VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2019 Lukáš Kořínek

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

UŽIVATELSKÉ ROZHRANÍ PRO EXPERTNÍ SYSTÉM
USER INTERFACE FOR AN EXPERT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Lukáš Kořínek

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Václav Jirsík, CSc.

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Lukáš Kořínek ***ID:*** 195359
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Uživatelské rozhraní pro expertní systém

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s problematikou diagnostických expertních systémů s architekturou klient-server.
2. Navrhněte a vytvořte rozhraní pro webovou aplikaci na straně uživatele.
3. Oživte serverovou část expertního systému NPSCORE na fakultní síťové infrastruktuře.
4. Propojte klientskou a serverovou část expertního systému.
5. Proveďte test funkčnosti realizace a rozbor dosažených výsledků.

DOPORUČENÁ LITERATURA:

KRECHLER, Michal. Diagnostický expertní systém. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav automatizace a měřicí techniky. Vedoucí práce Václav JIRSÍK.

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 20.5.2019

Vedoucí práce: doc. Ing. Václav Jirsík, CSc.
Konzultant: Ing. Petr Petyovský, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Bakalářská práce spočívá v návrhu a realizaci uživatelského rozhraní pro expertní systém. Jedná se o pravidlový diagnostický expertní systém NPS, jenž je vyvíjen na FEKT VUT. Rozhraní je provedeno jako dvojjazyčná webová aplikace založená na moderních technologiích, díky čemuž je dostupná z různých zařízení a má potenciál oslovit široký okruh uživatelů. Uživatelské rozhraní dále rozšiřuje jádro systému o funkce v oblastech autentizace a autorizace uživatelů, ukládání historie, správy uživatelů, správy znalostí a dalších. Aplikace běží na fakultní infrastruktuře a je aktivně testována nejen fyzickými uživateli, ale i automatizovanými testy. Teoretická část dokumentu se věnuje popisu problematiky expertních systémů, systému NPS a webových technologií (konkrétně aktuální trendy, komunikace client-server, testování, zabezpečení). V praktické části je popsáno vytvořené uživatelské rozhraní, architektura řešení, implementační detaily dílčích částí, způsoby testování a postup použitý ke zprovoznění na fakultní infrastruktuře.

KLÍČOVÁ SLOVA

Expertní systém, NPS, Znalost, Konzultace, Web, Klient - Server

ABSTRACT

This bachelor's thesis is about design and implementation of an expert system user interface. Concerned system is a universal rule-based diagnostic expert system called NPS, which has been developed at FEEC BUT. The interface has been created as a bilingual web application based on modern technologies, which might bring the opportunity to reach variety of devices and a broad user audience. It also enhances the system with means of user authorization, history browsing, user management, knowledge management and more. The application runs on faculty's infrastructure and is undergoing active testing by both physical users and automated tests. Theoretical part of the document describes knowledge and expert system problematics, followed by properties of the NPS system and discussion about web technologies (actual trends in the field, client to server communication, testing strategies and security). Practical part then consists of created interface overview, system architecture description, implementation details, testing and steps used to deploy the application on faculty's network infrastructure.

KEYWORDS

Expert system, NPS, Knowledge, Consultation, Web, Client - Server

KOŘÍNEK, Lukáš. *Uživatelské rozhraní pro expertní systém*. Brno, 2019, 82 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: doc. Ing. Václav Jirsík, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Uživatelské rozhraní pro expertní systém“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu práce panu doc. Ing. Václavu Jirsíkovi, CSc. a konzultantovi panu Ing. Petru Petyovskému, Ph.D. za podnětné konzultace, rady a podporu, které mi dopomohly k realizaci této práce.

Brno ....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Expertní systémy jsou počítačové programy, nástroje umělé inteligence, které za pomocí vhodně zakódovaných znalostí převzatých od expertů pomáhají řešit složité úlohy a asistují při rozhodovacích procesech. Nacházejí své uplatnění v medicíně, chemii, geologii, obchodu, matematice, při diagnostice poruch průmyslových zařízení a v dalších oblastech [3].

Systémy umělé inteligence se dnes soustřeďují převážně na užití metod strojového učení. Reprezentace „znalostí“, kterou si tyto algoritmy učením vytvoří, je pro člověka neprůhledná a její správnost či přesnost lze měřit pouze s využitím statistiky. Expertní systémy oproti tomu fungují zcela průhledně a deterministicky. Jejich znalosti jsou explicitně definovány člověkem, expertem, a zapsány znalostním inženýrem. Získávání a formulace znalostí pro expertní systém je dlouhodobý proces, nevyžaduje však rozměrnou množinu existujících dat ani hardware. Modularita znalostí umožňuje expertní systém snadno doplňovat o nové znalosti dle potřeby.

Uživatelé výpočetní techniky dnes stále více pracují s nástroji a službami umístěnými na webu. Webové služby s sebou přinášejí komfort v podobě sdílení dat, časové a prostorové dostupnosti, automatické aktualizace apod. Cílem práce je proto vytvořit webovou aplikaci v souladu s moderními trendy, která by využila potenciál webu jako vhodného média pro centralizované uložení klíčových dat i znalostí, snadné získávání uživatelů a další propagaci i rozvoj systému. Uživatelé musejí být schopni s expertním systémem pracovat kdykoliv a kdekoliv, z jakého zařízení chtějí.

Práce navazuje na předchozí práci „Diagnostický expertní systém“ Ing. Michala Krechlera [5], který vytvořil programovou realizaci matematického aparátu systému NPS vhodnou pro použití společně s webovou aplikací.

Kapitola 1 vysvětluje základní pojmy ze znalostní hierarchie a definuje expertní systémy, jejich principy a typy.

Kapitola 2 se zabývá expertním systémem NPS a programem NPSCore, který byl použit jako výpočetní jádro pro webové rozhraní.

Kapitola 3 pojednává o webových technologiích a aktuálních trendech v této oblasti. Kromě několika literárních zdrojů čerpá i z vlastních zkušeností autora práce.

Kapitola 4 rozebírá vytvořené uživatelské rozhraní pro expertní systém, jeho provedení a akce, které může uživatel konat.

Kapitola 5 se věnuje programovému řešení. Popisuje vytvořenou architekturu a implementační detaily klientské i serverové části aplikace.

Kapitola 6 popisuje provedení automatizovaných testů.

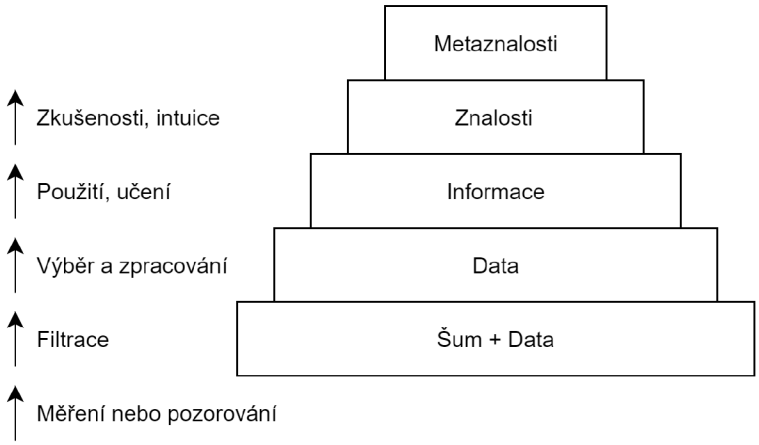
Kapitola 7 obsahuje požadavky aplikace a seznam kroků použitých pro její nasazení na server.

1 Teorie expertních systémů

V kapitole jsou objasněny základní pojmy z teorie expertních systémů.

1.1 Znalostní hierarchie

Ještě před úvodem do problematiky expertních systémů je nutné pochopit pojmy ze znalostní hierarchie. Výklad těchto pojmů se různí v závislosti na autoru a zde uvedené informace jsou v souladu s [3].



Obr. 1.1: Znalostní hierarchie

Každé pozorování přináší výsledky v podobě dat zatížených nežádoucím *šumem*. Šum nemá pro subjekt žádný význam a podle příčiny vzniku jej dělíme na [3]:

- technické nebo mechanické příčiny (poruchy),
- sémantické šumy (míra porozumění, neporozumění),
- psychologické šumy (míra vzájemné důvěry/nedůvěry).

Vhodnou filtrací šumu získáváme *data*. Data jsou zaznamenaná fakta mající pro subjekt nějakou vypovídací hodnotu. Mají potenciál stát se informacemi anebo informace tvořit. Získávání, zpracování a interpretace dat jsou základem chodu digitální techniky a objem dat, který je celosvětově ukládán a zpracováván neustále roste. V současné době se dá mluvit o exabytech (10^{18} bytů) dat zaznamenaných denně[1].

[1] z www.forbes.com - How Much Data Do We Create Every Day?

Dalším zpracováním, strukturováním a výběrem dat získáváme *informace*, které již mají v daném kontextu smysl a týkají se konkrétního problému či činnosti. Z informací již lze vyvozovat konkrétní závěry a odpovídají nám na otázky typu „Co?, Proč?, Kde?, Kdy?“.

Informace je vjem splňující 3 požadavky [3]:

- Subjekt, který vjem přijímá, musí být schopen vjem detekovat a rozumět mu.
- Subjekt musí vědět, co vjem znamená, co vypovídá o něm a o jeho okolí.
- Vjem musí mít pro přijímající subjekt nějaký význam.

Lidské rozhodování a způsoby, jakými řešíme problémy, však nejsou založeny na informacích, nýbrž na *znalostech* (vědomostech). Znalosti jsou vždy výsledkem aktivního učení a dlouhodobé zkušenosti s nějakým jevem, jejich získávání je dlouhodobý proces. Znalost definuje to, jak člověk reaguje na informace, které se k němu dostávají, a je také k určitému člověku (nositeli) nepřenositelně vázána. Odpovídá nám na otázku „Jak?“.

Znalost je schopnost využít své vzdělání, zkušenosti, hodnoty a odbornost jako rámec pro vyhodnocení dat, informací a jiných zkušeností k výběru odpovědi na danou situaci (M. Klein). [3]

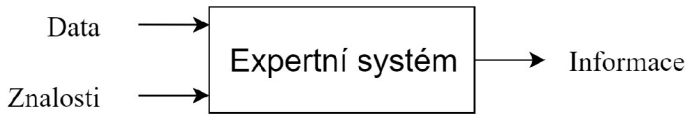
Posledním stupněm pyramidy jsou *metaznalosti*, zjednodušeně „znalosti o znalostech“. Pod těmi si lze představit schopnost vidět hlavní principy a plánovat do budoucna.

1.2 Expertní systémy

Expertní systémy jsou počítačové programy, simulující rozhodovací činnost experta při řešení složitých úloh a využívající vhodně zakódovaných znalostí, převzatých od experta, s cílem dosáhnout ve zvolené problémové oblasti kvality rozhodování na úrovni experta. (Feigenbaum a kol., 1988) [6]

Expertem (znalcem) je člověk, který disponuje znalostmi z oblasti, jež jsou z nějakého důvodu cenné a žádoucí. Typickými oblastmi, kde je potřeba využít znalosti experta, může být například diagnostika, správná interpretace výsledků, vyhledávání, učení a další. [5]

Expertní systémy, někdy též systémy znalostní, spadají mezi systémy umělé inteligence. Jejich cílem je poskytnutí rady svému uživateli, rady, která by mu dopomohla vyřešit nějaký problém. Vstupem expertního systému jsou znalosti experta (skupiny expertů) a data od uživatele nebo jiných zdrojů. Výstupem systému je poté informace pro uživatele, která má formu pořadí, ohodnocení a vzájemného odstupu jednotlivých hypotéz.



Obr. 1.2: Vnější pohled na expertní systém z hlediska znalostní hierarchie

Mezi hlavní rysy expertních systémů patří [6]:

- práce s neurčitostí (dána nejistotami od uživatele i nejistotami ve znalostech),
- užití heuristiky,
- dialogový režim,
- snadná modifikovatelnost znalostí uvnitř systému,
- schopnost rozhodování se na základě neúplných vstupních dat,
- dovednost vysvětlit či zdůvodnit své závěry.

V expertních systémech jsou znalosti centralizovány a explicitně vyjádřeny (vhodně zakódovány) ve formě tzv. báze znalostí. Platí, že kvalita expertního systému závisí více na zakódovaných znalostech, než na programovém řešení inferenčního mechanismu. Získáváním, formalizací a využíváním znalostí se zabývá obor *znalostní inženýrství*. Báze znalostí musí být transparentní a čitelná jak pro experta (aby ji mohl upravovat), tak i pro ostatní pracovníky dané společnosti, kteří se jejím prostřednictvím mohou učit [6].

Z hlediska návrhu expertního systému je důležitá především problematika [6]:

- reprezentace znalostí,
- řešení úloh a prohledávání stavového prostoru jako teoretického základu řešení řídicích mechanismů,
- zpracování neurčité informace.

Obecně jsou expertní systémy používány tam, kde neexistuje algoritmické řešení a je nutno provést určitou formu usuzování. Lépe si lze smysl expertních systémů představit na následujícím příkladu, kterým je návštěva lékaře. Člověk přijde k lékaři, sdělí mu své příznaky formou vzájemného dialogu, a lékař (expert) na základě svých znalostí a zkušeností stanoví diagnózu. Jeho rozhodování se neřídí pouze pravidlem *určitý symptom = určitá diagnóza*. Lékař využívá svoji intuici, míru jednotlivých symptomů, dřívější zkušenosti s pacientem a jiné okolnosti. Navíc možných diagnóz, ze kterých lékař vybírá, je velké množství.

Takové rozhodování je obtížné implementovat běžnými algoritmy a vyžaduje zvláštní řešení právě v expertních systémech. Algoritmické řešení takové úlohy si lze představit jako řadu po sobě jdoucích, pravděpodobně i vnořených, rozhodování typu:

Výpis 1.2.1: Algoritmická realizace expertního systému

```
IF (hodnota1 IN a) AND (hodnota2 IN b) ... THEN
     ...
ELSE
     ...
```

Pro každou hypotézu by musela být vytvořena jedna taková podmínka a pro každý relevantní údaj od uživatele samostatný logický výraz v této podmínce. Nehledě na délku takového programu by bylo obtížné jej upravovat, jelikož každá změna v podobě přidaného údaje od uživatele by ve výsledku musela být reflektována až ve všech těchto podmínkách.

1.2.1 Typy expertních systémů

Podle účelu a vnitřního uspořádání expertní systémy řadíme do těchto typů [6]:

Diagnostický systém - Na základě uživatelem zadaných symptomů (dat, faktů) nalezne nejpravděpobnější diagnózu (hypotézu). Využívá se dvou přístupů. Prvním z nich je přímé usuzování hypotéz ze symptomů s využitím povrchových (heuristických) znalostí, založených na dlouhodobém pozorování. Druhým přístupem je zase odvozování z prvotních principů, což spočívá ve využití hloubkových znalostí, tedy znalostí založených na modelech systémů. Množina výsledných hypotéz je u diagnostického systému zpravidla konečná a předem daná.

Plánovací systém - Na základě známého aktuálního stavu a požadovaného koncového stavu hledá optimální posloupnost kroků, tedy operátorů, kterými lze stavový přechod uskutečnit. Obsahuje subkomponentu provádějící kombinatorickou explozi (generátor možných řešení) a druhou subkomponentu využívající vstupních znalostí a dat pro omezení množství generovaných řešení. Vyhovující řešení pak bývají ohodnoceny podle míry své optimálnosti.

Hybridní/Kombinovaný systém - Sdílí vlastnosti obou předchozích. Jedná se zejména o systémy výukové a monitorovací.

Pro tuto práci jsou stěžejní expertní systémy diagnostické. Jejich implementace může být založena na [6]:

- pravidlech,
- rámcích,
- logickém programování.

1.2.2 Architektura diagnostických expertních systémů

Níže je uveden popis subkomponent diagnostického expertního systému a odpovídající blokové schéma [6].

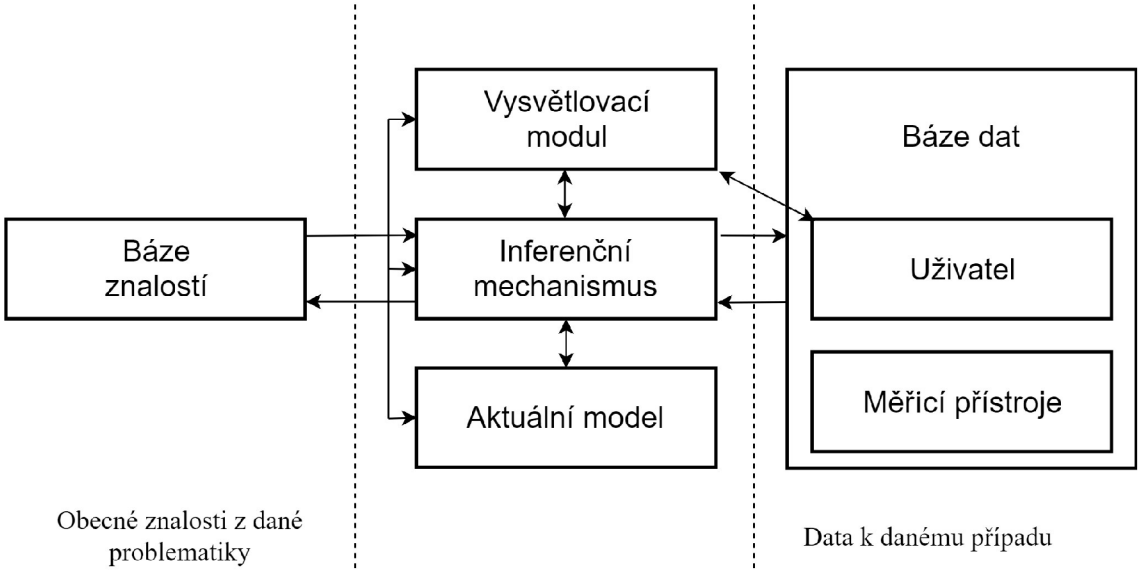
Báze znalostí - Jsou to explicitně vyjádřené a vhodně zakódované znalosti, z nichž systém čerpá. Typickou vlastností je transparentnost a modifikovatelnost (modularita).

Báze dat - Obsahuje data, která budou použita pro zodpovězení pokládaných dotazů. Typicky se jedná o přímé odpovědi uživatele, ale může jít i o data získáná programově se sítě, souborového systému či měřicích přístrojů.

Inferenční mechanismus - Je jádro systému zodpovědné za výběr pokládaných dotazů a úpravu aktuálního modelu po obdržení odpovědi (dat).

Aktuální model - Uchovává současný stav řešení úlohy. Tedy množinu poznatků, kterých bylo během konzultace až dosud dosaženo.

Vysvětlovací modul - Má za cíl uživateli zdůvodnit postup, jakým systém došel ke svým závěrům. Například proč byla položena daná otázka nebo proč byla zamítnuta určitá hypotéza.



Obr. 1.3: Schéma uspořádání diagnostického expertního systému [6]

2 Expertní systém NPS

Kapitola se zabývá expertním systémem NPS a jeho programovou realizací NPSCore, která byla použita jako výpočetní jádro pro webové rozhraní.

Systém NPS byl vyvinut na FEKT (FEI). V rámci práce [5] bylo vytvořeno oddělené výpočetní jádro bez uživatelského rozhraní, které je vhodné pro použití společně s webovou aplikací. V práci [5] je i k dispozici podrobnější popis systému.

NPS je diagnostický expertní systém. Jeho princip je založen na pravidlech, jedná se o pravidlový expertní systém. Inferenční mechanismus systému je navržen pro práci s orientovaným grafem, jehož součástí je několik různých typů uzlů a vazeb mezi nimi.

Nespornou výhodou tohoto systému je, že je univerzální a neorientuje se na určitou oblast či problematiku. Toho je docíleno jednoznačně definovanou syntaxí pro zápis bází znalostí, která popisuje jednotlivé uzly (orientovaného grafu) a jejich vzájemné vazby. Systém je následně schopen libovolné báze této syntaxi odpovídající otevírat ze souborů na disku a pracovat s nimi.

Práce se systémem probíhá na principu konzultace, tedy dialogu mezi programem a uživatelem. Uživateli je položena *otázka* a ten na ni odpoví. Odpověď je zpracována inferenčním mechanismem a dochází k úpravě aktuálního modelu. Úprava aktuálního modelu zahrnuje i uzly cílových *hypotéz*, které slouží jako výstup systému. Pokládané otázky jsou voleny dynamicky, na základě stavu aktuálního modelu. NPS nemá vysvětlovací modul a sám o sobě ani nepodporuje jakékoliv jiné zdroje pro bázi dat, než odpovědi uživatele [5].

Během konzultace dochází k postupné proměně pravděpodobnostních hodnot hypotéz a snižování přidružených hodnot neurčitosti.

Na konci konzultace je žádoucí mít co nejmenší množinu cílových hypotéz s co nejvyšší pravděpodobností a u zbývajících hypotéz naopak pravděpodobnost co nejnižší. [5]

Výsledné pravděpodobnostní ohodnocení hypotéz tedy nesmí mít velký vliv na pochopení výsledků uživatelem. Směrodatné je pořadí hypotéz a jejich vzájemný odstup.[1]

[1] I z tohoto důvodu je v rozhraní pravděpodobnostní hodnota vyjádřena jako tzv. body

2.1 Matematický aparát systému NPS

(Čerpáno z [5]) Veškeré výpočty pracují s pravděpodobností $p \in < 0, 1 >[-]$. Interně je však tato hodnota rozdělena na dvě hodnoty $(T, F) \in < 0, 1 >[-]$, což umožňuje kromě výsledné pravděpodobnosti vyjádřit i spor, nejistotu a nepřítomnost informace.

Pro dvojici (T, F) platí následující:

$(0, 1)$ - Výraz odpovídá pravdivému tvrzení.
$(1, 0)$ - Výraz odpovídá nepravdivému tvrzení.
$(1, 1)$ - Výraz znamená nepřítomnost informace.
$(1 - T)$ - Výraz vyjadřuje míru důvěry v pravdivost tvrzení.
$(1 - F)$ - Výraz vyjadřuje míru důvěry v nepravdivost tvrzení.
$(T \cdot F)$ - Výraz vyjadřuje hodnotu neurčitosti.
$(\frac{F}{T} = \frac{p}{1-p})$ - Výraz vyjadřuje hodnotu pravděpodobnostní šance.

Mezi těmito dvěma hodnotami existuje následující přepočet:

$$T = \frac{F \cdot (1 - p)}{p} \quad , resp. \quad F = \frac{p \cdot T}{1 - p} \tag{2.1}$$

Hodnoty (T, F) se volí dle následujícího pravidla. Pokud je $p \leq 0.5$, volí se $T = 1$ a F se dopočítá dle vztahu 2.1. Pokud je $p \geq 0.5$, volí se $F = 1$ a T se dopočítá. [5]

Pro dvojici (T, F) jsou definovány operátory:

- unární negace,
- binární konjunkce,
- binární disjunkce,
- skládání s odpovědí uživatele,
- skládání silou vazby.

2.2 Popis uzlů inferenčního mechanismu NPS

Orientovaný graf užívaný pro zápis znalostí a také v aktuálním modelu systému NPS definuje tyto typy uzlů [5]:

Pomocný uzel – Obsahuje pouze identifikátor, hodnotu pravděpodobnosti, a může obsahovat pravidla vazeb na jiné uzly.

Cílový uzel – Oproti pomocnému uzlu obsahuje navíc popis cílové hypotézy.

Dotazovatelný přímý uzel – U tohoto typu uzlu je navíc definována otázka, která se v průběhu konzultace položí uživateli. Pro tento typ uzlu se hodí otázky typu „výběr míry souhlasu s otázkou“, kdy uživatel vybere prostřednictvím odpovědi hodnotu pravděpodobnosti, do jaké míry s tvrzením v otázce souhlasí. Po zodpovězení otázky je aktualizována hodnota všech uzlů, které závisí dle pravidel na zodpovězeném uzlu.

Dotazovatelný přímý kvantitativní uzel – Tento dotazovatelný uzel slouží k položení otázky typu „výběr jedné možnosti ze seznamu“. Po zvolení odpovědi uživatelem je automaticky vždy přiřazena uzlu odpovídajícímu zvolené odpovědi hodnota $p = 1$ a ostatní uzly odpovídající otázkám zůstanou nezměněny.

Dotazovatelný běžný uzel – Tento typ uzlu nemá možnost definovat vlastní kontexty ani pravidla. Na rozdíl od dotazovatelného přímého uzlu se tento typ pokládá až v momentě, kdy byly zodpovězeny všechny vhodné dotazovatelné přímé uzly. V tento moment se z pravidel všech cílových uzlů vybere nezodpovězený běžný uzel, který má největší sílu vazby a jde o vhodnou vazbu. Vhodná vazba se určí tak, že předchozí hodnota pravděpodobnosti vazby je větší než aktuální hodnota pro disjunkci, respektive menší pro konjunkci. Každý dotazovatelný uzel může mít atribut „Speciální“; tento atribut ovšem nemá v současnosti v inferenčním mechanismu implementovánu jakoukoliv funkci.

Šíření informací mezi uzly a změny pravděpodobností jsou dány *pravidly* fungujícími jako operátory konjunkce anebo disjunkce. Dále systém podporuje také *kontexty* definující podmínky, za kterých může být konkrétní dotaz položen.

2.3 Program NPSCore

Program NPSCore je novým výpočetním jádrem systému NPS vytvořeným v rámci práce [5]. Je napsán v programovacím jazyce C++ pro platformu MS Windows a využívá hotové knihovny RapidXML[2] pro práci s XML datovými strukturami.

NPSCore byl cíleně vyvinut pro použití v rámci webové aplikace. Nedisponuje žádným grafickým uživatelským rozhraním (GUI), pracuje v prostředí příkazové řádky a komunikuje za pomocí XML jazyka. Program akceptuje příkazy v XML a odpovídá na ně opět v XML, včetně chybových zpráv. Při chybě vrátí textovou zprávu a kód chyby, které lze poté odpovídajícím způsobem zachytit a zpracovat v nadřazené aplikaci.

V programu byl zaveden nový formát pro ukládání bází znalostí s cílem umožnit vícejazyčné provedení báze a zobecnění obsahu jednotlivých uzlů tak, aby se nemuselo jednat o čistý text.

Podporovány jsou příkazy dvou typů, a sice „cmd“ pro práci na úrovni celého programu a „base“ pro práci na úrovni konkrétní znalostní báze. Seznam příkazů je uveden níže, podrobnější výčet lze nalézt v [5].

cmd load_base - Provede načtení báze znalostí a její inicializaci.
cmd close_base - Ukončí konzultaci a smaže aktuální báze znalostí.
cmd convert_base - Konvertuje mezi formáty bází znalostí.
cmd exit - Ukončí program.

base set_state - Nastavení konzultace do uloženého stavu.
base get_language - Vrátí seznam bází podporovaných jazyků.
base set_language - Nastaví jazyk, se kterým báze znalostí pracuje.
base get_info - Vrátí název aktuální báze znalostí a její popis.
base get_hypotesis - Vrátí hodnoty hypotéz odpovídající stavu konzultace.
base get_query - Vrátí aktuální text dotazu a množinu možných odpovědí.
base set_answer - Nastaví odpověď na aktuální dotaz.
base reset_query - Reset stavu aktuální otázky.
base reset_base - Reset celé konzultace do počátečního stavu.
base go_back - Posune konzultaci zpět na předchozí otázku.
base go_ahead - Posune konzultaci vpřed na následující otázku.

[2] http://rapidxml.sourceforge.net/

2.4 Formát bází znalostí podporovaný v NPSCore

Původní verze systému, NPS32, používala svůj vlastní formát bází znalostí *.n32*. Příklad zápisu báze v tomto formátu je v příloze B.0.1. Formát neumožňoval velkou variabilitu obsahu a byl náchylný k chybám při ruční editaci.

NPSCore zavedl nový formát *.nps*, zapsaný v podobě značkovacího jazyka XML. Použití XML bylo výhodné, jelikož splňovalo veškeré nároky na zápis struktury bází znalostí a již bylo využito pro komunikační rozhraní programu.

Program disponuje příkazem schopným staré formáty *.n32* do nového převést, díky čemuž není důvod, aby bylo webové rozhraní se starým formátem *.n32* kompatibilní. Dále bude popsána pouze struktura nového formátu. Příklad zápisu báze v *.nps* je opět přiložen v příloze B.0.2.

Oproti předchozí verzi programu NPS32 byla u programu NPSCore snaha zobecnit obsah všech informací zobrazovaných uživateli tak, aby bylo možné vedle prostého textu zobrazit během konzultace i multimediální obsah. Z tohoto důvodu byl obsah uzavřen do obecného elementu content s metainformacemi o obsahu a tento element program NPSCore zpracová tak, že jej v nezměněné formě předává v rámci odpovědi během konzultace. [5]

Soubor s definicí báze znalostí začíná hlavičkou, ve které jsou uvedeny základní údaje jako je verze báze znalostí, název, krátký popis a jazyky, které daná báze podporuje. Podoba hlavičky je následující [5]:

Výpis 2.4.1: Hlavička báze znalostí .nps

```
<?xml version="1.0" encoding="utf-8"?>
<knowledge_base version="1.0">
    <head default_answers="3">
        <cultures>
            <culture>cs-CZ</culture>
        </cultures>
        <name>
            {content}
        </name>
        <description>
            {content}
        </description>
    </head>
```

Za hlavičkou následuje tělo báze označené tagem `body`, ve kterém je seznam uzlů a jejich popis. Podoba uzlu je [5]:

Výpis 2.4.2: Uzel báze znalostí .nps

```
<node literal="{ID}" probability="{p}" type="{type}">
    <comment>
        {content}
    </comment>
    <contexts>
        <{less/greater} threshold="{thr}" literal="{ID}"/
           >
    </contexts>
    <answers>
        <answer probability="{p}">
            {content}
        </answer>
    </answers>
    <relationships>
        <relationship type="{conjunction/disjunction}"
           alpha="{a}">
            <{positive/negative} literal="{ID}"/>
        </relationship>
    </relationships>
</node>
```

Uzel je dán elementem typu `node`. Atribut `literal` udává vnitřní identifikátor uzlu, `probability` počáteční pravděpodobnostní hodnotu uzlu a `type` typ uzlu dle 2.2. Potomek `comment` uzavírá obecný obsahový blok `content` s popisem zobrazované otázky/odpovědi (uvedeno výše). V případě dotazovatelného uzlu obsahuje `node` ještě potomky `contexts` (s popisem kontextových vazeb na jiné uzly) a `answers` (s výčtem možných odpovědí dotazu). Uzly hypotéz či pomocné uzly obsahují pravidla uzavřená v elementu `relationships`.

3 Webové technologie

Kapitola pojednává o webových technologiích a aktuálních trendech v této oblasti. Kromě několika literárních zdrojů čerpá i z vlastních zkušeností autora práce.

Pro takové uživatelské rozhraní, které by umožnilo získat co největší množství potenciálních uživatelů, je dnes nejvhodnějším provedením webová aplikace.

Výhody webového provedení software jsou:

- snadná dostupnost z hlediska času, místa, platformy, zařízení,
- sdílení dat mezi uživateli a zařízeními,
- snadná propagace prostřednictvím vyhledávačů,
- menší problémy s verzováním, automatická aktualizace,
- není nutná instalace.

Naopak mezi hlavní nevýhody patří centralizace dat ode všech uživatelů (pokud aplikace neběží na tzv. soft layeru či cloudu), s tím spojená nutnost data zálohovat, větší náchylnost vůči útokům a nedostupnost offline.

3.1 World wide web

WWW vytvořil Sir Tim Berners Lee v roce 1989 [9] jako decentralizovanou síť pro sdílení informací mezi uživateli internetu prostřednictvím tzv. hypertextu. Tentýž člověk stál u zrodu prvního webového prohlížeče a konsorcia W3, které dodnes udává specifikace webových služeb. Hypertext je zapsán ve značkovacím jazyce HTML, který je syntakticky podobný (a částečně i kompatibilní) s XML.

Základním principem webu je komunikace mezi klientem zobrazujícím webové stránky a serverem, jenž tyto stránky tvoří na základě souborů, vstupů od uživatele a dat z databází. Vše probíhá prostřednictvím aplikačního protokolu HTTP. Pro prohlížení hypertextových dokumentů (webových stránek) slouží webové prohlížeče (v současnosti sestupně dle popularity Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Opera, ...[1]).

[1] dle `https://www.w3schools.com/browsers/`

Stránky na webu dostupné dělíme na [8]:

Webové prezentace - Cílí na prezentaci určité firmy, lidí či produktu. Důraz na vzhled, optimalizaci pro vyhledávače, reklamu.

Webové aplikace - Jsou zaměřené na transakce s daty, podporují různé procesy a záměry. Obsahují tzv. bussiness logiku = znají postupy, jak s daty manipulovat. Kladou důraz na funkčnost a uživatelskou přívětivost.

E-commerce aplikace - Nebo-li také e-shopy, jsou kombinací webové prezentace a webové aplikace.

Pro webové rozhraní expertního systému nejvíce odpovídá uvedená definice webové aplikace.

3.2 Komunikační protokol HTTP

HTTP, protokol pro přenos hypertextu, je jedním z protokolů aplikační vrstvy ISO/OSI síťového modelu. Pracuje na transportním protokolu TCP, počítá tedy se spolehlivým a spojovaným přenosem. Jeho označení je RFC 2616 (pro verzi HTTP/1.1), respektive RFC 7540 (HTTP/2). Specifikace tohoto protokolu je k dispozici na webové stránce [2]. Dnes je snaha o přechod na zabezpečený protokol HTTPS, navíc využívající protokol TLS (či jeho předchůdce SSL) pro šifrování spojení.

HTTP požadavek zpravidla vzniká prostřednictvím:

Načtení stránky z prohlížeče - Uživatel zadá adresu, obnoví stránku, přejde na odkaz či odešle formulář.

Automatického požadavku ze strany prohlížeče - Prohlížeč si vyžádá načtení zvlášť umístěných skriptů, stylů nebo multimediálního obsahu.

Požadavek typu XMLHttpRequest (AJAX) - Skript na straně klientova prohlížeče pro svou další činnost vyžaduje nová data.

Dotaz od jiného programu - Jiný program, jenž není webovým prohlížečem, si vyžádá data dostupná na síti WWW.

K požadavkům dochází pouze ze strany klienta, nikdy ne serveru. Požadavky dle svého účelu využívají vždy právě jednu z následujících metod:

GET - Získá odpověď, optimálně bez provedení transakcí nad daty.
HEAD - Získá pouze přidružené informace (hlavičku) odpovědi.
POST - Požadavek na úpravu dat na serveru přes prostředníka (službu).
PUT - Požadavek na úpravu konkrétních dat daných url adresou.
DELETE - Požadavek na smazání konkrétních dat daných url adresou.
TRACE - Slouží k diagnostice přesměrování.

Konkrétní serverové chování v závislosti na metodě je definováno implementací serverové části aplikace. V praxi často dochází k upozaďování metod PUT a DELETE ve prospěch metody POST.

Na každý požadavek ze strany klienta vytvoří HTTP server odpověď obsahující kromě textového obsahu ještě také hlavičky (např. s informací o délce odpovědi a jejím formátem) a návratový kód. Ten má za cíl příjemci odpovědi (prohlížeči nebo jinému programu) ve zkratce sdělit, jaký obsah může očekávat a jak by se měl zachovat. Při vývoji webových aplikací je jednou ze základních věcí právě správné zpracování různých návratových kódů, zejména kvůli ošetření chyb.

Základní kódy odpovědí jsou:

200 (OK) - Požadavek proběhl úspěšně.
301 (Permanent redirect) - Stránka přemístěna, je vyžadováno přesměrování.
302 (Temporary redirect) - Je vyžadováno jednorázové přesměrování.
400 (Bad request) - Tvar či obsah požadavku je chybný.
401 (Unauthorized) - Server vyžaduje přihlášení.
403 (Forbidden) - Server odmítá požadavek naplnit.
404 (Not Found) - Požadovaná stránka nenalezena.
500 (Internal server error) - Při zpracování požadavku nastala výjimka.

Více informací k HTTP kódům je uvedeno ve specifikaci protokolu [2].

3.3 API

Společně s rostoucím počtem nejrůznějších webových služeb a IoT zařízení připojených k internetu stále častěji dochází k výměně dat mezi těmito zařízeními, různými webovými službami, stránkami a aplikacemi. Tato výměna zpravidla probíhá automatizovaně, bez vědomí uživatele či vlastníka zařízení.

Služby, aplikace a zařízení si mezi sebou navzájem nepotřebují předávat celé webové stránky, nýbrž jen strukturovaná data. Standard HTML je pro tento účel nevyhovující. Z toho důvodu většinou komunikace probíhá prostřednictvím tzv. API (application program interface). Formát API může být takřka libovolný, i prostý text, nicméně v praxi se tvůrci webových služeb snaží o standardizaci. Sjednocením standardů různých stran je poté umožněna komunikace mezi různými systémy bez nutnosti implementovat nové zázemí pro analýzu a syntézu dat a často i možnost bez větších zásahů měnit účastníky komunikace.

Dnes se používají standardy:

REST - Komunikace založená na vzájemném posílání JSON[2] objektů. Vyznačuje se nízkou náročností jak na implementaci, tak i na zpracování. API nutně neposkytuje žádný návod jak jej používat a nemusí mít pevně definovanou strukturu.

SOAP - Založeno na předávání objektů ve formátu XML, vhodné spíše pro náročnější aplikace. Základem je soubor s definicí verze (normy) rozhraní, podporovaných metod API, jejich parametrů a významu. Tímto souborem se poté řídí čtení i posílání dat. Formát je kontrolován striktně a pro vytvoření správného výstupu je vhodné použít některou z hotových knihoven.

V praxi pro svou jednoduchost převažuje užití REST API.

3.4 Technologie na straně serveru

Každá webová aplikace disponuje svojí serverovou částí (*backendem*). Ta slouží jako generátor HTML stránek či jako zdroj a úložiště dat pro API. Zpravidla také disponuje metodami pro modifikaci dat. Obvykle se jedná o sloučení několika systémů dohromady, kdy každý využívá jiné jazyky a struktury.

Základem je služba HTTP serveru zpracovávající klientské požadavky, dnes buďto Apache httpd, nginx anebo Microsoft IIS.

HTTP server dál volá nějaký HTML preprocesor (jazykový interpreter), jenž má za cíl pro daný klientský požadavek vygenerovat odpovídající HTML stránku nebo výstup z API.

[2] `https://www.json.org/`

Většina programátorské práce na straně backendu se odehrává v jednom z těchto jazykových prostředí:

- PHP,
- NodeJS (JavaScript),
- Java,
- ASP.NET (C#, VB),
- Ruby,
- Python.

Existují i implementace využívající jiné technologie, ale ty jsou ve značné minoritě oproti zmiňovaným. Nejrozšířenější je jazyk PHP, který je interpretovaným jazykem podporujícím jak programování procedurální, tak i objektové (OOP). Pro větší projekty je vhodné jej doplnit o některý z frameworků (Symfony, Laravel, Zend, Nette), které dodají programu standardní strukturu v podobě MVC architektury a obsahují hotová řešení pro řadu běžných úkolů a problémů.

Preprocesory jsou zpravidla připojeny k nějaké databázi. Jedná se zejména o systémy relačních databází, se kterými se pracuje v jazyce SQL (MySQL, MariaDB, Microsoft SQL, Oracle). Existují však i tzv. NoSQL[3] řešení (MongoDB), využívající jiných principů.

Za použití relačních SQL databází může docházet k požadavku modelovat jeden a ten samý typ objektu zvlášť na úrovni databáze a zvlášť jako třídu v aplikaci pro preprocessor. S tím je spojena i obtížná (z hlediska optimalizace a řešení závislostí) problematika hydratace, čili plnění těchto entit na úrovni preprocesoru daty z entit na úrovni databáze. Pro usnadnění práce s datovými entitami byly pro různé preprocesory vytvořeny nástroje typu ORM, zobecňující databázové operace na úroveň příkazů preprocesorů (DoctrineORM[4] pro PHP a Entity framework[5] pro ASP.NET).

3.5 Technologie na straně klienta

Klientská část aplikace (*frontend*) je alespoň v omezeném rozsahu součástí drtivé většiny webů. Jejím cílem je dodat stránce vzhled a interaktivitu, které vytvoří vhodnou reprezentaci dat a uživateli zpříjemní procházení webu. Skládá se ze souborů s kaskádovými styly (CSS) definujících vzhled prvků stránky a skriptů (programu).

[3] https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL
[4] https://www.doctrine-project.org/projects/orm.html
[5] https://docs.microsoft.com/en-us/ef/

Moderní prohlížeče jsou založeny na skriptovacím jazyce JavaScript (dle standardu EcmaScript) a jiné jazyky přímo nepodporují. Primárním účelem tohoto jazyka je manipulace tzv. DOM (document object model), modelujícího vnitřní strukturu stránky tak, jak ji chápe prohlížeč. Komponenty v DOM jsou zpravidla vázány na jednotlivé elementy v kódu HTML. Jedná se však o jazyk neustále se rozvíjející a v souvislosti se zavedením živého standardu HTML5 získal mnoho dalších vlastností.

Výsledná frontendová aplikace bývá zpracovávána nástroji pro vytváření balíčků jako je Webpack a browserify. Nejčastější úkony prováděné těmito nástroji jsou:

- kompilace stylů vytvořených nějakým preprocesorem do nativního CSS,
- minifikace skriptů snižující objem přenesených dat,
- zneprůhlednění činnosti skriptů (kód je čitelný uživatelem),
- generování javascriptového kódu kompatibilního se staršími prohlížeči.

Jen malé množství webů využívá čistě nativní javascript bez použití nějaké nástavbové knihovny. Běžné manipulace s DOM jako je vyhledávání, přidávání, úprava či odebírání elementů na stránce jsou v něm poměrně složitou operací a není pro ně poskytnuta dostatečná flexibilita. Proto je v oblibě využít některého z frameworků pro tvorbu javascriptových aplikací, z nichž jsou v současné době nejrozšířenější:

- jQuery (není framework, ale často jeho roli nahrazuje),
- AngularJS,
- ReactJS,
- Vue.js.

Dále roste i obliba tzv. SPA (single page application, jednostránková aplikace), kdy se webová aplikace opírá o rozsáhlou frontendovou část fungující jako jedna průchozí webová stránka. Server negeneruje HTML stránky, slouží jen jako zdroj a modifikátor dat. Tato architektura umožňuje uživateli získat obdobnou zkušenost, jako u běžných (ne webových) aplikací.

3.6 Testování webových aplikací

Testování má za cíl kontrolovat zejména sémantickou (u interpretovaných jazyků i syntaktickou) správnost části programu či programu jako celku. Techniky pro webové aplikace jsou shodné s těmi používanými pro běžné aplikace.

Nejčastější a nejjednodušší forma testování je testování manuální. Uživatel (tester) se snaží provést určitou množinu operací nad aplikací a ověřuje jejich výstupy

a úspěšnost (i záměrnou neúspěšnost). Využití lidského kapitálu má však řadu nevýhod. U rozsáhlých aplikací může úprava jedné části kódu způsobit poruchu části zcela jiné a často není k dispozici dostatek prostředků k tomu, aby se neustále kontrolovalo vše. Navíc po objevení problému musí programátor nejdříve chybu zreprodukovat a zjistit její příčinu, což může být časově a někdy i technicky náročný proces.

Z těchto důvodů se používají algoritmické testy vytvořené formou přidruženého programu, nejčastěji napsaného ve stejném jazyce, jako program testovaný. Ty mohou být poté spouštěny programátorem při vývoji, na serveru při nasazení aplikace nebo na některém ze CI (continous integration) nástrojů (Travis CI, Jenkins, CircleCI, ...).

Základní typy testů jsou:

Unit testy (jednotkové testy) - Kontrolují správnost vstupů a výstupů metod samostatně pracující (oddělené) jednotky, nejčastěji objektové třídy. Pro separaci jednotek od závislostí na jiných částech programu se využívá tzv. mockingu, tedy jejich nahrazení prvkem imitujícím jejich chování.

Funkcionální testy - Kontrolují funkčnost aplikace jako celku a úspěšnost vykonávání jejích akcí. Příkladem může být ověření, že všechny stránky aplikace budou zobrazeny bez chyb a s tím správným obsahem.

Integrační testy - Testují vzájemnou kompatibilitu mezi různými subsystémy programu a produkované výstupy.

Testy uživatelského rozhraní - Simulují činnost člověka testera pro automatizaci repetitivních činností.

V jazyce PHP je nejběžnější užití testovací knihovny PHPUnit[6].

3.7 Zabezpečení webových aplikací

Prostředí webu je z hlediska možných rizik díky své přístupnosti nebezpečnější, než aplikace běžící offline na uživatelově počítači. Každá webová aplikace musí dbát na alespoň základní techniky ochrany dat svých uživatelů a zamezení akcí jdoucích proti logice bussiness modelu.

[6] https://phpunit.de/

Základní oblasti zabezpečení webových aplikací jsou:

- autentizace uživatelů,
- autorizace uživatelů a prováděných akcí,
- ochrana proti útokům.

Autentizace uživatelů, čili získávání a ověřování identity uživatele, zpravidla stojí na předávání uživatelského jména (emailu) a hesla, dnes již běžně doplněných o dvojfázové ověření skrze mobilní telefon. Ve speciálních případech se však využívá i jiných prostředků, jako jsou SSH klíče (kryptografie) a biometrické údaje. Po prvotním ověření uživatelovy identity je informace o ní uložena na straně serveru i prohlížeče, buďto s využitím tzv. session a cookie nebo předáváním vygenerovaných klíčů. Pro zamezení vzniku problémů je kritické použití zabezpečeného HTTPS protokolu, vhodná politika tvorby hesel a správné nakládání s nimi.

Autorizace je proces rozhodující, které akce (či návštěvy stránek) může daný uživatel provést a které nikoliv. Většinou bývá založena na přidělování role nebo skupiny rolí, přičemž každá role má v systému určitá práva. Pravidla přiřazující tato práva bývají uložena v databázi nebo přímo v kódu aplikace. Kromě rolí uživatelů je však důležité řešit i platnost požadované akce pro afektované objekty aplikace (např. uživatel nemůže mít přístup k úpravě článku, který je aktuálně veřejný a pro svou editaci musí být neprve stažen).

Webové aplikace, zvláště ty známé, mohou být cílem útoků hackerů, kteří využívají bezpečnostních děr v aplikacích ve svůj prospěch, například získáním citlivých dat. Mezi typické útoky patří [1]:

SQL injection - Využívá nežádoucí situace, kdy jsou parametry akce přímo zasílány jako dotazy na databázi. Zasláním určitých parametrů může dojít k získání citlivých dat z databáze nebo dokonce k její úpravě a smazání.

Cross-site scripting - Útočník využije nedostatečně ošetřených vstupů od uživatele pro spuštění vlastních škodlivých skriptů, jež poté mohou ovlivňovat uživatelovo chování.

Denial of Service - DoS útok nebo jeho distribuovaná varianta DDoS spočívá ve vytváření velkého množství požadavků na cílový server ve snaze jej zahltit tak, aby nebyl již dále schopný obsluhovat požadavky jiných uživatelů.

Útok na hesla hrubou silou - Útočník má vytvořenou sekvenci nejpoužívanějších hesel a jejich postupným zkoušením se snaží odhalit heslo uživatele.

Cross-Site Request Forgery - CSRF je útok, který nutí koncového uživatele provést nežádoucí akci ve webové aplikaci, do které je momentálně přihlášen. [7]

4 Realizované uživatelské rozhraní

Kapitola rozebírá vytvořené uživatelské rozhraní pro expertní systém, jeho provedení a akce, které může uživatel konat.

Diagramy stránek a akcí dostupných pro přihlášeného i nepřihlášeného uživatele jsou v příloze C.

4.1 Shrnutí požadavků na uživatelské rozhraní

Po četných konzultacích a dle zkušeností autora byly pro výsledné webové rozhraní stanoveny následující požadavky a cíle, které pomohly usměrnit jeho tvorbu:

- adopce responzivního designu pro běh na různých zařízeních a form factorech,
- možnost běhu serverové části na školní infrastruktuře (studentském serveru),
- klientská část stavějící na nových technologiích, podpora starých prohlížečů (Internet Explorer) se nepředpokládá,
- podpora vícera jazyků, zejména češtiny a angličtiny (včetně překladů),
- svižný běh bez vysokých nároků, škálovatelnost,
- jednoduchý a funkční design, který uživateli umožní co nejsnadněji provést to, co zrovna vyžaduje,
- implementace běžných funkcí jako je přihlašování, registrace, obnova zapomenutého hesla a emailové potvrzení založení účtu,
- možnost zpětné vazby ze strany uživatele,
- konfigurovatelnost aplikace skrze rozhraní,
- historie konzultací s expertním systémem musí být ukládána, pro daného uživatele dohledatelná a podporující tisk protokolů, tedy dokladů o provedené konzultaci,
- rozhraní umožňující komplexní formát zobrazovaných otázek a hypotéz, včetně podpory multimédií (obrázků, zvukových souborů a videí),
- plynulá navigace mezi částmi aplikace, zpracována nejlépe formou AJAXových požadavků (SPA aplikace),
- veřejně dostupný režim ukázky, který může představit systém novým uživatelům,
- backendová infrastruktura poskytující nástroje pro vývoj a správu životního cyklu bází znalostí,
- možnost provedení konzultace i na starší verzi báze pro zajištění opakovatelnosti,
- příprava na adopci postupů pro naplnění nařízení evropské unie GDPR.

4.2 Výsledné provedení

Rozhraní je jednostránkovou aplikací. Veškeré prováděné akce jsou konány okamžitě bez načítání stránky v prohlížeči, avšak s nutností čekat na data ze serveru. Čekání na požadavek ze serveru je indikováno načítacím kolečkem v příslušném prvku stránky nebo stisknutém tlačítku.

Interně je zachováno rozdělení na stránky. Jedná se však pouze o stránky logické (definují zobrazovaný obsah), nikoliv o stránky fyzické (prohlížeč zůstává stále na stejné adrese). Proto bude pojem „stránka“ dále během popisu používán, ale je jím myšlen pouze logický celek.

Stránka s aplikací obsahuje tyto hlavní prvky (odshora dolů):

Hlavička - Zobrazuje titulek „NPS“, navigaci a volbu jazyka.

Tělo stránky - Zobrazuje hlavní obsah závislý na právě navštívené stránce a právech uživatele.

Patička - Poskytuje obecné informace (autor, rok aktualizace) a odkazy na informativní stránky.

Všechny zobrazované texty disponují dvěma jazykovými variantami, českou a anglickou, jejichž volba je dostupná po celou dobu práce s rozhraním v pravé části hlavičky stránky.

Vzhled stránky je udělán responzivně a jednotlivé jeho prvky jsou upravovány dle aktuálních rozměrů zařízení a úrovně přiblížení. Kromě interakce myší se počítá i s užitím dotykového displeje, klávesnice není podporována (vyjma zadávání textů).

Množina stránek a akcí, které má uživatel k dispozici, je dána v závislosti na právech uživatele a jeho aktuální činnosti. Největší rozdíl je mezi přihlášeným a nepřihlášeným uživatelem. Pro přihlášeného jsou upředňostňovány akce nad bázemi znalostí a konzultacemi, výchozí stránkou je pro něj stránka s výběrem báze znalostí. Nepřihlášený uživatel má k dispozici akce spojené s autentizací a výchozí stránkou je pro něj úvodní stránka aplikace, jež obsahuje krátký popis NPS a expertních systémů.

4.3 Registrace a přihlášení uživatele

Nový uživatel se může do systému *zaregistrovat* volbou tlačítka „Registrovat účet/Register account“ na úvodní straně rozhraní. Otevře se mu formulář, kde zadá své jméno a příjmení, email a heslo. Email slouží jako identifikátor uživatelského účtu. Všechna pole poskytují okamžitou validaci hodnot psaných uživatelem. Heslo musí

mít minimálně 8 znaků a obsahovat písmena i číslice. Po vyplnění údajů musí uživatel potvrdit souhlas s poskytnutím osobních údajů a následně se zpřístupní odeslání formuláře.

Odesláním registračního formuláře bude uživatel vyzván ke kontrole své emailové schránky, do které mu přijde email s odkazem pro aktivaci účtu. Aktivace je nezbytná pro přihlášení nově vytvořeného uživatele do systému. Otevřením aktivačního odkazu bude účet aktivován a uživatel se může přihlásit.

Pokud je email zadaný při registraci již v databázi, bude uživateli umožněno buďto získat nový email k aktivaci účtu (pokud účet není aktivovaný) anebo zobrazena výzva k přihlášení.

Existující uživatel se do systému *přihlásí* volbou tlačítka „Přihlášení/Sign In“ na úvodní straně a zadáním svého emailu a hesla. Přihlašovací dialog nabízí volbu zapamatování emailové adresy pro příští přihlášení. Zapamatování hesla nabízeno není a je ponecháno v kompetenci prohlížeče.

Zapomněl-li uživatel *heslo*, může pod formulářem s přihlášením zvolit tlačítko „Zapomněli jste heslo?/Forgot your password?“. Následně bude vyzván k zadání své emailové adresy, na kterou mu po odeslání formuláře přijde email s odkazem na obnovu hesla. Po otevření odkazu uživatel zadá nové heslo ve stejném tvaru, jako při registraci.

4.4 Výběr báze znalostí

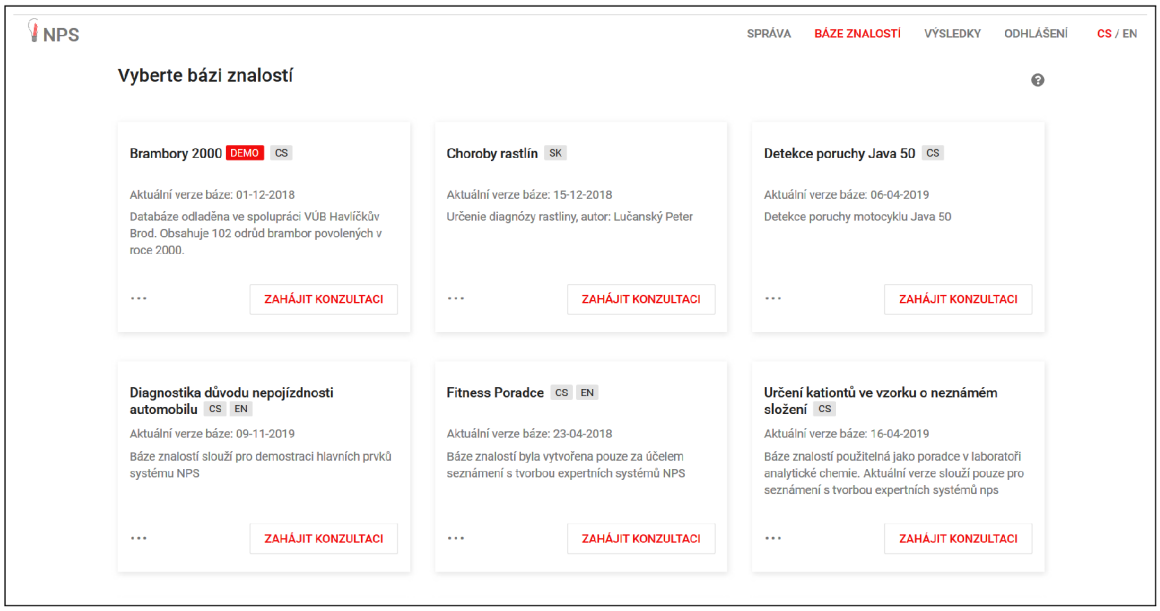
Po přihlášení do aplikace se uživatel dostane k výběru báze znalostí, na které chce provést konzultaci.

Zobrazí se seznam bází znalostí s informacemi o jejich jménu, popisu a verzi. Za jménem báze znalostí jsou štítky s podporovanými jazyky dané báze znalostí a také s jejím stavem (pokud se nejedná o produkční verzi báze). U každé báze znalostí je k dispozici tlačítko „Zahájit konzultaci/Start Consultation“ pro okamžité zahájení konzultace na nejnovější verzi zvolené báze.

Pokud již probíhá konzultace na zvolené bázi znalostí, bude po výběru tlačítka nabídnuta uživateli volba, zda si přeje pokračovat ve stávající konzultaci nebo začít novou.

Pod tlačítkem pro zahájení konzultace je také rozbalovač, jehož volbou dojde k zobrazení sekundárních akcí, tedy zobrazení historie pro danou bázi anebo výběru verze báze.

Volbou výběru verze báze se otevře nová stránka, na které bude zobrazena pouze jedna konkrétní báze znalostí a seznam jejích dostupných verzí identifikovaných datem svého vytvoření. Volbou verze ze seznamu dojde k zahájení konzultace pro danou verzi zvolené báze znalostí.



Obr. 4.1: Výběr báze znalostí ve webovém rozhraní (snímek obrazovky)

4.5 Vedení konzultace

Po výběru požadované báze znalostí dojde ke spuštění konzultace. V levé části obrazovky se zobrazí otázka s dostupnými odpověďmi a v pravé části seznam maximálně osmi hypotéz s největším bodovým ohodnocením. Na malých obrazovkách je seznam hypotéz posunut dolů, pod otázku.

Uživatel vede konzultaci postupným čtením otázek a volbou odpovědí klikem či dotykem. Přitom si prohlíží dostupné multimediální soubory a sleduje průběžné změny v obodování (pravděpodobnostních hodnotách) hypotéz.

Multimédia jsou zobrazována vedle textu otázky. Pokud je jich k dispozici pro danou otázku více, objeví se také ikony pro výběr typu média (obrázek, zvuk, video) anebo šipky pro posun mezi nimi.

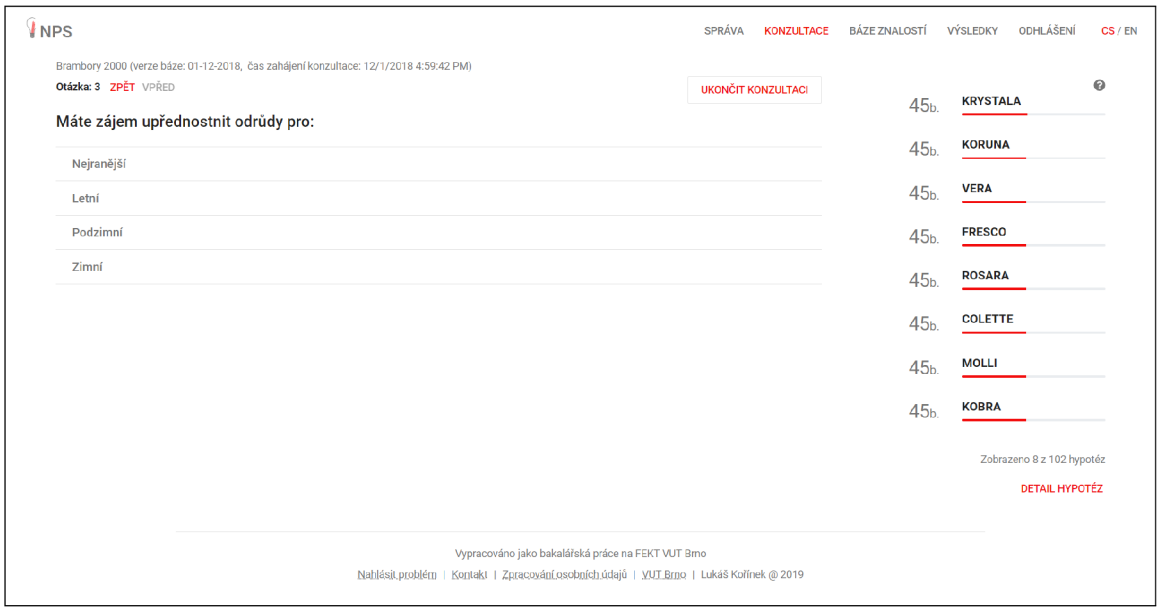
Během konzultace lze kdykoliv zobrazit detail hypotéz klikem na odpovídající tlačítko pod zobrazeným seznamem hypotéz. V detailu uživatel vidí všechny hypotézy, seřazené od té s největším bodovým ohodnocením po tu s nejnižším. Zároveň je zde kliknutím na kartu hypotézy možné zobrazit rozšířené popisky včetně multimédií (na stránce s konzultací jsou zobrazovány pouze názvy hypotéz).

Probíhající konzultaci lze kdykoliv přerušit a vrátit se k ní později prostřednictvím stránky s výběrem báze znalostí, stránky s výsledky konzultací nebo volbou navigační položky „Konzultace/Consultation“.

K ukončení konzultace dojde po získání odpovědí na všechny otázky systému nebo také explicitním ukončením ze strany uživatele volbou tlačítka „Ukončit kon-

zultaci/End consultation“, jež je umístěno nad otázkou.

Po ukončení konzultace dojde k zobrazení detailu konzultace, jenž bude popsán v nadcházející kapitole 4.6.



Obr. 4.2: Vedení konzultace ve webovém rozhraní (snímek obrazovky)

4.6 Protokolování a procházení výsledků

Dokončením konzultace dojde k otevření stránky *Detail konzultace*. Ta slouží jako webový protokol. Ve vrchní části stránky jsou zobrazeny základní údaje o konzultaci:

- jméno uživatele,
- datum a čas zahájení konzultace,
- datum a čas ukončení konzultace,
- název báze znalostí,
- verze (datum vytvoření) báze,
- počet zodpovězených otázek.

Následují dvě záložky. Na první je seznam výsledných hypotéz shodný s tím, jenž se zobrazuje na stránce *Detail hypotéz*. Druhá záložka obsahuje seznam zodpovězených otázek, pro které je ve výchozím stavu zobrazena vždy jen zvolená odpověď. Kliknutím na kartu dané otázky lze zobrazit celý obsah otázky včetně ostatních odpovědí a rozšířeného popisu.

Volbou tlačítka „Tisk protokolu/Print protocol“ v pravém horním rohu stránky dojde k nasazení tiskových stylů na tuto stránku, obsah záložky s otázkami se pře-

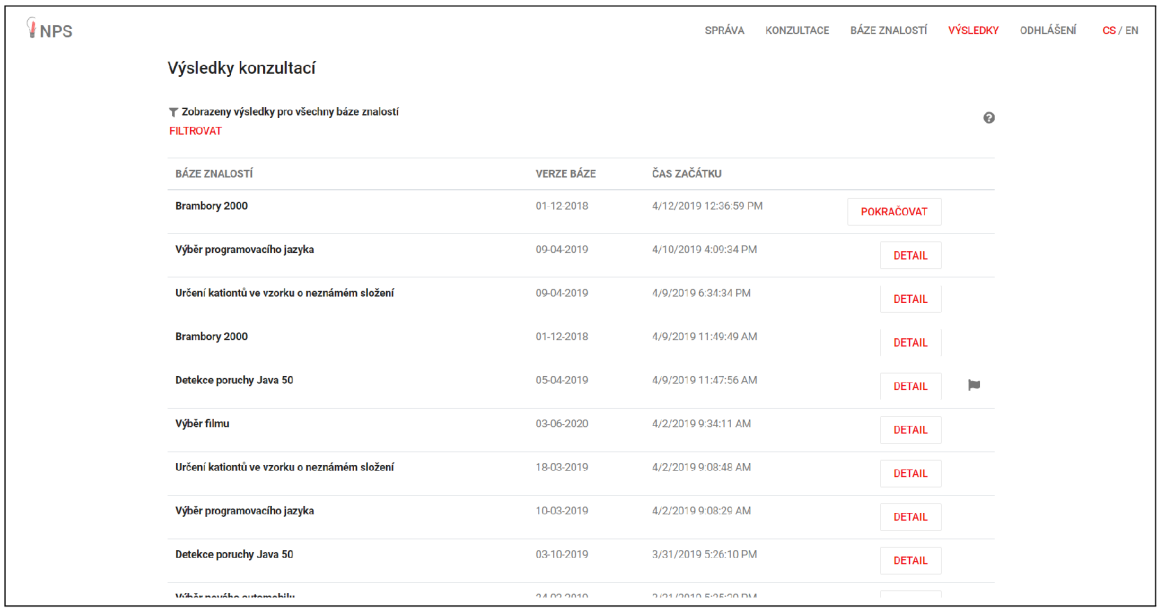
sune pod obsah záložky s hypotézami a bude otevřen tiskový dialog prohlížeče. Dokument jde následně vytisknout, ať už na papír nebo do formátu PDF.

Přihlášený uživatel má k dispozici výsledky (historii) všech svých konzultací a otevření jejich detailů, včetně tisku protokolů. Pro anonymního (nepřihlášeného) uživatele se historie neukládá.

K výsledkům se uživatel dostane volbou navigační položky „Výsledky/Results“. Jednotlivé konzultace budou zobrazeny formou tabulky, seřazené dle data zahájení od nejnovější po nejstarší.

Ke konzultacím, jež mají pro uživatele zvláštní význam, lze přidat příznak prostřednictvím symbolu vlajky zobrazeného po najetí na daný řádek konzultace. Konzultace s příznakem mají poté symbol vlajky zobrazen vždy a jsou tak oproti ostatním zvýrazněny. Opětovným kliknutím na daný symbol je možné příznak i odebrat.

Na stránce s výsledky je umístěn filtr umožňující zobrazení historie pouze pro jednu konkrétní bázi znalostí anebo také filtrování mezi všemi konzultacemi a konzultacemi s příznakem.



Obr. 4.3: Procházení výsledků ve webovém rozhraní (snímek obrazovky)

4.7 Uživatelské dotazy a hlášení chyb

Uživatel má k dispozici dvě formy zpětné vazby. Odkazy k nim jsou přístupné z patičky webu.

Jedná se o:

- kontaktní stránku s kontaktním formulářem,
- stránku pro nahlášení chyby.

Kontaktní formulář umožňuje zadat email a text zprávy, které budou následně odeslány emailem správci aplikace. Stránka pro nahlášení chyby funguje stejným způsobem, umožňuje však i přidání snímku obrazovky s chybou. V obou případech musí uživatel udělit souhlas se zpracováním svých osobních údajů zaškrtnutím příslušného formulářového pole.

4.8 Režim ukázky

Volbou tlačítka „Vyzkoušet aplikaci/Application demo“ na úvodní straně se uživateli spustí režim ukázky (demo). Ukázka má za cíl expertní systém přiblížit novým či potenciálním uživatelům.

V režimu ukázky jsou k dispozici tyto funkce:

- výběr z bází znalostí určených pro ukázku,
- vedení konzultace a zobrazení detailu hypotéz,
- zobrazení detailu konzultace.

4.9 Administrativní sekce

Uživatelé s rolí správce či manažera uživatelů mají zpřístupněnu navigační položku „Správa/Administration“.

Kliknutím se uživatel dostane do sekce pro správu. Cílová stránka je koncipována jako panel rozdělený na subsekce, v současné době pouze s jednou subsekcí „Uživatelé/Users“ (změna aktuální subsekce by byla možná volbou odpovídající ikony v levé části stránky).

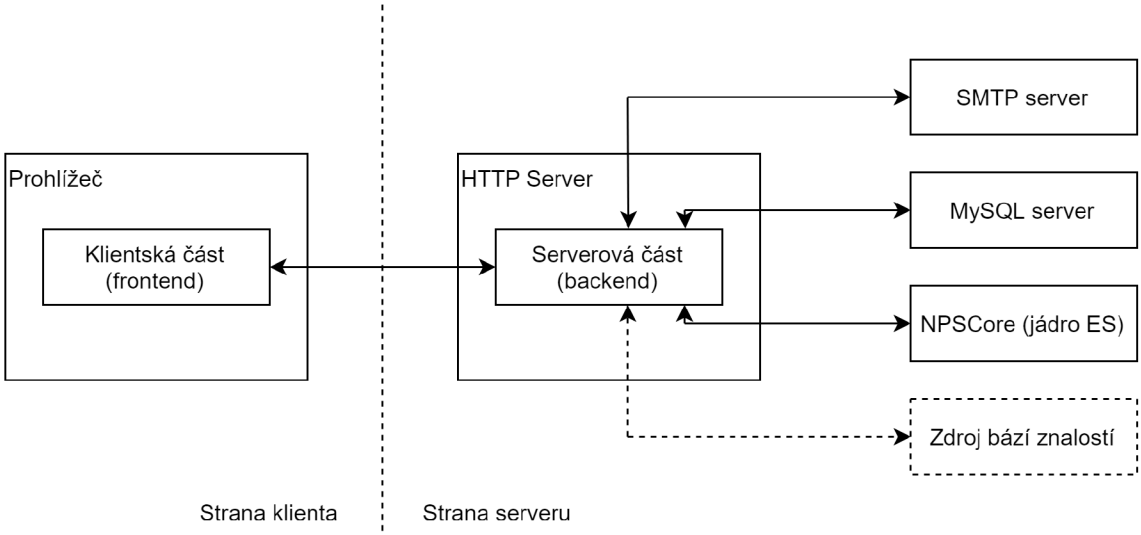
V subsekci uživatelů lze vidět seznam všech uživatelů v aplikaci. Volbou tlačítka „Role/Role“ je možné změnit roli danému uživateli. Tlačítkem „Přístupy/Accesses“ lze zase nastavovat uživatelovy přístupy k jednotlivým bázím znalostí. Subsekce disponuje i filtrem, pomocí kterého lze zobrazené uživatele filtrovat dle role či jména.

5 Programové řešení

Kapitola se věnuje programovému řešení. Popisuje vytvořenou architekturu a výslednou implementaci klientské i serverové části aplikace.

5.1 Architektura řešení

Řešení se skládá ze tří hlavních funkčních bloků. Jedná se o implementačně oddělené subsystémy, každý postavený na jiném souboru jazyků, knihoven a vývojových nástrojů. Konkrétně jde o frontendovou aplikaci (JavaScript, Vue.js, SCSS, Twitter Bootstrap 4, Webpack Encore), backendovou aplikaci (PHP, Symfony framework, Doctrine ORM, MVC, REST API, JWT, composer) a aplikaci expertního systému NPSCore (C++, cmake, make, g++/gcc). Dále řešení vyžaduje služby MySQL serveru (SQL databáze) a SMTP (emailového) serveru. Systém je doplněn i o vzdálený zdroj bází znalostí (složku nebo git repozitář), se kterým je schopný se synchronizovat.



Obr. 5.1: Blokové schéma architektury systému s webovým rozhraním a NPS

5.2 Práce se zdrojovými kódy

Všechny části aplikace jsou průběžně verzovány verzovacím systémem *Git* a umístěny na serveru *bitbucket.org*[1].

Jedná se o pět samostatných repozitářů:

- npscore s jádrem expertního systému,
- npsbackend se serverovou aplikací,
- npsfrontend s klientskou aplikací,
- npsknowledge sloužící jako zdroj bází znalostí,
- npscore_docs s dokumentací ve formátu LaTeX.

Uvedené repozitáře jsou neveřejné. Program NPSCore byl za účelem provedení nutných úprav a dosažení konzistence převeden z původního fakultního *SVN* (Apache Subversion) systému pod správu systému Git.

Vzhledem k tomu, že vývojový tým je jednočlenný, nebyly adoptovány žádné pokročilé metodiky užívání verzovacího systému a veškerý vývoj probíhal v jedné, `master`, větvi. Pro psaní kódu byly užity nepsané, ale konzistentní, coding standardy. Backendová aplikace využívá u všech[2] metod a tříd popisu v dokumentačním formátu phpDoc[3].

5.3 Provedené úpravy na programu NPSCore

Aby bylo umožněno nasazení aplikace na server, byl program NPSCore upraven a přeložen pro unixové systémy.

Postupně byly učiněny tyto kroky:

1. vytvoření nového projektu pro nástroj *cmake* v Microsoft Visual Studio 2017,
2. verzování projektu systémem Git,
3. úprava cest ve zdrojových kódech na unixově kompatibilní,
4. úprava algoritmu pro hledání souboru s defaultními odpověďmi,
5. užití direktiv preprocesoru pro podmíněný překlad u funkcí pro práci s řetězci a časovými intervaly,
6. vytvoření konfigurace pro nástroj cmake (hierarchie souborů CMakeLists.txt),

[1] `https://www.bitbucket.org`
[2] Vyjma testových tříd
[3] `https://www.phpdoc.org/`

7. kompilace programu pomocí programů *cmake* a *make*.

NPSCore se skládá z knihovny s matematickým aparátem a jádra, které ji volá. Při překladu je nutné nejdříve přeložit knihovnu (jako .dll nebo .so soubor) a až následně jádro NPSCore, které je s ní spojeno linkerem. Kompilace je prováděna na standardu C++ 14 a má nastavenou optimalizaci `-O3` pro kompilátor g++ (gcc).

Program se podařilo přeložit na platformách:

- Windows 10,
- Windows subsystem for Linux (Ubuntu),
- Raspberry Pi (Raspbian),
- studentský server FEST (FreeBSD).

5.4 Serverová část aplikace

Serverová aplikace slouží jako komunikační uzel mezi komponentami řešení a poskytuje metody pro získávání a manipulaci dat. Pro její vývoj byl zvolen jazyk PHP doplněný o *Symfony framework*[4] založený na MVC architektuře. Až na nutnou spouštěcí část je celý kód backendu postaven na objektově orientovaném programování (OOP). U PHP se počítá s kompatibilitou s verzí 5.6, což implikovalo užití LTS (long term support) verze Symfony 3.4.

Projekt spoléhá na závislosti zprostředkované správcem PHP balíků *composer*[5]:

- dg/bypass-finals,
- doctrine/doctrine-fixtures-bundle,
- doctrine/doctrine-migrations-bundle,
- doctrine/orm,
- lexik/jwt-authentication-bundle,
- phpunit/phpunit,
- symfony/symfony.

V příloze D je adresářová struktura backendu. Položky ve složce `src` odpovídají stejnojmenným jmenným prostorům.

[4] `https://symfony.com/`
[5] `https://getcomposer.org/`

5.4.1 Architektura serverové části aplikace

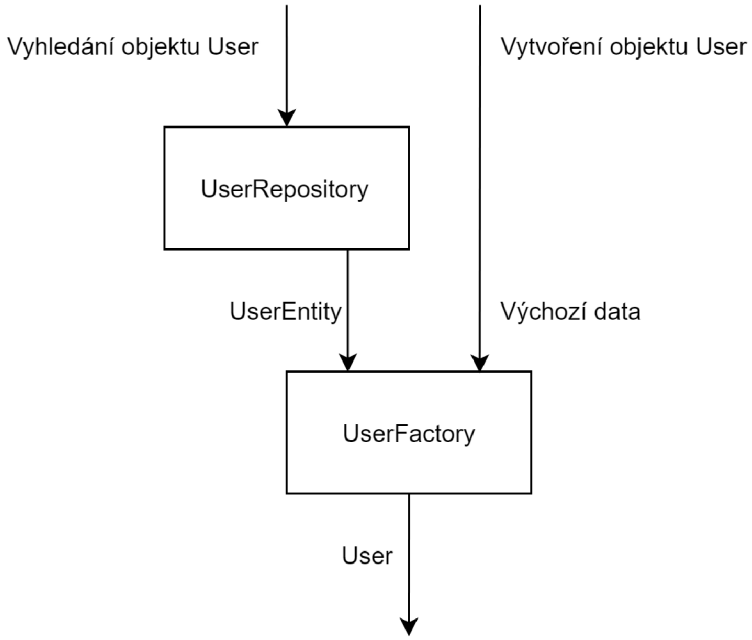
Backendová aplikace dodržuje principy MVC a rozděluje logiku mezi 3 vrstvy:

Model - Definuje vnitřní reprezentaci dat a operace nad nimi.
View - Definuje způsob, jakým budou data prezentována uživateli.
Controller - Je řídicí vrstvou aplikace. Manipuluje s modelem a poskytuje data View vrstvě.

Hlavními elementy modelové vrstvy aplikace jsou doménové objekty reprezentující objekty zájmu (uživatel, znalostní báze, konzultace). Za vytváření doménových objektů jsou zodpovědné továrny (*Factory*), za jejich vyhledávání v databázi repozitáře (*Repository*). Data objektů jsou uložena v entitách (*Entity*). Pro každou třídu doménových objektů existuje odpovídající třída továrny, repozitáře a entity.

Doménový objekt je užíván pro:

- vytvoření výstupu (odpovědi) aplikace na základě jeho dat,
- provedení změn na něm samém užitím některé z jeho metod,
- další vyhodnocení službami aplikace.



Obr. 5.2: Význam základních objektů modelové vrstvy

Mimo doménové objekty a třídy pro jejich správu jsou součástí modelu ještě služby (pro komunikaci s NPS, zpracování obsahu z NPS), samostatné datové objekty (pro práci s XML) a pomocné třídy pracující s řetězci a souborovým systémem.

Služby, repozitáře a továrny mají závislosti na jiných objektech a jsou používány v rámci vzoru *Dependency Injection* sloužícího k řešení závislostí. Řešení je implementováno frameworkem.

Řídicí vrstva obsahuje 3 typy objektů:

Controllery (řadiče požadavků) - Zpracovávají příchozí HTTP požadavky a vytvářejí pro ně odpověď. Obsahují *akce* (metody), které jsou mapovány na url adresy zadávané uživatelem.
Administrativní příkazy - Umožňují administraci z příkazové řádky.
Obsluhy událostí - Reagují na vzniklé události.

Vzhledem k tomu, že backend je proveden jako API, patří do vrstvy View pouze vestavěné funkce PHP pro kódování a dekódování JSON objektů.

Vzniklé požadavky, události, dotazy na databázi, chyby a varování jsou průběžně zaznamenávány do souborů logovací knihovnou *Monolog*, jež je dodávána jako součást Symfony frameworku.

5.4.2 Práce s databází

Komunikace s databází neprobíhá formou SQL dotazů, ale s využitím knihovny *Doctrine ORM* (viz kapitola 3.4).

Data aplikace jsou na úrovni modelu reprezentována formou databázových *entit*, což jsou třídy mající veřejné atributy popsané s využitím `@ORM` anotací (komentářů). Anotace atributů slouží jako definice databázových sloupců, entita jako definice databázové tabulky. Každá jedna instance entity reprezentuje jeden řádek z databázové tabulky a každý atribut této instance reprezentuje hodnotu odpovídající určitému sloupci. Entity jsou umístěny ve jmenném prostoru `NPS\Entity` a implementují společné rozhraní `IEntity`. Všechny entity mají atribut číselného identifikátoru `$id`, jenž slouží jako primární klíč databázové tabulky.

Operace s daty uvnitř aplikace probíhá:

1. získáním entity z repozitáře nebo vytvořením nové operátorem `new`,
2. aktualizací dat v entitě za pomocí přiřazení,
3. odesláním změn do databáze voláním Doctrine objektu `EntityManager`.

Aktualizace schématu databáze je prováděna *migrací* (knihovna doctrine/migrations). Na základě entit jsou voláním Doctrine získány SQL dotazy, které jsou poté do těchto migrací vkládány. Voláním administrativního příkazu jsou migrace aplikovány na databázi.

Užití migrací bylo zvoleno pro umožnění aktualizace existující (produkční) databáze beze ztráty dat, jelikož kromě DDL dotazů (dotazů pro změnu struktury databáze) umožňují i přidání další logiky pro zpracování dat a jejich úpravu pro nové schéma databáze.

Databázový diagram aplikace je v příloze C.

5.4.3 Třídy doménových objektů

Všechny doménové (modelové) objekty se nacházejí ve jmenném prostoru `NPS\Model` a implementují společné rozhraní `IDomainObject` obsahující metody pro získání databázové entity a identifikátoru daného objektu. Každá instance doménového objektu má jako atribut jednu instanci databázové entity a disponuje metodami pro její úpravu a validaci.

V aplikaci existují následující třídy doménových objektů:

Doménový objekt	Databázová reprezentace	Popis
`User`	`UserEntity`	uživatel aplikace
`Consultation`	`ConsultationEntity`	konzultace s expertním systémem
`KnowledgeBase`	`KnowledgeBaseEntity`	báze znalostí
`File`	`FileEntity`	multimediální soubor
`ErrorReportingQuery`	`ErrorReportingQueryEntity`	nahlášená chyba (zpětná vazba uživatele)
`UserQuestionQuery`	`UserQuestionQueryEntity`	dotaz (zpětná vazba uživatele)
`PasswordResetToken`	`PasswordResetTokenEntity`	token pro obnovu hesla
`UserActivationToken`	`UserActivationTokenEntity`	token pro aktivaci účtu

Tab. 5.1: Třídy doménových objektů

5.4.4 Tokeny, autentizace a autorizace

Hesla jsou ukládány v databázi do tabulky `user` (uživatel). Při ukládání hesla je k němu přidán tzv. *salt* (přídavný řetězec znesnadňující odhalení hesla), výsledný řetězec je zakódován jednosměrnou šifrou (hashovacím algoritmem) *bcrypt* a uložen do databáze. Při ověřování přihlašovacích údajů je získané heslo stejným způsobem zašifrováno a dochází k porovnání dvou šifer. Pro heslo je vyžadována minimální délka 8 znaků a obsah písmen i číslic.

Registrace uživatele nebo zažádání o obnovu hesla vedou na vytvoření objektu typu *Token* obsahujícím náhodný řetězec šifrovaný algoritmem *sha512*. Vytvořený token je posílán uživateli na email jako součást odkazu pro dokončení zvolené akce. Je použitelný pouze jednorázově a má omezenou platnost na dobu 30 minut.

Autentizace a autorizace uživatelů je založena na předávání JWT tokenů (užita knihovna *lexik/jwt-authentication-bundle*). Jedná se o objekty ve formátu JSON zašifrované pomocí šifrovacích SSH klíčů do tvaru nečitelného řetězce. Proces autentizace uživatele probíhá následujícím způsobem:

1. Na adresu `/login` uživatel pošle své přihlašovací údaje.
2. Dojde k vyvolání události, jež provede porovnání údajů s databází.
3. V případě shody údajů je vygenerován a zašifrován JWT token.
4. Token je zaslán v odpovědi zpět společně s informacemi o uživateli a jeho rolích.
5. Uživatel při každém požadavku přidá HTTP hlavičku „Authorization: Bearer x“, kde „x“ odpovídá získanému tokenu.
6. V backendu dojde k dešifrování a ověření pravosti/platnosti tokenu.
7. Pokud tokenu vypršela platnost, je uživatel odpovědí vyzván k jeho obnově.
8. Pokud je token neplatný, je navrácen HTTP kód 401 (viz 3.2).

Tokeny mají omezenou dobu platnosti na 30 minut a pro zvýšení bezpečnosti obsahují zašifrovanou informaci o *user-agent* sloužícím k identifikaci uživatelova zařízení a prohlížeče. Při každém požadavku je poté kontrolována shoda hodnoty *user-agent* v tokenu a v požadavku. Předpokládá se užití HTTPS protokolu[6], aby nedošlo k odposlechu tokenu při jeho předávání.

Každý uživatel aplikace disponuje právě jednou rolí. Na základě role je uživateli omezován přístup ke zdrojům v aplikaci a při požadavku na kteroukoliv akci je v `controller` kontrolováno, zda uživatel disponuje rolí nutnou pro její provedení. Role jsou uloženy v databázi formou řetězce.

[6]Na studentském serveru nebylo doposud implementováno.

Definované role jsou (vzestupně dle hierarchického postavení):

Role	Význam
Anonymní uživatel	výchozí stav pro všechny návštěvníky
ROLE_USER	standardní uživatel s právem k vedení konzultací na produkčních bázích znalostí
ROLE_TESTER	oproti standardnímu uživateli má právo užívat i báze označené pro testování
ROLE_ENGINEER	oproti testerovi má právo vést konzultace i na bázích ve stavu vývoje
ROLE_MANAGER	má právo na přidělování přístupů uživatelů k bázím znalostí a určování jejich rolí
ROLE_ADMIN	přebírá práva všech předchozích rolí, přiřazuje manažerské role

Tab. 5.2: Role uživatelů

5.4.5 Práce s bázemi znalostí

Báze znalostí jsou jediným doménovým objektem, který je fyzicky uložen na dvou místech. Mají svou databázovou reprezentaci uchovávající informace o autoru, verzi, stavu nebo číselných identifikátorech a poté soubor se samotnou definicí báze znalostí ve formátu *.nps* (viz kapitola 2.4).

Pro soubory s definicemi bází je vyhrazen zvláštní adresář v kořenové složce webu, ve kterém lze najít podadresáře rozdělené dle jména báze znalostí, ze kterého byly odstraněny všechny netisknutelné znaky a znaky mimo ASCII tabulku. V každém tomto podadresáři se poté nacházejí soubory s definicemi bází pojmenováné podle data jejího vytvoření. Výsledná cesta k bázi vypadá například následovně:

```
/bases/Brambory2000/20180802.nps
```

Jedna báze znalostí může mít N verzí (revizí) identifikovaných datem svého vytvoření. Tyto verze nejsou pouze atributem určité báze, ale samostatnými záznamy provázanými skrze číselný identifikátor `bindId`. Každá verze báze má svůj vlastní soubor s definicí. Toto uspořádání, jež je ilustrováno v následující tabulce, se během vývoje ukázalo býti výhodným. Převažují totiž operace nad jednou konkrétní verzí báze, než nade všemi verzemi dané báze.

ID	Název	Verze	Bind_ID
1	Brambory	2019-02-02	1
2	Brambory	2019-03-01	1
3	Notebooky	2019-01-29	2

Tab. 5.3: Databázová reprezentace bází znalostí

Pro báze znalostí byl stanoven životní cyklus, který umožňuje komplexní strategie v oblasti jejich vývoje a ladění. Jedná se o stavový automat se stavy:

`STATUS_DEV` - Odpovídá nové verzi báze znalostí, se kterou pracuje pouze znalostní inženýr.

`STATUS_TESTING` - Odpovídá alfa či beta verzi nové báze znalostí, přístupná znalostním inženýrům a pověřeným testerům.

`STATUS_PUBLIC` - Slouží pro produkční verze báze, které jsou zveřejněny a zpřístupněny uživatelům. Zahrnuje i starší verze dané báze.

`STATUS_DEMO` - Báze znalostí určená pro režim ukázky (dema), zobrazuje se i nepřihlášeným uživatelům.

`STATUS_DISABLED` - Identifikuje zastaralé či pokusné báze znalostí, které by se neměly nikomu zobrazovat.

Každý uživatel má prostřednictvím entity `AccessGrantEntity` přiřazené přístupy k jednotlivým bázím znalostí dle parametru `Bind_ID` (uživatel má vždy přístup ke všem verzím dané báze, avšak s ohledem na její status). Počítá se s tím, že během praktického používání systému bude mít uživatel přiděleny pouze ty báze, které potřebuje (patřící dané společnosti či skupině lidí).

Výběr dostupných bází probíhá ve službě `UserKnowledgeAccessGuard`, a to na dvou úrovních:

1. Ze všech bází se odfiltrují ty, ke kterým nemá uživatel přiřazené přístupy[7].
2. Ze zbylých bází jsou odfiltrovány báze dle kombinace stavu (výše) a role uživatele (tabulka 5.2).

[7]Filtrování bází dle přiřazených přístupů je na studentském serveru z výukových důvodů vypnuto.

Pro podporu ladění bází znalostí bylo potřeba umožnit rychlé nahrávání nových verzí bází znalostí. Za tímto účelem je v systému zabudován skript, který přečte obsah vybraného adresáře a nahraje v něm umístěné báze znalostí i multimediální soubory do systému a databáze.

Báze znalostí musejí být umístěny v podsložkách, kdy z každé podsložky je nahráván maximálně jeden soubor s bází znalostí. Nově nalezené soubory jsou přidány, existující aktualizovány. Pokud má soubor s bází znalostí jméno odpovídající platnému datu (ve formátu `yyyymmdd`), bude toto datum použito pro identifikaci verze přidané či aktualizované báze.

Adresář musí splňovat strukturu:

```
dir ..... synchronizační adresář
├── brambory ..... podsložka pro bázi brambory
│   ├── 20190102.nps ..... soubor s bází znalostí ve formátu .nps
│   └── obr.jpg ..... multimediální soubor, na který se báze odkazuje
└── notebooky ..... podsložka pro bázi notebooky
    └── 20190202.n32 .. soubor s bází znalostí ve formátu .n32, bude převeden na .nps
```

Výpis 5.4.1: Strom adresáře pro synchronizaci bází znalostí

Použití synchronizace s adresářem však není nejvhodnější řešení v produkčním prostředí. Vzhledem k povaze bází znalostí je vhodné je verzovat. K editaci bází se předpokládá užití externího textového editoru (ovládaného mimo server), jenž disponuje znalostí syntaxe jazyka XML. Proto bylo synchronizace s adresářem využito k implementaci synchronizace s git repozitářem (*npsknowledge*, kapitola 5.2).

Struktura synchronizačního adresáře zůstává zachována, adresářový strom je rozšířen na kořenové úrovni o podsložky indikující stav báze k synchronizaci.

```
repo ..... repozitář
├── prod ..... adresář pro produkční báze
│   └── brambory ..... podsložka pro bázi brambory
└── test ..... adresář pro testovací báze
    └── notebooky ..... podsložka pro bázi notebooky
```

Výpis 5.4.2: Strom repozitáře pro synchronizaci bází znalostí

5.4.6 Komunikace s programem NPSCore

Třídy aplikace pracují se společným rozhraním `IExpertSystemBridge`, jež deklaruje základní funkce expertního systému. Metody v rozhraní `IExpertSystemBridge` odpovídají rozhraní NPSCore.

Tyto metody jsou:

`startConsultation` - Zahájí novou konzultaci.
`loadConsultationState` - Načte probíhající konzultaci.
`saveConsultationState` - Uloží probíhající konzultaci.
`setLanguage` - Nastaví jazyk konzultace.
`getLanguage` - Získá jazyk konzultace.
`getSupportedLanguages` - Získá jazyky podporované danou bází znalostí.
`getQuestion` - Získá aktuální otázku.
`setAnswer` - Nastaví odpověď na aktuální otázku.
`forward` - Přejde na následující otázku, pokud již současná byla zodpovězena.
`back` - Přejde zpět na minulou otázku, pokud se nejedná o tu první.
`getHypothesis` - Získá hypotézy konzultace.
`endConsultation` - Ukončí aktuální konzultaci.
`isEnded` - Vrací logickou hodnotu, zda-li probíhající konzultace již byla ukončena.

Zavedením společného rozhraní je umožněno budoucí užití jiných expertních systémů, než pouze NPS. Zatím jedinou jeho implementací je však třída `NPSBridge` využívající Symfony komponenty *Process* pro komunikaci s asynchronně běžícím procesem NPSCore.

Zpracování příkazu pro expertní systém probíhá následovně:

1. vytvoří se odpovídající XML element a jeho řetězcová reprezentace,
2. pokud proces NPSCore neexistuje nebo neběží, vytvoří se nový,
3. do standardního vstupu procesu se zapíše řetězcový příkaz,
4. ve smyčce se čeká na zahájení výstupu z procesu (standardní výstup nebo standardní chybový výstup),
5. ve smyčce se čeká na ukončení výstupu z procesu (reakce na sestupnou hranu),
6. získaný standardní výstup je převeden na UTF-8 kódování,
7. získaný řetězec je převeden do XML elementu,
8. zkontroluje se, zda NPS nevyvolal interní výjimku (`ok` atribut elementu s odpovědí),

9. XML element je navrácen pro další použití v aplikaci.

Při komunikaci jsou kontrolovány následující chyby:

1. ukončení procesu během čtení výstupu (vyjma příkazu „exit“),
2. vytvoření standardního chybového výstupu (stderr),
3. selhání XML parseru při tvorbě XML elementu,
4. vznik interní výjimky NPSCore, získání chybové odpovědi.

Výše uvedený proces je znázorněn v diagramu, příloha C. Třída `NPSBridge` vytvoří proces při prvním pokusu o komunikaci a následně si jej drží až do zavolání svého destruktoru. Na úrovni PHP jsou implementovány obsluhy událostí, které zaručí ukončení procesu při vzniku výjimek. Při vývoji třídy byl kladen důraz na optimalizaci (snížení) počtu nutných volání expertního systému na nezbytné minimum. Třída má atributy, díky kterým si pamatuje poslední volání daných příkazů, a pokud to není nutné, pak volání vynechá.

Veškerá komunikace je průběžně zaznamenávána (logována) knihovnou `Monolog`. Každý den dojde k vytvoření nového log souboru.

Za účelem zjednodušení práce s XML elementy byla vytvořena vlastní stromová struktura `TraversableXMLElement`, jež je plně immutabilní (neměnná = každá změna na struktuře stromu vyústí ve vytvoření nové instance, nikoliv k úpravě existujícího) a umožňuje definovat XML příkazy řetězením metod, jako například:

Výpis 5.4.3: Užití třídy `TraversableXMLElement`

```
XML::create('cmd', ['type' => 'convert_base'])
->addChild(
    XML::create('src', ['type' => 'file'], $source)
)
->addChild(
    XML::create('dst', ['type' => 'file'], $destination)
)
```

5.4.7 Podpora multimédií

Aplikace podporuje dva formáty zápisu otázek a hypotéz, které mohly být zavedeny díky implementaci obecného bloku `content` v NPSCore (viz kapitola 2.4).

Podporované formáty jsou:

Jednoduchý (basic) - Element `content` má pouze textový obsah, který bude vrácen jako titulek.

Rozšířený (enhanced) - Element `content` obsahuje subelementy s titulkem, popisem, obrázky, videi či zvukovými soubory.

Příklad podoby rozšířeného formátu:

Výpis 5.4.4: Příklad podoby rozšířeného formátu pro zápis otázek a hypotéz

```
<content culture="cs-CZ">
    <title>Primární určení notebooku?</title>
    <description>Vyberte si, nač budete...</description>
    <image>https://x.y/z.png</image>
    <image>obr.jpg</image>
    <sound>snd.wav</sound>
    <video>https://www.youtube.com/embed/aabbcc</video>
</content>
```

Podpora multimédií se opírá o obecné rozhraní `IXMLContentParser`, které je obdobného účelu, jako zmíněné `IExpertSystemBridge`. Disponuje následujícími metodami:

`parsetText` - Pomocná metoda pro úprava textových řetězců.
`parse` - Přečte argumentem zaslaný content element a zpracuje získaný obsah.
`getTitle` - Vrátí titulek z elementu.
`getDescription` - Vrátí popisek z elementu.
`getImages` - Vrátí pole adres (cest) k obrázkům.
`getVideos` - Vrátí pole adres (cest) k videím.
`getSounds` - Vrátí pole adres (cest) ke zvukům.

Rozhraní tedy počítá se zavoláním metody `parse` nad XML elementem získaným z expertního systému a následným získáváním informací, jež si uchová do dalšího zavolání `parse`. Implementací tohoto rozhraní je třída `NPSParser` pro systém NPS.

Adresy multimédií mohou být zapsány jako externí url nebo formou interních jmen souborů. Multimediální obsah je při nahrávání na server vždy opatřen tagem (značkou) indikujícím `Bind_ID` (identifikátor) odpovídající báze znalostí, ke které patří. Kombinace tohoto tagu a názvu souboru (včetně přípony) poté slouží jako jedinečný identifikátor daného souboru, přes který je možné se na soubor odkázat z báze znalostí. Při procesu čtení obsahu jsou pak vnitřní názvy transformovány na url adresy souborů (interní, umístěné na stejném serveru). Podporu multimediálních souborů zajišťují doménové objekty typu `File`.

5.4.8 Administrativní příkazy

Symfony obsahuje komponentu *Console* umožňující definici a volání php skriptů (příkazů) kompatibilních s frameworkem z prostředí příkazové řádky. Tyto příkazy mají řadu výhod oproti standardním PHP skriptům, z nichž největší je možnost pracovat s dependency injection službami aplikace.

Typickým využitím příkazů je standardizace častých úkonů nad databází a filesystémem. Distribuce symfony má v sobě již zahrnuty administrativní příkazy frameworku a knihovny doctrine. Umožňuje však i tvorbu vlastních příkazů. Příkazy vracejí návratový kód. Kód `0` odpovídá úspěšnému provedení, nenulový kód odpovídá chybě.

Pro vykonání příkazu je třeba zavolat z příkazové řádky příkaz:

```
php bin/console kategorie:jmeno_prikazu
```

Doposud byly implementovány vlastní příkazy:

`base:add` - Na základě předané cesty k souboru přidá do systému novou bázi znalostí či vytvoří její novou verzi za pomocí služby `KnowledgeBaseFactory`.

`base:convert` - Využije funkci z NPSCore a překonvertuje předaný soubor ve formátu *.n32* do novějšího *.nps*.

`base:dbinvalidate` - Aktualizuje metadata bází znalostí uložená v databázi.

`base:disable` - Změní stav požadované báze znalostí na `STATUS_DISABLED`.

`base:git` - Synchronizuje báze znalostí v systému s git repozitářem (viz kapitola 5.4.5).

`base:sync` - Synchronizuje báze znalostí v systému s adresářem (viz kapitola 5.4.5).

`file:upload` - Nahraje do aplikace multimediální soubor přes službu `FileFactory`. Soubory jsou při nahrávání kopírovány do podadresářů dle aktuálního měsíce

a roku, aby nedošlo k přetečení limitu filesystému na počet souborů v jednom adresáři.

`user:add` - Přidá nového uživatele do aplikace pomocí `UserFactory`. Provede jeho registraci a aktivaci účtu.

`user:reset` - Nastaví uživateli nové heslo.

5.5 Klientská část aplikace

Klientská část aplikace prezentuje data uživateli. Je založena na javascriptovém frameworku *Vue.js*[8], ve verzi 2, a provedena jako jednostránková (SPA) aplikace. Vzhled je definován ve formátu SCSS (preprocesor *Sass*) a opírá se o hotové řešení *Twitter Bootstrap 4* i sadu ikon *Font Awesome*.

Vue.js je založen na „shadow DOM“ a vytváří si vlastní vnitřní reprezentaci elementů na stránce. Jeho podstatnými vlastnostmi jsou *stromový systém komponent* a *reaktivita*, kdy se zobrazovaná data automaticky mění na základě dat modelu. Komponenty frameworku jsou kompatibilní se standardem *Web components*[9].

Frontend je distribuován formou balíčků vytvářených nástrojem *Webpack Encore*, a to ve dvou konfiguracích, developerské a produkční. Tyto balíčky jsou pro usnadnění nasazení na server verzovány a pro jejich nasazení stačí je v nezměněné podobě překopírovat (nebo naklonovat z Git) do cílového adresáře.

Balíček s aplikací obsahuje:

HTML dokument - Výchozí soubor přes který se bude k aplikaci přistupovat. V části s hlavičkou má meta tagy, použité fonty, ikony, znakové sady a odkazy na ostatní soubory z balíčku. V jeho těle je jen jediný prázdný element, na kterém se spouští javascriptová aplikace.

Soubor kaskádových stylů - Definuje vzhled stránky. Zdrojové soubory ve formátu SCSS (Sass) jsou do tohoto jednoho souboru zkompilovány a následně také minifikovány pro snížení přenášeného datového objemu.

Soubor s aplikačním skriptem - Obsahuje kompletní kód aplikace v jazyce JavaScript včetně užitých pluginů. Opět se jedná o velké množství souborů, které jsou sloučeny dohromady, minifikovány a znečitelněny.

Soubor manifest.json - Zajišťuje správné nalezení souborů aplikace pro určité platformy.

Adresář s ikonami a použitými obrázky - Obsahuje grafiku webu.

[8] `https://vuejs.org/`

[9] `https://developer.mozilla.org/en-US/docs/Web/Web_Components`

Užití nástroje *Webpack Encore* je podmíněno instalací *NodeJS* a přidáním řady podpůrných pluginů skrze správce závislostí *yarn* (lze použít i *npm*). Nejdůležitější z použitých pluginů jsou:

babel - Převádí konstrukce javascriptu nového standardu (použito ES2017) do konstrukcí kompatibilních s většinou prohlížečů.
vue-loader - Umožňuje podporu Vue frameworku.
sass-loader - Umožňuje podporu SCSS stylů.
jQuery - Je vyžadován staršími knihovnami.
html-webpack-plugin - Vytváří výchozí HTML dokument balíčku.
inline-environment-variables-webpack-plugin - Vkládá do aplikace proměnné indikující cílovou konfiguraci (developerská/produkční).
webpack-encoding-plugin - Zajišťuje správné kódování souborů (*ISO-8859-2*).

Po získání HTML dokumentu uživatelem (načtením stránky v prohlížeči) se na elementu v něm vloženém spustí skript frameworku Vue, jenž vytvoří stránku s požadovaným obsahem. Pokud je to nutné, dojde také ke komunikaci se serverem pro získání dat.

Adresářová struktura klientské aplikace je přiložena v příloze D.

5.5.1 Vnitřní struktura klientské aplikace

Aplikace se skládá ze stromu komponent. Každá komponenta je umístěna v samostatném podadresáři a je definována čtyřmi soubory:

- souborem `index.vue` sloužícím jako deklarace komponenty,
- HTML šablonou `template.html` obsahující standardní HTML tagy a vkládající další komponenty,
- programovou částí komponenty `script.js`,
- styly v jazyce SCSS (soubor `style.scss`).

Komponenty jsou 3 typů:

Generické - Znovupoužitelné komponenty používané ve všech částech frontendu. Příkladem je ikona, rozbalovač nebo načítací kolečko.
Specializované - Komponenty znovupoužitelné v omezeném rozsahu (na určitých stránkách). Příkladem je otázka konzultace, položky detailu hypotéz.

Stránkové - Jsou použity pro reprezentaci logických stránek. Příkladem je výběr znalostních bází či historie konzultací.

Výchozím bodem aplikace je soubor `main.js` vytvářející instanci frameworku Vue a načítající do něj aplikační komponentu `App`. Ta dále vytváří komponenty `AppHeader`, `AppContainer`, `AppFooter` reprezentující hlavičku, tělo a patičku stránky.

Jak je uvedeno v kapitole 4.2, frontend je rozdělen na logické stránky. Přepínání stránek zabezpečuje knihovna *vue-router*, která v souboru `router/index.js` mapuje řetězcové cesty na stránkové komponenty, jež jsou poté skrze prvek `router-view` vkládány do `AppContainer`. Navigace na logickou stránku je indikována připojením zvolené cesty za znak `#` v url adrese prohlížeče a změnou titulku stránky (karty prohlížeče).

Příklad modifikace cesty při přechodu z úvodní strany na přihlášení:

Výpis 5.5.1: Úprava adresy prohlížeče při navigaci na jinou logickou stránku

```
stud.feec.vutbr.cz/~xkorin12/nps/#/
stud.feec.vutbr.cz/~xkorin12/nps/#/account/signin
```

Stránkové komponenty s komplexní logikou (registrace uživatele, historie konzultací) jsou realizovány jako stavové automaty. Změna stavu se navenek projeví změnou zobrazeného obsahu, stavy tedy reprezentují „podstránky“.

Framework neumožňuje vertikální kompozici (dědičnost) komponent, pouze kompozici horizontální za pomocí tzv. *mixinů*. Vložením mixinu do určité komponenty jsou její data, metody a události rozšířeny o ty, jež jsou mixinem definovány. Většina realizovaných komponent alespoň jeden mixin využívá.

V aplikaci jsou definovány mixiny:

acl - Definuje metody pro autorizaci uživatele na úrovni komponent.
backlink - Umí zkonstruovat cestu k aktuální logické stránce pro odeslání přes API.
form - Vkládá formuláře do komponent.
scroll - Reaguje na vertikální navigaci uživatele po stránce.
spinner - Umožňuje zobrazení načítacího kolečka při získávání dat ze serveru.
title - Mění titulek prohlížeče při navigaci mezi logickými stránkami.

5.5.2 Komunikace se serverem

Komunikace se serverem probíhá asynchronně za pomocí AJAX požadavků s využitím *Fetch API*[10]. Nad touto API byla implementována vlastní knihovna `ajax.js` standardizující konstrukci požadavků a zpracovávající chybové výstupy. Knihovna podporuje HTTP metody GET a POST.

Implementovaná knihovna zpracovává HTTP kódy:

200 - Úspěšný požadavek vyústí v zavolání požadované funkce.
500 - Chyba serveru vyvolá výjimku „serverError“, která musí být ošetřena na úrovni komponent.
400 - Špatné parametry požadavku jsou propagovány ke komponentám s výchozí chybovou hláškou. Buďto uživatel zadal nesprávné hodnoty anebo je chyba v konstrukci požadavků na straně frontendu.
401 - Uživatel nebyl serverem autorizován. Zpravidla se jedná o expirovaný autorizační token. Informace je předána `auth` modulu, který na ni reaguje přesměrováním uživatele na stránku s přihlášením.
jiné - Jiný HTTP kód vede k propagaci výjimky a jejímu zobrazení v konzoli prohlížeče.

Datový model aplikace je ukládán v tzv. *store* dodávaném knihovnou *vue-store*. Ten obsahuje nejen data získaná ze serveru, ale také data sdílená mezi komponentami aplikace. Jeho významnou vlastností je zapouzdření dat. Data uložená ve store jsou jen pro čtení a k jejich úpravě se používají funkce zvané *mutace* volané asynchronními *akcemi*. Asynchronní akce jsou postaveny na využití javascriptových *Promise* (příslibů). Data ve store mají platnost pouze po dobu uživatelovy relace, data s delší časovou platností jsou proto ukládána i v perzistentní paměti prohlížeče (`localStorage`).

Získávání dat ze serveru probíhá ve 4 krocích:

1. uživateli je zobrazen indikátor načítání (načítací kolečko),
2. komponenta zavolá akci store, jež vyvolá HTTP požadavek a vrátí příslib jeho splnění,
3. po úspěšném dokončení požadavku dojde k aktualizaci dat ve store prostřednictvím odpovídajících mutací,
4. výstup je propagován zpět do komponenty, indikátor načítání je odstraněn.

[10] `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API`

Pro oddělení nezávislých částí aplikace je store proveden modulárně a rozdělen na moduly `auth`, `consultation`, `knowledge`, `user` a `locale`.

Důležitou úlohou nakládání s daty ze serveru bylo zajištění jejich validity, čili platnosti pro daného uživatele a daný čas. Data uložená v klientovi musí být vždy v souladu s těmi na serveru. Nově přihlášený uživatel nesmí mít přístup k datům uživatele starého.

Validita dat je ošetřena v následujících mezních případech:

- spuštění aplikace nebo obnova stránky,
- změna uživatele,
- expirace autorizačního tokenu,
- změna jazyka,
- vstup do režimu ukázky.

Operace prováděné v mezních případech jsou zprostředkovány komponentou `App` a probíhají prostřednictvím tzv. *lifecycle callbacků* (reakcí na události při vzniku či změně stavu komponenty) nebo *watcherů* (reagují na změny hodnot uložených ve store). Stará data jsou zahozena a následně nahrazena novými.

5.5.3 Jazyky

Lokalizace textů je zprostředkována knihovnou *vuejs-localization*. Definice překladů je v adresářích `lang/cs` a `lang/en` a je rozdělena do několika souborů podle účelu (app, consultation, knowledge, texts). Jedná se o soubory jazyka javascript obsahující fráze uložené jako asociativní pole *klíč : hodnota*. Klíče jsou řetězcové a je k nim odkazováno z šablon jednotlivých komponent.

Příklad překladu fráze:

`<p>{{ $lang.consultation.hypothesesOrder }}</p>` (HTML šablona)
`hypothesesOrder: 'Pořadí hypotéz'` (lang/cs/consultation.js)
`hypothesesOrder: 'Hypotheses order'` (lang/en/consultation.js)

5.5.4 Podpora multimédií

Pro zobrazování multimediálního obsahu byly implementovány komponenty:

Thumbnail - Zobrazuje obrázky. Podporovány jsou všechny formáty zobrazitelné prohlížečem (`.png`, `.gif`, `.jpg`).

VideoFrame - Zobrazuje videa. Podporovány jsou odkazy na *youtube.com*, odkazy na *vimeo.com* a videa ve formátu `.mp4` (mpeg).

AudioFrame - Přehrává audio. Podporované formáty jsou `.m4a`, `.wav` a `.mp3`.

Při konzultaci jsou zmíněné komponenty součástí společného kontejneru `MediaBox`, jenž obsahuje ovládací prvky a umožňuje přepínat mezi zobrazenými multimédii.

6 Testování

Kapitola popisuje provedení automatizovaných testů.

Celé řešení bylo průběžně během vývoje ručně testováno a následně po nasazení předáno studentům, od kterých byla získána další zpětná vazba.

Pro backend aplikace byly vytvořeny automatizované testy (dle kapitoly 3.6) v jazyce PHP pracující na knihovně PHPUnit, verze 5.7.27. Testové třídy jsou umístěny v podsložce `/tests` kořenového adresáře serverové části. Běh testů probíhá při vývoji na lokálním počítači a také na cloudovém serveru *CircleCI*[1], kde jsou testy spouštěny v kontejnerizovaném prostředí po každém nahrání změn do systému Git. Všechny vytvořené automatizované testy jsou úspěšné.

Testování spočívá v provádění testovacích metod, kdy každá obsahuje sérii tzv. assertů (tvrzení). Tvrzení definuje a testuje podmínku, jakou musí jemu předaná data splňovat. Nesplněné tvrzení způsobí selhání dané testovací metody. Výsledek testů má tedy pouze logickou formu (úspěch či neúspěch).

Vzhledem k interpretované povaze jazyka PHP tyto automatizované testy testují správnost nejen sémantickou, ale i syntaktickou. Správná funkčnost jádra systému NPSCore z hlediska výsledných pravděpodobnostních hodnot hypotéz již byla otestována v rámci předcházející práce [5], v kapitole 4.2. V rámci této práce již proto testy kontrolují pouze syntaktickou korektnost výstupu NPSCore.

Na přiloženém CD je k dispozici soubor s konfigurací pro *CircleCI* a také doklad o úspěšném provedení testů (obsah CD je v příloze A).

6.1 Integrační test spojení s NPSCore

Integrační test s NPSCore implementovaný jako třída `NPSBridgeTest` testuje správnou funkci třídy `NPSBridge`, která zprostředkovává komunikaci backendu aplikace s NPSCore. Při tvorbě instance testované `NPSBridge` je použita skutečná cesta k aplikaci, všechny ostatní závislosti jsou mockovány (nahrazeny napodobeninami).

Třída testuje:

- funkčnost a synchronizaci meziprocesové komunikace,
- bezchybné provedení všech veřejných metod třídy `NPSBridge`,

[1] `https://circleci.com/`

- korektní výstupy systému pro užité testovací báze znalostí, včetně neměnnosti získáných pravděpodobnostních hodnot,
- přijatelnou časovou náročnost na provedení testů, aby mohla být zajištěna krátká odezva aplikace,
- konverzi formátu báze znalostí z *.n32* na *.nps*.

Pro tento test byly vyčleněny tři různé testovací báze znalostí umístěné v adresáři s testy:

Notebooky (.n32) - Slouží výhradně pro testování konverze formátu báze znalostí.

Notebooky (.nps) - Slouží zejména pro ověřování korektnosti výstupů, obsahuje 5 hypotéz.

Brambory (.nps) - Slouží hlavně pro testování časové náročnosti komunikace s procesem, obsahuje 102 hypotéz.

6.2 Funkcionální a jednotkové testy

Funkcionální testy testují správnou funkci řídicí vrstvy aplikace a funkčnost komponent, které tato vrstva volá. Konkrétně se jedná o testování tříd controllerů (řadičů) a administrativních příkazů. Všechny controller třídy i příkazy mají k sobě vytvořený odpovídající test.

Jelikož je třeba při testování používat databázi a dělat do ní zásahy, je pro každou testovou metodu funkcionálního testu vytvářena nová prázdná testovací databáze. Díky tomu si každý test může vytvořit nezbytné testovací objekty (tzv. *fixtury*, použitelné jako parametry pro volané akce) a provádět operace nad databází bez možnosti změny produkčních dat či vzájemného ovlivňování nezávislých testů. Testovací databáze nepoužívá připojení k MySQL serveru, ale pouze lokální soubor spravovaný knihovnou *sqlite*.

Testy controller tříd testují korektnost navrácených HTTP kódů pro volané akce a autorizaci k těmto akcím. Jsou navrženy tak, aby testovaly nejen odpovědi kladného charakteru (kód 200, případně 301, 302), ale i odpovědi záporného charakteru (kódy 400, 401, 404).

Všechny testové třídy pro controllery dědí z vlastního `ControllerTestCase` rozšiřující třídu frameworku Symfony `WebTestCase`. Hlavní součástí této třídy je testovací klient, také součást Symfony frameworku, který emuluje chování webového prohlížeče a vytváří dotazy na testované akce.

Testy administrativních příkazů testují jejich návratový kód a tím i jejich úspěšnost. Obdobně jako u funkcionálních testů controllerů, i pro příkazy byla vytvořena

vlastní testovací třída `CommandTestCase` dědící ze Symfony třídy `KernelTestCase`. Primárním účelem třídy `CommandTestCase` je umožnit práci s dependency injection službami aplikace.

Pro klíčové třídy modelu byly navrženy také jednotkové testy. Veškeré závislosti testovaných objektů byly pro tento účel mockovány (nahrazeny napodobeninami). Třídy jednotkových testů dědí z výchozí třídy `PHPUnit\Framework\TestCase` dodávané v PHPUnit a nebyla pro ně implementována žádná zvláštní logika.

7 Nasazení na fakultní server

Kapitola obsahuje požadavky aplikace a seznam kroků použitých pro její nasazení na server.

Aplikace je nasazena na fakultním studentském serveru FEST a je přístupná na webové adrese `http://www.stud.feec.vutbr.cz/~xkorin12/nps`. Práce na serveru probíhá přes vzdálené připojení *SSH* a *FTP* s využitím studentských přístupových údajů. Server splňuje nutné prerekvizity pro instalaci a běh aplikace, kterými jsou:

- HTTP server (Apache httpd),
- možnost definovat si vlastní `.htaccess` soubory,
- mailový server,
- PHP (minimálně verze 5.6),
- PHP moduly zip, PDO, pdo_mysql, pdo_sqlite, Phar, xml,
- mysql klient,
- git,
- cmake, make, g++,
- setfacl pro nastavení práv,
- zip ke snížení velikosti záloh,
- cron pro plánované úlohy,
- prostor na disku (alespoň 200MB na aplikaci a dočasné soubory).

Skripty pracující s NPS jsou umístěny v rámci domovské složky uživatele *xkorin12*, soubory aplikace v jejím podadresáři `WWW` (pro zpřístupnění přes HTTP). Aby byly získány kratší a uživatelsky přívětivější url adresy, byly skutečné cesty k serverovým i klientským částem aplikace nahrazeny symbolickým odkazem.

	Symb. odkaz	Skutečné umístění
Frontend	`~/WWW/nps`	`~/WWW/npsweb/npsfrontend`
Backend	`~/WWW/npsdata`	`~/WWW/npsweb/npsbackend`

Tab. 7.1: Umístění aplikace na serveru

Přístupové údaje a parametry, které aplikace vyžaduje, jsou nakonfigurovány v souboru `~/.profile` a kopírovány do souboru .htaccess v backendu. Pro aplikaci byla

správcem na serveru `palcat.feec.vutbr.cz` vytvořena databáze `npsweb-uamt` a stejnojmenný uživatel pro práci nad touto databází. Aplikace vyžaduje i SMTP mailový server, pro který byl užit účet uživatele *xkorin12*.

Nasazení (deploy) aplikace na server je zprostředkováno skriptem pro unixový terminál bash.

7.1 Nasazovací skript

Skript je postaven tak, aby bylo možné jej volat opakovaně. Vždy nahraje aktuální verze všech částí aplikace a zachová důležité soubory i obsah databáze. Jelikož je každý další jeho krok podmíněn bezchybným provedením toho předchozího, dojde při chybě k ukončení skriptu. Skript je k nahlédnutí na přiloženém CD (obsah v příloze A).

Kompilace a instalace jádra expertního systému NPSCore je zprostředkována samostatným skriptem, díky kterému je umožněna aktualizace jádra bez nutnosti úpravy souborů webu.

Jednotlivé kroky skriptu jsou:

1. Obsah databáze, bází znalostí, multimédií, emailů a logů je zálohován do předem připraveného adresáře.
2. Je vytvořen dočasný adresář, do kterého bude aplikace instalována.
3. Frontend je stažen ze systému git příkazem `git clone`.
4. Ze získaných souborů je vyjmut hotový produkční balíček s frontendovou aplikací. Balíček je přesunut do cílového podadresáře.
5. Backend je stažen ze systému git příkazem `git clone`.
6. V podadresáři je vytvořena adresářová struktura. Na vytvořených adresářích jsou nastaveny práva použitím příkazů *chmod* a *setfacl* tak, aby měli přístup uživatelé „xkorin12“ a uživatel „nobody“, přes kterého přistupuje PHP.
7. Zálohy bází znalostí a multimédií jsou obnoveny do nové instalace.
8. Soubor `.htaccess` v podadresáři s backendem je upraven včetně nastavení korektní url ke kořenové složce webu a proměnných prostředí (přístupových údajů pro aplikaci).
9. Parametry backendu jsou nastaveny kopií souboru `parameters.fest.yml.dist` do `parameters.yml`.
10. Instalace správce PHP balíků *composer* dle instrukcí na `www.getcomposer.org/download/`.
11. Získaný archiv s *composer* je modifikován pro umožnění běhu přes *open_basedir* restrikci nastavenou v PHP konfiguraci serveru.

12. Proběhne spuštění *composer* (dočasně řešeno i kopií připravené složky s balíky závislostí).
13. Migrace aplikují očekávané úpravy na struktuře a obsahu databáze.
14. Je vyčištěna souborová cache symfony frameworku.
15. Původní složka s aplikací je nahrazena adresářem s novou instalací.
16. Vytvoření symbolických odkazů dle tabulky 7.1.
17. Spuštění samostatného skriptu pro instalaci NPSCore.
    (a) NPSCore je stažen ze systému git příkazem `git clone`.
    (b) Program cmake vytvoří kompilační soubor Makefile.
    (c) Spuštění kompilace NPSCore pomocí programu make.
    (d) Vytvoření adresářového stromu s kompilovanými binárními soubory.

Přístup na git probíhá za pomocí SSH klíčů opatřených heslem. Před spuštěním skriptu jsou tyto klíče nahrávány do SSH agenta. Vždy je využita poslední verze větve master, bez stahování historie repozitáře (úspora datového toku).

8 Závěr

Práce se zabývala tvorbou uživatelského rozhraní pro expertní systém, které je dostupné na webové adrese `http://www.stud.feec.vutbr.cz/~xkorin12/nps`. Použité teoretické podklady jsou v kapitole 1 (teorie expertních systémů), kapitole 2 (popis použitého systému NPS) a kapitole 3 (webové technologie).

Realizované uživatelské rozhraní (kapitola 4) je založeno na moderních technologiích a splňuje požadavky vyjmenované v kapitole 4.1. Uživatel může vést konzultace s expertním systémem NPS bez ohledu na místo, zařízení a čas. Provedení ve formě webové aplikace umožňuje další rozvoj systému a jeho širší využití v praxi. Rozhraní disponuje možnostmi výběru báze znalostí, vedení konzultace, procházení výsledků proběhlých konzultací i protokolování. Správce má k dispozici administrativní sekci, kde může přiřazovat přístupy a role uživatelů systému. Noví uživatelé si mohou systém vyzkoušet prostřednictvím režimu ukázky.

Výsledná programová realizace uživatelského rozhraní (kapitola 5) má formu tří oddělených celků. Serverová část uchovává data a poskytuje metody pro jejich modifikaci, klientská část se serverem komunikuje a data prezentuje uživateli. Program jádra expertního systému NPSCore je upraven pro multiplatformní použití a přeložen na linuxovém serveru. Součástí řešení je i repozitář bází znalostí umožňující jejich efektivní vývoj, pro jehož podporu má aplikace zabudovaný životní cyklus bází znalostí i metody jejich verzování. Provedení serverové aplikace jako API umožňuje vývoj dalších klientů, například mobilní aplikace.

Testování (kapitola 6) probíhá ručně i algoritmicky. Výsledné řešení prochází aktivním testováním ze strany studentů předmětu BMPA, kteří prostřednictvím vytvořeného rozhraní ladí báze znalostí pro svůj semestrální projekt. V době psaní této práce bylo již v realizovaném rozhraní provedeno přes 1500 konzultací. Dále je testována i integrace serverové části aplikace se systémem NPS a další významné komponenty za pomocí funkcionálních či jednotkových testů. Tyto testy jsou nasazeny na cloudovém serveru CircleCI a jsou spouštěny při každé úpravě zdrojových kódů serverové části. Všechny testy procházejí bez chyb.

Aplikace je nasazena na studentský server FEST (kapitola 7) a má zde připravenou infrastrukturu pro snadnou aktualizaci na nejnovější verze jednotlivých součástí aplikace.

Své uplatnění systém NPS nalezne ve firemní sféře pro řešení reálných technických problémů a také na akademické půdě k výukovým účelům. V rámci dalších prací by bylo vhodné navrhnout a odladit nové báze znalostí, které by naplno využily potenciál realizovaného uživatelského rozhraní.

Literatura

[1] *HERBRANDSON, Joseph. Most Common Attacks Affecting Today's Websites [online].* 2014 [cit. 2018-11-26]. Dostupné z: https://blog.sucuri.net/2014/11/most-common-attacks-affecting-todays-websites.html

[2] *INTERNET ENGINEERING TASK FORCE. Hypertext Transfer Protocol Version 2 (HTTP/2) [online].* 2015 [cit. 2018-11-15]. Dostupné z: https://tools.ietf.org/html/rfc7540

[3] *JIRSÍK, Václav. Moderní prostředky v automatizaci - prezentace.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017.

[4] *KOŘÍNEK, Lukáš. Uživatelské rozhraní pro expertní systém.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/115118. Semestrální práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Václav JIRSÍK.

[5] *KRECHLER, Michal. Diagnostický expertní systém.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Václav JIRSÍK.

[6] *MAŘÍK, Vladimír, Olga ŠTĚPÁNKOVÁ, Jiří LAŽANSKÝ a kol. Umělá inteligence (2).* Praha: Academia, 1997. ISBN 80-200-0504-8.

[7] *OPEN WEB APPLICATION SECURITY PROJECT. Cross-Site Request Forgery (CSRF) [online].* 2018 [cit. 2018-12-08]. Dostupné z: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[8] *ŘEZÁČ, Jan. Web ostrý jako břitva.* 2. vydání. Brno: House of Řezáč, 2016. ISBN 978-80-270-0644-1.

[9] *WORLD WIDE WEB FOUNDATION. History of the web [online].* [cit. 2018-11-17]. Dostupné z: https://webfoundation.org/about/vision/history-of-the-web/

Seznam symbolů, veličin a zkratek

API	Application Programming Interface (aplikační programové rozhraní)
CSRF	Cross-site request forgery
CSS	Cascading style sheets
DOM	Document object model (model objektů v dokumentu)
FEI	Fakulta elektrotechniky a informatiky (předchůdce dnešních FIT a FEKT)
FEKT	Fakulta elektrotechniky a komunikačních technologií
FTP	File transfer protocol (protokol pro přenos souborů)
GDPR	General data privacy regulation (Obecná regulace ochrany osobních dat)
GUI	Graphical user interface (grafické uživatelské rozhraní)
HTML	Hypertext markup language (hypertextový značkovací jazyk)
HTTP	Hypertext transfer protocol (protokol pro přenos hypertextu)
HTTPS	Hypertext transfer protocol Secure (zabezpečený protokol pro přenos hypertextu)
IoT	Internet of things (internet věcí)
JSON	Javascript object notation (notace objektů pro jazyk javascript)
JWT	JSON Web Token (webový token ve formátu JSON)
MVC	Model - View - Controller (paradigma tvorby počítačových programů)
NoSQL	Not only SQL (nejen SQL)
OOP	Object oriented programming (objektově orientované programování)
ORM	Object-relational mapping (objektově relační mapování)
PHP	Hypertext preprocessor (preprocesor hypertextu)
SPA	Single page application (jednostránková aplikace)
SQL	Structured query language (jazyk pro strukturované dotazy)
SSH	Secure shell (zabezpečený přístup)
SSL	Secure sockets layer (zabezpečení transportní vrstvy)
TCP	Transfer control protocol (protokol pro řízení přenosu)
TLS	Transport layer security (zabezpečení transportní vrstvy)
VUT	Vysoké učení technické v Brně
WWW	World wide web
XML	Extensible Markup Language (rozšířitelný značkovací jazyk)

Seznam příloh

A Obsah přiloženého CD

Výpis A.0.1: Obsah přiloženého CD

```
CD
├── src ..... složka se zdrojovými kódy
│   ├── npsbackend ..... zdrojové kódy serverové části
│   ├── npsfrontend ..... zdrojové kódy klientské části
│   └── npscore ..... zdrojové kódy jádra expertního systému
├── docs ..... vygenerovaná programová dokumentace serverové části
├── xkorin12.pdf ..... dokument práce
├── test_vysledek.pdf ..... doklad o provedeném testu na CircleCI
├── fest.sh ..... skript pro nasazení na server
├── buildnpscore.sh ..... skript pro kompilaci jádra expertního systému
└── circleci.yml ..... konfigurace pro CircleCI
```

B Formáty bází znalostí

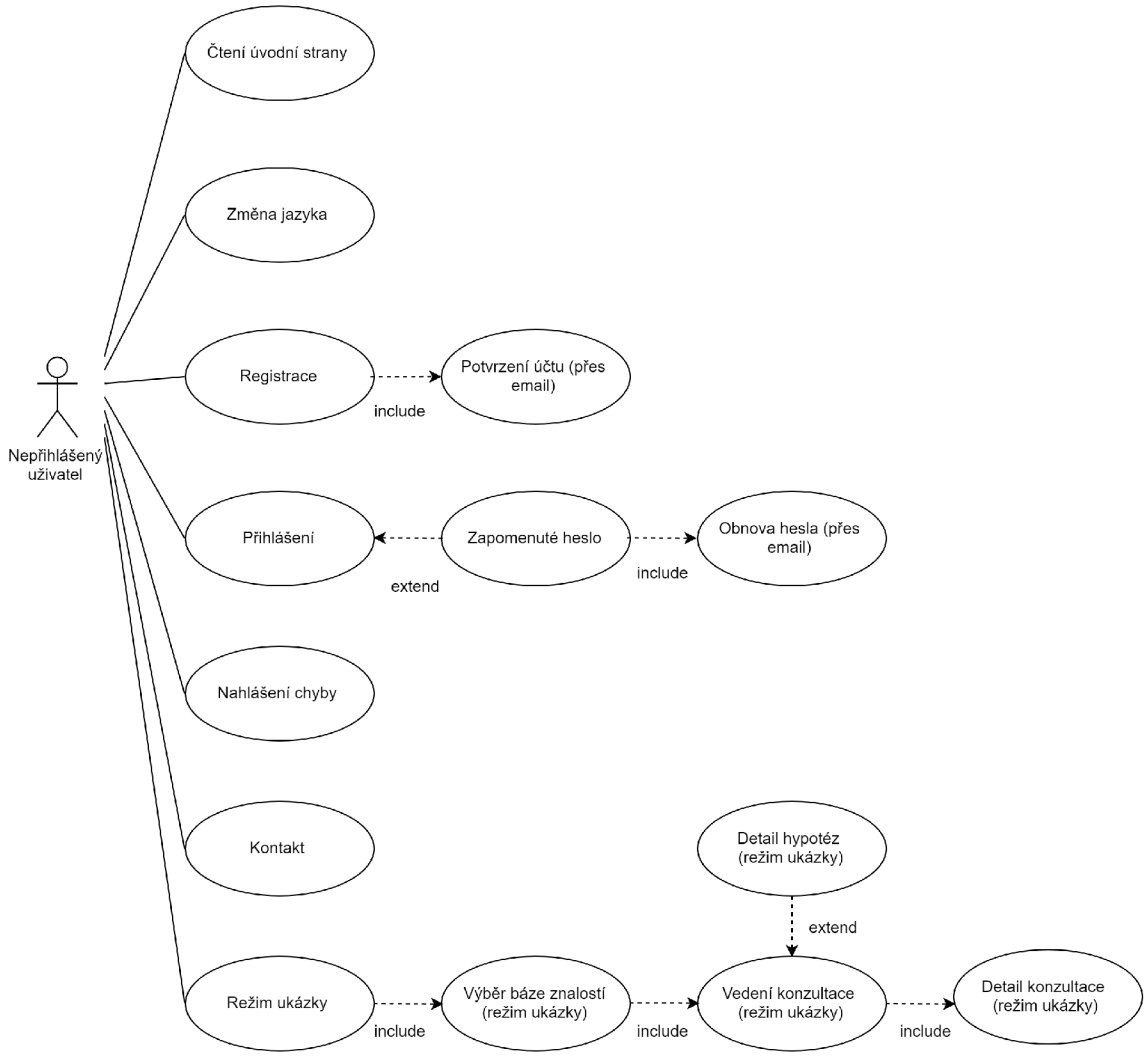
Výpis B.0.1: Příklad zápisu báze znalostí ve formátu n32

```
;.Brambory 2000
;; Databáze odladěna ve spolupráci s VÚB Havlíčkův
;; Brod. Obsahuje 102 odrůd brambor povolených v roce 2000.
.RANOBR   p=50%
.PRODUK   p=50%
.OBLAST   p=50%  D K
; Zvolte si pěstitelskou oblast, pro niž chcete provést výběr
   !
;*Ranobramborářská [1]*
;*Produkční [2]*
          RANOBR
          PRODUK
.VYNOS    p=10%  D
; Bude na honu využit vysoký výnosový potenciál odrůdy?
.NERANE   p=50%
.VELMIR   p=50%
.RANE     p=50%
.POLOR    p=50%
.POLOP    p=50%
.RANOST   p=50%  D K
```

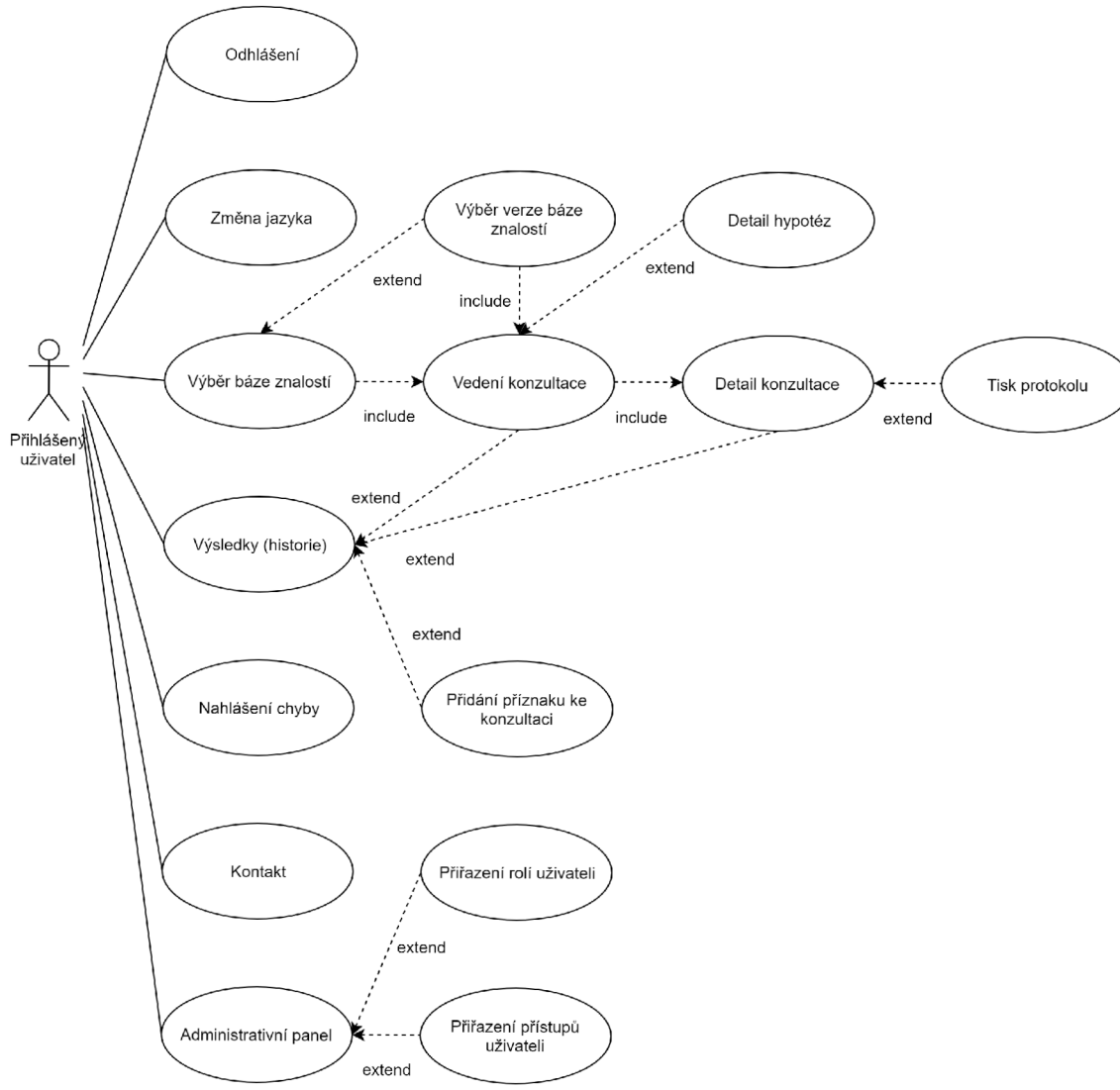
Výpis B.0.2: Příklad zápisu báze znalostí ve formátu nps

```
<?xml version="1.0" encoding="utf-8"?>
<knowledge_base version="1.0">
  <head default_answers="7">
    <name>
      <content culture="cs-CZ" type="text">Brambory 2000</
         content>
    </name>
    <cultures>
      <culture>cs-CZ</culture>
    </cultures>
    <description>
      <content culture="cs-CZ" type="text">
         Databáze odladěna ve spolupráci VÚB Havlíčkův Brod.
            Obsahuje 102 odrůd brambor povolených v roce
            2000.
      </content>
    </description>
  </head>
  <body>
    <node literal="RANOBR" probability="50.00" type="
       ancillary">
      <comment>
        <content culture="cs-CZ" type="text"/>
      </comment>
    </node>
    <node literal="PRODUK" probability="50.00" type="
       ancillary">
      <comment>
        <content culture="cs-CZ" type="text"/>
      </comment>
            <relationships>
                <relationship type="conjunction" alpha="99.00
                   ">
                    <negative literal="RANOBR"/>
                </relationship>
            </relationships>
    </node>
```

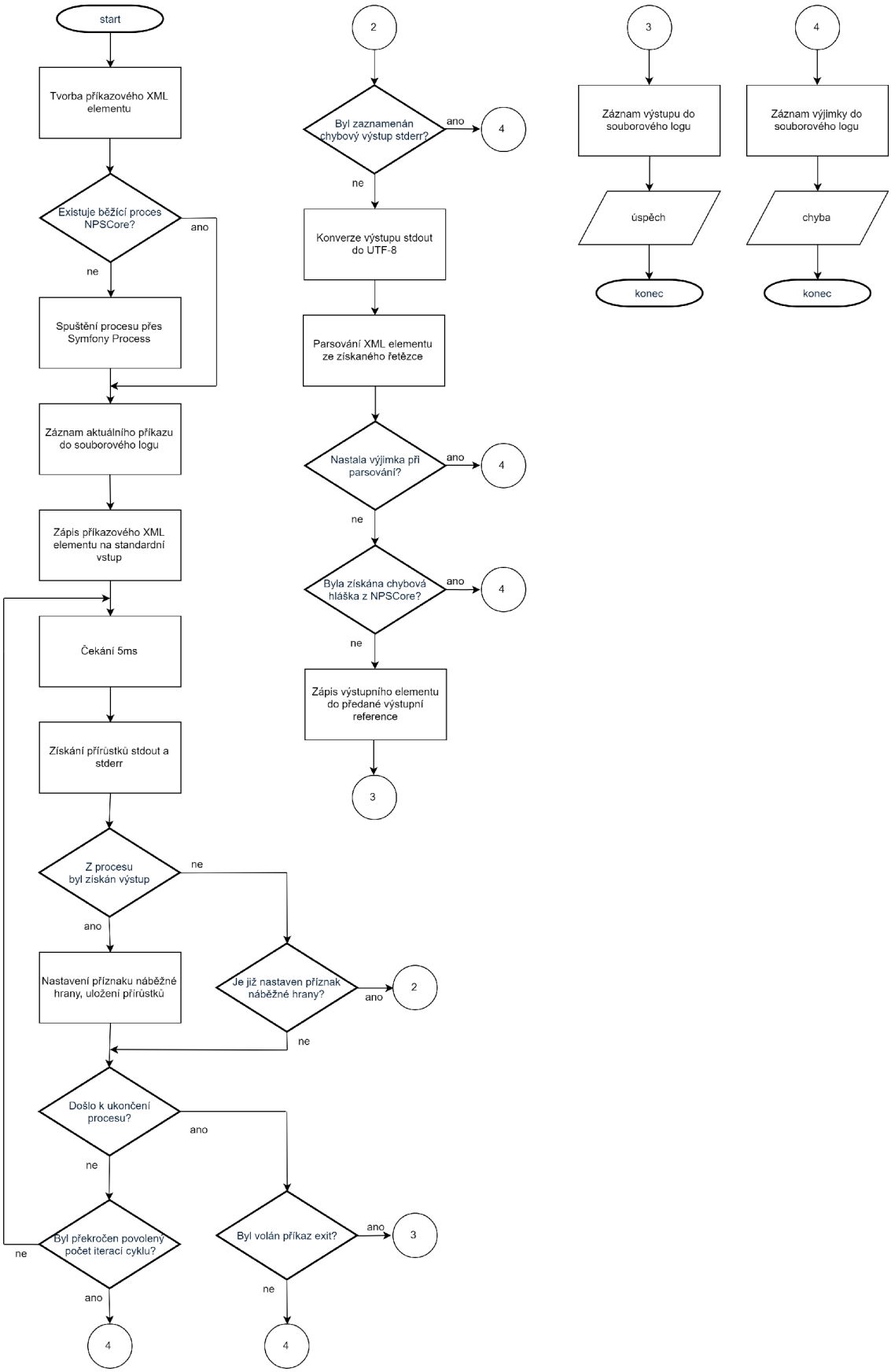
C Diagramy



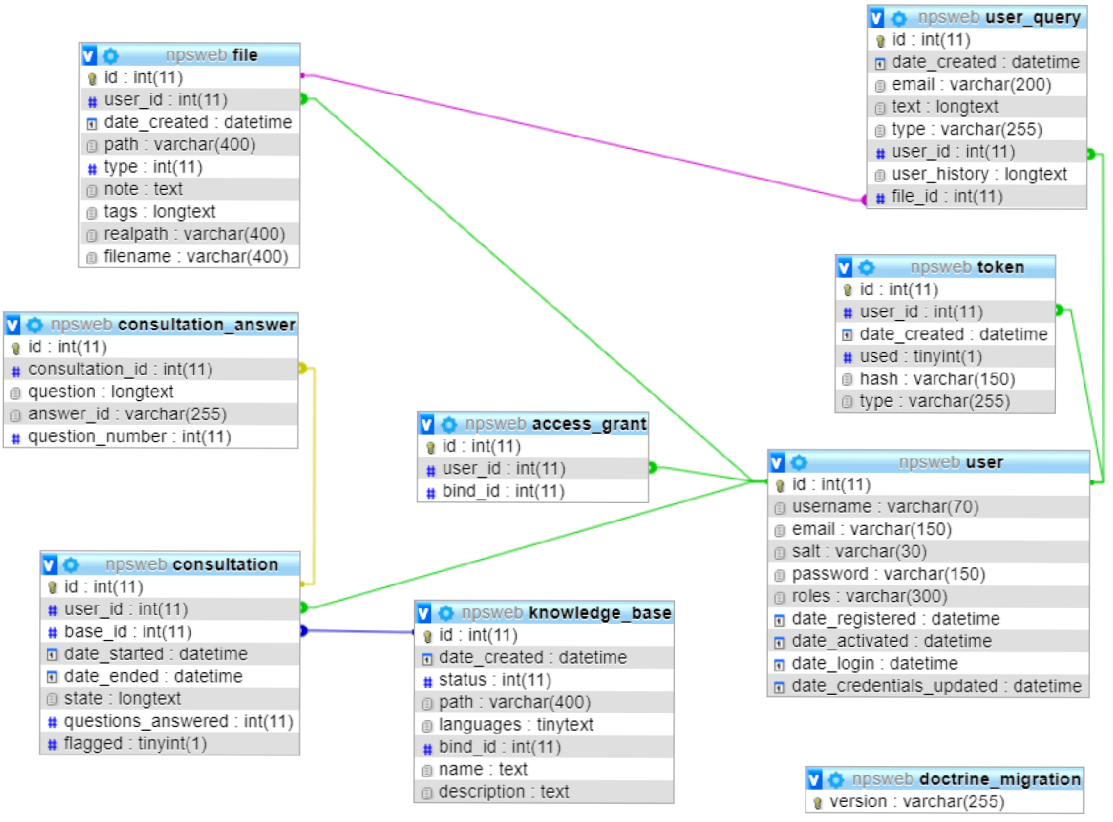
Obr. C.1: Use case diagram rozhraní (nepřihlášený uživatel)



Obr. C.2: Use case diagram rozhraní (přihlášený uživatel)



Obr. C.3: Vývojový diagram komunikace s NPSCore



Obr. C.4: Databázový diagram backend aplikace

D Adresářové stromy

Výpis D.0.1: Adresářový strom serverové aplikace

```
app
├── config ..... konfigurační soubory
└── Resources/views ..... šablony pro tvorbu emailů
bases ..... báze znalostí
bin
└── nps
    ├── unix ..... program NPSCore (unixová verze)
    └── source ..... zdrojový kód NPSCore
migrations ..... databázové migrace
src ..... zdrojový kód aplikace
└── NPS
    ├── Auth ..... pomocné třídy autentizace
    ├── Command ..... administrativní příkazy
    ├── Controller ..... controllery
    ├── DataFixtures ..... testovací data
    ├── DependencyInjection ..... obecné služby dependency injection
    ├── Entity ..... databázové entity
    ├── Etc ..... pomocné třídy
    ├── Event ..... typy událostí
    ├── EventSubscriber ..... obsluhy událostí
    ├── Exception ..... výjimky
    ├── Factory ..... továrny
    ├── Model ..... doménové objekty a jiné třídy modelu
    ├── Repository ..... repozitáře
    └── Test ..... testovací třídy
tests ..... testy
├── Knowledge
└── NPS
    ├── Command
    ├── Controller
    ├── DependencyInjection
    ├── Etc
    └── Model
uploads ..... neveřejné nahrané soubory
var ..... dočasné soubory
├── logs ..... logovací soubory
└── mail ..... odeslané emaily
vendor ..... balíčky získané přes composer
web ..... veřejně přístupná složka
└── assets ..... multimediální soubory
```

Výpis D.0.2: Adresářový strom klientské aplikace

```
src
├── component ..... vue komponenty
├── etc ..... filtry a pomocné funkce
├── img ..... grafika webu
├── lang ..... jazykové překlady
│   ├── cs ..... české překlady
│   └── en ..... anglické překlady
├── mixin ..... mixiny pro kompozici komponent
├── router ..... definice logických stránek
├── scss ..... styly platné pro celou aplikaci
└── store ..... vue store
    ├── auth ..... auth modul
    ├── knowledge ..... knowledge modul
    ├── consultation ..... consultation modul
    ├── locale ..... locale modul
    └── user ..... user modul
```

E Příklady zdrojových kódů

Výpis E.0.1: Výňatek ze třídy `NPSBridge`

```
/** Loads the knowledge base if necessary
* @param string $path
* @return bool */
private function loadBase($path)
{
    if($this->currentBasePath == $path)
    {
        return true;
    }

    if(!$this->cmdLoadBase($path))
    {
        return false;
    }

    $this->currentLanguage = null;
    $this->currentSupportedLanguages = null;

    return true;
}

/** Loads a knowledge base
* @param string $path Path to the knowledge base file
* @return bool */
private function cmdLoadBase($path)
{
    $this->currentBasePath = $path;

    return $this->command(
        XML::create('cmd', ['type' => 'load_base', 'src' => '
           file'], $path)
    );
}
```

Výpis E.0.2: Výňatek ze třídy `NPSBridgeTest`

```
/** @return NPSBridge */
private function get($language = 'cs-CZ')
{
    $logger = $this->getMockBuilder('Monolog\Logger')
        ->disableOriginalConstructor()
        ->getMock();

    $request = $this->getMockBuilder(Request::class)->
       disableOriginalConstructor()->getMock();
    $request->expects($this->any())->method('getLocale')->
       willReturn($language);

    $requestStack = $this->getMockBuilder(RequestStack::class
       )->getMock();
    $requestStack->expects($this->any())->method('
       getCurrentRequest')->willReturn($request);

    $fileRepo = $this->getMockBuilder(FileRepository::class)
       ->disableOriginalConstructor()->getMock();
    $fileRepo->expects($this->any())->method('getById')->
       willReturn(null);
    $parser = new NPSParser($fileRepo);

    return new NPSBridge($logger, $requestStack, strtoupper(
       substr(PHP_OS, 0, 3)) == 'WIN' ? 'bin/nps/win/NPSCore.
       exe' : './bin/nps/unix/npscore');
}

public function testStartConsultation()
{
    $nps = $this->get();
    $result = $nps->startConsultation(self::BASE);
    $message = $nps->popMessage();
    $this->assertTrue($result, $message);

    $this->assertInTime(200);
```

Výpis E.0.3: Výňatek ze třídy `ConsultationController`

```
/**
* @Route("/consultation/start", methods={"POST"})
*/
public function startAction(ConsultationFactory $factory,
   KnowledgeBaseRepository $baseRepository,
   UserKnowledgeAccessGuard $guard, Request $request)
{
    if(!$baseId = $request->get('baseId'))
    {
        throw new \NPS\Exception\MissingArgumentException('
           baseId');
    }

    if(!$base = $baseRepository->getById($baseId))
    {
        throw new \NPS\Exception\InvalidArgumentException('
           baseId');
    }

    if(!$guard->hasAccess($base, $this->getUser()))
    {
        throw new \NPS\Exception\InvalidArgumentException('
           baseId');
    }

    $consultation = $factory->create($base, $this->getUser())
       ;
    $this->flush();

    return $this->response([
        'consultation' => $this->createConsultationPayload($
           consultation)
    ]);
}
```

Výpis E.0.4: Výňatek z komponenty `ConsultationSession`

```
import Spinner from '../../mixin/spinner';
import Title from '../../mixin/title';
import Acl from '../../mixin/acl';
import ConsultationQuestion from '../ConsultationQuestion/
   index.vue';

export default
{
    name: 'ConsultationSession',

    mixins: [Spinner, Title, Acl],

    components:
    {
        'consultation-question': ConsultationQuestion
    },

    data: function()
    {
        return {
            backStack: 0,
            windowWidth: 0,
            consultationQuestionHeader: null
        };
    },

    computed:
    {
        consultation: function() { return this.$store.state.
           consultation.active },
        supported: function() { return this.$store.state.
           consultation.languages },

        hypothesis: function()
        {
            // Just peak the top x hypothesis
            const x = 8;// this.windowWidth > 768 ? 8 : 3;
```