FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Webová aplikace pro správu robotů - frontend
Student:	Erik Zatloukal
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Navrhněte a realizujte část webové aplikace pro správu robotů, která je určena pro laboratoř inteligentních vestavných systémů. Zaměřte se zejména na frontend, který bude prezentovat informace o robotech uložených v databázi, dovolí jejich editaci a další funkcionality spojené se správou robotů. V návrhu dbejte na grafickou stránku, funkčnost a snadnou ovladatelnost. Dbejte na kompatibilitu s často užívanými prohlížeči Chrom, FireFox, Internet Explorer a Edge, případně jejich mobilními verzemi. Na základě rešerše navrhněte vhodné technologie s ohledem na nutnost autorizovaného přístupu k datům, autentizaci uživatelů a snadné napojení na backend webové aplikace. Rozsah práce upřesněte po dohodě s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 21. listopadu 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webová aplikace pro správu robotů - front-end

Erik Zatloukal

Katedra softwarového inženýrství
Vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

14. května 2018

Poděkování

Tímto bych chtěl poděkovat Ing. Miroslavu Skrbkovi, Ph.D. za vedení mé bakalářské práce. Dále bych také chtěl poděkovat prarodičům, rodičům a přátelům za podporu a pochopení během tvorby práce a během celého studia.

Bakalářská práce byla spolufinancována Evropskou unií z Operačního programu Výzkum, vývoj a vzdělávání.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla. Souhlasím s tím, aby Dílo bylo veřejně šířeno pod některou z licencí Creative Commons 4.0 ve variantě BY a BY-SA nebo GPLv3.

V Praze dne 14. května 2018 ……

Odkaz na tuto práci

Zatloukal, Erik. *Webová aplikace pro správu robotů - front-end.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato bakalářská práce je věnována analýze, návrhu a implementaci designu uživatelského webové rozhraní a jeho propojení s jádrem souběžně vyvíjeného informačního systému pro Laboratoř inteligentních vestavných systémů.

Nejprve se práce zabývá analýzou vztahu dat, procesů a požadavků na systém s designem a návrhem webového rozhraní.

Dále práce navazuje návrhem rozhraní, technologiemi a vhodnou implementací (s jádrem systému).

V implementační/realizační části práce se bere důraz na flexibilitu rozhraní a budoucí rozšiřitelnost, pro případné očekávané změny, jak z pohledu využívání celého systému, tak z pohledu designového.

Klíčová slova vývoj webové aplikace, informační systém, front-end, design uživatelského webového rozhraní, správa vestavných systémů, výuka vestavných systémů, livs FIT ČVUT, html, css

Abstract

This thesis is mostly taking look at designing (graphicly) and implementing web user interface and integration of this interface into the core (back-end) of information system for Laboratory of embedded systems which is being developped simultaniously.

Firstly, the thesis will be aiming at analysis of relations between data, processes, functional system requirements and the design layout of web interface.

As a next subject, this thesis looks at designing user interface (logical and graphical), technology and advantages that they provide for integration with system core.

In implementation/realization section, emphasis is being placed on flexibility of the interface and its future expandability/extendebility, in case of changes in system functionality or even in case of needed design changes.

Keywords web application, information system, front-end, user web-interface design, embedded systems evidence, embedded systems education, livs FIT ČVUT, html, css

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Práce s webovými stránkami, jako jsou informační a evidenční systémy, je v dnešní době prakticky pro každého studenta, respektive vyučujícího, na vysoké škole samozřejmostí. To samé platí i obráceně. Při studiu na vysoké škole je standardem a výhodou takovýmto informačním systémem disponovat, pokud systém pomůže studentům a vyučujícím se studiem, respektive výukou daného studijního oboru.

Informační a evidenční systémy usnadňují práci s daty téměř jakékoliv povahy. To znamená, že ukládání, mazání, upravování a vyhledávání potřebných dat se může odehrávat na jednom místě. Realizace a použití takového systému tak eliminuje potřebu informace dohledávat, nebo ukládat na jiných místech, a tak zvyšovat časovou náročnost studia, bez přínosu pro studenta, respektive vyučujícího.

To však platí jen za předpokladu, že výše zmíněný systém disponuje intuitivním, flexibilním a motivujícím rozhraním, pomocí něhož jej lze ovládat.

Laboratoř inteligentních vestavných systémů (Livs) na Fakultě informačních technologií (FIT) Českého vysokého učení technického v Praze (ČVUT) takovýto systém potřebuje, a proto se tato práce zabývá návrhem a tvorbou této části projektu (uživatelské rozhraní, respektive front-end).

Jak bylo již uvedeno, systém je určen, jak pro studenty, tak pro vyučující, zúčastňující se na aktivitách v laboratoři (Livs). Více jej však pravděpodobně budou využívat právě vyučující této laboratoře. Hlavní náplní systému je evidence zařízení, projektů a informací s nimi spojenými.

Část systému je též prezentační a bude sloužit jako motivační část pro veřejnost/zájemce o studium v (Livs).

Při tvorbě rozhraní se práce bude zabývat převážně front-end částí systému. Tzn. grafický návrh designu a jeho implementace, návrh struktury webového rozhraní a souvislosti (informační provázanost) jednotlivých zobrazovaných částí, napojení na jádro systému, testování uživatelského rozhraní a jeho korekcí. Toto však není výhradní a nevylučuje občasný zásah do struktury

projektu jako takového. Tato práce dále pokračuje v následující struktuře:

V první kapitole se tato práce věnuje především analýze požadavků a stávajících řešení této problematiky. Dále se též v této kapitole práce věnuje analýze procesů, požadavků a případů užití systému. Krátce se zde práce zmiňuje i o spolupráci při tvorbě jádra a integraci rozhraní.

V druhé kapitole se práce věnuje návrhu grafického designu, uživatelského rozhraní a návrhu vhodných technologií pro integraci uživatelského rozhraní s jádrem aplikace.

Třetí kapitola popisuje implementaci/realizaci za pomoci zvolených technologií, propojení s jádrem aplikace, projevení bezpečnosti jádra na uživatelském rozhraní a strukturu komponent uživatelského rozhraní.

Kapitola čtvrtá, poslední, popisuje převážně testování uživatelského rozhraní.

Tato bakalářská práce je úzce vázána na souběžně vyvíjené jádro systému, jež je tématem bakalářské práce studenta jménem Dan Zatloukal (dále též označovaný jako back-end vývojář), který je též studentem FIT ČVUT v Praze.

Kapitola 1

Cíl práce

Cílem této bakalářské práce je na základě rešerše navrhnout a implementovat uživatelské webové rozhraní pro Laboratoř inteligentních vestavných systémů na Fakultě informačních technologií (FIT) ČVUT.

Rešeršní část práce si pak dává za cíl analyzovat vhodné postupy a techniky při tvorbě webového rozhraní a webových aplikací. Analyzovat požadavky, procesy, případy užití systému a vhodné metody pro realizaci a propojení rozhraní s jádrem aplikace.

Dalšími cíli rešeršní části práce je návrh designu webového uživatelského rozhraní na základě analýzy požadavků a již existující verze vzhledu pro Laboratoř inteligentních vestavných systémů a popsat vybraná řešení pro implementaci webové aplikace a jejich jednotlivé části.

Cílem praktické části práce je tvorba grafického designu a jeho implementace do webového uživatelského rozhraní.

Na to navazuje napojení rozhraní na jádro aplikace za pomoci zvolených technologií. Poté je důležité validovat kód webových stránek a v neposlední řadě testovat systém a následně provést korekci webového designu (na základě výsledků uživatelského testování).

Kapitola 2

Analýza

Problematika designu/návrhu a tvorby webových rozhraní nebo rozhraní jako takového, je velice rozsáhlá a není v ní vždy jednoznačné řešení určitého problému. Ve většině případů záleží na přesné specifikaci dat/informací a jejich využití, k přesnému určení, jak by se taková data měla zobrazovat. Proto je vhodné strávit nemalou část času vývoje aplikace pro zobrazování takových dat, zamyšlením se nad tím, jak budou data provázána a jaký mají informační význam. O tom se též zmiňuje například [1], nebo [2] (například v kapitole Metodologie návrhu).

2.1 Současná řešení

Tato část se, z výše uvedených důvodů, věnuje převážně analýze designu a uživatelských rozhraní obdobných nebo příbuzných informačních systémů.

2.1.1 Informační systémy ČVUT

ČVUT již využívá několik informačních systémů jako je například KOS, Edux, Moodle-vyuka, portál předmětu DBS, nebo v době tvorby této práce nový systém FIT klasifikace.

Tyto systémy se v některých aspektech podobají přístupu, jež je zapotřebí u evidence, respektive správy vestavných systémů a projektů pro Laboratoř inteligentních vestavných systémů.

Systém nejpodobnější potřebám pro Livs je portál předmětu DBS. Tento portál umožňuje aktivní zapojení jak žáků, tak vyučujících při tvorbě projektů zaměřených na databázové systémy. Akce, ve kterých si je portál předmětu DBS podobný s akcemi, jež je potřeba provádět pro Laboratoř inteligentních vestavných systémů:

- Ukládání informací a jejich následná prezentace.

- Aktivní tvorba obsahu, jak ze strany studenta, tak ze strany vyučujícího.
- Tvorba projektů a jejich následné využití na hardwaru.

Žádný z těchto systémů však nepracuje nad daty, jež by byly stejné povahy jako data, jež jsou potřeba pro evidenci, respektive správu vestavných systémů. Ze stejného důvodu není nezbytné se těmito systémy zabývat příliš detailně.

Přesto však má cenu se na některé tyto systémy podívat podrobněji z hlediska designu a analyzovat jeho benefity a zápory, jež by mohl přinést.

2.1.1.1 Informační systém Edux

Pro jednoduchost budeme nazývat systém Edux systémem informačním. I přes fakt, že tak jednoduché to v tomto případě není, systém jako samotný „*je rozhraním pro další systémy*“ [3].

Dle [3], účel systému je rozdělen následovně:

- Fáze přípravy materiálů.
  - Tvorba studijních materiálů.
  - Publikace studijních materiálů.
    - nastavení ACL.
- Fáze použití materiálů.
  - Import studentů z KOSu, tvorba skupin pro ACL.
  - Harmonogram studia (semestru).
  - Klasifikace.
  - Rozhraní pro další systémy.

Při analýze těchto faktů se zdá, že se Edux tolik neliší od systému, kterým se zabývá tato práce. Data, která se v tomto systému vyskytují, jsou však odlišné povahy. Na druhou stranu je to ale systém dlouhodobě osvědčený a otestovaný velkým počtem uživatelů. Proto je dobré analyzovat design a strukturu jeho rozhraní.

Rozhraní Eduxu disponuje velkým počtem ovládacích prvků. Tyto prvky jsou z velké části v podobě proklikávacích odkazů. Nikde nejsou viditelné žádné pohyblivé prvky, nebo eye candy prvky. Toto je pochopitelné, neboť v prostředí, kde je velké množství informací, je důležitá přehlednost a prioritu má ovladatelnost před vzhledem.

Důležitá je však především navigace. Navigace je statická a zůstává vždy na jednom místě. Vedlejší menu vlevo a hlavní nahoře v liště.

Dále je také důležité si povšimnout minimalistické hlavičky a zápatí, které však dodržuje dostatečný počet identifikačních prvků, takže je jasné, že se jedná o systém pro FIT ČVUT.

Tento vzhled však není moc responzivní. Což je vzhledem k povaze dat pochopitelné, ale pro systém Livs nevhodné.

Systém také volí minimalistický styl prvků vzhledem k počtu dat.

2.1.1.2 Informační systém FIT klasifikace

Tento informační systém je modernější než Edux. Tato skutečnost je vidět hned na úvodní stránce webu systému.

Uživatelské rozhraní dává přednost modernějším barvám a vzhledem k množství dat a ovládacích prvků si může dovolit zobrazovat i eye candy prvky. Takovýto design je motivující, působivý a přívětivý.

Důležité je opět si povšimnout relativně minimalistické hlavičky rozhraní, ve které je zakomponovaná hlavní navigace. Sekundární navigace je pak, stejně jako u systému Edux, na levé straně rozhraní. Drobečková navigace zde není přítomna, což je též vzhledem k řešení odkazů v levém menu pochopitelné.

Za zmínku však stojí následující:

- Hlavičce chybí jakákoliv identifikace školy, nebo fakulty, což je doporučováno v grafickém manuálu ČVUT [4].
- Funkčnost webového rozhraní je závislá na funkčnosti JavaScriptu.
- Levé menu se chová jinak při skrytí na velkém monitoru než na malém.
- Kvalitní responzivní design přidává na profesionálním dojmu a lákavosti rozhraní.

2.1.1.3 Webový portál DBS

Tento portál je velice rozsáhlý a mnohokrát rozšiřovaný. „*Hlavním důvodem vzniku tohoto systému bylo ulehčení práce vyučujícím ve zmíněném předmět Databázové Systémy, neboť spousta práce jež vyučující vykonával se dala automatizovat*“. [5]

Úplně stejným případem je v době této práce právě Laboratoř inteligentních vestavných systémů.

Tento portál volí podobný přístup k designu a uživatelskému rozhraní jako dříve zmiňovaný systém FIT klasifikace a Edux.

Hlavička a zápatí volí minimalistický styl. Stejně jako u FIT klasifikace, hlavička obsahuje hlavní menu a sekundární menu je opět na levé straně s absencí drobečkové navigace (viz. slovník: drobečková navigace).

Za zmínku stojí následující:

- Hlavičce chybí jakákoliv identifikace školy, nebo fakulty, což je doporučováno v grafickém manuálu ČVUT [4]. V tomto případě však barevná paleta trochu objasňuje situaci.
- Funkčnost webového rozhraní je závislá na funkčnosti JavaScriptu stejně jako u FIT klasifikace.
- I v takto složitém prostředí je rozhraní schopné responzivního chování.

2.1.2 Současné řešení LIVS

Laboratoř inteligentních vestavných systémů má sice v době tvorby této práce již webový portál s implementovaným uživatelským rozhraním, avšak toto rozhraní slouží jinému účelu, než je evidence vestavných systémů. Slouží spíše jako omezený pracovní prostor postavený na DokuWiki. Správcem tohoto pracovního prostoru je Ing. Miroslav Skrbek, Ph.D.

Pro evidenci vestavných systémů a správu projektů není v době psaní této práce zhotoveno žádné uživatelské rozhraní, jež by bylo přizpůsobeno zmíněné problematice.

A tudíž veškerou výuku a správu zařizuje právě vyučující přímo na své pracovní stanici (ne nutně).

To znamená, že činnosti jako:

- Nahrávání software do robotů SoftBank Nao.
- Nahrávání software do ostatních vestavných systémů.
- Ukládání souborů a dat potřebných pro používání robotů Nao a vestavných systémů.

Probíhají všechny manuálně, decentralizovaně.

2.1.2.1 Aktuální procesy

Tato podsekce plynule navazuje na informace zmíněné výše a zaměřuje se na specifikaci procesů, jež probíhají v této laboratoři v době tvorby této práce.

Procesy zde budou uváděny jen stručně, neboť jak vyplývá z následujících paragrafů, většinu relevantních procesů nelze v současném stavu laboratoře efektivně provádět.

Následující procesy jsou řazeny dle relevance vůči tématu, jímž se zabývá tato práce (sestupně).

Evidence zařízení neprobíhá v Livs vůbec, neboť neexistuje efektivní řešení pro tento proces. Neexistuje tedy způsob, jímž lze přehledně zjistit počet, stav a informace o všech vestavných systémech, jež jsou ve správě laboratoře. Proto se vždy musí znovu informace dohledávat, nebo konstantně udržovat v paměti.

Evidence projektů je stejným případem jako evidence vestavných systémů. V tomto případě však existuje možnost projekty v určité míře spravovat a vytvářet v aktuálním portálu Livs. Tento proces však probíhá jako zanesení potřebných informací k projektu do textové podoby v DokuWiki, jež běží na výše zmíněném portálu. Tento proces se však ani v aktuálním stavu moc nevyužívá.

Nahrávání software do zařízení probíhá decentralizovaně přímo z pracovní stanice uživatele.

Kroky nutné k realizaci:

1. Dohledání informací o zařízení.
    - ip adresa,
    - mac adresa.
2. Připojení se na dané zařízení.
3. Nahrání potřebného souboru do zařízení.
4. Ukončení spojení se zařízením.

Toto musí provést pro každé zařízení, do nějž chce daný soubor nahrát.

2.1.2.2 Nevýhody designu a rozhraní

Pokud budeme vycházet z faktu že „*s příchodem novějších technologií přichází i nová zařízení, která umožňují uživatelům přistupovat k Internetu a zobrazovat webové stránky na těchto zařízeních, která se od sebe vzájemně odlišují různými atributy. A proto by tomuto způsobu mělo být přizpůsobeno i uživatelské rozhraní*“ [6], pak není designově systém vhodný pro zobrazování velkého počtu informací a ovládacích prvků na menších obrazovkách.

Zejména je nevhodná hlavička portálu a neaktuální designový návrh jednotlivých komponent.

2.1.2.3 Hotové šablony

Samozřejmě také musíme vzít v potaz již hotové šablony, jež lze pořídit a nasadit na potřebný systém.

Pokud pominu skutečnost, že by bylo nemístné pro takovýto projekt použít již připravenou šablonu, tak stále existuje dostatek pádných důvodů proč tak neudělat.

Mezi ně patří například:

- Šablona nemůže být upravována neomezeně.
- Případná nutnost změny šablony a její následná opětovná úprava.
- Hledání vhodné šablony může být časově náročné.
- Možné omezení při napojování na jádro.
- V případě změny povahy dat nebo jejich roli v systému, existuje možnost, že se opět projeví první bod tohoto seznamu.

2.2 Požadavky

„*Požadavek je podmínka nebo vlastnost, která musí být splněna pro naplnění zakázky.*“ [7] (přeložil Erik Zatloukal)

Dle [7] požadavky též specifikují pomocí jakých technologií by systém měly být realizovány a provozovány.

V této sekci se práce zabývá požadavky na systém, poskytnutými zadavatelem projektu.

Specificky jsou zde vybrány požadavky na front-end aplikace. Některé z těchto požadavků ovšem mohou zasahovat i do specifikací pro jádro, pokud se daný požadavek bude týkat obou částí.

2.2.1 Funkční požadavky

Funkční požadavky jsou možnosti/procesy/schopnosti, jimiž systém musí disponovat, aby vyřešil daný problém/problematiku. Jinak řečeno, jsou to požadavky přímo na specifickou činnost, již lze v systému vykonávat. Toto vychází ze studie [7].

F1 - Evidence informací pro zařízení
Rozhraní v rámci aplikace bude umožňovat zobrazení, editaci a vytváření informací potřebných pro používání robotů Nao a zařízení v rámci laboratoře (Livs). V rámci těchto informací musí též zobrazovat jestli je zařízení připojeno nebo ne. Informace o zařízeních budou brány z databáze jádra aplikace.

priorita: vysoká,

typ: funkční požadavek,

složitost: vysoká (velké množství informací).

F2 - Evidence informací o projektech
Rozhraní v rámci aplikaci umožní zobrazování, editaci a vytváření informací pro projekty v rámci laboratoře(Livs). Tyto informace zahrnují i komponenty s projektem související, jako jsou například konfigurace nebo verze. Tyto informace a komponenty se budou ukládat do databáze systému.

priorita: vysoká,

typ: funkční požadavek,

složitost: vysoká (velký počet souvisejících informací a komponent).

F3 - Autorizace a autentizace
Rozhraní aplikace umožní uživateli se autentizovat, a tím přihlásit do svého účtu napojeného na jádro systému. Rozhraní též umožní efektivně zakrývat zabezpečený obsah neoprávněným uživatelům.

priorita: vysoká,

typ: funkční požadavek,

složitost: střední.

F4 - Tvorba webových stránek
Rozhraní aplikace umožní uživateli vytvářet a následně zobrazovat a editovat webové stránky skrze jádro systému.

priorita: nízká,

typ: funkční požadavek,

složitost: nízká – střední (záleží na komplexnosti).

F5 - Editace, tvorba a zobrazení informací pro ostatní části systému

Rozhraní v rámci aplikace umožní editaci, tvorbu a zobrazení informací pro uživatele, práva, komponenty a skripty (ne nutně úprava/tvorba u skriptů) v rámci laboratoře(Livs). Tyto informace a komponenty se budou získávat a ukládat do databáze systému.

priorita: nízká – střední,

typ: funkční požadavek,

složitost: střední – vysoká (velké množství informací).

Funkční požadavek F5 nemusí nutně být dodán ke konci tvorby této práce. Nicméně je stále požadavkem plynoucím z povahy systému.

2.2.2 Nefunkční požadavky

Jsou opakem funkčních požadavků. To znamená, že jsou to obecné požadavky na systém/aplikaci. Například požadavky na dostupnost, přenositelnost, výkon nebo požadavky omezující. Toto vychází ze studie [7].

N1 - Důraz na grafickou stránku
Rozhraní má mít motivující vzhled pro studenty i vyučující. Grafická stránka by měla být unikátní.

priorita: střední,

typ: nefunkční požadavek,

složitost: vysoká (unikátní grafický návrh).

K tomuto požadavku je nutné uvést fakt, že návrh designu musím realizovat jako tvůrce front-endu a to vzhledem k datům, se kterými jsem se dřív nesetkal. Při realizaci projektu nebyla vyžádána účast grafického designera ze strany zákazníka.

N2 - Snadná a nepodmíněná ovladatelnost
Rozhraní by mělo být snadno ovladatelné. Nemělo by obsahovat složité prvky ovládání. Rozhraní by mělo být funkční bez nutnosti využívat nestandardní technologie na straně klienta.

priorita: střední,

typ: nefunkční požadavek,

složitost: nízka–střední (při správném návrhu).

N3 - Kompatibilita s častými prohlížeči
Rozhraní musí být možné používat na často využívaných webových prohlížečích, jimiž jsou Google Chrome, FireFox, Internet Explorer, dále též jako IE, a Edge a jejich mobilní verze.

priorita: střední,

typ: nefunkční požadavek,

složitost: střední (IE je speciální připad prohlížeče, kde některé věci fungují naopak než v ostatních zmíněných).

2.3 Používané prohlížeče a zobrazování

V návaznosti na poslední požadavek sekce 2.2.2 se tato sekce zabývá analýzou užívání jednotlivých prohlížečů, případně jejich konkrétními verzemi. Dále se tato sekce též zabývá často používanými rozlišeními obrazovek na straně uživatele webového prohlížeče.

2.3.1 Prohlížeče v České Republice

Ze statistik poskytovaných na gs.statcounter.com vychází následující tabulka.

Tabulka 2.1: Analýza využívání webových prohlížečů v ČR

Využití prohlížečů v České Republice						
Prohlížeče	Chrome	Firefox	Safari	IE	Opera	Edge
Procentuální zastoupení	58,84 %	15,01 %	8,02 %	6,4 %	4,49 %	3,46 %

Šedivé kolonky jsou prohlížeče, jež nejsou součástí požadavku **N3**.

2.3.2 Rozlišení obrazovky v České Republice

Dle nefunkčních požadavků na systém, respektive rozhraní ze sekce 2.2.2 je také vhodné znát nejpoužívanější rozlišení napříč všemi webovými prohlížeči. Díky této znalosti lze pak jednodušeji provádět některá rozhodnutí při návrhu či implementaci.

Desktopová zařízení (stolní)
Následující tabulka pochází z gs.statcounter.com a zobrazuje procentuální zastoupení jednotlivých rozlišení pro desktopová zařízení, jež jsou využívány při webovém zobrazování. Statistika zobrazuje nejpoužívanější rozlišení (ne všechny).

Tabulka 2.2: Analýza nejčastěji používaných rozlišení obrazovek v ČR (stolní zařízení)

Nejčastěji používaná rozlišení v ČR						
Rozlišení	1920x1080	1680x1050	1366x768	1536x864	1280x1024	1024x768
Procentuální zastoupení	19,7 %	4,6 %	19,93 %	6,58 %	5,41 %	13,91 %

Mobilní zařízení
Následující tabulka pochází z gs.statcounter.com a zobrazuje procentuální zastoupení jednotlivých rozlišení pro mobilní zařízení, jež jsou využívány při webovém zobrazování. Statistika zobrazuje nejpoužívanější rozlišení (ne však všechny).

Tabulka 2.3: Analýza nejčastěji používaných rozlišení obrazovek v ČR (mobliní zařízení)

Nejčastěji používaná rozlišení v ČR (mobilní)						
Rozlišení	360x640	375x667	320x568	320x570	640x360	320x534
Procentuální zastoupení	51,53 %	9,04 %	6,32 %	2,94 %	2,69 %	2,44 %

2.3.3 Prohlížeče celosvětově

Podobné údaje, tentokrát však nejen pro ČR, jsou k nalezení na w3schools.com, odkud pochází následující tabulka.

Tabulka 2.4: Analýza využívání prohlížečů celosvětově

2018	Chrome	Edge/IE	Firefox	Safari	Opera
February	77.9 %	4.1 %	11.8 %	3.3 %	1.5 %
January	77.2 %	4.1 %	12.4 %	3.2 %	1.6 %

2017	Chrome	IE/Edge	Firefox	Safari	Opera
December	77.0 %	3.9 %	12.4 %	3.3 %	1.6 %
November	76.8 %	4.3 %	12.5 %	3.3 %	1.6 %
October	76.1 %	4.1 %	12.1 %	3.3 %	1.2 %
September	76.5 %	4.2 %	12.8 %	3.2 %	1.2 %
August	76.9 %	4.3 %	13.1 %	3.0 %	1.2 %
July	76.7 %	4.2 %	13.3 %	3.0 %	1.2 %
June	76.3 %	4.6 %	13.3 %	3.3 %	1.2 %
May	75.8 %	4.6 %	13.6 %	3.4 %	1.1 %
April	75.7 %	4.6 %	13.6 %	3.7 %	1.1 %
March	75.1 %	4.8 %	14.1 %	3.6 %	1.0 %
February	74.1 %	4.8 %	15.0 %	3.6 %	1.0 %
January	73.7 %	4.9 %	15.4 %	3.6 %	1.0 %

2.3.4 Rozlišení obrazovky celosvětově

Pro úplnost statistik je též vhodné se podívat i na celosvětové výsledky těchto průzkumů. I tyto statistiky mohou být užiteční pro budoucí rozvoj nebo lepší představu, jak je potřeba aplikaci přizpůsobit.

Následující tabulka pochází z w3schools.com a zobrazuje procentuální zastoupení jednotlivých rozlišení pro mobilní i stolní zařízení, jež jsou využívány při webovém zobrazování. Statistika zobrazuje nejpoužívanější rozlišení (ne všechny).

Tabulka 2.5: Analýza využívaných rozlišení celosvětově

Date	Other high	1920x1080	1366x768	1280x1024	1280x800	1024x768	Lower
January 2018	32.9%	18%	34%	4%	3%	2%	6.1%
January 2017	31.6%	17%	35%	5%	4%	3%	4.4%
January 2016	30.7%	18%	35%	6%	4%	3%	3.3%
January 2015	32.7%	16%	33%	7%	5%	4%	2.3%
January 2014	34%	13%	31%	8%	7%	6%	1.0%
January 2013	36%	11%	25%	10%	8%	9%	1.0%
January 2012	35%	8%	19%	12%	11%	13%	2%
January 2011	50%	6%		15%	14%	14%	1%
January 2010	39%	2%		18%	17%	20%	4%
January 2009	57%					36%	7%
January 2008	38%					48%	14%
January 2007	26%					54%	20%
January 2006	17%					57%	26%
January 2005	12%					53%	35%
January 2004	10%					47%	43%
January 2003	6%					40%	54%

Dále je také potřeba zjistit které verze prohlížečů se používají. To platí obzvláště pro IE, neboť v jednotlivých verzích se podporovaná funkcionalita výrazně mění. To není tak kritické u ostatních prohlížečů jako například Chrome nebo FireFox. Proto zde nejsou statistiky pro každý z nich, ale jen statistika pro IE, jež je zobrazena v následující tabulce.

Následující tabulka vychází ze statistik z w3schools.com a ukazuje procentuální zastoupení jednotlivých verzí IE z pohledu užívání (nejen v ČR).

Tabulka 2.6: Analýza využívání prohlížeče Internet Explorer

2018	Total	Edge16	Older Edge	IE11	Older IE
February	4.1	1.2	0.7	2.1	0.1
January	4.1	1.1	0.9	2.0	0.1

2017	Total	Edge 16	Edge15	Edge14	Edge13	Edge12	IE11	Older
December	3.9	0.6	0.7	0.3	0.1	0.1	2.0	0.1
November	4.3	0.4	1.1	0.3	0.1	0.0	2.4	0.0
October	4.1	0.1	1.2	0.3	0.1	0.1	2.3	0.0
September	4.2		1.1	0.4	0.1	0.1	2.5	0.0
August	4.3		0.8	0.4	0.2	0.1	2.6	0.2
July	4.2		0.7	0.6	0.2	0.1	2.6	0.0
June	4.6		0.6	0.8	0.2	0.1	2.6	0.3
May	4.6		0.3	1.1	0.2	0.1	2.7	0.2
April	4.6		0.1	1.2	0.2	0.1	2.8	0.2
March	4.8			1.4	0.2	0.2	2.9	0.1
February	4.8			1.1	0.2	0.1	3.1	0.3

2.4 Případy užití

Tato sekce se zabývá analýzou případů užití. Případy užití slouží k popsání činností jednotlivých aktérů v systému. Je to tedy „*specifikace posloupnosti činností, včetně proměnných posloupností a chybových posloupností. Jaké systémy, podsystémy nebo třída může vykonávat prostřednictvím interakce s vnějšími aktéry.*“ [8]

V případě takto velkého projektu (jako je projekt pro Livs), respektive vývojového týmu, není potřeba se do detailu zabývat všemi případy užití. Proto zde jsou uvedeny jen ty, které pomohou pochopit chod systému a specifikovat složitější požadavky.

2.4.1 Seznam účastníků

Vydefinovat přesně všechny účastníky zde není tak jednoduchou záležitostí. Respektive vydefinovat účastníky tak, aby přesně reflektovali realitu systému, neboť každý jednotlivý uživatel může mít přidělena jiná práva. Z tohoto důvodu zde budou uvedeni aktéři s právy, která budou mít nejpravděpodobněji přiřazena ve většině případů.

Díky výše uvedenému faktu tak můžeme orientačně zobrazit jaké akce budou jakými aktéry v systému pravděpodobně vykonávány.

Účastníci:

- nepřihlášený uživatel,
  - V systému vykonává roli pozorovatele.
  - Přistupuje k veřejnému obsahu, jako jsou veřejné projekty a zobrazuje informace o nich.
- student,
  - Dohledává informace.
  - Využívá výkoné funkce.
- zaměstnanec,
  - Eviduje a přistupuje k zařízením.
  - Připravuje materiály.
  - Využívá výkoné funkce.
- vyučující,
  - Stejné jako zaměstnanec.
  - + Spravuje projekty.
- administrátor.
  - Spravuje práva pro přístup k zabezpečeným materiálům a zařízením.
  - Eviduje a upravuje veškerá data u kterých to systémem povoluje.

2.4.2 Vydefinování případů užití

V této sekci jsou vydefinovány případy užití, jež jsou na základě požadavků nejdůležitější. Také zde nejsou uvedeny jednotlivé scénáře.

Scénáři se tato práce nezabývá, neboť to není potřeba. Tým jenž provádí analýzu, je totiž tentýž, jako tým návrhový a implementační.

Rozsah a složitost případů užití také nejsou tak komplikované, a tak je modelování scénářů redundantní.

V některých případech užití se vyskytuje výraz **oprávněný uživatel**. Tento výraz souvisí s již výše zmíněným systémem autorizace. Není to tudíž určitá role, ale libovolná role s přidělenými právy k dané akci.

UC1 - Zobrazit informace o zařízení
Zobrazí oprávněnému uživateli, většinou studentovi, informace o požadovaném zařízení evidovaném v systému na uživatelské rozhraní.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má oprávnění zobrazit dané zařízení.

výstupní podmínky:
- Žádné.

,

UC2 - Upravit údaje o zařízení
Umožňuje oprávněnému uživateli, většinou vyučujícímu nebo zaměstnanci, upravovat údaje, které jsou obsaženy v korespondující entitě k danému zařízení.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má oprávnění upravovat dané zařízení.

výstupní podmínky:
- Všechny zadané informace jsou validní.
- Uživatel vyplnil všechna povinná pole.

,

UC3 - Evidovat nové zařízení
Umožňuje oprávněnému uživateli, většinou vyučujícímu nebo zaměstnanci, evidovat nové zařízení v systému a upravit jeho údaje, které jsou následně obsaženy v korespondující entitě k danému zařízení.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má oprávnění vytvářet zařízení.

výstupní podmínky:
- Všechny zadané informace jsou validní.
- Uživatel vyplnil všechna povinná pole.

,

UC4 - Zobrazit informace o projektu
Zobrazí oprávněnému uživateli, většinou studentovi, informace o požadovaném projektu evidovaném v systému na uživatelské rozhraní.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má oprávnění zobrazit daný projekt.

výstupní podmínky:
- Žádné.

,

UC5 - Upravit údaje o projektu
Umožňuje oprávněnému uživateli, většinou vyučujícímu, upravovat údaje, které jsou obsaženy v korespondující entitě k danému projektu.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má oprávnění upravit daný projekt.

výstupní podmínky:
- Všechny zadané informace jsou validní.
- Uživatel vyplnil všechna povinná pole.

,

UC6 - Evidovat nový projekt
Umožňuje oprávněnému uživateli, většinou vyučujícímu, evidovat nový projekt v systému a upravit jeho údaje, které jsou následně obsaženy v korespondující entitě k danému zařízení.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má oprávnění vytvářet projekty.

výstupní podmínky:
- Všechny zadané informace jsou validní.
- Uživatel vyplnil všechna povinná pole.

,

UC7 - Nastavení práv k zabezpečenému obsahu
Administrátor (většinou) nebo oprávněný uživatel nastaví požadovaná práva k přístupu, editaci, nebo tvorbě obsahu (zvoleného) určitému uživateli.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má oprávnění vytvářet, respektive upravovat práva.

výstupní podmínky:
- Pokud jde o vytváření, pak nesmí stejné právo již existovat.

,

UC8 - Vytvoření nového uživatele
Administrátor (většinou) nebo oprávněný uživatel vytvoří nového uživatele v rámci systému a nastaví mu základní informace v korespondující entitě.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má administrátorská práva.

výstupní podmínky:
- Uživatel v systému ještě neexistuje.
- Nový uživatel má validní data.
- Nový uživatel má zadaná všechna povinná pole.

,

UC9 - Vytvoření webové stránky
Administrátor (většinou) nebo oprávněný uživatel vytvoří novou webovou stránku, jež se uloží do databáze systému.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má právo vytvářet webové stránky.

výstupní podmínky:
- Žádné.

,

UC10 - Spuštění nebo přesunutí souboru na zařízení
Umožní oprávněnému uživateli, většinou vyučujícímu nebo žákovi, spustit nebo přesunout soubor na vybrané zařízení z rozhraní aplikace.

vstupní podmínky:
- Uživatel je plně autentizován.
- Uživatel má právo na výkoné funkce v systému.

výstupní podmínky:
- Požadované zařízení je stále připojeno.

,

2.4.3 Splnění požadavků

Pro kontrolu je také třeba uvést tabulku splněných požadavků.

Tabulka 2.7: Mapování požadavků a případů užití

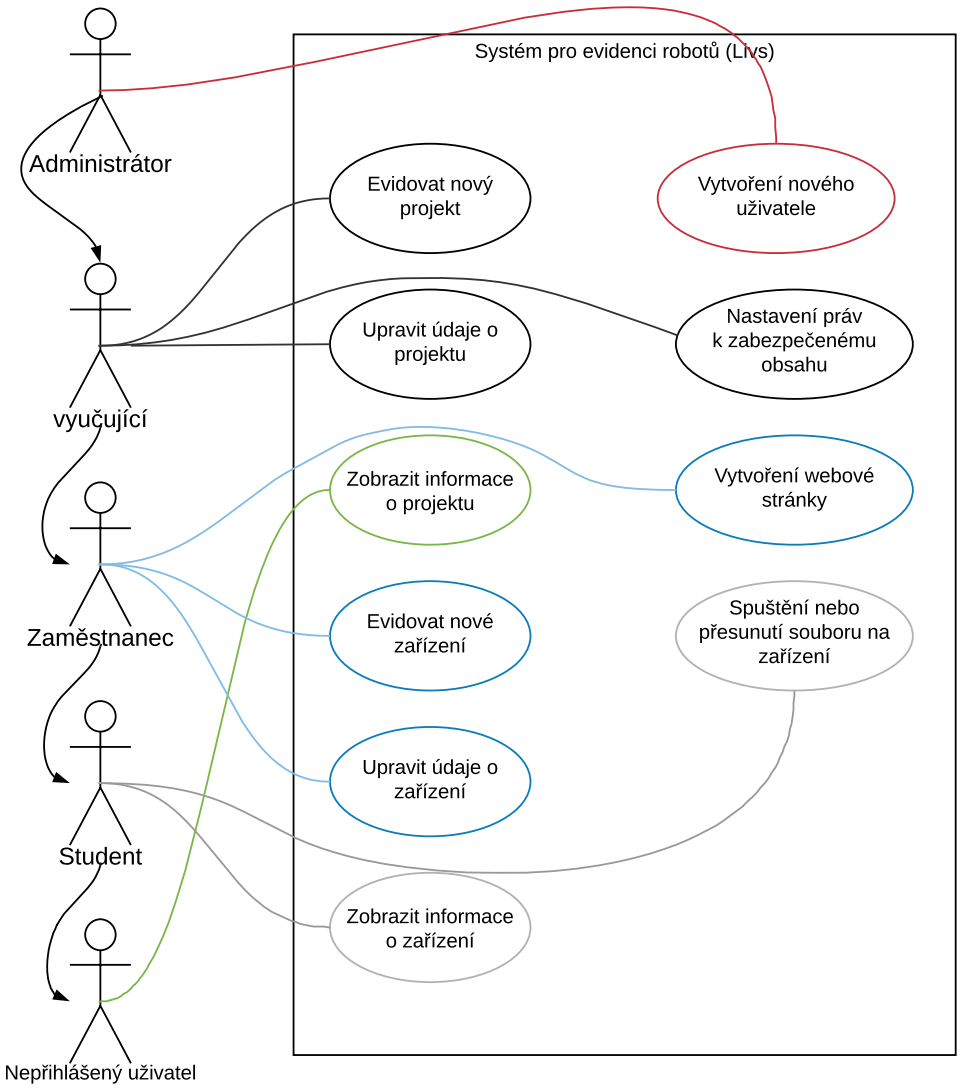
Splněné funkční požadavky										
	funkční požadavky									
Požadavky	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10
F1	✓	✓	✓							
F2				✓	✓	✓				
F3							✓	✓		
F4									✓	
F5										✓

Každý případ užití by se měl mapovat na některý z funkčních požadavků. Výše uvedená tabulka právě toto mapování zobrazuje.

2.4.4 Moedel případů užití

V následujícím obrázku jsou zobrazeny role a jejich nejpravděpodobnější případy užití (používané danými rolemi).

Obrázek 2.1: UML diagram případů užití



2.5 Závislost na jádru systému

V úvodu této práce je zmíněn fakt, že uživatelské rozhraní, respektive front-end, jež je tématem této práce, úzce souvisí se souběžně vyvíjeným jádrem sys-

tému pro Projekt LIVS, respektive Laboratoř inteligentních vestavných systémů (Livs).

Jádro aplikace nebo-li back-end je svázaný s vývojem front-endu a to obzvláště v případě, že předem není známá přesná specifikace dat nebo informací, jež bude potřeba zobrazovat.

Vyvíjet je lze zvlášť, avšak „*aby program/systém fungoval, musí být vyřešeny obě části*“ [8, s. 2]. Ani jedna z částí sama o sobě není použitelným produktem pro produkci. Jinými slovy „*v generalizované podobě jsou tedy pojmy front-end a back-end označením dvou různých částí jednoho celku — softwarové, respektive webové aplikace*“ [8, s. 2].

V případě volby agilní metodiky, o které se budeme zmiňovat dále, nebo její obdoby, je pak potřeba na základě iterací měnit back-end i front-end současně. Z těchto iterací se totiž programátor / programátorský tým dozví, jestli prezentovaná verze odpovídá potřebám zadavatele, popřípadě co je potřeba změnit (viz. sekce 2.6.2.2). Toto platí hlavně v případě menších projektů s přesně nevymezenými požadavky. Tvrzení též vychází z [8] [9].

Z výše uvedených důvodů se další kapitola bude věnovat, respektive analyzovat metodiky a postupy vhodné pro spolupráci na těchto dvou částech aplikace.

2.6 Metodiky návrhu a vývoje

Tato sekce se zaměřuje na analýzu metod, postupů a metodik, jejichž znalost by v návrhu, respektive implementaci, mohla být užitečná. Díky jejich znalosti by měla být rozhodnutí uskutečňována v návrhu, respektive implementaci, snazší.

„*Primárním faktorem při vývoji informačního systému jsou lidé. Metodiky však tuto skutečnost dostatečně nezohledňují.*“[10] Zde musím zdůraznit, že s tímto faktem plně souhlasím. V některých případech volba špatných nebo přehnaně komplikovaných postupů/metodik, může vést k značným problémům mezi vývojáři. Obzvláště pokud mají odlišnou úroveň znalostí. Proto se tomu zde tato práce také věnuje.

S tímto úzce souvisí zdůvodnění na konci sekce 2.5

2.6.1 Vývoj a týmová spolupráce

Projekty jež vyžadují spolupráci mezi jednotlivými vývojáři též vyžadují specifikaci společného přístupu k práci a komunikaci. Toto platí především pro projekty větších rozměrů. Ne však výhradně. Projekty menších rozsahů také vyžadují určitou formu komunikace mezi vývojáři/programátory v případě, že vývoj částí systému/aplikace není nezávislý. Tvrzení též vychází z [8] [9].

Díky sekci 2.5 víme, že je důležité se tím zabývat i v této práci.

2.6.1.1 Tým a jeho struktura

V případě, že je projekt rozsáhlejší a jeho vývoj je časově omezený, je pak logickým řešením rozdělení vývoje jednotlivých částí systému/aplikace mezi více vývojářů. Tím docílíme rychlejšího vývoje a efektivnější implementace (preciznější). To je ovšem pravdivé pouze pokud jsou vývojáři schopni spolupracovat.

V případě větších projektů je potřeba tým organizovat. U menších projektů už to není tak jasné. Záleží pak na vývojářích, jakým způsobem chtějí problematiku vývoje řešit.

Volba organizace týmu je závislá, jak už bylo zmíněno, na velikosti projektu, ale i na osobní preferenci každého účastníka. Poslední část této věty nevychází z žádného literárního zdroje, ale často tomu tak bývá v praxi.

Tabulka z [9] nám říká, že volit lze například z:

Tabulka 2.8: Analýza organizace týmů

Organizace týmů	
Nestrukturované týmy	Strukturované týmy
Dělí práci podle objemu. Mohou být organizovány jako:	Dělí práci podle profese. Mohou být organizovány jako:
• „Osamělí vlci“ • „Horda“ • „Demokratická skupina“	• „Chirurgický tým“ • „Tým hlavního programátora“ • „Agilní skupina“ • „Více-týmová organizace“

Jednotlivé skupiny zde nebudou do detailu rozebírány. Všechna výše uvedená tvrzení, stejně tak jako výše uvedenou tabulku můžeme též (v obdobné formě [ne výhradně]) nalézt například v [9].

2.6.2 Metodiky vývoje software

V této sekci je práce zaměřena na analýzu skupiny metodik, jejichž znalost je užitečná pro sekci návrhu této práce (při volbě metodik vývoje).

Jak se zmiňuje Alena Buchalcevová v knize Metodiky vývoje a údržby informačních systémů [10], volba správné metodiky pro vývoj je poměrně složitou záležitostí, neboť možností je mnoho.

Nejprve uvedeme definici metodiky dle Aleny Buchalcevové, již z výše zmíněné knihy. „*metodika budování IS/ICT definuje principy, procesy, praktiky, role, techniky, nástroje a produkty používané při vývoji, údržbě a provozu in-*

formačního systému, a to jak z hlediska softwarově inženýrského, tak z hlediska řízení.“ [10]

2.6.2.1 Rigorózní metodiky

Tyto metodiky jsou založeny na tom, že při vývoji aplikace/systému se nejprve plánuje a až poté realizuje.

Některé z těchto metodik dovolují iterativní vývoj, ale zpravidla jsou postaveny na vodopádovém modelu.

„*Rigorózní metodiky, vycházejí z přesvědčení, že procesy při budování IS/ICT lze popsat, plánovat, řídit a měřit.*“ [10]

Tvorba materiálů potřebných k dodržení těchto metodik je poměrně rozsáhlá. To však zabere velké množství času, ve kterém zákazník nebude schopný sledovat pokroky a efektivně testovat, jestli zvolené řešení odpovídá jeho potřebám (pokud nejsou striktně zadány požadavky).

Z těchto výše uvedených důvodů nejsou tyto metodiky vhodné pro potřeby projektu povahy stejné, nebo podobné projektu, jemuž se věnuje tato práce. Z tohoto důvodu je zde ani nebudeme dále podrobněji analyzovat.

Tyto informace vycházejí též ze zdrojů [10] [8] [9].

2.6.2.2 Agilní metodiky

Tyto metodiky, jak se též zmiňují například zroje [10] nebo [8], jsou určeny pro projekty, kdy je potřeba rychlé zavedení systému/aplikace.

Jedná se o metodiky, které umožňují přizpůsobit se měnícím se požadavkům tím způsobem, že se systém/aplikace nejprve nebo ve velmi raném stádiu projektu začne vyvíjet a tato část systému/aplikace se poté předá zákazníkovi. Zákazník vyjádří své připomínky a další požadavky na následující úpravy a cyklus se opakuje. Tato tvrzení též vychází ze zdrojů [10] [8] [9].

Každá agilní metodika má své specifické vlastnosti. Tyto vlastnosti ovlivňují nejen přístup k práci, ale také výsledek práce.

Agilní metodiky se z pravidla vyznačují iterativním přístupem.

To znamená:

- Produkt je dodáván v určitém časovém intervalu (iteraci).
  - Po částech.
  - Často je dodávána určitá funkcionalita.
  - Zákazník se často účastní.
- Produkt je postupně rozšiřován.
- Požadavky se průběžně upravují.
- Vykonávají se automatické testy (ve většině).

Fáze celého projektu
Jak lze zjistit z [8, s. 43], tento iterativní přístup má 4 hlavní fáze:

1. nultá iterace,
    - Tato část je též konečnou částí pokud už není co zhotovit.
2. analýza změny,
3. implementace změny,
4. ukázka zákazníkovi.
    - Odtud se vždy vrací k bodu 1 (nultá iterace).

2.6.2.3 Příklady metodik

Následující tabulka vychází z [9] [10].

Tabulka 2.9: Příklady metodik vývoje

Metodiky	
Agilní	Rigorózní
• SCRUM • Crystal Clear • XP - Extreme Programming • Feature–Driven Development • OpenUP	• OPEN • Rational Unified Process • Enterprise Unified Process

2.7 Technologie

V této sekci se práce zabývá analýzou technologií, které jsou pro řešení problematiky front-end vývoje dostupné.

2.7.1 Nezbytné pro vývoj webové aplikace

Existují technologie bez kterých se vývojář front-endu webové aplikace prakticky neobejde.

Těmito jazyky jsou:

- HTML,
- CSS,
- JavaScript (ne vždy).

Jazyky HTML a CSS zde nebudou podrobněji rozebírány, neboť jejich použití je pro projekt stejné povahy, jako kterým se zabývá tato práce prakticky nezbytné, a tak není důvod je analyzovat.

2.7.1.1 JavaScript, a použití

JavaScript je jazyk, který se obzvláště hodí pokud je zapotřebí vylepšit nebo zpříjemnit „user experience“ neboli ve volném překladu uživatelskou zkušenost. Tím je myšleno chování webové stránky/rozhraní tak jak ho pozoruje uživatel.

Jeho nevýhodou je též to, že obsah, jež se upravuje právě pomocí JavaScriptu nevidí některé vyhledávacími a indexovací enginy. Proto je též potřeba dát si pozor na to jaký obsah JavaScript upravuje. Toto tvrzení též vychází z [11].

Indexování
Pro obecný přehled přiložím tabulku jež výše uvedené tvrzení potvrzuje. Tato tabulka pochází z [11]. Pro úplnost je v testu indexování zahrnuto i html5.

Tabulka 2.10: Analýza indexování JavaScriptu a jejich frameworků u nejznámějších vyhledávacích enginů

Test indexování JavaScriptu a jeho frameworků						
	HTML5	React	AngularJS	AngularJS2	JQuery	PlainJS
Google	✓	✓	✓	✗	✓	✓
Bing	✓	✗	✗	✗	✗	✗
Yahoo!	✓	✗	✗	✗	✗	✗
Ask	✓	✓	✓	✗	✓	✓
Ostatní	✓	✗	✗	✗	✗	✗

Na druhou stranu podle [11] Ask a Google pokrývají 64% celého vyhledávacího marketu.

Nevhodné použití
Tento jazyk není vhodné používat pokud stejného cíle lze dosáhnout pomocí jiného jazyka. Jinými slovy „if it can be done in another language, it should be done in another language“ [12] (první věta je prakticky překladem této věty).

Toto obzvláště platí pokud by takové funkcionality zabrala více času, než ji realizovat pomoci jiného jazyka.

Vhodné použití
Naopak vhodné je tento jazyk použít v případě potřeby responzivních prvků ve smyslu reakce na akci. Například vykonání nějaké funkce na základě kliknutí myší na určitý element stránky.

2.7.1.2 JavaScript a AJAX

je užitečným způsobem jak aplikaci obohatit o interaktivnější prvky, nebo o obnovení informací zobrazovaných na stránce bez nutnosti obnovovat celou stránku.

Nevhodné použití
AJAX není vhodný v případě, že se informace na dané stránce nemění, nebo není potřeba provádět malé úpravy obsahu na stránce, jako například přidání komentáře.

Vhodné použití
Prakticky kdykoliv, kdy je vhodné změnit obsah stránky, nebo upozornit uživatele na změnu obsahu.

To znamená například:

- validace formulářů,
- komentáře,
- filtrování dat.

Výše uvedená tvrzení jsou v obdobné formě též uvedena v [13].

2.7.2 Frameworky (front-endové)

V této části jsou uvedeny krátce popsané jednotlivé front-end frameworky, jejichž znalost pomůže v návrhu při výběru technologií.

Důležitými frameworky jsou, například podle [14] a [15], následující:

- Bootstrap (CSS,HTML, JavaScript),
- React (JavaScript),
- Vue (JavaScript),
- Ember (JavaScript),
- Backbone (JavaScript),
- Angular (JavaScript),

2.7.2.1 Bootstrap

Je asi nejznámějším frameworkem pro tvorbu front-end webových aplikací. Je to vývojářský toolkit, neboli také „sada nářadí“ pro HTML, CSS a JavaScript.

Za jeho pomoci se dá rychle navrhnout prototyp designu, nebo se na něm dá postavit i celá aplikace.

Poskytuje responzivní znovupoužitelné komponenty, pomocí nichž se dá vytvořit layout (rozložení) i styly pomocí kterých se dá navrhnout poměrně vzhledný design. Tyto komponenty se také dají upravovat pomocí vlastních stylů.

2.7.2.2 React

Je open-source JavaScriptovou knihovnou pro tvorbu uživatelských rozhraní.

Tento framework je deklarativně a komponentově orientovaný. To znamená že každá stránka nebo view je tvořena z enkapsulovaných komponent. Tyto komponenty pak dohromady tvoří komplexní uživatelské rozhraní.

Stránky se díky těmto komponentám nemusí obnovovat při změně dat jako celek. Obnovuje se jen daná komponenta.

2.7.2.3 Vue

Je progresivní framework pro tvorbu uživatelských rozhraní podobně jako React.

Tento framework lze použít jak na celou aplikaci, tak jen na její část.

Stejně jako React umožňuje měnit data a informace na stránce pomocí komponent, a to bez nutnosti obnovovat celou stránku. To je též díky využití soběstačných a znovupoužitelných komponent.

2.7.2.4 Ember

Je JavaScriptový framework pro tvorbu robustních a komplexních webových aplikací. Je založený na MVVM (Model View ViewModel) aplikačním návrhovém vzoru.

Tento framework má zabudované best-practices a vlastní templatovací knihovnu.

Stejně jako již předešlé zmíněné frameworky poskytuje možnost obnovovat data a informace na stránce bez nutnosti celou stránku obnovovat. To též zvládá za pomoci komponent a Ember Data.

2.7.2.5 Backbone

Je jedním z mladších JavaScriptových frameworků s RESTful JSON interfacem a je založen na Model View Presenter (MVP) aplikačním návrhovém vzoru.

Tento framework je odlišný od výše zmíněných. Poskytuje spíše strukturu a model pro stavbu a synchronizaci jednotlivých částí aplikace.

2.7.2.6 Angular

Je JavaScriptový framework, který používá Model View Controller (MVC) aplikačním návrhovém vzoru.

Umožňuje používání HTML jako templatovacího jazyka a umožňuje rozšiřovat syntaxi HTML k vyjádření aplikačních komponent.

Stejně jako React nebo Vue je komponentově založeným frameworkem a tudíž poskytuje stejné výhody. Tento framework je však větší a tak je složitější se jej naučit používat.

Výhodou oproti Reactu je však to, že je open source.

Kapitola 3

Návrh

3.1 Struktura systému

Zde se nachází ilustrace a popis struktury systému a rozdělení zodpovědnosti mezi vývojáře. V následujících vyobrazeních se nachází jenom části systému spravované front-end a back-end vývojáři.

Aplikací v tomto diagramu je míněna webová aplikace a uživatelským rozhraním, je míněno webové uživatelské rozhraní.

Legenda k obrázku 3.1 - Ilustrace struktury systému:

zeleně vyznačená plocha:
Části realizované front-end vývojářem.

modře vyznačená plocha:
Části realizované back-end vývojářem.

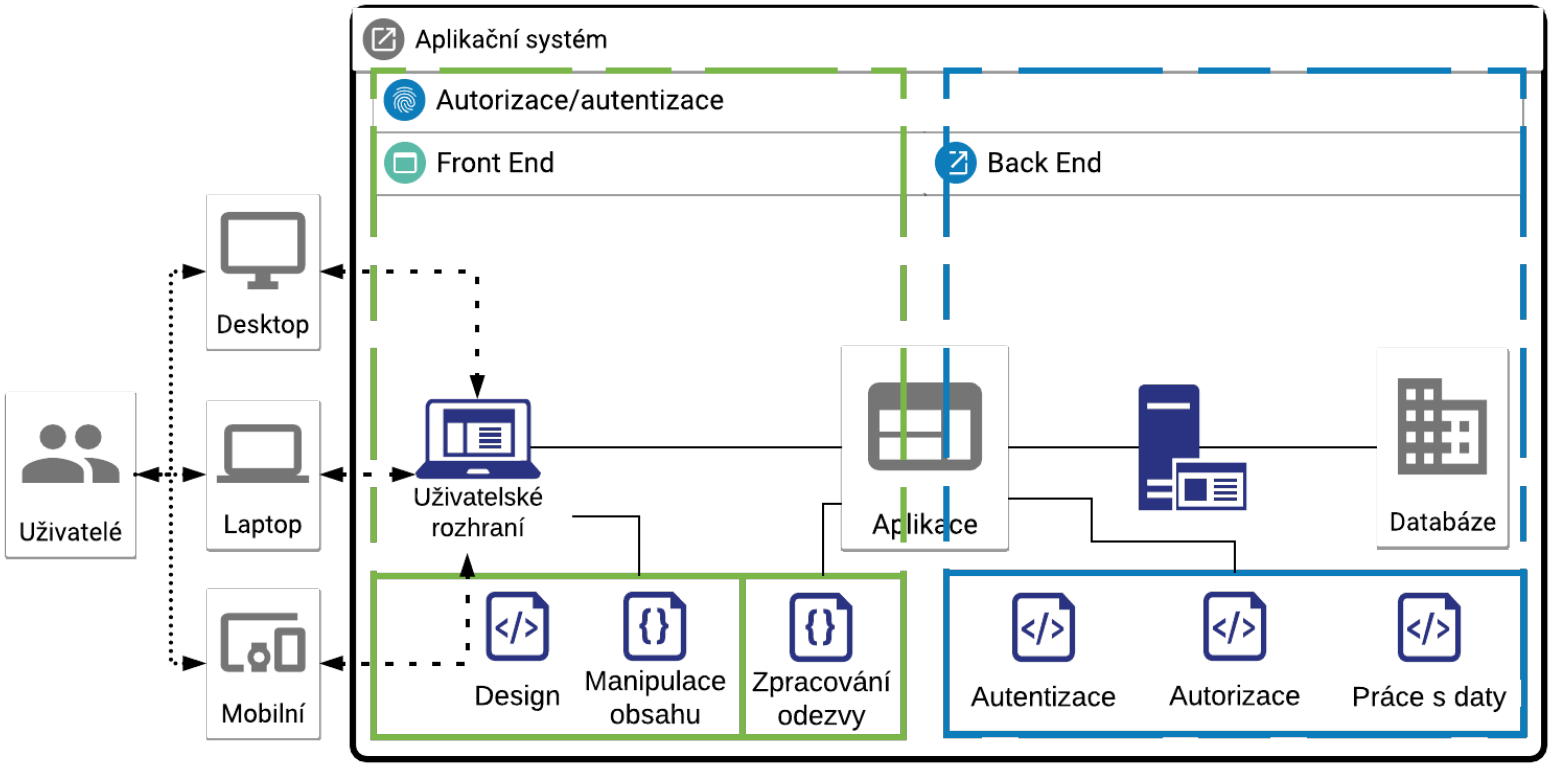
plná čára: Logické spojení (například konfigurace konfiguruje databázi).

přerušovaná čára: Logická komunikace.

tečkovaná čára: Přímá komunikace (například uživatel s pracovní stanicí).

ikony a obrázky: Jsou jen ilustrativní, jejich význam vyplývá z textu u nich uvedeného (nemají sémantický význam).

Obrázek 3.1: Ilustrace struktury systému



3.2 Metodiky

Jak je již řečeno v sekci 2.6 - Metodiky návrhu a vývoje, pokud na systému spolupracuje více vývojářů, je před začátkem implementace systému jako takového, zvolit vhodnou organizaci týmu a metodiku týmového návrhu a vývoje.

Na tomto projektu sice spolupracují jenom dva vývojáři, ale vyvíjí zvlášť front-end a back-end. A jak je též již zmíněno v sekci 2.6, takový vývoj si žádá určitou míru spolupráce.

3.2.1 Organizace týmu

Vzhledem k velikosti týmu není moc složité rozhodování ohledně jeho organizace. V tomto případě se jeví jako naprosto logické volit z nestrukturovaných týmů. To též potvrzuje například [9].

Volit tedy můžeme jak je zmíněno v sekci 2.6.1 - Vývoj a týmová spolupráce (v tabulce 2.8) z :

- „Osamělý vlci“,
- „Horda“,
- „Demokratická skupina“

Z těchto se nejvíce hodí „demokratická skupina“. Vhodná by byla i „horda“, ale demokratická skupina lépe vystihuje přístup, na kterém jsme se s vývojářem back-endu shodli.

Pod touto nestrukturovanou organizací týmu tedy tento vývojový tým vyvíjí tuto aplikaci. Toto rozhodnutí vychází částečně z informací poskytnutých v [9].

3.2.2 Volba metodiky

Když tým ví pod jakou organizací bude pracovat, pak je potřeba též vybrat metodiku vhodnou pro vývoj projektu dané povahy.

Jak je již zmíněno v sekci 2.6.2 - Metodiky vývoje software, pro vývoj projektu stejné povahy jako je projekt pro Laboratoř inteligentních vestavných systémů (Livs), nejsou vhodné rigorózní metodiky (také někdy nazývané jako klasické) z důvodu dlouhého plánování a zaměření na větší vývojové týmy (většinou).

U agilních metodik se na druhou stranu velice hodí aktivní účast zadavatele/zákazníka, neboť to je přesně to, co je za potřebí pří realizaci tohoto projektu. Proto je pro tento projekt vhodnější metodika agilní.

Z agilních metodik jež jsou uvedeny v sekci 2.6.2 - Metodiky vývoje software (v tabulce 2.8), se pak nejvíce hodí Extrémní Programování (XP), respektive její lehce upravená verze pro konkrétní požadavky tohoto projektu.

3.2.2.1 Extrémní programování (XP)

Tato metodika, podle [9], upřednostňuje následující:

- Týmovou spolupráci a interakci před formálními procesy a nástroji.
- Fungující software před obsáhlou, komplexní dokumentací.
- Spolupráci mezi zákazníkem a vývojáři před specifikacemi, zadáními, dohodnutými kontrakty.
- Rychlou adaptaci na změny v zadání před dodržováním předem stanovených pravidel.

Z této metodiky se také velice hodí vývoj v iteracích a práce ve dvojici. Iterace se hodí z toho důvodu, že každý týden se koná plánovaná schůzka se zadavatelem. Jak je uvedeno výše, metodika použita pro vývoj tohoto systému se pevně nedrží definice. To se například projevuje na tom, že práce ve dvojici probíhá jen občas.

Sdílení kódu se také velice hodí. Sdílení kódu a technologie k jeho realizaci použité jsou probrány v sekci 3.3 - Technologie.

Také je v úpravě této metodiky zrušeno omezení na počet hodin týdně a testování je prováděno průběžně po vytvoření nové funkcionality systému/aplikace.

3.3 Technologie

Pro vývoj navrhl back-end vývojář framework Symfony. Tento php framework je stabilní, široce podporovaný a používaný velkým počtem současných webových vývojářů. S tímto frameworkem má vývojový tým tohoto projektu zkušenosti z předešlých projektů. Jako databázový systém byl navrhnut systém MySQL. Tyto dvě technologie se staly základem pro stavbu systému pro tento projekt.

Volba těchto dvou technologií záležela převážně na back-end vývojáři. Proto nejsou uvedeny v analýze. Symfony ale lehce zasahuje i do front-endu, a tak je vhodné alespoň popsat k čemu slouží.

Symfony
Pro částečné upřesnění je potřeba uvést, že je to webový aplikační framework. Poskytuje set php komponent pro tvorbu webových stránek nebo aplikací, širokou komunitu a detailně zpracovanou dokumentaci, díky které je vývoj značně jednodušší.

Symfony komponenty jsou oddělené a znovupoužitelné.

3.3.1 Sdílení kódu

Pro sdílení kódu existuje několik možností. Nejznámějšími jsou například git a svn. Vzhledem k potřebám tohoto projektu je jednoznačným vítězem git. Díky faktu, že oba vývojáři již s gitem pracovali, tato volba také ušetří čas potřebný pro výboj. Proto se tato sekce ani nevyskytuje v analýze.

3.3.2 Základní technologie

Jak je již uvedeno v sekci 2.7.1 - Nezbytné pro vývoj webové aplikace, bez technologií jako je css a html se při vývoji webového front-endu, nebo konkrétněji webového rozhraní, vývojář daleko nedostane.

Z tohoto důvodu jsou tyto dvě technologie jasnou volbou, jako základní technologie pro implementaci webového rozhraní.

3.3.2.1 JavaScript

U JavaScriptu už to není tak jednoduché. V sekci 2.7.1.1 - Nevhodné použití, se analýza zmiňuje o faktu, že pokud lze určitou funkcionalitu realizovat pomocí jiného jazyka, než je JavaScript, tak by se tak mělo udělat.

I přes fakt, že se toto tvrzení vyskytuje v analýze tohoto projektu, je to pravda jen částečně. Proto jsem se rozhodl pro kompromis.

Kompromis vzhledem k povaze projektu
Tento projekt, jak již bylo zmíněno v úvodu a analýze, bude pokračovat i po dokončení této práce. Proto jsem se rozhodl pro následující řešení:

- Aplikace bude v základu plně funkční bez JavaScriptu.
- JavaScript bude realizován jako dodatečné vylepšení pro již funkční aplikaci.
- K pozdější realizaci JavaScriptu bude pravděpodobně využit framework.

3.3.3 Frameworky

V této sekci se práce zaměřuje na návrh a volbu vhodných frameworků pro jednotlivé technologie a části aplikace.

3.3.3.1 JavaScript

Jak říká sekce 3.3.2.1 - JavaScript, volba JavaScript frameworku je relevantní až pro konečnou fázi projektu. Tento framework je však možné zvolit i předem.

Ze sekce 2.7.2 - Frameworky (front-endové), se nejvíce hodí framework React, díky jeho popularitě a tudíž i jednoduššímu řešení problémů, a také díky nepříliš náročnému principu vývoje v rámci jeho prostředí.

Výběr tohoto frameworku není v této fázi finální, a tak tu není detailně probrán.

3.3.3.2 HTML,CSS,JS

Jasnou volbou pro framework v tomto ohledu je, již v sekci 2.7.2 zmíněný, Bootstrap. Tento framework je též velmi rozšířený. To se hodí obzvláště pokud se má projekt dále předávat dalším vývojářům. Zaručuje též dobrou kompatibilitu s webovými prohlížeči.

Přesto jsem však tento framework zvolil jen pro návrh základního designu a následně předělal za pomoci Flexboxu.

Toto rozhodnutí je uskutečněno kvůli nutnosti pevné kontroly nad tím, co se s jednotlivými elementy děje a kvůli nutnosti tvorby velké části designu s velice specifickým chováním. Také by nebylo z mého pohledu vhodné použít takovouto technologii pro tuto práci, neboť velice ulehčuje práci bez nutné znalosti její vnitřní funkcionality.

3.3.3.3 Templating - twig

V rámci Symfony je doporučovaný šablonovací (templating) systém (engine) Twig, který je prakticky nejlepší volbou, pokud chceme plně využít všech možností frameworku Symfony.

Výhodou tohoto šablonovacího systému ve spojení s frameworkem Symfony je též to, že přímo v dokumentaci Symfony je velké množství návodů a podrobně popsaných postupů, jak tyto dvě technologie dohromady používat.

Proto jsem jej též zvolil jako hlavní šablonovací systém pro tento projekt.

Twig
Opět je dobré uvést krátky detailnější popis. Jako takový je Twig flexibilním, rychlým a spolehlivým šablonovacím systémem (templating engine) pro PHP. Jednoduše řečeno, twig pomáhá organizovat, zjednodušit a zpřehlednit PHP kód jako každý jiný templatovací engine.

Twig však poskytuje velké množství funkcí, jež některé templatovací enginy neposkytují. Toto platí obzvláště právě ve spojení se Symfony.

3.4 Návrh designu

Pokud se navrhuje design uživatelského rozhraní většího projektu je vhodné před samotnou implementací designu vytvořit návrh nebo náčrt rozložení jednotlivých prvků a informací.

To obzvláště platí, pokud se má dané uživatelské rozhraní měnit v závislosti na velikosti obrazovky, nebo pokud obsahuje složitější informační struktury. Například hodně provázané informace. O této problematice se též zmiňuje [16].

Tyto návrhy nebo náčrty lze vytvářet různými způsoby. Jaký způsob si designér uživatelského rozhraní zvolí je na jeho vlastním úsudku. Většinou se však rozhoduje dle časových omezení na tvorbu projektu a rozpočet, jak se též zmiňuje v [16].

Příklad možného postupu:

1. náčrt,
2. wireframe,
3. wireframe (high-def),
4. vizuální zobrazení,
5. kódování.

A mnoho dalších o kterých se zmiňuje například [17]. O tom co je to wireframe se zmiňuje sekce 3.4.1 - Wireframing.

S ohledem na časové omezení tohoto projektu, respektive této práce, je použit jednodušší a rychlejší postup.

Použitý postup (opakovaný):

1. wireframe,
2. kódování,
3. úprava (na základě vizuálního zobrazení dat),
4. kódování.

3.4.1 Wireframing

Wireframe je návrh strukturálního rozložení stránky tvořený z jednoduchých sémantických tvarů, čar, případně vyznačení funkčních prvků nebo důležitých částí webu. Většinou se používají v počátečních fázích návrhu uživatelského rozhraní.

V tomto projektu je to trochu složitější, neboť byly wireframy navrhovány z počátku s malou znalostí dat, jež je potřeba zobrazovat, a tak je bylo potřeba průběžně upravovat a přizpůsobovat zobrazovaným datům.

Ve wireframech od počátku zůstalo hlavní rozložení navigačních prvků v rozhraní, nebo také hlavička a zápatí rozhraní. V této sekci budou zobrazeny jen poslední verze wireframů, neboť původní návrhy nejsou vůči současnému řešení aktuální.

Vzhledem k časovému omezení nebyly vytvořeny wireframy pro všechny stránky, ale jen pro ty klíčové.

3.4.1.1 Desktopové wireframy

Zde jsou k dispozici wireframy pro desktopová (stolní) zařízení. To znamená pro rozlišení vyšší jak 750 pixelů na šířku.

Všechny tyto wireframy byly navrhovány pro rozlišení 1920x1080 neboť, jak vyplývá ze sekce 2.3.2 - Rozlišení obrazovky v České Republice a 2.3.4 - Rozlišení obrazovky celosvětově, nejčastěji používaná desktopová rozlišení jsou buď vyšší nebo podobně veliká, jako právě rozlišení 1920x1080. Toto rozlišení je také známe jako full-hd a s výhledem do budoucna je rozumné se zaměřit, i s ohledem na téma projektu, spíše na vyšší rozlišení.

Při navrhování byl brán ohled i na dostupnost a dodržení Nielsonovy heuristiky. Tato heuristika je probrána v sekci 3.5 - Design a Nielsonova Heuristika.

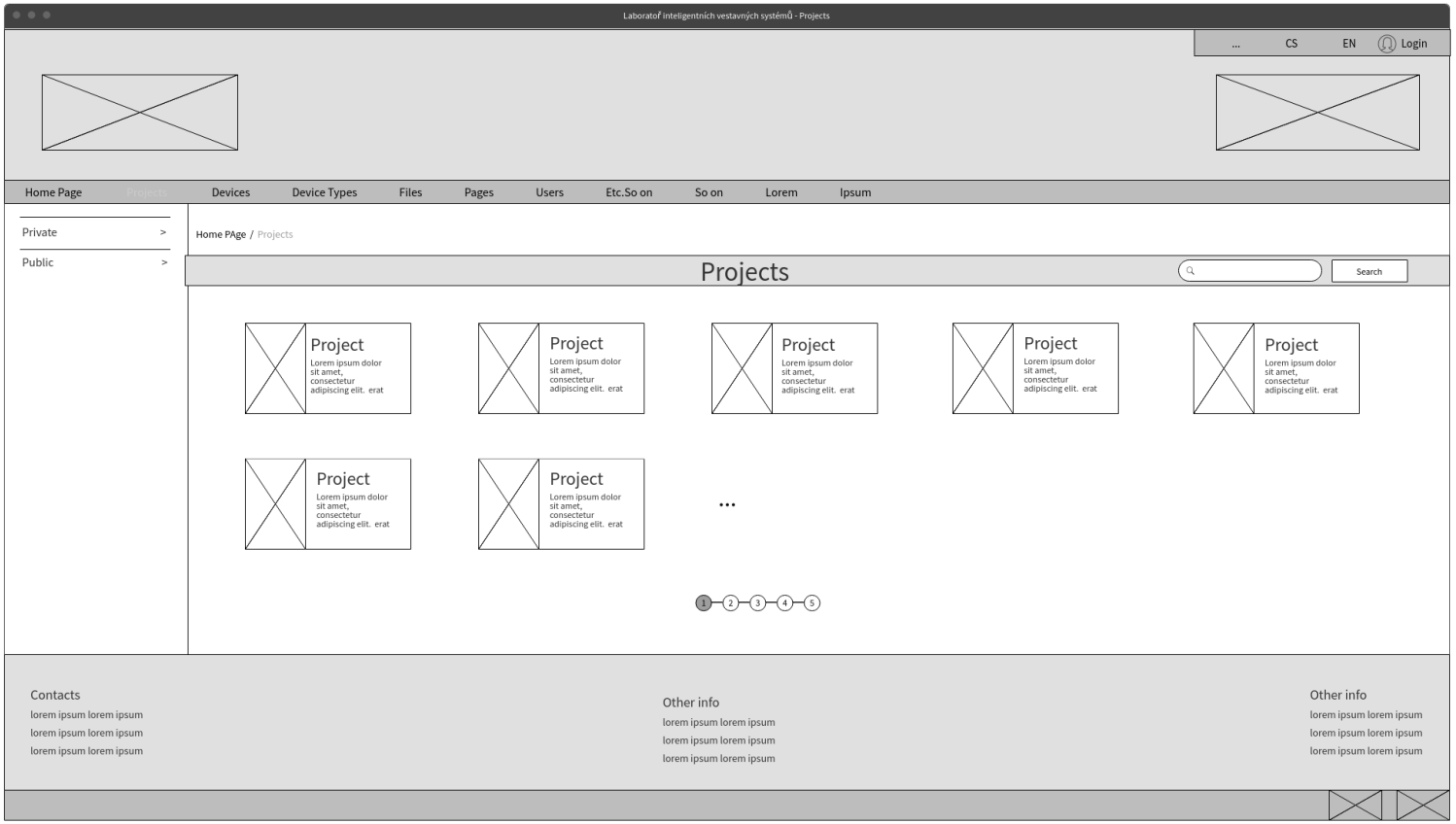
Následující dva wireframy jsou návrhem struktury rozhraní, nebo-li v tomto případě stránek, pro zobrazování veřejně dostupných a privátních projektů.

První wireframe (obrázek 3.2) je tedy návrhem pro veřejné projekty a druhý wireframe (obrázek 3.3) je návrhem pro privátní projekty. Tyto návrhy jsou pak také použity pro ostatní entity, které mají obdobnou informační strukturu. To znamená, že například další privátní entity nejvyšší úrovně (nejsou potomkem jiné entity) budou mít stejné rozhraní, jako privátní projekty.

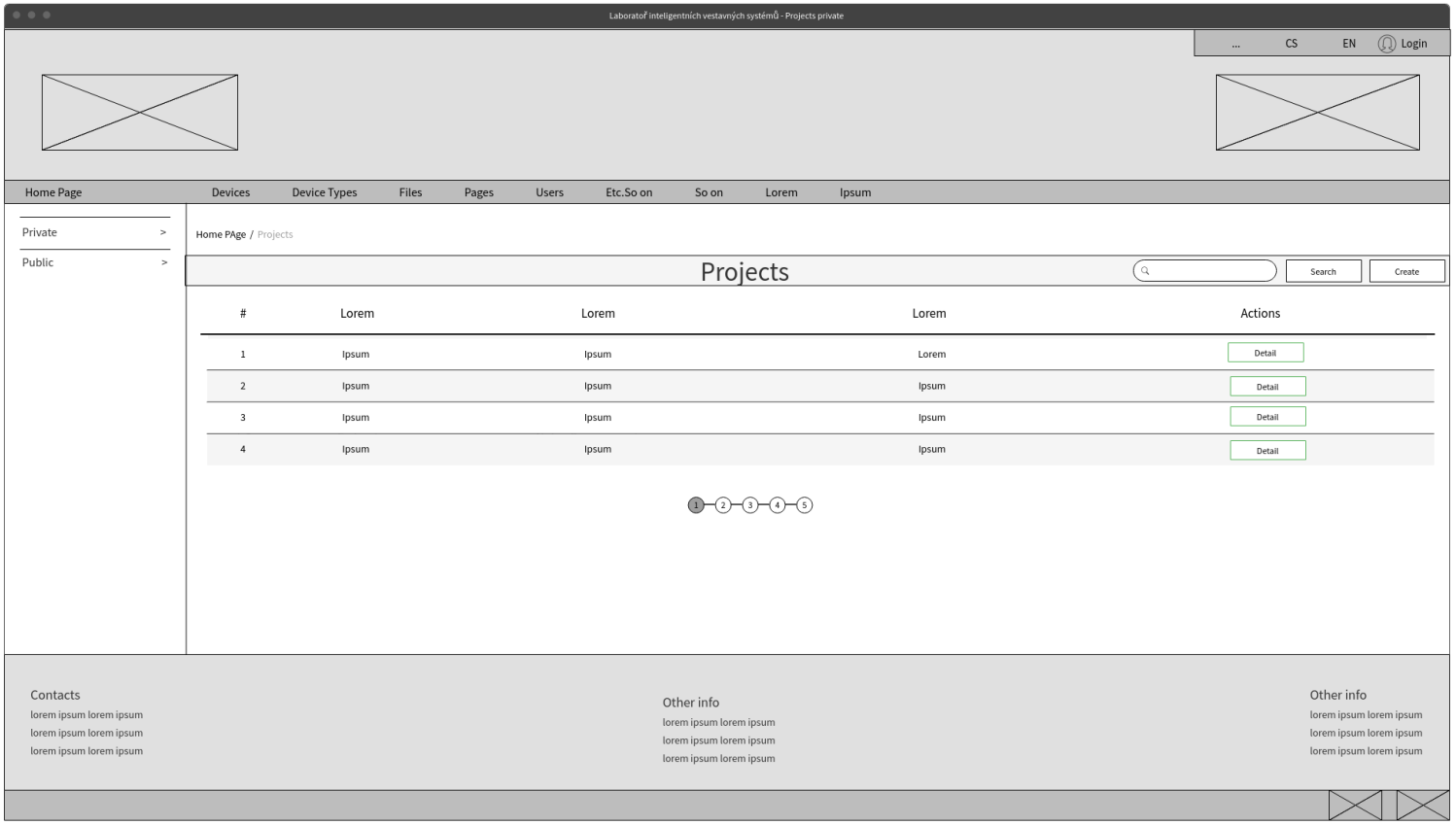
Třetí wireframe (obrázek 3.4) zobrazuje návrh rozhraní, nebo-li stránky pro „komponenty“ určité entity. Komponentou je míněno libovolně zanořená entita, která je potomkem některé z entit nejvyšší úrovně.

Čtvrtý wireframe (obrázek 3.5) zobrazuje návrh rozhraní, nebo-li stránky pro detail určité entity. Tento detail bude přítomný u většiny entit, jejichž obsah je potřeba organizovat.

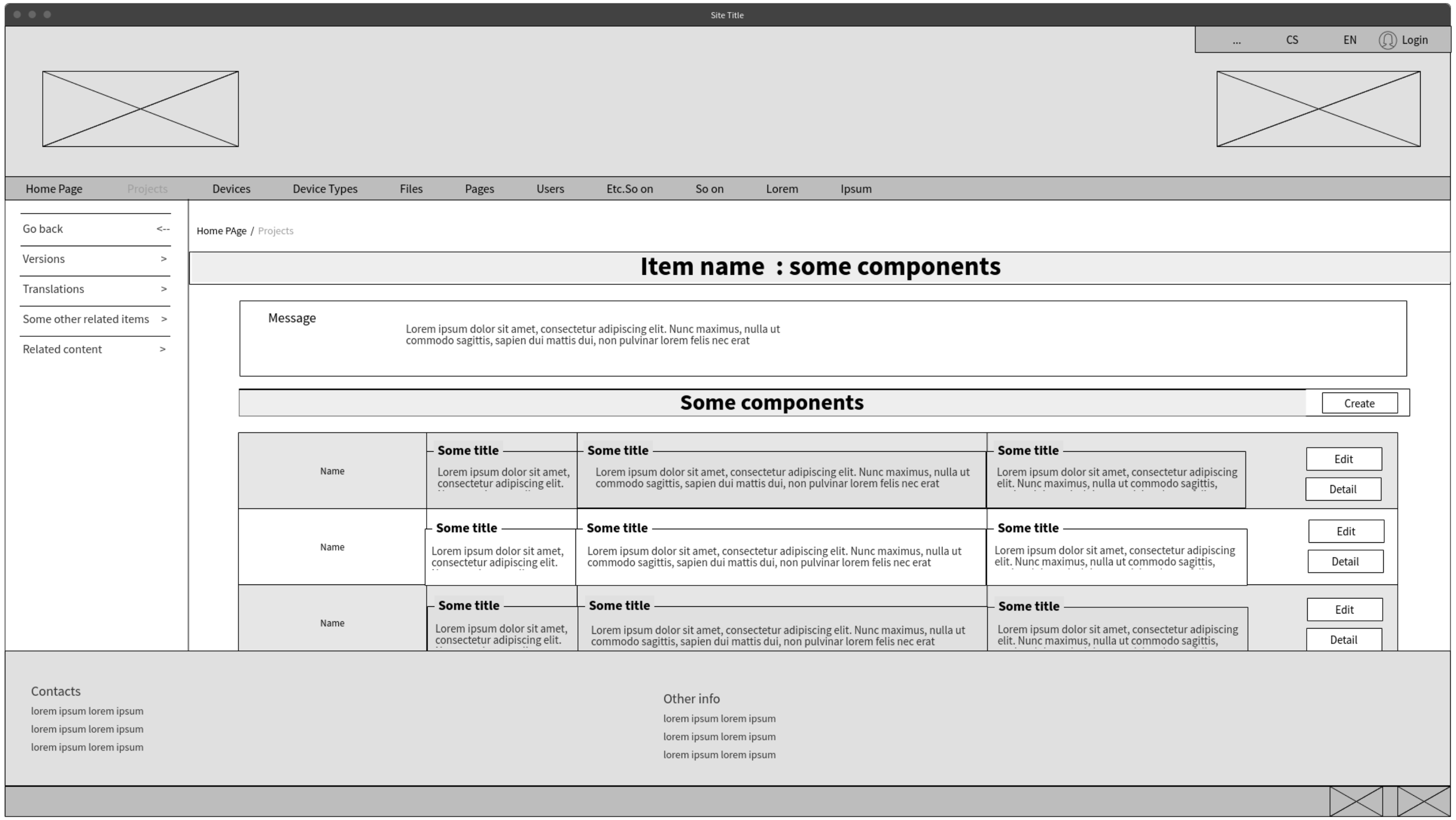
Obrázek 3.2: Wireframe stránky veřejných projektů



Obrázek 3.3: Wireframe stránky privátních projektů



Obrázek 3.4: Wireframe stránky komponent entity



Obrázek 3.5: Wireframe stránky detailu entity

Ve wireframech se vyskytují prvky, které je potřeba vysvětlit:

Rámeček se vsazeným nadpisem:
Tento element vyznačuje obsah, jež bude v budoucím rozvoji systému možné rozšiřovat a upravovat jeho umístění nebo způsob zobrazení na stránce. Z tohoto důvodu se u některých těchto elementů vyskytují ovládací prvky, jež v současném stavu aplikace nejsou přítomny, neboť by vyžadovali použití JavaScriptu k použitelnému využití. Tímto se však dále zabývá kapitola 4.1.1 - Implementační technologie.

Tři tečky: Naznačují opakující se obsah stejné povahy, jako jim předchází. Například další projekty, nebo řádky v tabulce.

Navigace: Navigace, jako jsou menu a drobečková navigace, slouží jen pro orientační zobrazení a nezobrazují aktuální stav aplikace.

3.4.1.2 Mobilní wireframy

V této sekci jsou zobrazeny wireframy pro mobilní zařízení, respektive jeden wireframe, nebo-li wireframy pro rozlišení menší jak 750px šířky.

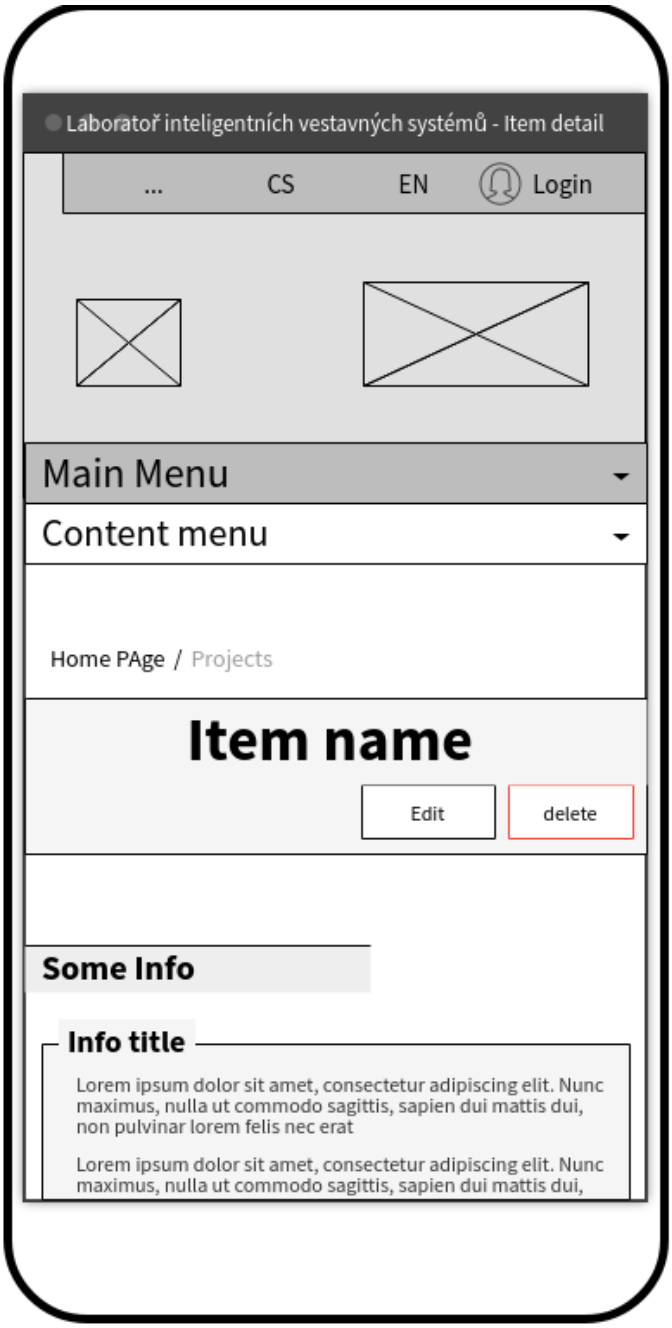
Z důvodů časového omezení a počtu stránek jsem zvolil tvorbu pouze jednoho wireframu pro mobilní zařízení, kde je zobrazeno hlavně rozložení hlavičky. Toto rozložení je velice důležité, neboť hlavička nesmí zabírat moc prostoru, ale zároveň by neměla být nepřehledná.

Návrh wireframu pro mobilní zařízení je vytvořen v rozlišení 360x640, neboť je to dle sekce 2.3.2 - Rozlišení obrazovky v České Republice jedno z nejpoužívanějších rozlišení.

Wireframe (obrázek 3.6) zobrazuje návrh rozhraní, nebo-li stránky pro detail určité entity. Z obsahu samotné entity je vidět jen nadpis a část prvního odstavce, proto je v následujícím odstavci krátce popsáno rozložení obsahu při rozlišení pod 750px na šířku.

Popis rozložení obsahu
Jednotlivé části obsahu entit a komponenty jsou při mobilním zobrazení vždy roztaženy na celou šířku obrazovky. Pokud se však více takovýchto částí i po roztažení na maximální šířku jejich obsahu vejdou vedle sebe, zobrazí se tak. Všechny takovéto části jsou seřazeny pod sebou.

Obrázek 3.6: Wireframe stránky detailu entity pro mobilní zařízení



Výše zobrazené wireframy jsou ve zmenšené verzi. V plné velikosti jsou k dispozici v přílohách, nebo-li v přiloženém Kompaktním disku (CD) (viz. Obsah přiloženého CD).

3.5 Design a Nielsonova Heuristika

V této sekci je probrána Nielsonova Heuristika a její vliv na design uživatelského rozhraní v tomto projektu.

Tato sekce se nachází až po sekci Návrh designu právě proto, aby objasnila některá rozhodnutí uvedená ve zmíněné sekci.

Zjednodušeně řečeno, Nielsonova heuristika je sada deseti pravidel pomocí kterých expert (nejlépe někdo jiný než navrhoval design) zkontroluje správnost a použitelnost uživatelského rozhraní z pohledu uživatele. Z toho vyplývá, že těmito pravidly by se měl designér při tvorbě/návrhu uživatelského rozhraní řídit, pokud má za cíl dosáhnout funkčního a uživatelsky přívětivého rozhraní.

Těchto deset pravidel je uvedeno níže a řešení problémů je vždy uvedeno u každého z nich.

Nejprve je však též vhodné zmínit, že právě rozhodnutí nepoužít v počátečním stavu aplikace technologii JavaScript (detailněji v sekci 3.3.2.1), částečně též vychází ze snahy dodržet tato pravidla ve všech stádiích a stavech aplikace.

Následující pravidla lze též nalézt v [18].

Pravidla:

1. Viditelnost stavu systému.

    - Při návrhu designu je vzhledem k povaze informací a absenci JavaScriptu aplikace vždy v jednom stavu (není mezi stavy).
    - Proto je v každém stavu aplikace jako jeho identifikace viditelná drobečková navigace, unikátní označení stránky reprezentující stav (například titulek), nebo případně flash zpráva oznamující změnu stavu.
    - Jediný problém nastává při vykonávání skriptů (dříve zmíněné výkonné funkce), kdy je bez pomoci JavaScriptu zobrazení stavu realizováno zpětnou vazbou od skriptu ve formě zprávy na uživatelském rozhraní.

2. Shoda mezi systémem a realitou.

    - Zde nastává první problém. Jelikož se jedná o systém pro neprozkoumanou problematiku, kde ani nejsou k dispozici přistroje, pro jejichž evidenci je navrhován, je poměrně těžké zjistit jaká je vlastně realita manipulace s takovými přístroji.

- Tento problém též vede na problémy s překladem do češtiny. V aplikace se vyskytují některé výrazy, které v překladu do češtiny mají více významů, neboť jsou to například odborné výrazy. Tyto překlady jsou též předmětem uživatelského testování.
- Části aplikace, jež nejsou vázány na zmíněné přístroje (vestavné systémy), jsou na druhou stranu navrhovány právě tak, aby reflektovali realitu. Například použití všeobecně známých ikon k určitým akcím.

3. Minimální zodpovědnost.

- Přesto, že je tento systém stavěný tak, aby neoprávněná osoba nemohla provést nevratnou operaci (Access Control Lists (ACL)), i tímto je potřeba se zabývat.
- V aplikaci jsou u každých nevratných akcí vytvořeny dotazovací zprávy. V této zprávě je uvedeno, jestli chce uživatel opravdu dokončit akci nebo se vrátit zpět.
- Ne všechny akce lze univerzálně vrátit, ale tyto akce se dají vždy vrátit například při vytvoření nového obsahu, odstraněním vytvořeného obsahu.
- Pokud však uživatel potvrdí některou akci, v jádru aplikace není vedena historie takových akcí a tak potvrzovací dialogy značí, pomocí historie, nevratné akce.
- Výjimkou jsou ovšem znovu skripty (výkonné funkce). Tyto akce jsou mimo kontrolu front-endu i back-endu. Jejich tvůrce musí nejprve funkce poskytnout, aby se daly zakomponovat do systému. V rámci aplikace se však lze vždy vrátit zpět z vykonávání skriptu.
- V žádném stavu aplikace, se bez příkazu uživatele nevykonávají samočinné akce. Uživatel má vždy kontrolu.

4. Shoda s použitou platformou a obecnými standardy.

- Zde není vzhledem k povaze projektu problém.
- Webová stránka je designovaná tak, aby i jako webová stránka vypadala a chovala se tak.
- Uživatelské rozhraní se nesnaží vylepšit grafickou stránku nebo funkcionalitu pomocí speciálních grafických prvků.
- Je zde snaha o maximální využití zobrazované plochy informačním obsahem.
- V rozhraní je snaha o minimální používání zkratek a akronymů.

5. Prevence chyb.

- Prevence chyb částečně souvisí i s výše zmíněným dotazováním před provedením nevratné akce. Právě tyto dotazovací dialogy jsou také prevencí chyby.
- Před zobrazením dialogu však uživatel musí vyplnit správně data jež chce vytvářet/upravovat/mazat. V případě, že je nezadá správně, objeví se flash zpráva nebo upozornění na špatně zadaná pole.

6. „Kouknu a vidím“.
   - V rozhraní nikdy není žádná akce, kterou by mohl uživatel vykonat aniž by k ní měl přístup.
   - Menu jsou přístupná ve všech stavech aplikace. V současné verzi vždy řazena od hlavičky stránky. Nepohybují se s uživatelem po stránce.
   - Kde se uživatel nachází je též vyznačeno, jak už bylo zmíněno, pomocí drobečková navigace a titulku na stránce.
7. Flexibilita a efektivita.
   - Jak už bylo též zmíněno, uživatel, který prvek vidět nepotřebuje, jej nevidí.
   - Klávesové zkratky nebo podobné funkce zatím aplikace nepodporuje. Případné řešení této problematiky bude řešeno až v pozdějších stádiích vývoje.
   - V rozhraní je minimální počet ovládacích prvků. Další se navrhnou a implementují po zpětné reakci uživatelů.
8. Estetický a minimalistický design.
   - Toto je základním zaměřením tohoto rozhraní. Vzhledem k počtu informací a požadavkům je potřeba mít motivující, estetický, ale zároveň informačně efektivní design.
   - Výše zmíněný bod však též naráží na problém relevantních informací. Je těžké odfiltrovat informace relevantní a irelevantní pro uživatele, pokud ještě neexistují uživatelé (jak je tomu zde).
   - Proto se v designu snaží rozhraní graficky zvýraznit důležité části. To je však problematické v případě mobilního zobrazení. Kde postranní menu zabírá moc místa, a tak na úkor zmatení uživatele je přesunuto do hlavičky.
9. Pomoc uživateli s pochopením a napravením vzniklé chyby.
   - Toto je velice důležitá problematika. Proto se chybám snaží uživatelské rozhraní předcházet.

- V případech kdy chyba nastane, je uživateli popsána chyba pomocí flash zprávy. Případné řešení je též zobrazeno v této zprávě.
- Pokud celý systém/aplikace selže, je uživateli též zobrazena chybová hláška popisující chybu, případně jak může pokračovat dále. Nepoužívají se, cílené skupině, neznáme hlášky.

10. Help a dokumentace.

- Tato část systému je otázkou budoucího rozvoje.
- Není vhodné sepisovat dokumentaci k měnícímu se rozhraní. V daném časovém rozmezí by to znamenalo zbytečné zdržení.

3.6 Architektura systému

V této sekci je probrán střet front-endu a back-endu v architektuře systému projektu. Také je zde krátký popis na jaké architektuře je aplikace/systém vybudován.

K oddělení dat od jejich prezentace je v aplikaci použit návrhový vzor Model View Controller (MVC). Symfony framework nijak nebrání takovémuto oddělení logiky dat od jejich prezentace.

3.6.1 Model View Controller (MVC)

V poslední době je to jeden z nejrozšířenějších návrhových vzorů. Jeho popis je poměrně jasný : „*Architektura MVC dělí aplikaci na 3 logické části tak, aby je šlo upravovat samostatně a dopad změn byl na ostatní části co nejmenší. Tyto tři části jsou Model, View a Controller. Model reprezentuje data a business logiku aplikace, View zobrazuje uživatelské rozhraní a Controller má na starosti tok událostí v aplikaci a obecně aplikační logiku.*“[19]

Následující obrázek popisuje výše uvedené provázání jednotlivých částí a jejich interakci.

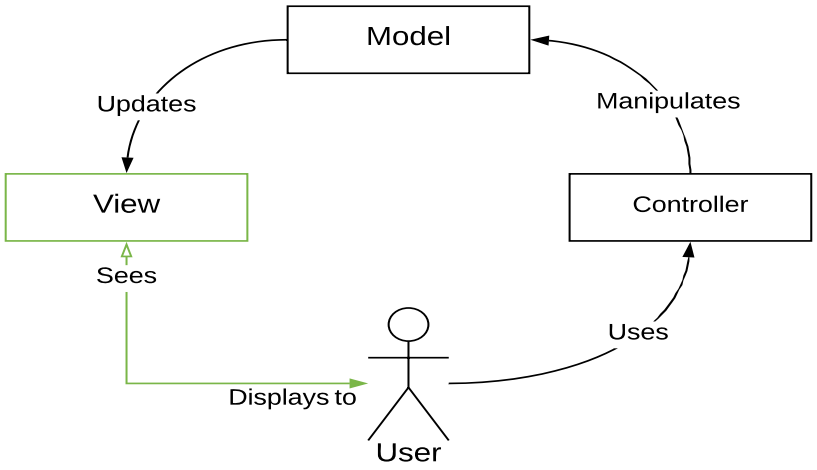
3.6.2 Provázanost front-endu a back-endu

Z předešlé podsekce, víme jak vypadá struktura aplikace postavené na MVC návrhovém vzoru. Tyto části však nespadají vždy přesně pod front-end nebo back-end, i přes fakt, že se jedná jenom o prezentační část aplikace. Model spadá převážně pod back-end a View převážně pod front-end. Controller je však přesně mezi front-endem a back-endem.

Proto jsme s vývojářem back-endu dohodli spolupráci právě na této části aplikace. Toto je taky jedním z důvodů proč je nutná spolupráce a sdílení kódu při tvorbě tohoto systému.

Z tohoto důvodu se při vývoji aplikace často stane, že část Controlleru je psaná back-end vývojářem a část front-end vývojářem.

Obrázek 3.7: Model View Controller (MVC)



Zpravidla to bývá tak, že back-end vývojář připravil strukturu a abstraktní entity potřebné ke správnému fungování Controlleru a front-end vývojář pak vyplňuje funkcionalitu Controlleru nebo dodatečné potřebné funkce pro zobrazování částí aplikace/systému. Na druhou stranu Controllery *api* a *security* jsou v současném stavu napsány back-end vývojářem.

V některých případech je tomu však i tak, že akci v Controlleru píše front-end vývojář.

Jak je výše již zmíněno, Model a View už ale jsou prakticky jasně oddělené, a tak lze určit přesně, které z těchto částí se převážně věnuje právě tato práce. Tou částí je, jak již vyplývá z textu, právě View (a samozřejmě Controller).

Kapitola 4

Realizace

V této kapitole se práce zabývá konkrétními kroky, jež vedly k zhotovení aplikace a detailnějším pohledem na některé jeho části z pohledu samotného vývoje. Je možné, že v některých sekcích bude zmíněna i spolupráce s back-end vývojářem a testování jádra systému/aplikace. Převážně se tato kapitola však věnuje front-endu.

Také se zde práce zabývá expertním a uživatelským testováním.

4.1 Design

Tato část realizace probíhala v oddělené větvi (frontend) verzovacího systému git. Repozitář pro projekt tedy byl již založen, ale spolupráce s vývojářem back-endu probíhala minimálně.

Přípravný krok
Tento krok je z pohledu návrhového probrán již v sekci 3.4.1 - Wireframing. Zde jde však o popis realizace návrhu pomocí wireframu.

Wireframy byly tvořeny pomocí nejmenovaného webového nástroje, který neumožňoval držet dostatečný počet wireframů online, a tak byla jejich tvorba ukládána na disk.

Toto ukládání probíhalo mimo git repozitář projektu po celý průběh projektu a proto bohužel nejsou k dispozici původní verze návrhů.

Původní návrhy byly postupně upravovány viz. následující kroky.

První krok
K realizaci uživatelského rozhraní, z pohledu implementace (ne návrhu), je realizace jeho designu. Jak je již zmíněno v sekci 3.4.1 - Wireframing, návrh byl zhotoven pomocí wireframingu v počátku vývoje aplikace (viz přípravný krok).

Poté za pomoci základních technologií pro tvorbu webové aplikace (viz sekce 2.7.1) a frameworku Bootstrap (viz sekce 2.7.2.1) byl vytvořen vizuální interaktivní návrh aplikace. Do tohoto návrhu byla manuálně vložena částečná data z jádra systému (konkrétněji z databáze).

Tento původní vizuální návrh je možné nalézt ve výše uvedené větvi git repozitáře.

Druhý krok
Na základě vizuálního zobrazení dat byla provedena úprava designu přímo v kódu a až poté v samotných wireframech.

Po dokončení těchto úprav byl design předveden zadavateli v podobě interaktivního rozhraní. Zadavatel jeho podobu odsouhlasil.

Třetí krok
Tento krok je spíše dodatkem. Úpravy designu totiž neprobíhaly jen při fázi implementace návrhu/designu. Tyto úpravy, nebo-li krok dvě, též nadále probíhaly i při integraci do jádra systému a navazujícím vývoji.

4.1.1 Implementační technologie

V této sekci jsou uvedeny technologie, které nebyly brány v potaz během analýzy ani návrhu a bylo pro ně rozhodnuto až při implementaci. Konkrétněji zde jsou technologie použité pro rychlejší a efektivnější realizaci designu.

4.1.1.1 SASS/SCSS

Sass je jazykem pro stylizování webových stránek stejně jako například css s tím rozdílem, že tento jazyk rozšiřuje právě uvedený stylovací jazyk css. Scss je pak obdobnou syntaxí jazyka sass.

Díky tomuto jazyku lze rychleji, kompaktněji a efektivněji definovat a používat styly.

S tímto jazykem však též přichází zodpovědnost nezneužívat jeho funkcí. Často tento jazyk svádí k hlubokému zanořování elementů, což se poté projevuje na výkonu uživatelského rozhraní.

Uvedená technologie má však jednu nevýhodu a to tu, že prohlížeče nepřevádějí syntaxi jazyka automaticky na jimi zobrazitelné styly. Řešením tohoto problému je například následující sekce.

4.1.2 Gulp

Gulp je takzvaný „task runner“. To znamená, že umožňuje automatizované spouštění určitého procesu („task“) na základě zadané podmínky/pravidla.

Hodí se například při automatické kompilaci sass souborů. V tomto případě je též použit k automatické prefixaci stylů a minimalizaci výsledného „style sheetu“, nebo-li stylové šablony.

4.2 Integrace, vývoj, implementace

Tato sekce se zaměřuje na popisování postupu při integraci, následném vývoji a implementaci funkcionalit a komponent (back-end i front-end) uživatelského rozhraní, do jádra systému/aplikace.

Po dokončení práce na designové stránce, nebo-li ve fázi, kdy už bez testování na datech nebylo možné vylepšovat design a rozložení prvků v uživatelském rozhraní, byly složky a součásti designu přemístěny do hlavní větve git repozitáře.

V této fázi vývoje systému už byla nezbytná spolupráce s vývojářem back-endu.

V jednotlivých sekcích se vyskytují odkazy na souborovou strukturu. Tato struktura je krátce popsána v sekci 4.4 - Souborová struktura systému a vždy začíná relativně od kořenové složky aplikace.

4.2.1 Controllery

Tato podsekce se nejprve zabývá úvodní integrací designu/návrhu rozhraní s jádrem aplikace, specificky tedy s Controllery.

Druhá část této podsekce se pak zabývá následným vývojem a implementací dalších aplikačních částí v Controllerech.

Controllery pro jednotlivé entity se nacházejí v souborové struktuře takto:

```
src
└── RobotManagementSystemBundle
    └── Controller
        └── Entity
```

4.2.1.1 Integrace

Aby bylo možné implementovat design/návrh rozhraní na reálných datech, je nejprve potřeba implementovat akce Controllerů korespondujících s danými entitami.

Při implementaci je udržována snaha zbytečně nezanášet akce kódem tak, aby bylo možné poznat, co se v jednotlivých akcích vykonává za příkazy. Zde se též objevují první problémy spolupráce nad Controllery.

V několika případech bylo z důvodů nedostatků v jádru potřeba předělávat některé akce (i několikrát). Proto je možné, že se v některých z nich objevuje lehká inkonzistence. Ze stejného důvodu se též integrace a následný vývoj front-endu výrazně zpomalil.

Při implementaci Controllerů narazíme na další potřebný krok a tím je zvolení templatu, nebo-li šablony. V tomto případě jde o šablonu psanou ve

zvoleném jazyku Twig. Vizuální návrh do této doby nebyl realizován pomocí těchto šablon. Na to navazuje následující sekce.

4.2.1.2 Vývoj a implementace

Pro vývoj, který následuje integraci, je asi jedinou důležitou věcí zmínka o nestandardních injektovaných službách (anglicky injected services).

Při vývoji uživatelského rozhraní bylo potřeba efektivně vytvářet breadcrumbs (drobečková navigace), hlavní menu a obsahové menu.

Pro breadcrumbs (drobečková navigace) byl použit whiteoctober Breadcrumbs bundle (balíček) dostupný z repozitáře WoBreadcrumbs.

Tento bundle (balíček) se používá velice specifickým způsobem. Proto menu a další navigace, jež byla tvořena front-end vývojářem, jsou z důvodu konzistence též provedeny jako služby (services). Díky tomuto faktu je i kód Controlleru čistější, neboť není zapotřebí menu posílat do Twig šablony. Stačí zavolat, pomocí Twig rozšíření, funkci na vykreslení potřebného menu v dané Twig šabloně.

Ukázka Twig rozšíření:

```
public function __construct(
                ContainerInterface $container,
                MainMenu $mainMenu
                )
{
        $this->container = $container;
        $this->mainMenu = $mainMenu;
}

public function getFunctions()
{
return array(
        new \Twig_SimpleFunction("my_render_main_menu",
        array($this, "renderMainMenu")),
        );
}

public function renderMainMenu()
{
        return $this->mainMenu->render();
}
```

Díky tomuto je pak možné pro vykreslení menu zavolat přímo z Twig šablony příkaz:

```
my_render_main_menu()
```

Položky těchto navigací se nastavují v Controlleru nebo přímo ve třídě dané navigace.

Co největší čistoty kódu v těchto komponentách je potřeba dosáhnout vzhledem k nutným definicím navigací pro každou stránku a tudíž i akci Controlleru. Tím vzniká zanášení Controlleru kódem, který nejde už jinak zkrátit.

4.2.2 Šablony (templaty)

Tato podsekce se nejprve zabývá úvodní integrací, nebo-li převedením designu/návrhu rozhraní do šablon jazyka pro Twig.

Druhá část této podsekce se pak zabývá následným vývojem a strukturou twigových šablon.

Šablony (templaty) pro jednotlivé entity se nacházejí v souborové struktuře takto:

```
app
└── Resources
    └── views
        ├── pages ..... všechny entity co se mapují na webové stránky
        ├── security ..... např. přihlašování
        └── default ..... např. formuláře
```

4.2.2.1 Integrace

K zakomponování designu do jádra aplikace bylo též potřeba přepsat jej z čistého html do Twig syntaxe.

To není nijak obtížný úkol. Důležitější však je zvolit takový přístup, který nebude v budoucnu omezovat další rozšiřování těchto šablon. Toho lze díky nástroji jako je Twig lehce docílit, a to případně i zpětně.

Pro budoucí rozvoj je však pro každou stránku vytvořena nová šablona, i v případě, že není potřeba její rozložení oproti rodičovské šabloně upravovat. Tato šablona, jak vyplývá z textu, dědí od nadřazené rodičovské šablony. Všechny šablony dědí od hlavní layout šablony (až na výjimky).

4.2.2.2 Vývoj a implementace

Zde po zvážení původních požadavků na systém/aplikaci, kdy nebylo jisté jestli je potřeba obsah entit upravovat, rozšiřovat, nebo měnit za běhu aplikace jejich umístění / grafické zobrazení, byla zvolena trochu netradiční cesta.

Při vývoji a implementaci Twig šablon byla snaha o maximální znovupoužitelnost jejich částí. Proto byly navrženy takzvané samostatně stojící (standalone) šablony. Tyto šablony přijmou strukturu a tu transformují na html elementy. Tato struktura je nazvána entitním záznamem, neboť zpravidla reflektuje určitou entitu. Výhody, nevýhody a fungování těchto záznamů jsou probrány v sekci 4.3 - Entitní záznamy. Tyto šablony lze nazývat též chytré šablony.

Chytrá šablona je schopná díky Twigu rekurzivně transformovat vloženou strukturu (entitní záznam) na html entity.

Chytrá šablona využívá předdefinované rekurzivní makro. Chytrou šablonou je například šablona pro vykreslování detailu nebo listu určitých entit.

V tomto případě je lehce zneužita flexibilita Twigu. Běžně se používá spíše k řešení jednoduchých vykreslovacích pravidel.

Rozložení informací jednotlivých entit se pak řeší v samotných entitních záznamech heslovitým zápisem. O tom více ale v sekci 4.3 - Entitní záznamy.

Bezpečnost je zaručena též přímo v rekurzivním vykreslování šablon. Proto je zajištěno to, že pokud se některý element zobrazuje kde by neměl, je jisté, že má špatně nastavená práva v záznamu entity.

Mimo chytré šablony je bezpečnost zaručena stejným způsobem, jen ne rekurzivně. Twig totiž dovoluje skrýt elementy, jež by neoprávněný uživatel neměl vidět.

Nevýhody tohoto řešení jsou například:

- Horší přehlednost a orientace pro případného nového vývojáře.
- Náročnost na hw serveru.
- Pokud jsou špatně napsané styly, projeví se to v rekurzi.

Výhody tohoto řešení jsou například:

- Zaručená funkcionalita.
- Funguje-li jeden prvek typu, fungují všechny.
  - Tyto šablony fungují rekurzivně.
  - Při funkčnosti určitého pravidla je zaručeno, že bude fungovat všude (se správnými styly).
- Jednoduché přidávání nových pravidel pro vykreslování.
- Ve chvíli, kdy je hotová šablona, už se k ní vývojář nemusí vracet.

  - Všechny úpravy rozložení informací probíhají v záznamu entity.

- Méně kódu v šablonách stránek. Jednotné vykreslování.

Chytré šablony jsou umístěny například následovně:

```
app
└─ Resources
   └─ views
      └─ standalone ..... šablony nejsou svázány s určitou stránkou
         └─ components ..... částečné šablony
            ├─ lists ..... typ šablony: list / pseudo list
            │  ├─ items_list_detail.html.twig ..... detail entity
            │  └─ …
            └─ tables ..... typ šablony: tabulka
               ├─ administrative_table_default.html.twig tabulka
               │  entit
               └─ …
```

4.3 Entitní záznamy

V této sekci se práce zaměřuje na popis takzvaných entitiních záznamů. Tyto záznamy jsou vždy úzce spojeny s vykreslovacím systémem, v tomto případě Twig, což je též částečně probráno v sekci 4.2.2 - Šablony (templaty).

Důvodem této konstrukce je původní zadání požadavků. Z počátku projektu nebylo jasné, jestli bude v budoucnu potřeba obsah rozšiřovat, upravovat jeho formátování, pozici, strukturu a podobně. Též nebylo jasné, jestli to bude řešit přímo samotná aplikace nebo její budoucí rozšíření. Z tohoto důvodu bylo potřeba vytvořit co nejflexibilnější strukturu.

Tyto záznamy lze nalézt následovně:

```
src
└─ RobotManagementSystemBundle
   └─ FrontEnd ..... všechny php komponenty související s front-endem
      └─ Template ..... všechny entity, které se týkají šablon
         └─ Lists
            └─ Records ..... záznamy
               ├─ Entity
               └─ * ..... kterákoliv složka a její potomci
```

4.3.1 Záznamová položka

Tyto položky úzce souvisí s entitními záznamy. Informace jsou v entitních záznamech ukládány právě do těchto struktur.

Tyto struktury/položky specifikují jaké povahy jsou informace v ní uložené. Příkladem je formulářová položka, která potřebuje speciální pole na vykreslení formuláře. Její typ je pak rozeznáván pravidlem kontrolující třídní typ v chytré šabloně.

Speciální položky jsou vždy potomky *GenericItem*.

Proměnná content (obsah) drží obsah dané informace. V případě speciálních typů položek, např.: formulář, pak drží fallback (záložní) obsah.

Proměnná heading (titulek) drží obsah titulku dané informace.

Tyto položky lze nalézt následovně:

```
src
└─ RobotManagementSystemBundle
   └─ FrontEnd ..... všechny php komponenty související s front-endem
      └─ Template ..... všechny entity, které se týkají šablon
         └─ Lists
            └─ Items ..... záznamové položky
               ├─ GenericItem.php ..... základní položka
               └─ …
```

4.3.2 Princip

Dále jsou záznamy též úzce svázány s modelem (většinou entita), jež transformují. Tyto entity slouží totiž k organizaci dat, která probíhá těsně před samotným vykreslováním. Tato transformace dat má za úkol utřídit jednotlivé informace z modelu a rozšířit je o data, která stroj nedokáže správně vyhodnotit bez pomoci vývojáře. Například přidá správný nadpis a jeho překlad, zvolí co má informační hodnotu a co ne a zvolí oprávnění zobrazovat jednotlivé informace.

Tato struktura je schopna držet informace v textové a heslovité podobě o tom jak má být zobrazena při vykreslování. Proto při změně vykreslovacího systému stačí vytvořit šablonu, jež informace transformuje na potřebné elementy.

4.3.2.1 Struktura záznamu

Záznam je jednoduše řečeno sadou zanořených polí. Je tomu tak, aby bylo vždy možné, pokud je zapotřebí, záznam rozšířit.

Záznam v současné podobě trpí několika nedostatky. V budoucím rozvoji systému však budou nedostatky odstraněny. Tyto nedostatky nebrání funkčnosti, ale nejsou ideálním řešením.

„Pole podporovaných zobrazení“ je pole zadefinovaných struktur, jež chytrá šablona může využít. Například detail entity.

Každá položka tohoto pole se skládá ze záznamových položek (*RecordItem*). Tyto položky a jejich pořadí určuje jak se daná informace bude ve finálním vykreslení zobrazovat. Tyto položky též mohou být zanořené do sebe pomocí speciálních záznamových položek nebo jednoduše dalších polí položek.

4.3.2.2 Definice záznamu

Než začne vývojář definovat záznam, musí zadefinovat, co se má přidat do jeho struktury v *RecordsListTransformer* třídě. Toto je jednou z chyb aplikace (nebyl čas vhodně upravit *RecordsListTransformer*, a tak je to rozsáhlá factory method).

Záznam je vždy vytvořen vývojářem. Vývojář zadefinuje proměnné, které k transformaci informací potřebuje, zvolí podporovaný typ modelu a zvolí bezpečnostní pravidla.

Poté již definuje, jak bude vypadat záznam o určitém zobrazování modelu v „poli podporovaných zobrazení“. Například, jak bude vypadat detail entity nebo její zobrazení v tabulce.

4.3.2.3 Inicializace struktury

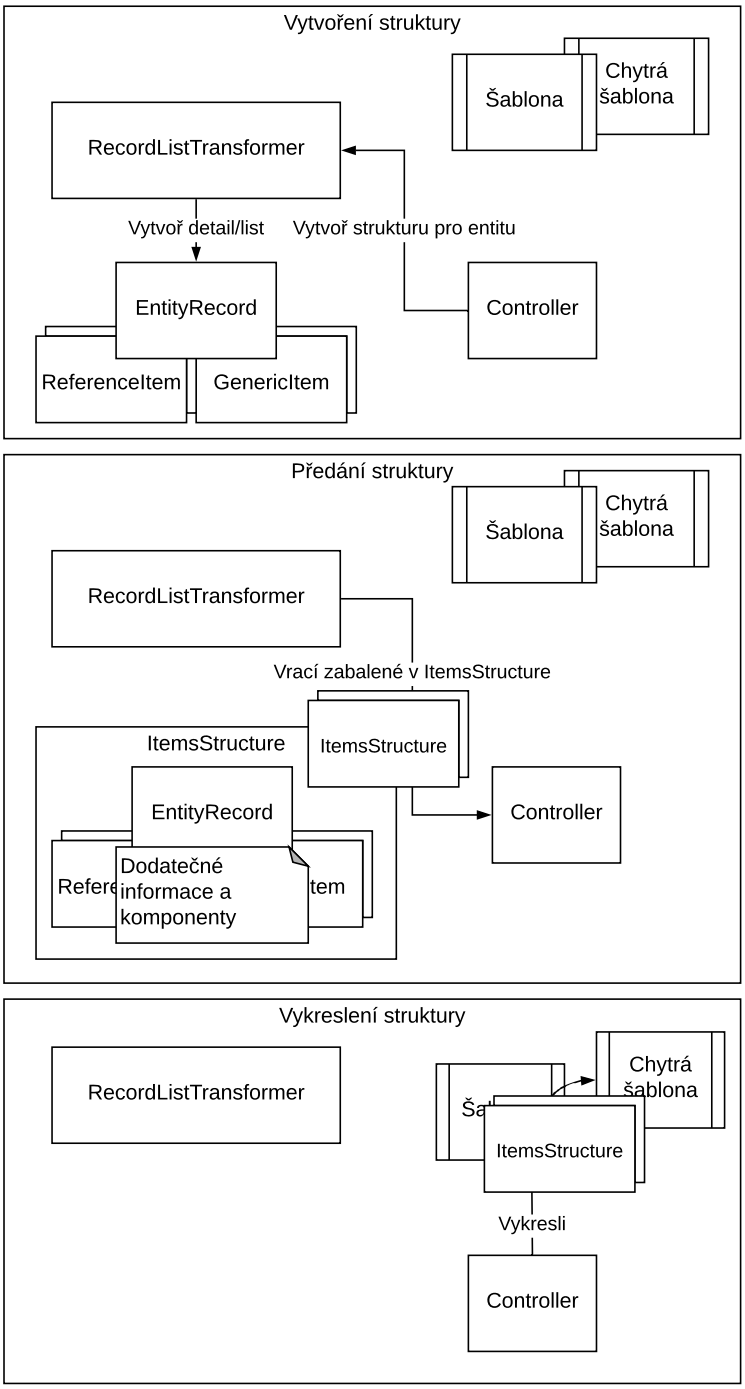
Inicializace záznamové struktury není inicializací záznamu. Tato struktura může obalovat více záznamů pod jedním zobrazením. Typickým příkladem je struktura pro chytrou šablonu tabulky.

Inicializace struktury probíhá ve třech krocích:

1. Controller před voláním vykreslovací funkce zavolá *RecordsListTransformer* a požádá o vytvoření struktury pro požadovaný model.
2. *RecordsListTransformer* vytvoří entitní záznam, respektive záznamy, pro požadovaný model a přidá správný nadpis nebo jiné informace.
3. Tato struktura je poslána k vykreslování a zde se pomocí chytré šablony vykreslí.

Pro ilustraci této procedury je níže uvedený diagram. Procedura by se též dala zobrazit pomocí stavového diagramu. V tomto případě je však účelem pokusit se jednoduše přiblížit jak procedura probíhá. Ne ji detailně popsat. Proto diagram též volí ilustrativní ikony pro třídy a skupiny tříd. Jsou zvoleny tak, aby se jednotlivé části daly lépe odlišit. Nemají tedy sémantický význam vázaný na vzhled dané ikony.

Obrázek 4.1: Zobrazení entitního záznamu na reálném příkladu.



4.3.3 Výhody

Největší výhodou této konstrukce je možnost budoucího napojení, například JavaScript komponenty umožňující aktivní úpravu umístění, případně způsob zobrazení, obsahu zobrazovaného v chytrých šablonách. Po implementaci chytré šablony se informační struktura obsahu modelu (například entity) řeší jen na jednom místě pomocí heslovité definice. Další výhodou je velice rychlá a efektivní úprava zobrazení nebo umístění určité informace v dané sekci stránky.

Uživatel bude moci v budoucnu přidávat popisky nebo vybírat informace, které chce zobrazovat. V každém záznamu budou funkce, které zavolá z prostředí uživatelského rozhraní a ty přidají požadovaný obsah do záznamu entity.

4.3.4 Ukázka kódu záznamu

Tato sekce je umístěna zde, neboť navazuje na předchozí sekci Výhody. Je zde totiž možné vidět, jak vypadá vytváření informační struktury pro zobrazení určité entity.

```
public function transformEntityToItemsList($entity)
{
$this->createReferenceVariables($entity);
return [
        GenericRecord::ITEM_GROUP_DETAIL =>
        $this->createDetailLayout($entity),
];
}

private function createReferenceVariables(User $entity)
{
$this->userReference = $this->createSimpleReferenceItem("", "",
        ["userDetail", "editUser"],
        [["user" => $entity->getId()],
         ["user" => $entity->getId()]],
        [$this->translate("common.detail"),
        $this->translate("common.edit")],
        [UserPermissionProvider::PERM_READ,
         UserPermissionProvider::PERM_EDIT],
        [false, false],
        "",
        NestedItem::DIRECTION_VERTICAL
        );
}
```

Výše uvedená ukázka ukazuje inicializaci záznamu. Níže uvedená ukázka navazuje definicí jednotlivých záznamových položek. Tyto položky se pak zobrazují v chytrých šablonách. Ukázky kódu provádí překlad uvnitř chytré šablony. V aplikaci se však většina překladů provádí z interních důvodů mimo šablony, i když toho šablony jsou schopny.

```
private function createDetailLayout(User $entity)
{
return [
        [//row 1
        new HeadingItem("common.info", 0),
        new NestedItem("common.common_info", [
          new DetailItem("common.id",$entity->getId(), 0),
          new DetailItem("common.username",$entity->getUsername(), 0),
          new DetailItem("common.valid_to",
                 ($entity->getValidTo()) ?
                  $entity->getValidTo():'not restricted', 0),
        ], 4),
];
}
```

Každá z položek se zobrazí na základě specifikace jejího typu ve výsledné stránce následovně.

Pro ilustraci je zde poskytnuto následující vyobrazení mapování kódu na reálné elementy stránky.

Toto vyobrazení je složeno ze dvou částí. Jednou částí je kód, který je uvedený výše v této sekci a druhou částí je reálný obraz detailu uživatele zachycený přímo v aplikaci, v době jejího vývoje.

Proto se mohou v jeho rozhraní nacházet nepřeložené výrazy nebo dokonce chyby rozložení. Proto je prosím omluvte.

Popis vyobrazení

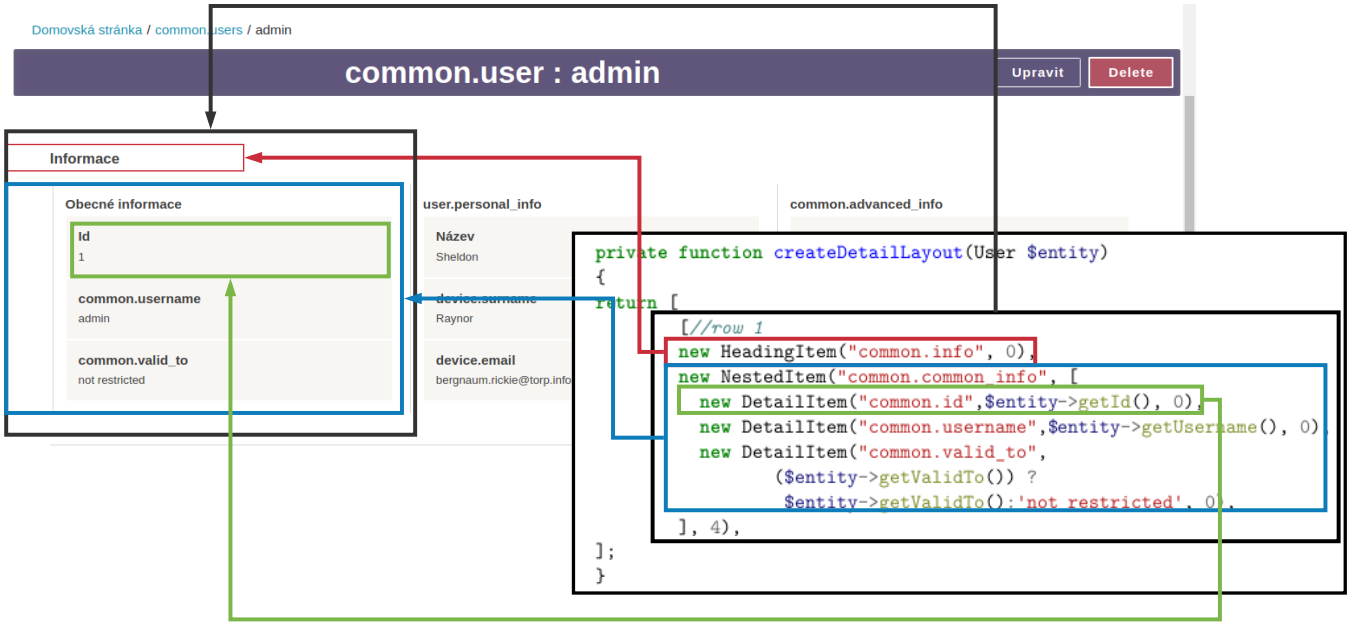
V levé a horní části vyobrazení vidíme obraz pořízený v rozhraní aplikace. Konkrétně tedy v detailu určitého uživatele. V pravé dolní části vidíme kód, který byl již dříve vyobrazený výše.

Od rámečků ohraničujících kód vždy vede ukazatel stejné barvy jako je rámeček na část uživatelského rozhraní, kde se kód projeví po vykreslení.

Z tohoto vyobrazení je vidět, jak chytrá šablona postupuje od vrchu a postupně se zanořuje níže. Informace, které jsou uvedené v tomto kódu stačí šabloně pro vykreslení částí tak, jak jsou zobrazeny.

Níže uvedený obrázek lze též nalézt v plné velikosti v přiloženém CD (viz. Obsah přiloženého CD).

Obrázek 4.2: Inicializace struktury záznamu pro model



4.3.5 Nevýhody

Tato konstrukce přináší do aplikace poměrně velkou režii a zdržení při implementaci. Jelikož nebyla vyvíjena za jedním účelem, je místy zbytečně rozsáhlá.

Je potřeba nejprve upravit databázi, aby se tato struktura dala plně využít.

4.4 Souborová struktura systému

V této kapitole je krátce popsána souborová struktura celé aplikace. Není tu detailně probrána celá struktura, neboť to nespadá pod front-end. Přes tento fakt v této souborové struktuře se front-end části také vyskytují, jak bylo již zmíněno dříve.

Souborová struktura tohoto projektu se řídí souborovou strukturou Symfony frameworku. Proto jde spíše o to, jestli jsou dodrženy správné praktiky (best-practices) umístění jednotlivých komponent systému. To je též důvodem nutnosti uvádět v této práci tuto kapitolu.

4.4.1 Symfony aplikační struktura

Zde je nutné upozornit, že struktura aplikace se řídí strukturou pro Symfony framework verze 3.4 (ne 4.0).

Následující stanovení se týkají této aplikace. Nepopisují tedy doporučenou strukturu, nýbrž strukturu této aplikace.

Struktura je následující:

- `app` ..... konfigurace, šablony (templates), překlady
- `bin` ..... spustitelné soubory, například konzole symfony
- `src` ..... komponenty aplikace psané v jazyce php
- `tests` ..... testy
- `var` ..... cache („keš“), logy
- `vendor` .. balíčky, komponenty a dodatky třetí strany (např.: drobečková navigace)
- `web` ..... kořenová složka webové části aplikace
- `web-dev` ..... vývojová složka pro webovou část aplikace

Složka web-dev stojí za zmínku, neboť jsou v ní umístěny vývojářské pomůcky pro front-end vývojáře a webovou část aplikace. Těmito pomůckami jsou například nástroje jako Gulp a Sass (popsány v sekci 4.1.1).

Složka web pak slouží k ukládání výsledků práce ze složky web-dev, které jsou vhodné pro produkční prostředí webové části.

4.4.2 Struktura frontend části aplikace

Tento projekt/aplikace, jak už bylo mnohokrát zmíněno, je sdílená přes verzovací systém git s vývojářem back-endu. Proto některé složky spadají pod obě části (back-end i front-end) a některé ne.

Zde jsou vypsány ty složky, které spadají alespoň částečně pod front-end.

4.4.2.1 Složky spadající plně pod frontend

Složky zobrazené v následující souborové struktuře spadají plně pod front-end. To znamená, že do nich, až na výjimky, nezasahoval vývojář back-endu.

```
app
    resource
        themes ..... obsahuje Twig „témata“ (themes) pro formuláře
        translations ..... překlady rozhraní
        views ..... všechny Twig šablony
src
    RobotManagementSystemBundle
        FrontEnd . všechny php komponenty pro front-end (např.: entitiní
        záznamy)
        Traits ..... traity (v tomto stavu aplikace)
web ..... kořenová složka webové části aplikace
web-dev ..... vývojová složka pro webovou část aplikace
```

4.4.2.2 Složky spadající částečně pod frontend

Složky zobrazené v následující souborové struktuře spadají částečně pod front-end. To znamená, že některé soubory, nebo části souborů byly vytvořeny jak front-end vývojářem, tak back-end vývojářem.

Pro jednoduchost zde nejsou vypsané všechny soubory. Důležité je ale zmínit, že všechny Abstraktní třídy a Interface třídy vytvořil zcela back-end vývojář.

```
src
    RobotManagementSystemBundle
        Controller
            Entity .....v současném stavu né na všech souborech
            Main ..... teprve v budoucím vývoji aplikace
```

Třídy ve složce Controller/Entity v současném stavu nejsou všechny implementovány. Proto to neplatí pro všechny soubory. V budoucím stavu aplikace to však platit bude, neboť tyto Controllery je třeba implementovat pro chod aplikace. Více v sekci 4.5 - Přehled a budoucí rozvoj aplikace

Obdobné tvrzení platí i pro složku Controller/Main, kde v současné době všechny soubory jsou vyplněny back-end vývojářem.

4.5 Přehled a budoucí rozvoj aplikace

V této kapitole se práce věnuje popisu technického stavu aplikace (zaměřeno na front-end), účelu stávající verze systému/aplikace a krátkému popisu, jak se bude projekt a aplikace rozvíjet v budoucnu, případně kdo za to bude zodpovědný.

4.5.1 Přehled

Systém/aplikace je v době tvorby této práce v alfa verzi vývoje a tudíž není otevřený veřejnému testování. Tato verze systému/aplikace má za účel vybudovat stabilní základ, poskytnout funkční testovatelnou verzi, otestovat design a odhalit nedostatky v konstrukci a uživatelském rozhraní aplikace.

Verze systému/aplikace mohou být následující:

Alfa: Verze určená počátečnímu vývoji a internímu testování.

Beta: Verze určená k rozšiřování a korekci aplikace na základě uživatelského testování (veřejného),

Production: Verze určené k produkčnímu použití a rozšiřování.

4.5.2 Technický pohled

Zde jsou uvedeny krátké statistiky sloužící jako přehled rozsahu projektu.

Front-end část aplikace se skládá celkem ze 179 zdrojových souborů. Z toho je:

- 68 html/Twig,
- 70 php,
- 37 scss,
- 4 ostatní (například vygenerované).

Tyto soubory dohromady obsahují **21828 řádků kódu**. Tento kód však může být částečně tvořen například komentáři.

Nástroj *SLOCCount* umožňuje spočítat reálný počet řádků kódu. Tento nástroj tedy napočítal cca **14500 řádků reálného kódu**. Z těchto řádků je cca **7000 kód psaný v jazyce php**. K tomuto je potřeba jen dodat, že některé ze Symfony anotací se tomuto programu jeví jako komentáře, a tak je nepočítá.

4.5.2.1 Ilustrace systému

Následující ilustrace vyobrazuje rozložení komponent a částí systému. Též přiřazuje zodpovědnost jednotlivým vývojářům. V ilustraci je vidět struktura systému z detailnějšího pohledu. Aplikací v tomto diagramu je míněna webová aplikace a uživatelským rozhraním, je míněno webové uživatelské rozhraní.

Obecný pohled na tuto problematiku lze nalézt na obrázku 3.1 - Ilustrace struktury systému.

Legenda k obrázku 4.3 - Ilustrace struktury systému - detailní:

zeleně vyznačená plocha:
Části realizované front-end vývojářem.

modře vyznačená plocha:
Části realizované back-end vývojářem.

plná čára: Logické spojení (například konfigurace konfiguruje databázi).

přerušovaná čára: Logická komunikace.

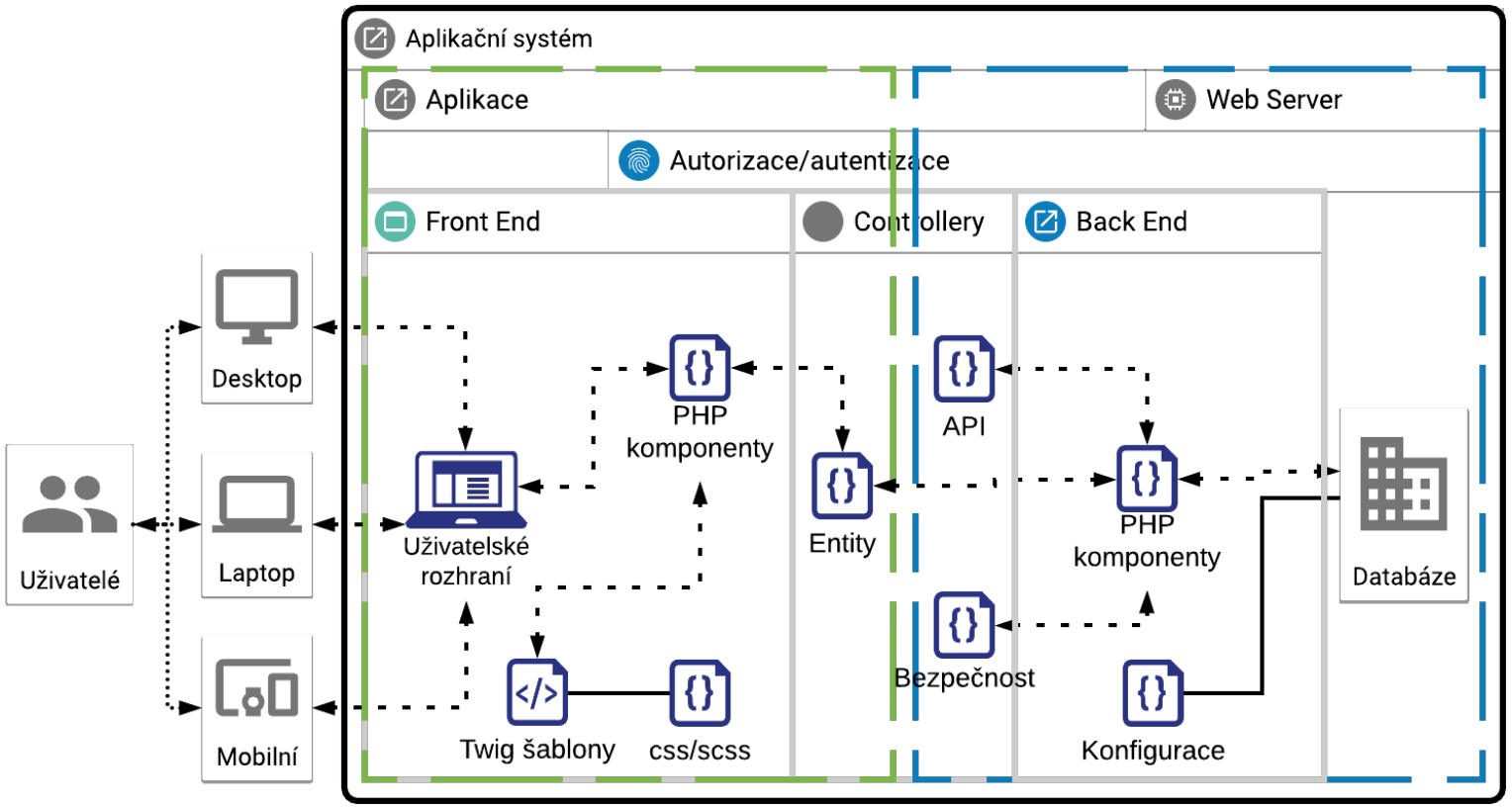
tečkovaná čára: Přímá komunikace (například uživatel s pracovní stanicí).

ikony tříd: Zobrazují logické celky. To znamená, že všech 68 html/Twig souborů je zde zobrazeno jako jedna ikona. Obrázek v nich obsažený je též jen ilustrativní.

ikony zařízení: Zobrazují různé druhy zobrazení, respektive velikosti zobrazovacích ploch, které musí uživatelské rozhraní poskytnout.

ostatní ikony a obrázky:
Jsou jen ilustrativní, jejich význam vyplývá z textu u nich uvedeného (nemají sémantický význam).

Obrázek 4.3: Ilustrace struktury systému - detailní



4.5.3 Pokračování vývoje

Jak již bylo zmíněno v úvodu této práce, vývoj aplikace nekončí dokončením této práce. Vývoj je plánován minimálně do poloviny roku 2018. Pravděpodobně však bude pokračovat i do tři-čtvrtiny toho samého roku.

Vývojářský tým má zůstat po dobu vývoje stejný. Předpokládá se však, že v budoucnu se k projektu připojí i další vývojáři.

Rozšiřování systému je též do budoucna plánováno v podobě nových komponent nebo částí systému.

4.5.4 Rozvoj aplikace

V této sekci se práce zmiňuje o technickém rozvoji aplikace. To znamená rozšiřování funkcionalit a uživatelského rozhraní.

Zaměřuje se pouze na dobu vývoj pod stávajícím front-end vývojářem.

4.5.4.1 JavaScript

V sekci 3.3.2.1 - JavaScript, je zmíněno, že v budoucnu se aplikace bude rozšiřovat o technologii JavaScript a AJAX.

Ve výše zmíněné sekci je též zmíněno, že se za tímto účelem pravděpodobně bude zakomponovávat JavaScript framework do front-endu.

Obě dvě tvrzení jsou velice pravděpodobná, ale nejsou jistá. Tato rozhodnutí záleží na budoucích požadavcích zadavatele.

4.5.5 Entitní záznamy

V sekci 4.3 - Entitní záznamy, se práce zmiňuje o faktu, že jsou entitní záznamy navrženy tak, aby se jejich obsah dal z uživatelského rozhraní, za použití správných technologií, rozšiřovat a upravovat (včetně způsobu zobrazení informace).

Toto vede na možné budoucí rozšíření o funkce a komponenty, které tyto akce umožní. Proto jsou také možným budoucím rozvojem.

Pravděpodobnou situací je to, že právě JavaScriptový framework bude využit pro manipulaci s těmito záznamy z uživatelského rozhraní.

Je nutné provést úpravy v databázi aplikace pro správnou funkčnost těchto záznamů. Pro záznamy by se v entitách musel vytvořit způsob, jakým by si uložily rozšířený obsah.

4.5.6 Scripty

V této práci se prakticky nevyskytují zmínky o scriptech jež vytváří třetí vývojář (nezávislý na front-endu). S tímto vývojářem není potřeba žádná spolupráce.

V budoucnu by se to však mohlo změnit, v případě, že by se vytvářelo rozhraní na front-endu pro efektivnější manipulaci se scripty nebo přímo se zařízeními.

4.6 Testování

V této kapitole práce probírá jak testování v průběhu vývoje systému/aplikace, tak v konečné fázi realizace (praktické části) této práce (ne celého projektu).

Provedeno zde bude též expertní a uživatelské testování uživatelského rozhraní.

4.6.1 V průběhu vývoje

Velká část testování jak uživatelského rozhraní aplikace, tak celého systému probíhala během realizace/implementace systému/aplikace.

Převážně se jednalo o expertní průchody aplikací za účelem odhalení nedostatků nebo chyb.

Nalezené chyby byly přidány do listu následujících úprav, nebo ohlášeny jako chyby ve verzovacím systému git (gitlab).

Většina z těchto chyb však byla okamžitě po zjištění vyřešena.

Druhým způsobem testování bylo testování průchodu aplikací, za účelem dosáhnutí určitého výsledku nebo akce (též expertní).

Odhalené nedostatky vůči Nielsonově Heuristice byly vždy ihned odstraněny (též lze najít částečně v sekci 3.5).

Třetím způsobem testování bylo povolání nezúčastněného uživatele, aby prošel aplikaci a nalezl chyby/nedostatky.

Nedostatky byly poznamenány a většina z nich hned vyřešena. Některé nedostatky však vyžadovali implementaci nových struktur a takové nedostatky byly uvedeny do listu nadcházejících úprav, nebo ohlášeny jako chyby ve verzovacím systému git (gitlab).

4.6.2 Na konci praktické části

Ke konci praktické části této práce bylo provedeno několik testování jak expertních, tak uživatelských. Toto testování bylo převážně zaměřeno na uživatelské rozhraní aplikace.

Výsledky testování vedly na úpravy uživatelského rozhraní.

4.6.2.1 Expertní vyhodnocení - Heuristický průchod

Při expertním vyhodnocování existuje metoda, kdy se vytvoří fiktivní persony, pomocí nichž se následně aplikace prochází a testuje se věrohodnost jednotlivých jeho artefaktů.

Tato metoda zde nebude použita, neboť z mého hlediska nepřináší téměř žádný benefit do průchodu aplikace a jeho testování. Jednodušší metodou je jednoduše představa neznalosti aplikace. Z mého pohledu je to daleko efektivnější a rychlejší způsob. Obzvláště pak v případě jako je tento, kdy expertem jsem já jako front-end vývojář a druhým expertem je back-end vývojář. Proces vymýšlení person je též poměrně zdlouhavý.

Kognitivní průchod
V kognitivním průchodu je probráno jen několik hlavních cílů uživatele. Cílů je v systému totiž prakticky nespočetné množství a ne všechny akce jsou v konečném stavu praktické části této práce podporovány.

Tento postup je uveden v [20].

Budeme se tedy ptát na otázky (též pocházející z [20]):

- Má uživatel vše pro výběr a provedení akce? Což znamená:
  - Má uživatel důvod snažit se provést právě takovou akci?
  - Ví uživatel kde se nachází?
  - Je prvek spouštějící akci vidět nebo uživateli znám?
  - Mají všechny ostatní prvky v očích daného uživatele nižší prioritu?
- Má uživatel vše pro porozumění odezvě? Což znamená:
  - Je odezva patrná příp. čas odezvy přiměřený okolnostem?
  - Je odezva srozumitelná?
  - Dá se odlišit odezva spuštění akce a dokončení akce, je-li to třeba (např. grafická odezva tlačítka oproti dokončení vyvolané akce)?

Pokud dosáhneme u některé z otázek záporné odpovědi, znamená to, že návrhář designu musí opravit design uživatelského rozhraní, tak aby se na danou otázku dalo kladně odpovědět. Toto též pochází z [20].

Evidence zařízení - přechod na stránku zařízení

- V tomto případě má uživatel, který je oprávněný, vše pro dosažení cíle.

  - Akce je nezbytná pro dosažení cíle a je jasným prvním krokem k jeho dosáhnutí.
  - Uživatel má k dispozici více označení pozice ve stránkové struktuře.
  - Prvek je přístupný skrze hlavní menu a je na první pohled vidět.
  - Ostatní prvky jsou stejně významné, ale jasně odlišitelné vizuálně i polohově (částečné ano).
- Odezva je jasná.
  - Čas odezvy je přiměřený akci.
  - Odezva v podobě změny polohy ve stránkové struktuře je adekvátní.
  - Tlačítko viditelně reaguje na stisk — to znamená že ano.

Evidence zařízení - přechod na vytvoření zařízení

- Uživatel, který je oprávněný, má k dispozici vše pro dosažení cíle.
  - Zde už uživatel nemusí váhat, neboť akce je jeho cílem.
  - Uživatel má k dispozici více označení pozice ve stránkové struktuře.
  - Prvek je u seznamu zařízení a je na první pohled rozeznatelný.
  - Ostatní prvky jsou méně významné při hledání tlačítka.
- Odezva je jasná.
  - Čas odezvy je přiměřený akci.
  - Odezva v podobě změny polohy ve stránkové struktuře je adekvátní.
  - Tlačítko viditelně reaguje na stisk — to znamená že ano.

Evidence zařízení - vytvoření zařízení

- Uživatel, který je oprávněný, má k dispozici vše pro dosažení cíle.
  - Zde uživatel jasně vidí že je na dosah cíle, a tak je jasné, že má motivaci dokončit akci.
  - Uživatel má k dispozici více označení pozice ve stránkové struktuře. Hlavní indikací je však povaha prvků.
  - Prvky pro vytvoření jsou seskupeny ve středu obrazovky v tradiční formulářové formě.
  - V tuto chvíli jsou ostatní prvky zanedbatelné v očích uživatele.
- Odezva je dostačující.
  - Čas odezvy je přiměřený akci.

  - Odezva v podobě zprávy a obnovení stránky je z počátku lehce matoucí, ale lehce pochopitelná.
  - Tlačítko viditelně reaguje na stisk a zpráva je též dobře viditelná. V případě neúspěchu jsou zprávy při prvním střetnutí lehce matoucí.

Z tohoto průchodu je zjistitelná nedostatečná odezva při neúspěšném zakončení akce. Ve chvíli, kdy si uživatel všimne co udělal špatně je však jasný další postup a opakovaná poslední akce.

Přihlášení uživatele - přechod na přihlášení

- Každý uživatel má dostatek informací k dokončení tohoto cíle.
  - Uživatel je aplikací k tomuto přechodu vyzván automaticky, pokud přistupuje k zabezpečenému obsahu. Pokud ho nevyzve aplikace, má potřebu tuto akci vykonat pouze pokud je s aplikací seznámen (což není špatně v tomto případě).
  - Pro uživatele je jasně viditelné, že je žádán o přihlášení.
  - Prvek je přístupný skrze menu v pravém horním rohu.
  - Ostatní prvky jsou jasně odlišitelné a pro uživatele zanedbatelné.
- Odezva je jasná.
  - Čas odezvy je přiměřený akci.
  - Odezva v podobě přesunutí uživatele na přihlašovací stránku je standardní.
  - Tlačítko reaguje dostatečně.

Přihlášení uživatele - přihlášení

- Každý uživatel má možnost akci dokončit.
  - Uživatel jasně vidí, že je cíl jeho akcí na dosah, a tudíž je motivován k výkonu akce.
  - Pro uživatele je jasně viditelné, že je žádán o zadání přihlašovacích údajů.
  - Prvky jsou přístupné ve středu obrazovky.
  - Žádné jiné prvky nejsou k dispozici.
- Odezva je dostatečná.
  - Čas odezvy je přiměřený akci.

  - Odezva v podobě přihlášení, respektive informování o špatně zadaných údajích je dostačující.
  - Tlačítko reaguje jasně.

V případě úspěšného přihlášení se uživatel musí podívat, jestli je opravdu přihlášen do míst, kde původně začínal akci. To je však automatická reakce na tuto akci. Proto je tento způsob informování dostatečný. V případě neúspěchu je odezva v podobě zprávy na obrazovce jasná.

Jak je vidět kognitivní průchody jsou poměrně zdlouhavé, a proto zde jsou jen tyto dva příklady. Průchody byly provedeny pro všechny podporované akce, ve zkrácené a nezdokumentované formě. To znamená, že experti si zadali cíle a následně kontrolovaly jednotlivé body kognitivního průchodu.

Proto se dále kognitivnímu průchodu práce věnuje jen v sekci 4.6.2.2 - Uživatelské vyhodnocení.

Nielsonova Heuristika

V sekci 3.5 - Design a Nielsonova Heuristika, se již práce věnuje Nielsonově Heuristice při navrhování systému. Je však vhodné se tomuto věnovat i nyní ke konci praktické části.

Jsou zde kontrolovány následující body z [20]. U každého bodu je krátká analýza jeho splnění:

- Ví uživatel vždycky, kde se nachází (navigace). Ví uživatel vždycky, v jakém je aplikace stavu?
  - Aplikace indikuje stavy změnou stránky (prozatím), a proto kdykoliv, kdy uživatel mění stav, mění i stránku. Stránky mají vždy minimálně dva indikátory polohy. Mezi indikátory patří např.: drobečková navigace
- Jsou názvy, popisky a jinak používané termíny v souladu s terminologií cílové skupiny?
  - Na tuto otázku se odpovídá už o něco hůře, neboť cílových skupin je několik (vyučující,studenti,zaměstnanci). Termíny jsou srozumitelné studentovi i vyučujícímu. Zaměstnancem však může být neznámá osoba.
  - Termíny bude také potřeba znovu přeložit a několikrát projít, než dosáhnou konečného stavu.
- Jsou názvy, popisky a jinak používané termíny srozumitelné cílové skupině? Jsou názvy kategorií (například v menu) voleny s ohledem na srozumitelnost pro cílovou skupinu?
  - Na tuto otázku je odpověď obdobná, jako na otázku předcházející.

  - Tyto termíny jsou z části subjektem budoucího rozvoje.

- Poskytuje každá akce zpětnou vazbu srozumitelnou pro uživatele?

  - Některé akce postrádají dostatečnou indikaci úspěchu/neúspěchu.
  - To je též subjektem budoucího rozvoje, neboť ne všechny akce jsou podporovány v této fázi projektu.

- Může se uživatel dostat z každé situace? Obejde se takové opuštění bez nutnosti opakovat dlouhou sekvenci akcí? Je použití tlačítek ← a → správné?

  - Uživatel má vždy možnost se z akce vrátit zpět do původní pozice. I v případě vyhledávání.

- Jsou všechny objekty, akce a jiné prvky vidět, kdykoliv je jich potřeba, aniž by si je uživatel musel pamatovat?

  - Žádné prvky nejsou uživateli skryty pokud má oprávnění je použít.
  - Výjimkou je případ mobilního zobrazení, kdy jsou obě menu (hlavní a obsahové [boční]) minimalizována ve vrchní části stránky.

- Jsou všechny akce, uspořádání a význam konzistentní s očekáváním cílové skupiny (zvyky odjinud…)

  - Aplikace se snaží o co největší konzistenci a dodržuje webové konvence umísťování prvků.
  - Chyby v umístění prvků jsou průběžně napravovány.

- Odpovídá vzhled aplikace a ovládacích prvků cílové skupině?

  - Vzhled se snaží dosáhnout moderního nádechu kombinovaného s administrativními prvky.
  - Tento přístup je pro ČVUT FIT logický, neboť je fakulta mladá a moderní.

- Je plocha zobrazení využita přiměřeně (vzhledem k celkovému vzhledu)?

  - Plocha upřednostňuje zobrazování informací, před zkrášlovacími prvky a to i v mobilním zobrazení.
  - Rozložení je adekvátní.

4.6.2.2 Uživatelské vyhodnocení

Expertní vyhodnocení je důležitou součástí testování. Neřekne však tolik důležitých informací, jako testování uživatelské.

V tomto případě testování prováděla skupina uživatelů. Tato skupina se skládala ze tří nezaujatých uživatelů, kteří s podobnými systémy nepracují a jednoho uživatele, který studuje obdobný obor zaměření této práce.

Tito uživatelé se mohli radit, ale ne se navzájem pozorovat.

Heuristický průchod
Tato metoda je zde probrána jen heslovitě. Nejsou zde podrobně rozebrány jednotlivé akce, neboť byl výsledek dost často naprosto stejný, jako expertní vyhodnocení.

Při kognitivním průchodu měli nezkušení uživatelé nejprve problém s nedostatečným překladem dané verze aplikace. Navzájem se však poradily se zkušenějším uživatelem a postupně si aplikaci prošli.

Po prvních několika minutách jim byly zadány úkoly k vykonání. U všech úkolů uživatelé uspěli, až na malé výjimky, kdy nerozuměli zadání, nebo nevěděli čeho mají docílit.

Uživatelé též podávali zpětnou vazbu o tom, co od systému/aplikace očekávají.

Na základě těchto poznatků byly pro důležitější poznatky vytvořeny záznamy na verzovacím systému git (gitlab).

Struktura aplikace i systému se též ukázala lehce komplikovanější. Zanořené entitní vazby bylo pro uživatele těžké si zapamatovat. Toto bude muset být též v budoucím rozvoji zváženo.

Všichni uživatelé však požadovali:

- Filtry vyhledávání.
- Vyhledávání i u ostatních entit (nejen projekty).
- Varování špatně vyplněného formuláře.
- Lépe indikované volby jazyka.
- Lepší překlady.

Z těchto poznatků si lze vytvořit obraz nejdůležitějších prvků budoucího rozvoje. Musíme brát na vědomí i to, že tito uživatelé nemají možnost otestovat systém/aplikaci na vestavných systémech.

Proto budou v budoucnu probíhat i další fáze testování.

Závěr

Cílem práce bylo na základě rešerše navrhnout a implementovat/realizovat část aplikace/systému pro evidenci vestavných systémů se zaměřením na front-end. Rešeršní část práce měla za úkol analýzu a návrh vhodných technologií pro propojení s back-end částí systému. Praktická část práce by měla být schopna prezentace, editace a vytváření informací o vestavných systémech a projektech uložených v databázi se zaměřením obzvláště na grafickou stránku a kompatibilitu s často používanými prohlížeči Chrome, FireFox, Internet Explorer a Edge (i mobilní verze).

Realizoval jsem designový návrh a pomocí Twigu, respektive kombinace Twigu, css a html, jsem implementoval uživatelské rozhraní, na tomto návrhu založené. Toto rozhraní je součástí sdílené (s back-end vývojářem) webové aplikace, založené na frameworku Symfony.

Zakomponování uživatelského rozhraní do webové aplikace představovalo implementaci Controllerů a struktur, jež je v rozhraní třeba využít (v jazyce php).

Realizované celky uživatelského rozhraní zajišťují:

- Zobrazování, editaci a vytváření informací o projektech a jejich závislostech. To zahrnuje též následující:
  - verze projektu,
  - konfigurace verze,
  - přílohy/přiřazení konfigurace verze,
  - překlady projektu.
- Zobrazování, editaci a vytváření informací o vestavných systémech a jejich závislostech (až na výjimku produkčního zařízení). To zahrnuje též následující:

  - modely,
  - přílohy modelů,
  - jednotlivé typy (např.: Arduino),
- Zobrazování, editace a vytváření informací o uživatelích (nebo uživatelů jako takových).
- Autorizace (za pomoci ACL).
- Správu ACL a omezování obsahu na jeho základě.

Při realizaci jednotlivých částí jsem se zaměřil na konzistenci a univerzálnost vykreslování komponent. Díky chytrým šablonám, zmíněným v sekci 4.3, je podporována rozsáhlá úprava zobrazování dat v jednotlivých částech aplikace a též je tak aplikace připravena pro budoucí rozvoj.

Tyto části jsem pravidelně testoval z hlediska funkčnosti jak front-endu, tak back-endu.

Kompatibilita aplikace je zajištěna na všech výše uvedených prohlížečích (testováno na nejnovějších verzích prohlížečů) i jejich mobilních verzích. Tohoto jsem dosáhl za pomoci Flexboxu, automatického prefixovacího nástroje a pravidelného testování aplikace vůči kompatibilitě s prohlížeči.

Vzhled aplikace je inspirován již existujícími webovými aplikacemi pro ČVUT a zásadami moderního webového designu.

Díky výsledkům analýzy jsem byl schopný vybrat takové technologie a metodiky, které mě při implementaci/realizaci neomezovaly a umožňují též budoucí rozvoj aplikace.

Zvolené technologie probrané v návrhu, se ukázaly jako efektivní a vhodné pro napojení na jádro aplikace (back-end).

V analýze jsem částečně zahrnul i technologie použitelné pro budoucí rozvoj aplikace.

Výhled do budoucna odhaluje několik možných budoucích rozšíření / návazných projektů. Vytvořená studie nezahrnuje detailní testování systému/aplikace, které by bylo vhodné v budoucnu realizovat. Proto by bylo možné se tomuto později věnovat, i z pohledu uživatelů využívající samotný systém/aplikaci (až bude systém/aplikace v provozu).

Jak se též zmiňuje sekce 4.5 - Přehled a budoucí rozvoj aplikace, v budoucnu by bylo možné aplikaci rozšířit o JavaScript a určitou formu správy obsahu a informací. Aplikace/systém je lehce rozšiřitelný, a tak je budoucí rozvoj, v podobě komponent rozšiřující jeho funkcionalitu, prakticky neomezený.

Bibliografie

1. ŠUBRTA, Václav. Web design a uživatelská rozhraní: Návrh uživatelského rozhraní webové aplikace. In: *gml.vse.cz* [online]. 7. 5. 2014 [cit. 2018-04-24]. Dostupné z: `http://gml.vse.cz/data/oppa-webdesign/ui.html`.
2. HELAN, BcA. Petr. *Vývoj front-endových aplikací* [online]. Praha, 2010 [cit. 2018-04-24]. Diplomová práce. Univerzita Tomáše Bati ve Zlíně, Fakulta multimediálních komunikací. Vedoucí práce Mgr. Pavel KRUTIL. Dostupné z: `https://digilib.k.utb.cz/bitstream/handle/10563/14332/helan_2010_dp.pdf?sequence=1&isAllowed=y`.
3. BARINKL (ed.). *Předvedení systému EDUX* [online]. 2009. 15.9. 2009, 09:08 [cit. 2018-04-22]. Dostupné z: `https://edux.fit.cvut.cz/prezentace/2009-06-25`.
4. FISHEROVÁ, Kristina et al. (ed.). *Grafický Manuál Identity ČVUT* [online]. 2015 [cit. 2018-04-22]. Dostupné z: `https://www.cvut.cz/sites/default/files/content/e254fb38-e72d-463b-8c9f-cb0435416f29/cs/20161215-graficky-manual-identity-cvut-v-praze.pdf`.
5. SÝKORA, Jan. *Optimalizace hodnocení a automatické kontroly semestrální práce v předmětu Databázové systémy.* Praha, 2015. Bakalářská práce. České vysoké učení technické v Praze Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.
6. COUFAL, Jaromír. *Design webového rozhraní, pro různé typy zobrazovacích zařízení* [online]. Brno, 2012 [cit. 2018-04-18]. Diplomová práce. Masarykova Universita v Brně, Fakulta informatiky. Vedoucí práce CSc. doc. ING. JIŘÍ SOCHOR. Dostupné z: `https://is.muni.cz/th/251432/fi_m/COUFAL-Design_prizpusobivych_webovych_rozhrani.pdf`.
7. ESCALONA, M. JOSÉ; KOCH, NORA. Requirements Engineering for Web Applications: A Comparative Study. *Journal of Web Engineering.*

2003, roč. 2, č. 3, s. 193–212. ISSN 1540-9589. Dostupné také z: http://dl.acm.org/citation.cfm?id=2011127.2011132.

8. MACHYNKA, Jan. *Vývoj front-endových aplikací* [online]. Praha, 2013 [cit. 2018-04-23]. Diplomová práce. Univerzita Karlova v Praze, Filozofická fakulta. Vedoucí práce Ph.D ING. MARTIN SOUČEK. Dostupné z: https://is.cuni.cz/webapps/zzp/detail/93637/.
9. RICHTA, Karel. Metodiky vývoje software, MDA [přednáška]. In: *edux přednášky, předmět SI1* [online]. 2011 [cit. 2018-04-23]. Dostupné z: https://edux.fit.cvut.cz/oppa/BI-SI1/prednasky/BI-SI1-P10m.pdf.
10. BUCHALCEVOVÁ, Alena. *Metodiky budování informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky*. 1. vyd. Praha: Grada, 2005. ISBN 80-247-1075-7.
11. GÓRALEWICZ, Bartosz. Going Beyond Google: Are Search Engines Ready for JavaScript Crawling & Indexing? In: *moz.com/blog* [online]. 29. 8. 2017 [cit. 2018-04-30]. Dostupné z: https://moz.com/blog/search-engines-ready-for-javascript-crawling.
12. COYIER, Chris. You Know You Should Use JavaScript When…. In: *css-tricks.com* [online]. 21. 9. 2009 [cit. 2018-04-30]. Dostupné z: https://css-tricks.com/you-know-you-should-use-javascript-when/.
13. KYRNIN, Jennifer. When to Use Ajax and When Not To: What to Do When You Get the "Ajax Call" from Your Boss. In: *www.lifewire.com* [online]. 26. 5. 2017 [cit. 2018-04-30]. Dostupné z: https://www.lifewire.com/when-to-use-ajax-3466246.
14. WALES, Michael. 3 Web Dev Careers Decoded: Front-End vs Back-End vs Full Stack. In: *blog.udacity.com* [online]. 8. 4. 2014 [cit. 2018-04-18]. Dostupné z: https://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html.
15. TRIPATHI, Sushil Kumar. TOP 5 FRONT-END JAVASCRIPT FRAMEWORKS IN 2018. In: *kelltontech.com*. 9. 2. 2018. Dostupné také z: https://www.kelltontech.com/kellton-tech-blog/top-5-front-end-javascript-frameworks-2018.
16. DWORMAN, Garett. When To Prototype, When To Wireframe: How Much Fidelity Can You Afford? In: *usabilitygeek.com* [online]. 10. 3. 2014 [cit. 2018-05-03]. Dostupné z: https://usabilitygeek.com/when-to-prototype-when-to-wireframe-fidelity/.
17. LIM, Winnie. A Beginner's Guide to Wireframing. In: *webdesign.tutsplus.com* [online]. 18. 6. 2012 [cit. 2018-05-03]. Dostupné z: https://webdesign.tutsplus.com/articles/a-beginners-guide-to-wireframing--webdesign-7399.

18. ŽIKOVSKÝ, Pavel. Návrh uživatelských rozhraní [přednáška]. In: *edux přednášky, předmět MI-NUR* [online]. 2011 [cit. 2018-05-04]. Dostupné z: `https://edux.fit.cvut.cz/oppa/MI-NUR/prednasky/06-Heuristick%C3%A1%20evaluace.pdf`.

19. BERNARD, Borek. MVC a další prezentační vzory: Úvod do architektury MVC. *Zdroják* [online]. 7. 5. 2009, č. 1 [cit. 2018-05-05]. ISSN 1803-5620. Dostupné z: `https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/`.

20. BARINKL (ed.). *Kognitivní a heuristický průchod* [online]. 2016. 4.3. 2016, 08:52 [cit. 2018-05-08]. Dostupné z: `https://edux.fit.cvut.cz/courses/BI-ZWU/tutorials/10/start`.

21. CHRISTENSSON, P. Framework Definition. In: *techterms.com* [online]. 7. 3. 2013 [cit. 2018-04-18]. Dostupné z: `https://techterms.com/definition/framework`.

22. ALANFELD et al. (ed.). *What is JavaScript?: A high-level definition* [online]. 2018. 4.3. 2018, 9:14:47 [cit. 2018-04-22]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript`.

23. DIEGOAF9 et al. (ed.). *Getting Started: What's AJAX?* [online]. 2018. 23. 4. 2018, 3:32:11 [cit. 2018-04-22]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started`.

24. BRIPMCCANN et al. (ed.). *How CSS works: What is CSS?* [online]. 2018. 15. 2. 2018, 3:32:47 [cit. 2018-04-30]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works`.

25. ARAI et al. (ed.). *HTML basics: So what is HTML, really?* [online]. 2018. 27. 1. 2018, 10:35:33 [cit. 2018-04-30]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics`.

26. IT-SLOVNIK.CZ TEAM (ed.). *Drobečková navigace* [online]. © 2008–2018 [cit. 2018-04-22]. Dostupné z: `https://it-slovnik.cz/pojem/drobeckova-navigace`.

27. IT-SLOVNIK.CZ TEAM (ed.). *PHP* [online]. © 2008–2018 [cit. 2018-05-02]. Dostupné z: `https://it-slovnik.cz/pojem/php`.

28. DOKUWIKI. *DokuWiki* [online]. Ed. 79.110.36.25. 2015 [cit. 2018-04-26]. Dostupné z: `https://www.dokuwiki.org/cs:dokuwiki?rev=1451083355`.

Dodatek **A**

Slovník

FIT klasifikace Informační systém pro evidenci klasifikace pro Fakulta informačních technologií (FIT) ČVUT.

AJAX „*Neboli Asynchronní JavaScript A XML je použitím XMLHttpRequest objektu ke komunikaci se serverem. Může posílat i přijímat informace v různých formátech.*“ [23](překlad dle autora).

back-end „*je systém v pozadí, který nemá nativní uživatelské rozhraní, většinou slouží jako úložiště dat, řídících procesů nebo má obě funkce zároveň*“ [8].

best-practices Postupy a methody, jež jsou považovány za nejefektivnější a nejpřijatelnější. Jinými slovy také dobré procvičení..

css „*Je jazyk, který specifikuje jak se bude dokument prezentovat uživatelům.*“ [24](překlad dle autora).

DokuWiki „*DokuWiki je standardy dodržující, jednoduše použitelná forma Wiki, zaměřená především na vytváření dokumentace všeho druhu.*“ [28].

drobečková navigace „*je seznam odkazů, které naznačují cestu, kterou návštěvník prošel než se na konkrétní stránku dostal (či případně indikují umístění stránky v rámci struktury celého webu)*“ [26].

Edux Víceúčelový informační systém pro podporu výuky na ČVUT.

eye candy Věc nebo objekt, který je příjemný a přívětivý na pohled.

Flexbox Nová metoda rozmisťování elementů pomocí css3 stylování. Též může být vnímán jako modul, za jehož pomoci lze realizovat rozložení elementů..

framework „*Softwarový framework je platforma pro vývoj softwarových aplikací. Poskytuje základ na kterém může developer vyvíjet sfotware pro určitou platformu. [21]* “ .

front-end Pro tuto práci míněno ve webovým technologiích znamená: „*uživatelská strana webové stránky, nebo-li to s čím je uživatel v interakci. Cokoliv od fontu, barev, menu až po responzivitu samotného webu*“ [14](překlad dle autora).

gitlab Verzovací systém s webovým i uživatelským rozhraním.

html „*Je značkovací jazyk, který definuje strukturu obsahu.*“ [25](překlad dle autora) Ke specifikaci struktury obsahu se používají takzvané elementy..

JavaScript „*Je skriptovací programovací jazyk, který umožňuje implementaci komplexních prvků na webové stránce. Například animace, nebo dynamické změny obsahu*“ [22](překlad dle autora).

KOS Informační studijní systém Komponenta Studium (KOS)
Provozovatel: Výpočetní a informační centrum.

Moodle-vyuka Informační systém pro podporu výuky na ČVUT.

php „*PHP je hypertextový preprocesor, technologie pro generování dynamických HTML stránek na straně serveru.*“ [27].

portál předmětu DBS Webový portál pro předmět DBS na Fakulta informačních technologií (FIT) ČVUT s možností evidovat projekty a aktivně tvořit obsah, jak vyučujícími, tak studenty.

Projekt LIVS Projekt spolufinancován Evropskou unií dostupný na portálu: https://livs.fit.cvut.cz/.

předmět Databázové Systémy předmět vyučující převážně jazyk SQL a databázové systémy jež jej využívají..

Symfony Php framework pro vývoj webových aplikací.

Twig Templatovací framework pro webové aplikace.

Dodatek B

Seznam použitých zkratek

ČVUT České vysoké učení technické v Praze.

ACL Access Control Lists.

CD Compact Disc.

FIT Fakulta informačních technologií.

KOS Komponenta Studium.

Livs Laboratoř inteligentních vestavných systémů.

MVC Model View Controller.

MVP Model View Presenter.

XP Extrémní Programování.

Dodatek C

Instalační příručka

Následující instalační příručku lze též nalézt v obsahu přiloženého CD (viz. D - Obsah přiloženého CD).

Tato příručka stručně popisuje kroky potřebné pro spuštění aplikace na vlastní pracovní stanici.

Základním předpokladem pro úspěšné spuštění aplikace je vlastnictví administrátorských práv minimálně nad složkou, ve které bude aplikace umístěna.

Aplikace vyžaduje úložiště informací, a to nejlépe v podobě MySQL databáze. Aplikace podporuje jakékoliv úložiště, podporované Symfony doctrine. Create (vytvářecí) skripty jsou však psány právě pro MySQL databáze.

V případě použití jiného databázového serveru než MySQL, je za potřebí skripty upravit.

Dále také pro instalaci aplikace budete potřebovat Composer nebo využít přiložené verze Composeru v podobě souboru **composer.phar**, ve složce **src/application/Sources/robot-management-system**

(A) Příprava databáze

a) Připravte (tzn. vytvořte/vyčistěte) databázi, která bude sloužit jako úložiště pro tuto aplikaci.

b) Z prostředí zvoleného databázového serveru, spusťte nad databází pro tuto aplikaci skript (s nejvyšším číslem verze) ze složky **src/application/database/SQL**, pro vytvoření databáze.

(B) Instalace aplikace

a) Zkopírujte složku **src/application/Sources/robot-management-system** do libovolného adresáře, na kterém máte práva (nejlépe plná administrátorská).

b) Přesuňte se do složky **robot-management-system** a spusťte příkaz **php** [***cesta k souboru composer.phar***] **install**.
   - V případě, že máte Composer již nainstalovaný globálně, stačí z výše uvedené složky spustit příkaz **composer install**.
   - Tento krok stáhne všechny nejnovější knihovny, při prvním spuštění a inicializuje konfigurační soubor **parameters.yml**.
   - V případě konfliktů knihoven smažte složku **robot-management-system/vendor** a celý krok opakujte.

c) V případě nutnosti úpravy parametrů, jako například uživatel databáze, heslo databáze, název databáze, lze tak učinit skrze **robot-management-system/app/config/parameters.yml**.
   - Tento soubor již neupravujte po dokončení instalace.

(C) Konfigurace serveru

a) Naveďte zvolený webový server do složky **robot-management--system/web**.
   - Index aplikace se nachází zde, v podobě souboru **app.php**.

b) V konfiguraci webového serveru přidejte této složce **AllowOverride All**.
   - To povolí přepsání pomocí .htaccess souborů.
   - Aplikace potřebuje přístup i ke složkám, které se nacházení o jednu úroveň víš (../var ../app atd.), tedy tyto složky musejí být aplikaci také dostupné.
   - Webový server také musí mít read, write, edit práva na většinu podsložek složky **robot-management-system**.

(D) V případě nefunkčnosti aplikace

a) Pokud aplikace hlásí chyby nebo nefunguje, je zapotřebí přesunout se do **robot-management-system** a spustit následující příkazy.
   - **php bin/console cache:clear –env=dev** (vyčištění cache pro dev prostředí).
   - **php bin/console cache:clear –env=prod** (vyčištění cache pro production prostředí).

b) Pokud předešlý krok nepomohl, je třeba analyzovat logy.
   - **robot-management-system/var/log** (aplikační logy).
   - logy zvoleného webového serveru.

Pokud jste dokončili úspěšně všechny kroky, aplikace je připravena k použití.

Testovací data do databáze lze nahrát pomocí insert skriptů nacházejících se ve složce **src/application/database/SQL**.

Dodatek D

Obsah přiloženého CD

- `src`
  - `application` ..... zdrojové kódy aplikace
  - `thesis` .... zdrojová forma práce ve formátu LaTeX se všemi součástmi
    - `pictures` ..... diagramy a obrázky použité v textu práce
      - `wireframes` ..... wireframy použité v textu práce
      - `screenshots` ..... obrázky (screenshoty) z webové aplikace
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `zadani.pdf` ..... text práce ve formátu PS
- `docu` ..... dkumentace aplikace a postup ke spuštění aplikace
  - `setup.pdf` ..... návod ke spuštění aplikace
  - `app_docu` ..... vygenerovaná dokumentace aplikace (složka)