VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANALYTICKÉ WEBOVÉ PROSTŘEDÍ PRO ZPRACOVÁNÍ ZACHYCENÉ SÍŤOVÉ KOMUNIKACE

NETWORK FORENSIC ANALYTICAL WEB-BASED PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ AMBROŽ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN PLUSKAL**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/20375/2017/xambro11

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Ambrož Tomáš, Bc.**
Obor: Počítačové sítě a komunikace
Téma: **Analytické webové prostředí pro zpracování zachycené síťové komunikace**
Network Forensic Analytical Web-Based Platform
Kategorie: Web

Pokyny:

1. Seznamte se s aktuálním řešením zpracování síťové komunikace nástrojem Netfox Detective.
2. Proveďte návrh webového analytického prostředí pro zobrazení detailů síťové komunikace a extrahovaných informací z aplikačních protokolů.
3. Implementujte s využitím Dockeru a DotVVM.
4. Proveďte zhodnocení výsledků z pohledu uživatelské přívětivosti, ergonomie práce a srovnejte s alternativními nástroji.

Literatura:

- Strauss, D. (2017). *C# 7 and .NET Core Cookbook*. Birmingham: Packt Publishing.
- Price, M. (2017). *C# 7 and .NET Core modern cross-platform development : create powerful cross-platform applications using C# 7, .NET Core, and Visual Studio 2017 or Visual Studio Code*. Birmingham, UK: Packt Publishing.
- MATOUŠEK, P., PLUSKAL, J., RYŠAVÝ, O., VESELÝ, V., KMEŤ, M., KARPÍŠEK, F. and VYMLÁTIL, M. (2015). *Advanced Techniques for Reconstruction of Incomplete Network Data*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering., vol. 2015, no. 157, pp. 69-84. ISSN 1867-8211.

Při obhajobě semestrální části projektu je požadováno:

- body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Pluskal Jan, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

V současnosti probíhá velká část komunikace skrze počítačové sítě. Objem této komunikace se každým rokem zvětšuje. To má za následek, že se zvyšují i nároky na výpočetní výkon. Za účelem zvýšení výpočetního výkonu se vyplatí proces zpracování komunikace pro forenzní účely paralelizovat. Výzkumná skupina NES@FIT v jednom ze svých projektů vytvořila nástroj Netfox Detective. Z tohoto nástroje je v plánu vytvořit distribuovaný systém pro zpracování zachycené komunikace za účelem forenzní analýzy. Jednou z částí je rozhraní, pomocí kterého by se distribuovaný systém ovládal. V této diplomové práci se budu zabývat tvorbou webového rozhraní pro nástroj Netfox Detective, který je v současné době desktopovou aplikací. Webové rozhraní by se poté dalo po drobnějších úpravách použít pro paralelizovanou variantu aplikace. Webové rozhraní bude zprostředkovávat stejné informace jako desktopová varianta. Pro získání informací pro forenzní analýzu bude využívat stejně jako desktopová varianta framework Netfox Framework. Výhoda webového rozhraní oproti desktopové variantě uživatelského rozhraní je v tom, že uživateli k tomu, aby mohl přistoupit k webovému rozhraní, postačuje zařízení s webovým prohlížečem. Což znamená, že uživatel může pracovat na jakémkoliv operačním systému.

Abstract

At present a big part of communication passes off through computer network. Amount of the communication magnifys every year. It has a consequence that claims on computing power raise. The procedure of processsing of the communication for forensic purposes pays off to parallel for the purpose of increase of computing power. Research group NES@FIT created an instrument Netfox Detective in one of its projects. From this instument it is planed to create a distibuted system for the processing of intercepted communication for the purpose of forensic analysis. Interface is one of parts with the help of its the distributed system will be operated. In my theses I will concern with the creation of the web interface for the instument Netfox Detective which is a desktop application presently. Web interface use, after little modifications, for paralleled version of application. Web interface will mediate the same informations as the desktop version. For the obtaining of informations for forensic analysis it will use framework Netfox Framework identically as the desktop version. Advantage of web interface compared to the desktop version is that user who approach to web interface need device with web browser. It means that user can work whichever operation system.

Klíčová slova

Netfox Detective, DotVVM, HTML, Bootstrap, HangFire

Keywords

Netfox Detective, DotVVM, HTML, Bootstrap, HangFire

Citace

AMBROŽ, Tomáš. *Analytické webové prostředí pro zpracování zachycené síťové komunikace*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Pluskal

Analytické webové prostředí pro zpracování zachycené síťové komunikace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Pluskala. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Ambrož
23. května 2018

Poděkování

Rád bych poděkoval Ing. Janu Pluskalovi za vedení mojí diplomové práce.

Obsah

Kapitola 1

Úvod

Komunikace na internetu neustále roste. Z pohledu forenzní analýzy to znamená, že objem dat, která jsou potřeba zpracovat, roste. Důsledkem této skutečnosti je větší potřeba orgánů činných v trestním řízeni mít nástroj, který by umožnit pohodlně získat informace pro forenzní analýzu ze zachycené síťové komunikace, čemuž se věnuje výzkumná skupina NES@FIT v jednom ze svých projektů. Jedná se o projekt Netfox, ve kterém výzkumná skupina spolupracuje s Ministerstvem Vnitra České Republiky. Projekt se zabývá vývojem nástroje pro získávání forenzních dat pro vyšetřování trestné činnosti orgány činnými v trestním řízení. Nástroj k tomuto určený se nazývá Netfox Detective. Jedná se o desktopovou aplikaci. Aplikace umí ze vstupních souborů se zachycenou komunikací extrahovat informace pro forenzní analýzu. Cílem diplomové práce je vytvořit webové rozhraní, které by bylo náhradou grafického uživatelského rozhraní desktopové aplikace. Aby webové rozhraní poskytovalo stejná data jako desktopová varianta, tak bude používat pro extrahování dat k forenzním účelům framework Netfox Framework. Na framework je napojena i desktopová aplikace.

Druhá kapitola diplomové práce bude věnována popsání současného stavu implementace aplikace Netfox Detective. Budou zde zmíněny technologie, které aplikace využívá pro implementaci. Dále budou vysvětleny návrhové vzory, které se v implementaci budou vyskytovat a je nutné je vysvětit pro pochopení, jak celá aplikace funguje. Nakonec se budeme v kapitole zabývat funkcemi některých důležitých komponent a postupem zpracování vstupního souboru se zachycenou komunikací na extrahovaná data užitečná pro orgány činné v trestním řízení.

Třetí kapitola se zaměří na návrh implementace webového rozhraní. Návrh implementace nastiňuje jaké uživatelské role by ve webovém rozhraní měly být. U rolí se pak specifikuje, co každá role umožňuje uživateli provádět za akce a co se mu může zobrazit. Dále se v kapitole vymezí, co by měla umožňovat správa uživatelů a správa vyšetřování. Poslední konkretizovanou částí jsou požadavky na použití určitých technologií při implementaci webového rozhraní. Na závěr budou popsány návrhy podoby uživatelského rozhraní některých částí webového rozhraní.

Čtvrtá kapitola se pak bude zabývat implementací jednotlivých částí webového rozhraní. Mezi tyto části patří správy uživatelů, správy vyšetřování a tou nejdůležitější částí propojení webového rozhraní a Netfox Framework. V popisu propojení bude zmíněno například jak probíhá zpracování síťové komunikace ze vstupních souborů, extrahování forenzních dat za účelem analýzy nebo zobrazení extrahování dat. Dále budou také popsány použité frameworky a knihovny, které budou použity při implementaci. U frameworku DotVVM bude ještě navíc vysvětleno, jak probíhá zpracování požadavku na zobrazení jakékoli stránky.

V páté kapitole se seznámíme, jak bude otestována implementace. Takže v kapitole nalezneme popis jednotlivých testů týkajících se dílčích částí implementace jako správy uživatelů, správy vyšetřování a propojení s Netfox Framework. Na závěr bude pak popsáno, jak testy dopadly.

V šesté kapitole, což je poslední kapitola této práce, nalezneme shrnutí čeho se v práci dosáhlo. Dále bude taky rozebráno, jak je webové rozhraní příjemné z pohledu uživatelské přívětivosti a ergonomie.

Kapitola 2

Netfox Detective

Nástroje pro forenzní účely v počítačových sítích mohou být rozděleny do dvou hlavních skupin. První skupinou nástrojů je NFAT (Network Forensic Analysis Tool). Tato skupina nástrojů dovoluje správci sítě monitorovat počítačovou síť, umožňuje shromažďovat informace o probíhajícím provozu na počítačové síti a pomáhají při vyšetřování nelegální činnosti na počítačových sítích. Druhá skupina NSM (Network Security and Monitoring) se spíše zaměřuje na monitorování a správu počítačové sítě. Mezi zástupce této skupiny se řadí například Wireshark, TCPDump či Microsoft Network Monitor.

Netfox Detective lze zařadit do skupiny NFAT. Je to tedy nástroj určený pro extrahování obsahu z aplikačních protokolů za účelem forenzní analýzy síťového provozu. Netfox Detective je vyvíjen v rámci výzkumné skupiny NES@FIT na Fakultě informačních technologií Vysokého učení technického v Brně, která pracuje na projektu Netfox. Projekt je také podporován Ministerstvem Vnitra České republiky. Nástroj Netfox Detective je navrhnut pro pokročilou rekonstrukci a analýzu zachycených dat síťového provozu se zaměřením na emaily (včetně webových emailů), rekonstrukci HTTP (Hyper Text Transport Protocol), detekci a rekonstrukci hovorů přes VoIP (Voice over IP). Framework používá pokročilých technik a heuristik pro seskupení odposlechnutých dat, identifikaci provozu na aplikační vrstvě, rekonstrukce původních konverzací a zobrazení získaných dat z aplikační vrstvy pro vyšetřovatele.

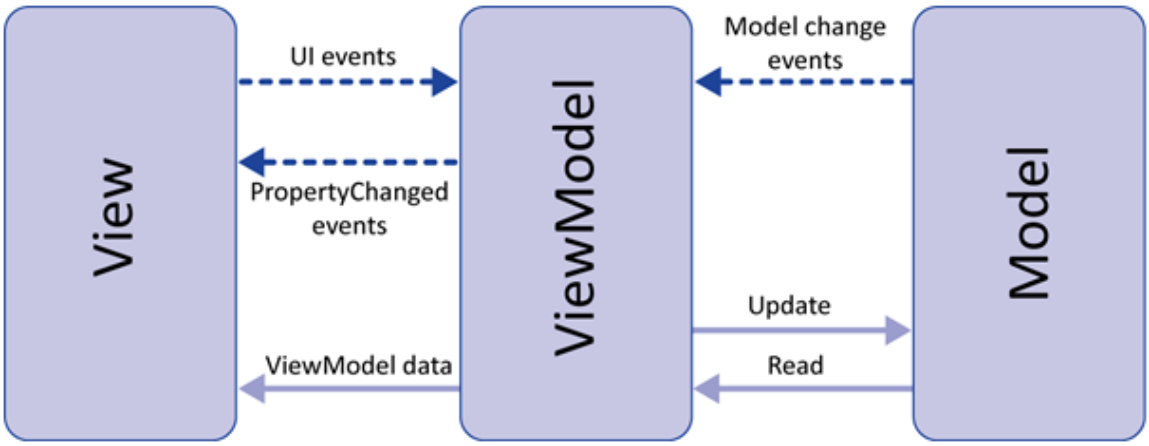
Netfox Detective je v současné chvíli vyvíjen na platformě .NET Framework 4.7, konkrétně v programovacím jazyce C#. Jedná se o technologii vyvíjenou společností Microsoft. Netfox Detective je spustitelný na počítačích s operačními systémy Windows 7 a novějšími verzemi tohoto operačního systému[10, 11]. Při vývoji aplikace byly použity dva návrhové vzory, jedná se o návrhový vzor MVVM a o DI (Dependency Injection). DI je návrhový vzor, který umožňuje snížit závislosti mezi jednotlivými komponentami aplikace. Jednou z výhod snížení závislostí mezi komponentami aplikace je ta, že jednotlivé komponenty aplikace se lépe testují[7]. Návrhový vzor MVVM bude v této kapitole popsán podrobněji, protože je nesmírně důležitý pro pochopení, jak celá aplikace funguje a bude se podle něj implementovat i webové rozhraní.

2.1 MVVM

MVVM je návrhový vzor pomocí kterého lze při vývoji aplikace oddělit uživatelské rozhraní od logiky aplikace. Po rozdělení těchto dvou částí aplikace se zdrojový kód aplikace stává přehlednější a následné úpravy aplikace či vývoj a rozšíření aplikace jsou jednoduší. Další

výhodou, kterou oddělením uživatelského rozhraní od logiky aplikace získáme, je zlepšení možnosti testovat jednotlivé části aplikace[2, 4]. Návrhový vzor MVVM se skládá z následujících částí[4]:

- *Model* je třída, která zajišťuje definici dat, se kterými logika aplikace pracuje. Například pokud by se jednalo o webovou aplikaci, tak model bude definovat, jak se data budou ukládat do databáze a jakým způsobem se budou z databáze získávat.
- *View* je třída definující uživatelské rozhraní. Bude tedy definovat, kde se dané informace zobrazí a jakým způsobem. Případně může informovat zasláním zprávy ViewModel o nějaké události uživatelského rozhraní, kupříkladu kliknutí na tlačítko.
- *ViewModel* je třídou, kterou můžeme chápat, jako spojovací článek mezi Modelem a View, jak může být patrné z obrázku 2.1. Každá třída View má k sobě korespondující třídu ViewModel. ViewModel přijímá data od Modelu a ty zpracuje tak, aby do View posléze tyto data předal v korektním formátu, kterému třída View rozumí. Pokud dojde ke změně dat Modelu, tak o tom ViewModel informuje pomocí notifikace View. A stejně tak i pokud dojde k nějaké události v uživatelském rozhraní, například kliknutí na tlačítko nebo vyplnění textového pole, zašle View do ViewModelu notifikaci o události a ten následně událost vyhodnotí a podle toho změní data v Modelu.

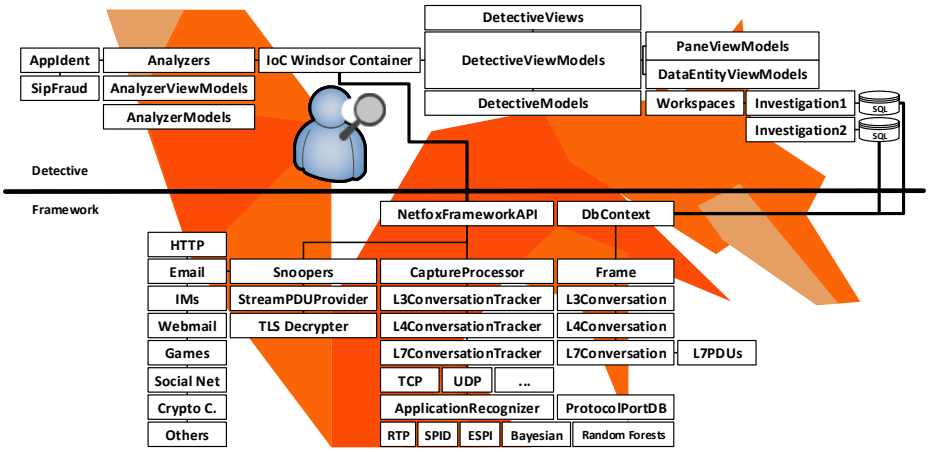


Obrázek 2.1: Vztahy mezi vrstvami View, Viewmodel, Model.[4]

2.2 Architektura Netfox Detective

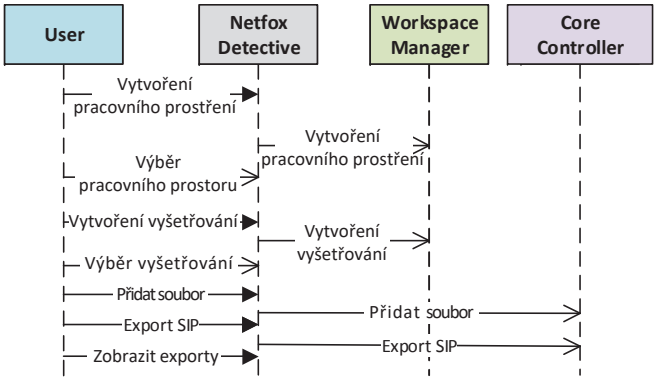
Nyní si popíšeme, jak aplikace a její důležité části fungují. Když se aplikace spouští, tak se musí vytvořit nový pracovní prostor či načíst již vytvořený. Pracovní prostor reprezentuje adresářovou strukturu. V pracovním prostoru se ukládají jednotlivá vyšetřování. Pracovní prostor může mít jedno či více vyšetřování. V rámci vyšetřování se ukládají soubory s odposlechnutou síťovou komunikací. Tato skutečnost je patrná i z obrázku 2.3. Nástroj můžeme rozdělit na dvě základní části. Těmito částmi jsou Framework a Detective. Rozdělení je naznačeno na obrázku 2.2. Ve spodní části obrázku se nachází množina komponent představující část Netfox Framework, která je zodpovědná za zpracování vstupních souborů

se zachycenou komunikací. Ve vrchní části obrázku jsou pak komponenty patřící do grafického uživatelského rozhraní aplikace Netfox Detective. Grafické rozhraní pak zajišťuje interakci s uživatelem a zobrazení výsledků zpracovaných částí Netfox Framework[10]. Řízení zpracování dat má na starost komponenta NetfoxFrameworkAPI. Řídící komponenta komunikuje s komponentami CaptureProcessor a Snoopers. Architektura propojení řídící komponenty NetfoxFrameworkAPI je naznačená na obrázku 2.2. CaptureProcessor rozdělí



Obrázek 2.2: Architektura nástroje Netfox Detective.[10]

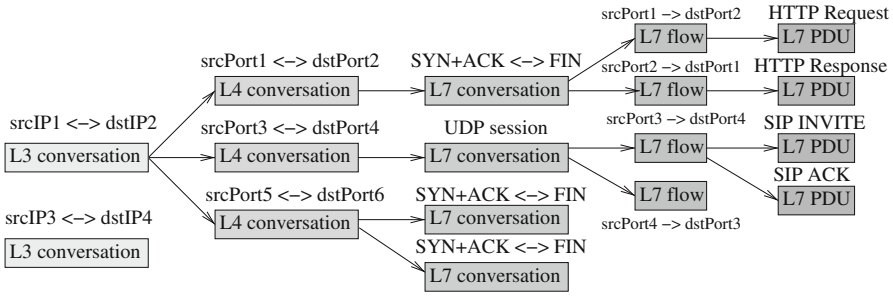
jednotlivé rámce v komunikaci podle jejich příslušnosti k příslušné konverzaci. Konverzace lze chápat jako pár kolekcí shromážděných rámců v odchozím i v příchozím směru síťového provozu seřazených podle časového razítka. Zpracování rámců na konverzace začíná zpracováním L3 konverzací. Určení, do které L3 konverzace rámec patří, probíhá na základě IP adres odesilatele a příjemce. Následně se zpracují L4 konverzace. Ty jsou definovány pomocí použitých čísel portů a použitého transportního protokolu. Posledním typem konverzací, které se vytváří, jsou L7 konverzace. V případě, že se v zachycených datech vyskytnou dvě



Obrázek 2.3: Schéma založení pracovního prostoru a přidání vstupního souboru.[11]

komunikace, které využívají transportní protokol UDP a komunikují na stejných zdrojových a cílových portech, tak není možné tyto komunikace rozlišit. Příkladem této skutečnosti mo-

hou být dvě aplikace využívající protokol SIP. Obě aplikace budou komunikovat na stejném zdrojovém i cílovém portu, a to například na portu 5060 pro všechny SIP konverzace. Proto může být L4 UDP konverzace považována za L7 konverzaci. Nyní popsané zjištění můžeme vidět i na obrázku 2.4, kde je naznačeno rozdělení rámců zachyceného provozu do jednotlivých typů konverzací.



Obrázek 2.4: Zpracovaní rámců z komunikace na konverzace.[3]

2.2.1 Pmlib

Jedná se o knihovnu napsanou v jazyce C#. Knihovna poskytuje rozhraní pro manipulaci se soubory se zachycenou komunikací. Knihovna podporuje různé typy formátů souborů, do kterých lze ukládat zachycenou síťovou komunikaci. Jsou to například Wireshark/ TCP-Dump LibPcap, Microsoft Network Monitor či Pcap-ng formát. Dále knihovna PmLib zpracovává vstupní soubory, aby poskytla možnost přistoupit k jednotlivým síťovým vrstvám (L2, L3 a L4) rámců uložených v souboru se zachycenou komunikací. Pro zpracování rámců využívá knihovnu PacketDotNet 0.13. Tato knihovna podporuje protokoly Ethernet, LinuxSSL, PPP, PPPoE, IP, TCP a další.[9]

2.2.2 ApplicationRecognizer

Za bližší zmínění stojí rozhodně jedna z nejdůležitějších komponent nástroje Netfox Detective nazývající se ApplicationRecognizer. Komponenta má na starost identifikaci aplikačního protokolu, který se v komunikaci vyskytuje. Existuje několik způsobů, jak aplikační protokol určit. Nejjednodušší metodou je rozhodovat na základě čísel portů, která jsou přiřazena organizací IANA (Internet Assigned Numbers Authority). Bohužel tato metoda selhává na konverzacích, kde aplikace používají dynamicky přidělené porty, peer-to-peer komunikaci, video streaming atd.

Pokročilé metody zkoumají obsah nesených dat a pro určení aplikačního protokolu se snaží nalézt nějaký charakteristický vzor, který se může vyskytovat ať už v hlavičce nebo v nesených datech. Pokročilé metody, které jsou použity v nástroji Netfox Detective jsou dvě:

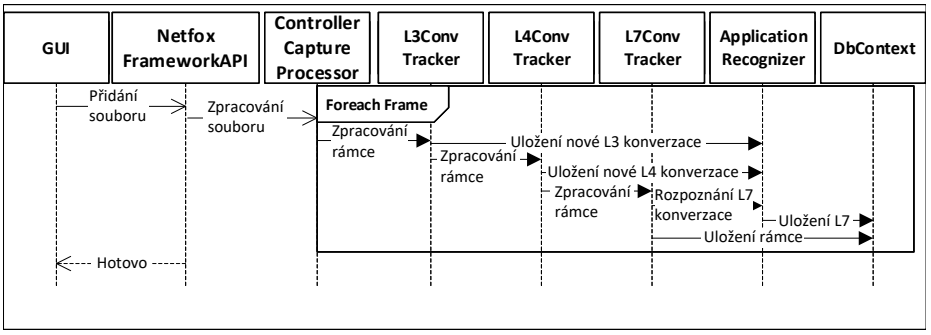
- **RTP fingerprinting**
  Používá se pokud nelze určit protokol na základě portu. Metoda používá vícestupňový klasifikátor, který sleduje minimální délku RTP hlavičky, číslo verze RTP a typ dat nesených v RTP.[3]

- **SPID (Statistical Protocol Identification)**
  S touto metodou jako první přišel Erik Hjemvik. Zakládá se na metodě učení s učitelem, jedná se o jednu z metod strojového učení. Tato metoda strojového učení potřebuje trénovací sadu, v níž bude ke konkrétním vzorkům provozu přiřazena správná klasifikace. Poté si algoritmus vytvoří databázi modelů protokolů, do které si uloží otisky aplikačních protokolů.[3]

2.2.3 Zpracování vstupního souboru

Nyní bude vysvětleno, jak nástroj Netfox Detective funguje jako celek při získávání forenzních důkazů pro orgány činné v trestním řízení. Prvně uživatel pomocí grafického uživatelského rozhraní vloží soubor se zachycenou komunikací. Formát souboru by měl být takový, který podporuje knihovna PmLib, jak bylo uvedeno výše. Po vložení souboru se zavolá knihovna PmLib, jenž z rámců uložených ve vstupním souboru vytvoří kolekci.

Následně se pomocí ConversationTrackeru asynchronně zpracují všechny typy konverzací pro daný rámec z kolekce. Výsledky jednotlivých typů konverzací se uloží do SQL databáze pomocí komponenty DbContext. Typy konverzací mohou být například L3, L4 a L7, kde čísla odpovídají číslům vrstev ISO/OSI síťového modelu. L7ConversationTraker vytváří konverzace, které probíhají na aplikační vrstvě nad transportními protokoly TCP nebo UDP a vytváří objekty L7PDU, které reprezentují zprávy aplikačního protokolu. Objekty L7PDU jsou tvořeny bez jakékoliv znalosti daného aplikačního protokolu. Při vytváření konverzací a rekonstrukci se využívají čísla portů a segmentová čísla v protokolu TCP pro odhalení chybějících dat či neukončeného TCP spojení. Proces zpracování vloženého a vstupního souboru se zachycenými daty můžeme sledovat na obrázku 2.5.



Obrázek 2.5: Zpracování vstupního souboru na konverzace.[11]

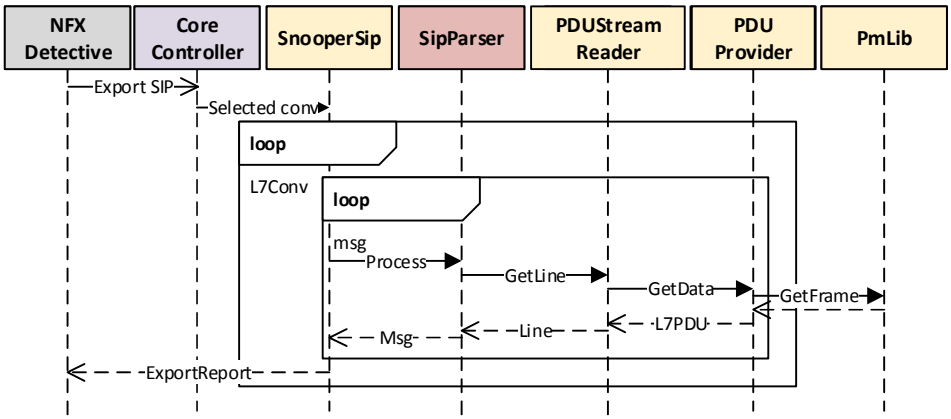
Krok, který nyní následuje, je pro úspěšnou forenzní analýzu stěžejní. V této fázi zpracování se musí určit, jakým aplikačním protokolem se v L7 konverzaci komunikuje. O identifikaci aplikačního protokolu, jak už bylo výše řečeno, se stará ApplicationRecognizer. Identifikace aplikačního protokolu je zásadní z toho důvodu, že na základě identifikace se vybere modul, který z příslušných zpráv aplikačního protokolu reprezentovaných L7PDU, extrahuje informace důležité pro forenzní analýzu. Když bychom tedy určili aplikační protokol špatně, tak bychom v lepším případě nedostali žádné informace. V tom horším případě bychom mohli extrahovat informace, které neodpovídají realitě.

2.2.4 Snooper

Modul zajišťující extrahování informací pro forenzní analýzu se nazývá Snooper. Samozřejmě Netfox Detective obsahuje Snooper modulů více, protože každý Snooper zpracovává určitý aplikační protokol. Snooper moduly si Netfox Detective načítá dynamicky, z čehož plyne, že když budeme chtít přidat nový Snooper, tak nemusíme kompilovat celou aplikaci znova, ale postačuje zkompilovat Snooper a jeho binární soubory nakopírovat do složky s binárními soubory Netfox Detective. Výstup jednoho modulu Snooper může být napojený na vstup druhého modulu za účelem další rekonstrukce provozu.

Uveďme si příklad, kdy se výstup zpracování protokolu HTTP předá na vstup modulu, který provádí rekonstrukci web mailu. Snooper po dokončení analýzy exportuje data obsažená v konverzacích spolu s metadaty získanými při analýze aplikačního protokolu v daném vyšetřování.

Každý Snooper obsahuje vlastní modely, view-modely a pohledy pro detailní zobrazení dat získaných během rekonstrukce. Například HTTP snooper zobrazuje rekonstruovaný web, email snooper vypíše rekonstruované emaily, VOIP snooper sestavování hovoru s RTP streamy pro možné přehrání atd. Na obrázku 2.6 si můžeme všimnout příkladu znázorňujícího extrahování dat z protokolu SIP. Prvně přijde zpráva řídící komponentě od uživatelského rozhraní, aby extrahovala data pro forenzní účely z protokolu SIP. Řídící komponenta spustí snooper učený pro SIP protokol a předá mu konverzace, jenž byly identifikovány, že v nich probíhá komunikace tímto protokolem. Snooper pak projde všechny přiřazené L7 konverzace, ze kterých si extrahují SIP zprávy. Po extrahování SIP zpráv se vytvoří ExportReport, kde se uvedou kupříkladu čísla volajícího a volaného, délka ukončeného hovoru, stav hovoru, zda se navazuje spojení hovoru, hovor probíhá či hovor byl již ukončen atd.[11]



Obrázek 2.6: Příklad extrahovaní forenzních informací z protokolu SIP.[11]

Kapitola 3

Návrh implementace webového rozhraní

Návrh implementace webového rozhraní se bude dělit do několika částí. V první části budou konkretizovány požadavky, které budou pro webové rozhraní požadovány. V rámci požadavků bude vymezeno, co by měla umožňovat správa uživatelů a specifikace uživatelských rolí. Dále bude potřeba upřesnit, co by mělo webové rozhraní umožnit ve správě vyšetřování. Do poslední části bude zahrnut návrh rozmístění prvků uživatelského rozhraní některých částí webového rozhraní.

Uživatelské role

Webové rozhraní by mělo obsahovat dvě uživatelské role. Roli administrátor a roli vyšetřovatel. Uživatel s rolí vyšetřovatel by měl mít přístup pouze do vyšetřování, která vlastní nebo do nichž dostal přístup od vlastníka nějakého vyšetřování. Vlastníkem vyšetřování je vždy uživatel, který vyšetřování založí. Dále by vyšetřovatel neměl mít možnost upravovat informace o vyšetřování (název, popis vyšetřování) a zároveň by neměl mít možnost smazat vyšetřování, pokud není jeho vlastníkem. Uživatel s rolí administrátora by pak měl plný přístup do všech vyšetřování a měl by mít možnost měnit informace o vyšetřování a smazat vyšetřování bez nutnosti mít oprávnění přistupovat do vyšetřování nebo bez nutnosti být vlastníkem vyšetřování.

Požadavky na správu uživatelů

Ve správě uživatelů bychom kromě vytváření a mazání uživatele měli být schopni nastavit, zda daný uživatelský účet je aktivní či nikoli, to znamená, když uživatelský účet nebude aktivní, tak se uživatel s tímto účtem nebude moci přihlásit. Vytváření a mazání uživatelských účtů bude moci provádět pouze administrátor. Dále bude dostupná stránka s informacemi o profilu uživatele, na níž bude mít přístup jen konkretní uživatel. Zde nalezne informace o svém účtu, například jméno vlastníka uživatelského účtu, přiřazenou roli, atd. Dále si uživatel bude moci na této stránce změnit heslo.

Požadavky na správu vyšetřování

Správa vyšetřování by měla umožnit vytváření a mazání vyšetřování, kde práva uživatelů na vytváření a mazání vyšetřování jsou popsána výše. Vlastník vyšetřování může:

- editovat informace o vyšetřování,
- přidávat vyšetřovatele do vyšetřování,
- odebírat vyšetřovatele z vyšetřování.

Vlastník vyšetřování může být změněn, a to buď administrátorem nebo samotným vlastníkem vyšetřování.

Požadavky na použití technologií

Mezi technickými požadavky je použití programovacího jazyka C#. Tento požadavek vychází z toho, že Netfox Framework je napsaný právě v tomto programovacím jazyce. Dalším požadavkem je použití návrhového vzoru Dependency Injection. Použití tohoto návrhového vzoru se poté využije pro lepší testování jednotlivých částí webového rozhraní.

3.1 Návrh rozmístění komponent webového rozhraní

Poslední částí kapitoly bude popis návrhu rozložení uživatelských prvků na vybraných stránkách webové aplikace. Návrhy zobrazení jsou vytvořeny pomocí wireframů, což se dá chápat jako takový náčrt rozmístění prvků na webové stránce. Wireframe se dělá vždy při návrhu webových prezentací. Poté co zákazník návrh odsouhlasí, se podle wireframu udělá grafický návrh, kterému následně kodér vytvoří šablonu.

Úvodní strana

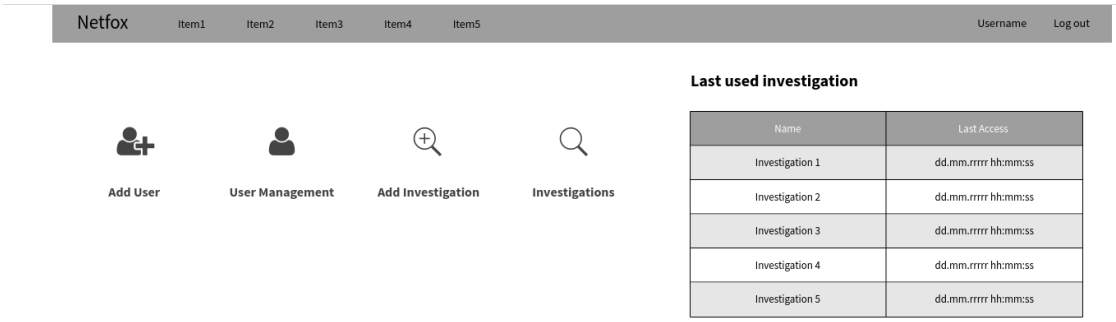
První wireframe, který si vysvětlíme, je znázorněn na obrázku 3.1. Jedná se o návrh úvodní stránky po přihlášení. V horní části si můžeme všimnout šedého obdélníku, který reprezentuje lištu pro menu. Menu se bude zobrazovat na všech stránkách webového rozhraní kromě stránky s formulářem pro přihlášení. Když začneme z levé strany lišty, uvidíme text „Netfox“, jenž je odkazem na úvodní stranu. Za textem následují položky menu. V pravé části se potom nachází tlačítko pro odhlášení a uživatelské jméno jako odkaz na stránku s profilem uživatele. V obsahové části budou ikony pro rychlé přejítí, například na stránku s nastavením, na stránku se správou uživatelů, správu vyšetřování atd. Vedle ikon by se zobrazovala poslední otevřená vyšetřování, jejichž počet by nepřesáhl pět. Při kliknutí na vyšetřování se pak zobrazí pohled daného vyšetřování.

Přihlašovací stránka

Nyní popisované wireframy jsou k nalezení v přílohách. Dále si charakterizujeme stránku s formulářem pro přihlášení, viz obrázek A.1. Ta obsahuje:

- logo,
- pole pro uživatelské jméno,

- pole pro heslo,
- tlačítko pro odeslání formuláře s přístupovými údaji.



Obrázek 3.1: Wireframe pro úvodní stránku po přihlášení.

Po stisknutí tlačítka pro odeslání přístupových údajů se přístupové údaje odešlou na webový server. Přístupové údaje jsou na webovém serveru ověřeny. V případě, že dojde při ověřování přístupových údajů k chybě, zobrazí se příslušná chybová hláška nad poli pro vyplnění přístupových údajů.

Výpis dostupných vyšetřování

Wireframe pro pohled zobrazení výpisu vyšetřování zachycuje obrázek A.2. V levé části pod horním menu se nachází název sekce. Pod ní by měla být zobrazena vyšetřování, jenž vyšetřovatel otevřel naposledy. Počet posledně navštívených vyšetřování bude pět. V obsahové části jsou vypsána dostupná vyšetřování. Dostupnými vyšetřováními je myšleno vyšetřování, které vyšetřovatel vlastní či k nim má povolený přístup od vlastníka, jak je zmíněno výše ve specifikaci požadavků.

Počet vyšetřování na jednu stánku by měl být patnáct. Stránkováním se bude řešit situace, že vyšetřovatel bude mít více něž oněch patnáct vyšetřování na stránku. Tedy pod tabulkou s výpisem vyšetřování se zobrazí komponenta umožňující nám přecházet mezi jednotlivými stránkami. Nad tabulkou s výpisem vyšetřování jsou vidět dvě tlačítka, jedno bude sloužit k přidání nového vyšetřování a druhé na odebrání vybraných vyšetřování. Dále si můžeme ve wireframu všimnout u jednotlivých vyšetřování dvou ikonek a to tužky a koše. Při kliknutí na tužku se zobrazí pohled pro editaci daného vyšetřování a při kliknutí na koš se dané vyšetřování smaže. Když uživatel klikne na název vyšetřování, dostane se na detailní pohled konkrétního vyšetřování.

Vytváření a editace vyšetřování

Další wireframe se týká pohledu pro vytváření a editaci vyšetřování, viz obrázek A.3. Tento pohled se liší pouze obsahovou částí. V obsahové části budou dvě záložky. V jedné bude možné vyplnit název vyšetřování, popis vyšetřování a vlastníka vyšetřování. V druhé záložce pak bude vyšetřovatel vybírat ostatní vyšetřovatele, kteří budou mít k vyšetřování přístup. Výběr bude probíhat tak, že se vypíše tabulka s vyšetřovateli a u každého vyšetřovatele

bude checkbox. Po jeho zaškrtnutí vyšetřovatel vybere uživatele, který dostane přístup k vyšetřování. Nad záložkami se nacházejí dvě tlačítka:

- Tlačítko *Save* pro vytvoření či uložení změn vyšetřování.
- Tlačítko *Cancel* umožní zrušení změn a vrácení zpět na výpis vyšetřování.

Detail vstupního souboru

Dalším zmíněným pohledem je znázorněn wireframe na obrázku A.4. Jedná se o pohled zobrazující detailní informace o souboru se zachycenou komunikací. Nalevo můžeme vidět dvě sbalovací menu:

1. Menu sloužící pro soubory se zachycenou komunikací nahrané vyšetřovatelem do vyšetřování.
2. Menu s dostupnými aplikačními protokoly, u nichž si může vyšetřovatel prohlédnout exporty. Při kliknutí na položku v menu se vyšetřovatel dostane do výpisu exportů patřící danému protokolu.

V obsahové části pak jsou záložky. Jednotlivé záložky pak obsahují výpis statistik, například kolik je v komunikaci konverzací daného typu, kolik rámců prošlo daným směrem, kolik rámců obsahuje soubor atd. V dalších záložkách se pak zobrazují tabulky z L3, L4 a L7 konverzacemi, které se vyskytují v souboru. Dle typu konverzace můžeme u záznamu konverzace nalézt informace o:

- času počátku a konce konverzace,
- zdrojové adrese a portu,
- cílové adrese a portu,
- transportním protokolu,
- počtu rámců a bajtů odeslaných od klienta,
- počtu rámců a bajtů přijatých klientem,
- počtu poškozených rámců,
- počtu extrahovaných bajtů,
- počtu chybějících rámců a bajtů,
- popisu nesených dat.

A opět je zde omezení na počet zobrazených konverzací na stránku. Na stránce je možné zobrazit patnáct konverzací. Když je počet překročen, objeví se komponenta pro přechod mezi stránkami. V poslední záložce se pak budou zobrazovat rámce v daném souboru.

Exporty forenzních informací

Wireframe vyobrazuje výpis exportů z daného protokolu, viz obrázek A.5. Ve wireframu si můžeme všimnout záložek pro:

- exporty,
- chatové zprávy,
- soubory,
- obrázky,
- hovory.

V záložce pro exporty bude tabulka se záznamy exportů. U záznamů budou uvedeny informace, kupříkladu jakého časového rozmezí se exporty týkají, typ exportovaného objektu. Po kliknutí na některý záznam exportu se přejde na stránku s detailním pohledem na export.

Tím se dostáváme k definici dalšího pohledu pro zobrazení informací z exportů. Wireframe s tímto pohledem nalezneme na obrázku A.6. V tomto pohledu bude minimálně jedna záložka. V záložce se na levé straně nachází seznam exportovaní objektů. Tento seznam bude sloužit pro volbu objektu. V pravé části záložky se pak zobrazí veškeré vlastnosti obsažené ve zvoleném exportovaném objektu. Další záložky pak závisí na konkretním aplikačním protokolu.

Kapitola 4

Implementace

Kapitola o implementaci se bude nejprve věnovat popisu důležitých frameworků a knihoven, které velmi zjednodušily samotnou implementaci webového rozhraní. Následně se budeme v kapitole zabývat, jak je implementováno přihlašování uživatelů a správa uživatelů. Další popsanou částí implementace bude propojení webového rozhraní a Netfox Framework.

4.1 Použité frameworky a knihovny

Mezi použité frameworky patří hlavně DotVVM, u které bude zmíněno k čemu slouží a způsob generování požadovaných stránek. Dalšími použitými frameworky jsou Bootstrap usnadňující vývoj webových stránek předefinovanými styly a Chart.js pro zobrazení grafů. Poslední zmíněnou knihovnou je Hangfire. Ta slouží pro spouštění úloh na pozadí webového serveru.

4.1.1 Dotvvm

Pro vývoj aplikací pod platformou .NET může programátor využít oficiálních řešení jako jsou ASP.NET MVC nebo ASP.NET Web API v kombinaci s nějakým javascriptovým frameworkem jako React, Angular nebo Knockout. Toto řešení není ale úplně pohodlné při vývoji větších aplikací, protože u větších aplikací většinou budeme potřebovat další prvky na stránce, například nějaké posuvníky či komponentu pro zadávání kalendářního data. V tom případě sáhneme po další javascriptové knihovně řešící tento problém. Při vkládání více knihoven může dojít k velmi nepříjemné věci a to k tomu, že javascriptové knihovny mohou být v konfliktu. To znamená, že se budou navzájem ovlivňovat a bude docházet k neočekávanému chování jednotlivých komponent uživatelského rozhraní. Za tím účelem je dobré použít nějaký framework poskytující všechny potřebné komponenty.

V rámci tvorby webového rozhraní pro Netfox Detective byl zvolen framework DotVVM, kde Dot znamená .NET Framework a VVM znamená MVVM návrhový vzor. DotVVM používá návrhový vzor MVVM popsaný v první kapitole. Výhodou tohoto frameworku je, že nevyžaduje po programátorovi znalost jakékoliv javascriptové knihovny, aby mohl poskytované komponenty používat. Postačuje znalost C#, HTML a CSS. Práce s frameworkem je velmi jednoduchá. Pohled stránky se definuje za pomoci syntaxe značkovacího jazyka HTML a přidaných značek pro DoVVM, proto se tento jazyk nazývá DOTHTML. Pro použití komponent z DotVVM se dále používá následující značka `dot:název komponenty`. Data z viewmodelu se pak následně získávají pomocí data bidingu. Důležité je také přidat na začátek každého pohledu anotaci říkající, která třída je viewmodelem pro daný pohled.

Viewmodel je pak standardní C# třídou. Ve třídě jsou pak pouze definovány vlastnosti a metody, například pro obsluhu události kliknutí na nějaké tlačítko.[5]

Obrázek 4.1: Schéma zpracování požadavku v DotVVM.[14]

U Dotvvm stojí rozhodně za zmínku proces, jak se zpracovává úvodní požadavek při prvním načtení stránky (HTTP GET požadavku) a jak probíhá zpracování provedení příkazu na stránce (AJAX HTTP POST), například při kliknutí na tlačítko, čímž zavoláme metodu ve viewmodelu. V levé části následujícího obrázku 4.1 jsou znázorněny jednotlivé fáze procesu zpracování odpovědi na úvodní požadavek(HTTP GET) a v pravé části obrázku pak fáze procesu zpracování příkazu na stránce (AJAX HTTP POST). Po přijetí požadavku na první načtení stránky se nejprve vytvoří třída viewmodelu požadované stránky. Poté se provedou metody Init(), Load() a PreRender(). Po provedení těchto metod se provede renderování HTML stránky. Před odesláním vyrenderované HTML stránky se musí serializovat ještě viewmodel patřící ke stránce. Následně se HTML a serializovaný viewmodel pošle klientovi.

Když se zaměříme na zpracování provedení akce na stránce, na obrázku můžeme vidět pár rozdílů. V prvním řadě, po vytvoření viewmodelu a jeho inicializaci metodou Init(), proběhne deserializace hodnot viemodelu zaslaných od klienta a přiřazení hodnotám právě vytvořeného viewmodelu. Po provedení metody Load() se vykoná požadovaný příkaz, například zavolání metody ve viewmodelu. Po provedení příkazu se zavolá metoda PreRender(), ale po jejím vykonání se již neprovádí renderování HTML, ale aktuální stav viewmodelu se serializuje a odešle se klientovi.[14]

4.1.2 Bootstrap, HangFire a Chart.js

Bootstrap je open source framework pro vývoj webových stránek a aplikací využívající HTML, CSS a Javascript. Framework obsahuje předpřipravené komponenty. Jedná se například o navigaci, formulářové prvky, ikony, stránkování, drobečková navigace atd. Veškeré komponenty ve frameworku se chovají responzivně, což zajišťuje, že výsledná webová stránka se korektně zobrazí ve všech možných velikostech rozlišení displeje.[1]

Další knihovnou, která bude využita ve webovém rozhraní je HangFire. Jde o knihovnu umožňující spouštět výpočetní úlohy na pozadí webového serveru[12, 8]. Tuto funkcionalitu využijeme pro zpracování vstupních souborů se zachycenou komunikací. Důvodem je, že není žádoucí, aby při odeslání požadavku na extrahování forenzních dat z komunikace, kupříkladu kliknutím na tlačítko, webový prohlížeč čekal, až se na webovém serveru extrahování provede a webový server na HTTP požadavek zašle odpověď webovému prohlížeči.

Poslední důležitou knihovnou, která se využívá v implementaci, je open source javascriptová knihovna Chart.js umožňující jednoduše vykreslit spoustu grafů[1]. K tomu, aby knihovna fungovala správně potřebuje, aby webový prohlížeč podporoval HTML 5, což v dnešní době podporují všechny prohlížeče. Knihovna podporuje responzivní zobrazení a je testována pro většinu prohlížečů používaných v současnosti.

4.2 Struktura projektu Netfox.Web

Ještě než se začneme zabývat samotnou implementací jednotlivých komponent webového rozhraní, tak bude popsána struktura projektu Netfox.Web:

- View - složka obsahuje pohledy pro jednotlivé stránky,
- ViewModel - obsahuje třídy viewmodelů pro jednotlivé pohledy,
- Interface - rozhraní pro viewmodely, třídy poskytující informace z databáze atd.,
- Infrastructure - nachází se zde instalátor kontejneru pro Dependency injection, třídy implementované podle návrhového vzoru Factory atd.,
- Service - zde můžeme nalézt třídy, pomocí kterých se provádí propojení s Netfox.Framework, správa uživatelů, správa vyšetřování atd.,
- Entity - obsahuje třídy reprezentující entity v databázi,
- DTO - obsahují třídy zastupující roli DTO (Data Transfer Object),
- Mapping - zde se definuje, jak se jednotlivé Entity namapují na DTO a naopak, jak se DTO převedou na Entity,

[1] http://www.chartjs.org/samples/latest/

- Template - obsahuje soubory týkající se stylů stránky a skripty v jazyku Javascript,
- Persistence - třídy definující databázový kontext.

Než se v kapitole začneme zabývat implementací řešení jednotlivých částí webového rozhraní, bude dobré zmínit, jak se například přistupuje k datům v databázi nebo co je třeba upravit, aby se v rámci webové aplikace mohl použít návrhový vzor *Dependency Injection*. Pro implementaci návrhového vzoru *Dependency Injection* se ve webovém rozhraní využilo knihovny *Castle.Windsor*, jelikož se používá i v Netfox Frameworku. Kontejner pro registrování komponent se inicializuje v konfiguračním souboru pro webové rozhraní, tedy ve *Statup.cs*. Po vytvoření instance kontejneru se zavolá instalátor, který zaregistruje jednotlivé komponenty webového rozhraní. Registrují se především viewmodely, třídy poskytující služby, kupříkladu manipulaci s daty v databázi atd. Aby kontejner mohl být použit v rámci DotVVM, bylo potřeba přidat do konfigurace DotVVM vlastní třídu implementující rozhraní `IViewModelLoader` z důvodu, aby se při načítání stránky použil pro získávání viewmodelu a všech jeho potřebných závislostí kontejner vytvořený na začátku konfigurace webového rozhraní.

Pro přístup k datům a jejich úpravám se používá návrhový vzor *Unit Of Work*. To umožňuje aplikaci si zaznamenávat požadované změny databáze v určeném databázovém kontextu.

Pro manipulaci s jednotlivými entitami se ve webovém rozhraní používá návrhový vzor `Repository`. Návrhový vzor umožňuje odstínit přístup k úložišti perzistentních dat.[6]

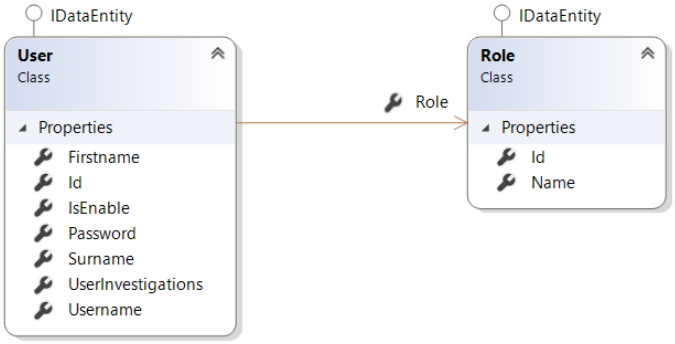
4.3 Autentizace uživatelů

V podkapitole bude popsáno, jak je implementováno přihlašování uživatelů. Tato část nebyla řešena na desktopové variantě, protože nebylo potřeba. Uživatelé měli totiž aplikaci nainstalovanou lokálně a ukládat svá vyšetřování mohli také lokálně. Autentizaci uživatelů bude částečně zajišťovat DotVVM. Ověření a ukládání údajů o uživateli do databáze bude nutné si navrhnout pro naše potřeby vlastní. Abychom mohli použít autentizaci za pomoci DotVVM, musí se nejprve naimportovat NuGet balíček `Microsoft.OWiN.Security.Cookies`. Poté je nutné v souboru *Startup.cs* OWIN Cookies zaregistrovat.[13]

Pro autentizaci se v databázi budou muset vytvořit dvě tabulky, jednu pro uživatele a druhou pro uživatelské role. Každý uživatel bude moci mít pouze jednu roli. V systému jsou zatím dvě role, a to administrátor a vyšetřovatel. Co můžou jednotlivé role provádět, je zmíněno výše ve specifikaci požadavků.

Nyní si popíšeme vlastnosti entit `User` a `Role`. Vztah obou entit je vyobrazen na obrázku 4.2. Z názvu obou entit je zřejmý jejich význam. Entita `User` představuje uživatele v systému a entita `Role` reprezentuje role v systému. Začneme entitou `User`. Jak již víme každá entita musí mít implementováno rozhraní `IDataEntity`, z toho vyplývá, že entita `User` obsahuje vlastnost Id, která reprezentuje primární klíč. Dalšími vlastnostmi, které entita obsahuje jsou `Firstname` a `Surname`, ty představují jméno a příjmení uživatele. Z vlastnosti `Username` jsme schopni se dozvědět uživatelské jméno uživatele. V `Password` nalezneme heslo, respektive hash hesla. Hašovací funkce používaná v rámci webového rozhraní, je SHA-256, takže očekáváme, že hash hesla má délku 64 znaků. Vlastnost `IsEnabled` bude reprezentovat stav, jestli se daný uživatel bude moci přihlásit pomocí svých přihlašovacích údajů či ne. To znamená, že pokud bude potřeba znemožnit přístup uživateli do systému, tak bude stačit účet deaktivovat místo toho, aby se musel mazat. Poslední vlastností, u které zbývá vysvětlit její význam, je vlastnost `Role`. Účelem této vlastnosti je určit jakou roli má daný uživatel

přiřazenou. Entita `Role` má pouze dvě vlastnosti, a to `Id` a `Name`. Jak je již z názvů jasné, tak `Id` bude sloužit jako jednoznačný identifikátor role a `Name` bude název role.



Obrázek 4.2: Znázornění závislosti entit pro uživatele a role uživatelů diagramem.

Dále se v podkapitole budeme zabývat implementací přihlášení do webového rozhraní. Pohled pro přihlášení je definovaný v šabloně `login.dohtml` a viewmodel pro pohled je třída `LoginViewModel`. Třída `LoginViewModel` obsahuje vlastnosti:

- `Username` se používá pro přístup k obsahu pole pro uživatelské jméno.
- `Password` pro uložení k obsahu pole s heslem.
- `ErrorMessage` slouží pro uložení chybové hlášky při ověřování uživatelského jména a hesla. Když není nastavena, znamená to, že nedošlo k žádné chybě.
- `UserService` umožňuje přístup k třídě poskytující služby týkající se uživatelů.

U vlastností `Username` a `Password` je anotace `[Required(ErrorMessage = "Username is required.")]` určující, že tyto dvě vlastnosti budou validovány a definuje chybovou hlášku pro případ, že validace nebude úspěšná. Uvedený příklad anotace se vztahuje k poli pro uživatelské jméno.

Prvním krokem, který se při pokusu o přihlášení provede, je validace vstupu, v tomto případě kontrola, zda jsou vyplněna všechna vstupní pole, tedy je-li vyplněno uživatelské jméno a také heslo. Pokud se vypíše chybová hláška, není nějaké z polí vyplněno. Po vyplnění obou polí se pokusí verifikovat přihlašovací údaje. Verifikace se provede metodou `VerifyCredetials()` instance třídy `UserService`, které předáme uživatelské jméno a hash hesla. Hash hesla získáme zavoláním metody `HashPassword()` ze stejné třídy, metoda zahashešuje vstupní řetězec funkcí SHA-256. Pokud metoda `VerifyCredetials()` nevrátí id uživatele, tak to znamená, že vyplněné uživatelské jméno a heslo nesouhlasí. V tom případě se do vlastnosti zapíše chybová hláška, že bylo zadáno špatné uživatelské jméno nebo heslo a proces přihlašování skončí. Naopak když z metody `VerifyCredetials()` dostaneme id uživatele, tak verifikace proběhla úspěšně a můžeme pokračovat v procesu přihlašování.

Ze získaného id uživatele metodou `GetUser()` se získá instance třídy `UserDTO`. Zde nás může napadnout proč metoda vrátila objekt UserDTO a ne přímo objekt User, který přece reprezentuje entitu pro uživatele. Je to z toho důvodu, že DotVVM serializuje veškeré uložené vlastnosti viewmodelu, takže i hash hesla uživatele, který obsahuje právě třída User. Z toho důvodu se ve viewmodelu nepracuje přímo s entitou, ale s DTO. Třída `UserDTO`

má jediný rozdíl oproti `User` a to ten, že neobsahuje vlastnost `Password` s hashem hesla uživatele. O mapování těchto dvou tříd se pak stará nástroj AutoMapper.

Následně z vlastnosti `IsEnable` z instance třídy `UserDTO` zjistíme, zda se uživatel může přihlásit pod tímto účtem. Nemá-li to povoleno, vypíše se chybová hláška, že účet není aktivní. V opačném případě se pokračuje dál v procesu přihlašování.

Nyní jak je popsáno v dokumentaci DotVVM, bude potřeba vytvořit objekt `ClaimIdentity`, který uchovává údaje o uživateli. V našem případě to bude uživatelské jméno a role, kterou má uživatel přiřazenou. Po vytvoření objektu reprezentující identity uživatele je třeba vytvořit autentizační token a uložit ho do cookies. V DotVVM se toto dělá přes metodu `Context.GetAuthentication().SignIn()`. Vstupním parametrem metody je právě objekt, který reprezentuje identitu uživatele, jenž se vytvořil v předchozím kroku.

Posledním krokem přihlášení, který je při úspěšném přihlášení nutné udělat, je nastavit přesměrování na stránku, která se zobrazí po přihlášení. To se v DotVVM dá zařídit metodou `Context.RedirectToRoute()`. Jako vstupní parametr se uvede název cesty v routovací tabulce, jenž je zapsaná v metodě `ConfigureRoutes` třídy `DotvvmStartup`. Přihlášený uživatel se bude moci po ukončení práce ze systému odhlásit kliknutím na tlačítko s ikonou znázorňující odhlášení. Po kliknutí na tlačítko bude zavolána metoda, která provede odhlášení uživatele. V metodě se bude muset zneplatnit cookie s autorizačním tokenem, které se do cookies uložily při přihlášení. To se provede metodou `Context.GetAuthentication().SignOut()`. Po zneplatnění cookie je nutné uživatele přesměrovat na stránku s přihlašovacím formulářem.

4.4 Správa uživatelů

Správa uživatelů je implementována dvěma pohledy. Jedná se o pohled pro výpis všech uživatelů, ze kterého lze odebírat uživatele, přecházet na druhý pohled, ze kterého se dají upravit údaje stávajícího uživatele či přidat nového uživatele.

Pohled, sloužící pro výpis přehledu uživatelů, se vypisuje podle šablony `UserManagement.dothtml` a funkci viewmodelu zajišťuje třída `UserManagementViewModel`. Než bude popsáno, jak se zajišťují veškeré funkce, které viewmodel obstarává, bylo by dobré zmínit nejprve jeho strukturu. Vlastnosti viewmodelu `UserManagementViewModel` jsou následující:

- Users slouží k uchovávání záznamů pro výpis uživatelů.
- UserIDs uchovává seznam id uživatelů, kteří jsou vybráni.
- IsAllSelected je vlastnost pro checkbox, jestli se mají vybrat všichni uživatelé nebo se má výběr všech uživatelů zrušit.
- Service umožňuje přístup k třídě poskytující služby týkající se uživatelů.

Skutečnost, že se na stránky se správou uživatelů měli dostat pouze administrátoři, lze dosáhnout v DotVVM anotací `[Authorize(Roles = new[] {"Administrator"})]`. Parametr se může dát buď před definici třídy viewmodelu dané stránky nebo před definici konkrétní metody. Když parametr bude aplikován na třídu viewmodelu, tak uživatel k tomu, aby mohl přistoupit na stránku, bude potřebovat danou roli uvedenou v definici. Pokud bude parametr pouze před definicí metody, role bude potřeba jen k provedení metody, která provede nějakou akci, například smazání nějakého záznamu z databáze.

Načítání uživatelů, jenž se mají zobrazit ve výpisu, se děje v metodě `PreRender()`. Kdy se volá metoda `PreRender()` je popsáno viz podkapitola 4.1.1. V metodě se nejprve

nastaví parametry zobrazení výpisu. Mezi parametry zobrazení patří počet zobrazených záznamů na stránku, v tomto případě patnáct záznamů na stránku. Pokud počet záznamů překročí patnáct, zobrazí se pod výpisem komponenta na přechod mezi jednotlivými stránkami se záznamy. Dalším parametrem je způsob řazení, zde je nastaveno sestupné řazení. Posledním parametrem, který se nastavuje, je, podle jaké vlastnosti se bude výpis řadit. V tomto výpisu se řadí pomocí vlastnosti `Username`, tedy podle uživatelského jména.

Po nastavení parametrů zobrazení se metodou `FillUsersList()` z třídy `UserService` získá seznam uživatelů. Metodě se předá odkaz na vlastnost `Users`. Metoda následně získá z databáze záznamy o uživatelích, kterými naplní předaný objekt odkazem v parametru metody. Uživatelé se dají mazat dvěma způsoby a to buď jeden konkrétní nebo více uživatelů naráz. Pokud chceme mazat jednoho konkrétního uživatele, lze kliknout na ikonu koše u konkrétního uživatele. Jestli chceme vymazat více uživatelů, musíme je nejprve vybrat zaškrtnutím checkboxu u konkrétních uživatelů, které chceme vybrat. Po vybrání uživatelů na odebrání, stačí kliknout na tlačítko *Remove* nad výpisem uživatelů.

Popsané způsoby na straně aplikace probíhají následovně. Při odebrání jednoho uživatele, tedy kliknutím na ikonu koše u konkrétního uživatele, se zavolá metoda `RemoveUser()`, jejímž parametrem je id uživatele, kterého chceme odebrat. V metodě se zavolá opět `RemoveUser()` nacházející se ve třídě `UserService` a jejímž parametrem je také id uživatele. Provedením metody se z databáze odebere uživatel.

Při odebírání více uživatelů se id vybraných uživatelů pro smazání ukládá do seznamu `UserIDs`. Po kliknutí na tlačítko *Remove* se zavolá metoda `RemoveSelectedUsers()`. V metodě se projde postupně seznam s id vybraných uživatelů a pro každé id se zavolá metoda `RemoveUser()`.

Pokud chceme přidat uživatele, klikneme na tlačítko *New User*. Kliknutí nás přesměruje na druhý pohled zajišťující správu uživatelů. To samé se stane, když chceme informace o uživateli editovat a klikneme na ikonu tužky u konkrétního uživatele. Rozdílem bude jen to, že nyní budeme zasílat id uživatele, které chceme editovat.

Druhý pohled je definovaný šablonou `user.dothtml` a viewmodel třídou `UserViewModel`. Viewmodel obsahuje tyto vlastnosti:

- UserId obsahuje id uživatele, kterého chceme editovat.
- Password je vlastnost pro pole, do kterého se zadává heslo.
- Data jedná se o objekt třídy UserDTO. Jsou v ní uloženy všechny údaje o uživateli, kterého právě editujeme nebo vytváříme.
- IsNewUser slouží k indikaci, zda se jedná o vytváření nového uživatele nebo jde o editaci již existujícího.
- Roles je seznam použitelných rolí pro uživatele.
- SelectedRoleID reprezentuje id přiřazené role uživateli.

Při načítání pohledu se při zavolání metody `PreRender()` prování nastavení vlastností `Data` a `SelectedRoleID`. Nastavení probíhá pouze pokud se jedná o první načtení stránky. Předtím, než se dostaneme k nastavení, ověří se, jestli se jedná o vytváření nového uživatele nebo o editování existujícího uživatele. Pokud není předaným parametrem id uživatele, tak se jedná o vytváření nového uživatele a vlastnosti `Data` se tím pádem nastaví nová instance třídy `UserDTO` a vlastnosti `SelectedRoleID` se nastaví id první role ze seznamu `Roles`. Naopak je-li parametrem předané id uživatele, tak se vlastnosti `Data` nastaví objekt

`UserDTO`. Objekt vrátí metoda `GetUser()` třídy `UserService`, které předáme id uživatele. U vlastnosti `SelectedRoleID` se nastaví hodnota id role v objektu uloženého v `Data`.

Z pohledu pro vytváření nového uživatele nebo z editace existujícího uživatele se dá odejít bez vytvoření nového uživatele nebo provedení změn u uživatele kliknutím na tlačítko *Cancel*. Kliknutí na tlačítko způsobí přesměrování na pohled s výpisem uživatelů. Když se klikne na tlačítko *Save* dojde nejprve k validaci požadovaných hodnot vlastností, jestli jsou vyplněny. Pokud validace proběhne úspěšně, zavolá se metoda `Save()` viewmodelu. V metodě se zjistí opět skutečnost, jedná-li se o přidávání či úpravu uživatele. Jedná-li se o přidání uživatele, zavolá se metoda `AddUser()`, které se předá objekt `UserDTO` s vyplněnými daty o uživateli, druhým předávaným parametrem je id vybrané role a třetím parametrem je heslo nového uživatele. V druhém případě, jedná-li se o úpravu uživatele, tak se zavolá metoda `Save()` třídy `UserService`, které se předá jen objekt `UserDTO` s vyplněnými daty o uživateli a id vybrané role.

4.5 Profil uživatele

Profil uživatele zobrazuje informace o uživateli a umožňuje uživateli měnit své heslo k účtu. Profil uživatele je definován šablonou `profile.dohtml` a viewmodel obstarává třída `ProfileViewModel`. Struktura viewmodelu je následující:

- Password je vlastnost pro pole, do kterého se zadává současné heslo.
- NewPassword je vlastnost pro pole, do kterého se zadává nové heslo.
- IsPasswordDialogDisplayed indikuje stav, zda má být modální okno zobrazené.
- User jedná se objekt `UserDTO`, ve které jsou uloženy informace o uživateli.
- ErrorMessage slouží pro uložení chybové hlášky v případě chybně zadaného současného hesla.
- Service umožňuje přístup k třídě poskytující služby týkající se uživatelů.

Při načítání stránky se v metodě `Init()` získá objekt UserDTO k aktuálně přihlášenému uživateli a to metodou GetUser(), které se předá uživatelské jméno aktuálně přihlášeného uživatele. Získaný objekt se uloží do `User` ve viewmodelu.

Když uživatel chce provést již zmíněnou změnu hesla, provede to tak, že klikne na tlačítko *Change Password*. Kliknutím na tlačítko se změní u vlastnosti `IsPasswordDialogDisplayed` hodnota z *false* na *true*. To způsobí zobrazení modálního okna s formulářem obsahující pole pro současné heslo a pole pro nové heslo. Uživatel musí vyplnit obě pole, jinak při pokusu o změnu hesla dojde k chybě při validaci. Pokud validace vstupu projde v pořádku, tak se zavolá metoda `ChangePassword()`, ve které se zavolá metoda `ChangePassword()` ze třídy `UserService`. Metodě se předají následující parametry:

- id aktuálně přihlášeného uživatele,
- současné heslo,
- nové heslo.

Metoda následně zkontroluje, zda se současné heslo shoduje s heslem v databázi. V případě, že hesla souhlasí, tak se změna hesla provede a návratová hodnota bude *true*, jinak návratová hodnota metody bude *false*. Podle návratové hodnoty se v případě úspěchu hodnota vlastnosti `IsPasswordDialogDisplayed` změní zpět na *false*, čímž se modální okno skryje. V případě neúspěchu se zobrazí chybová hláška, že vyplněné současné heslo nesouhlasí s heslem v databázi.

Modální okno lze samozřejmě zavřít i bez toho, aby uživatel musel změnit heslo a to tak, že klikne na tlačítko *Cancel*. To způsobí zavolání metody `Cancel()` a skryje modální okno změnou hodnoty vlastnosti `IsPasswordDialogDisplayed` na *false*.

4.6 Správa vyšetřování

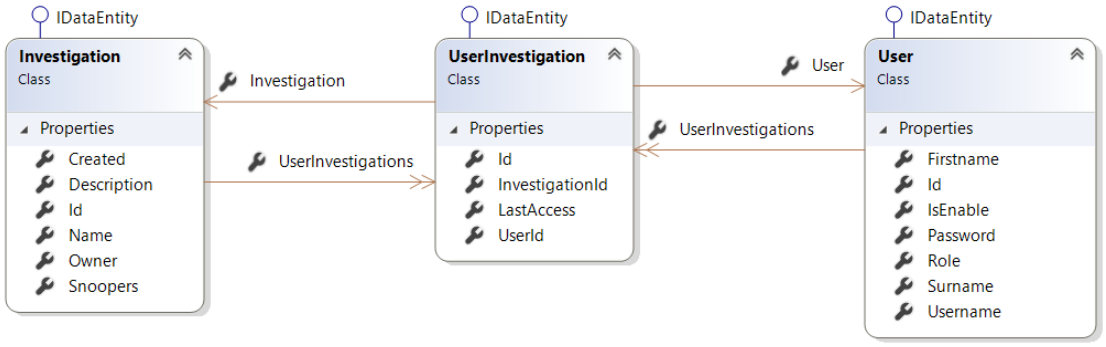
Správa vyšetřování je opět jako správa uživatelů implementována dvěma pohledy. Jedná o pohled sloužící pro výpis seznamu dostupných vyšetřování a o pohled pro vytváření či editaci vyšetřování. Pro výpis seznamu vyšetřování je pohled zapsaný v šabloně `overview.dot-html` a viewmodel je definován třídou `InvestigationOverviewViewModel`. Viewmodel se skládá z vlastností:

- Investigations slouží k uchovávání záznamů pro výpis vyšetřování.
- InvestigationIDs uchovává seznam id vyšetřování, která jsou vybrána.
- IsAllSelected je vlastnost pro checkbox, která indikuje, jestli se mají vybrat všechny vyšetřování nebo se má výběr zrušit.
- Service umožňuje přístup k třídě poskytující služby týkající se manipulace s vyšetřováními.

Výpis dostupných vyšetřování pro daného uživatele probíhá podobným způsobem, jako tomu bylo u výpisu uživatelů ve správě uživatelů. Nejprve tedy v `PreRender()` dojde k nastavení parametrů výpisu (maximální počet záznamů na stránku, podle jaké vlastnosti se bude řadit a zda se bude řadit sestupně či vzestupně). Pak se pokračuje načtením záznamů pro zobrazení stejně jako u výpisu uživatelů. Zde je ale použita jiná metoda pro získávání záznamů z databáze. Dále také tato metoda nemůže jen tak vybrat všechny záznamy z dané tabulky v databázi, protože je třeba zajistit chování výpisu, jak bylo popsáno ve specifikaci požadavků. Tedy zajistit, aby administrátor měl přístup do všech vyšetřování a vyšetřovatelé pouze do vyšetřování, které vlastní nebo ke kterým dostali přístup.

Pro získání seznamu vyšetřování, které se mohou uživateli zobrazit se využívá metody `FillInvestigationsList()` z třídy `InvestigationService`. Metodě se kromě objektu datasetu předává také uživatelské jméno aktuálně přihlášeného uživatele. Potom si metoda získá objekt reprezentující uživatele, ze kterého si následně ověří, jakou má uživatel roli. Pokud má roli administrátora, tak do seznamu záznamů doplní všechny záznamy vyšetřování z databáze. Pokud se naopak zjistí, že se jedná o uživatele s rolí vyšetřovatele, tak se z databáze pomocí dotazu získají záznamy vyšetřování, které mají vlastníka se stejným id, jako má aktuální uživatel nebo vyšetřování, která jsou spárovaná pomocí vazební tabulky s aktuálním uživatelem, což znamená, že uživatel má přístup do daného vyšetřování. Vztah entit `User` a `Investigation` je znázorněno na obrázku 4.3.

Po naplnění seznamu dostupných vyšetřovaní se ještě musí u každého záznamu vyšetřování stanovit, zda mají být zobrazeny možné akce s daným vyšetřováním. Akcemi k danému vyšetřování je myšleno mazání a úprava konkrétního vyšetřování. Informace o tom, zda se



Obrázek 4.3: Znázornění závislosti mezi entitou uživatele a entitou vyšetřování diagramem.

můžou akce u vyšetřování zobrazit, uchovává vlastnost `CanEditRemove`, kterou má třída `InvestigationDTO` navíc o proti entitě `Investigation`.

Druhý pohled implementací se do jisté míry podobá pohledu pro vytvoření či úpravu údajů o uživateli. Pohled je popsán šablonou `investigation.dothtml` a pozici viewmodelu zastává třída `InvestigationViewModel`. Pohled kromě úpravy dat o vyšetřování musí také umožnit výběr vyšetřovatelům, kteří mají do vyšetřování přístup. Proto výčet vlastností, které viewmodel má je poněkud delší.

- InvestigationId obsahuje id vyšetřování, které chceme editovat.
- Investigation slouží k uchovávání záznamů pro výpis vyšetřování.
- Investigators slouží k uchovávání seznamu uživatelů pro výběr, kdo z uživatelů bude mít přístup do vyšetřování.
- InvestigatorIDs uchovává seznam id uživatelů, kteří mají mít přístup k vyšetřování.
- InvestgationService umožňuje přístup k třídě poskytující služby týkající se vyšetřování.
- UserService umožňuje přístup k třídě poskytující služby týkající se uživatelů.
- IsNewInvestigation slouží k zjištění, zda se jedná o nové vyšetřování nebo o vyšetřování existující.
- IsAllSelected je vlastnost pro checkbox, která indikuje, jestli se mají vybrat všichni uživatelé nebo se má výběr zrušit.

U načítání pohledu se postupuje, jak již bylo zmíněno, podobně jako u úpravy uživatele. V metodě `PreRender()` se opět musí kontrolovat, jestli se jedná o prvotní načtení. Pokud se o něj jedná zkontroluje se, zda jde o vytváření nového nebo úpravu existujícího vyšetřování. Opět se to pozná podle předaného id odkazem. V závislosti na této skutečnosti se pak vytvoří buď nový objekt InvestidationDTO nebo se získá tento objekt pro existující vyšetřování pomocí metody `GetInvestigation()` ze třídy InvestigationService a do seznamu InvestigatorIDs se přidají id vybraných uživatelů, kteří mají přístup do vyšetřování.

Jako u správy uživatelů lze pohled opustit kliknutím na tlačítko *Cancel*, kdy prohlížeč přejde zpět na výpis dostupných vyšetřování. Když se vyšetřování bude ukládat, tak se opět

rozlišuje, zda se má vytvářet nové vyšetřování nebo se vyšetřování upravuje. Při vytváření se volá metoda `AddInvestigation()`, které se předává objekt s daty o vyšetřování a id vybraných uživatelů s uděleným přístupem do vyšetřování. Metoda zajistí vytvoření záznamu o vyšetřování v databázi a zaznamená také do vazební tabulky vazby mezi uživateli a daným vyšetřováním, aby uživatelé měli přístup do vyšetřování. Poté ještě metoda zajistí vytvoření adresářové struktury vyšetřování, kam se posléze například ukládají vstupní soubory s odposlechnutou komunikací.

4.7 Propojení s Netfox Framework

Propojení webového rozhraní s Netfox Framework by se dalo rozdělit na dvě části:

1. Část kdy je potřeba zajistit extrahování konverzací ze vstupních souborů s odposlechnutou komunikací a extrahování forenzních dat z extrahovaných konverzací.

2. Část zabývající se zobrazením extrahovaných konverzací a extrahovaných forenzních informací.

Realizace extrahování konverzací a export forenzních dat probíhá následovně. Napojení na Netfox Framework za účelem extrahování konverzací a extrahování obstarává třída `ProcessCaptureService`. Takže se nejprve popíše třída `ProcessCaptureService` a až poté celý proces extrahování konverzací a forenzních dat. Třída `ProcessCaptureService` provádí veškerou svou inicializaci propojení s Netfox Framework v konstruktoru. Konstruktoru je potřeba předat parametr kontejner s registrovanými komponentami, které bude vyžadovat a druhým nezbytným parametrem je id vyšetřování, pro které je potřeba extrahování konverzací a forenzních dat provést. Nejprve se v konstruktoru zjistí jaké jsou všechny dostupné snoopery. Poté se nastaví informace o vyšetřování pomocí třídy `InvestigationInfo`. V objektu třídy `InvestigationInfo` je především nutné nastavit správné:

- id vyšetřování,
- název,
- cestu, kam se ukládají adresářové struktury k jednotlivým vyšetřováním.

Nastavení těchto údajů je nesmírně důležité, protože se pomocí nich ve třídě `InvestigationInfo` generují cesty v rámci konkrétního vyšetřování, například cesta, kde jsou uloženy vstupní soubory se zachycenou komunikací. Následně se musí ještě nastavit kontext k databázi patřící k danému vyšetřování. Po nastavení výše uvedených vlastností se provede kontrola, zda daná databáze existuje a pokud neexistuje, tak se vytvoří. Tímto krokem končí inicializace třídy `ProcessCaptureService` a nyní se může spustit buď zpracování vstupních souborů se zachycenou komunikací na konverzace pomocí metody `ProcessCapture()` nebo se může spustit extrahování dat metodou `ExportData()`.

Teď když víme jak funguje hlavní třída, která provádí extrakci dat, tak si nyní můžeme popsat jak funguje celý proces zpracování. V první řadě musí uživatel přistoupit do vyšetřování a to tak, že klikne na jméno vyšetřování ve správě vyšetřování. Poté se mu zobrazí na levé straně seznam vstupních souborů a exportů protokolů. Pro přidání musí uživatel kliknout na ikonu zeleného plus. To má za následek, že se nastaví ve viewmodelu vlastnost `IsUploadDialogDisplayed` na *true* a tím se zobrazí modální okno za účelem nahrání nového vstupního souboru na webový server. Modální okno obsahuje komponentu z Dotvvm,

která se nazývá FileUpload[2]. Pro účely této práce jsem vybral komponentu z balíku business pack, který je placený. Komponenta FileUpload z balíčku business pack umí oproti své základní variantě, která je zdarma, drag&drop, což je jistě pro každého uživatelsky přívětivější, než zadávat cestu ručně pomocí prohlížeče souborů. Komponenta umožňuje nahrání více souborů současně.

Pro použití komponent je potřeba nastavit cestu k dočasnému úložišti, kam se má nahraný soubor dočasně uložit. To se nastavuje v souboru *Startup.cs*, kde, jak bylo již zmíněno výše v této kapitole, je soubor obsahující konfiguraci webové aplikace. Po nastavení dočasné složky bude potřeba zajistit zpracování vloženého souboru. To uděláme tak, že komponentě FileUpload můžeme nastavit atribut `UploadCompleted`, který řekne jakou metodu má ve viewmodelu zavolat po dokončení nahrání souborů na webový server do složky s dočasnými soubory.

V tomto případě se po nahrání souborů na webový server volá metoda `Process()`. V metodě `Process()` se pro každý nahraný soubor volá metoda `StartProcessCapture()`, jenž spustí HangFire úlohu, která zpracuje následujícím způsobem vstupní soubor. Jako vstup dostane úloha id vyšetřování, k němuž soubor patří a cestu, kde se nachází nahraný soubor. Úloha nejprve překopíruje soubor z dočasného úložiště do složky se vstupními soubory daného vyšetřování. Po překopírování souboru úloha spustí zpracování konverzací ze vstupního souboru se zachycenou komunikací a to pomocí metody `ProcessCapture()` ze třídy `FrameworkController` a předá metodě umístění.

Po spuštění HangFire úloh se modální okno skryje. Po dokončení zpracování HangFire úlohy se objeví nahraný soubor v levé části. Po kliknutí na odkaz nás přesměruje na stránku s detaily o souboru. Tento pohled bude popsán později v druhé části propojení.

Pro extrahování forenzních dat se musí kliknout na tlačítko *Export Data*. To spustí metodu `StartProcessCapture()`, které se předá id vyšetřování, na kterém chceme spustit export. Metoda `StartProcessCapture()` spustí export forenzních dat ze všech souborů se zachycenou komunikací v konkrétním vyšetřování. Úloha si pak vybere všechny L7 konverzace, jenž chce zpracovat. Následně použije metodu `ExportData()` třídy `FrameworkController`, které předá jaké Snoopery má při exportování použít, vybrané L7 konverzace, ze kterých se mají extrahovat forenzní data, cestu ke složce, kde se mají ukládat exporty. Po provedení extrahovaní jsou extrahovaná data uložena v databázi. Tím se dostáváme k druhé části implementace propojení webového rozhraní a Netfox Framework, čím je zobrazení exportovaných dat.

Pro zobrazení extrahovaných dat se využívají dvě třídy:

1. Třída `CaptureService`, která poskytuje veškerá data pro pohled s detailními informacemi o konkrétním souboru se zachycenými daty.

2. Třída `ExportService`, pomocí níž se získávají data pro zobrazení exportovaných forenzních dat.

Pro pohled zobrazení má třída `CaptureViewModel` na starost viewmodel a soubor `capture.dothtml` šablonu. Ve viewmodelu jsou definovány datasety pro zobrazení všech tabulek, které se v tomto pohledu nacházejí. Jedná se o L3, L4 a L7 konverzace, pak ještě o rámce a seznam použitých aplikačních protokolů, seznam použitých transportních protokolů. Dále ve viewmodelu můžeme najít vlastnost `ConversationsStatistics`, která obsahuje statistiku o aktuálním souboru. Kupříkladu můžeme zde nalézt informace kolik soubor obsahuje rámců, kolik má konverzací, kolik dat bylo přeneseno jakým směrem, počet

[2] https://www.dotvvm.com/docs/controls/businesspack/FileUpload/latest

rozpoznaných protokolů atd. Poslední tři vlastnosti, které ještě viewmodel obsahuje jsou vlastnosti týkající se zobrazení grafu použitých aplikačních protokolů. Jsou to vlastnosti:

- `ChartLabels` uchovává jak jsou pojmenovány jednotlivé aplikační protokoly v grafu.
- `ChartData` uchovávají hodnoty v procentech použití jednotlivých aplikačních protokolů.
- `CharChartColors` slouží pro definici, jaké barvy mají být použity.

V metodě `PreRender()` probíhá jako vždy nastavení parametrů zobrazení tabulek a získání záznamů do datasetů. K získání dat do všech tabulek se využívají konkrétní metody ze třídy `CaptureService`, například pro L3 konverzace je to metoda `FillL3Conversation()`. Po získání všech dat se ještě připravují textové řetězce pro inicializaci grafu s aplikačními protokoly.

Pro zobrazení dat exportů z databáze se využívá dvou pohledů. Jeden pohled reprezentuje přehled všech exportů pro daný protokol a druhý pohled slouží pro výpis detailu exportu. Tyto dva pohledy je nutné vytvořit pro každý snooper.

Z toho důvodu bylo potřeba předělat současné snoopery, protože snoopery kromě modelů a třídy pro extrahování v sobě mají součásti z Netfox Detective vztahující se k technologii WPF(Windows Presentation Foundation), kterou používá desktopová aplikace, například viewmodely a pohledy. Proto bylo nutné oddělit tyto části do samostatného projektu nazvaného například *SnooperFTP.WPF*. Dále se u těchto projektů muselo nastavit po sestavení vykonání příkazu, který nakopíruje sestavený projekt do složky s binárními soubory Netfox Detective, jako tomu bylo u původního projektu snooperu.

Poté co se oddělily prvky WPF od snooperů, tak se mohou vytvořit projekty nazvané, kupříkladu *SnooperFTP.WEB*, který bude obsahovat výše zmíněné pohledy pro přehled a detail exportu.

Dále bylo nutné zařídit, aby DotVVM umožnilo zobrazovat stánky uložené mimo hlavní projekt s webovým rozhraním, to totiž ve výchozím stavu DotVVM nepovoluje. Za účelem povolení načítání stránek mimo projekt webového rozhraní bylo potřeba nahradit třídu zodpovědnou za načítání šablon. Ve výchozím stavu se o načítání souborů stará třída `DefaultMarkupFileLoader` implementující rozhraní `IMarkupFileLoader`. Takže se zatím vytvořila nová třída `NetfoxMarkupFileloader`, která dědí třídu `DefaultMarkupFileLoader`. Jediný rozdíl vůči výchozí třídě je v metodě `GetMarkup`, kde se vypnula kontrola na umístění souboru se šablonou.

Pohledy přehledu exportů a detailu exportu pro konkrétní snoopery definované například v projektu *SnooperFTP.WEB* vyžínají masterpage, která je definovaná v hlavním projektu. Každý pohled využívá jinou masterpage. Pro přehled exportů je masterpage uložena v souboru `ExportOverview.dotmaster` a funkci viewmodelu zajišťuje třída `ExportOverviewViewModel`, tuto třídu pak dědí všechny třídy viewmodelů vyskytujících se v projektech pro jednotlivé snoopery. Masterpage pro přehled exportů zajišťuje výpis seznamu exportů, seznam hovorů, obrázků atd. Pohled pro detail exportu používá masterpage `ExportDetail.dotmaster`, jejíž viewmodel obstarává třída `ExportDetailViewModel`. Masterpage pro detail pak poskytuje výpis generického pohledu, kde jsou vypsány všechny zprávy dotyčného protokolu. Pro zavedení specifické záložky s jedinečným pohledem pro určitý protokol, lze zapsat v contentpage v projektu pro konkrétní snooper. A to tak, že do jedné komponenty *Content*, která v atributu *ContentPlaceHolderID* bude mít hodnotu *tabsheader*, umístíme název záložky a do druhé komponenty *Content* s hodnotou *tabscontent* parametru *ContentPlaceHolderID* umístíme obsah záložky.

Poslední věcí kterou je potřeba udělat, aby pohledy mimo hlavní projekt webového rozhraní fungovaly, je přidat záznam do routovací tabulky. Přidávání záznamů pohledů jednotlivých snooperů se děje v konfiguračním souboru *Startup.cs*. Zde se pomocí Windsor kontejneru zjistí všechny registrované snoopery, pro které se pak následně přidá záznam do routovací tabulky. Po přidání záznamů do routovací tabulky se pak budeme moci dostat pomocí URL na konkrétní pohled daného snooperu.

Kapitola 5

Testování

V této kapitole se budeme zabývat testováním implementace popsané v předchozí kapitole. Mezi testovanými částmi tedy jsou správa uživatelů, správa vyšetřování a propojení webového rozhraní a Netfox Framework. Za účelem testování byl vytvořen projekt pojmenovaný *Netfox.Web.Tests*. V projektu se nacházejí unit testy pro již zmíněné testované části. Než si popíšeme jednotlivé unit testy, zaměříme se nejprve na to, jak je řešena testovací třída `TestViewModelBase`. Z třídy totiž posléze vycházejí všechny další třídy určené pro testování. Třída `TestViewModelBase` obsahuje následující vlastnosti:

- Container - slouží pro kontejner pro registraci komponent v rámci návrhového vzoru Dependency Injection,
- Configuration - konfigurace DotVVM kontext použitý při vytváření viewmodelů,
- UnitOfWork - přístup k databázovému kontextu,
- Roles - přístup k rolím v databázi,
- Users - přístup k uživatelům uložených v databázi,
- Investigations - přístup k vyšetřováním uložených v databázi,
- UserService - poskytuje služby týkající se uživatelů,
- InvestigationService - poskytuje služby týkající se vyšetřování.

Teď si popíšeme jednotlivé metody této třídy a jejich význam. První metodou je `Init()`. Jedná se o metodu, která se spouští před prováděním celého testování. To znamená, že se metoda zavolá pouze jednou, ať se v rámci testování spouští pouze jeden test nebo více testů naráz, které se spouští postupně za sebou. K tomu, aby se metoda volala tímto způsobem, je potřeba nad metodu napsat anotaci *OneTimeSetUp*. V metodě `Init()` se nejprve provede nastavení aktuálního adresáře na složku, kde se nachází zkompilované binární soubory pro testování. Poté se provede zaregistrování potřebných komponent do kontejneru a mapování entit na DTO. Poslední věcí, kterou ještě metoda vykoná, je přiřazení instance konkrétních objektů všem vlastnostem, které se používají pro přístup do databáze nebo poskytování služeb.

Metoda `SetUp()` je označena anotací *SetUp*. Použití této anotace má za následek, že se bude metoda volat před každým testem. Pokud spustíme více testů naráz, tak bude metoda volat před začátkem každého testu. Při provedení metody `SetUp()` se vytvoří konfigurace pro DotVVM, kterou pak využíváme při vytváření viewmodelů pro testování. Dále

se vytvoří dvě role, a to role administrátora a vyšetřovatele. Role je třeba vytvořit, protože se pro každý test vytváří nová databáze podle connection stringu uvedeného v konfiguračním souboru *app.config*. Název testovací databáze, jenž se pro každý test vytváří, je *NetfoxDBTest*.

Nyní nám zbývají projít metody volané po provádění testu. Opět zde jsou dvě metody. Jedna se provádí po dokončení všech spuštěných testů a druhá, která se spouští po dokončení každého testu. Metoda `Cleanup()`, která je označena anotací *OneTimeTearDown*. Tato metoda se spustí pouze jednou, a to po dokončení všech spuštěný testů. Metoda provede jen jedinou věc, a tou je, že nad kontejnerem zavolá metodu `Dispose()`, která provede vyprázdnění kontejneru s registrovanými komponentami. Zmiňovanou druhou metodou je `Clean()`. Požadavek, aby se metoda spouštěla vždy po dokončení testu, zajistíme anotací *TearDown*. Metoda tedy po dokončení každého testu zajistí, že se smaže testovací databáze, která se vytvořila v metodě `Init()`.

5.1 Správa uživatelů

Když jsme vysvětlili, jak je řešena třída `TestViewModelBase`, tak se můžeme pustit do vysvětlení testování správy uživatelů pro třídu `UserManagementTests`, která zajišťuje testování správy uživatelů, dědí z třídy `TestViewModelBase`. Tím, že se třída `UserManagementTests` dědí od třídy `TestViewModelBase`, zajišťuje, že se přeberou metody, které se mají vykonat před a po spuštění testů, které jsou uvedeny výše. V rámci testování správy uživatelů jsou testovány tyto případy:

- přidání nového uživatele,
- odebrání uživatele,
- odebrání vybraných uživatelů,
- úprava údajů o uživateli,
- odebrání všech uživatelů kliknutím na vybrat vše,
- pokus o přihlášení s nevyplněním hesla,
- pokus o přihlášení se správnými údaji,
- pokus o přihlášení s nesprávným heslem,
- pokus o přihlášení s uživatelem, který není aktivní,
- změna hesla.

Nyní si popíšeme jednotlivé případy. Pro každý z uvedených případů se ve třídě `UserManagementTests` vyskytuje metoda. Před každou testovací metodou se nachází anotace *Test*. Tím, že použijeme anotaci *Test*, označíme metodu za testovací.

Přidání nového a editace existujícího uživatele

Prvním uvedeným testovaným případem je přidání nového uživatele. Testování tohoto případu má na starost metoda `TestAddUser()`. Nejdříve je potřeba přidat záznam do routovací

tabulky pro odkaz na pohled s výpisem uživatelů, protože se po úspěšném přidání uživatele na tento pohled přechází. Routovací tabulku, do které se přidává záznam, lze nalézt v DotVVM konfiguraci, k níž se dostaneme pomocí vlastnosti `Configuration`. Poté je třeba vytvořit kontext požadavku. Pro účely testování se používá pro reprezentaci kontextu třída `TestDotvvmRequestContext`. Při vytváření instance kontextu se nastaví vznikající instanci třídy konfigurace s přidaným záznamem v routovací tabulce a inicializuje se slovník pro parametry. Jak se kontext vytvoří, umístíme do jeho slovníku s parametry novou položku. Položka bude mít klíč `UserId` a bude obsahovat hodnotu `Guid.Empty`. Zbývá už jen vytvořit samotný viewmodel. Roli viewmodel v tomto testu zastupuje třída `UserViewModel`. Vytvářenému viewmodelu předáme kontext.

Po vytvoření viewmodelu se dostáváme konečně k samotnému testování. Podíváme-li se, jak probíhá přidávání uživatele ve webovém rozhraní, kde se nejprve provede metoda `PreRender()`, pak uživatel zadá veškeré údaje o uživateli a poté klikne na tlačítko *Save*, které kliknutím zavolá metodu `Save()`, jenž provede přidání uživatele. Budeme z tohoto procesu vycházet. Nejprve zavoláme metodu `PreRender()`. Jakmile se metoda provede, vyplní se údaje o uživateli, to znamená jméno, příjmení, uživatelské jméno a roli. Tyto údaje jsou ve vlastnosti `Data`. Dále nesmíme zapomenou vyplnit heslo, to se nachází ve vlastnosti `Password`, které se při vytváření uživatele také zadává. Až se doplní všechny potřebné údaje pro vytvoření uživatele, tak se zavolá metoda `Save()` z viewmodelu. Po provedení metody by tedy mělo dojít k přesměrování na pohled s výpisem uživatelů. To, že dojde k přesměrování, se zjistí tak, že se při vykonávání metody bude očekávat zachycení výjimky `DotvvmInterruptRequestExecutionException`. Po zachycení výjimky se ověří, jestli ji vyvolalo opravdu přesměrování. Dále se ještě ověří, zda přesměrování proběhlo na správnou url adresu a jestli se uživatel v databázi uložil se správnými údaji. Pokud souhlasí i tyto dvě podmínky, tak test přidání uživatele proběhl úspěšně.

Test editace existujícího uživatele je realizován metodou `TestEditDataUser()`. Metoda je částečně stejná jako testovací metoda pro přidání nového uživatele. Stejně jako při vytváření uživatele se přidá do routovací tabulky záznam pro pohled s výpisem uživatelů a vytvoří se kontext. Prvním rozdílem je, že se vytvoří uživatel, který se bude editovat a do slovníku s parametry se umístí položka s klíčem `UserId` a hodnotou id přidaného uživatele do databáze. Pak se zavolá metoda `PreRender()`, což způsobí, že se načtou data o uživateli, kterého chceme editovat. Následně se tedy upraví vlastnosti `IsEnable` a `Surname`. Dále se pokračuje jako při testu přidání nového uživatele a to voláním metody `Save()` z viewmodelu. Potom se čeká na zachycení výjimky indikující přesměrování. Až se výjimka zachytí, dojde ke kontrole, zda se údaje o uživateli opravdu v databázi změnily.

Odebrání uživatelů

První testem, který se týká odebrání uživatelů, je test odebrání jednoho uživatele. Testujeme situaci, kdy uživatel klikne na ikonu koše u konkrétního záznamu uživatele. Testování vykonává metoda `TestRemoveUser()`. Aby mohla testovat odebírání uživatele, musí nejprve uživatele vytvořit. Uživatel se vytvoří pomocí metody `AddUser` ze třídy `UserService`. Dále se zkontroluje, jestli se uživatel do databáze zapsal. Následně je třeba vytvořit kontext, jako tomu bylo v předchozím případě, jen se do něj nebude přidávat žádný parametr. Vytvořený kontext poté zase použijeme při vytváření viewmodelu. Tentokrát je viewmodelem třída `UserManagementViewModel`. Pak zavoláme metodu `PreRender()`, která naplní seznam uživatelů. Ze seznamu uživatelů se pak vezme id jediného uživatele v databázi a předá se jako

parametr metodě `RemoveUser()`. Pak už se jen zkontroluje, jestli v databázi není žádný uživatel. Pokud tam nebude, tak test proběhl úspěšně.

Následující testy jsou do jisté míry, až na drobné rozdíly, podobné tomu předchozímu. Jedná se o testy na odebrání vybraných uživatelů nebo vybrání všech uživatelů pomocí checkboxu v záhlaví tabulky s uživateli a jejich smazání. Na začátku obou testů je potřeba, jako v prvním testu na odebírání, přidat nějaké uživatele. Zde přidáme celkem pět uživatelů. Poté se zkontroluje, jestli je v databázi opravdu pět uživatelů. Je-li tomu tak, vytvoří se kontext a viewmodel. Opět se zavolá metoda `PreRender()`. Od toho bodu se obě metody odlišují.

Při testování odebrání pouze vybraných uživatelů se do seznamu id vybraných uživatelů přidají id prvních dvou uživatelů z výpisu. Tím se simuluje výběr uživatele zaškrtnutím checkboxů u záznamu. Následně zavolám z viewmodelu metodu `RemoveSelectedUsers()`. Po vykonání metody se zkontroluje, zda v databázi zbyli 3 uživatelé, v případě že ano znamená to úspěch testu.

U testování situace, kdy uživatel vybere všechny uživatele pomocí checkboxu v záhlaví tabulky s uživateli a bude je chtít smazat, se postupuje po vykonání metody `PreRender()` stejně jako v předešlém testu. Následně se vlastnosti `IsAllSelected` z viewmodelu nastaví hodnota *true* a zavolá se metoda `SelectAllUsers()`, také se nacházející ve viewmodelu. Provedení těchto dvou operací představuje kliknutí na checkbox v záhlaví tabulky. Potom se zavolá opět metoda `RemoveSelectedUsers()` a poté se ověří, jestli se smazali z databáze všichni uživatelé. Není-li v databázi žádný záznam, tak test proběhl úspěšně.

Přihlašování do systému

U přihlašování se testují čtyři případy, ke kterým může dojít. Testy si jsou opět částečně podobné, takže si zase popíšeme první test do detailu a u ostatních testů týkajících se přihlašování budou zdůrazněny pouze rozdíly vůči prvnímu testu na přihlášení. První testovaný případ je, že uživatel zadá správné uživatelské jméno a heslo. Test zajišťuje metoda `TestLogin`. Jako u ostatních testů se nejprve musí přidat uživatel do databáze. Pomocí jeho přihlašovacích údajů se pak v testech budeme pokoušet přihlašovat do systému. Dále se musí opět vytvořit kontext a viewmodel. Pro pohled, kde se nachází přihlašovací formulář, slouží jako viewmodel třída `LoginViewModel`. Při vytváření viewmodelu se rovnou nastavují hodnoty uživatelského jména a hesla. Následně se volá metoda `DoLogin()`, což odpovídá situaci, kdy uživatel klikne na tlačítko *Login*. Pokud při provádění metody `DoLogin()` dojde k zachycení výjimky typu `NotSupportedException` a výjimka bude obsahovat zprávu „This method can be used only in OWIN hosting!“, tak to znamená, že kontrola přihlašovacích údajů proběhla úspěšně a dochází k přidávání cookie pomocí `SingIn()` v OWIN, což znamená, že uživatel se přihlásil a tím je tento test úspěšný.

Dalším případem, ke kterému může u přihlášení dojít, je, že uživatel zadá špatné uživatelské jméno nebo heslo. Tento případ se otestuje metodou `TestLoginBadPassword()`. Metoda má stejný průběh jako výše popisovaná metoda `TestLogin()` do momentu, kdy se vytváří viewmodel a nastavují se hodnoty pro uživatelské jméno a heslo. Zde se nastaví správné uživatelské jméno a špatné heslo. Následně se také zavolá metoda `DoLogin()`. Pokud zachytíme výjimku `NotSupportedException`, která ponese zprávu uvedenou v předchozím testu, tak test skončí neúspěchem, protože se uživatel přihlásil, aniž by znal správné heslo. Pokud se výjimka nezachytí, tak se na konci testu ještě ověří, zda chybová hláška uložená v `ErrorMessage` je „Invalid username or password!“. Obdobný test je pak test na to, jestli je možné se přihlásit bez zadání nějakého z požadovaných polí. Pro tento test je určena me-

toda `TestLoginEmptyInput()`. Jedním z rozdílů oproti metodě `TestLoginBadPassword()` je vyplnění hesla, kde se zadá vlastnosti pro heslo prázdný řetězec. Druhým rozdílem je, že se na konci nekontroluje obsah vlastnosti `ErrorMessage`.

Posledním případem přihlašování je přihlašování s účtem, který není aktivní. Metoda `TestNotActiveLogin`, jenž testuje tento případ, tedy musí vytvořit uživatele, který má ve vlastnosti `IsEnable` hodnotu *false*. Poté vytvoří kontext a viewmodel. Při vytváření se pak přiřadí správné uživatelské jméno a heslo. Následně se zavolá metoda `DoLogin()` a opět se nesmí zachytit výjimka `NotSupportedException`, což by znamenalo neúspěch v testu. Pokud se výjimka nezachytí, na konci testu se ještě kontroluje, zda chybová hláška obsahuje řetězec „Account is not active!“. Pokud ho obsahuje, tak test skončil úspěšně.

Změna hesla

Poslední popisovaný test, týkající se správy uživatelů, je test změny hesla uživatele. Testování změny hesla má na starost metoda `TestChangePassword()`. Stejně jako v předešlých testech je třeba vytvořit uživatele v databázi. Dále se musí vytvořit kontext a viewmodel. Změnu hesla uživatel provádí na pohledu, který zobrazuje profil uživatele, takže o viewmodel se bude starat třída `ProfileViewModel`. Po vytvoření se viewmodelu přiřadí hodnoty vlastnostem viewmodelu. Mezi vlastnosti, které je potřeba nastavit, patří `User`, do které se přiřadí objekt `UserDTO`, reprezentující aktuálního uživatele, `Password`, které se předá současné heslo a `NewPassword`. Po doplnění vlastností nutných pro změnu hesla se zavolá z viewmodelu metoda `ChangePassword()`. Jakmile se změna hesla provede, tak už stačí zkontrolovat, jestli se změna hesla provedla a tím pádem test proběhl úspěšně. Kontrola se provede tak, že se porovná hash hesla u záznamu uživatele v databázi s hashem hesla, které se nastavilo vlastnosti `NewPassword` ve viewmodelu. V případě, že se hashe hesel shodují, tak ke změně hesla došlo.

5.2 Správa vyšetřování

Pro testování správy vyšetřování je určena třída `InvestigtionTests`. Tato třída se také dědí ze třídy `TestViewModelBase`. Třída obsahuje část podobných testů jako třída `UserManagementTests` pro testování správy uživatelů. Jedná se například o testování přidání a editaci vyšetřování, odebírání vyšetřování. V těchto testech je rozdílem, že se vytváří za účelem testování nejen uživatel, ale i vyšetřování a u editace či přidání vyšetřování se nastavuje místo id užvatele id vyšetřování a výsledky se kontrolují vůči záznamům vyšetřování uložených v databázi. Proto bude popsán u této testované části pouze test zobrazení vyšetřování různým uživatelům.

Testování zobrazení vyšetřování provádí metoda `TestInvestigationList()`. V metodě se otestuje, zda se administrátorovi zobrazí všechny vyšetřování, pak jestli se vyšetřovateli zobrazí pouze vyšetřování, které vlastní a posledním případem je situace, jestli se uživateli zobrazí vyšetřování, do kterých je přiřazen. Metoda tedy musí vytvořit tři uživatele a to, administrátora s uživatelským jménem *admin* a dva vyšetřovatele *owner* a *user*. Následně vytvoří tři vyšetřování, kdy u jedno vyšetřování je vlastníkem administrátor a u zbylých dvou je vlastníkem vyšetřovatel *owner* a k jednomu z těchto vyšetřování má povolený přístup vyšetřovatel *user*. Ještě je potřeba vytvořit instanci viewmodel v tomto případě instanci třídy `InvestigationOverviewViewModel`.

Potom se pro každého ze tří uživatelů volá metoda `FillInvestigationsList()`, které se předá reference na objekt z vlastnosti `Investigations` a uživatelské jméno uživatele.

Pak se zkontroluje počet záznamů, které se získají pro uživatele metodou `FillInvestigationsList()`. Prvně se proces provede pro administrátora, u kterého se očekává, že bude mít zobrazena všechna vyšetřování, což jsou tři vyšetřování. Další na řadě je vyšetřovatel *owner*, který vlastní dvě vyšetřování a v žádném dalším není přiřazený, takže se očekává, že bude mít zobrazené pouze dvě vyšetřování. Posledním testovaným je vyšetřovatel *user*. Ten žádné vyšetřování nevlastní a je přiřazený k jednomu vyšetřování. Očekávaný počet zobrazených vyšetřování je jen jedno vyšetřování. Pokud počty všech vyšetřování u všech uživatelů souhlasí, tak test dopadl úspěšně.

5.3 Propojení s Netfox Framework

Posledním testem, který se v rámci otestování implementace prováděl, je test propojení webového rozhraní a Netfox Framework. V testu se nejprve otestoval proces zpracování souboru se zachycenou komunikací na konverzace. Poté se ještě v testu otestuje exportování forenzních dat. Test pro otestování propojení se nachází v metodě `TestFramework()` v třídě `ExportTests`. Třída jako všechny předešlé třídy dědí třídu `TestViewmodelBase`. Ve třídě se pak přidává inicializace továrny třídy `CaptureService`. Inicializaci provádí metoda, jenž se provádí před začátkem testování. Před touto metodou se vykoná metoda `Init()` z děděné třídy. V testovací metodě `TestFramework()` se v první řadě vytvoří testovací uživatel a vyšetřování. Pak dojde na vytvoření instancí tříd `ProcessCaptureService` a `CaptureService`. Poté se zkontroluje, zda se opravdu vytvořily. Test pak pokračuje spuštěním zpracování souboru se zachycenou komunikací. Pro testovací účely byl použit testovací soubor pro protokol FTP `ftp_xkarpi03_02.pcap` umístěný v testovací sadě pro testování Netfox. Zpracování vstupního souboru započne zavoláním metody `ProcessCapture()` z třídy `ProcessCaptureService`. Po zpracování se zavolá metoda `GetPmCaptureList()`, aby se získal seznam zpracovaných souborů. Zde se zkontroluje, zda se v seznamu vyskytuje jeden prvek.

Teď zbývá v testu otestovat už jen extrahování forenzních dat. Aby se extrahování dat započalo, zavolá se metoda `ExportData()` z třídy `ProcessCaptureService`. Po ukončení extrahování se vytvoří instance třídy `ExportService` a pomocí její metody `FillSnooperExportDataSet()` získáme seznam exportů, z něhož zjistíme počet exportů. Očekávaný počet exportů je pět. Dále ještě sečteme všechny počty extrahovaných objektů v exportech. Očekávaný počet součtu extrahovaných objektů je jednadvacet. Pokud počet extrahovaných objektů souhlasí s očekávaným počtem, tak test skončí úspěšně.

5.4 Výsledky testů

V rámci testování byly provedeny všechny výše popisované testy. Testy výše uvedené proběhly všechny úspěšně. Z toho lze odvodit, že testované implementované části webového rozhraní fungují korektně.

Kapitola 6

Závěr

Jedním z cílů diplomové práce bylo seznámit se a prozkoumat zpracování síťové komunikace nástrojem Netfox Detective. Tím se zabývá druhá kapitola práce, kde jsem nejprve shrnul, co nástroj umí. Dále jsem v kapitole zmínil, jaké technologie byly použity pro implementaci nástroje Netfox Detective. Následně byly popsány vybrané komponenty Netfox Detective, které jsou nutné pro extrahování informací pro forenzní účely. Týkalo se to zvláště komponent PmLib a ConversationTracker, pomocí nichž dochází ke zpracování vstupních souborů na konverzace jednotlivých síťových vrstev. Dále také komponenty ApplicationRecognizer mající na starost identifikování protokolu, který se ke komunikaci používá. Nakonec jsem zmínil komponentu Snopper, která má za úkol ze síťové komunikace získat data pro forenzní účely. Pro každý aplikační protokol existuje daný Snopper. Poté co jsem vysvětlil účel jednotlivých komponent, tak jsem se pustil do objasnění, jak Netfox Detective funguje jako celek. Začal jsem tím, co se začne dít, když uživatel vloží vstupní soubor se zachycenou komunikací až po zobrazení extrahovaných informací.

V třetí kapitole práce jsem se pak zabýval druhým cílem práce a to je návrh implementace webového rozhraní pro Netfox Detective. Nejprve jsem definoval požadavky kladené na jednotlivé části webového rozhraní jako jsou správa uživatelů, správa vyšetřování a uživatelské role. Dále jsem popsal návrhy grafického uživatelského rozhraní některých částí webové aplikace. Návrhy jsem znázornil wireframy.

V rámci práce bylo implementováno webové rozhraní. Nejstěžejnějším framworkem pro celou implementaci je DotVVM. Framework podporuje vývoj webových aplikací pro obě .NET platformy, tedy .NET Framework a .NET Core. Další použité knihovny či frameworky jsou zmíněny v páté kapitole. V rámci webového rozhraní uživatel může vytvářet, mazat, upravovat další uživatele, ale musí k tomu být v roli administrátora. Dále uživatel může vytvářet, mazat a upravovat vyšetřování. Tyto akce může provádět jen pokud je administrátorem nebo vlastníkem vyšetřování. K vyšetřování pak má přistup buď administrátor, vlastník nebo vyšetřovatel s přiděleným přístupem k danému vyšetřování. V konkrétním vyšetřování pak uživatel může přidávat soubory se zachycenou komunikací. Zpracované soubory se pak zobrazí v levé části po rozbalení menu se soubory. Při kliknutí na soubor se zobrazí detailní informace o souboru jako jsou například statistky, konverzace na určitých síťových vrstvách atd. Dále je umožněno spustit export forenzních dat. Dokončení se pak zobrazí exporty v menu pro výpis exportů pro jednotlivé protokoly. Dále se mohou zobrazit detailní informace o exportovaném objektu. Detailní informace jsou pro každý protokol jiné.

Jednotlivé části implementace byly otestovány. Všechny popsané testy v kapitole šest proběhly úspěšně. Mezi testovanými částmi jsou správa uživatelů, správa vyšetřování a propojení webového rozhraní s Netfox Framework. U správy uživatelů a vyšetřování se testovalo

přidání, úprava a varianty odebrání záznamů. U vyšetřování se pak testovalo navíc zobrazení vyšetřování pro určitého uživatele. Propojení s Netfox Framework bylo otestováno zpracování konverzací ze vstupního souboru a exportování forenzních dat z konverzací.

Nyní zhodnotíme implementované webové rozhraní z hlediska uživatelské přívětivosti. První stránku, kterou každý uživatel uvidí, je stránka s přihlašovacím formulářem. Vzhled a ovládání stránky je stejné, jaké bychom očekávali u každé takové stránky. Obsahuje tedy pole pro přihlašovací údaje a tlačítko pro přihlášení. Ovládání je zde tedy intuitivní, protože uživatel přesně ví, co má kam vyplnit a co učinit v případě neúspěchu při přihlašování v přihlašovacím procesu, jelikož chybové hlášky se zobrazí dostatečně zvýrazněné a s výstižným textem o jako chybu se jedná.

Další stránkou, na kterou se uživatel dostane po úspěšném přihlášení, je výchozí stránka, která obsahuje rychlé ikony pro rychlou navigaci ve webovém rozhraní a seznam posledně navštívených vyšetřování. Takovouto stránku má každá lepší webová aplikace, jako příklad můžeme uvést nějaký redakční systém, jako je Joomla. Rychlé ikony uživateli umožní se dostat například na stránku, kde bude moci vytvořit vyšetřování jedním kliknutím a nebude se muset proklikávat skrz další pohledy. K tomu účelu jsou vypsaná poslední vyšetřování, protože je velice pravděpodobné, že uživatel bude chtít po přihlášení vstoupit znova do posledně navštíveného vyšetřování.

Další pozitivní věcí, co se týče uživatelské přívětivosti, je, že ovládání webového rozhraní je intuitivní. Ve webovém rozhraní je to zařízeno několika skutečnostmi. Uživatel vždy ví, kde se zrovna nachází, například to může zjistit z nadpisu. Uživatel také vždy ví co, může udělat na dané stránce. Další vlastností jakou určuje webové rozhraní svoji přivětivost, je že komponenty vykonávající stejnou činnost mají být na stejném místě a vypadat stejně. Tuto skutečnost webové rozhraní splňuje. To, že komponenty vypadají stejně, zajišťuje DotVVM a Bootstrap. Příklad toho, že komponenty jsou na stejném místě, může být stránkovaní záznamů v tabulkách či umístění tlačítek v jednotlivých pohledech. Takže se dá z uvedených skutečností vyvodit, že webové rozhraní je uživatelsky přívětivé a ergonomické.

Nyní bych chtěl porovnat webové rozhraní s podobným nástrojem *Network Miner*. Byť je nástroj pouze v desktopové variantě, nic nebrání tomu porovnat ho z hlediska uživatelského prostření. Obě uživatelská rozhraní jsou podobně složitá na pochopení a na učení se pracovat rychle v daném prostření. Výhodou webového rozhraní je, dle mého názoru, větší přehlednost ve výpisu statistik o vstupních souborech. U výpisu exportovaných dat ve webovém rozhraní není takový chaos, lze vybrat konkretní protokol, jehož exporty se mají zobrazit.

Literatura

[1] Bootstrap · The most popular HTML, CSS, and JS library in the world. [online], [cit. 2018-01-10].
URL https://getbootstrap.com/

[2] Akinshin, A.: *Getting Started with Knockout.js for .NET Developers*. Community experience distilled, Packt Publishing, 2015, ISBN 9781783984015.

[3] Matoušek, P.; Pluskal, J.; Ryšavý, O.; aj.: Advanced Techniques for Reconstruction of Incomplete Network Data. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, ročník 2015, č. 157, 2015: s. 69–84, ISSN 1867-8211.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10864

[4] Microsoft: Implementing the Model-View-ViewModel Pattern. [online], [cit. 2018-01-10].
URL https://msdn.microsoft.com/en-us/library/ff798384.aspx

[5] Microsoft: Introduction. [online], [cit. 2018-01-10].
URL https://www.dotvvm.com/docs/tutorials/introduction/latest

[6] Microsoft: The Repository Pattern. [online], [cit. 2018-05-18].
URL https://msdn.microsoft.com/en-us/library/ff649690.aspx

[7] Nagel, C.: *Professional C# 7 and .NET Core 2.0*. Wiley, 2018, ISBN 9781119449249.

[8] Odinokov, S.: Hangfire Overview. [online], [cit. 2018-01-10].
URL https://www.hangfire.io/overview.html

[9] Pluskal, J.: NetFox.Framework - The network forensic extandable analysis tool. In *Proceedings of the 20th Conference STUDENT EEICT 2014 Volume 2*, Brno University of Technology, 2014, ISBN 978-80-214-4923-7, s. 280–282.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10692

[10] Pluskal, J.: Netfox Detective 2.0 - Nástroj pro síťovou forenzní analýzu. Technická zpráva, 2017.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11567

[11] Pluskal, J.; Matoušek, P.; Ryšavý, O.; aj.: Netfox Detective: A tool for advanced network forensics analysis. In *Proceedings of Security and Protection of Information (SPI) 2015*, Brno University of Defence, 2015, ISBN 978-80-7231-997-8, s. 147–163.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10863

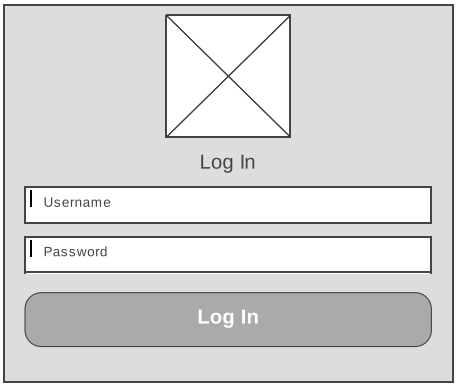
[12] Ragupathi, M.; De Sanctis, V.; Singleton, J.: *ASP.NET Core: Cloud-ready, Enterprise Web Application Development*. Packt Publishing, 2017, ISBN 9781788293204.

[13] RIGANTI: Authentication and Authorization. [online], [cit. 2018-01-10]. URL https://www.dotvvm.com/docs/tutorials/advanced-authentication-authorization/latest

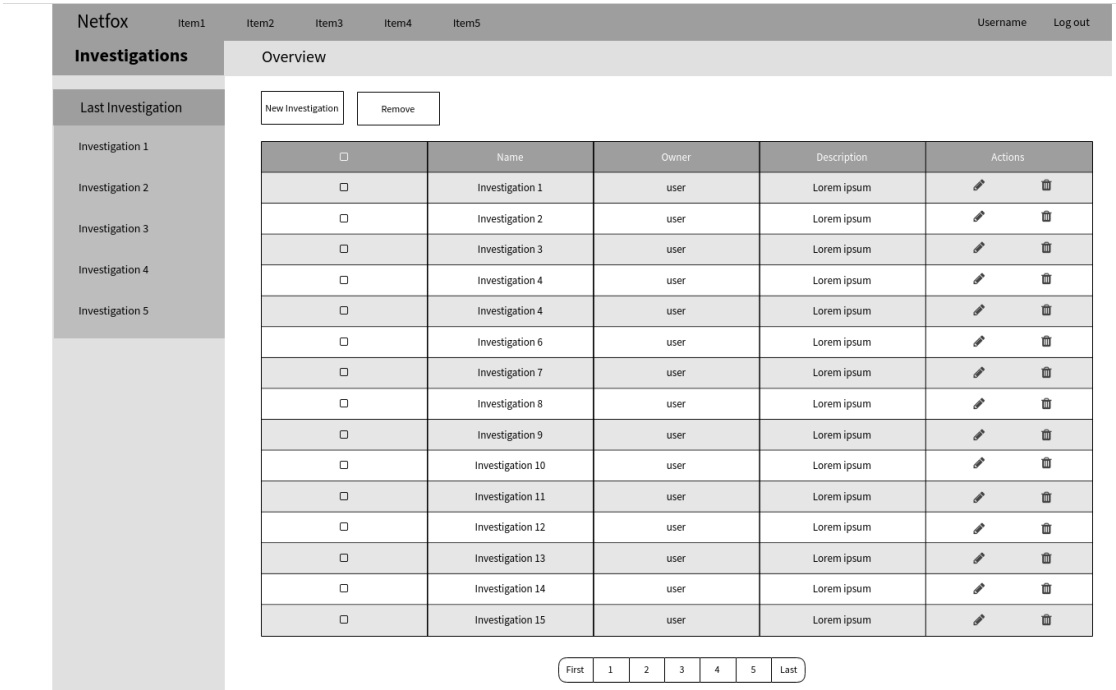
[14] RIGANTI: ViewModels. [online], [cit. 2018-05-18]. URL https://www.dotvvm.com/docs/tutorials/basics-viewmodels/latest

Příloha A

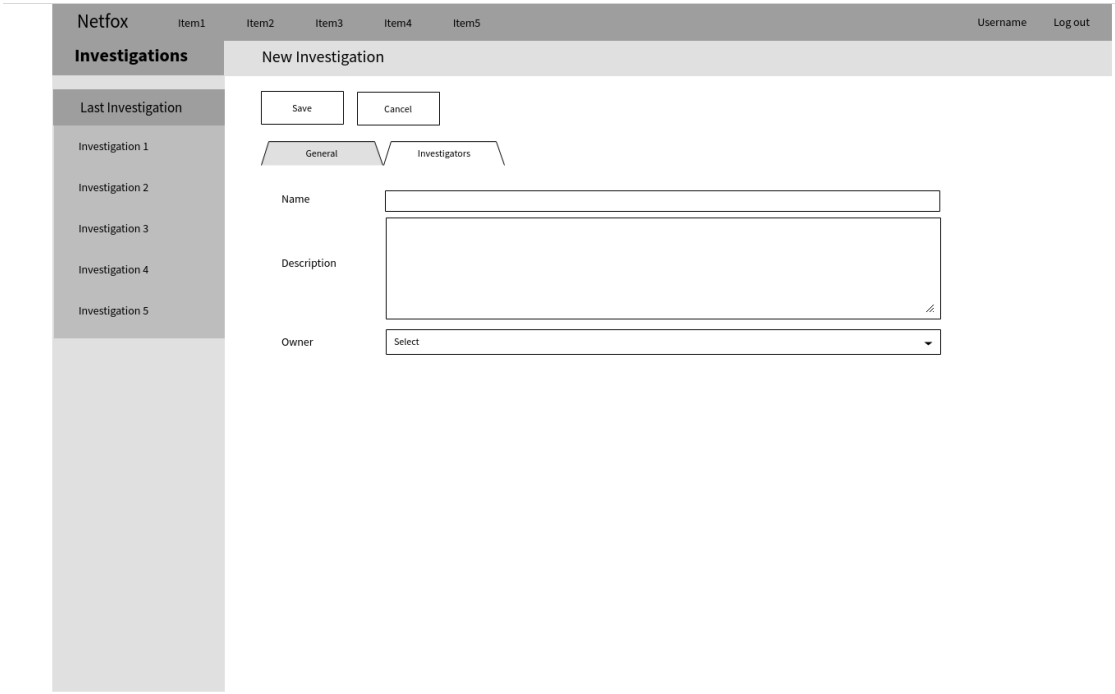
Wireframy



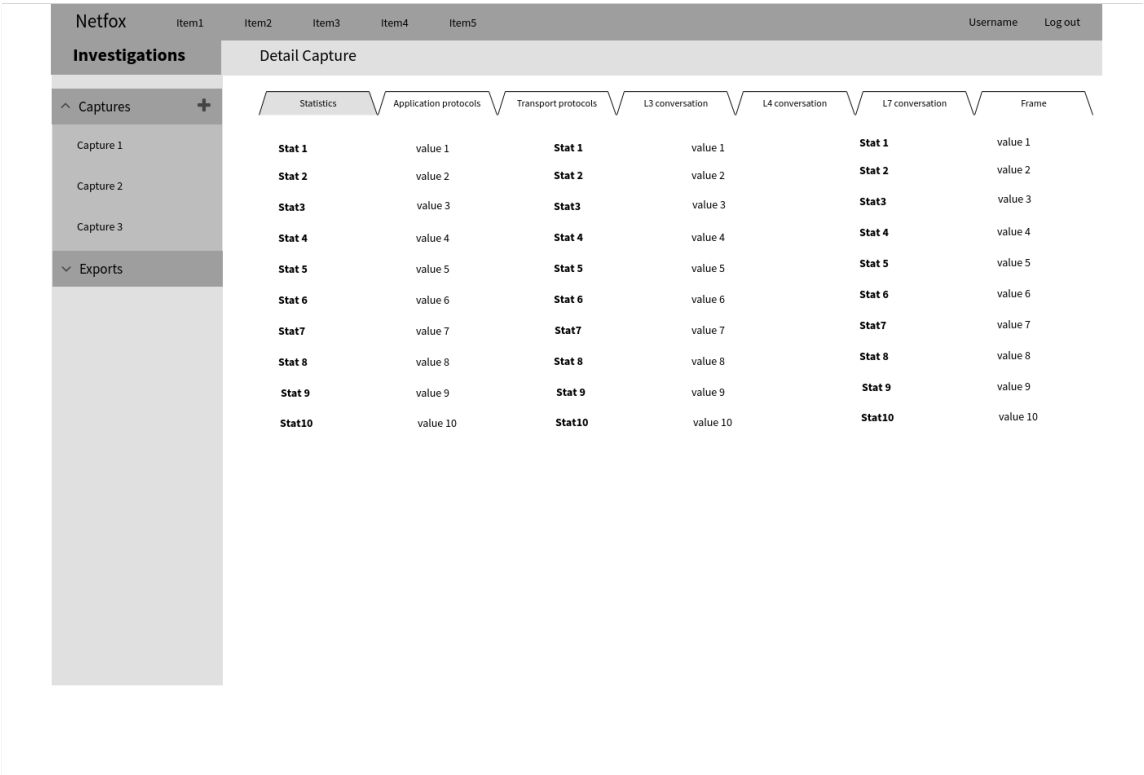
Obrázek A.1: Wireframe pro stránku s přihlašovacím formulářem.



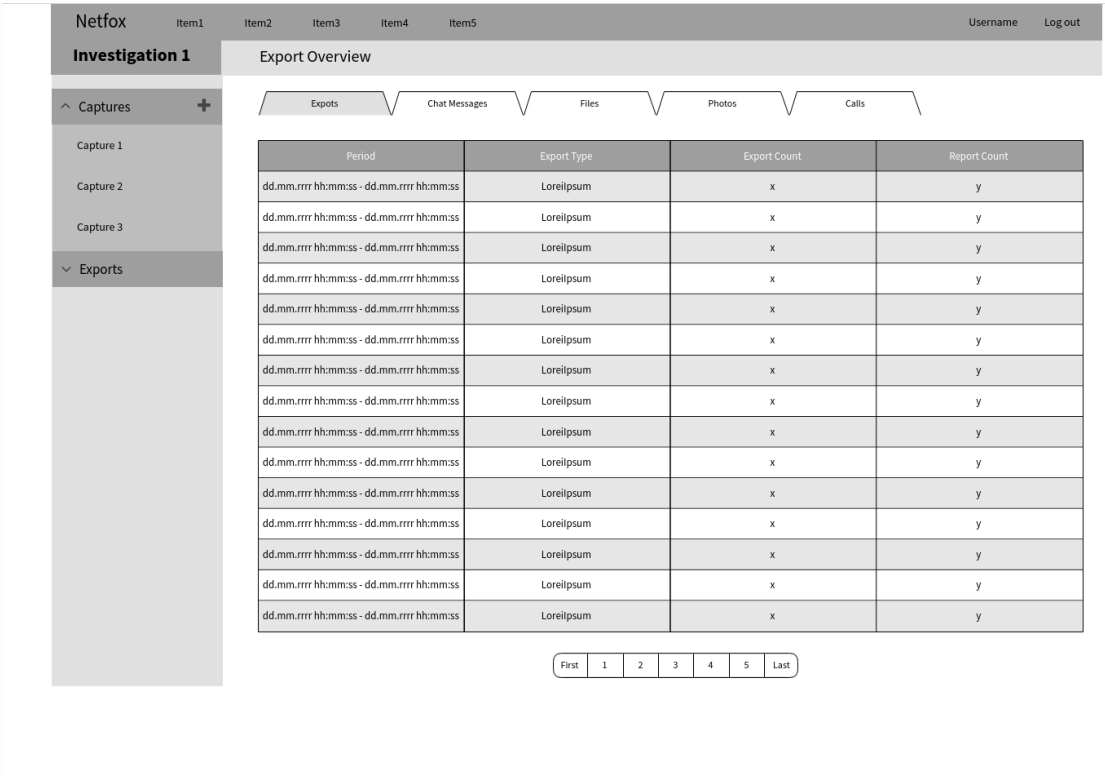
Obrázek A.2: Wireframe pro stránku přehledem vyšetřování.



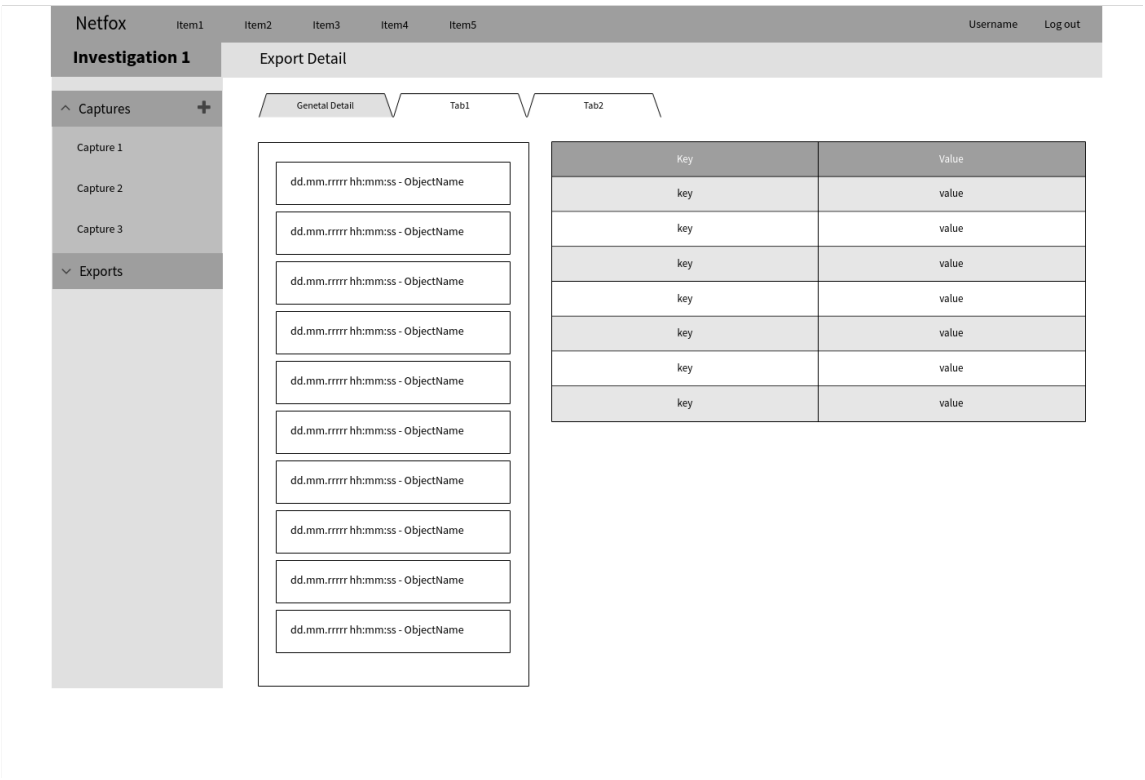
Obrázek A.3: Wireframe pro stránku pro přidání nebo úpravu vyšetřování.



Obrázek A.4: Wireframe pro stránku s detailními informacemi o souboru se zachycenou komunikací.



Obrázek A.5: Wireframe stránky s výpisem přehledu exportů.



Obrázek A.6: Wireframe stránky s detailními informacemi o exportu.

Příloha B

Obsah DVD

DVD obsahuje následující soubory a složky:

- **doc** - složka se zdrojovými soubory textu diplomové práce pro systém LaTeX
- **src** - složka se zdrojovými soubory implementace
- **DP.pdf** - text diplomové práce