systéme Debian 7 v servletovom kontajneri Apache Tomcat 7 a s použitím relačnej databázy PostgreSQL 9.

N.6 Serverová časť aplikácie vyžaduje aspoň 1GB RAM.

N.7 Administrátor serveru zabezpečí, aby bolo na serveri vždy dostatok voľného miesta na disku pre zápis pacientových súborov. Odporúčané minimum je aspoň 100 GB.

3.2 Modelovanie prípadov užitia

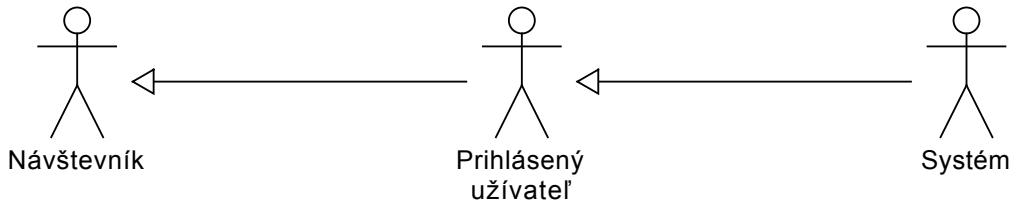
Modelovanie prípadov užitia je forma inžinierstva požiadavok. Tento proces typicky zahrňuje tieto tri kroky: 1) nájdenie hraníc systému; 2) nájdenie aktérov; 3) špecifikácia prípadov užitia alebo scenárov. Výstupom tohto procesu je model prípadov užitia, zložený z týchto štyroch častí: [6, s. 57]

- aktéri – role ľudí alebo vecí, ktoré používajú systém;
- prípady užitia – časti funkcionality systému, ktoré využíva aktér;
- vzťahy medzi aktérmi a prípadmi užitia;
- hranice systému – ohraničenie prípadov užitia a zadefinovanie, čo patrí do modelovaného systému.

Aby som sa vyhol prílišnému opakovaniu v jednotlivých modeloch, používam v nich stereotyp `«include»`. Tento stereotyp je druh vzťahu medzi dvoma prípadmi užitia, ktorý umožňuje použiť rovnaký spôsob užitia na viacerých miestach. Ďalší stereotyp, ktorý používam je typu `«extend»`. Tento druh vzťahu rozširuje už existujúci prípad užitia o nové správanie. Inými slovami, prípady užitia vo vzťahu typu `«include»` sú priamou súčasťou nadradeného prípadu užitia, kým prípady užitia rozšírené o ďalšie prípady užitia stereotypom `«extend»` fungujú nezávisle na týchto rozšírených prípadoch užitia. [6, s. 84-85] Podľa odporúčaní v [6, s. 91] je vhodné sa vyhýbať príliš častému používaniu týchto stereotypov, keďže zhoršujú čitateľnosť modelu užitia. Tieto stereotypy je vhodné používať iba v prípadoch, keď ich použitie zjednodušuje model a robí ho zrozumiteľnejší.

3.2.1 Aktéri systému

Systém rozoznáva troch aktérov – návštevníka, prihláseného užívateľa a systém. Jediná operácia dostupná návštevníkovi je prihlásenie. Ostatné funkcie systému sú dostupné iba pre autentizovaných užívateľov a systém. Prístup užívateľa k jednotlivým operáciam podlieha autorizácii na základe užívateľských oprávnení.



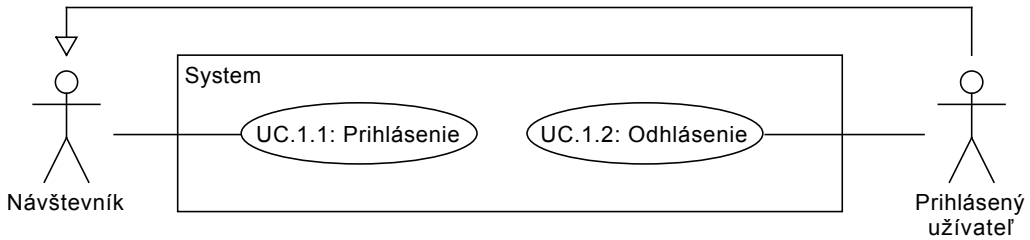
Obr. 3.1: UML Use Case: Aktéri systemu

3.2.2 Prípady užitia

V tejto sekcii sú zadefinované jednotlivé prípady užitia, ktoré môžu vykonávať aktéri systému a vzťahy medzi aktérmi a prípadmi užitia. Takisto sú tu zadefinované hranice systému, v ktorých sa nachádzajú tieto prípady užitia. Tieto prípady užitia pokrývajú funkčné požiadavky zadefinované v sekcii 3.1.1.

3.2.2.1 Prihlásenie a odhlásenie

Tento model prípadov užitia pokrýva funkčné požiadavky **F.1.1**. Návštevník sa v tomto modeli môže prihlásiť, čím sa stane prihláseným užívateľom, ktorý sa následne môže odhlásiť. Prípady užitia sú znázornené na diagrame 3.2.

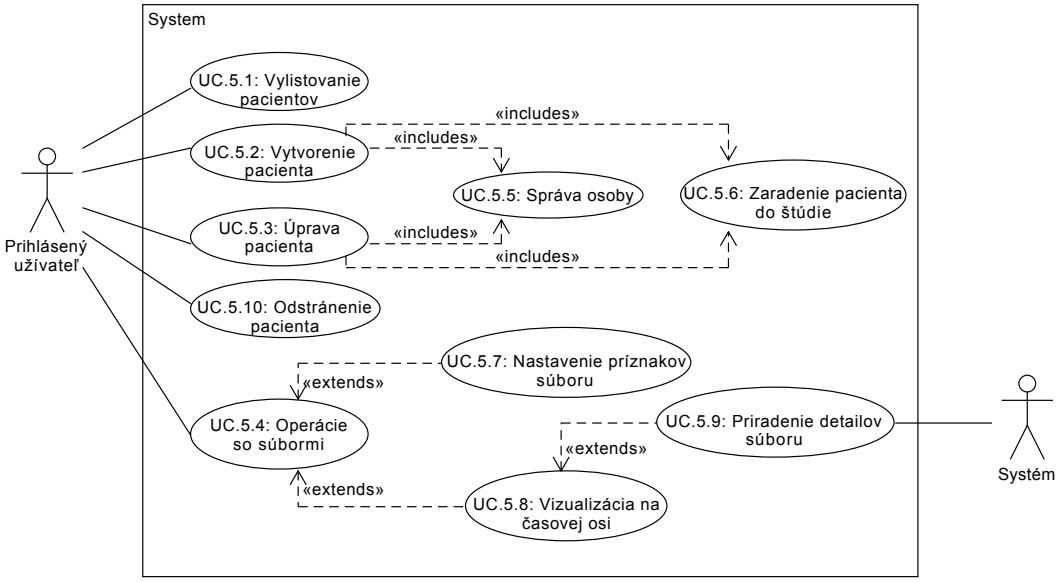


Obr. 3.2: UML Use Case: Prihlásenie a odhlásenie

3.2.2.2 Správa pacientov

Tento model prípadov užitia pokrýva funkčné požiadavky **F.5**. Prihlásený užívateľ môže v tomto modeli vylistovať zoznam pacientov do tabuľky, v ktorej je možné filtrovať alebo usporiadavať záznamy. Ďalej môže vytvárať, upravovať alebo odstraňovať pacientov. K pacientom je možné spravovať súbory, ku ktorým sa dajú následne priradiť príznaky. Nahraté súbory je možné vizualizovať na časovej osi na základe priradených detailov súboru. Tento detail súboru je typicky napríklad náhľad nahratého obrázku, ktorý automaticky vygeneruje systém alebo extrahovaná časová informácia zo súboru. Časovú

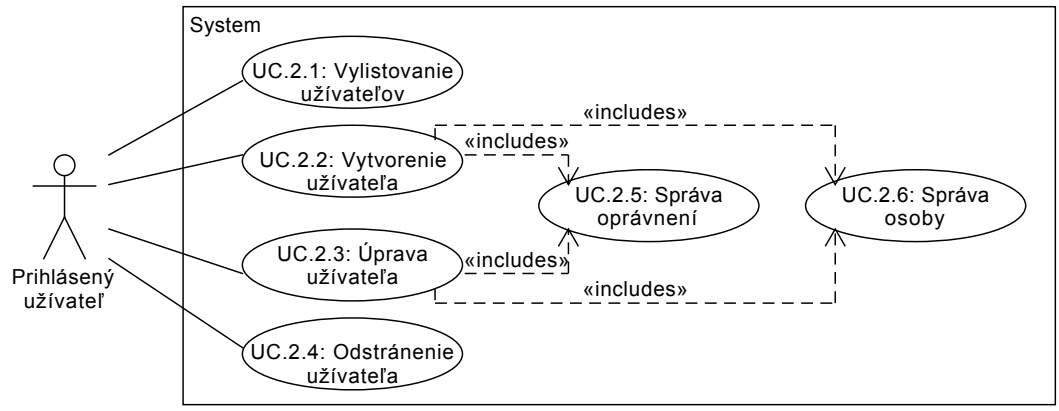
informáciu môže následne upravovať aj prihlásený užívateľ. Prípady užitia sú znázornené na diagrame 3.3.



Obr. 3.3: UML Use Case: Správa pacientov

3.2.2.3 Správa užívateľov

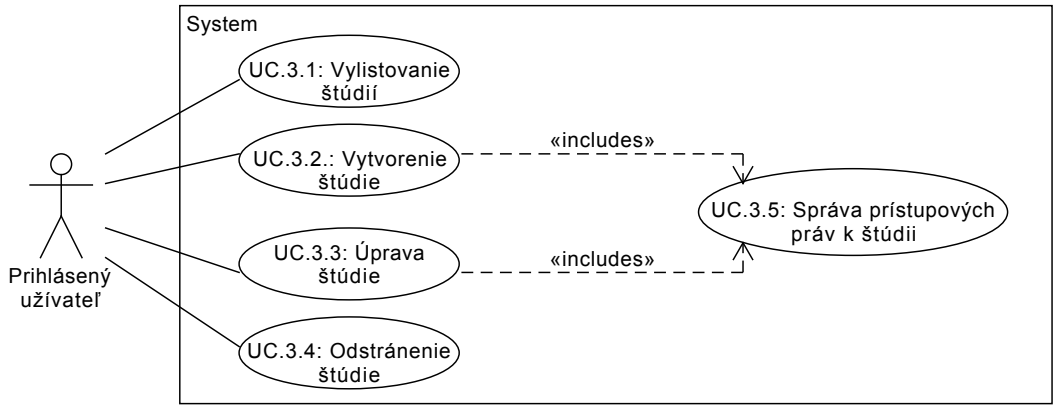
Tento model prípadov užitia pokrýva funkčné požiadavky **F.2**. Prihlásený užívateľ môže v tomto modeli vylistovať zoznam užívateľov do tabuľky, v ktorej je možné filtrovať alebo usporiadavať záznamy. Ďalej môže vytvárať, upravovať alebo odstraňovať užívateľov. Prípady užitia sú znázornené na diagrame 3.4.



Obr. 3.4: UML Use Case: Správa užívateľov

3.2.2.4 Správa štúdií

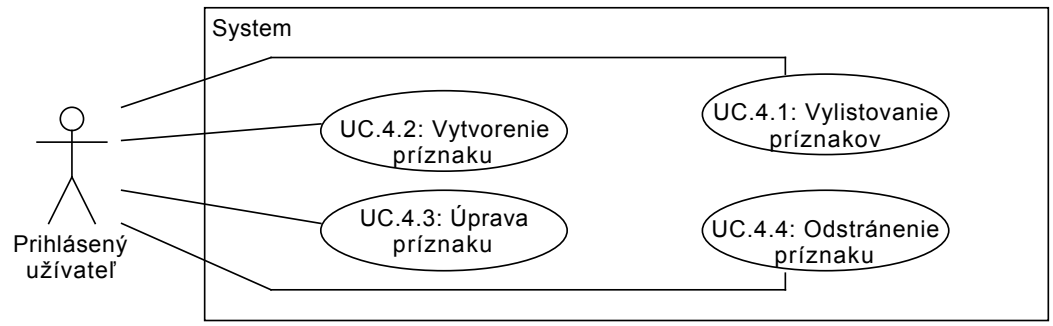
Tento model prípadov užitia pokrýva funkčné požiadavky **F.3**. Prihlásený užívateľ môže v tomto modeli vylistovať zoznam štúdií do tabuľky, v ktorej je možné filtrovať alebo usporiadavať záznamy. Ďalej môže vytvárať, upravovať alebo odstraňovať štúdie. K štúdiám je možné nastavovať prístupové práva pre jednotlivých užívateľov systému. Prípady užitia sú znázornené na diagrame 3.5.



Obr. 3.5: UML Use Case: Správa štúdií

3.2.2.5 Správa príznakov

Tento model prípadov užitia pokrýva funkčné požiadavky **F.4**. Prihlásený užívateľ môže v tomto modeli vylistovať zoznam príznakov do tabuľky, v ktorej je možné filtrovať alebo usporiadavať záznamy. Ďalej môže vytvárať, upravovať alebo odstraňovať príznaky. Prípady užitia sú znázornené na diagrame 3.6.



Obr. 3.6: UML Use Case: Správa príznakov

3.3 Kontrola splnenia požiadavok

V tabuľke 3.1 je ukázané, ako jednotlivé prípady užitia pokrývajú funkčné požiadavky na systém. V riadkoch tabuľky sú uvedené funkčné požiadavky zadefinované v sekcii 3.1.1, v stĺpcoch sú prípady užitia zo sekcie 3.2.

	UC.1.1	UC.1.2	UC.2.1	UC.2.2	UC.2.3	UC.2.4	UC.2.5	UC.2.6	UC.3.1	UC.3.2	UC.3.3	UC.3.4	UC.3.5	UC.4.1	UC.4.2	UC.4.3	UC.4.4	UC.5.1	UC.5.2	UC.5.3	UC.5.4	UC.5.5	UC.5.6	UC.5.7	UC.5.8	UC.5.9	UC.5.10
F.1.1.1	✓																									
F.1.1.2		✓																								
F.2.1				✓	✓																					
F.2.2							✓																			
F.2.3								✓																		
F.2.4				✓	✓																					
F.2.5			✓																							
F.2.6.1				✓																						
F.2.6.2					✓																					
F.2.6.3						✓																				
F.3.1									✓	✓																
F.3.2								✓																		
F.3.3												✓														
F.3.4.1									✓																	
F.3.4.2										✓																
F.3.4.3											✓															
F.4.1														✓	✓											
F.4.2													✓													
F.4.3.1														✓												
F.4.3.2															✓											
F.4.3.3																✓										
F.5.1																					✓					
F.5.2																	✓									
F.5.3.1																		✓								
F.5.3.2																			✓							
F.5.3.3																										✓
F.5.4																						✓				
F.5.5.1																				✓						
F.5.5.2																				✓						
F.5.5.3																							✓			
F.5.5.4																									✓	
F.5.6.1																									✓	
F.5.6.2																								✓	✓	
F.5.6.3																									✓	

Tabuľka 3.1: Pokrytie funkčných požiadavok prípadmi užitia

3.4 Modelovanie procesov

Pre znázornenie workflow niektorých prípadov užitia, ktoré boli zadefinované v 3.2 budem v tejto sekcii používať vždy textový popis v bodoch spolu s activity diagramom pre lepšie znázornenie procesu.

3.4.1 Základné CRUD operácie

CRUD je akronym používaný pre popis štyroch základných operácii – ***c**reate* (vytvorenie), ***r**ead* (čítanie, vylistovanie), ***u**pdate* (úprava), ***d**elete* (odstránenie). Vzhľadom na to, že workflow týchto operácii je v niektorých prípadoch užitia veľmi podobný, sú tieto operácie popísané všeobecne pre všetky prípady užitia, ktoré takúto operáciu používajú.

3.4.1.1 Vytvorenie a editácia

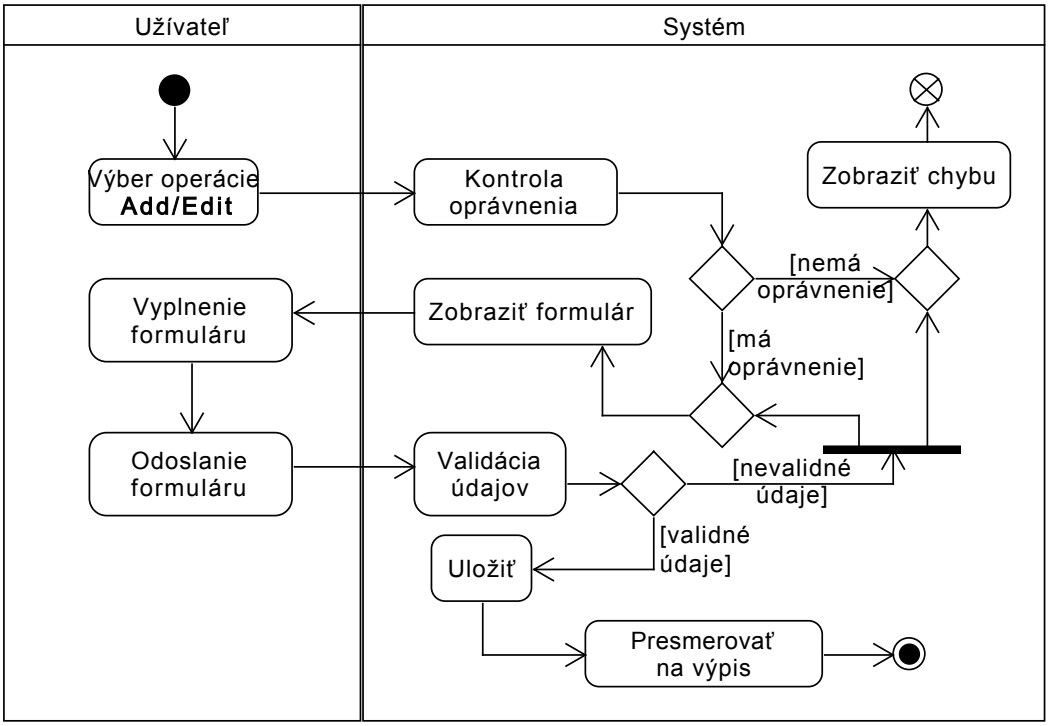
Postupnosť týchto krokov je znázornená v activity diagrame 3.7.

1. Užívateľ klikne na tlačítko `Add` pre pridanie nového záznamu, prípadne `Edit` pre úpravu existujúceho záznamu.
2. Na serveri prebehne autorizácia užívateľa a kontrola, či má užívateľ dostatočné práva pre vytváranie resp. úpravu záznamov.
3. Ak užívateľ vybral operáciu `Edit`, prebehne kontrola, či daný záznam existuje. Ak záznam existuje, zobrazí jeho údaje vo formulári. Ak neexistuje, zobrazí chybovú hlášku. Kvôli zjednodušeniu nie je tento krok zakreslený v diagrame 3.7.
4. Ak má užívateľ dostatočné oprávnenie, zobrazí sa mu formulár pre zadanie údajov. Ak dostatočné oprávnenie nemá, zobrazí sa mu chybová hláška.
5. Užívateľ vyplní a odošle formulár.
6. Ak sú všetky údaje, ktoré boli zadané vo formulári validné, server uloží záznam a presmeruje užívateľa na výpis všetkých záznamov. Ak údaje neboli validné, zobrazí sa nad formulárom chybová hlášku.

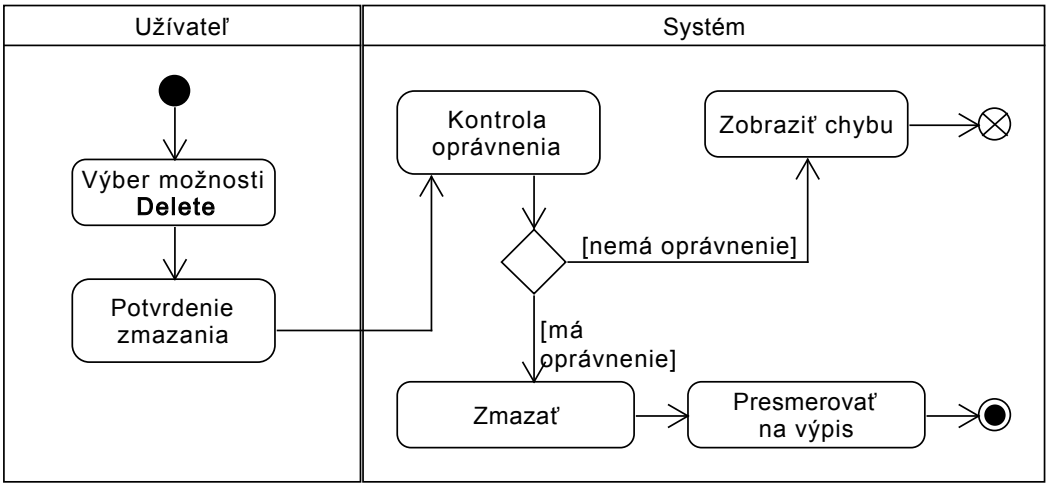
3.4.1.2 Mazanie

Postupnosť týchto krokov je znázornená v activity diagrame 3.8.

1. Užívateľ klikne na tlačítko `Delete` pri príslušnom zázname.
2. Užívateľovi vyskočí potvrdzovací dialóg, ktorý musí potvrdiť, aby sa zabránilo chybnému zmazaniu záznamu.
3. Na serveri prebehne autorizácia užívateľa a kontrola, či má užívateľ dostatočné práva pre mazanie záznamov.
4. Na serveri prebehne kontrola, či zvolený záznam exstuje. Ak neexistuje, zobrazí sa chybová hláška. Kvôli zjednodušeniu nie je tento krok zakreslený v diagrame 3.8.
5. Ak má užívateľ dostatočné oprávnenie, záznam sa zmaže a užívateľ bude presmerovaný na výpis všetkých záznamov. Ak dostatočné oprávnenie nemá, zobrazí sa mu chybová hláška.



Obr. 3.7: UML Activity: CRUD operácia create a update



Obr. 3.8: UML Activity: CRUD operácia delete

3.4.1.3 Vylistovanie záznamov

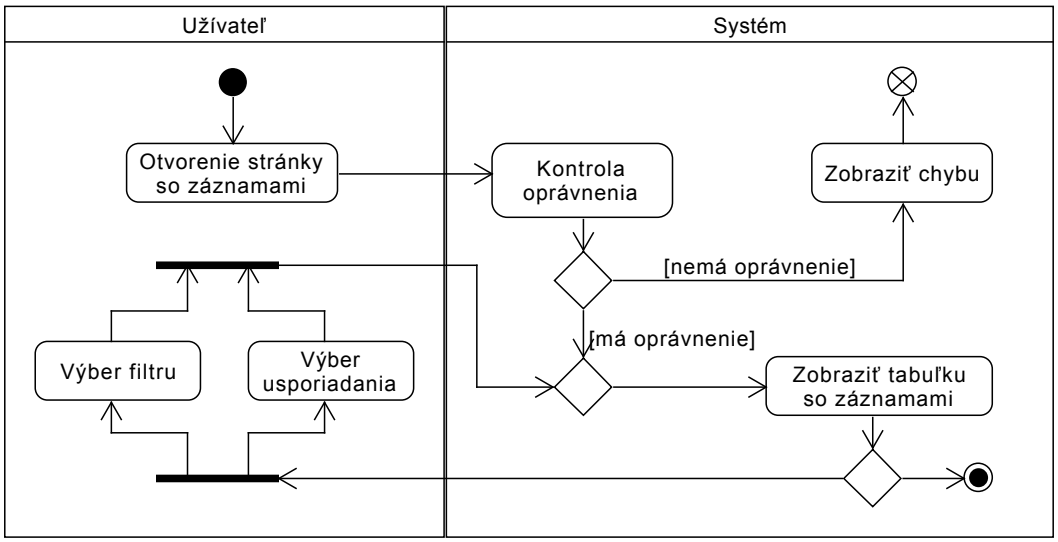
Postupnosť týchto krokov je znázornená v activity diagrame 3.9.

1. Užívateľ si v menu vyberie, ktorú stránku so záznamami chce otvoriť (pacienti, štúdie, príznaky, ...).

2. Na serveri prebehne autorizácia užívateľa a kontrola, či má užívateľ dostatočné práva pre zobrazenie záznamov.

3. Ak má užívateľ dostatočné oprávnenie, zobrazí sa mu tabuľka s vylistovanými záznamami. Ak dostatočné oprávnenie nemá, zobrazí sa mu chybová hláška.

4. V tabuľke je užívateľovi umožnené filtrovať a usporiadavať záznamy podľa rôznych kritérií.



Obr. 3.9: UML Activity: CRUD operácia read

3.4.2 Ďalšie operácie

Okrem základných CRUD operácií, ktoré boli popísané v predchádzajúcej sekcii sú v systéme ďalšie operácie, ktoré si vyžadujú detailný popis. Medzi tieto operácie patri správa súborov a ich vizualizácia na časovej osi.

3.4.2.1 Správa súborov

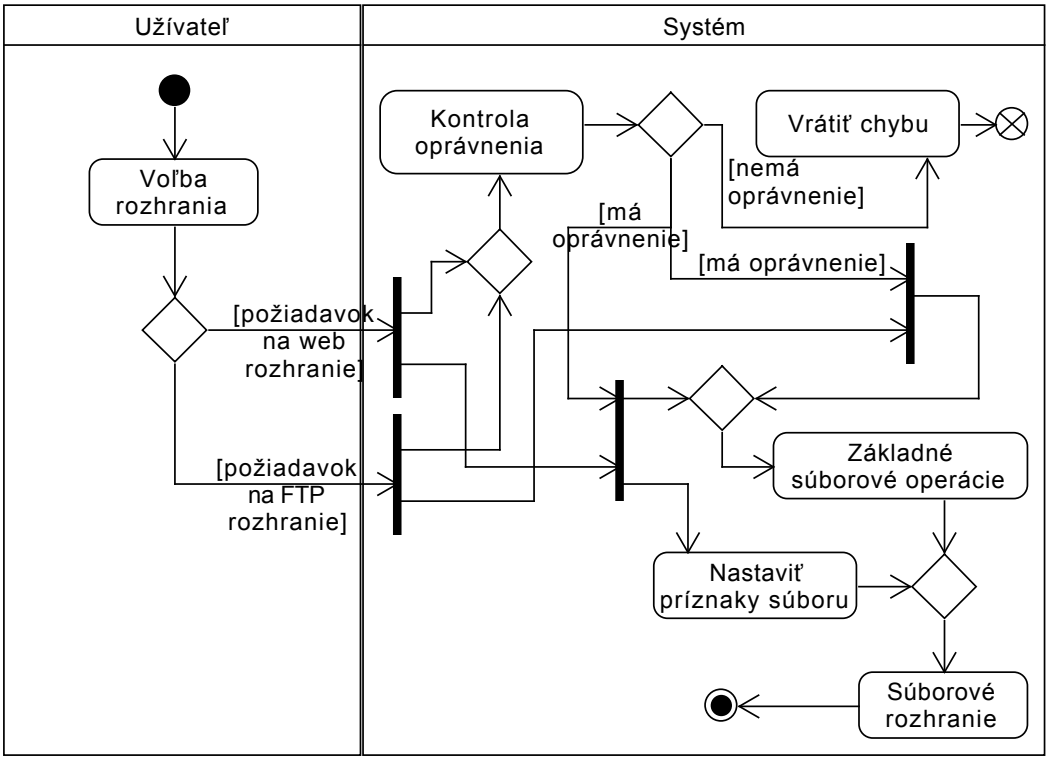
Postupnosť týchto krokov je znázornená v activity diagrame 3.10.

1. Užívateľ má dve možnosti, akým spôsobom chce spravovať súbory. V prvom kroku si teda zvolí, či chce súbory spravovať cez FTP protokol alebo pomocou webového prehliadača a pošle požiadavok na server.

2. Na serveri prebehne autorizácia užívateľa a kontrola, či má užívateľ dostatočné práva pre správu súboru (resp. pre požiadavku, ktorú užívateľ zaslal).

3. Ak má užívateľ dostatočné oprávnenie a prichádza cez FTP rozhranie, sú mu umožnnené základné súborové operácie (`create`, `upload`, `move`, `rename`, `download`, `delete`). Ak má užívateľ dostatočné oprávnenie a prichádza cez webové rozhranie, je mu okrem základných súborových operácií umožnené aj nastavovať príznaky k súborom. Ak dostatočné oprávnenie nemá, je mu vrátená chybová hláška.

4. Príslušnú súborovú operáciu vykoná súborové rozhranie, ktoré užívateľovi vráti odpoveď (napr. zoznam súborov, výsledok súborovej operácie, prípadne chybu, apod.).



Obr. 3.10: UML Activity: Správa súborov

3.5 Doménový model

V tejto sekcii popisujem entity, ktoré systém rozoznáva. Na class diagrame 3.11 sú vidieť vzťahy medzi jednotlivými entitami. Okrem toho ku každej entite uvádzam textový popis pre vysvetlenie jej významu a tabuľku s popisom atribútov danej entity. Pri popise atribútov uvádzam aj platformovo špecifický typ atribútu a integritné obmezenia, ktoré však už formálne patria do návrhu systému.

3.5.1 Popis entít doménového modelu a ich atribútov

3.5.1.1 Entita `Person`

Predstavuje abstraktnú osobu, ktorá môže byť užívateľom alebo pacientom. Spôsob dedenia tejto entity je typu `table-per-subclass`.

3.5.1.2 Entita `PersonAddress`

Predstavuje adresu osoby. Atribúty tejto entity sú popísané v tabuľke 3.2.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`street`	`String`	Ulica.	
`city`	`String`	Mesto.	
`zipCode`	`boolean`	PSČ.	

Tabuľka 3.2: Atribúty entity `PersonAddress`

3.5.1.3 Entita `Country`

Predstavuje krajinu. Atribúty tejto entity sú popísané v tabuľke 3.3.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`name`	`String`	Názov krajiny.	unikátny

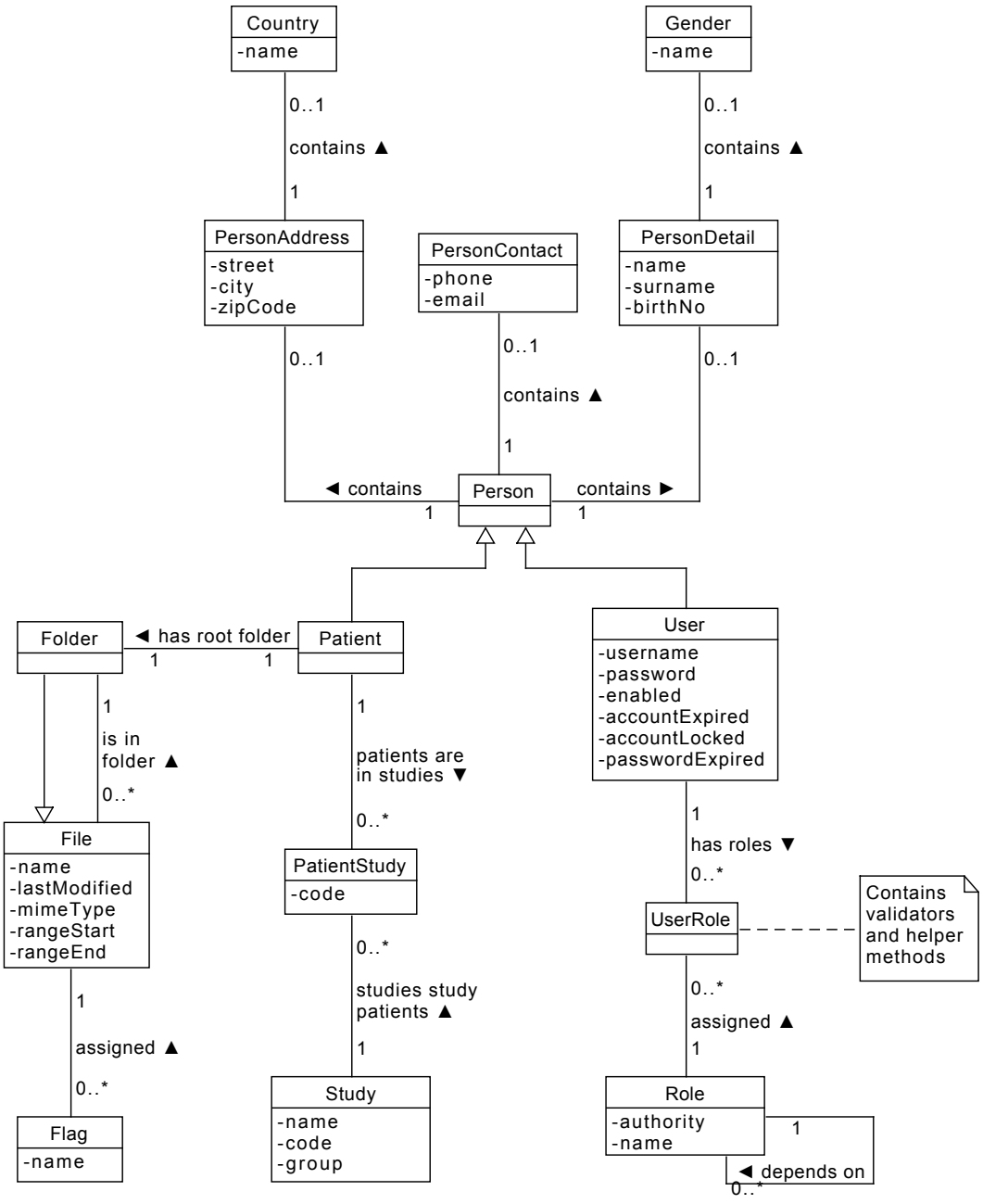
Tabuľka 3.3: Atribúty entity `Country`

3.5.1.4 Entita `PersonContact`

Predstavuje kontakt na osobu. Atribúty tejto entity sú popísané v tabuľke 3.4.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`phone`	`String`	Telefónne číslo.	
`email`	`String`	E-mail.	

Tabuľka 3.4: Atribúty entity `PersonContact`



Obr. 3.11: UML Class: Doménový model

3.5.1.5 Entita `PersonDetail`

Predstavuje ďalšie detaily o osobe. Atribúty tejto entity sú popísané v tabuľke 3.5.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`name`	`String`	Krstné meno.	
`surname`	`String`	Priezvisko.	
`birthNo`	`String`	Rodné číslo.	

Tabuľka 3.5: Atribúty entity `PersonDetail`

3.5.1.6 Entita `Gender`

Predstavuje pohlavie. Atribúty tejto entity sú popísané v tabuľke 3.6.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`name`	`String`	Názov pohlavia.	unikátny

Tabuľka 3.6: Atribúty entity `Gender`

3.5.1.7 Entita `User`

Predstavuje užívateľa, ktorý dedí atribúty od spoločnej entity `Person`. Užívateľ je typ osoby, ktorému je umožnené sa prihlasovať do systému. Atribúty tejto entity sú popísané v tabuľke 3.7.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`username`	`String`	Prihlasovacie meno užívateľa.	neprázdny, unikátny
`password`	`String`	Hash prihlasovacieho hesla.	neprázdny
`accountExpired`	`boolean`	Nepoužíva sa.	
`accountLocked`	`boolean`	Nepoužíva sa.	
`passwordExpired`	`boolean`	Nepoužíva sa.	

Tabuľka 3.7: Atribúty entity `User`

3.5.1.8 Entita `Role`

Predstavuje užívateľské oprávnenie, ktoré môže byť priradené viacerým užívateľom. Zároveň môže mať jeden užívateľ priradených viacero oprávnení. Vzťah typu `M:N` medzi entitami `Role` a `User` je dekomponovaný pomocou entity `UserRole`, ktorá obsahuje pomocné metódy pre prácu s oprávneniami a rôzne validačné pravidlá. Užívateľské oprávnenia môžu závisieť na ďalších užívateľských oprávneniach a tvoriť tak stromovú hierarchiu oprávnení. Atribúty tejto entity sú popísané v tabuľke 3.8.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`authority`	`String`	Kód oprávnenia.	neprázdny, unikátny
`name`	`String`	Orientačný názov oprávnenia.	neprázdny

Tabuľka 3.8: Atribúty entity `Role`

3.5.1.9 Entita `Patient`

Predstavuje pacienta. Pacient môže byť zaradený do viacerých štúdií. Pacient má priradený jeden koreňový adresár, v ktorom má uložené záznamy o jeho súboroch.

3.5.1.10 Entita `Study`

Predstavuje štúdiu do ktorej môže byť priradený pacient. Štúdia môže obahovať viacero pacientov. Atribúty tejto entity sú popísané v tabuľke 3.9.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`name`	`String`	Názov štúdie.	neprázdny
`code`	`String`	Kód štúdie.	neprázdny, unikátny
`group`	`String`	Názov výskumnej skupiny.	

Tabuľka 3.9: Atribúty entity `Study`

3.5.1.11 Entita `PatientStudy`

Predstavuje pacienta v štúdii. Atribúty tejto entity sú popísané v tabuľke 3.10.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`code`	`String`	Kód pacienta v štúdii.	

Tabuľka 3.10: Atribúty entity `PatientStudy`

3.5.1.12 Entity `File` a `Folder`

Predstavujú súbor. Ak je entita typu `Folder`, ide o adresár. Adresár je tiež súbor. Adresár môže obsahovať súbory. Pacient má vždy jeden hlavný adresár, v ktorom môžu byť ďalšie súbory. Súboru je možné nastaviť príznaky. Spôsob dedenia medzi entitami `File` a `Folder` je typu `table-per-hierarchy`. Atribúty týchto entít sú popísané v tabuľke 3.11.

3.5.1.13 Entita `Flag`

Predstavuje príznak súboru. Súbor môže mať priradených viacero príznakov. Atribúty tejto entity sú popísané v tabuľke 3.12.

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`name`	`String`	Názov súboru.	unikátny v rámci jedného adresáru
`lastModified`	`Date`	Dátum a čas poslednej úpravy (resp. vytvorenia) súboru.	
`mimeType`	`String`	MIME typ súboru.	
`rangeStart`	`Date`	Začiatok intervalu, v ktorom sa súbor nachádza.	
`rangeEnd`	`Date`	Koniec intervalu, v ktorom sa súbor nachádza.	

Tabuľka 3.11: Atribúty entít `File` a `Folder`

Názov atribútu	Typ atribútu	Popis	Integritné obmedzenia
`name`	`String`	Názov príznaku.	unikátny

Tabuľka 3.12: Atribúty entity `Flag`

3.6 Užívateľské oprávnenia

Ako bolo zadefinované v sekcii 3.2.1, systém rozoznáva troch rôznych aktérov. Okrem toho ale jednotlivé užívateľské operácie podliehajú autorizácii na základe užívateľských oprávnení. Každé užívateľské oprávnenie môže dediť ďalšie oprávnenia. Dedenie oprávnení je tranzitívne, teda ak oprávnenie C dedí oprávnenie B a zároveň oprávnenie B dedí oprávnenie A, potom užívateľ s oprávnením C automaticky získavá aj oprávnenia B a A. Zoznam užívateľských oprávnení, spolu s ich popisom a pravidlami dedenia je uvedený v tabuľke 3.13.

Názov oprávnenia	Popis oprávnenia	Zdedené oprávnenia
P_FLAG_VIEW	Prezeranie príznakov.	
P_FLAG_EDIT	Úprava a pridávanie príznakov.	P_FLAG_VIEW
P_FLAG_DELETE	Mazanie príznakov.	P_FLAG_EDIT
P_STUDY_VIEW	Prezeranie štúdií.	
P_STUDY_EDIT	Úprava a pridávanie štúdií.	P_STUDY_VIEW, P_PATIENT_ALL
P_STUDY_DELETE	Mazanie štúdií.	P_STUDY_EDIT
P_PATIENT_VIEW	Prezeranie pacienov.	
P_PATIENT_ALL	Prezeranie pacienov vo všetkých štúdiách.	P_PATIENT_VIEW
P_PATIENT_PERSONAL	Prezeranie pacienov vo všetkých štúdiách.	P_PATIENT_VIEW
P_PATIENT_EDIT	Úprava a pridávanie pacientov.	P_PATIENT_PERSONAL
P_PATIENT_STUDIES	Organizácia pacientov do štúdií.	P_STUDY_VIEW, P_PATIENT_ALL
P_PATIENT_FILES	Správa súborov pacienta.	P_FLAG_VIEW, P_PATIENT_VIEW
P_PATIENT_DELETE	Mazanie pacientov.	P_PATIENT_EDIT, P_PATIENT_STUDIES, P_PATIENT_FILES
P_USER_VIEW	Prezeranie užívateľov.	
P_USER_EDIT	Úprava a pridávanie užívateľov.	P_USER_VIEW
P_USER_DELETE	Mazanie užívateľov.	P_USER_EDIT
P_USER_PERMISSIONS	Správa užívateľských oprávnení	P_USER_DELETE, P_PATIENT_PERSONAL, P_PATIENT_DELETE, P_STUDY_DELETE, P_FLAG_DELETE

Tabuľka 3.13: Užívateľské oprávnenia

Kapitola 4

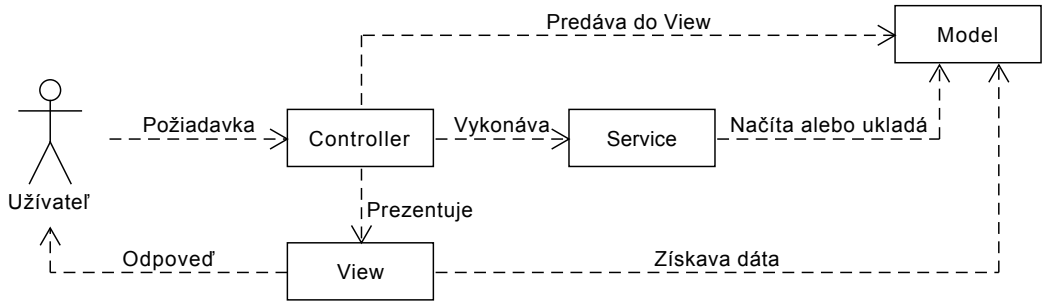
Návrh

4.1 Model architektúry

Za model architektúry som zvolil architektonický vzor *Model–view–controller (MVC)*. Pre tento vzor som sa rozhodol, najmä z dôvodu, že s ním mám dobrú skúsenosť z iných projektov a aplikácie postavené na tomto vzore majú vďaka oddeleným vrstvám prehľadný zdrojový kód s nízkou náchylnosťou na chyby.

Ako je uvedené v [7, s. 24]: "MVC sa skladá z troch druhov objektov. *Model* je aplikačný objekt, *View* je jeho vyjadrenie na obrazovke a *Controller* definuje spôsob, ako užívateľské rozhranie reaguje na užívateľský vstup. Pred MVC mali návrhy užívateľských rozhraní tendenciu zlučovať tieto objekty dokopy. MVC ich oddeľuje, a tým zvyšuje ich tvárnosť a ďalšie použitie."

Okrem štandardných troch vrstiev používam v aplikácii ešte štvrtú – servisnú vrstvu, kde je umiestnená väčšina business logiky, ktorá by inak musela byť umiestnená v kontroléroch. Tým je zvýšená miera znovupoužiteľnosti aplikačnej logiky a kontroléry sú prehľadnejšie. Schéma architektúry je zjednodušene znázornená v diagrame 4.1. Diagram je inšpirovaný podľa [8].



Obr. 4.1: Schéma modelu architektúry

4.2 Spolupráca objektov

Spolupráca jednotlivých objektov v systéme sa často znázorňuje pomocou sekvenčných diagramov. Tento diagram ukazuje nielen komunikáciu medzi objektami, ale aj poradie, v akom táto komunikácia prebieha. Jednotlivé objekty sú v diagrame zakreslené v obdĺžnikoch v hornej časti diagramu. Od objektov sú vedené vertikálne čiary (tzv. *lifelines*), ktoré znázorňujú inštanciu daného objektu. Z týchto inštancií sú potom vedené horizontálne šípky označujúce smer komunikácie a druh posielanej správy (napríklad názov volanej metódy). Vodorovnou prerušovanou šípkou je znázornená odpoveď. Obdĺžnikom vedeným cez *lifelines* sú znázornené riadiace štruktúry.

4.2.1 Základné CRUD operácie

Podobne ako v sekcii 3.4.1, sú tieto operácie vzhľadom na ich podobnosť v rôznych prípadoch užitia popísané iba všeobecne.

4.2.1.1 Vytváranie a editácia

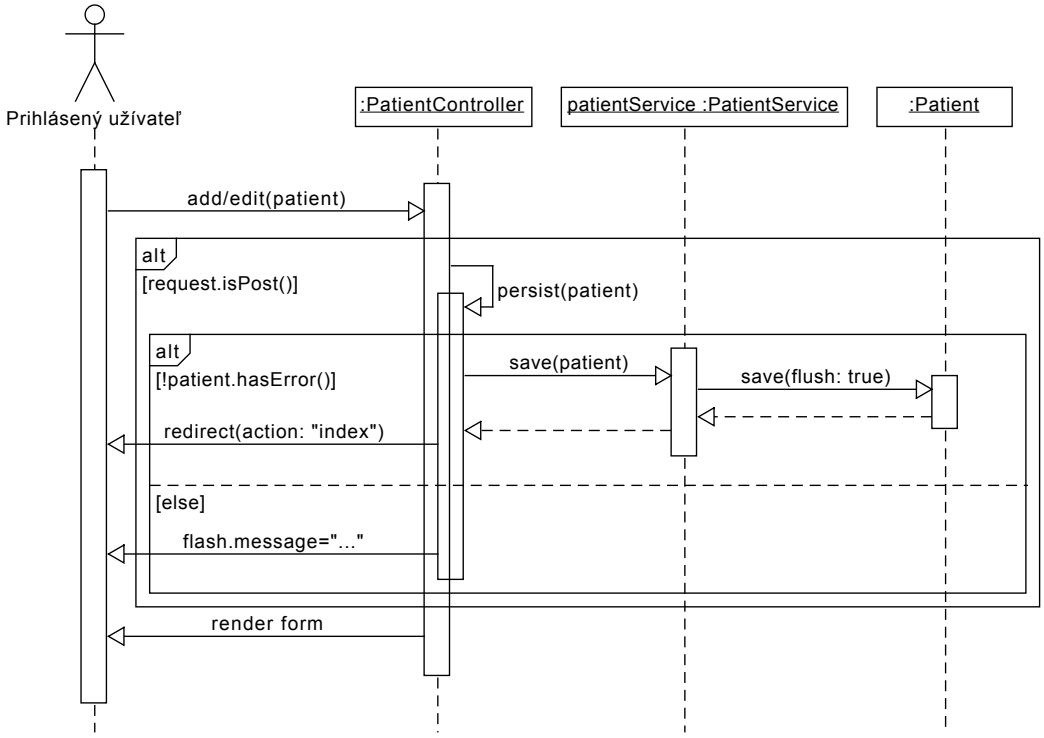
Po zaslaní požiadavku na vytvorenie resp. editáciu záznamu sa zavolá metóda `add()` resp. `edit(Entity entity)` príslušného kontroléra, ktorý užívateľovi zobrazí formulár na zadanie vstupných údajov. Po odoslaní formuláru sa v kontroléri zavolá rovnaká metóda, tentokrát však požiadavka príde pomocou HTTP metódy POST, na čo systém reaguje volaním metódy `persist(Entity entity)` v danom kontroléri. Táto metóda sa pokúsi uložiť zadané vstupné údaje zaslaním požiadavky servisnej triede, ktorá vykoná pokus o uloženie dát. Ak takýto pokus o uloženie zlyhá (napríklad z dôvodu integritných obmedzení), zobrazí sa užívateľovi chyba (tzv. *flash message*) a formulár, v ktorom môže upraviť vstup a pokúsiť sa ho znovu odoslať. Ak bolo uloženie záznamu úspešné, užívateľ je metódou `redirect(action: 'index')` presmerovaný na stránku s vylistovaním všetkých záznamov.

Táto spolupráca objektov je znázornená na konkrétnom príklade vytvárania/úpravy pacienta v diagrame 4.2.

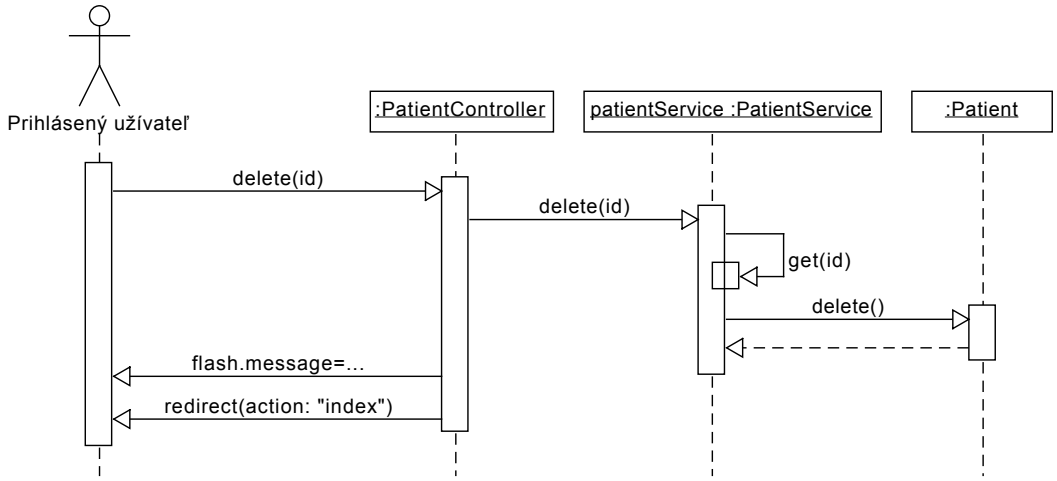
4.2.1.2 Mazanie

Po zaslaní požiadavku na odstránenie záznamu sa zavolá metóda `delete(id)` príslušného kontroléra, ktorá požiadavku prepošle servisnej triede. Servisná trieda sa pokúsi podľa id vyhľadať záznam a odstrániť ho. Po dokončení operácie je užívateľ metódou `redirect(action: 'index')` presmerovaný na stránku s vylistovaním všetkých záznamov. Informácia o tom, či bola operácia úspešná alebo sa vyskytla chyba je užívateľovi oznámené pomocou *flash message*.

Táto spolupráca objektov je znázornená na konkrétnom príklade odstránenia pacienta v diagrame 4.3.



Obr. 4.2: UML Sequence: CRUD operácia create a update



Obr. 4.3: UML Sequence: CRUD operácia delete

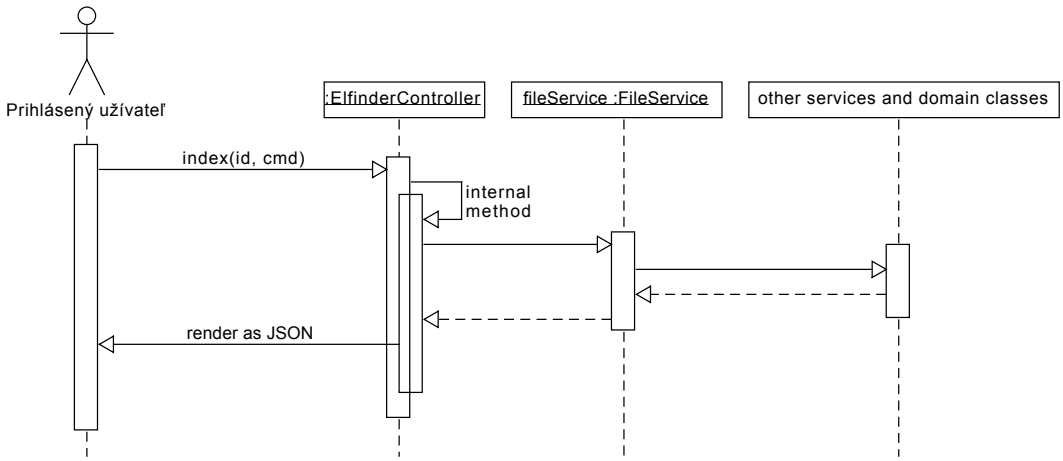
4.2.1.3 Vylistovanie záznamov

Vylistovanie do tabuľky, spolu so všetkými súvisejúcimi operáciami (filtrovanie, stránkovanie, usporiadavanie) riadi doplnok EasyGrid. V kontroléri sa pomocou DSL nakonfiguruje táto tabuľka zadaním doménovej triedy, s ktorou bude doplnok pracovať a stĺpcov, ktoré budú v tabuľke zobrazené.

4.2.2 Správa súborov cez webové rozhranie

Užívateľ zasiela kontroléru požiadavky, kde medzi parametrami je názov súborovej operácie, ktorú chce vykonať (vylistovanie súborov, vytvorenie adresára, ...) a ID pacienta, ktorého sa operácia týka. Kontrolér zavolá internnú metódu, podľa typu súborovej operácie na vykonanie. Tato metóda komunikuje so servisnou triedou, ktorá tieto operácie vykonáva. Servisná trieda obvykle komunikuje s ďalšími servisnými alebo doménovými triedami v systéme. Odpoveď je užívateľovi zaslaná vo forme JSON objektu.

Táto spolupráca objektov je znázornená na diagrame 4.4



Obr. 4.4: UML Sequence: Správa súborov cez webové rozhranie

4.2.3 Správa súborov cez FTP rozhranie

Pri FTP rozhraní musí najskôr prebehnúť inicializácia FTP serveru, čo vykonáva systém pri spustení aplikácie. Následne sa môžu na FTP server pripájať užívatelia a robiť súborové operácie.

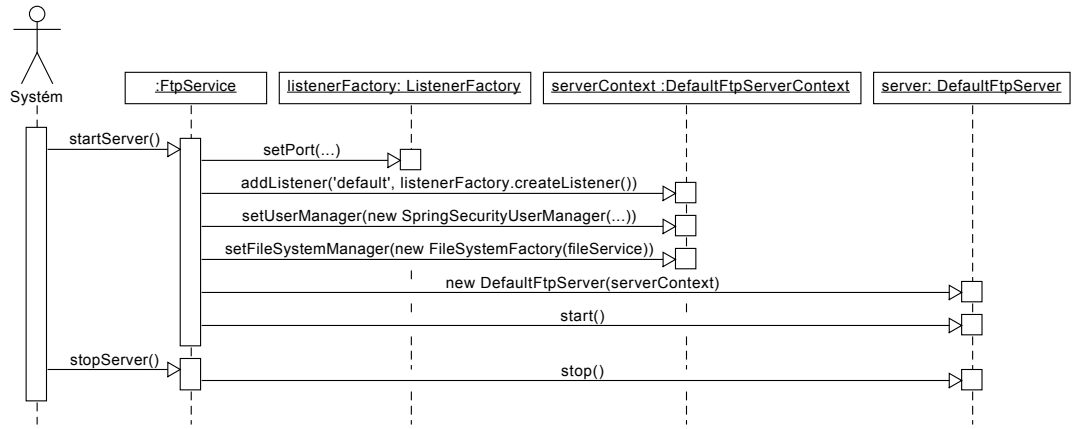
4.2.3.1 Inicializácia FTP serveru

Inicializácia FTP serveru je proces, ktorý prebieha po zavolaní metódy `startServer()` v servisnej triede `FtpService`. Táto trieda vytvorí inštanciu serverového kontextu, ktorý

zapuzdruje nastavenia serveru a inštancie ďalších potrebných tried. FTP server je nutné nastaviť tak, aby vedel správne komunikovať s ostatnými časťami aplikácie. V prvom rade je potrebné nastaviť *listener*, ktorý bude počúvať na zvolenom porte. Užívateľský manažér `SpringSecurityUserManager` implementuje rozhranie `UserManager` a slúži na autentifikáciu užívateľov, ktorí sa budú prihlasovať na FTP server. `FileSystemFactory` je továrňa, ktorá vytvára inštancie triedy `FileSystemView` a slúži na abstrahovanie pohľadu na súborový systém. V tomto prípade teda trieda `FileSystemView` zabezpečuje komunikáciu so servisnou triedou `FileService`. V poslednom kroku sa nastavený serverový kontext predá konštruktoru triedy `DefaultFtpServer` a zavolaním metódy `start()` začne FTP server prijímať prichádzajúce FTP spojenia.

Metódou `stopServer()` v servisnej triede `FtpServer` je možné zastaviť chod FTP serveru.

Táto spolupráca objektov je znázornená v diagrame 4.5.

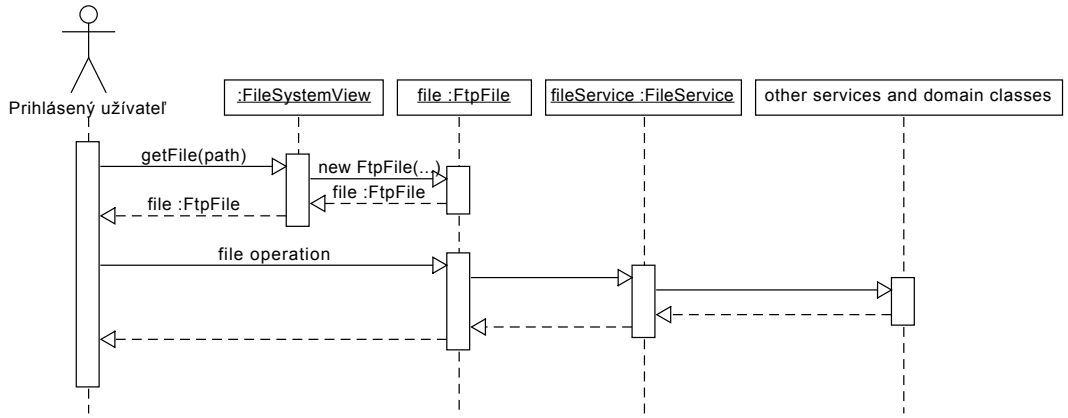


Obr. 4.5: UML Sequence: Inicializácia FTP serveru

4.2.3.2 Súborové operácie

Po inicializácii zabezpečuje väčšinu logiky, ktorá sa týka FTP prokolu samotný FTP server. Po otvorení spojenia sa návštevník autentizuje na základe pravidiel definovaných v `SpringSecurityUserManager` a stane sa tak prihláseným užívateľom. Nad množinou súborov potom volá súborové operácie. Každý takýto súbor reprezentuje trieda `FtpFile`, ktorej inštanciu získa cez továrňu `getFile(path)` v triede `FileSystemView`. Nad týmto súborom potom prevádza súborové operácie. Trieda `FtpFile` komunikuje so servisnou triedou `FileService`, ktorá zabezpečuje konkrétne prevedenie týchto súborových operácií.

Táto spolupráca objektov je znázornená v diagrame 4.6.



Obr. 4.6: UML Sequence: Súborové operácie na FTP serveri

4.3 Návrh užívateľského rozhrania

Podľa [9, s. 1] je užívateľské rozhranie najdôležitejšou časťou akéhokoľvek počítačového systému. Zo všeobecných princípov tvorby užívateľského rozhrania podľa [9, s. 41-51] som vybral niekoľko princípov, ktoré boli zohľadnené pri návrhu uživateľského rozhrania tejto aplikácie:

Estetickosť Estetický design je vizuálne atraktívny pre užívateľa, udržuje jeho pozornosť, zlepšuje prístup k počítačovému systému a práca s takýmto systémom je príjemnejšia. Naopak, absencia estetickosti disorientuje užívateľa, spomaľuje a odrádza ho od používania takéhoto systému. Konstrast medzi prvkami, zoskupovanie prvkov, zarovnanie prvkov, trojrozmerná reprezentácia prvkov, používanie jednoduchých a efektívnych grafických prvkov.

Jasnosť Vizuálne prvky a text by mali byť pochopiteľné, jednoznačné, podobné konceptom z reálneho sveta, bez používania nezrozumiteľného (IT) žargónu. Používanie vizuálnych prvkov, metafor, vhodná voľba slov.

Kompatibilita Prispôsobenie sa užívateľovi, ktorý bude systém používať. Uvedomenie si, že pohľad a prístup programátora a užívateľa nemusí byť rovnaký. Zohľadnenie prípadných predchádzajúcich verzií systému, na ktorý je užívateľ zvyknutý.

Zrozumiteľnosť Systém by mal byť pochopiteľný, v ideálnom prípade bez nutnosti čítať dlhé používateľské manuály. Postupnosť krokov v zrozumiteľnom a zmysluplnom poradí. Užívateľ rýchlo vie kam sa pozerať, čo/kedy/prečo/ako robiť.

Konzistencia Rovnaká akcia by mala priniesť vždy rovnaký výsledok. Funkcie a umiestnenie prvkov sa nemení v priebehu času. Podobné prvky majú podobný vzhľad, podobné použitie, správajú sa podobne.

Kontrola Užívateľ má pocit kontroly nad systémom. Akcie sú vykonávané na explicitnú žiadadosť užívateľa, sú vykonané čo najrýchlejšie a v prípade dlho trvajúcej akcie je možné takúto akciu prerušiť/pozastaviť. Užívateľ nie je zbytočne obťažovaný chybovými hláškami.

Priamosť Všetky akcie dostupné pre užívateľa sú viditeľné a s predvídateľným výsledkom. Zníženie mentálnej záťaže užívateľa na minimum.

Efektivita Minimalizácia potreby užívateľa pohybovať rukami a očami. Brať do úvahy prechody užívateľa medzi jednotlivými súčasťami systému, aby na seba nadväzovali čo najzrejmejším spôsobom. Vyhnutie sa častej potrebe striedania vstupných zariadení ako myši a klávesnice.

Tolerovať chyby Faktom je, že užívateľ robí chyby. Zobraziť by sa mu mali ale iba chyby, ktoré sú pre neho relevantné. Kde je to možné, pokúsiť sa automaticky opraviť chyby za užívateľa. Nedovoliť užívateľovi spôsobiť katastrofickú chybu.

Užívateľ ďalej podľa [9, s. 64] reaguje na zle navrhnuté užívateľské rozhranie dvoma spôsobmi:

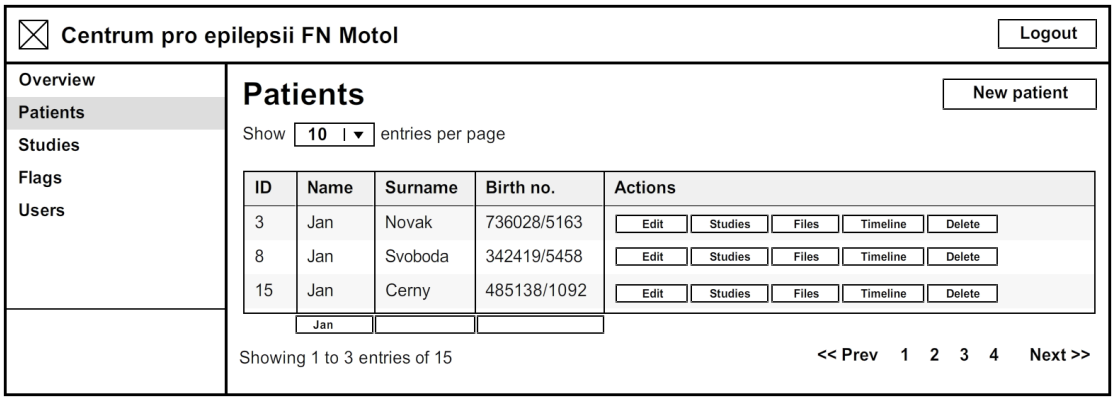
Psychologicky Zmätok, nepríjemný pocit, frustrácia, panika, stres, znudenosť.

Fyzicky Prerušenie používania systému, používanie iba časti systému, potreba prostredníka medzi systémom a potenciálnym užívateľom, nutnosť modifikácia úlohy tak, aby spĺňali obmedzené možnosti systému.

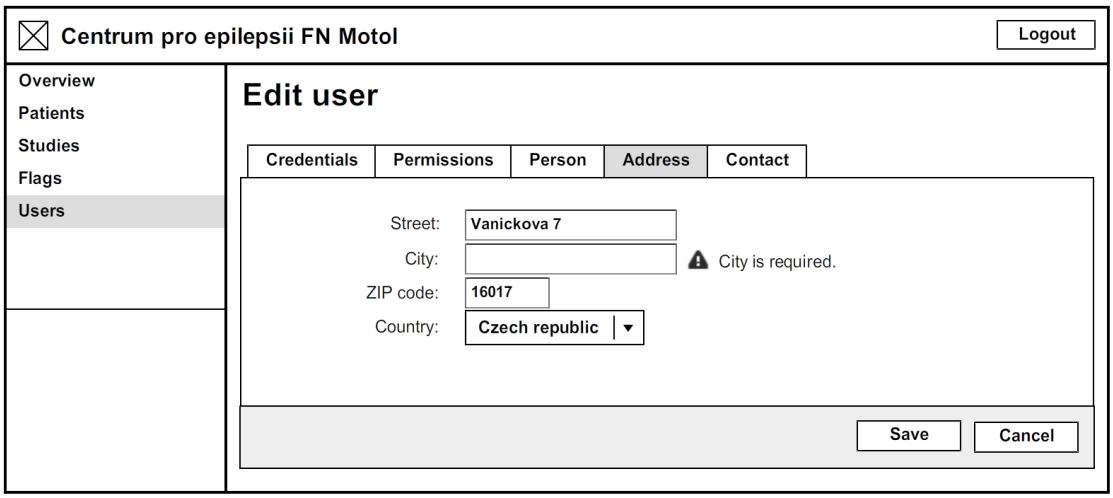
4.3.1 Drôtený model

Drôtený model (*wireframe*) je návrh rozloženia jednotlivých prvkov na pripravovanej web stránke alebo aplikácii. Pomocou jednoduchých čiar znázorňuje komponenty stránky a ako do seba zapadajú. Je to nevyhnutný prvý krok k formálnemu založeniu vizuálu budúcej web stránky a slúži všetkým ľuďom zapojených do procesu tvorby stránky. Pre jednoduché aplikácie zvyčajne postačuje jeden drôtený model na vytvorenie šablóny, pre komplexnejšie projekty je potrebných niekoľko takýchto modelov. Nie je ale zvykom robiť drôtený model pre každú možnú obrazovku aplikácie. [10, s. 128-131]

Pre vytvorenie drôteného modelu tejto aplikácie som použil online aplikáciu WireframePro (dostupné na adrese `http://www.mockflow.com/`). Na obrázku 4.7 je vidieť vizuálny návrh užívateľského rozhrana správy pacientov – tabuľka v ktorej sa dá vyhľadávať, stránkovať a usporiadavať záznamy. Pri každom zázname je zoznam možných akcií. Na obrázku 4.8 je vizuálny návrh užívateľského rozhrana úpravy užívateľa – formulár a ukážka validácie údajov.



Obr. 4.7: Drôtený model správy pacientov



Obr. 4.8: Drôtený model úpravy užívateľa

Kapitola 5

Použité technológie

5.1 Programovací jazyk Groovy

Groovy je objektovo orientovaný programovací jazyk pre platformu Java, ktorý je svojimi vlastnosťami inšpirovaný Pythonom, Ruby alebo Smalltalkom. Kompilátorom je prekladaný priamo do binárneho kódu spustiteľného v Java Virtual Machine (JVM). Syntax tohto jazyka vychádza z Javy a väčšina zdrojových kódov v jazyku Java je preložiteľná do binárnej podoby aj pomocou Groovy prekladača, môže sa však líšiť sémantika takéhoto kódu. Groovy je oproti Jave rozšírený o niektoré nové vlastnosti (tzv. *syntaktický cukor*) – nepovinné bodkočiarky na oddeľovanie príkazov, nepovinné kľúčové slovo `return`, nepovinná špecifikácia dátového typu pomocou pomocou slova `def`, triedy sú implicitne `public`, nepovinné používanie niektorých zátvoriek, automatické gettery a settery, automatický konštruktor, natívna syntax pre niektoré dátove štruktúry, nové operátory, atď. [11]

Listing 5.1: Ukážka jazyka Groovy: Výpis zoznamu zamestnancov

```
class Person { String name, surname }
def employees = [
    new Person(name: "Peter"),
    new Person(name: "Jan", surname: "Novy")
]
employees.each { Person p ->
    println (([p.name, p.surname] - null).join("␣"))
}
```

5.2 Aplikačný framework Grails

Grails (`https://grails.org/`) je open-source webový aplikačný framework pre programovací jazyk Groovy. Pre dosiahnutie čo najvyššej produktivity dodržuje tento framework paradigmu *convention over configuration*. Framework sa pôvodne volal Groovy on Rails (podľa populárneho Ruby on Rails), neskôr sa však začal používať iba skrátený názov Grails.

Grails je framework, ktorý priamo integruje mnoho už existujúcich technológií využívaných pri vývoji webových aplikácií. Ide napríklad o ORM postavené na Hibernate, šablónovací systém Groovy Server Pages (GSP), webový framework Spring MVC, interaktívna konzola a build systém postavený na Gradle, embedded serverový kontajner Tomcat, dependency injection postavené na Spring IoC, internacionalizácia (i18n) postavená na Spring MessageSource, testovacie frameworky Spock a JUnit alebo konfigurácia transakcií pomocou Spring Transaction Abstraction. Všetky tieto technológie sú v Grails frameworku ľahko použiteľné vďaka rozsiahlemu používaniu *doménovo špecifického jazyka (DSL)* a absencii konfigurácie pomocou XML súborov.

Detailne popísať všetky funkcie a vlastnosti tohto frameworku by bolo rozsahom na samostatnú prácu, preto v tejto sekcií popíšem iba niektoré základné aspekty, ktoré boli použité, pri implementácii tejto aplikácie.

5.2.1 Grails Command Line

Grails Command Line je nástroj pre príkazový riadok, ktorý slúži na základnú prácu s Grails projektami. Pomocou tohto nástroja je možné vytvárať nové projekty, spúšťať aplikáciu alebo testy, vytvárať servlety, inštalovať doplnky a mnoho ďalších funkcií. Príkazy, ktoré tento nástroj podporuje sa spúšťajú zadaním `grails [názov príkazu]` do príkazového riadku operačného systému. Napríklad príkaz `grails create-app motol` vytvorí prázdny skeleton projekt s názvom *motol*, príkazom `grails run-app` sa potom tento projekt spustí v integrovanom Tomcat serveri. Zoznam všetkých dostupných príkazov je možné vylistovať zadaním zadaním `grails help`.

Pri použití vhodného IDE s podporou pre Grails framework sa dá priamemu používaniu tohto nástroja z veľkej časti vyhnúť, keďže tieto príkazy volá po zvolení príslušnej akcie v pozadí samotné IDE.

5.2.1.1 Grails Object Relational Mapping (GORM)

GORM je implementácia objektovo-relačného mapovania pre Grails. Je postavená na existujúcom open-source ORM riešení Hibernate 4 a podporuje väčšinu možnosti, ktoré toto ORM ponúka. Vďaka možnostiam, ktoré poskytuje jazyk Groovy je však práca s týmto ORM značne jednoduchšia.

Listing 5.2: Ukážka použitia GORM

```
class Author {
    String name
    static constraints = { name unique: true }
}
class Book {
    String name
    Integer rating
    static hasMany = [authors: Author]
}
def author = new Author(name:"Stephen King")
```

```
// Persist new book to database
def book = new Book(name: "The Green Mile", rating: 7)
book.authors = [author]
book.save(flush: true)

// Print all books written by Stephen King and with rating greater than 5
Book.findAllByAuthorsInListAndRatingGreatherThan([author], 5)
    .each { Book b -> println b.author.name + ': ' + b.name }
```

5.2.1.2 Servisná vrstva

Tým, ako rastie veľkosť aplikácie a zvyšuje sa počet kontrolérov, sa často sa stáva, že viacero kontrolérov používa tú istú logiku. Servisné triedy predstavujú jednoduchý spôsob, ako zapuzdriť znovupoužiteľnú biznis logiku. Inštancie týhto tried sú automaticky zaregistrované do aplikačného kontextu a vďaka depenedency injection sú tak použiteľné kdekoľvek v aplikácii. [12, s. 133]

5.2.1.3 Groovy Server Pages (GSP)

GSP je technológia, pre vytváranie dynamicky generovaných web stránok v HTML. Použitím sa podobá na ASP alebo JSP. Z pohľadu MVC architektúry, predstavuje GSP vrstvu *view*. Vo väčšine prípadov platí, že každá metóda kontroléru má vlástný GSP súbor uložený v adresári `grails-app/views` s príponou .gsp. Tieto súbory môžu okrem HTML tagov obsahovať aj GSP tagy a výrazy začínajúce symbolom `$`.

Listing 5.3: Ukážka GSP

```
<g:applyLayout name="default">
    <body>
        <h1>${user ? "Edit user " + user.name : "Add user"}</h1>
        <g:form action="submit">
            <g:render template="userForm" />
        </g:form>
    </body>
</g:applyLayout>
```

5.2.1.4 Grails zásuvné moduly (pluginy)

Okrem základných funkcií, ktoré poskytuje priamo Grails framework, existujé veľké množstvo pluginov, ktorých vývoj je udržiavaný Grails komunitou. Zoznam týchto pluginov je dostupný na adrese `https://grails.org/plugins/`. Pre inštaláciu nového pluginu je potrebné vložiť riadok do konfiguračného súboru `BuildConfig.groovy` a plugin bude automaticky nainštalovaný pri ďalšom spustení aplikácie, prípadne zavolaním príkazu `grails install-plugin` *`[názov pluginu]`*.

V tejto aplikácií sú zo stoviek dostupných pluginov použité najmä nasledujúce pluginy (nejedná sa o úplný zoznam):

tomcat:7.0.55 integrovaný serverový kontajne Tomcat 7, používaný počas vývoja aplikácie.

hibernate4:4.3.5.5 pridáva podporu pre ORM Hibernate 4.

spring-security-core:2.0-RC4 je knižnica Spring Security Core, ktorá slúži na autentizáciu užívateľov a ich autorizáciu pomocou užívateľských oprávnení.

asset-pipeline:1.9.6 je plugin Asset Pipeline pre správu statických zdrojov – JavaScriptov a CSS súborov. Tieto zdroje je následne možné automaticky spájať do jedného súboru, komprimovať alebo inak spracovávať.

less-asset-pipeline:1.10.0 rozširuje základný Asset Pipeline plugin o LESS preprocesor pri spracovávaní CSS súborov.

coffee-asset-pipeline:2.0.7 rozširuje základný Asset Pipeline plugin o podporu CoffeeScriptu.

jquery:1.11.1 pridáva aplikácii podporu pre JavaScript multiplatformovú knižnicu jQuery.

jquery-ui:1.10.4 pridáva aplikácii podporu pre sadu doplnkov jQuery UI.

twitter-bootstrap:3.3.1 pridáva aplikácii podporu pre CSS framework Bootstrap.

easygrid:1.6.9 je plugin Easygrid, ktorý umožňuje deklaratívny spôsob definovania Data Gridu, teda tabuľky, ktorá slúži na prezentovanie záznamov užívateľovi. Okrem vylistovania záznamov užívateľovi tento plugin podporuje aj nadefinovanie akcií pre jednotlivé záznamy, filtrovanie, stránkovanie a usporiadavanie záznamov.

elfinder-resources:2.0.0 pridáva podporu pre webový súborový manažér elFinder.

geb:0.10.0 pridáva podporu pre Geb – automatizácia prehliadača pri funkcionálnom testovaní aplikácie.

5.3 Relačná databáza PostgreSQL

V tejto aplikácií pristupujem k databáze iba cez ORM, ktoré podporuje viaceré databázové platformy. Podľa [13] patria v súčasnosti medzi najpopulárnejšie RDBMS platformy Oracle, MySQL, Microsoft SQL Server, PostgreSQL a IBM DB2. Z dôvodu ušetrenia nákladov je však preferovaná databáza, ktorú je možné používať zadarmo - teda PostgreSQL alebo MySQL. Pre potreby tejto webovej aplikácie neprináša ani jedna databáza zásadné výhody. Nakoľko mám ale s PostgreSQL viac skúseností z iných projektov, rozhodol som sa uprednostniť túto databázu.

5.4 Apache FtpServer

Pre splnenie požiadavky **F.5.5.2** je v aplikácii integrovaný FTP server Apache FtpServer (`http://mina.apache.org/ftpserver-project/`). Tento FTP server je možné v

základnom nastavení namapovať iba priamo na adresár v súborovom systéme, čo bolo pre potreby tejto aplikácie nedostatočné. Keďže je ale napísaný v jazyku Java, implementáciou príslušných rozhraní som rozšíril funkcie tohto servera tak, aby prístup k súborom prebiehal cez spoločné súborové rozhranie.

Sieťová podpora tohto FTP servera je postavená na asynchrónnej IO knižnici Apache MINA. Vďaka tomu je umožnené, aby k aplikácii pristupovalo cez FTP viacero klientov súčasne.

5.5 Technológie front-endu

Front-end je vrstva webovej aplikácie, ktorá je viditeľná bežným užívateľom aplikácie. Ide o programovacie jazyky a ďalšie technológie, ktoré sú vykonávané priamo v prehliadači užívateľa. Základnou technológiu pre programovanie webového užívateľského rozhrania tvoria jazyky HTML a CSS, ktorých špecifikáciu zastrešuje organizácia World Wide Web Consortium (W3C). Okrem tohto sa často využíva aj skriptovací jazyk JavaScript alebo rôzne rozširujúce technológie ako Flash, Silverlight alebo Java applety.

Značkovací jazyk HTML vznikol v 90. rokoch vo forme krátkeho dokumentu, popisujúceho akým spôsobom tvoriť webové stránky pomocou elementov. Mnohé z týchto elementov slúžili k popisu obsahu stránky, ako napríklad nadpisy, odseky alebo zoznamy. Tento jazyk postupne prešiel vývojom až do dnešnej podoby. [14, s. XVI]

Jazyk CSS vznikol v roku 1996, teda až niekoľko rokov po HTML. Kým jazyk HTML slúži na popísanie samotného obsahu dokumentu, CSS slúži k definovaniu, ako bude tento obsah vyzerať v užívateľovom prehliadači. [14, s. XVI]

JavaScript je dynamický programovací jazyk, umožňujúci interakciu s užívateľom, asynchrónnu komunikáciu alebo dynamickú úpravu stránky. [15]

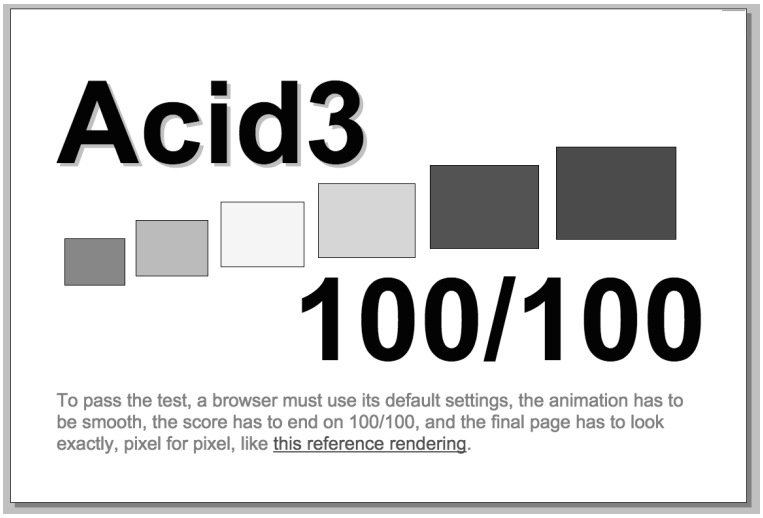
Keďže je front-end časť aplikácie spúšťaná priamo v užívateľovom prehliadači, je nutné, aby vývojári týchto prehliadačov dodržiavali webové štandardy, ktoré zaručia, že sa webová stránka bude správať rovnako naprieč rôznymi prehliadačmi.

Pre kontrolu webových prehliadačov slúži Acid3 test, ktorý na stupnici 0 až 100 udáva, ako dôsledne užívateľov prehliadač dodržuje požiadavky na webové štandardy a JavaScript. Acid3 test je dostupný na adrese `http://acid3.acidtests.org/` a väčšina moderných prehliadačov dnes týmto testom prejde s plným skóre.

Dodržovať štandardy by v ideálnom prípade mala aj samotná webová aplikácia, aj keď striktné dodržiavanie týchto štandardov je v praxi často problematické. Pre tento účel existujú validátory kódu HTML, CSS alebo ďalších jazykoch, ktoré špecifikuje organizácia W3C. Zoznam týchto validátorov je k dispozícii na adrese `http://www.w3.org/QA/Tools/`.

5.5.1 HTML5

HTML5 je v súčasnosti najnovšia verzia značkovacieho jazyka HTML, používaná pre prezentáciu obsahu webových stránok. HTML5 dokumenty sa skladajú okrem samotného textového obsahu aj z HTML tagov a ich atribútov, teda kľúčového slova ohraničeného



Obr. 5.1: Acid3 test v prehliadači Google Chrome 41

lomenými zátvorkami. Takýto tag môže byť párový (tvoriaci dvojicu otváracieho a ukončovacieho tagu) alebo nepárový. Presnú špecifikáciu jazyka HTML5 určuje organizácia W3C a podľa tejto špecifikácie sú následne vyvýjané jednotlivé webové prehliadače. HTML5 oproti predchádzajúcim verziám pridáva nové tagy pre lepšiu podporu multimédií, nové druhy vstupov, vlastné dátové atribúty, validátory, a ďalšie novinky.

Listing 5.4: Ukážka jednoduchého HTML5 dokumentu

```
<!DOCTYPE html>
<html lang="en-us">
    <body>
        <h1>Article title</h1>
        <p class="c1">Text of the article</p>
        <img src="img.jpg" class="right" alt="Image description">
    </body>
</html>
```

5.5.2 CSS3 a LESS

CSS alebo kaskádové štýly definujú, ako bude vyzerať obsah v užívateľovom prehliadači. Najčastejšie býva na serveri uložený v samostatnom textovom súbore, môže byť však aj priamo súčasťou HTML dokumentu. CSS3 je najnovšou verziou jazyka CSS, ktorá tento jazyk rozšírila napríklad o možnosť používať tiene, zahnuté rohy alebo farebné prechody.

LESS je preprocesor jayzka CSS, ktorý zefektívňuje základnú syntax tohto jazyka, čím sa kaskádové štýly zapísané v LESS stávajú v porovnaní s CSS čitateľnejšie a vývoj udržateľnejší. Pred použitím zápisu kaskádových štýlov v LESS v prehliadači je nutné tento kód najskôr skompilovať do CSS.

Listing 5.5: Ukážka použitia LESS

```
@textcolor: red;
@bordercolor: blue;
.custom-border (@radius: 5px) {
    border-radius: @radius;
    border: 1px @bordercolor solid;
}
#content {
  div.box {
    .custom-border(7px);
    color: @textcolor;
  }
}
```

5.5.3 JavaScript a CoffeeScript

JavaScript je, ako už názov napovedá, skriptovací jazyk na strane klienta, ktorým sa dá za behu modifikovať obsah stránky a asynchrónne komunikovať so serverom pomocou technológie Ajax. Pre zlepšenie čitateľnosti kódu a lepšej udržateľnosti vývoja používam jazyk CoffeeScript, ktorý základnú syntax jazyka JavaScript rozširuje o *syntaktický cukor*. Podobne ako je LESS prekladané do CSS, tak aj CoffeeScript je nutné pred použitím v prehliadači preložiť do JavaScriptu.

5.5.4 jQuery a jQuery UI

Ide o dve knižnice, ktoré dnes patria medzi štandard pri vývoji moderných web aplikácii. Podľa [16] je knižnica jQuery najpopulárnejšou JavaScript knižnicou, ktorá je použitá na viac ako 70% z 10.000 najnavštevovanejších web stránkach. Táto knižnica zjednodušuje napríklad prácu s DOM, vytváranie animácií, prácu s udalosťami alebo Ajaxom. Keďže zápis kódu v jazyku JavaScript sa môže v rôznych prehliadačoch líšiť, knižnica jQuery poskytuje multiplatformové rozhranie pre písanie užívateľských skriptov.

jQuery UI je sada doplnkov, efektov a ďalších rozšírení postavených na knižnici jQuery. Jednoduchým spôsobom je tak možné na stranku pridať podporu napríklad pre drag&drop, tooltipy, výber dátumu z kalendáru, modálne okná a ďalšie doplnky.

5.5.5 Bootstrap

Bootstrap je webový front-end framework, ktorý poskytuje kolekciu CSS a JavaScript súborov pre zefektívnenie tvorby webových aplikácií. Kaskádové štýly sú definované pomocou LESS, preto je možné ich jednoducho prispôsobiť potrebám konkrétnej aplikácie. Po použití tohto frameworku, je šablóna stránky rozdelená na 12 stĺpcov, do ktorých sa vkladajú jednotlivé komponenty – napríklad formuláre, ovládacie prvky, menu, panely, dialógy a ďalšie.

5.5.6 elFinder

elFinder je pokročilý webový správca súborov. Pomocou Ajaxu si vymieňa správy so serverom a umožňuje užívateľovi operácie so súbormi – výpis súborov, nahrávanie, mazanie, vytváranie súborov, premiestňovanie alebo vyhľadávanie.

5.6 Implementačné prostredie

Samotná implementácia tejto aplikácie a väčšina testovania prebiehala na osobnom počítači s operačným systémom Mac OS X Yosemite. Medzi ďalšie softwarové nástroje používané pri vývoji patrí najmä IDE a nástroj pre správu databázy.

5.6.1 IntelliJ IDEA 14

IndelliJ IDEA 14 je integrované vývojové prostredie (IDE) pre vývoj počítačového softvéru. Je vyvýjané firmou JetBrains so sídlom v Prahe. Toto vývojové prostredie priamo podporuje jazyk Groovy a webový framework Grails. Firma JetBrains na svojich web stránkach uvádza, že ide o najlepšie IDE pre vývoj Groovy a Grails aplikácií, ktoré bolo pre mňa navyše ľahko dostupné vďaka študentskej licencii poskytnutej Informačným a výpočetným centrom ČVUT.

5.6.2 Navicat Premium 11

Navicat je nástroj pre správu databázy. Na rozdiel od PostgreSQL konzoly PSQL, ktorá je nainštalovaná spolu s databázou PostgreSQL alebo GUI aplikácie pgAdmin prináša Navicat kvalitnejšie užívateľské rozhranie, čím zefektívňuje prácu s touto databázou.

Kapitola 6

Realizácia

Táto kapitola stručne popisuje konkrétne kroky, ktoré boli prevedené v procese implementácie systému podľa predchádzajúcej analýzy a návrhu.

6.1 Inicializácia projektu

Inicializácia nového Grails projektu prebehla príkazom `grails create-app motol`, resp. volaním tohto príkazu z IDE. Adresárová štruktúra prázdneho projektu vyzerá, v závislosti na verzii Grails, približne takto:

```
%PROJECT_HOME%
├── grails-app
│   ├── assets ..... Statické zdroje (CSS, LESS, JS, CoffeScript, obrázky, apod.)
│   ├── conf ..... Konfiguračné súbory
│   ├── controllers ..... Kontroléry
│   ├── domain ..... Doménové triedy
│   ├── i18n ..... Súbory pre internacionalizáciu aplikácie
│   ├── migrations ..... Databázové migrácie
│   ├── services ..... Servisné triedy
│   ├── taglib ..... Definície vlastných tagov v GSP
│   ├── utils ..... Ďalšie zdrojové kódy dostupné v aplikačnom kontexte
│   ├── views ..... GSP pohľady
│   │   └── layouts ..... GSP šablóny
├── lib ..... Knižnice, ktoré nie sú manažované cez správcu doplnkov
├── scripts ..... Grails skripty, ktoré môžu byť volané z konzoly
├── src ..... Externé zdrojové kódy
├── target ..... Binárny výstup z kompilátora
├── test ..... Automatické testy
│   ├── functional ..... Funkcionálne testy
│   └── integration ..... Integračné testy
```

```
    └── unit ..... Jednotkové testy
├── web-app ..... Statické zdroje, ktoré nie sú spracovávané Asset pluginom
├── wrapper ..... Knižnice potrebné pre kompiláciu projektu
├── application.properties ..... Základné informácie o projekte
├── grailsw ..... Konzola pre systémy UN*X
└── grailsw.bat ..... Konzola pre systém Windows
```

Po inicializácií je takto vytvorený prázny projekt priamo pripravený na spustenie v integrovanom Tomcat serveri príkazom `grails run-app`.

Príkazmi `grails create-domain-class` *`[názov]`* resp. `grails create-controller` *`[názov]`* sa potom v takomto projekte vytvoria nové doménové triedy resp. kontroléry. Veľkou výhodou je, že projekt nie je nutné po každej zmene znova celý kompilovať a spúšťať, ale Grails automaticky vykoná potrebné zmeny.

6.2 Konfigurácia projektu

Grails nepoužíva pre konfiguráciu XML dokumenty, ako je tomu zvykom pri niektorých iných Java EE projektoch. Konfigurácia projektu je uložená takmer výlučne v adresári `%PROJECT_HOME%/grails-app/conf`, v súboroch s využitím Groovy DSL. Vďaka používaniu paradigmy *convention over configuration* sa konfigurujú iba tie nastavenia, ktoré sa líšia od predvolených nastavení. Súborová štruktúra adresára s konfiguráciou vyzerá takto:

```
%PROJECT_HOME%/grails-app/conf
├── hibernate ..... Dodatočné nastavenia ORM Hibernate
├── spring ..... Dodatočné nastavenia Spring
├── BootStrap.groovy ..... Definícia akcií pri spustení/ukončení aplikácie.
├── BuildConfig.groovy ..... Nastavenia vytvárania aplikácie, definícia závislostí.
├── Config.groovy ..... Nastavenia aplikácie a pluginov.
├── DataSource.groovy ..... Nastavenie JDBC.
└── UrlMappings.groovy ..... Pravidlá mapovania URL na kontroléry.
```

V prvom rade je potrebné nastaviť v súbore `DataSource.groovy` správne prístup do databázy v pomocou JDBC reťazca a voľbou stratégie mapovania doménových modelov do databázy. Ja som ako stratégiu mapovania v produkčnom a vývojovom prostredí zvolil `update`, teda pri spustení aplíkacie prebehne kontrola, či doménové triedy zodpovedajú štruktúre tabuliek v databázi a prípadné rozdiely sa upravia bez straty dát. Pri integračnom a funkcionálnom testovaní je použitá stratégia `create-drop`, teda pri každom spustení aplikácie sa najskôr zmaže obsah pôvodnej databázy.

V súbore `BuildConfig.groovy` bolo potrebné zadefinovať závislosti a prípadne aj repozitáre, ktoré aplikácia využíva tak, ako boli navrhnuté v kapitole 5.

Konkrétne nastavenia jednotlivých pluginov sú zadefinované v súbore `Config.groovy`. Sú tu teda zadefinované, okrem iného, aj nastavenia zabezpečenia, keďže tie podliehajú

pluginu Spring Security.

6.3 Zabezpečenie aplikácie

Pre splnenie požiadavok **F.1** je nutné aplikáciu zabezpečiť pred neautentizovaným a neautorizovaným prístupom. Túto funkcionalitu splňuje plugin Spring Security. Volaním príkazu `grails s2-quickstart cz.cvut.feld.isarg User Role` sa vytvoria tri doménové triedy, potrebné pre ukladanie informácií o užívateľoch a ich oprávnení:

`User` Entita predstavujúca užívateľa systému.

`Role` Entita predstavujúca oprávnenie užívateľa systému.

`UserRole` Entita predstavujúca vzťah medzi užívateľom a oprávnením. Obsahuje aj pomocné metódy, pre správu oprávnení užívateľa.

6.3.1 Anotácie

V kontroléroch a servisných triedach je možné používať anotáciu `@Secured`, ktorá udáva, že daná trieda alebo metóda podlieha kontrole prístupu. Navyše je možné špecifikovať, aké oprávnenia užívateľ potrebuje pre prístup k takejto metóde alebo triede:

Listing 6.1: Zabezpečenie kontroléra pomocou anotácií

```
package cz.cvut.feld.isarg
import grails.plugin.springsecurity.annotation.Secured

@Secured(['ROLE_P_PATIENT_VIEW'])
class PatientController {
        @Secured(['ROLE_P_PATIENT_EDIT'])
        def edit(Patient patient) { ... }

        @Secured(['ROLE_P_PATIENT_DELETE'])
        def delete(Patient patient) { ... }
}
```

6.3.2 Statické pravidlá

Okrem zabezpečenia na úrovni kontrolérov a servisných služieb je možné kontrolovať prístup aj priamo na úrovni URL adries. Je napríklad nevhodné, aby kontrole prístupu podliehali JavaScripty alebo CSS súbory. To sa dá ošetriť nastavením v súbore `Config.groovy`:

Listing 6.2: Nastavenie prístupu pre URL adresy

```
grails.plugin.springsecurity.controllerAnnotations.staticRules = [
  '/':           ['permitAll'],
  '/assets/**': ['permitAll'],
  '/logout/**': ['permitAll']
]
```

6.3.3 Prihlásenie a odhlásenie

Plugin Spring Security v základnom nastavení poskytuje formulár pre prihlásenie a akciu pre odhlásenie užívateľa. Ak je teda užívateľovi odmietnutý prístup k nejakej metóde, bude presmerovaný na prihlasovací formulár vygenerovaný týmto pluginom. Je samozrejme možné používať pre prihlásenie vlastný formulár, pre potreby tejto aplikácie to však nie je potrebné.

6.4 Správa súborov

Ako bolo povedané v sekcii 2.3.1, správa súborov v tejto aplikácii pozostáva z dvoch súborových rozhraní (FTP a web). Tieto dve rozhrania zdieľajú jedno spoločné súborové rozhranie, cez ktoré prebiehajú všetky súborové operácie.

6.4.1 Spoločné súborové rozhranie

Spoločné súborové rozhranie poskytuje aplikácii množinu metód pre správu súborov, či už ide o vytváranie súborov/adresárov, nahrávanie/sťahovanie dát zo súborov do streamu, premiestňovanie, vyhľadávanie, vytváranie náhľadov, atď. Z pohľadu implementácie ide o Grails servisnú triedu s názvom `FileService`. Zvyšok aplikácie teda nemusí vedieť o vnútornej reprezentácii súborov v súborovom systéme a databázi, stačí ak správne používa metódy tohto súborového rozhrania.

6.4.2 Webové rozhranie

Cez webový prehliadač sa dajú súbory spravovať pomocou súborového manažéra elFinder. Klientská časť tohto manažéra, napísaná v JavaScripte, si cez Ajax vymieňa so serverom správy vo formáte JSON. Serverová časť tohto manažéra (nazývaná *connector*) bola pôvodne napísana v jazyku PHP, ale jej funkcionalita nebola pre potreby tejto aplikácie dostatočná. Na základe dostupnej dokumentácie bol preto implementovaný nový *connector* v jazyku Groovy s použitím Grails frameworku. Tento *connector* využíva na prácu so súbormi servisnú triedu `FileService`.

6.4.3 FTP rozhranie

Ako už bolo povedané v sekcii 5.4, FTP rozhranie je implementované rozšírením existujúcej Java implementácie FTP servera s názvom Apache FtpServer. Chovanie tohto FTP servera bolo potrebné upraviť tak, aby súborové operácie prebiehali cez servisnú triedu `FileServive`. To bolo bolo prevedené implementáciou Java rozhraní `FileSystemFactory`, `FileSystemView`, `FtpFile` a `UserManager` podľa dostupnej dokumentácie [17]. Tento FTP server sa spúšťa pri štarte webovej aplikácie, čo je zadefinované v súbore `BootStrap.groovy`.

6.5 Vizualizácia súborov

Aplikácia dokáže vizualizovať rôzne typy súborov na časovej osi. Je pre to potrebné spracovávať obsah nahrávaných súborov.

6.5.1 Obrázky

Pre vytváranie obrázkov je využitá knižnica Thumbnailator, ktorá dokáže čítať rôzne formáty obrázkov. Výsledný náhľad obrázku je užívateľovi posielaný zmenšený vo formáte JPEG.

Listing 6.3: Vytvorenie náhľadu obrázku knižnicou Thumbnailator

```
int w = params.w ? Integer.parseInt(params.w) : 100
int h = params.h ? Integer.parseInt(params.h) : 100
OutputStream os = new ByteArrayOutputStream()
Thumbnails.of(file).size(w, h).outputFormat("jpg").toOutputStream(os)
```

6.5.2 Časový údaj

Pre získanie časového údaju (bod v čase alebo časový interval) je nutné prečítať túto informáciu z binárnych súborov, ktoré môžu byť rôzneho typu. V tejto sekcii je bližšie popísané získavanie tejto informácie z rôznych formátov.

6.5.2.1 European Data Format (EDF)

EDF je súbor používaný v medicíne pre zaznamenávanie časových rád. V kontexte tejto aplikácie teda pôjde najmä o záznamy z elektród. Dátum začiatku záznamu a dĺžky trvania je zapísaný v hlavičke takéhoto súboru.

Listing 6.4: Pseudokód čítania časového intervalu z EDF

```
def edfDateRange(File file) {
  // Read first 300 bytes from the file
  byte[] h = file.read(300)

  // Read date from correct position in the file
  Date startDate = h.substring(168, 184) as Date
  Integer records = h.substring(236, 244) as Integer
  Integer duration = h.substring(244, 252) as Integer
  Date endDate = startDate.clone()
  endDate.addSeconds(records * duration)
  return [startDate, endDate]
}
```

6.5.2.2 BrainLab

Produkty nemeckej firmy BrainLab používané v neurochirurgii, ukladajú dáta v dvoch súboroch. Prvý súbor má príponu `*.sts` a obsahuje metadáta, v súbore s príponou `*.sig`

sú potom uložené samotné dáta z meracieho prístroja. Pre získanie časového intervalu je však nutné prečítať oba súbory. Časový údaj je tu vyjadrený vo forme desatinného čísla zodpovedajúceho počtu sekúnd od začiatku nášho letopočtu, preto je nutné takýto údaj správne konvertovať.

Listing 6.5: Pseudokód čítania časového intervalu z BrainLab

```
def stsSigDateRange(File sts, File sig) {
  // Read number of channels from STS file
  Integer pos = sts.findFirstOccurrence("CStdCoDec")
  Integer channels = sts.readFromPosition(pos + 2, 2)

  // Read minimum and maximum time value from SIG file
  byte[8] buf
  def is = new FileInputStream(sts)
  Double tMin = -1, tMax = -1
  while(f.read(buf, 0, 8)) {
    is.skip(64*2*channels)
    Double t = buf as Double
    if (t > tMax) tMax = t
    else if (t < tMin || tMin == -1) tMin = t
  }

  // Number of seconds from 1. January 0 to 1. January 1970
  Long epoch = 62167305600

  // Seconds value to date conversion
  Date startDate = new Date(((tMin as Long) - epoch)*1000)
  Date endDate = new Date(((tMax as Long) - epoch)*1000)
  return [startDate, endDate]
}
```

6.5.2.3 Digital Imaging and Communications in Medicine (DICOM)

Ide o ďalší z formátov používaný v medicíne. Samotný súbor obsahuje množstvo rôznych druhov záznamov, pre potreby aplikácie však postačuje čítanie časovej informácie. Pre čítanie DICOM využívam existujúcu Java knižnicu dcm4che (`http://www.dcm4che.org/`).

Listing 6.6: Pseudokód čítania času z DICOM

```
def dcmDate(File file) {
  // Read DICOM file
  def dcm = new DicomInputStream(file)
  def date = new Date()

  // Iterate through DICOM entries
  dcm.iterator().each { entry ->
    if (entry.name == "DA")
      date.setDate(entry.date)
    else if (entry.name == "TM") {
      def t = entry.getString()
      def hour = t.substring(0,2)
```

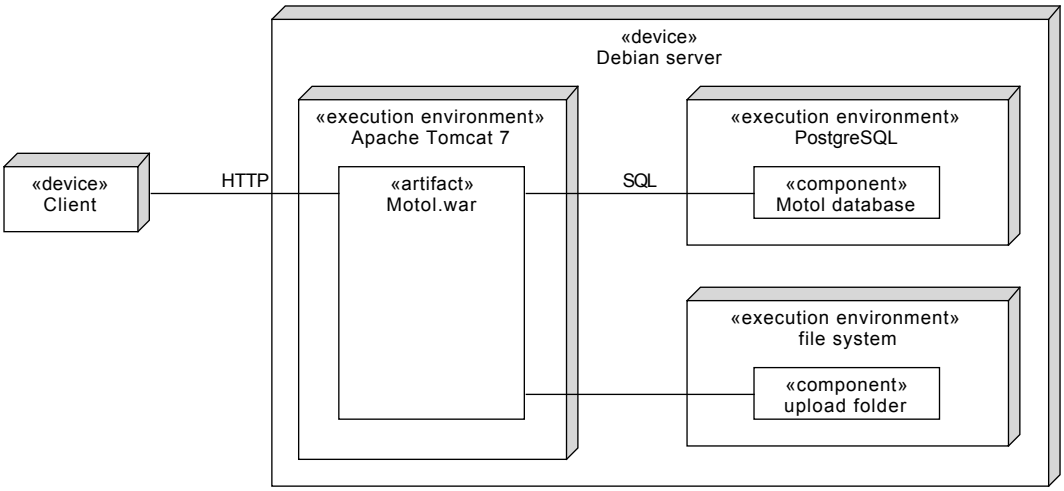
```
        def minute = t.substring(2,4)
        def second = t.substring(4,6)
        date.setTime(hour, minute, second)
      }
    }
    return date
}
```

6.6 Nasadenie

Aplikácia je distribuovaná vo forme WAR balíčku, ktorý obsahuje všetko potrebné pre nasadenie do produkčného prostredia. Aplikácia predpokladá chod na serveri s OS Debian 7. Na serveri musí byť nainštalovaný webový kontajner Tomcat 7, ako aj relačná databáza PostgreSQL 9.4 v štandardnom nastavení.

6.6.1 Model nasadenia

Model nasadenia je znázornený na diagrame 6.1.



Obr. 6.1: UML Structure and Deployment: Model nasadenia

6.6.2 Postup nasadenia

Nasledujúce príkazy je nutné vykonávať v termináli pod užívateľom `root`.

1. Inštalácia Apache Tomcat 7 prebieha príkazom `apt-get`:

```
apt-get update
apt-get install openjdk-7-jre tomcat7 tomcat7-docs tomcat7-admin
```

Následne je nutné pridať do Tomcat serveru nového užívateľa. To sa urobí úpravou súboru `tomcat-users.xml`:

```
nano /etc/tomcat7/tomcat-users.xml
```

Sekcia `tomcat-users` sa upraví nasledovne:

```
<tomcat-users>
    <role rolename="manager-gui" />
    <role rolename="admin-gui" />
    <user username="admin" password="password"
      roles="manager-gui,admin-gui"/>
</tomcat-users>
```

Pre aktualizáciu zmien je nutné ešte reštartovať server:

```
/etc/init.d/tomcat7 restart
```

Na záver je vhodné skontrolovať, či Tomcat Web Application Manager funguje korektne. To sa urobí otvorením adresy `http://your_ip_address:8080/manager/` vo webovom prehliadači, kde by malo byť možné sa prihlásiť pomocou mena `admin` a hesla `password`.

2. Inštalácia PostgreSQL prebieha opäť príkazom `apt-get`:

```
apt-get install postgresql postgresql-contrib
```

Spustenie Postgres terminálu:

```
su - postgres
psql
```

V termináli je potrebné nastaviť prístupové heslo:

```
ALTER USER postgres WITH PASSWORD 'postgres';
\q
```

Následne sa vytvorí nová databáza príkazom:

```
createdb motol -h localhost -U postgres
```

3. Vytvorenie adresára na ukladanie súborov pacienta:

```
mkdir /home/motol-data
chmod 777 /home/motol-data
```

4. Posledným krokom je nasadenie WAR balíčka na server. To sa urobí pomocou webového rozhrania `http://your_ip_address:8080/manager/`. Po tomto kroku je aplikácia dostupná na adrese `http://your_ip_address/`, kde je možné sa prihlásiť pomocou mena `admin` a hesla `test`. FTP server počúva na porte `60021` a je možné sa k nemu prihlásiť použitím rovnakých údajov.

Kapitola 7

Testovanie

Testovanie je dôležitou súčasťou procesu tvorby aplikácie. Automatické testovanie tejto aplikácie prebieha v troch fázach. Najskôr sa otestujú niektoré zložitejšie komponenty, izolovane od zvyšku systému v *jednotkových testoch*. Následne sa pomocou *integračných testov* testuje previazanosť týchto komponentov v rámci kontextu celej aplikácie. Systém ako celok je v úlohe black-boxu testovaný cez webový prehliadač pomocou *funkcionálnych testov*. V poslednom kroku je výsledný systém nasadený u zákazníka, ktorý ho testuje vlastnou sadou testov v *akceptačnom teste*.

Jednotlivé kroky v prvých troch fázach testovania boli zadefinované pomocou frameworku Spock (`https://code.google.com/p/spock/`). Pre zlepšenie čitateľnosti zápisu testu, sa v tomto frameworku používá šesť rôznych blokov, pomenovaných `setup`, `when`, `then`, `expect`, `cleanup` a `where`. Každý test predstavuje metódu v testovacej triede a tieto testy sa spúšťajú v lineárnom poradí.

Autmatické testy aplikácie je možné spustiť príkazom `grails test-app`.

7.1 Jednotkové testy

Na najnižšej úrovni sa aplikácia testuje v *jednotkových testoch (unit tests)*. Týmto testom sa testujú jednotlivé triedy a ich metódy oddelene od zvyšku aplikácie. Výhodou takéhoto testu je, že prebehne rýchlo, keďže sa nemusia načítavať všetky komponenty aplikácie, nepristupuje sa do databázy, apod. Takýto test je vhodný napríklad na testovanie validačných pravidiel doménových tried alebo servisných tried, kde je väčšina business logiky aplikácie.

Je zrejmé, že pri testovaní jednotlivých komponentov izolovane je potrebné ošetriť vzájomné závislosti týchto komponent. To sa rieši tzv. *mockovaním*. *Mockovanie* je technika, ktorou sa poskytne falošná verzia tried alebo metód, na ktoré je testovaná komponenta závislá. [12, s. 192].

Listing 7.1: Jednotkový test servisnej triedy s ukážkou mockovania

```
@TestFor(FileService)
@Mock(Folder)
```

```
class FileServiceSpec extends Specification {
  def "test basic folder operations"() {
    setup:
      Folder f_sub = service.createFolder("sub_folder", null)
      Folder f_root = service.createFolder("root_folder", f_sub)

      // Create a mock, that can be called 1 to 4 times
      def mockPatientService = mockFor(PatientService)
      mockPatientService.demand.list(1..4) {
        -> [new Patient(folder: f_root)] }
      service.patientService = mockPatientService.createMock()

    expect:
      service.getFolder("root_folder").name == "root_folder"
      service.getFolder("root_folder/sub_folder").name == "sub_folder"
      shouldFail { service.getFolder("fake_folder") }

    when:
      service.delete(f_sub.id)
    then:
      shouldFail { service.getFolder("root_folder/sub_folder") }
  }
}
```

7.2 Integračné testy

Integračný test testuje, ako medzi sebou spolupracujú jednotlivé komponenty nejakého väčšieho celku aplikácie. Týmto testom testujem najmä kontroléry, ktoré sú typicky závislé na viacerých servisných a doménových triedach. Pre spustenie takéhoto testu, sa musí načítať celé Grails prostredie spolu so všetkými pluginmi a prístupom do testovacej databázy, preto je o niečo pomalší ako jednotkový test.

7.3 Funkcionálne testy

Ešte vyšším stupňom testovania webovej aplikácie je funkcionálny test. Tento test prebieha priamo v užívateľovom prehliadači, ktorý posiela HTTP požiadavky a kontroluje, či aplikácia funguje korektne. Testovať aplikáciu manuálne týmto spôsobom by bolo príliš zdĺhavé, využívam preto framework pre automatizáciu funkcionálnych testov Geb (`http://www.gebish.org/`), ktorý dobre spolupracuje už s používaným testovacím frameworkom Spock. Tento framework používa nástroj Selenium WebDriver, ktorý podporuje automatizáciu najpoužívanejších webových prehliadačov. Framework Geb používa pre definovanie testov syntax podobnú knižnici jQuery.

Každý funkcionálny test pozostáva okrem samotnej špecifikácie testu aj z tzv. *page objektov*, ktorý definuje, s ktorými elementami jednej konkrétnej webovej stránky sa bude v danom teste pracovať. Takýto objekt sa môže ďalej skladať z viacerých modulov, teda znovupoužiteľnú abstraktnú reprezentáciu častí stránky (napríklad navigačné

menu, tabuľka, formulár, apod.). Používaním *page objektov* sa zjednoduší zápis a zlepší čitateľnosť samotného zápisu testu.

Listing 7.2: *Page objekt* stránky pre pridanie pacienta

```
class AddPage extends Page {
  static url = "patient/add"
  static at = { title == "Add patient" }
  static content = {
    // Define some elements which we will use in the test
    addForm { $("form") }
    addButton { $("input", value: "Save patient") }
  }
}
```

Listing 7.3: *Page objekt* stránky s výpisom pacientov

```
class IndexPage extends Page {
  static url = "patient/index"
  static at = { title == "Patients" }
  static content = {
    addPatientButton(to: AddPage) { $("a.btn", text: "New patient") }
    patientsTable { $("div.grid table", 0) }
    patientsTableRows(required: false) {
      patientsTable.find("tbody").find("tr") }
    patientRow { module TableRow, patientsTableRows[it] }
  }
}
```

Listing 7.4: Špecifikácia testu, ktorý bude vykonaný v užívateľovom prehliadači

```
@Stepwise
class PatientSpec extends GebSpec {
  def "list empty patients"() {
    when:
      to IndexPage
    then:
      // Index page contains one table with zero rows
      patientsTable.size() == 1
      patientsTableRows.size() == 0
  }

  def "add a patient"() {
    when: addPatientButton.click()
    then: at AddPage
  }

  def "fill details in the form"() {
    when:
      addForm.'detail.name' = "meno"
      addForm.'detail.surname' = "priezvisko"
      addButton.click()
    then:
      at IndexPage
```

```
  }

  def "search newly added patient and delete him"() {
    when:
      search_name = "meno"
      search_name = "priezvisko"
      withConfirm { patientRow(0).action('Delete').click() }
    then:
      at IndexPage
  }
}
```

7.4 Akceptačný test

Najvyšším stupňom testovania tejto aplikácie bol akceptačný test nasadenej aplikácie v testovacom prostredí u zadávateľa práce. Aplikácia bola nasadená na virtuálny server ČVUT FEL, kde si mohol zadávateľ priamo overiť funkčnosť aplikácie. Chyby nájdené v tomto testovaní boli v čo najkratšej dobe opravené.

Záver

V úvode tejto práci som analyzoval exitujúce riešenie používané výskumnou skupinou ISARG pre správu pacientov, zaradených do epileptochirurgického programu FN Motol. Používané riešenie pre správu pacientov výskumnej skupine nepostačuje a ďalšie existujúce riešenia sú pre potreby skupiny nevhodné, bolo preto potrebné implementovať nový informačný systém na mieru.

V práci som navrhol a implementoval nový informačný systém pre správu pacientov, ktorý rozširuje funkcie súčasného systému najmä o organizovanie pacientov do štúdií, správu súborov cez webové aj FTP rozhranie, nastavovanie prístupových práv užívateľov a štúdií a vizualizovanie vybraných typov súborov na časovej osi. Stručne som popísal najdôležitejšie aspekty realizácie a technológie, ktoré boli pri realizácií použité.

Prínos práce

Tento informačný systém bol nasadený na produkčný server ČVUT FEL a po dôkladnom otestovaní v štyroch stupňoch testovania je pripravený na použitie v praxi. Nové funkcie, ktoré tento systém prináša, zefektívnia spravovanie dát pacientov zaradených do výskumného programu.

Čistý čas vynaložený prácou na tomto projekte som odhadol na 450 hodín. Pri predpokladaných mzdových nákladoch vo výške 500 Kč/hod teda realizácia tohto informačného systému formou bakalárskej práce priniesla úsporu nákladov 225.000 Kč.

Systém bol vyvinutý s použitím moderných technológií a do budúcnosti je jednoduchým spôsobom rozšíriteľný o nové funkcie. Jedným z možností rozšírenia systému je napríklad podpora pre vizualizáciu ďalších typov súborov alebo štatistické vyhodnocovanie údajov, uložených do systému.

V neposlednom rade mala táto práca pre mňa aj osobný prínos, keďže som si rozšíril praktické a teoretické zručnosti s vývojom webovej aplikácie.

Literatúra

[1] Roman, G. C.: A Taxonomy of Current Issues in Requirements Engineering. *Computer*, April 1985, ISSN 0018-9162.

[2] Hickson, I.: Acid Tests - The Web Standards Project. March 2008, [Cited 2015-03-10]. Dostupné z: `http://www.acidtests.org/`

[3] Postel, J.: *RFC 765: File Transfer Protocol specification.* June 1980. Dostupné z: `https://tools.ietf.org/html/rfc765`

[4] J. Postel, J. R.: *RFC 959: File Transfer Protocol (FTP).* Network Working Group, October 1985. Dostupné z: `https://tools.ietf.org/html/rfc959`

[5] M. Horowitz, S. L., Cygnus Solutions: *RFC 2228: FTP Security Extensions.* Network Working Group, October 1997. Dostupné z: `https://tools.ietf.org/html/rfc2228`

[6] Arlow, J.; Neustadt, I.: *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition).* Addison-Wesley Professional, 2005, ISBN 978-0321321275.

[7] Gamma, E.; Helm, R.; Johnson, R.; aj.: *Návrh programů pomocí vzorů.* Grada, 2003, ISBN 978-8024703022.

[8] Business Process and Data with MVC (Model-View-Controller). 2015, [Cited 2015-04-19]. Dostupné z: `https://leahayes.wordpress.com/2011/04/21/business-process-and-data-with-mvc-model-view-controller/`

[9] Galitz, W. O.: *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques (3th Edition).* Wiley, 2007, ISBN 978-0470053423.

[10] Garrett, J. J.: *The Elements of User Experience: User-Centered Design for the Web and Beyond (2nd Edition).* New Riders, 2010, ISBN 978-0321683687.

[11] König, D.; King, P.; Laforge, G.; aj.: *Groovy in Action, Second Edition*. Manning, 2009, ISBN 978-1935182443.

[12] Smith, G.; Ledbrook, P.: *Grails in Action, Second Edition*. Manning, 2014, ISBN 978-1617290961.

[13] DB-Engines Ranking of Relational DBMS. March 2015, [Cited 2015-03-10]. Dostupné z: `http://db-engines.com/en/ranking/relational+dbms`

[14] Castro, E.; Hyslop, B.: *HTML5 & CSS3 Visual QuickStart Guide (7th Edition)*. Peachpit Press, 2011, ISBN 978-0321719614.

[15] Flanagan, D.: *JavaScript: The Definitive Guide (4th Edition)*. O'Reilly Media, 2001, ISBN 978-0596000486.

[16] Leading JavaScript technologies share on the web. 2015, [Cited 2015-03-10]. Dostupné z: `https://www.similartech.com/categories/javascript`

[17] FtpServer Documentation – Apache MINA. 2015, [Cited 2015-04-13]. Dostupné z: `https://mina.apache.org/ftpserver-project/documentation.html`

Dodatok **A**

Zoznam použitých skratiek

Ajax asynchronous JavaScript and XML

CRUD Create, read, update and delete

CSS Cascading Style Sheets

ČVUT České vysoké učení technické

DICOM Digital Imaging and Communications in Medicine

DOM Document Object Model

DSL Domain-specific language

ECoG Electrocorticography

EDF European Data Format

EEG Electroencephalogram

FTP File Transfer Protocol

GUI Graphical User Interface

HTML HyperText Markup Language

i18n Internationalization and localization

IDE Integrated development environment

IoC Inversion of control

ISARG Intracranial Signal Analysis & Research Group

JDBC Java Database Connectivity

JSON JavaScript Object Notation

JVM Java virtual machine

MIME Multipurpose Internet Mail Extensions

MVC Model–view–controller

NAS Network-attached storage

ORM Object-relational mapping

PHP PHP: Hypertext Preprocessor

PSČ Poštové smerové číslo

RDBMS Relational database management system

RFC Request for Comments

SFTP SSH File Transfer Protocol

SMB Server Message Block

SQL Structured Query Language

SSH Secure Shell

UI User interface

W3C World Wide Web Consortium

WebDAV Web Distributed Authoring and Versioning

XML Extensible Markup Language

Dodatok **B**

Zoznam použitých pojmov

cloud úložisko Služba pre ukladanie dát, kde fyzický hardware je poskytovateľom takejto služby rozložený na viacero serverov, často v rôznych geografických lokáciách a ktorý využívajú viacerí používatelia pre ukladanie dát. Výhodou pre používateľa takejto služby je, že odpadá nutnosť prevádzkovať vlastný server.

convention over configuration Koncept softwarového designu, ktorý si kladie za cieľ minimalizovať počet konfiguračných rozhodnutí, ktoré musí vývojár softwaru vykonať. Explicitne sa konfigurujú sa iba tie nastavenia, ktoré sa odlišujú od predvoleného (konvenčného) nastavenia.

dependency injection Konkrétny typ *inversion of control*, kde namiesto toho, aby trieda sama získavala inštancie ďalších tried, na ktorých je závislá, sú jej tieto inštancie dodané (injektované) pri inicializácii tejto triedy.

doménovo špecifický jazyk Jazyk, ktorého vyjadrovacia schopnosť je obmedzená na jednu konkrétnu aplikačnú doménu.

flash message Správa, ktorá sa zašle užívateľovi v nasledujúcom HTTP požiadavku po prevedení nejakej akcie. Môže ísť napríklad o notifikáciu alebo chybovú hlášku.

inversion of control Návrhový vzor, v ktorom použitá knižnica volá metódy aplikácie. Častým typom IoC je *dependency injection*.

objektovo-relačné mapovanie Technika programovania pre automatickú konverziu medzi triedami aplikácie, ktoré sú napísané v objektovo orientovanom jazyku a relačnou databázou.

syntaktický cukor Alternatívna syntax programovacieho jazyka, ktorá si kladie za cieľ zlepšiť čitateľnosť zdrojového kódu.

table-per-hierarchy Stratégia pre mapovanie dedičnosti entít v ORM systéme Hibernate. Podstatou tejto stratégie je, že všetky triedy v hierarchii entít sú do databázy

namapované na jednu spoločnú tabuľku. Táto tabuľka navyše obsahuje tzv. *discriminator column*, ktorým sa rozlišuje, ktorá inštancia podtriedy v danej hierarchii je reprezentovaná záznamom v tabuľke. Stĺpce výslednej tabuľky tvorí zjednotenie atribútov všetkých podtried v hierarchii.

table-per-subclass Stratégia pre mapovanie dedičnosti entít v ORM systéme Hibernate. Podstatou tejto stratégie je, že každá trieda v hierarchii entít je do databázy uložená v samostatnej tabuľke.

Dodatok C

Obsah priloženého CD

- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF
- `motol.war` ..... inštalačný balíček implementácie vo formáte WAR
- `readme.txt` ..... stručný popis obsahu CD a pokyny na inštaláciu