ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA PEDAGOGICKÁ
KATEDRA VÝPOČETNÍ A DIDAKTICKÉ TECHNIKY

NÁVRH A TVORBA WEBOVÉHO SYSTÉMU PRO SPRÁVU SOFTWARU A LICENCÍ

BAKALÁŘSKÁ PRÁCE

Martin Laštovka
Informatika se zaměřením na vzdělávání

Vedoucí práce: Mgr. Miroslav Zíka.

Plzeň, 2024

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně s použitím uvedené literatury a zdrojů informací.

V Plzni, 25. dubna 2024

.....

vlastnoruční podpis

Rád bych poděkoval Mgr. Miroslavu Zíkovi za jeho podporu, trpělivost, odborné vedení a cenné rady během celého procesu psaní této práce.

OBSAH

SEZNAM ZKRATEK

ZČU – Západočeská univerzita v Plzni.

KVD – Katedra výpočetní a didaktické techniky fakulty pedagogické ZČU.

HTML – HyperText Markup Language, značkovací jazyk pro vytváření webových stránek.

CSS – Cascading Style Sheets, jazyk pro popis vzhledu HTML prvků.

JS – JavaScript, programovací jazyk pro tvorbu interaktivních prvků na webu.

DOM – Document Object Model, interní reprezentace struktury HTML prvků.

PHP – Hypertext Preprocessor, programovací jazyk pro vývoj dynamických webových stránek.

CMS – Content Management Systém, systém pro správu obsahu na webových stránkách.

SQL – Structured Query Language, dotazovací jazyk pro práci s relačními databázemi.

DBMS – DataBase Management System, software pro práci a manipulaci s databází.

MVC – Model View Controller, architektura aplikace oddělující jednotlivé vrstvy aplikace.

AJAX – Asynchronous JavaScript And XML, technologie umožňující upravovat obsah stránky bez nutnosti obnovit celou stránku.

ÚVOD

Bakalářská práce se zabývá návrhem a následnou tvorbou webového systému pro správu licencí softwaru. Systém je koncipován jako evidence licencí softwaru na katedře Výpočetní a didaktické techniky fakulty pedagogické Západočeské univerzity v Plzni, ale může být využit pro celou fakultu. V aktuální době není na katedře KVD jednotná centrální evidence licencí softwaru a jednotlivé záznamy jsou roztříštěny na více místech. Tato bakalářská práce má tedy za cíl vytvořit systém, kde se budou všechny záznamy shromažďovat a evidovat na jednom místě. Systém bude zároveň fungovat pod jednotným přihlášením ZČU.

Práci lze rozdělit na teoretickou a praktickou část. V teoretické části budou rozebrány použité technologie a v praktické části představeny výstupy z návrhu a realizace aplikace systému.

První kapitola bude pojednávat o cílech práce, jež budou realizovány v praktické části, zároveň bude definovat hlavní cíl a jeho dílčí cíle.

Druhá kapitola bude hlavní teoretickou částí práce, budou v ní vysvětleny technologie a principy, které budou při realizaci práce použity. V kapitole budou zmíněny jazyky HTML, CSS, JS a PHP, dále technologie SQL databází a princip responzivního designu.

Třetí kapitola se bude zaměřovat na problematiku frameworků, jejich definici a využití. V kapitole budou uvedeny příklady tří frameworků a následně zde bude vybrán konkrétní framework, ve kterém bude systém na správu licencí realizován.

Čtvrtá kapitola se bude věnovat návrhu systému na základě poskytnutých dat, od prvotního wireframu přes grafický návrh až po návrh databáze systému.

Pátá kapitola se bude zabývat realizací systému na správu licencí v rámci webové aplikace. V kapitole budou ukázky programování a kódování samotných stránek aplikace.

Šestá kapitola bude věnována návodu, jak systém zprovoznit a pracovat s ním. Fungování systému zde bude vysvětleno na různých scénářích, které pokrývají funkce systému, například vložení nového záznamu licence, nebo upravení stávajícího záznamu.

1 CÍLE

Hlavním cílem práce je navrhnout, nakódovat a naprogramovat webový systém pro správu licencí softwaru na základě poskytnutých dat o licencích. Tento cíl lze rozdělit na dvě části – návrhovou část a realizační část.

1.1 NÁVRHOVÁ ČÁST

V návrhové části je cílem navrhnout webovou aplikaci na správu licencí softwaru na základě poskytnutých dat. V jejím rámci lze definovat samostatné dílčí cíle.

Dílčí cíle návrhové části

- Vytvořit wireframe aplikace.
- Vytvořit grafický návrh aplikace na základě wireframu.
- Vytvořit návrh databáze na základě poskytnutých dat o licencích.
- Vybrat framework pro tvorbu aplikace.

1.2 REALIZAČNÍ ČÁST

V realizační části je úkolem převzít výstupy z návrhové části a přetavit je do výsledného výstupu v podobě webové aplikace na správu licencí softwaru. V rámci realizační části lze definovat samostatné dílčí cíle.

Dílčí cíle realizační části

- Tvorba databáze systému na základě návrhu databáze.
- Nakódování jednotlivých stránek a částí aplikace podle grafického návrhu.
- Vyřešení přihlašování do aplikace přes jednotné přihlašování ZČU Shibboleth.
- Naprogramování funkcí systému a komunikace s databází.

2 TEORETICKÝ ÚVOD DO POUŽITÝCH TECHNOLOGIÍ

Následující kapitola popisuje dílčí kroky od návrhu po realizaci webové stránky. V kapitole jsou zmíněny důležité technologie, principy a programovací jazyky, které jsou použity při tvorbě webové aplikace systému na správu licencí.

2.1 WIREFRAME A GRAFICKÝ NÁVRH

Wireframe a grafický návrh představují prvotní fázi vývoje webové aplikace, kde se na základě požadavků klienta nejprve vytvoří struktura webu a následně grafická podoba aplikace.

Před realizací wireframu a grafického návrhu probíhá komunikace produktu s klientem, což lze chápat, jako nultou přípravnou fázi projektu, kde se tvoří zadání a požadavky klienta na funkce a vzhled aplikace.

2.1.1 WIREFRAME

Wireframe, v češtině se lze setkat i s názvem drátěný model, lze chápat jako zjednodušený vizuální prototyp rozložení webové stránky. Nejsou pomocí něho řešeny barvy, typografie, podoba ikon ani jiná grafická podoba prvků na webu. Ukazuje pouze možnosti uspořádání a umístění obsahových prvků i ovládacích prvků na webu, členění samotné stránky a základní interakce s obsahem. Díky jednoduchosti lze wireframe snadno upravovat, měnit nebo navrhnout více variant rozložení určitého prvku. (MICHÁLEK, 2017)

Wireframe může být v podobě jednoduché skici komponent na papíře, ale i ve formě propracovanějšího modelu navrženého v programu tomu určeném, příkladem můžou být webové aplikace Figma a UXPin, případně desktopová aplikace Adobe XD. (MICHÁLEK, 2017)

2.1.2 GRAFICKÝ NÁVRH

Na základě vypracovaného drátěného modelu je realizován grafický návrh, jež je propracovanější, detailnější vizuální podoba aplikace a vybraných stránek. V něm se již pracuje s výběrem barev, písma a konkrétních multimédií. Návrh by měl respektovat rozložení a uspořádání prvků, jenž bylo stanoveno ve wireframu. Na základě grafického návrhu se následně stránky kódují do HTML a CCS. (MICHÁLEK, 2017)

V návrhu se kromě stránek navrhuje i grafický manuál aplikace (styleguide), který obsahuje: (MICHÁLEK, 2017)

- barevné schéma s vybranou primární, sekundární a konverzní barvou, s barvami pozadí a textů;
- typografii udávající vzhled použitého písma, výběr konkrétních fontů, specifikaci velikostí jednotlivých úrovní nadpisů a vzhledu odstavců;
- podobu primárních tlačítek, sekundární tlačítek, odkazů v textu a jejich vzhledu při najetí kurzoru myši;
- možné další komponenty, které se v aplikaci budou často opakovat.

2.1.3 RESPONZIVNÍ DESIGN

Responzivní design je přístup k navrhování webových stránek a aplikací, který zajišťuje optimální zobrazení obsahu na zařízeních s různými velikostmi obrazovek. Proto je již od fáze drátěného modelu a grafického návrhu zohledňován také design pro mobilní zařízení, která disponují menší velikostí displeje než klasické PC. (MICHÁLEK, 2017)

V posledních letech navíc skokově vzrůstá procentuální podíl zobrazení webů skrze mobilní zařízení, takže responzivní design je nutnost. (HOWARTH, 2024)

Návrh pro mobilní zařízení se od návrhu pro PC liší v několika aspektech – od rozdílného rozložením prvků na stránce, přes úpravu jednotlivých komponent webu, až po přizpůsobený styleguide a změny ve velikostech písma. (MICHÁLEK, 2017)

Pokud chceme mít náš design co nejlépe responzivní, měli bychom se řídit principem Mobile First. „*Mobile First je způsob návrhu uživatelského rozhraní, který z pohledu důležitosti staví mobilní zařízení minimálně na úroveň tradičních počítačů s velkými obrazovkami.*“ (MICHÁLEK, 2017, s. 194)

2.2 HTML

HTML (HyperText Markup Language) je značkovací jazyk, který se používá při tvorbě webových stránek, kde se pomocí něj vytváří obsah a struktura stránek, jež je až následně stylována do požadované vizuální podoby obsahu, viz kapitola CSS. (NIXON, 2014)

Jazyk HTML vznikl již v 90. let 20. století, za tu dobu se neustále vyvíjel až do aktuálně používané verze HTML5. To oproti dřívějším verzím přinesla řadu vylepšení například ve strukturování obsahu stránky. (CASTRO, 2022)

2.2.1 JAZYK HTML

K vytvoření struktury a obsahu se v HTML používají značky (nazývané tagy, případně elementy), pomocí kterých se popisuje webová stránka. Ty mohou být pro samotný obsah stránky jako jsou odstavce, obrázky, seznamy, odkazy a další, nebo tagy pro strukturu stránky, které nám mohou pomoci seskupovat určité tagy dohromady, lépe obsah organizovat a popisovat části webové stránky, například hlavička, patička a navigace. (CASTRO, 2022)

Tagy se zapisují do špičatých závorek a mohou být párové či nepárové. Párové tagy mají vždy počáteční tag a koncový tag, přičemž mezi ně je vkládán jejich obsah. Nepárové značky se zapisují samostatně jen počátečním tagem. V počátečním tagu se mohou vyskytovat atributy, sloužící jako popis parametrů tagu. (CASTRO, 2022)

Praktické použití tagu *<p>* (odstavec) a atributu *class* (třída) s hodnotou *popis* lze vidět na obrázku č. 1 níže.

```
<p class="popis">Tohle je text odstavce.</p>
```

Obrázek 1: Příklad použití tagu <p> v HTML (zdroj: vlastní)

Pomocí tagů se vytváří HTML dokument, který definuje strukturu a obsah webové stránky. (CASTRO, 2022)

2.2.2 STRUKTURA A INTERPRETACE HTML DOKUMENTU

Při vytváření HTML dokumentu je vhodné dodržovat základní strukturu dokumentu. Tu můžeme rozdělit na několik částí: (CASTRO, 2022)

- tag *<!DOCTYPE html>*, jenž nám značí, že se jedná o dokument jazyka HTML5;
- počáteční tag *<html>* s atributem a hodnotou *lang="cs"*, určující začátek HTML dokumentu a příslušný jazyk, ve kterém je webová stránka psána. V tomto případě tedy čeština;
- počáteční tag *<head>*, značící začátek záhlaví dokumentu;
- tag *<meta>* s atributem a hodnotou *charset="UTF-8"*, definující znakovou sadu dokumentu. Tedy konkrétně utf-8;
- počáteční tag *<title>* a konečný tag *</title>*, v nichž se nachází název aktuální webové stránky, zobrazující se například v kartě webu v prohlížeči;
- koncový tag *</head>*, uzavírající záhlaví dokumentu;

- počáteční tag *<body>* a konečný tag *</body>*, tělo dokumentu obsahující obsah stránky;
- koncový tag *</html>*, ukončující HTML dokument.

Na obrázku č. 2 je vyobrazen příklad jednoduché struktury HTML dokumentu.

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="UTF-8">
    <title>Název webu</title>
</head>
<body>
    <p>Zde bude obsah zobrazující se na stránce.</p>
</body>
</html>
```

Obrázek 2: Příklad struktury HTML dokumentu (zdroj: vlastní)

Webová stránka napsaná v HTML se interpretuje pomocí webových prohlížečů, které čtou HTML dokument na základě struktury dokumentu a daných tagů následně vykreslují uživateli veškerý obsah, jenž se nachází v těle dokumentu. (CASTRO, 2022)

2.3 CSS

CSS (Cascading Style Sheets), v češtině též jako kaskádové styly, je jazyk, pomocí něhož lze upravovat vizuální podobu HTML prvků. Díky CSS se může webové stránce dodat vlastní design. Aktuálně používanou verzí je CSS3. (CASTRO, 2022)

Ke stylování HTML využívá CSS pravidla, ty určují výběr konkrétního elementu webové stránky, pro který se bude styl uplatňovat a zajistí vizuální podobu elementu. Pravidla se mohou navzájem přepisovat a doplňovat. (NIXON, 2014)

CSS pravidlo se skládá ze: (CASTRO, 2022)

- selektoru: určuje výběr HTML elementu;
- vlastnosti: udává, co se bude vizuálně upravovat;
- hodnoty: říkající, jak bude prvek vypadat.

Níže na obrázku č. 3 je uveden příklad pravidla, jenž přiřadí elementu *p* vlastnost *color* s hodnotou *red* a vlastnost *font-weight* o hodnotě *bold*. Tento prvek odstavce bude mít červenou barvu a bude zvýrazněný tučným písmem. Je zde zároveň vidět použití selektoru podle názvu konkrétního prvku, odstavce.

```
p {
    color: red;
    font-weight: bold;
}
```

Obrázek 3: Příklad CSS pravidla (zdroj: vlastní)

Selektorem může být název tagu, ale pro přesnější určení konkrétního elementu se používají selektory *class* (třída) a *id*, což jsou jedny z možných atributů HTML prvků. Atribut *class* lze použít u více elementů najednou, naopak *id* musí být v kontextu jedné stránky unikátní a použit pouze u jednoho elementu. (NIXON, 2014)

Selektory mají svou určitou váhu, díky této vlastnosti, se dá hovořit o přepisování pravidel, kdy pravidlo s nejvyšší váhou přepíše pravidla ostatní. Nejvyšší váhu mají inline styly u HTML prvku v atributu *style*, následuje priorita *id*, *class* a nakonec názvu prvku. (NIXON, 2014)

2.3.1 ZPŮSOBY VLOŽENÍ CSS

CSS styly lze do webové stránky vložit: (NIXON, 2014)

- pomocí atributu *style* u HTML prvku, což má omezené možnosti stylování;
- pomocí tagu *<style>* v HTML souboru, jenž se aplikuje jen na konkrétní stránku;
- pomocí externího *.css* souboru s pravidly, který se bude načítat do webu přes tag *<link>*, tento způsob stylování je doporučený, protože dokáže dobře oddělit a strukturovat pravidla.

2.3.2 VYUŽITÍ CSS PRO RESPONZIVNÍ WEBOVÉ STRÁNKY

Ke tvorbě responzivní webové stránky se využívá řada CSS funkcionalit, tou hlavní je aplikování Media Queries. Dále to může být použití moderních responzivních layout řešení Flexbox a Grid Layout. (MICHÁLEK, 2017)

Media Queries jsou v podstatě podmínky umožňující aplikaci CSS pravidel na základě různých technických vlastností zařízení – například šířka zařízení, poměr stran a orientace zařízení. Nejčastěji je využívána vlastnost šířky zařízení, se kterou je spojován pojem bod

zalomení (breakpoint) Hodnota bodu zalomení říká, od jaké velikosti se začnou aplikovat podmínky Media Queries. (MICHÁLEK, 2017)

Aby bylo použití Media Queries na mobilních zařízeních vůbec patrné, musí mít HTML dokument v hlavičce tag *<meta>* se zápisem níže na obrázku č. 4. Ten říká mobilním zařízením, ať používají své přirozené rozlišení, a tím pádem ať načtou responzivní verzi webu, místo standartní desktopové verze. Ta by například na telefonu mohla být opticky malá a špatně by se ovládala.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Obrázek 4: Meta tag pro nastavení přirozeného rozlišení na mobilních zařízeních (zdroj: vlastní)

Na obrázku č. 5 lze vidět příklad použití Media Queries s podmínkou pro maximální šířku zařízení s hodnotou 767 pixelů. Pomocí tohoto zápisu bude do šířky 767 pixelů velikost nadpisu *h1* 40 pixelů. Pokud bude šířka větší, nebude se pravidlo v Media Query uplatňovat a velikost nadpisu *h1* bude 60 pixelů.

```
h1 {
    font-size: 60px;
}

@media only screen and (max-width: 767px) {
    h1 {
        font-size: 40px;
    }
}
```

Obrázek 5: Příklad použití Media Queries v CSS (zdroj: vlastní)

Pro responzivní web je dále klíčové použití responzivního layoutu. Zde je možné využít řešení pomocí flexboxů nebo ještě modernějších grid layoutů, případně jejich kombinace. Tyto řešení sází na jednoduchou úpravu rozložení prvků v nich obsažených, možnost zarovnávání prvků a řízení šířky. (MICHÁLEK, 2017)

2.4 JAVASCRIPT

JavaScript, zkráceně také JS, je vysokoúrovňový programovací jazyk, který se využívá k zajištění interaktivních a dynamických funkcí na webu. Pomocí něho je možné měnit obsah i vzhled stránek a reagovat na vzniklé události. Jeho zpracování probíhá ve webovém prohlížeči uživatele. (PEHLIVANIAN, 2014)

JS je ve vývoji od poloviny 90. let 20. století, jeho role v prostředí programovacích jazyků stále roste a využívá čím dál tím více. Název JavaScript může evokovat určité spojení s programovacím jazykem Java, ta ovšem s JS nemá nic společného a název JavaScript, byl pouze marketingovým trikem, jenž měl JS pomoci se v začátcích více prosadit a těžit z populární Javy. (NIXON, 2014)

2.4.1 JAZYK JAVASCRIPT

JS využívá k manipulaci s HTML prvky DOM model (Document Object Model) – interní reprezentace struktury HTML prvků na webové stránce, kde každý prvek v HTML dokumentu je samostatným objektem s vlastními atributy a metodami. Pro přístup k určitému HTML prvku se využívají různé způsoby jeho výběru, například: (NIXON, 2014)

- pomocí atributu id, kde se vybere jeden konkrétní prvek;
- pomocí atributu class, kde se vyberou všechny instance prvků s danou třídou;
- pomocí jména tagu, kde se vyberou všechny instance prvků daného tagu.

V příkladu na obrázku č. 6 je použita metoda výběru pomocí id prvku. Vybírá se element s atributem *id* a hodnotou *popisek*, u kterého se pomocí JS mění jeho styl, konkrétně se jedná změnu barvy textu na modrou barvu.

```
<p id="popis">Tohle je text odstavce.</p>

<script>
    document.getElementById('popis').style.color = 'blue';
</script>
```

Obrázek 6: Příklad výběru prvku pomocí id v JS (zdroj: vlastní)

JS podporuje, kromě standartního přístupu k programování také objektový přístup, což lze vidět na DOM modelu. Dále je JS slabě typovaný jazyk, u jeho proměnných se tedy nemusí uvádět datové typy. K určení datového typu používá JS interpreter. Ten ovšem nerozhoduje

tak intuitivně, jak by se mohlo na první pohled zdát. Při práci s proměnnými se musí dávat pozor a řádně se rozmyslet, co se do nich bude ukládat. (NIXON, 2021)

2.4.2 VYUŽITÍ JAZYKA JAVASCRIPT

Z počátku se JS používal jen jako jednoduchý jazyk pro úpravy webových stránek. V současnosti dospěl do stádia, kdy se přes JS můžou vytvářet komplexní funkce webových aplikací, jako jsou například validace formulářů, funkce drag and drop, dynamická úprava vzhledu obsahu stránky z důvodů lepšího zpřístupnění webu. Dále lze vytvářet i celé komponenty, příkladem mohou být galerie, responzivní mobilní navigace a slideshow. (PEHLIVANIAN, 2014)

2.4.3 ZPŮSOBY VLOŽENÍ JAVASCRIPTU

Stejně jako CSS, tak i JS je nutné do stránky vložit. K tomu lze využívat dvě metody: (PEHLIVANIAN, 2014)

- psaní kódu přímo do HTML dokumentu konkrétní stránky, kde se použije tag *<script>*, v němž se bude psát JS;
- psaní kódu do externího *.js* souboru, který je pak potřeba načítat do stránky prostřednickým tagu *<script>* a atributu *src*.

Kromě způsobu vložení JS je důležité i místo vložení. Pokud bude JS kód vložen v záhlaví HTML dokumentu může nastat chyba – když se začne vykonávat takto umístěný JS kód, tak nemusí mít ještě plně načtený DOM model HTML dokumentu. Pokud se kód bude odvolávat na HTML element skrze DOM, který ještě nebyl načten, povede to k chybám, protože prohlížeč nebude schopen najít vybraný element. Problému lze předejít umístěním JS na konec těla HTML dokumentu, případně jeho ponecháním v záhlaví dokumentu, ale s podmínku ošetření vykonávání JS až po načtení DOM modelu. (PEHLIVANIAN, 2014)

2.5 PHP

PHP (Hypertext Preprocessor) je programovací jazyk, který se využívá k dynamickým funkcím na webu. Vykonávání PHP na rozdíl od JS probíhá na straně serveru, kde se zpracuje a vygeneruje HTML, které se následně posílá klientovi. To znamená, že klient v běžných situacích nedokáže zjistit, jak vypadal původní PHP kód. (The PHP Group, 2024)

Původ PHP sahá do roku 1994, kdy byl pouze sadou několika skriptů v jazyce C, následně se jeho vývoj rozběhl a do PHP přibyla například funkce komunikace s databází. Od PHP 2.0

se jednalo o samostatný jazyk, což položilo základ jeho hlavního využití pro vývoj dynamických webových stránek. Aktuální používá verze je PHP 8.3. (The PHP Group, 2024)

2.5.1 JAZYK PHP

Aby kód v PHP fungoval, musí být uložen v souboru ve formátu *.php*, kde stále fungují běžné HTML značky. V praxi lze vzít celou stránku v HTML a vložit ji do souboru *.php*. Samotná stránka se pak zobrazuje stejně, jako by byla ve formátu *.html*. (NIXON, 2021)

Kromě samotného typu souboru musí být PHP kód umístěn mezi speciální značky, které říkají, že se jedná o PHP kód a server jej má zpracovat. Úvodní značka *<?php* se píše na začátek PHP kódu na konci se píše značka *?>*. Server kód uvnitř značek zpracuje a pošle zpět do webového prohlížeče uživatele – výstupem je obyčejná stránka v HTML bez PHP kódu. (NIXON, 2021)

Na obrázku č. 7 je možné vidět příklad kódu, který po zpracování serverem vrátí do stránky text *Ahoj světe* pomocí příkazu *echo*, což je příkaz pro přímé vypisování dat zpět na stránku.

```php
<?php
    echo "Ahoj světe.";
?>
```

Obrázek 7: Příklad vypsání dat v PHP (zdroj: vlastní)

Práce s proměnnými v PHP je podobná jako v JS. Nemusí se zde uvádět datový typ proměnné, ten je automaticky určen její hodnotou. V novějších verzích PHP, ale přibyla možnost datový typ vyžadovat a definovat. Pro práci s proměnnými je v PHP určen znak dolaru $, jenž se píše vždy před název proměnné. Textové řetězce se píši do uvozovek, čísla bez nich. Pro spojení řetězce s proměnou se používá tečka. (NIXON, 2021)

Příklad na obrázku č. 8 ukazuje práci s proměnou *$vysledek*, do které se přiřadí výsledek součtu 1 + 2. Proměnná se následně vypisuje zpět do webové stránky pomocí příkazu *echo* i s připojeným textovým řetězcem.

```php
<?php
    $vysledek = 1 + 2;
    echo "Výsledek je: " . $vysledek;
?>
```

Obrázek 8: Příklad práce s proměnnou v PHP (zdroj: vlastní)

V PHP je často používán objektový přístup k programovaní. Samotné PHP má bohatou knihovnu objektů a metod umožňujících programátorům rychlejší práci a šetřit tím napsaný kód. (NIXON, 2021)

2.5.2 VYUŽITÍ JAZYKA PHP

Hlavním využitím PHP je tvorba dynamických webových stránek, kdy se obsah generuje na základě interakcí uživatele. Příkladem je komunikace webu s databází. PHP umí obsluhovat databázi a pracovat s uloženými daty – například umožňuje čtení dat z databáze a zápis do ní. Další aplikací jazyka PHP může být práce s webovými formuláři na stránce, kde PHP umí získat data z odeslaného formuláře a dále je zpracovávat. (NIXON, 2021)

PHP se také používá jako základ pro různá CMS řešení. Jedním z nejpopulárnějších je *WordPress*, který je postaven kompletně na PHP. Při práci ve *WordPressu* si lze v URL řádku prohlížeče všimnout různých PHP souborů, se který uživatel pracuje. (NIXON, 2021)

Kromě CMS řešení se PHP používá i jako základ komplexních frameworků – příkladem mohou být *Laravel*, *Symphony* a *Nette*.

2.6 DATABÁZE A JAZYK SQL

Databáze by se dala definovat jako organizovaná struktura sloužící pro ukládání dat, ve které lze data snadno strukturovat, ukládat a vyhledávat. Jedním z typů používaných databází je relační databáze. (POKORNÝ, a další, 2020)

Relační databázi lze chápat jako množinu tabulek se sloupci a řádky, v nichž jsou data ukládána. Řádek tabulky se dá označit jako konkrétní záznam dat v databázi a sloupec tabulky jako specifický atribut dat, který má konkrétní parametry, například datový typ. Mezi tabulkami jsou relace (vazby) propojující tabulky mezi sebou pomocí primárních a cizích klíčů. (NIXON, 2021)

2.6.1 DBMS A DATABÁZOVÝ SYSTÉM

Pro práci s databázemi je potřeba DBMS (DataBase Management System), v češtině jako systém řízení báze dat. DBMS je centrální software spravující databáze, který se stará o práci a manipulaci s databázemi. (POKORNÝ, a další, 2020)

Představitelem DBMS je příkladem *MySQL*, což je jeden z nejpopulárnějších DBMS pro vývoj webových stránek. Převážně díky tomu, že je volně dostupný pro užívání a snadno přizpůsobitelný dalšímu rozvoji webu. Pro komunikaci využívá jazyk SQL. (NIXON, 2021)

Termín databázový systém je kombinace DBMS a databáze samotné. Databázi by bez DBMS nešlo obsluhovat a manipulovat s ní, tudíž je DBMS při programování a používání aplikací nezbytný. (POKORNÝ, a další, 2020)

2.6.2 JAZYK SQL

SQL (Structured Query Language) je dotazovací jazyk pro práci s relačními databázemi. Jazyk umožňuje spravovat, mazat a vytvářet databáze, dále dokáže manipulovat s daty uvnitř databáze a vytvářet nad nimi dotazy. SQL dotazem je myšlen příkaz pro manipulaci s daty. Existují příkazy pro výběr, vložení, mazání a aktualizaci dat. (POKORNÝ, a další, 2020)

Výběrovým příkazem je příkaz *SELECT*, pomocí něhož lze vybírat data z tabulek. Tento příkaz lze strukturovat do několika složek: (POKORNÝ, a další, 2020)

- složka *SELECT* obsahuje seznam všech sloupců, ze kterých se budou data vybírat;
- složka *FROM* obsahuje seznam tabulek, nad nimiž je dotaz prováděn;
- složka *WHERE* obsahuje logickou podmínku, jenž musí data splňovat, aby se do výběru dostala.

Dle Pokorného a Valenty se jedná o nejsložitější konstrukci v SQL. *„Největší logická složitost je skryta v příkazu SELECT. Již na počátku celého procesu dotazování v SQL je třeba si uvědomit, ze kterých tabulek se budou data vybírat.“* (Pokorný a Valenta, 2020, s. 75)

Uvedený příklad dotazu na obrázku č. 9 vybírá *jmeno* a *prijmeni* z tabulky *uzivatele*, za podmínky, že *role* uživatele bude rovna řetězci *admin*.

```
SELECT jmeno, prijmeni
FROM uzivatele
WHERE role = 'admin';
```

Obrázek 9: Příklad dotazu v SQL (zdroj: vlastní)

Aktualizačními příkazy jsou příkazy *INSERT*, *DELETE*, *UPDATE*, které se používají k manipulaci s daty. (POKORNÝ, a další, 2020)

- *INSERT* pro vkládání dat do databáze,
- *DELETE* pro mazání dat z databáze,
- *UPDATE* pro aktualizaci dat v databázi.

2.6.3 PHPMYADMIN

Pro přístup do databázového systému lze využít aplikaci *phpMyAdmin*, která může běžet na webovém serveru. Aplikace je nástrojem umožňující přes grafické rozhraní spravovat

relační databáze a pracovat s nimi, tabulkami a samotnými uloženými daty. Databáze se dají pomocí aplikace exportovat i importovat. (NIXON, 2021)

Přímo v aplikaci se dají vytvářet nové databáze, do kterých lze rovnou přidávat tabulky se sloupci a nastavovat jejich příslušné parametry, také lze rovnou přidávat a upravovat data. Případně se dá v aplikaci rovnou dotazovat pomocí jazyka SQL. (NIXON, 2021)

3 FRAMEWORKY

Kapitola níže se zaměřuje na frameworky – od jejich samotné definice, přes příklady frameworků až po výběr konkrétního frameworku, v němž bude realizována webová aplikace systému na správu licencí.

3.1 DEFINICE FRAMEWORKU

Framework lze definovat jako strukturu, která pomáhá při vývoji softwaru. Může se použít jako základní kámen aplikace, takže vývojář nemusí začínat s vývojem softwaru od nuly, ale může stavět na již připravených funkcích, nástrojích a souborové struktuře. Vývojář se díky frameworku věnuje více logice a psaní samotné aplikace než vytváření jejího úplného základu. To vede k úspoře času, financí, ale také k větší bezpečnosti aplikace, jelikož samotný framework a jeho zabezpečení je od autorů testováno. (Codecademy Team, 2021)

Hlavní výhody používaní frameworků: (Codecademy Team, 2021)

- předpřipravené funkcionality: zrychlení vývoje na základě již hotového základu funkcí a nástrojů v podobě frameworku;
- testovací a ladící nástroje: framework může disponovat sadou testovacích nástrojů, které zrychlují hledání chyb a ladění samotné aplikace;
- čistý a strukturovaný kód: jasně daná souborová strukturu frameworku a členění kódu přispívá k celkové přehlednosti;
- bezpečnost: framework zpravidla zahrnuje i vestavěné bezpečností funkce, jenž chrání aplikaci a její data proti běžným hrozbám, k ochraně přispívají i aktualizace frameworku, které krom nových funkcionalit mohou přinést i lepší aktuálnější zabezpečení;
- menší výskyt duplicitního kódu: struktura frameworku dovoluje psát aplikaci způsobem, kdy vývojář může používat často se opakující funkce editovatelné z jednoho místa, což vede ke snazší údržbě kódu;
- rozšiřitelnost: aplikaci psanou ve frameworku lze snadno rozšiřovat, vzhledem k tomu, že se staví pořád na stejném již připraveném základu, tak se dále programují jen další funkce a moduly aplikace.

Většinou se frameworky pojí s konkrétním programovacím jazykem, například PHP nebo JS. Zároveň se mohou dělit podle platformy, využití a komplexnosti, existují tak příkladem specifické frameworky pro vývoj aplikací na mobilní zařízení, nebo pro vývoj webových aplikací. (Codecademy Team, 2021)

3.2 PŘÍKLADY FRAMEWORKŮ

3.2.1 FRAMEWORK NETTE

Nette je český PHP framework, který se zaměřuje na bezpečnost a modularitu. Využívá strukturu MVC a obsahuje širokou škálu balíčků. Framework má kompletní dokumentaci v českém jazyce společně s českým fórem. Nette by se dalo označit i jako skupina knihoven a komponent, které jdou používat samostatně. (GRUDL, 2024)

Komponenty v Nette mají zabudované bezpečností prvky již v základu. Jednotlivé balíčky Nette ušetří vývojáři práci s vytvářením vlastních metod, jelikož obsahují velké množství funkcionalit. Nette má například balíčky pro formuláře a databáze nebo vlastní ladící nástroj Tracy. (GRUDL, 2024)

Aktuální verzí Nette je verze 3.2 běžící na minimální verzi PHP 8.1. (GRUDL, 2024)

Framework využívá šablonovací systém Latte, který je jednoduchý na použití, jelikož má intuitivní syntaxi podobnou PHP, lze v něm tedy příkladem pracovat s proměnnými a cykly. (GRUDL, 2024)

3.2.2 FRAMEWORK LARAVEL

Laravel je moderní PHP framework vyznačující se svojí jednoduchostí, elegantní syntaxí a robustností. Framework obsahuje řadu užitečných nástrojů například pro komunikaci s databází nebo pro testování aplikace. Laravel disponuje rozsáhlou dokumentací, která je pouze v anglickém jazyce. Ta obsahuje kromě klasické dokumentace i video návody. (Laravel Holdings Inc., 2024)

Aktuálně používanou verzí Laravelu je verze 11, jež vyžaduje minimální verzi PHP 8.2. (Laravel Holdings Inc., 2024)

Framework staví na MVC struktuře. Pro zobrazení obsahu využívá šablonovací systém Blade. Dále může Laravel implementovat moderní JS frameworky Vue a React pro vývoj vizuální části webu. (Laravel Holdings Inc., 2024)

Laravel nabízí vývojářům startovací balíčky aplikací, které v sobě mají vytvořené všechny potřebné funkcionality. Vývojář je schopen jedním příkazem získat připravenou aplikaci s hotovou správou uživatelů, přihlašováním a registrací do aplikace. (Laravel Holdings Inc., 2024)

3.2.3 FRAMEWORK SYMFONY

Symfony je populární PHP framework a zároveň sada samostatných opakovaně použitelných komponent do PHP. Framework má bohatou dokumentaci, která je dostupná v několika jazycích s absencí češtiny. Symfony poskytuje zdarma kompletní kurz vedoucí vývojáře od začátku vývoje až po programování pokročilejších funkcí aplikací. (Symfony SAS, 2024)

Framework je založen na konceptu MVC struktury a orientuje se hlavně na rychlost a přizpůsobitelnost. Vývojář si může nakonfigurovat jednotlivé části frameworku podle sebe a vybrat jen ty balíčky nebo komponenty, které bude potřebovat. (Symfony SAS, 2024)

Komponenty jsou v Symfony shlukovány do balíčků. Například balíček pro debuggování obsahuje několik samostatných komponent pro lazení webové aplikace. Balíčky jsou obsluhovány skrze nástroj Symfony Flex. Ten se stará o jejich kompletní správu a aktualizace. (Symfony SAS, 2024)

Aktuálně nejnovější verzí Symfony je verze 7.0.6, která vyžaduje verzi PHP 8.2 a vyšší. (Symfony SAS, 2024)

Framework využívá šablonovací systém Twig. Tento systém má jednoduchou syntaxi šablon podobnou PHP. Šablony jsou flexibilní a rozšiřitelné, jelikož si vývojář může například definovat vlastní tagy a rozšíření. (Symfony SAS, 2024)

3.3 VÝBĚR FRAMEWORKU PRO REALIZACI SYSTÉMU NA SPRÁVU LICENCÍ

Pro vývoj aplikace na správu licencí byl zvolen framework Nette. Hlavním faktorem byla jeho komplexnost a jednoduchost. Framework disponuje širokou škálou balíčků, které pokrývají hlavní potřeby webové aplikace. Například balíčky pro tvorbu formulářů, správu uživatelů a obsluhu databáze. Framework používá šablonovací systémem Latte, který má v sobě integrované automatické *escapování* (nahrazování speciálních znaků za jejich bezpečné varianty), což zvyšuje bezpečnost aplikace. Dále obsahuje efektivní ladící nástroj Tracy k debuggování aplikace. V neposlední řadě Nette nabízí dokumentaci kompletně v českém jazyce, a to včetně fóra, kde se nachází řada častých dotazů s odpověďmi.

3.4 POUŽITÍ FRAMEWORKU NETTE

3.4.1 STRUKTURA FRAMEWORKU

Framework Nette využívá strukturu MVC (Model View Controller), což je aplikační architektura s cílem obecně rozdělit jednotlivé vrstvy aplikace na části dle funkce. (GRUDL, 2024)

Rozdělení aplikace na obecná části dle MCV: (GRUDL, 2024)

- část model definuje logiku aplikace, představuje její funkční základ a je nezávislá na zbylých dvou částech;
- část controller zpracovává a vyhodnocuje interakce uživatele a volá příslušnou logiku z části model;
- část view reprezentuje výsledky požadavků z controlleru a zobrazuje předaná data v šablonovacím systému.

Tato struktura je v Nette lehce upravena, respektive její názvosloví, význam a použití je totožné. Místo klíčového slova controller používá Nette slovo presenter. Část view je nazvána výstižnějším pojmenováním template (šablona). (GRUDL, 2024)

Adresářová struktura v Nette: (GRUDL, 2024)

- složka *app*, což je hlavní složka webové aplikace, kde se nachází presentery, šablony, definice URL adres a případné další vývojářem vytvořené logické celky aplikace;
- složka *bin*, ve které se vykonávají skripty spouštěné z příkazové řádky;
- složka *config*, jež obsahuje konfigurační soubory aplikace;
- složka *log*, kde se uchovávají chybové logy;
- složka *temp*, která obsahuje všechny dočasné soubory, například cache soubory;
- složka *vendor*, kde se nacházejí zdrojové soubory Nette a jeho balíčků;
- složka *www*, což je veřejná složka odkud se webová aplikace spouští, složka zároveň obsahuje mediální soubory a CSS soubory.

Vzorovou předpřipravenou strukturu aplikace lze stáhnout v git adresáři Nette, nebo ji nainstalovat pomocí aplikace composer a příkazu *composer create-project nette/nazev-aplikace*. (GRUDL, 2024)

3.4.2 PRESENTERY

Presentery představují ve frameworku Nette jednotlivé stránky webu. V presenteru se pracuje s interakcemi uživatele, vykonávají se zde volané metody. Zároveň presenter slouží jako most mezi modelovou a zobrazovací částí. Samotný presenter je psán v jazyce PHP. (GRUDL, 2024)

Presenter je tvořen svojí třídou *NazevPresenter* a jejími metodami (*Nazev* reprezentuje jakýkoliv název dané stránky). Mezi metodami presenteru mají největší význam metody *render*. Ty se používají k vyvolání příslušné akce na stránce, výchozí metodou je *renderDefault*, která volá akci *default* při zadání URL */nazev/* příslušné stránky. Dalším příkladem může být akce *edit* s metodou *renderEdit* volající se pomocí URL */nazev/edit*.

Metody akce po svém vykonání spouštějí určenou šablonu stránky. Pro akci *default* to je šablona *default.latte*, pro akci *edit* to je *edit.latte*. Jednotlivé akce na stránce, ale nemusejí mít svoji vlastní šablonu. Tímto principem je možno na jednu konkrétní stránku nahlížet více pohledy. (GRUDL, 2024)

Jednou z dalších činností presenteru je předávání dat do šablony stránky, to probíhá v *render* metodách. (GRUDL, 2024)

Na obrázku č. 10 lze vidět předání proměnné *soucet* z presenteru do šablony stránky ve výchozí *render* metodě.

```
public function renderDefault()
{
    $soucet = 1 + 2;
    $this->template->soucet = $this->database->$soucet;
}
```

Obrázek 10: Ukázka předání proměnné z presenteru do šablony stránky (zdroj: vlastní)

Kromě předávání informací skrze proměnné může presenter na stránku zasílat informace pomocí *flash* zpráv, což jsou textové zprávy, nejčastěji informující o výsledků nějakého úkonu. Zprávy se vytvářejí pomocí metody *flashMessage*. Tyto zprávy fungují i po přesměrování na jinou stránku a dají se členit do kategorií. V aplikaci se zprávy můžou následně nastylovat podle příslušného významu. Jednotlivé kategorie jsou například *success*, *error* a *warning*. (GRUDL, 2024)

Ukázka na obrázku č. 11 představuje generování flash zprávy v presenteru. Obsahem zprávy je textový řetězec *Nejste přihlášen* a kategorie zprávy je *warning*.

```
$this->flashMessage('Nejste přihlášen', 'warning');
```

Obrázek 11: Ukázka použití flash zprávy v presenteru stránky (zdroj: vlastní)

Na obrázku č. 12 lze vidět vykreslenou flash zprávu na webové stránce.

Obrázek 12: Ukázka vykreslené flash zprávy na stránce (zdroj: vlastní)

3.4.3 URL ADRESY

URL adresy stránek se v Nette tvoří pomocí třídy *Router* v souboru *RouterFactory.php*, kde se na základě zadané URL adresy volá příslušný presenter stránky a jeho akce. V souboru jsou definované jednotlivé adresy a k nim přiřazeny presentery s akcí. Pro vložení odkazu na konkrétní stránku aplikace slouží ve frameworku jednoduchá syntaxe *název_presenteru:název_akce*. Tato syntaxe se používá, jak v presenterech stránek, tak i v šablonách. (GRUDL, 2024)

Na obrázku č. 13 je vidět příklad definované URL adresy pro stránku *kontakty*, která se nachází na adrese */kontakty/* a využívá presenter *Kontakty* s výchozí akcí.

```
$router->addRoute('<presenter>/kontakty<action>', 'Kontakty:default');
```

Obrázek 13: Ukázka definování URL adresy stránky (zdroj: vlastní)

3.4.4 ŠABLONY

Nette používá k psaní view části aplikace šablonovací systém Latte. Šablona je tvořena HTML kódem, ve kterém je tvořen obsah stránky a klíčovými slovy systému Latte psanými ve složených závorkách. Výsledný soubor má koncovku *.latte*. Syntaxe Latte je velice podobná PHP. Šablonovací systém podporuje běžné programovací struktury, takže se v šabloně můžou používat cykly, podmínky a proměnné. (GRUDL, 2024)

Obrázek č. 14 ukazuje příklad vypsání proměnné *soucet* předané z presenteru stránky a vypsané v šabloně stránky. Zároveň je použita podmínka *if*set, tedy pokud by se presenteru stránky proměnná nepředala, tak by se příslušná část stránky ani nevykreslila. (GRUDL, 2024)

```
{ifset $soucet}
    <h5>Součet je: {$soucet}</h5>
{/ifset}
```

Obrázek 14: Ukázka vypsání proměnné v šabloně stránky předané z presenteru (zdroj: vlastní)

Kromě klíčových slov Latte disponuje *n:atributy*, ty se chovají jako normální atributy HTML tagu akorát umožňují pracovat uvnitř atributu s Latte funkcemi. Tyto atributy jsou například *n:class*, *n:block* a *n:href*. (GRUDL, 2024)

Na obrázku č. 15 je znázorněno použití atributu *n:href*, který odkazuje na stránku *Nazev* a volá akci *edit*. V šabloně stránky tedy jde jednoduše odkazovat na presentery stránek bez znalosti jejich skutečné URL adresy.

```
<a n:href="Nazev:edit">Odkaz na stránku Název s akcí edit</a>
```

Obrázek 15: Ukázka použití n:atributu v šabloně stránky (zdroj: vlastní)

Latte umožňuje vkládat jednotlivé šablony do sebe a definovat hlavní šablonu. Ta se nazývá *layout*. V hlavní šabloně jde definovat základní strukturu HTML dokumentu a následně do této šablony pomocí bloků propisovat šablony presenterů. (GRUDL, 2024)

Ukázka na obrázku č. 16 prezentuje hlavní šablonu *layout.latte*, do které se pomocí příkazu *include content* vkládá obsah ze šablony presenteru definovaný v bloku *content*. Dále se ze šablony presenteru propisuje blok *title* pro název stránky.

```
<!DOCTYPE html>
<html lang="cs">
    <head>
        <meta charset="utf-8">
        <title>{include title}</title>
    </head>
    <body>

        {include content}

    </body>
</html>
```

Obrázek 16: Ukázka šablony layout (zdroj: vlastní)

Na obrázku č. 17 je vidět šablona presenteru s definovaným blokem *content* a *title*. Blok *title* je definovaný u elementu nadpisu první úrovně pomocí atributu *n:block*.

```
{block content}
    <section>
        <h1 n:block=title>Název</h1>

        <p>
            Lorem ipsum dolor sit amet, consectetur
            adipisici elit,sed eiusmod tempor incidunt
            ut labore et dolore magna aliqua.
        </p>
    </section>

{/block}
```

Obrázek 17: Ukázka šablony s blokem content (zdroj: vlastní)

4 NÁVRH SYSTÉMU NA SPRÁVU LICENCÍ

Tato kapitola se věnuje návrhu systému webové aplikace na správu licencí, kdy návrhy vychází z dat a požadavků na funkce systému poskytnutých od vedoucího práce. Kapitola se podrobněji zaměřuje na tvorbu wireframu a základního rozložení aplikace, grafického návrhu systému a návrhu databáze systému.

4.1 DEFINICE FUNKCÍ A POŽADAVKŮ NA SYSTÉM

Na začátku tvorby webové aplikace pro správu licencí byly stanoveny potřebné funkce a požadavky na systém, což pomohlo s jeho samotným návrhem. Definici těchto požadavků pomohl sestavit a definovat vedoucí práce Mgr. Miroslav Zíka, tak aby systém vyhovoval potřebám katedry a datům, jež se v aplikaci budou spravovat.

Hlavní požadavky na systém:

- zabezpečení systému, jež dovoluje zobrazit uložené informace o licencích pouze autorizovaným uživatelům po přihlášení do systému;
- integrace jednotného přihlašovacího systému ZČU Shibboleth pro přihlášení uživatelů do webové aplikace;
- funkce správy uživatelů, kteří mohou webovou aplikaci využívat, přidávání nových uživatelů a mazání současných uživatelů;
- schopnost evidovat jednotlivé záznamy licencí a jejich dat;
- funkce uživatele vkládat nové záznamy licencí, upravovat stávající záznamy licencí a v případě potřeby i stávající záznamy licencí odstranit ze systému;
- funkce uživatele vkládat nová data do záznamu licence, upravovat stávající data v záznamu licence a v případě potřeby i odstranit nepotřebná data ze záznamu licence;
- podoba evidovaných záznamů licencí by měla odpovídat struktuře a potřebám dat samotných licencí, kdy se struktura dat jednotlivých licencí může lišit.

Zároveň bylo důležité, aby kromě výše zmíněných funkcí byla navržená webová aplikace jednoduchá na ovládání, měla přehledný funkční design a responzivní chování. Zajištění těchto kroků bylo klíčové pro efektivní a uživatelsky příjemnou práci se systémem na různých typech zařízeních.

4.2 NÁVRH DATABÁZE SYSTÉMU

V návrhu vhodné podoby databáze pro webovou aplikaci systému bylo myšleno na efektivní správu a udržení konzistence uložených dat. Při návrhu databáze bylo potřeba

zvážit i možnost rozšíření funkcí systému, takže daný návrh je zároveň flexibilní k pozdějším úpravám a přidávání dalších dat do databáze.

Návrh počítal s reálnými daty, která bude databáze o licencích uchovávat. Na základě těchto informací byla databáze navržena. Reálná data licencí byla získána od vedoucího práce, který podnikl na katedře KVD šetření mezi pracovníky katedry, jehož cílem bylo získání reálných záznamů licencí.

Název atributu licence	**Hodnota atributu licence**
Název	Ashampoo Photo Converter 2
Počet kusů	20
Určeno pro	Windows
Link na stažení exe	řetězec znaků
Propojený účet	kvdlic@kvd.zcu.cz
Datum pořízení	7. prosince 2022
Objednávka	2212070035
Licenční klíče	20 unikátních klíčů

Tabulka 1: Ukázka dat licence (zdroj: KVD FPE ZČU)

Shromážděný vzorek reálných dat licencí nebyl velký, ale přesto se z něj dala vyčíst specifika informací a atributů jednotlivých licencí, které bude potřeba uchovávat. Na základě těchto dat byl vybrán počet čtyř základních atributů, jež obsahovala každá licence. Těmito atributy byly název licence, datum pořízení licence, počet licencí a systém.

Další atributy licencí byly zvoleny jako doplňkové, protože by se nevyskytovaly u více licencí zároveň. Toto řešení bylo kompromisem, aby bylo možno flexibilně spravovat data a zároveň nevznikaly zbytečné prázdné sloupce u jednotlivých záznamů licencí v databázi.

Kromě záznamů licencí bylo potřeba v databázi dále evidovat údaje uživatelů, kvůli potřebě přihlašování přes ZČU Shibboleth.

Při definování atributů jednotlivých tabulek se zároveň určily parametry atributů, například datové typy. Dále se vždy přidal atribut primárního klíče a v případně potřeby také cizího klíče.

Výsledný databázový model se skládal ze tří tabulek:

- tabulka *Licence* obsahovala základní atributy licence,
- pomocná tabulka *Licence_parametry* obsahovala další doplňková data licence,
- tabulka *Uzivatele* obsahovala data uživatelů.

Tabulka Licence

Navržená tabulka obsahovala základní atributy licence ve formě sloupců tabulky a unikátní primární klíč *ID_licence*.

Sloupce tabulky:

- *ID_licence*: primární klíč, unikátní, celé číslo;
- *Nazev*: textový řetězec;
- *Datum_porizeni*: datum;
- *Pocet_licenci*: celé číslo;
- *System*: textový řetězec.

Tabulka Licence_parametry

Navržená tabulka obsahovala další data licencí, která nejsou přímo součástí tabulky *Licence*, unikátní primární klíč *ID_parametr* a cizí klíč *ID_licence*. Další data se tvořila pomocí názvu dalšího parametru licence a jeho hodnoty, kdy přes cizí klíč *ID_licence* byla tato informace napojena do tabulky *Licence*. Tabulky *Licence* a *Licence_parametry* byly v návrhu mezi sebou propojeny vazbou primárního a cizího klíče, kdy druh vazby byl 1:N, tedy jedna licence může mít N parametrů.

Sloupce tabulky:

- *ID_ parametr*: primární klíč, unikátní, celé číslo;
- *ID_licence*: cizí klíč, celé číslo;
- *Nazev_parametru*: textový řetězec;
- *Hodnota*: textový řetězec.

Tabulka Uživatelé

Navržená tabulka obsahovala data uživatelů, konkrétně název *Orion* konta daného uživatele, jméno uživatele a unikátní primární klíč *ID_uzivatel*.

Sloupce tabulky:

- *ID_uzivatel*: primární klíč, unikátní, celé číslo;
- *Orion*: textový řetězec;
- *Jmeno*: textový řetězec.

Na obrázku č. 18 lze vidět návrh databázového modelu obsahujícího tři tabulky databáze. Tabulky *Licence* a *Licence_parametry* jsou navzájem propojeny pomocí vazby primárního a cizího klíče. Tabulka *Uzivatele* stojí samostatně nezávisle na zbylých dvou tabulkách. Dále je u tabulek vidět určení primárního klíče, případně cizího klíče a datové typy sloupců.

Licence

PK				
ID_licence	*Nazev*	*Datum_porizeni*	*Pocet_licenci*	*System*
int	varchar	date	int	varchar
1	Affinity Designer 2	02.08.2023	20	Windows

1 : N

Licence_parametry

PK	FK		
ID_parametr	*ID_licence*	*Nazev_parametru*	*Hodnota*
int	int	varchar	varchar
1	1	Propojený účet	kvdlic@kvd.zcu.cz
2	1	Activation ID	8AfPaP623htJ7n3

Uzivatele

PK		
ID_uzivatel	*Orion*	*Jmeno*
int	varchar	varchar
1	lastovka	Martin Laštovka

Obrázek 18: Návrh databáze systému (zdroj: vlastní)

4.3 STRUKTURA STRÁNEK A WIREFRAME SYSTÉMU

4.3.1 STRUKTURA STRÁNEK

Na definovaných požadavcích systému se vytvořila optimální struktura stránek webové aplikace, na které se mohlo nadále stavět v dalších krocích návrhu.

Struktura stránek vypadala následovně:

- přihlašovací stránka, kde by se uživatelé přihlašovali do systému;
- dashboard stránka, která byla navržena jako rozcestník, kde by byly nejdůležitější akce a odkazy na další stránky;
  - stránka přehledu licencí, což by byla tabulka všech záznamů licencí;
  - stránka přidání nové licence, kde by se vkládaly nové záznamy;
  - stránka zobrazení licence, kde by se vypisovaly všechny údaje o konkrétním záznamu licence a zároveň by zde mohla probíhat editace záznamu;
  - stránka přehledu uživatelů, která by zobrazovala všechny uživatele a také by umožňovala správu uživatelů.

4.3.2 WIREFRAME SYSTÉMU

Wireframe systému vznikal v programu Adobe XD na základě požadavků, které byly předem definovány. Požadavky jako takové neurčovaly přímo požadovaný vzhled a podobu aplikace, ale určovaly hlavně funkce systému. Okolo hlavních funkcí a stránek ze struktury se následně wireframe realizoval. Wireframe tvořil jednoduché rozložení stránky webové aplikace, které bylo realizováno pro rozlišení desktopového a mobilního zařízení.

Hlavní myšlenkou při tvorbě drátěného modelu byl koncept uzavření jednotlivých definovaných funkcí do samostatných celků (dlaždic), což mělo vést k přehlednosti a jednoduchosti aplikace. Uživatelé by měli jasný přehled, kde se na stránce nachází, byli by schopni nalézt potřebné informace bez zdlouhavého hledání a provádět jednotlivé úkony v systému efektivně. Jednotlivé dlaždice ve wireframu obsahovaly příkladem prvek navigace webu, rychlé volby (komponenta usnadňující práci se systémem) nebo možnost přihlášení. Koncept dlaždic počítal s tím, že každá funkce by měla svou samostatnou dlaždici na stránce a ta jasně určovala její hranice působení. Toto omezení by předcházelo situacím, kdyby si uživatel nemusel být jist, jaké ovládací prvky patří k dané komponentě. Dále se koncept dlaždic využil při rozvrhnutí rozložení stránky, kdy se dlaždice skládaly do mřížky.

Při realizaci se navrhnuly wireframy jen pro typové stránky, mezi které patřila stránka přihlašování, stránka dashboardu a obecná stránka, kde se mohla měnit hlavní funkční část stránky podle potřebného kontextu.

Návrh wireframu rozděloval obecnou stránku na tři hlavní části:

- na hlavičku webu v horní části stránky, kde se nacházel název aplikace, informace o přihlášeném uživateli a ovládání přihlašování, část s hlavičkou se vyskytovala na všech stránkách v návrhu;
- na boční panel v levé části, kde bylo navigační menu aplikace a případné rychlé volby, tato část se v návrhu proměňovala v kontextů dané stránky;
- na hlavní funkční část v pravé části stránky, kde byl nejdůležitější obsah na stránce, což mohla být například tabulka s přehledem licencí, rychlé volby nebo formulář pro přidání záznamu licence.

U wireframu byl kladen důraz na responzivní zobrazení, které plně rozvíjelo koncept funkčních celků, které se na menších rozlišeních skládaly vedle sebe a pod sebe. Rozhraní tak zůstalo stále přehledné i na mobilních zařízeních.

Na ukázce na obrázku č. 19 je možné vidět navržený wireframe pro obecnou stránku na desktopovém rozlišení aplikace. Z příkladu wireframu je zřejmé rozložení struktury stránky do tří základních částí. Komponenty a dlaždice v hlavní funkční části se můžou měnit na základě potřeby konkrétní stránky.

Obrázek 19: Wireframe obecné stránky systému (zdroj: vlastní)

Další návrhy wireframů stránek webové aplikace je možné vidět v přílohách této práce, konkrétně se jedná o přílohy č. 1, 2 a 3. Příloha č. 1 se zabývá návrhem přihlašovací stránky, příloha č. 2 se zabývá návrhem dashboardu aplikace a příloha č. 3 ukazuje možné rozložení prvků na stránce dashboardu na mobilním zařízení.

4.4 GRAFICKÝ NÁVRH SYSTÉMU

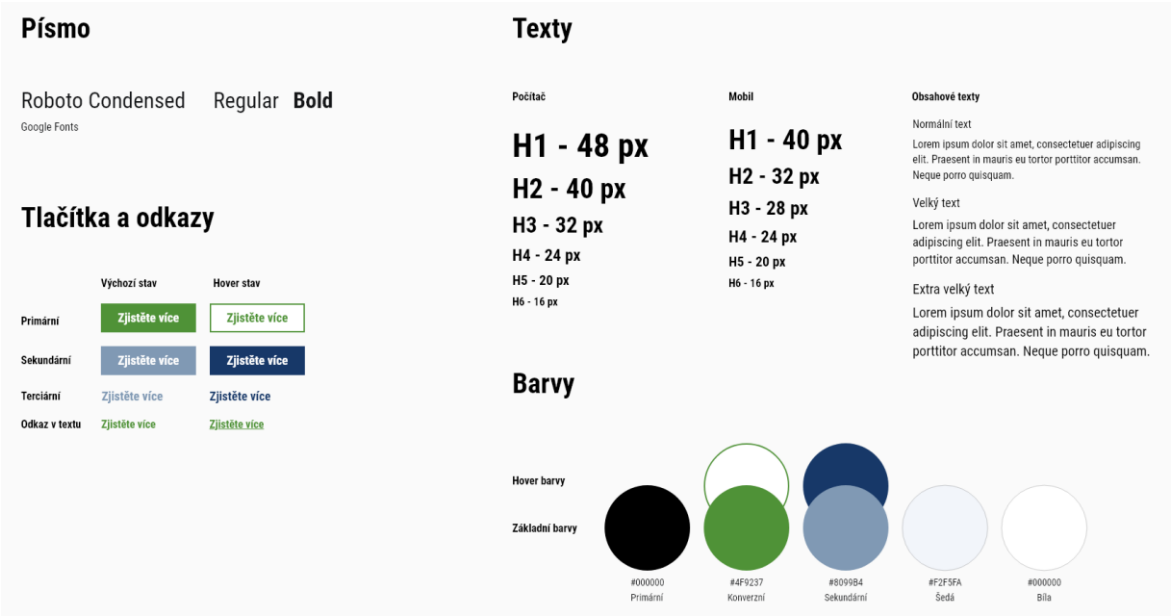
Grafický návrh webové aplikace stavěl na návrhu wireframu, jakožto základním rozvržení webu. Dále rozvíjel koncept uspořádání jednotlivých funkcí do dlaždic, aplikoval připravené rozložení stránek a strukturu webu. Návrh byl realizován v programu Adobe XD, stejně jako wireframe.

Prvním krokem před tvorbou grafického návrhu webové aplikace bylo vytvoření styleguidu aplikace, který pomohl udržet konzistentní vizuální vzhled v celém projektu a přispěl k jasnému uchopení návrhu. Styleguide definoval základní údaje a prvky, které se využívají napříč celým návrhem.

Údaje a prvky definované ve styleguidu:

- barevná paleta webu, kdy se vybrala primární, sekundární a konverzní barva, spolu s dalšími barvami pro pozadí a texty, případně doplňková barva k barvám vybraným při najetí kurzoru myši;
- typografie webu, kde se určil font *Roboto Condensed*, definovala se velikost písma pro všechny úrovně nadpisů a pro obsahové texty;
- podoba tlačítek a odkazů na webu, kdy se navrhnula podoba primárního tlačítka, sekundárního tlačítka a dalších odkazů, zároveň se určil vzhled tlačítek a odkazů při najetí kurzoru myši.

Ve styleguide se mohlo vyskytovat více definovaných údajů, ale pro potřeby aplikace to nebylo nutné. Dále se při navrhování styleguidu určila maximální vnitřní šířka aplikace, která byla stanovena na hodnotu 1180 pixelů.



Obrázek 20: Styleguide grafického návrhu systému (zdroj: vlastní)

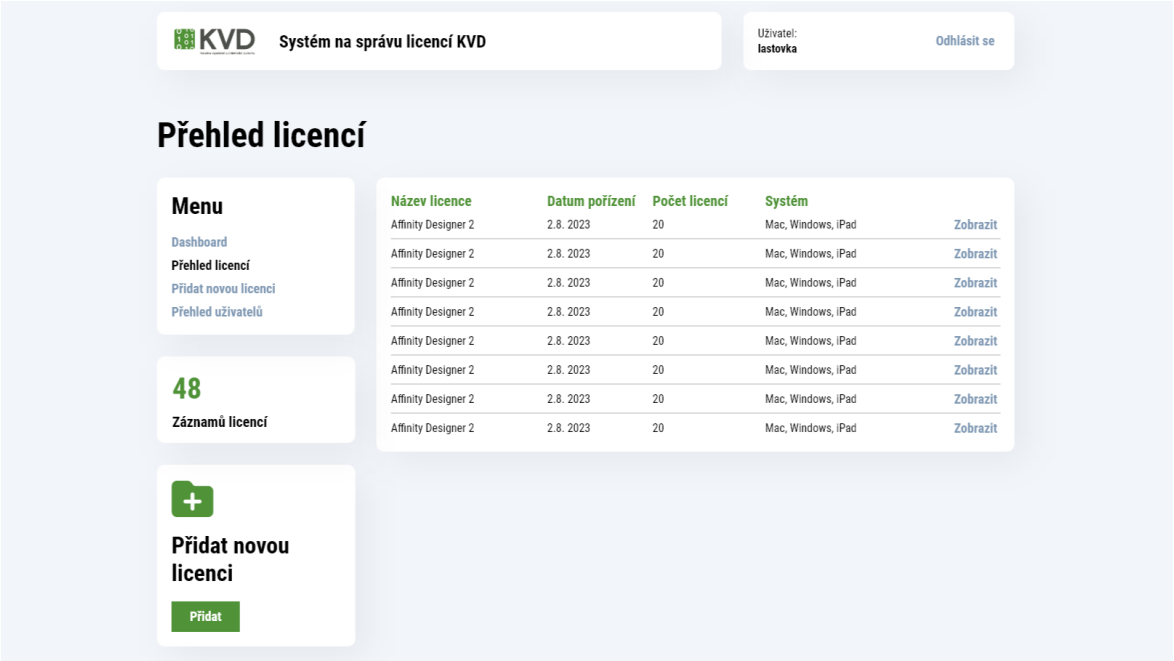
Styleguide webové aplikace se inspiroval barevností dosavadních stránek katedry KVD (www.kvd.zcu.cz), odkud převzal i použitý font. To vedlo k ucelení vizuální podoby aplikace a celkovému dojmu, že by aplikace spadala pod katedru KVD, jelikož na uživatele vystupovala podobným designovým jazykem. Ten se vyznačoval primární černou barvou, doplňkovou zelenou barvou pro konverzní prvky a sekundární světle modrou barvou, to vše fungovalo na převážně světlém pozadí.

Po realizaci styleguidu aplikace se vytvářely desktopové návrhy jednolitých stránek, dle struktury stránek, která se vytvářela společně s wireframem. Během tvorby stránek se naplno rozvinul koncept dlaždic a určila se jejich výsledná grafická podoba. Hojně

se využívala tlačítka, která pomohla s přehledností ovládání aplikace a podle úrovní důležitosti rozlišila jejich funkci jen na základě barvy nebo vzhledu. V grafickém návrhu se také určila podoba hlavních funkčních částí stránek, jež vycházely z wireframu obecné stránky.

V grafickém návrhu se kromě desktopových návrhů realizoval i jeden návrh pro mobilní zařízení, který sloužil jako vizuální demonstrátor pro rozložení prvků na menších rozlišeních.

Ukázka na obrázku č. 21 představuje navrženou stránku s přehledem licencí. Je možné vidět jednotlivé dlaždice zarovnané do mřížky, kde každá dlaždice má právě jednu funkci. Vzhled návrhu aplikuje barvy a komponenty ze styleguidu, stejně tak vybranou typografii.



Obrázek 21: Navržená stránka s přehledem licencí (zdroj: vlastní)

V přílohách práce je možno vidět další návrhy stránek, jedná se o přílohy č. 4 a 5. Příloha č. 4 zobrazuje celou kolekci grafických návrhů stránek, příloha č. 5 ukazuje grafický návrh stránky dashboardu pro mobilní zařízení, na kterém je vidět, jak se jednotlivé dlaždice skládají pod sebe.

Během realizace grafického návrhu v Adobe XD byla využita komplexnost programu, kdy se nastavila řada globálních parametrů podle styleguidu již na začátku tvorby návrhu. Takto byla implementována například barevná paleta webu, typografie a tlačítka, to vyústilo k jednotnosti návrhu a snazší práci při jeho vytváření.

5 REALIZACE SYSTÉMU

Následující kapitola se zabývá realizací systému ve formě webové aplikace, která se zde prezentuje v ukázkách jednotlivých stránek a modulů aplikace. Kromě samotné webové aplikace kapitola řeší i problematiku vytvoření databáze a práce s ní. V kapitole se využívají návrhy vytvořené v předchozí kapitole, jenž se zde transformují v nakódovanou a naprogramovanou webovou aplikaci.

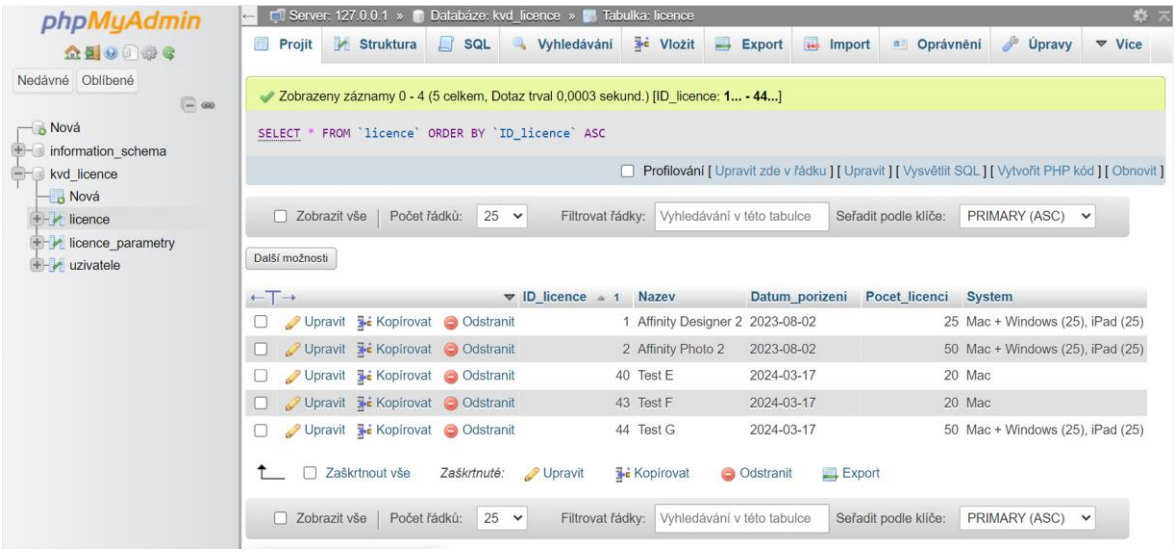
5.1 REALIZACE DATABÁZE V PHPMYADMIN

Databáze systému byla realizována v aplikaci *phpMyAdmin* na lokálním webovém serveru, kde byla celá webová aplikace systému vyvíjena. Podoba databáze odpovídala navrženému návrhu databáze z předchozí kapitoly 4.2. Databáze se tedy skládala ze tří tabulek *Licence*, *Licence_parametry* a *Uzivatele*.

Tvorba databáze probíhala čistě skrze grafické rozhraní, kde se vytvořily samotné tabulky, sloupce a jejich parametry. Nejdříve se vždy vytvořila konkrétní tabulka a následně její sloupce, u kterých se vždy vybral požadovaný datový typ na základě návrhu. Nejčastěji se jednalo o datový typ *varchar*, což je typ pro textové řetězce. Dále se používaly datové typy *int* (celé číslo) a *date* (datum). Důležitým krokem při vytváření sloupců tabulky bylo nastavit primární klíč tabulky a nastavit mu parametr *auto increment*, který zajišťuje automatické navyšování hodnoty při vložení nového záznamu do tabulky. Tato funkcionalita vedla k jednodušší práci s databází, jelikož se tento krok už nemusel řešit na straně webové aplikace.

Realizace databáze v *phpMyAdmin* měla výhodu, že šla databáze snadno exportovat ve formátu SQL, což se hodilo při předávání webové aplikace vedoucímu práce. Ten mohl databázi následně jednoduše importovat na svém prostředí.

Na obrázku č. 22 lze vidět ukázku tabulky *Licence* v databázi systému. Tabulka obsahuje několik vzorových záznamů licencí.



Obrázek 22: Vytvořená databáze v phpMyAdmin (zdroj: vlastní)

5.2 PRÁCE S DATABÁZÍ VE WEBOVÉ APLIKACI

Po realizaci databáze v *phpMyAdmin* se ve webové aplikaci musely nakonfigurovat údaje pro přístup k této databázi. Bez konfigurace přístupu by se aplikace nebyla schopná připojit k databázi a dále s ní pracovat. Nastavení probíhalo v konfiguračním souboru frameworku *local.neon*.

Obrázek č. 23 ukazuje konfiguraci připojení k databázi na lokálním serveru. Pro připojení je potřeba adresa serveru, jméno databáze, jméno účtu a heslo k účtu.

```
database:
    dsn: 'mysql:host=127.0.0.1;dbname=kvd_licence'
    user: 'root'
    password: ''
```

Obrázek 23: Nastavení přístupů do databáze (zdroj: vlastní)

Pro práci s databází byla v aplikaci vytvořena samostatná třída *DatabaseModel*, která obsluhovala všechny dotazy směřované na databázi. Tato třída využívala třídu *Connection*, ta je součástí frameworku Nette, konkrétně části *Database Core* a je hlavním nástrojem frameworku pro práci s databázemi.

Na obrázku č. 24 je možné vidět část třídy *DatabaseModel* s atributem *database* a konstruktorem. V konstruktoru třídy se realizuje napojení na třídu *Connection* předáním její instance do atributu *database*.

```php
use Nette\Database\Connection;

// Trida DatabaseModel pro pripojeni a praci s databazi
class DatabaseModel
{
    // Atribut database pro ulozeni instance tridy Connection
    private $database;

    // Konstruktor tridy DatabaseModel
    public function __construct(Connection $database)
    {
        $this->database = $database;
    }
```

Obrázek 24: Třída DatabaseModel a její konstruktor (zdroj: vlastní)

Ve vytvořené třídě *DatabaseModel* byly dále vytvořeny specifické metody pro obsluhu databáze systému, například metody pro vkládání nových záznamů licencí, úpravu stávajících záznamů licencí, získání informací o licencích nebo přidání nových uživatelů.

Pro práci s databází v presenteru konkrétní stránky bylo potřeba vytvořit instanci třídy *DatabaseModel* a následně volat příslušné metody třídy.

Na obrázku č. 25 níže je ukázka volání metody *getLicencesCount* v presenteru stránky, kde je instance třídy *DatabaseModel* uložená v atributu presenteru *database*. Data vrácená z databáze pomocí metody *getLicencesCount* se předávají dále do šablony stránky. V tomto případě se jedná o celkový počet evidovaných licencí v systému.

```php
// Vraceni dat s poctem licenci
$this->template->countLicences = $this->database->getLicencesCount();
```

Obrázek 25: Volání metody getLicencesCount v presenteru (zdroj: vlastní)

5.2.1 ČTENÍ Z DATABÁZE

Ve třídě *DatabaseModel* byly vytvořeny metody pro čtení dat z databáze, ty se používaly například pro zobrazení přehledu všech licencí. Vybraná data se převážně vracela v podobě pole, se kterým se dále pracovalo v presenteru a šabloně stránky.

Obrázek č. 26 ukazuje metodu *getLicencesAll* ve třídě *DatabaseModel*, tato metoda získává všechna data záznamů licencí z tabulky *Licence*. Následně tato data ve formě pole navrací.

```php
// Metoda vraci celou tabulku licence
public function getLicencesAll()
{
    $result = array();

    $DB_data = $this->database->query('SELECT * FROM licence ORDER BY Nazev ASC');

    foreach ($DB_data as $row) {
        $result[] = array(
            "id"=>$row->ID_licence,
            "nazev"=>$row->Nazev,
            "datum_porizeni"=>date_format($row->Datum_porizeni, "d.m.Y"),
            "pocet"=>$row->Pocet_licenci,
            "system"=>$row->System
        );
    }

    return $result;
}
```

Obrázek 26: Metoda getLicencesAll ve třídě DatabaseModel (zdroj: vlastní)

5.2.2 ZÁPIS DO DATABÁZE

Třída *DatabaseModel* kromě čtení dat z databáze realizovala přidávání nových dat a aktualizaci dat stávajících. Přidávání a aktualizování dat probíhalo čistě skrze formuláře webové aplikace. Formulář po odeslání předal data ke zpracování do presenteru stránky, kde se volala příslušná metoda třídy *DatabaseModel*. Metoda vždy po vykonání vracela stav úkonu s daty, zda byla data úspěšně uložena, nebo se zda vyskytla chyba.

Na obrázku č. 27 je vidět metoda *addLicence*, což je hlavní metoda pro přidávání nových záznamů licencí, ve třídě *DatabaseModel*. V metodě se ukazuje, jak přidávání jednotlivých předem daných údajů, tak i cyklus pro přidávání dalších parametrů licence. Vstupem metody jsou data z formuláře a po vykonání metody se vrací zpět status o provedení úkonu.

```php
// Metoda pridava novou licenci
public function addLicence($FORM_data)
{
    try {
        $this->database->query('INSERT INTO licence', [
            'Nazev' => $FORM_data['nazev'],
            'Datum_porizeni' => $FORM_data['datum_porizeni'],
            'Pocet_licenci' => $FORM_data['pocet'],
            'System' => $FORM_data['system']
        ]);

        $ID_licence = $this->database->getInsertId();
        $dalsiPoleCount = $FORM_data['dalsiPoleCount'];

        if ($dalsiPoleCount > 1) {
            for ($i = 1; $i <= $dalsiPoleCount - 1; $i++) {
                $this->database->query('INSERT INTO licence_parametry', [
                    'ID_licence' =>  $ID_licence,
                    'Nazev_parametru' => $FORM_data['dalsiNazev' . $i],
                    'Hodnota' => $FORM_data['dalsiHodnota' . $i]
                ]);
            }
            return $ID_licence;
        } else {
            return $ID_licence;
        }

    } catch (Exception $e) {
        return 'Chyba: ' . $e->getMessage();
    }
}
```

Obrázek 27: Metoda addLicence ve třídě DatabaseModel (zdroj: vlastní)

5.2.3 ODSTRANĚNÍ DAT Z DATABÁZE

Pro případ odstraňování dat disponovala třída *DatabaseModel* řadou metod k tomu určených. U těchto metod bylo důležité myslet na zachování integrity dat, proto bylo nutné při požadavku na smazání záznamu licence prohledávat obě tabulky s daty licencí – *Licence* a *Licence_parametry*. Nepomohla tedy nastat situace, kdy v jedné z tabulek mohla zůstat zapomenutá data z již smazaného záznamu licence.

Obrázek č. 28 ukazuje metodu *deleteLicenceById* ve třídě *DatabaseModel*. Metoda odstraňuje data konkrétní licence z databáze na základě id. Při odstraňování se prohledává tabulku *Licence* a *Licence_parametry*.

```
// Metoda maze licenci na zaklade id
public function deleteLicenceById($id)
{
    $result = $this->database->query('DELETE FROM licence WHERE ID_licence = ?', $id);
    $this->database->query('DELETE FROM licence_parametry WHERE ID_licence = ?', $id);

    return $result->getRowCount();
}
```

Obrázek 28: Metoda deleteLicenceById ve třídě DatabaseModel (zdroj: vlastní)

Odstranění dat ze systému bylo ošetřeno dvoukrokovým ověřením, uživatel tedy musel nejdříve kliknout na požadavek pro smazání a následně ho ještě potvrdit. V případě vytvoření požadavku na odstranění dat se uživateli vždy zobrazil výsledek provedeného požadavku, tedy zda se nějaká data smazala, nebo ne.

Na obrázku č. 29 lze vidět volání metody *deleteLicenceById*, která odstraňuje záznam licence na základě jejího id. Dále je vyobrazena konstrukce podmínky ověření smazání dat. Když se z metody databáze vrátí hodnota jedna, tak to aplikace vyhodnotí, jako úspěšné smazání dat, v opačném případě budou hodnoty znázorňovat chybný požadavek.

```
// Vraceni potvrzeni smazani licence
$result = $this->database->deleteLicenceById($id);
if ($result == 1) {
    $this->flashMessage('Licence smazána', 'success');
    $this->redirect('PrehledLicenci:');
    exit;
} else {
    $this->flashMessage('Chybný požadavek smazání licence ', 'error');
    $this->redirect('PrehledLicenci:');
    exit;
}
```

Obrázek 29: Volání metody pro smazaní licence v presenteru (zdroj: vlastní)

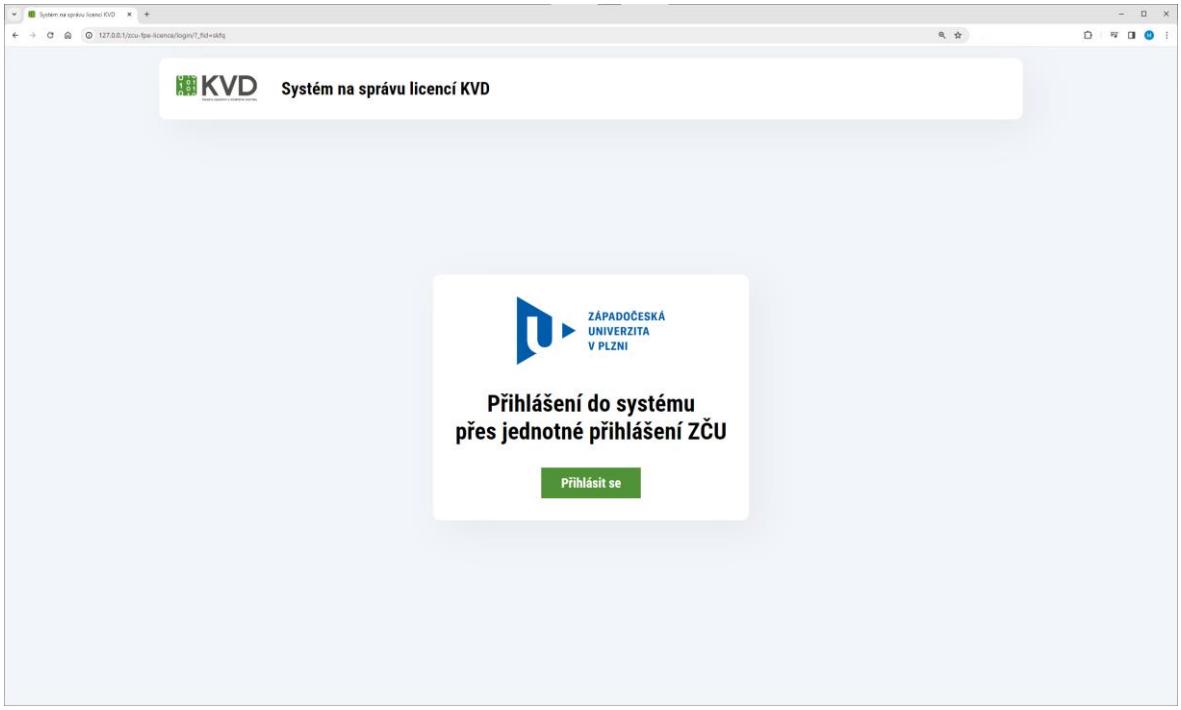
5.3 REALIZACE STRÁNEK WEBOVÉ APLIKACE

Realizace webové aplikace probíhala ve frameworku Nette, který byl vybrán v kapitole 3.3 této práce a následně se framework detailněji popisoval v kapitole 3.4.

5.3.1 REALIZACE PŘIHLAŠOVÁNÍ DO APLIKACE PŘES ZČU SHIBBOLETH

Hlavní stránkou webové aplikace pro nepřihlášené uživatele byla stránka přihlašování. Stránka integrovala přihlašování skrze jednotný přihlašovací systém ZČU Shibboleth. Pokud se chtěl uživatel přihlásit do aplikace musel projít ověřením přes Shibboleth a zároveň se vyskytovat na seznamu vybraných uživatelů. Seznam uživatelů se dal upravovat přímo ve webové aplikaci.

Stránka se skládala z hlavičky aplikace a dlaždice s přihlášením do aplikace. Na stránce nebyla žádná políčka pro vyplnění údajů, jelikož se po stisknutí tlačítka *Přihlásit se* uživatel přesměroval na přihlášení na stránkách ZČU.



Obrázek 30: Realizovaná stránka přihlášení (zdroj: vlastní)

Pokud se uživatel na stránce ZČU Shibboleth autorizoval, tak byl přesměrován zpět na stránku přihlášení se do aplikace spolu s jeho daty o přihlášení na stránce ZČU. V URL adrese stránky na přihlášení se poslal parametr *stagUserTicket* pro případnou další komunikaci se systémem ZČU, dále se poslaly parametry s informacemi o uživateli *stagUserName*, *stagUserRole* a další informace zakódované v parametru *stagUserInfo* ve formátu JSON. Aplikace následně zpracovala předaná data a vyhodnotila, zda uživatele přihlásit či nikoliv.

Přihlašování do aplikace využívalo funkci identit, což byla již připravená funkcionalita ve frameworku Nette. S identitami se pracovalo skrze třídu *SimpleIdentity*, ta je součástí balíčku *Security*, což je balíček obsahující nástroje pro autentizaci a práci s uživateli. Když se uživatel úspěšně autorizoval a vyskytoval se na seznamu uživatelů aplikace, tak mu byla vytvořena nová identita. Identita představovala sadu informací o přihlášeném uživateli, která se uchovává po celou dobu přihlášení.

Na obrázku č. 31 níže je ukázka kódu zpracovávajícího proces vytvoření nové identity *SimpleIdentity* s daty uživatele, proces se dá rozdělit do několika kroků. Nejdříve se ověří existence přijatých dat ze Shibbolethu, pak se kontroluje shoda uživatelského jména se seznamem uživatelů. Následně se vytváří nová identita uživatele z údajů přejatých ze Shibbolethu a uživatel se pod touto identitou přihlašuje do aplikace. Následně probíhá nastavení expirace přihlášení metodou *setExpiration* na interval třicet minut. Po uplynutí této doby bude uživatel vyzván k opětovnému přihlášení. Posledním krokem přihlášení je přesměrování na dashboard aplikace.

```
// Overeni zda mame data ze shibbolethu
if(!empty($_SERVER['QUERY_STRING']) && isset($_GET['stagUserTicket'])){
    $data = $this->shibbolethData($_SERVER['QUERY_STRING']);

    // Overeni zda jsou data ze shibbolethu v kompletni
    if ($data == false) {
        $this->flashMessage('Neplatný požadavek na přihlášení', 'warning');
    } else {
        foreach ($users as $usersItem) {
            // Overeni loginu
            if($data['orionLogin'] == $usersItem){

                // Vytvoreni identity na zaklade udaju z json
                $identity = new SimpleIdentity(
                    $data['orionLogin'], // ID
                    $data['role'], // Role
                    [ // Pole dalsich evidovanych dat
                        'stagUserTicket' => $data['stagUserTicket'],
                        'prijmeni' => $data['prijmeni'],
                        'jmenoPrijmeni' => $data['jmenoPrijmeni'],
                        'email' => $data['email'],
                        'role' => $data['roleNazev']
                    ]
                );

                // Vytvoreni identity
                $this->getUser()->login($identity);
                $this->getUser()->setExpiration('30 minutes');
                $this->flashMessage('Úspěšné přihlášení', 'success');
                $this->redirect('Dashboard:');
            }
        }
        // Neplatny orion username
        $this->flashMessage('Uživatel nemá přístup do systému', 'warning');
    }
}
```

Obrázek 31: Kód přihlašování do aplikace a vytvoření identity uživatele (zdroj: vlastní)

Na dalších stránkách aplikace se vždy kontroloval stav přihlášení uživatele, aby se zaručila dostupnost informací v systému jen vybrané skupině přihlášených uživatelů. To bylo zásadní k zabezpečení aplikace před neoprávněným přístupem.

Obrázek č. 32 prezentuje metodu *userLoggedIn*, která kontroluje stav přihlášení uživatele na zabezpečených stránkách. V případě, že by uživatel nebyl přihlášen a snažil se přistoupit na stránku, tak by byl automaticky přesměrován na stránku s přihlášením.

```
// Metoda overeni prihlaseni uzivatele
public function userLoggedIn()
{
    $user = $this->getUser();

    if ($user->isLoggedIn()) {

    } else {
        $this->flashMessage('Nejste přihlášen', 'warning');
        $this->redirect('Login:');
    }
}
```

Obrázek 32: Metoda ověření přihlášení uživatele na stránce (zdroj: vlastní)

5.3.2 REALIZACE DASHBOARDU

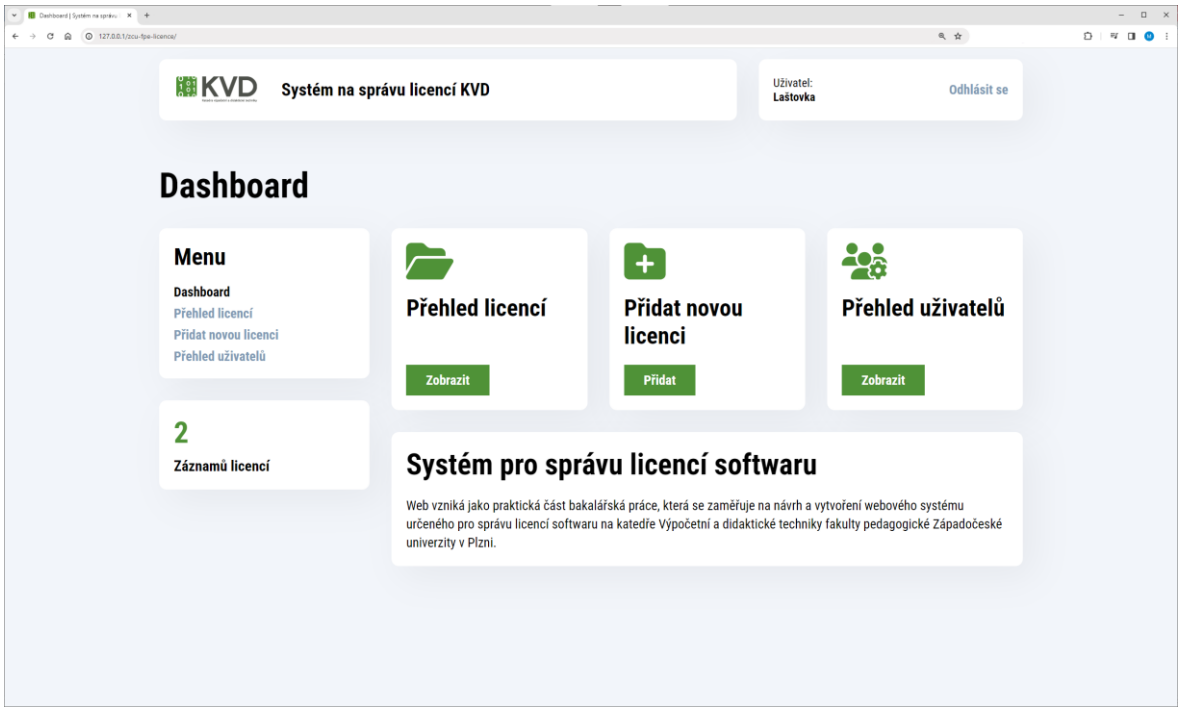
Hlavní stránkou webové aplikace po přihlášení byla stránka dashboard. Stránku tvořila hlavička systému, boční panel a hlavní funkční část obsahující dlaždice rychlé volby.

Hlavička aplikace byla rozdílná od přihlašovací stránky tím, že se rozdělila na dvě samostatné části, kde v jedné bylo logo a název systému. Ve druhé části byla informace o přihlášeném uživateli a možnost odhlásit se. Tato hlavička byla dále všech dalších stránkách.

Boční panel tvořila dlaždice s navigačním menu a dlaždice se statistikou celkového počtu licencí. Obdobný boční panel měly všechny další stránky. V panelu vždy figurovala dlaždice s navigací, zbytek dlaždic byl individuální v rámci kontextu dané stránky.

Hlavní funkční část se skládala ze tří dlaždic rychlé volby a textové dlaždice s popisem webové aplikace. Dlaždice rychlé volby tvořily odkazy na nejpoužívanější akce v systému:

- odkaz na přehled všech licencí,
- odkaz na přidání nové licence,
- odkaz na přehled uživatelů aplikace.



Obrázek 33: Realizovaná stránka dashboardu (zdroj: vlastní)

Dlaždice rychlé volby, dlaždice statistiky a dlaždice navigačního menu byly definovány jako globální bloky řízené z jednoho místa, které se upravovaly v samostatných šablonách. Do šablony stránky se jen vkládal odkaz na příslušný blok. Pokud se například prováděly změny v navigaci, tak se upravovala jen šablona bloku s navigací a změny se všude automaticky propsaly.

Na obrázku č. 34 je HTML struktura šablony dashboard stránky, kde se vkládají globální bloky. Bloky se vkládají pomocí příkazu *include* a následné cesty k bloku.

```
<section class="page__content">
    <div class="con-content">
        <div class="con-content__inner-1 con-sidebar">
            {include '../blocks/menu.latte'}
            {include '../blocks/box-stats-licence.latte'}
        </div>

        <div class="con-content__inner-2">
            <div class="con-box">
                {include '../blocks/box-prehled-licence.latte'}
                {include '../blocks/box-nova-licence.latte'}
                {include '../blocks/box-prehled-uzivatele.latte'}
            </div>

            <div class="bg__white box__flex box__default_style box__padding">
                <h2>Systém pro správu licencí softwaru</h2>
                <p class="text__large">
                    Web vzniká jako praktická část bakalářská práce, která se zaměřuje
                    na návrh a vytvoření webového systému určeného pro správu licencí
                    softwaru na katedře Výpočetní a didaktické techniky fakulty
                    pedagogické Západočeské univerzity v Plzni.
                </p>
            </div>
        </div>
    </div>
</section>
```

Obrázek 34: HTML struktura dashboardu v šabloně stránky (zdroj: vlastní)

Obrázek č. 35 níže ukazuje globální blok navigace s odkazy na jednotlivé stránky. Pokud odkaz přísluší aktuálně zobrazené stránce, přidá se mu třída *active* a přes CSS se změní jeho vizuální styl. Toto je řešeno přes atribut *n:class*, který dokáže pracovat s proměnnými, podmínkami a funkcemi. Ověření aktuální stránky probíhá přes funkci *isLinkCurrent*.

```
<div class="con-menu bg__white box__default_style box__padding">
    <h3>Menu</h3>
    <nav class="con-menu__inner">
        <ul class="menu__list">
            <li><a n:href="Dashboard:default" n:class="isLinkCurrent() ? active,
            button__tertiary, menu__link">Dashboard</a></li>
            <li><a n:href="PrehledLicenci:default" n:class="isLinkCurrent() ? active,
            button__tertiary, menu__link">Přehled licencí</a></li>
            <li><a n:href="PridatLicenci:default" n:class="isLinkCurrent() ? active,
            button__tertiary, menu__link">Přidat novou licenci</a></li>
            <li><a n:href="Uzivatele:default" n:class="isLinkCurrent() ? active,
            button__tertiary, menu__link">Přehled uživatelů</a></li>
        </ul>
    <nav>
</div>
```

Obrázek 35: HTML struktura šablony bloku menu (zdroj: vlastní)

5.3.3 REALIZACE STRÁNKY S PŘEHLEDEM LICENCÍ

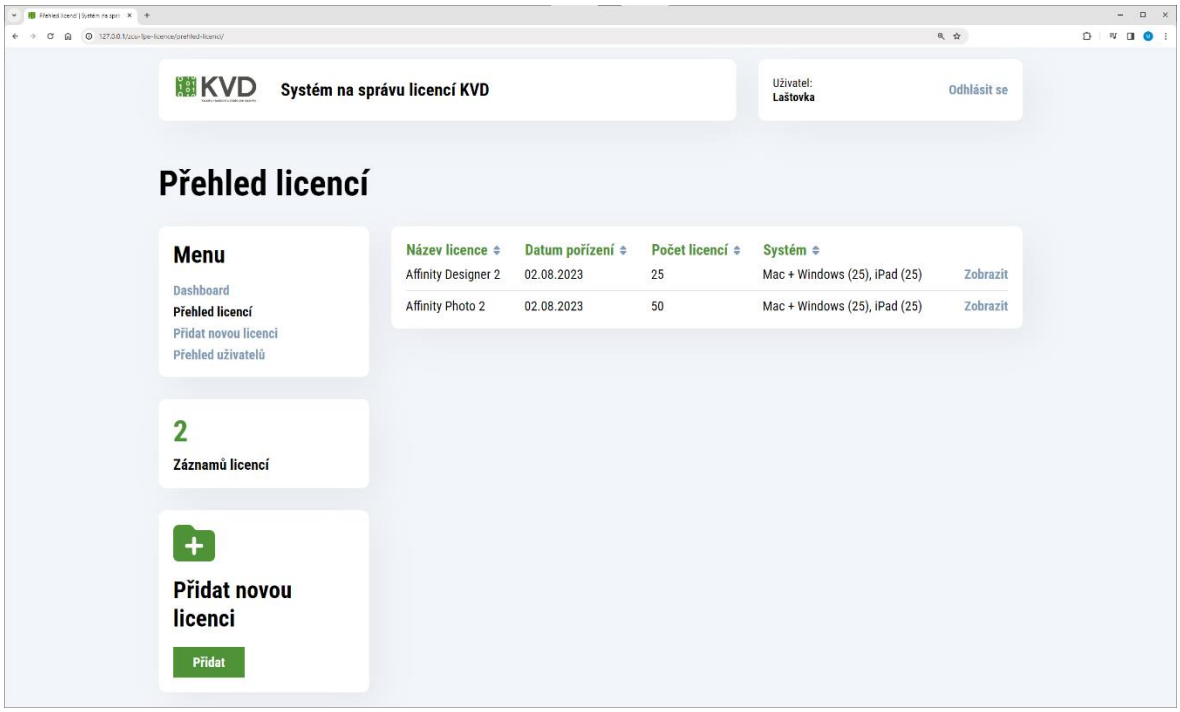
Stránka s přehledem licencí zobrazovala všechny uchované záznamy licencí v systému ve formě tabulky. Stránka se skládala z hlavička systému, bočního panelu a hlavní funkční části.

Boční panel na stránce obsahoval blok navigace, blok statistiky o počtu licencí a blok rychlé volby s odkazem na přidání nové licence.

Hlavní funkční část stránky tvořila tabulku se záznamy licencí. Tabulka obsahovala všechny prvky definované v návrhu databáze jako základní atributy každé licence. Tabulku tedy tvořil:

- sloupec s názvem licence,
- sloupec s datumem pořízení licence,
- sloupec s počtem licencí konkrétní licence,
- sloupec se systém licence,
- sloupec s tlačítkem pro zobrazení detailu licence.

Zobrazené záznamy šlo filtrovat podle hodnot v daných sloupcích, výjimkou byl sloupec s odkazem, kde tato možnost nebyla. Výchozí zobrazení licencí bylo seřazeno podle názvu licence.



Obrázek 36: Realizovaná stránka s přehledem licencí (zdroj: vlastní)

Na obrázku č. 37 je prezentována struktura tabulky záznamů licencí v šabloně stránky, do které se následně propisují data z databáze získaná v presenteru stránky ve formě pole. Vypsání jednotlivých hodnot je realizováno použitím cyklu *foreach*, jeho použití umožňuje šablonovací systém Latte. Kromě základních atributů licence se zde používá i id licence jenž pomáhá složit URL adresu dané licence.

```
<table class="text__regular table__default">
    <thead>
        <tr>
            <th>Název licence</th>
            <th>Datum pořízení</th>
            <th>Počet licencí</th>
            <th>Systém</th>
        </tr>
    </thead>
    <tbody>
        {foreach $allLicences as $row}
        <tr>
            <td>{$row['nazev']}</td>
            <td>{$row['datum_porizeni']}</td>
            <td>{$row['pocet']}</td>
            <td>{$row['system']}</td>
            <td><a n:href="Licence:default $row['id']">Zobrazit</a></td>
        </tr>
        {/foreach}
    </tbody>
</table>
```

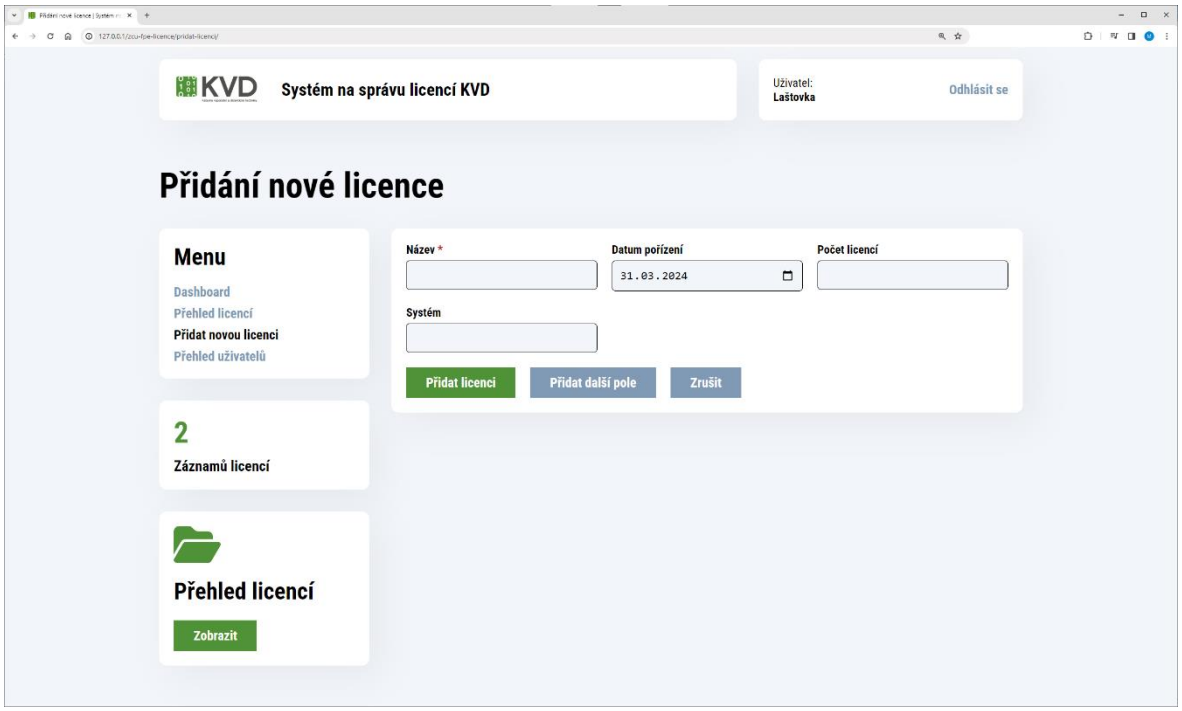
Obrázek 37: Vypsání dat licencí předaných z presenteru v šabloně stránky (zdroj: vlastní)

5.3.4 REALIZACE STRÁNKY PRO PŘIDÁNÍ NOVÉ LICENCE

Stránka pro přidání nového záznamu licence obsahovala formulář pro vyplnění dat o licenci. Skládala se z hlavičky systému, bočního panelu a hlavní funkční části.

Boční panel na stránce obsahoval blok navigace, blok statistiky o počtu licencí a blok rychlé volby s odkazem na přehled všech licencí.

Hlavní funkční část na stránce tvořil formulář pro přidání nového záznamu licence. Základní část formuláře tvořila pole pro vyplnění základních atributů licence. Další atributy licence se vkládaly pomocí dalších polí. Další pole se do formuláře přidávala stisknutím tlačítka *Přidat další pole*, kdy stisknutí vedlo k přidání dvou polí. Jedno pole bylo pro název dalšího atributu a druhé pro jeho hodnotu. Tímto způsobem šlo přidat libovolné množství dalších informací ke konkrétnímu záznamu licence. Data z dalších polí se ukládala do databázové tabulky *Licence_parametry*.



Obrázek 38: Realizovaná stránka pro přidání nové licence (zdroj: vlastní)

Přidáváním polí dalších atributů licence nedocházelo k obnovení celé stránky, ale jen k překreslení formuláře. To se realizovalo voláním AJAX požadavku (požadavek umožňující pracovat s kódem stránky bez nutnosti obnovené celé stránky) ze šablony do presenteru stránky. Požadavek předával data z formuláře do presenteru, který následně získaná data propsal zpět do formuláře spolu s nově přidanými poli, formulář se následně na stránce musel překreslit. Pro překreslení konkrétní části v šabloně stránky používá framework Nette snippety, ty označují místo určené k překreslení upraveným obsahem. Pro obsluhu snippetů a AJAX požadavků byla použita Nette knihovna *naja*.

Na obrázku č. 39 je kód v presenteru stránky, který zpracovává volaný AJAX požadavek ze šablony společně s předanými daty. Data jsou pomocí metody *setValue* nastavena jako výchozí hodnoty konkrétních polí formuláře. Posledním příkazem na obrázku je příkaz pro překreslení snippetu *formPridaniLicenceSnippet* v šabloně stránky pomocí metody *redrawControl*.

```php
if ($this->isAjax()) {
    $dalsiPoleCount = $data['dalsiPoleCount'];

    $form = $this['licenceAddForm'];

    // Nastaveni predanych hodnot zpet do poli formulare
    $form->getComponent('nazev')->setValue($data['nazev']);
    $form->getComponent('datum_porizeni')->setValue($data['datumPorizeni']);
    $form->getComponent('pocet')->setValue($data['pocet']);
    $form->getComponent('system')->setValue($data['system']);

    // Pridani dalsich poli a nastaveni jejich hodnot
    if ($dalsiPoleCount > 1) {
        $form->addGroup('Další pole');
        for ($i = 1; $i <= $dalsiPoleCount - 1; $i++) {
            $form->addText('dalsiNazev' . $i, 'Další pole ' . $i . ' - název')
                    ->setValue($data['dalsiNazev' . $i]);
            $form->addText('dalsiHodnota' . $i, 'Další pole ' . $i . ' - hodnota')
                    ->setValue($data['dalsiHodnota' . $i]);
        }

        $form->addText('dalsiNazev' . $dalsiPoleCount, 'Další pole ' . $dalsiPoleCount . ' - název');
        $form->addText('dalsiHodnota' . $dalsiPoleCount, 'Další pole ' . $dalsiPoleCount . ' - hodnota');
    } else {
        $form->addGroup('Další pole');
        $form->addText('dalsiNazev' . $dalsiPoleCount, 'Další pole ' . $dalsiPoleCount . ' - název');
        $form->addText('dalsiHodnota' . $dalsiPoleCount, 'Další pole ' . $dalsiPoleCount . ' - hodnota');
    }

    $dalsiPoleCount++;
    $form->getComponent('dalsiPoleCount')->setValue($dalsiPoleCount);

    // Prekresleni formulare s novymi poli
    $this->redrawControl('formPridaniLicenceSnippet');
}
```

Obrázek 39: Kód zpracování AJAX požadavku pro přidání dalšího pole (zdroj: vlastní)

Níže na obrázku č. 40 je vidět ukázka struktury stránky v šabloně. Kromě tlačítek formuláře je zde celý formulář generován z presenteru stránky a je umístěn ve snippetu, aby se mohl jeho obsah načítat bez obnovení celé stránky.

```latte
<div class="con-content__inner-2">
    <div class="con-licence bg__white box__flex box__default_style box__padding">

        {snippet formPridaniLicenceSnippet}
            {$licenceAddForm}
        {/snippet}

        <div class="button-group">
            <button class="button__primary" type="submit" form="form-licence-add">Přidat licenci</button>
            <a id="buttonAddField" href="#" class="button__secondary">Přidat další pole</a>
            <a class="button__secondary" n:href="Dashboard:default">Zrušit</a>
        </div>
    </div>
</div>
```

Obrázek 40: HTML struktura v šabloně se snippetem (zdroj: vlastní)

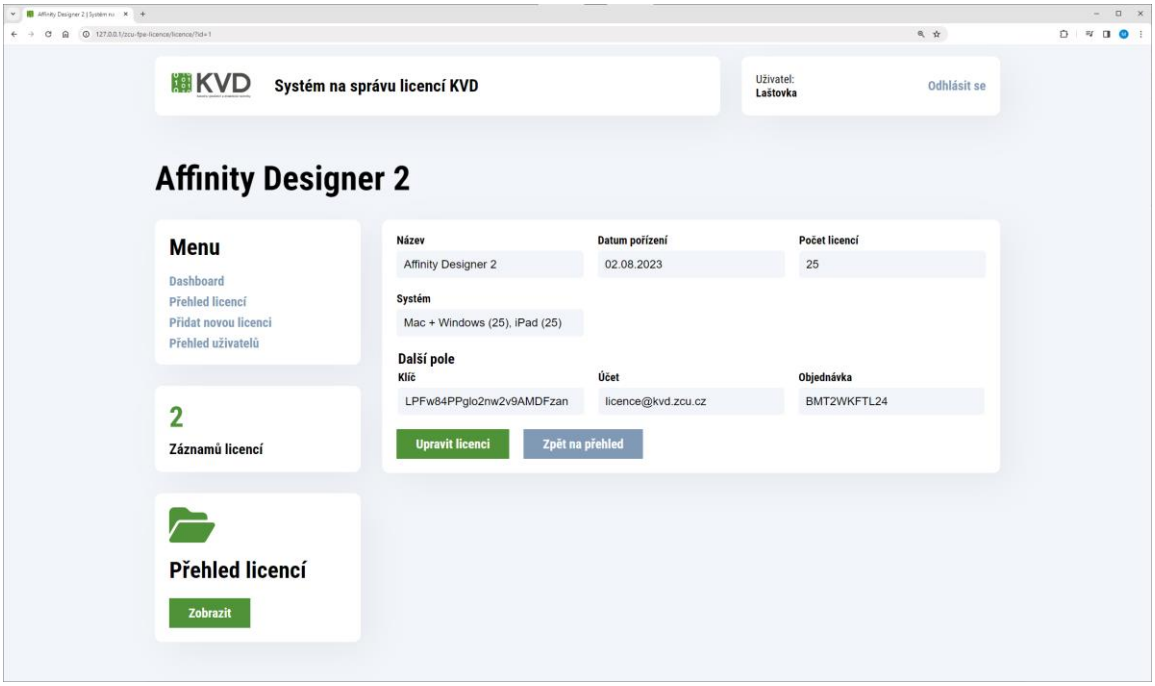
5.3.5 REALIZACE STRÁNKY PRO ZOBRAZENÍ LICENCE

Stránka zobrazení záznamu licence prezentovala data licence v polích formuláře. Tato data nešla upravovat a byla zde jen v režimu pro čtení. Stránka se skládala z hlavičky systému, bočního panelu a hlavní funkční části.

Boční panel na stránce byl totožný jako na stránce s přidáním nové licence.

Hlavní funkční část stránky tvořil formulář s informacemi o konkrétním záznamu licence. Informace byly vyplněny v příslušných polích formuláře. Jednotlivá pole měla znemožněnou možnost editace (atribut *disabled*). Formulář byl rozdělen na dvě části, kdy v první části byly základní atributy licence a ve druhé části další doplňkové atributy licence.

Ze stránky zobrazení licence šlo přejít na stránku její editace pomocí tlačítka *Upravit licenci*.



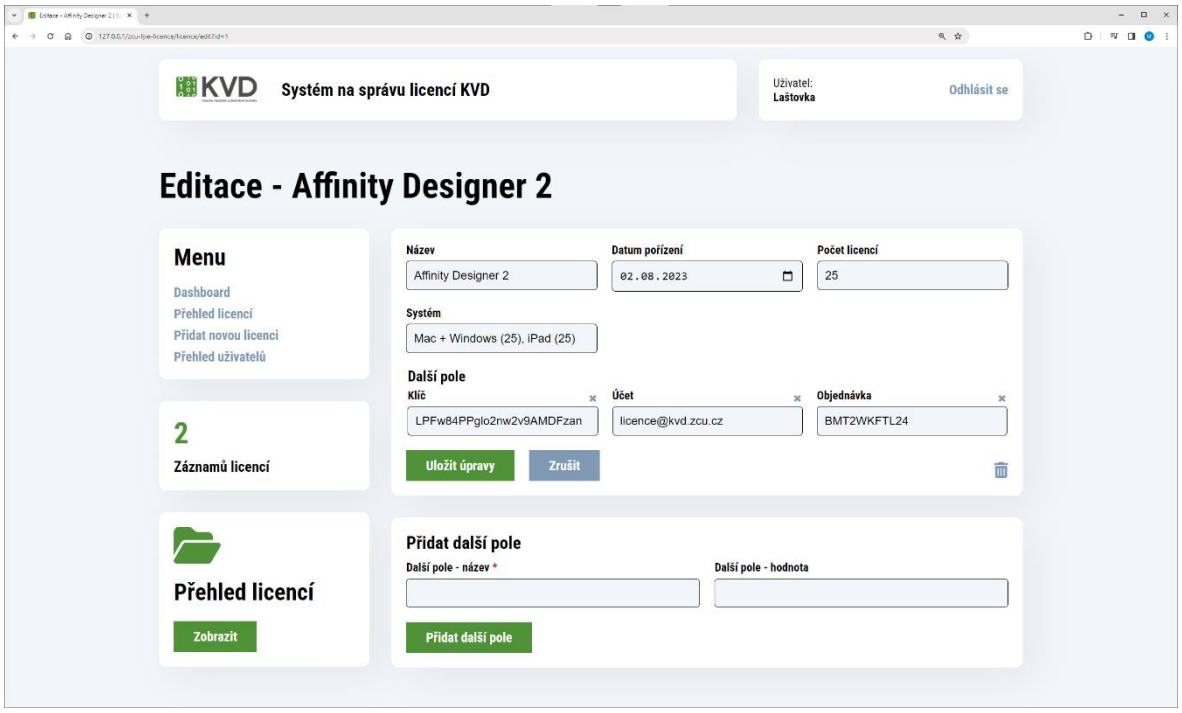
Obrázek 41: Realizovaná stránka pro zobrazení licence (zdroj: vlastní)

5.3.6 REALIZACE STRÁNKY PRO ÚPRAVU LICENCE

Stránka pro úpravu záznamu licence prezentovala data licence v editovatelných polích formuláře. Stránka se skládala z hlavičky systému, bočního panelu a hlavní funkční části.

Stránka nebyla samostatnou stránkou, ale jen zobrazením jiné akce presenteru stránky pro zobrazení licence. Pro pouhé zobrazení dat licence se používala akce *Default* (výchozí akce stránky), pro možnost úprav dat se používala akce *Edit*. Obě akce měly své vlastní šablony, které se lišily jen v hlavní funkční části stránky, jinak byly totožné.

Hlavní funkční část stránky tvořily dvě dlaždice s formuláři. První obsahovala formulář s daty licence načtenými z databáze. Políčka v tomto formuláři šla editovat a tím upravovat data licence. Formulář se skládal ze dvou částí, kdy v první části byla pole základních atributů licence a ve druhé části pole dalších doplňkových atributů licence. Jednotlivá pole s dalšími atributy šla odstranit pomocí ikony křížku nad příslušným polem, před smazáním pole vždy proběhlo potvrzení smazání. Dále se v první dlaždici ještě nacházela ikona koše, ta sloužila k odstranění celého záznamu licence, opět s potvrzením smazání. Pro uložení úprav bylo nutné stisknout potvrzovací tlačítko *Uložit úpravy*, to ukládalo pouze změnu obsahu polí formuláře. Smazání dalšího pole fungovalo nezávisle na tlačítku uložení.



Obrázek 42: Realizovaná stránka pro úpravu licence (zdroj: vlastní)

Druhá dlaždice obsahovala formulář pro přidání dalšího pole (atributu) licence. Zde bylo nutné vždy vyplnit název dalšího pole a jeho hodnotu, následně se pole přidalo stisknutím tlačítka *Přidat další pole*. Po obnovení stránky došlo k přidání dalšího pole do první dlaždice s daty licence.

Metoda *formAddField* na obrázku č. 43 zpracovává data z formuláře pro přidání dalšího pole licence, tedy název pole a jeho hodnotu. Další pole se vkládá do databázové tabulky *Licence_parametry* pomocí metody *addLicenceParameter*.

```php
// Metoda pridani dalsich poli
public function formAddField(Form $form, $data): void
{
    if (is_null($data)) {
        $this->flashMessage('Chyba načtení dat z formuláře', 'error');
        $this->redirect('Licence:edit', $this->getParameter('id'));
        exit;
    }

    $result = $this->database->addLicenceParameter($data);

    if ($result != 0) {
        $this->flashMessage('Přidáno další pole', 'success');
    } else {
        $this->flashMessage('Chyba v přidání dalšího pole'  . $result, 'error');
    }

    $this->redirect('Licence:edit', $this->getParameter('id'));
}
```

Obrázek 43: Metoda zpracování odeslaného formuláře přidání dalšího pole (zdroj: vlastní)

Před smazáním dat licence se vždy vyžadovalo potvrzení smazání, to mělo zabránit nechtěnému odstranění dalších polí formuláře a smazání celého záznamu licence.

Obrázek č. 44 prezentuje JS funkci *deleteItem* v šabloně stránky, která přidává ověření před smazáním záznamu licence. Funkce využívá JS metodu *confirm* zobrazující dialogové okno prohlížeče s možností potvrzení a zrušení. Podle rozhodnutí uživatele vrací metoda true (pravda), nebo false (nepravda). Na základě vyhodnocení podmínky se případně volá akce presenteru.

```javascript
// Funkce na potvrzeni smazani zaznamu licence
function deleteItem(ID) {
    if (confirm("Opravdu chcete smazat tento záznam licence?")) {
        window.location.href = "delete?id=" + ID;
    } else {
        return;
    }
}
```

Obrázek 44: Funkce potvrzení smazání licence (zdroj: vlastní)

Na obrázku č. 45 je vidět metoda *renderDelete*, což je akce presenteru pro smazání záznamu licence. Metoda odstraňuje licenci na základě jejího id předaného v při volání akce. Po ověření vyplněného id se licence maže prostřednictvím metody *deleteLicenceById*.

```php
// Metoda pro smazani licence
public function renderDelete($id)
{
    // Overeni prihlaseni
    $this->userLoggedIn();

    // Osetreni neexistujiciho id v URL
    if (!isset($id)) {
        $this->flashMessage('Chybná URL pro smazání licence', 'error');
        $this->redirect('PrehledLicenci:');
        exit;
    }

    // Vraceni potvrzeni smazani licence
    $result = $this->database->deleteLicenceById($id);
    if ($result == 1) {
        $this->flashMessage('Licence smazána', 'success');
        $this->redirect('PrehledLicenci:');
        exit;
    } else {
        $this->flashMessage('Chybný požadavek smazání licence ', 'error');
        $this->redirect('PrehledLicenci:');
        exit;
    }
}
```

Obrázek 45: Metoda akce Delete pro smazání záznamu licence (zdroj: vlastní)

5.3.7 REALIZACE STRÁNKY UŽIVATELŮ

Stránka uživatelů byla vytvořena ke správě uživatelů webové aplikace. Na stránce bylo možno přidat nové uživatele, kteří by měli přístup do aplikace. Stránka se skládala z hlavičky systému, bočního panelu a hlavní funkční části.

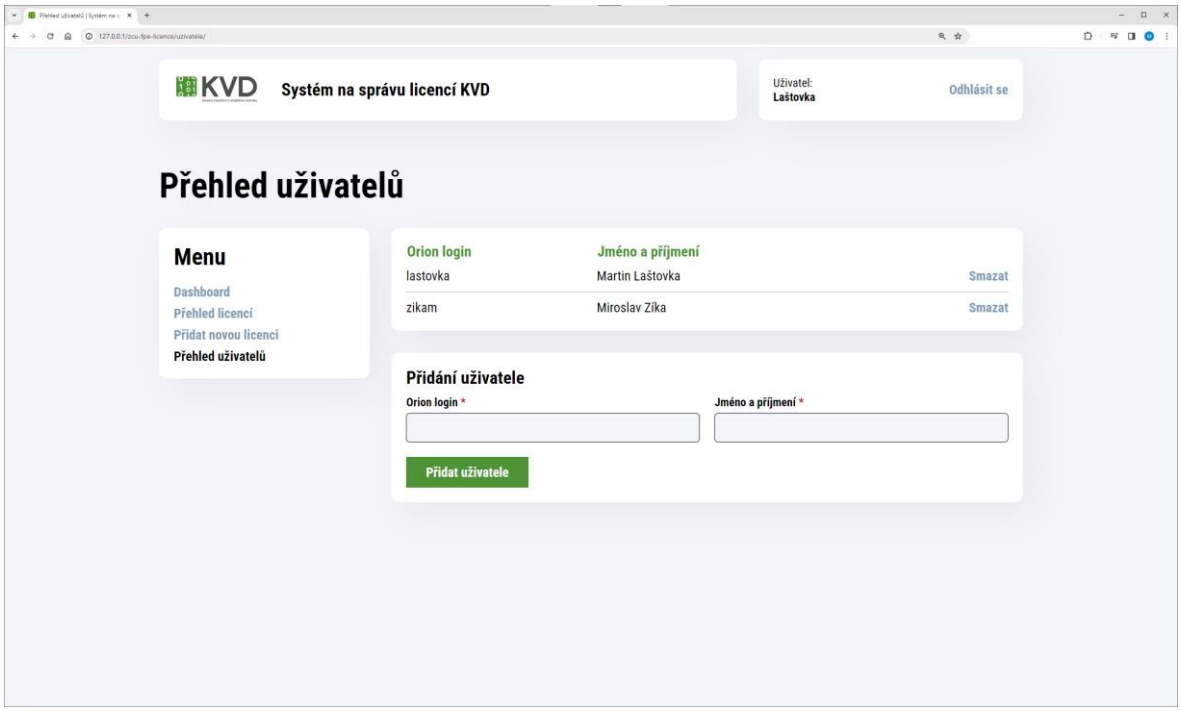
Boční panel na stránce obsahoval pouze blok navigace. Hlavní funkční část stránky tvořila tabulka s přehledem uživatelů a formulář pro přidání nového uživatele.

Tabulka obsahovala všechny uživatele a jejich informace získané z databáze aplikace. Samotnou tabulku tvořily tři sloupce:

- sloupec s orion loginem uživatele,
- sloupec se jménem a příjmením uživatele,
- sloupec s odkazem pro smazání konkrétního uživatele.

Uživatel byl schopen skrze tabulku smazat uživatele ze systému, a tedy mu zabránit v přístupu do webové aplikace. Před smazáním se zobrazilo ověření, zda uživatele opravdu smazat.

Formulář pro přidání uživatele byl relativně jednoduchý, jelikož obsahoval pouze dvě pole. Pole orion loginu uživatele a pole pro jeho jméno a příjmení.



Obrázek 46: Realizovaná stránka uživatelů (zdroj: vlastní)

Na obrázku č. 47 je ukázka metody *createComponentUserAddForm*, která generuje formulář pro přidávání uživatelů. V metodě se využívá třída *Form* knihovny *Forms* frameworku Nette a její metody, například metoda pro přidání textového pole formuláře *addText*. Po vytvoření celého formuláře vrací metoda kompletní formulář k propsání v šabloně stránky.

```
// Metoda vytvoreni formulare pro upravu dat licence
protected function createComponentUserAddForm(): Form
{
    $form = new Form;
    $form->addProtection();
    $form->setHtmlAttribute('class', 'form_record_add');
    $form->setHtmlAttribute('id','form-user-add');
    $form->addHidden('id');
    $form->addText('orion', 'Orion login')
            ->setRequired('Zadejte orion login');
    $form->addText('jmeno', 'Jméno a příjmení')
            ->setRequired('Zadejte prosím jméno');

    $renderer = $form->getRenderer();
    $renderer->wrappers['controls']['container'] = 'div class="con-form"';
    $renderer->wrappers['pair']['container'] = 'div class="con-form__item"';
    $renderer->wrappers['label']['container'] = null;
    $renderer->wrappers['control']['container'] = null;

    $form->onSuccess[] = [$this, 'formSaveData'];
    return $form;
}
```

Obrázek 47: Metoda vytvoření formuláře pro přidání uživatele (zdroj: vlastní)

6 NÁVOD NA PRÁCI SE SYSTÉMEM

Tato kapitola se věnuje návodu na instalaci a na práci se systémem. Kapitola vysvětluje fungování systému na scénářích, které pokrývají funkcionality systému.

6.1 INSTALACE A ZPROVOZNĚNÍ SYSTÉMU

Pro zprovoznění aplikace systému je požadován webový server s minimální verzí PHP 8.2. Zároveň je potřeba, aby webový server podporoval soubory *.htaccess*, které systém používá pro správu odkazů. Mohlo by se jednat například o server Apache.

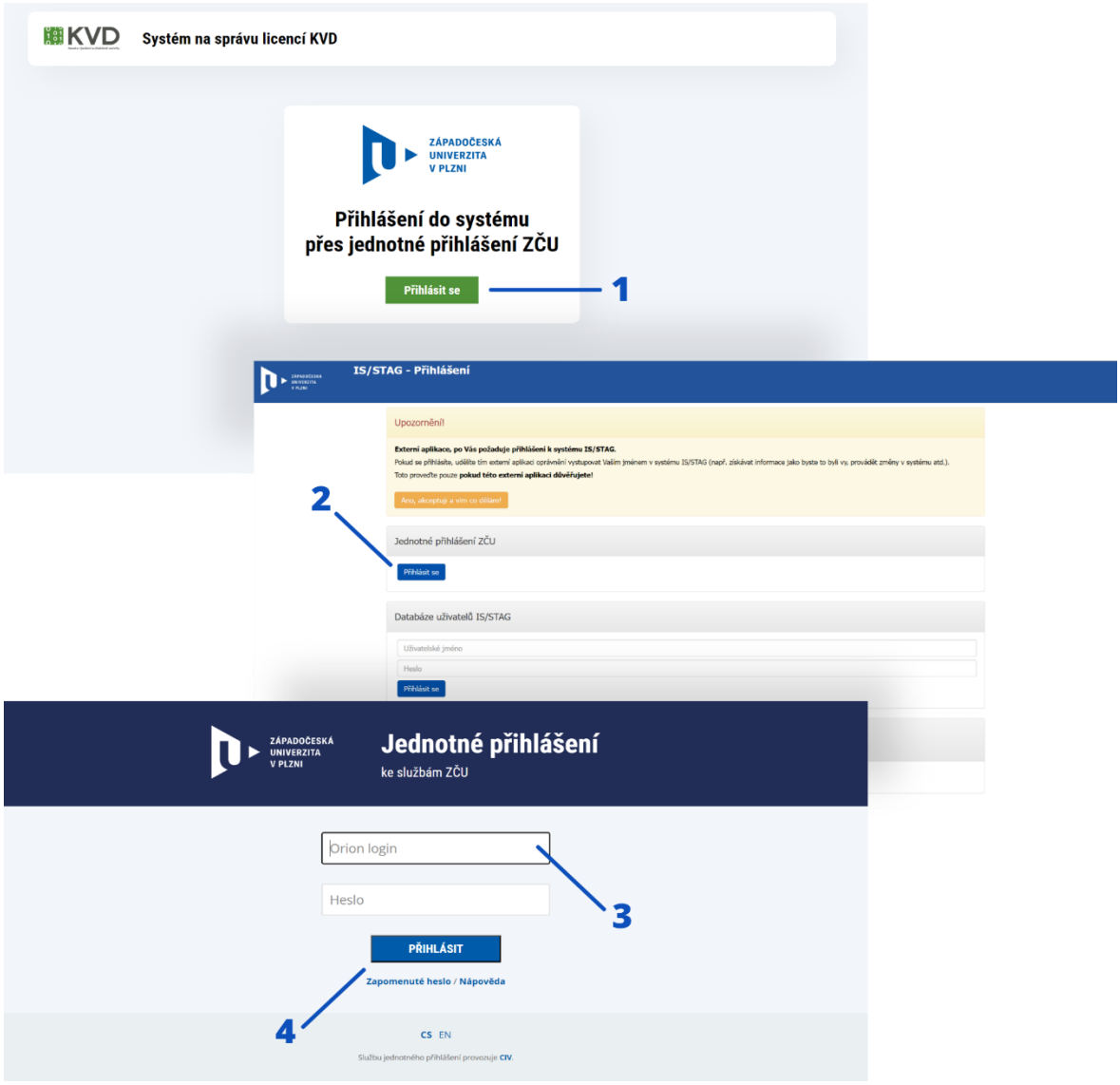
Soubory webové aplikace jsou obsaženy v příloze č. 6 této práce. Příloha obsahuje archiv souborů *aplikace.zip*. V tomto archivu se nachází zdrojové kódy aplikace, vyexportovaná databáze ve formátu SQL a návod na zprovoznění aplikace.

Pro instalaci aplikace je nutné nejdříve extrahovat soubory z archivu *aplikace.zip*. Soubory aplikace jsou ve složce *zcu-fpe-licence* a je potřeba je umístit do příslušného adresáře na hostingu, obvykle je tento adresář nazývám *www*. Pro správné fungování aplikace je dále nutné pro složky *temp* a *log* nastavit zápisová práva, jelikož se do složek ukládají dočasné soubory a chybová hlášení aplikace.

Databáze aplikace je obsažena v souboru *kvd_licence.sql*. Databázi je možné importovat skrze aplikační rozhraní databázového serveru na hostingu. Pro zprovoznění připojení k databázi aplikace je potřeba upravit údaje v souboru *local.neon*, který se nachází ve složce *config*. Zde se definují údaje pro připojení k databázi jako adresa databázového serveru, jméno databáze, uživatelské jméno a heslo uživatele. Aktuální údaje odpovídají databázovému serveru na lokálním vývojovém prostředí.

Posledním krokem, aby bylo možné se do aplikace přihlásit je změna URL adres pro zpětné přesměrování z přihlašovací stránky ZČU v šabloně stránky pro přihlašování *default.latte*. Šablona se nachází v adresáři */app/Presenters/templates/Login/*. Zde je potřeba změnit hodnotu parametru *originalURL* v URL adresách odkazujících na přihlašování ZČU. Novým parametrem bude doména, kde se systém zprovozňuje. Podoba nových URL odkazů bude ve formátu *https://stag-ws.zcu.cz/ws/login?originalURL=NAZEV_DOMENY/login/*, kde řetězec *NAZEV_DOMENY* představuje hodnotu, kterou je nutné nahradit.

6.2 PŘIHLÁŠENÍ SE DO APLIKACE



Obrázek 48: Návod na přihlášení do systému (zdroj: vlastní)

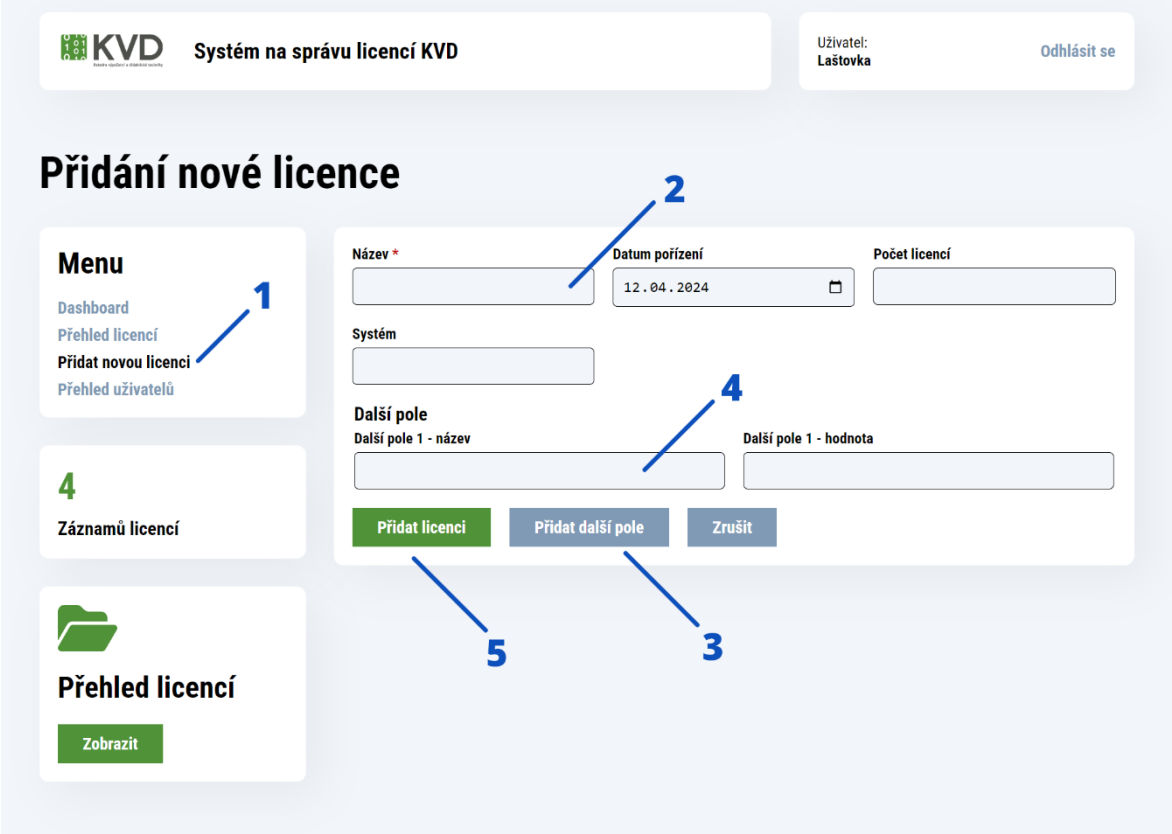
Přihlašování do aplikace systému je řešeno přes jednotné přihlašování ZČU. Uživatel musí být zároveň na seznamu povolených uživatelů, jinak ho aplikace nepřihlásí a nedostane se do systému. Po úspěšném přihlášení je uživatel přesměrován na stránku dashboard.

Popis kroků pro přihlášení se do webové aplikace:

- krok 1: stisknout na přihlašovací stránce aplikace tlačítko *Přihlásit se*;
- krok 2: po přesměrování kliknout na tlačítko *Přihlásit* se v sekci jednotného přihlášení ZČU;
- krok 3: na stránce jednotného přihlášení ZČU vyplnit potřebné údaje, *Orion login* a *Heslo*;
- krok 4: na stránce přihlášení ZČU stisknout tlačítko *Přihlásit*.

6.3 SCÉNÁŘE PRÁCE SE SYSTÉMEM

6.3.1 PŘIDÁNÍ LICENCE



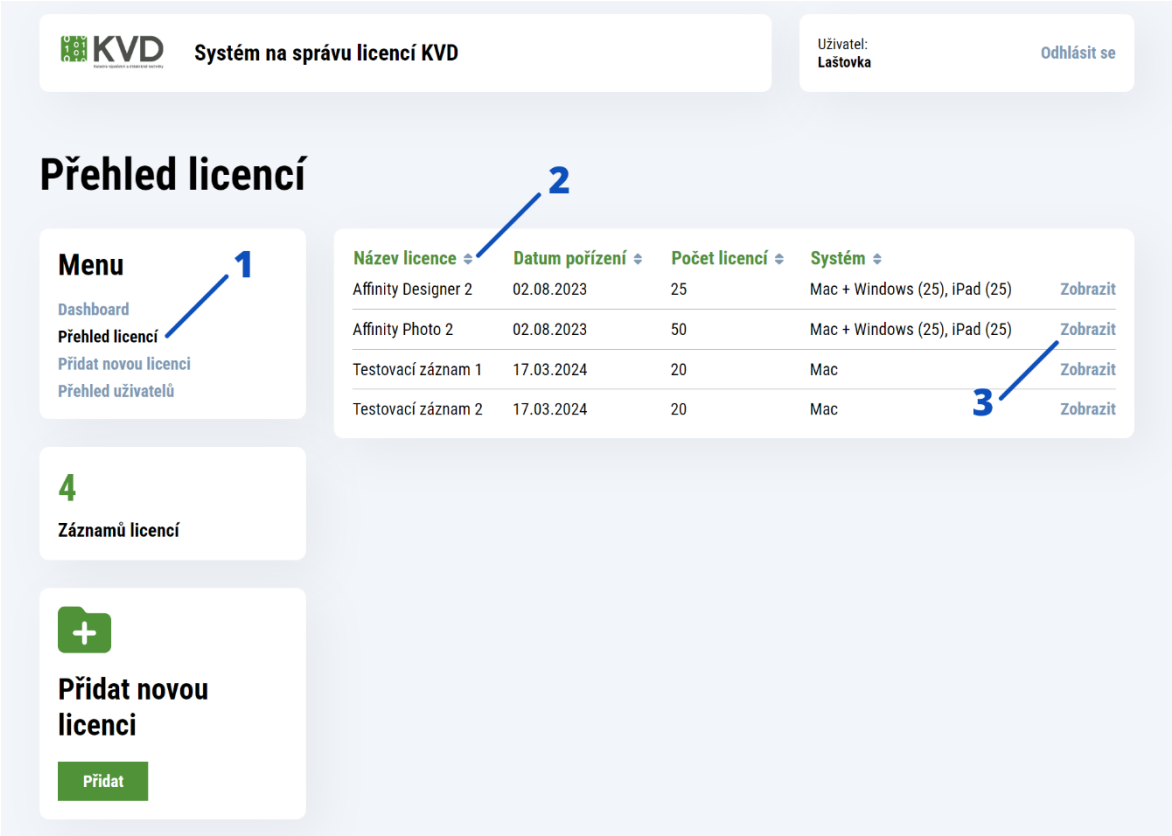
Obrázek 49: Návod na přidání nového záznamu licence (zdroj: vlastní)

Pro přidání nového záznamu licence je nutné vyplnit políčka formuláře, která obsahují jednotlivé atributy licence. Další atributy lze přidat přes funkci další pole.

Popis kroků pro přidání nového záznamu licence:

- krok 1: vstoupit na stránku přidání licence skrze odkaz v menu aplikace;
- krok 2: vyplnit jednotlivá pole formuláře s atributy;
- krok 3: v případě potřeby přidat další pole licence pomocí tlačítka *Přidat další pole*;
- krok 4: vyplnit další pole, pokud bylo nějaké přidáno;
- krok 5: potvrdit přidání záznamu licence stisknutím tlačítka *Přidat licenci*.

6.3.2 ZOBRAZENÍ LICENCE



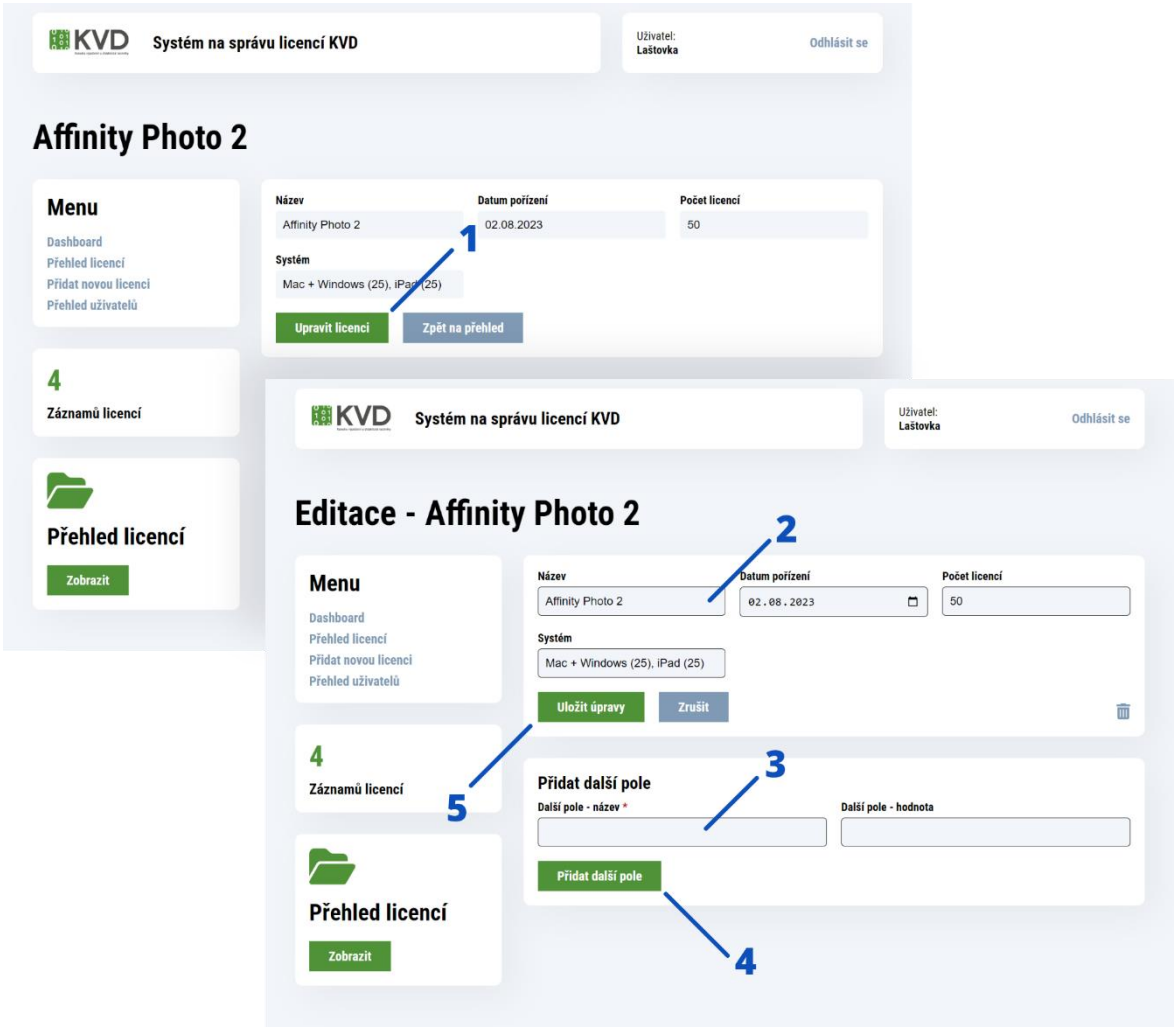
Obrázek 50: Návod na zobrazení záznamu licence (zdroj: vlastní)

Zobrazení záznamů licencí je řešeno tabulkou, kde jsou vypsány hlavní atributy licence. Pomocí těchto atributů se dají záznamy dále řadit. Po výběru konkrétního záznamu se otevře stránka s detailem licence.

Popis kroků pro zobrazení záznamu licence:

- krok 1: vstoupit na stránku s přehledem licencí skrze odkaz v menu aplikace;
- krok 2: pokud bude potřeba, použít možnost řazení licencí podle atributů pomocí ikon vedle názvů atributů;
- krok 3: vybrat si konkrétní záznam licence a zobrazit jeho detail pomocí tlačítka *Zobrazit*.

6.3.3 UPRAVENÍ LICENCE



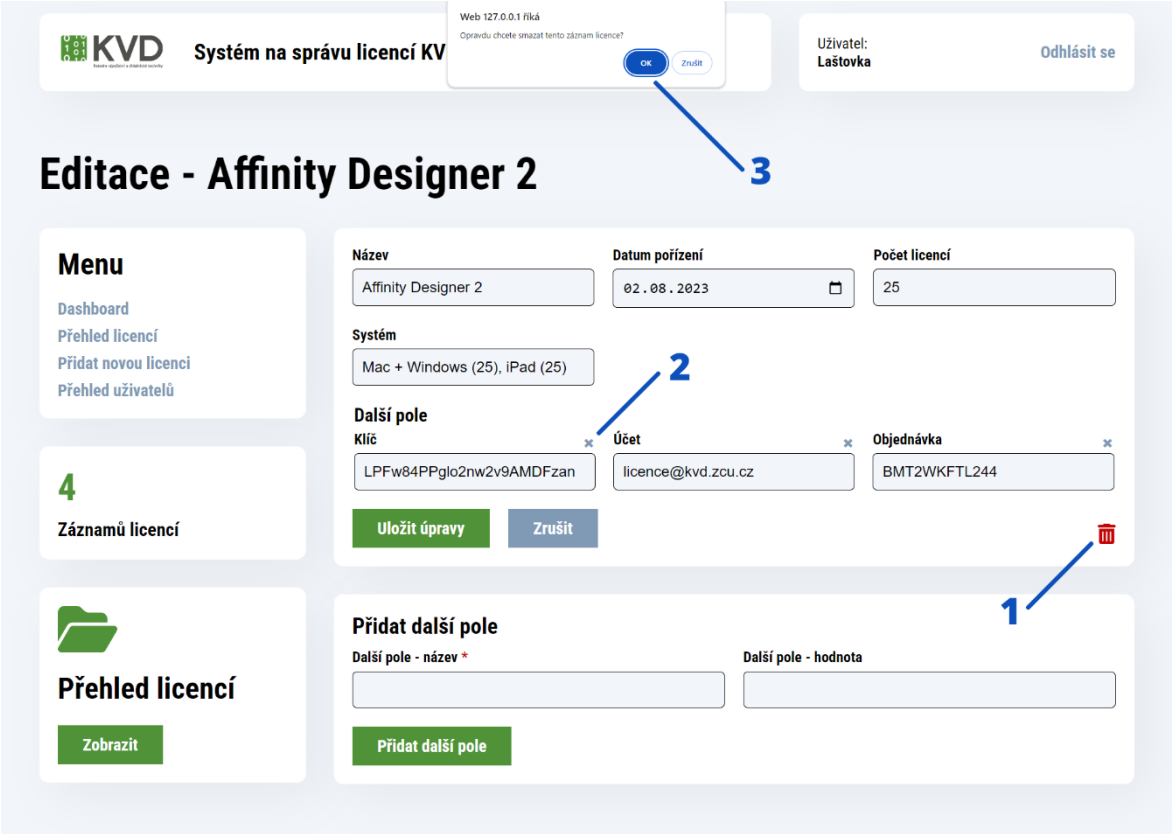
Obrázek 51: Návod na úpravu záznamu licence (zdroj: vlastní)

Editace licence je řešena pomocí polí formuláře, do kterých se načtou data o licenci. Data v polích se mohou upravit a přepsat na nové hodnoty. Zároveň je zde možnost přidat nová další pole pro evidenci dalších atributů licence.

Popis kroků pro upravení záznamu licence:

- krok 1: vstoupit do režimu editace záznamu ze stránky detailu licence pomocí tlačítka *Upravit licenci*;
- krok 2: upravit jednotlivá pole atributů licence;
- krok 3: v případě potřeby přidání dalšího pole licence vyplnit název a hodnotu dalšího pole;
- krok 4: pokud byly vyplněny údaje dalšího pole, tak jej přidat pomocí tlačítka *Přidat další pole*;
- krok 5: uložit provedené úpravy pomocí tlačítka *Uložit úpravy*.

6.3.4 ODSTRANĚNÍ LICENCE A ATRIBUTŮ LICENCE



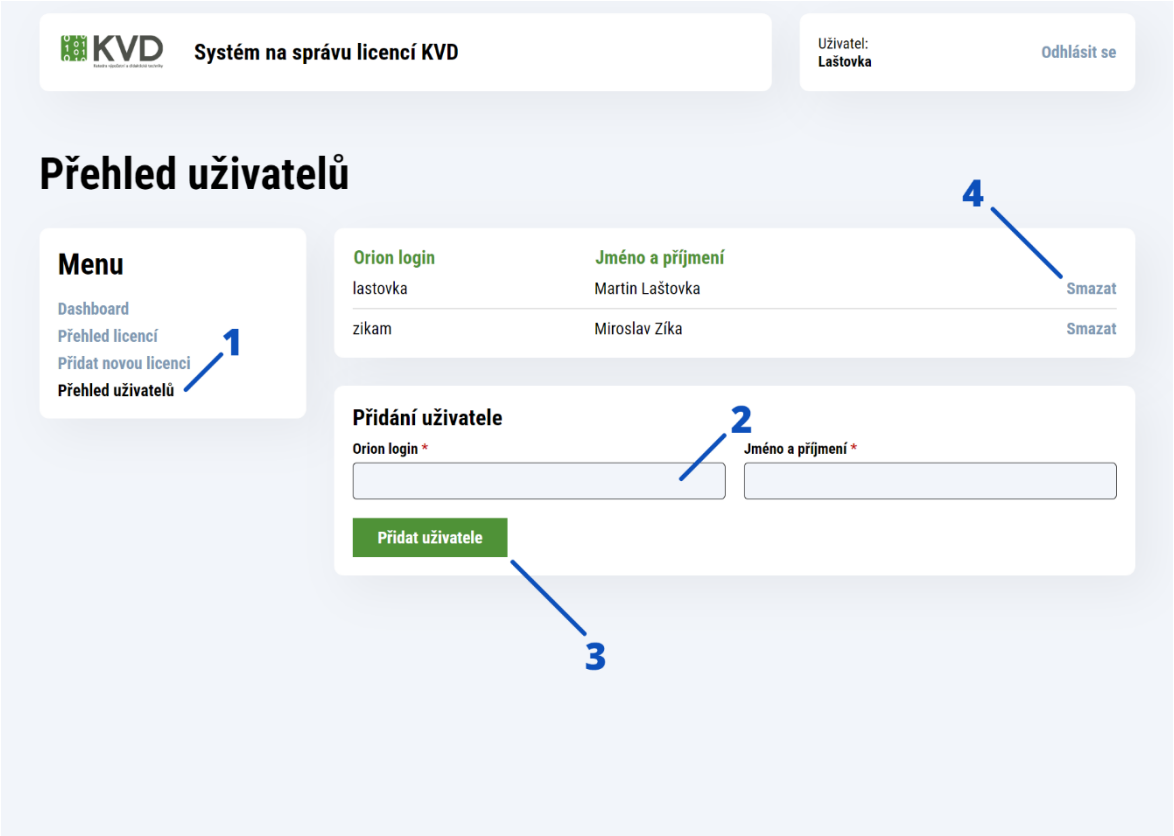
Obrázek 52: Návod na smazání záznamu licence a parametrů licence (zdroj: vlastní)

Pro odstranění záznamu licence a parametrů licence slouží příslušné ikony v editačním režimu licence. Požadavek smazání je po kliknutí na ikonu ještě nutno potvrdit.

Popis kroků pro odstranění licence a atributů licence:

- krok 1: pro smazání celého záznamu licence je potřeba kliknout na ikonu koše v editačním režimu záznamu;
- krok 2: v případě odstraňovaní dalšího pole licence kliknout na ikonu křížku nad příslušným dalším polem;
- krok 3: potvrdit požadavek smazání pomocí dialogového okna prohlížeče.

6.3.5 SPRÁVA UŽIVATELŮ



Obrázek 53: Návod na práci se správou uživatelů (zdroj: vlastní)

Správa uživatelů umožňuje přidávat nové uživatele na seznam uživatelů aplikace. Dále se skrze správu dají uživatelé smazat, což jim zabrání v přihlášení se do aplikace.

Popis kroků pro práci ve správě uživatelů:

- krok 1: vstoupit na stránku s přehledem uživatelů skrze odkaz v menu aplikace;
- krok 2: pro přidání nového uživatele je nutné vyplnit jeho Orion login a jméno do polí formuláře;
- krok 3: pokud byly vyplněny údaje uživatele, tak jej přidat pomocí tlačítka *Přidat uživatele*;
- krok 4: v případě odstraňovaní uživatele kliknout na odkaz *smazat* u příslušného uživatele a následně potvrdit požadavek smazání pomocí dialogového okna prohlížeče.

ZÁVĚR

Cílem bakalářské práce bylo navrhnout a vytvořit webový systém pro správu licencí softwaru na katedře Výpočetní a didaktické techniky fakulty pedagogické Západočeské univerzity v Plzni. Systém řešil centrální evidenci záznamů licencí, jelikož v době realizace práce takový systém na katedře KVD neexistoval a záznamy byly roztříštěny na více místech.

Práce se dělila na teoretickou a praktickou část. V teoretické části byly popsány technologie použité při vývoji systému. Praktická část byla dále rozdělena na návrhovou a realizační část.

V návrhové části se na základě poskytnutých dat o licencích vytvořily návrhy systému a vybral se framework pro realizaci aplikace. Wireframe, grafický návrh, návrh databáze a návrh struktury aplikace byly později využity při vývoji webové aplikace.

Realizační část praktické části se zabývala tvorbou aplikace systému ve vybraném frameworku. V práci byla popsána tato část na ukázkách jednotlivých stránek aplikace. Fungování aplikace bylo dále popsáno na návodech, které pokrývaly funkce systému.

RESUMÉ

Bakalářská práce se zabývá návrhem a následnou tvorbou webového systému pro správu licencí softwaru.

Práce je rozdělena na teoretickou a praktickou část. V teoretické části jsou popsány technologie použité při vývoji systému. V praktické části jsou představeny návrhy systému a realizace samotné webové aplikace.

Úvodní dvě kapitoly práce se věnují cílům práce a teoretickému úvodu k technologiím vyvíjené webové aplikace.

Další kapitoly se řadí do praktické části práce. Je v nich vybrán framework, ve kterém je aplikace systému vyvíjena. Navrhují se zde grafické návrhy aplikace a databáze. Nakonec je zde kapitola věnovaná realizaci webové aplikace systému.

Poslední kapitola práce se věnuje fungování samotného systému, které je vysvětleno na návodech, jež pokrývají funkce systému.

Výsledkem bakalářské práce je vytvořená webová aplikace systému pro správu licencí softwaru. Tato aplikace naplňuje předem stanovené požadavky vedoucího práce.

RESUMÉ

This bachelor thesis is focused on the design and following development of a web system for the management of software licenses.

The thesis is divided into theoretical and practical parts. The theoretical part describes the technologies used in the development of the system. The practical part presents the design of the system and the implementation of the web application itself.

The first two chapters of the thesis are related to the objectives of the thesis and the theoretical introduction to the technologies of the developing web application.

The next chapters are part of the practical part of the thesis. They select the framework in which the system application is developed. The graphical designs of the application and the database are proposed here. At the end, there is a chapter dedicated to the development of the web application of the system.

The last chapter of the thesis is dedicated to the system operation itself, which is explained using tutorials that cover the system functions.

The result of the bachelor thesis is a web application for software license management system. The application accomplishes the predefined requirements of the thesis supervisor.

SEZNAM LITERATURY

CASTRO, Elizabeth a HYSLOP, Bruce. 2022. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek.* 2. vydání. Brno : Computer Press, 2022. ISBN 978-80-251-5045-0.

Codecademy Team. 2021. What Is a Framework? *Codecademy.* [Online] 23. 09. 2021. [Citace: 2024-04-01]. Dostupné z: https://www.codecademy.com/resources/blog/what-is-a-framework/.

GRUDL, David. 2024. 7 důvodů, proč používat Nette. *Nette Framework.* [Online] 2024. [Citace: 2024-04-01]. Dostupné z: https://nette.org/cs/10-reasons-why-nette.

—. 2024. Dokumentace Nette. *Oficiální dokumentace Nette.* [Online] 2024. [Citace: 2024-01-05]. Dostupné z: https://doc.nette.org/.

—. 2024. Jak fungují aplikace? *Nette Dokumentace.* [Online] 2024. [Citace: 2024-04-05]. Dostupné z: https://doc.nette.org/cs/application/how-it-works.

—. 2024. Latte je synonymum bezpečnosti. *Latte šablony.* [Online] 2024. [Citace: 2024-04-01]. Dostupné z: https://latte.nette.org/cs/safety-first.

—. 2024. Latte tagy (makra). *Latte šablony.* [Online] 2024. [Citace: 2024-04-05]. Dostupné z: https://latte.nette.org/cs/tags.

—. 2024. Nette. *Nette Framework.* [Online] 2024. [Citace: 2024-04-01]. Dostupné z: https://nette.org/cs/.

—. 2024. Presentery. *Nette Dokumentace.* [Online] 2024. [Citace: 2024-04-05]. Dostupné z: https://doc.nette.org/cs/application/presenter.

—. 2024. Slovníček pojmů. *Nette Dokumentace.* [Online] 2024. [Citace: 2024-04-01]. Dostupné z: https://doc.nette.org/cs/glossary.

—. 2024. Šablony. *Nette Dokumentace.* [Online] 2024. [Citace: 2024-04-05]. Dostupné z: https://doc.nette.org/cs/application/templates.

—. 2024. Údržba a kompatibilita s PHP. *Nette Framework.* [Online] 2024. [Citace: 2024-04-01]. Dostupné z: https://nette.org/cs/maintenance.

—. 2024. Vytváření odkazů URL. *Nette Dokumentace.* [Online] 2024. [Citace: 2024-04-05]. Dostupné z: https://doc.nette.org/cs/application/creating-links.

—. 2024. Začínáme s Latte. *Latte šablony.* [Online] 2024. [Citace: 2024-04-01]. Dostupné z: https://latte.nette.org/cs/guide.

HOWARTH, Josh. 2024. Internet Traffic from Mobile Devices (Jan 2024). *Exploding Topics.* [Online] 05. 01. 2024. [Citace: 2024-01-18]. Dostupné z: https://explodingtopics.com/blog/mobile-internet-traffic.

Laravel Holdings Inc. 2024. Frontend. *Laravel.com.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://laravel.com/docs/11.x/frontend.

—. 2024. Installation. *Laravel.com.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://laravel.com/docs/11.x.

—. 2024. Installation. *Laravel.com.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://laravel.com/docs/11.x/installation.

—. 2024. Starter Kits. *Laravel.com.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://laravel.com/docs/11.x/starter-kits.

MICHÁLEK, Martin. 2017. *Vzhůru do (responzivního) webdesignu.* Praha : vlastním nákladem autora, 2017. ISBN 978-80-88253-00-6.

NIXON, Ronin. 2021. *Learning PHP, MySQL & JavaScript: a step-by-step guide to creating dynamic websites.* Sixth edition. Beijing : O'Reilly Media, 2021. ISBN 978-1-492-09382-4.

—. 2014. *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5.* Sebastopol : O'Reilly Media, 2014. ISBN 978-1-491-91866-1.

Patrik, Štípek. 2019. *Porovnání PHP Frameworků.* [Online] Bakalářská práce, Hradec Králové: Univerzita Hradec Králové, Fakulta informatiky a managementu, 2019. [Citace: 2024-04-01]. Dostupné z: https://theses.cz/id/yzzixn/STAG91077.pdf.

PEHLIVANIAN, Ara a NGUYEN, Don. 2014. *JavaScript okamžitě.* Brno : Computer Press, 2014. ISBN 978-80-251-4163-2.

POKORNÝ, Jaroslav a VALENTA, Michal. 2020. *Databázové systémy.* 2. přepracované vydání. Praha : Česká technika – nakladatelství ČVUT, 2020. ISBN 978-80-01-06696-6.

Symfony SAS. 2024. Installing & Setting up the Symfony Framework (Symfony Docs). *Symfony.com.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://symfony.com/doc/current/setup.htm.

—. 2024. Introduction. *Documentation - Twig.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://twig.symfony.com/doc/3.x/intro.html.

—. 2024. Symfony Releases. *Symfony.com.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://symfony.com/releases.

—. 2024. The technological benefits of Symfony in 6 easy lessons. *Symfony.com.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://symfony.com/six-good-technical-reasons.

—. 2024. What is it about? *Symfony.com.* [Online] 2024. [Citace: 2024-04-10]. Dostupné z: https://symfony.com/doc/6.2/the-fast-track/en/0-intro.html.

The PHP Group. 2024. History of PHP. *PHP.net.* [Online] 2024. [Citace: 2024-01-24]. Dostupné z: https://www.php.net/manual/en/history.php.php.

—. 2024. What is PHP? *PHP.net.* [Online] 2024. [Citace: 2024-01-24]. Dostupné z: https://www.php.net/manual/en/intro-whatis.php.

SEZNAM OBRÁZKŮ, TABULEK, GRAFŮ A DIAGRAMŮ

PŘÍLOHY

Příloha 1: Wireframe přihlašování do systému (zdroj: vlastní).

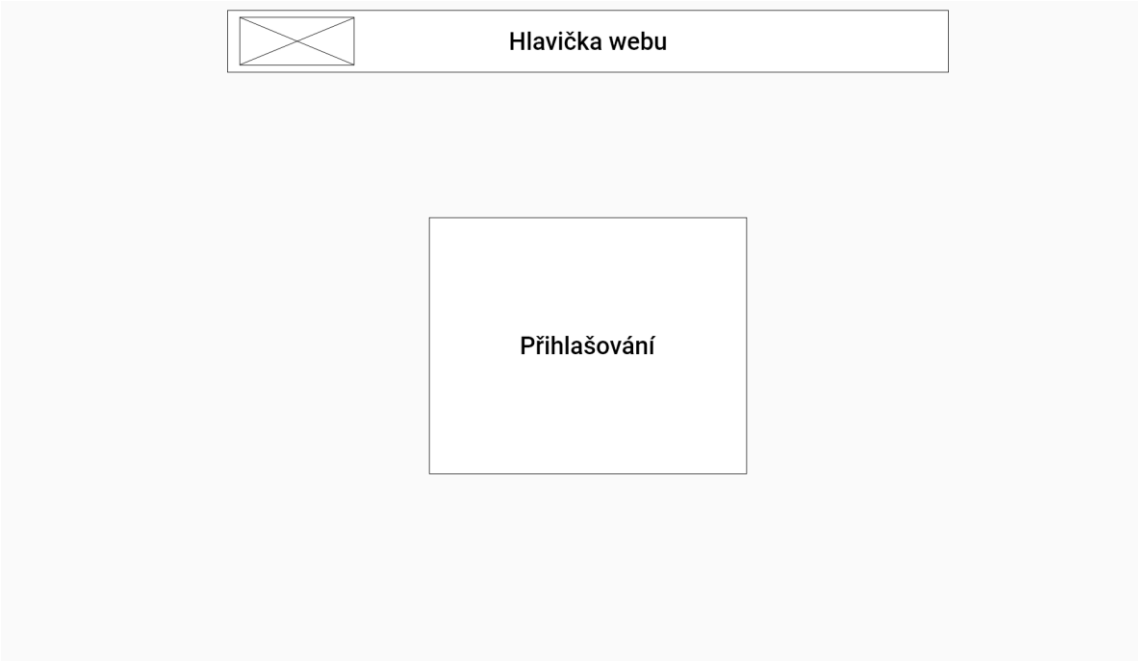
Příloha 2: Wireframe dashboardu systému (zdroj: vlastní).

Příloha 3: Wireframe dashboardu systému na mobilní zařízení (zdroj: vlastní).

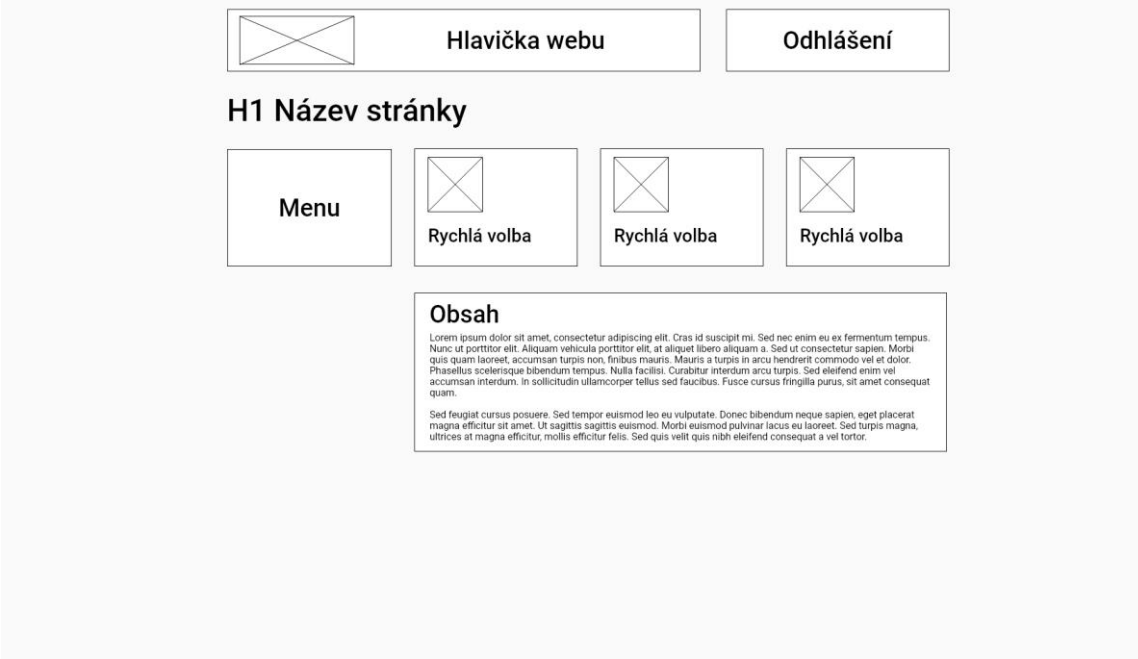
Příloha 4: Navržené stránky v grafickém návrhu systému (zdroj: vlastní).

Příloha 5: Navržená stránka dashboardu pro mobilní zařízení (zdroj: vlastní).

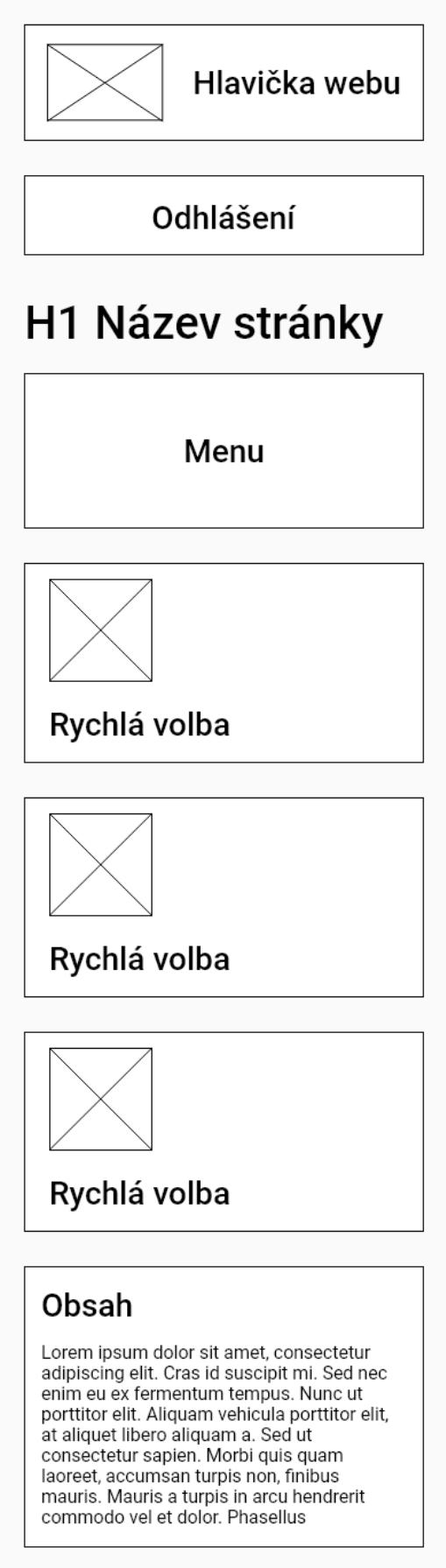
Příloha 6: Zdrojové soubory webového systému a návod na jeho zprovoznění (zdroj: vlastní).



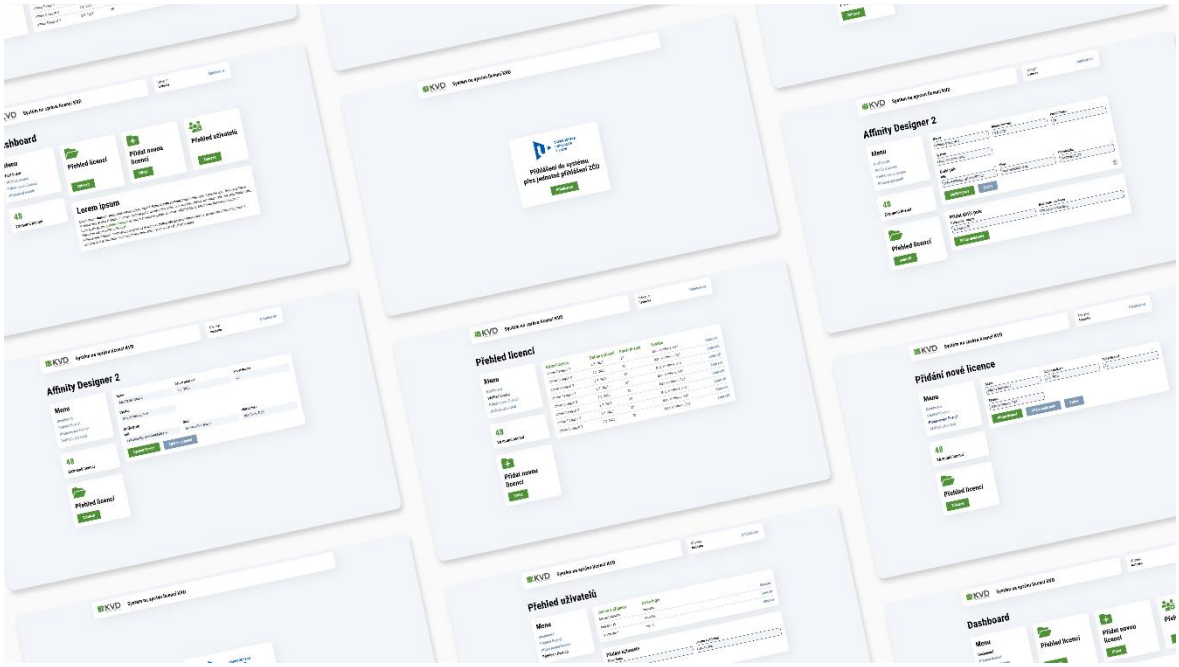
Příloha 1: Wireframe přihlašování do systému (zdroj: vlastní)



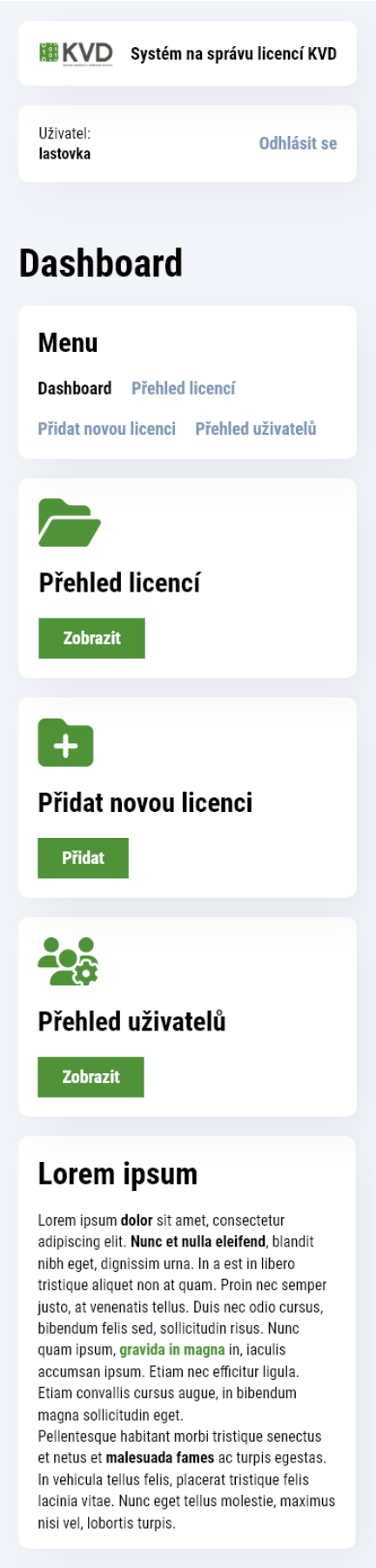
Příloha 2: Wireframe dashboardu systému (zdroj: vlastní)



Příloha 3: Wireframe dashboardu systému na mobilní zařízení (zdroj: vlastní)



Příloha 4: Navržené stránky v grafickém návrhu systému (zdroj: vlastní)



Příloha 5: Navržená stránka dashboardu pro mobilní zařízení (zdroj: vlastní)