FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Administrační rozhraní systému pro automatizované hodnocení úloh z programování
Student:	Jiří Zikán
Vedoucí:	Mgr. Jakub Růžička
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Analyzujte, navrhněte a implementujte administrační rozhraní pro systém automatizovaného hodnocení domácích úloh z programování SharpTest, jehož funkční prototyp byl vytvořen v rámci projektů BI-SP1/2. Rozhraní musí umožnit vyučujícím efektivní správu studentů, předmětů a úloh. Dále musí poskytnout přehled o dosažených výsledcích jednotlivých studentů v rámci předmětů i úloh a možnost jejich manuálního ohodnocení. Klíčová je taktéž integrace a plná kompatibilita s již existujícím systémem.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 17. prosince 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Administrační rozhraní systému pro automatizované hodnocení úloh z programování

Jiří Zikán

Katedra softwarového inženýrství
Vedoucí práce: Mgr. Jakub Růžička

30. července 2020

Poděkování

Mé poděkování patří Mgr. Jakubu Růžičkovi za odborné vedení, ochotu a trpělivost, kterou mi věnoval při zpracování mé bakalářské práce. Poděkovat bych chtěl také Janu Kuběnovi za odborné poradenství v oblasti počítačové bezpečnosti. Nakonec bych rád poděkoval Bc. Kristýně Panešové za podporu a své rodině za zázemí, které mi poskytla.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 30. července 2020 . . . . . .

Odkaz na tuto práci

Zikán, Jiří. *Administrační rozhraní systému pro automatizované hodnocení úloh z programování*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Bakalářská práce se zabývá využitím metod softwarového inženýrství při vývoji aplikace administračního rozhraní systému SharpTest zaměřujícího se na automatizované hodnocení úloh z programování. Hlavním cílem práce je analyzovat, navrhnout a realizovat aplikaci administračního rozhraní, která se stane nástrojem pro efektivní správu zmíněného systému. Rešeršní část práce vymezuje základní pojmy týkající se třívrstvých webových aplikací a blíže popisuje dále použité technologie a postupy. Navazující praktické části práce odrážejí životní cyklus vývoje softwarového produktu. Provedena je analýza východisek práce zahrnující analýzu požadavků, procesů i problémové domény. Analyzována je taktéž struktura stávajícího systému SharpTest a všechny vlastnosti, které jsou klíčové pro provedení integrace administračního rozhraní. Při návrhu aplikace je využito třívrstvé architektury. Datová vrstva je integrována se stávajícím systémem a návrh se dále zabývá volbou a odůvodněním technologií pro zbývající vrstvy. Navržena je i struktura aplikačního a uživatelského rozhraní. Během realizace jsou na implementační úrovni popsány jednotlivé komponenty, ze kterých výsledná aplikace sestává. Serverová část aplikace je implementována ve frameworku Express.js běžícím na platformě Node.js. Klientská část aplikace je založena na frameworku Vue.js. Pro zajištění responzivity uživatelského rozhraní je využito knihovny Bootstrap s jejím rozšířením BootstrapVue. Na závěr je popsán proces testování aplikace včetně jejího zhodnocení skutečným uživatelem. Výsledkem bakalářské práce je produkčně využitelný software splňující všechny požadavky zadavatele.

Klíčová slova administrační rozhraní, vzdělávací technologie, webová aplikace, jednostránková aplikace, aplikační rozhraní REST, uživatelské rozhraní, Node.js, Express.js, Vue.js, Bootstrap

Abstract

The bachelor thesis deals with the development of an administration interface application and its integration into SharpTest – a system for automated assessments of programming tasks. The main goal of this thesis is to analyze, design and implement application that will become an effective tool for administration of mentioned system. Literature review chapter of this thesis aims to establish basic concepts related to three-tier architecture of web applications and to introduce technologies and procedures later used in implementation phase. Following chapters reflect phases of standard Software Development Life Cycle (SDLC). Performed analysis consists of requirements, process and domain analysis. The structure of SharpTest system and all necessary details required for successful integration are also described. In design phase the three-tier architecture is chosen for this application. Data tier is integrated with existing system and the design further deals with the choosing of right technologies for other tiers. Application programming interface and user interface are also designed in this chapter. Implementation details of individual components of the application are explained in the next chapter. Express.js framework on Node.js platform is used for the server-side part of the application and Vue.js framework is used for the client-side. Responsive user interface is made with Bootstrap and BootstrapVue libraries. Last chapter focuses on the testing phase of SDLC including end-user feedback. Created application is production-grade software satisfying all requirements set by the client.

Keywords administration interface, educational technology, web application, single page application, REST application interface, user interface, Node.js, Express.js, Vue.js, Bootstrap

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vývoj softwaru je jednou z disciplín spadajících pod rozsáhlý obor informačních technologií, která nalézá širokou škálu uplatnění. I kvůli tomu je však v poslední době kladen stále větší důraz na její efektivitu. Softwarová řešení se stávají rozsáhlejšími a sofistikovanějšími díly, a tedy i nástroje a metody softwarového inženýrství musejí držet krok s narůstajícími požadavky uživatelů směřujícími k vysoké stabilitě, bezpečnosti, nízké chybovosti a intuitivnímu uživatelskému rozhraní.

Mezi oblasti, ve kterých informační technologie i vývoj softwaru nalézají uplatnění, patří i jejich samotná výuka. Ve své bakalářské práci budu navazovat na stávající softwarové řešení SharpTest, které umožňuje automatizované hodnocení úloh z programování. Tento systém slouží pro podporu výuky programování na středních školách a jeho vývoj probíhá ve spolupráci s pražskou střední školou zaměřenou na výuku informačních technologií. Jeho snahou je usnadnit práci pedagogickým pracovníkům a poskytnout rychlou zpětnou vazbu studentům. V současné chvíli však systém neobsahuje administrační rozhraní nezbytné pro jeho rozsáhlejší produkční použití.

Cílem mé bakalářské práce tedy bude využít metod softwarového inženýrství a analyzovat, navrhnout a realizovat aplikaci administračního rozhraní. Ta umožní správu celého systému SharpTest, od správy údajů o studentech přes přidávání a úpravu jednotlivých předmětů a úloh až po možnost efektivního hodnocení odevzdaných řešení a zobrazení přehledových statistik. Administrační rozhraní bude integrováno se stávajícím systémem. Nejedná se však o pouhé uživatelské rozhraní, nýbrž o samostatnou vícevrstvou aplikaci, která bude závislá pouze na nejnižší datové vrstvě.

Z hlediska struktury bude práce sestávat z pěti hlavních kapitol, které odrážejí životní cyklus vývoje softwarového produktu. První z těchto kapitol představuje literární rešerši vymezující základní pojmy a shrnující užité technologie. Druhá kapitola obsahuje analýzu všech východisek práce. Analyzovány budou požadavky na administrační rozhraní, problémová doména

i vlastnosti stávajícího systému, které jsou klíčové pro provedení zamýšlené integrace. Třetí kapitola se následně zaměří na architekturu a návrh hlavních součástí aplikace, jako je například aplikační či uživatelské rozhraní. Předposlední kapitola popíše samotnou realizaci serverové i klientské části aplikace včetně podstatných implementačních detailů. Poslední kapitola se bude na závěr zabývat testováním funkčnosti realizované aplikace.

Kapitola 1

Rešerše

Literární rešerše zpracovaná v této kapitole má za cíl objasnit vybrané odborné pojmy pojící se s webovými aplikacemi a jednotlivými vrstvami jejich třívrstvé architektury. Dále obsahuje také výčet a bližší popis vybraných technologií, které jsou použity v navazujících částech práce. Citováno je z tuzemské i zahraniční odborné literatury a oficiálních dokumentací využitých aplikačních rámců, knihoven nebo souvisejících technologií.

1.1 Webová aplikace

Webová aplikace je takovým druhem aplikace, která je poskytována z webového serveru prostřednictvím počítačové sítě. K jejímu spuštění pak není nezbytná instalace na zařízení uživatele, neboť se o její zobrazení stará webový prohlížeč [1]. Na rozdíl od běžné webové stránky se webová aplikace liší tím, že implementuje zvolenou aplikační logiku [2, s. 22]. Aplikační logika se může nacházet jak na straně serveru, tak na straně klienta, respektive webového prohlížeče [2, s. 31]. I na základě tohoto kritéria dělíme webové aplikace na následující dva typy.

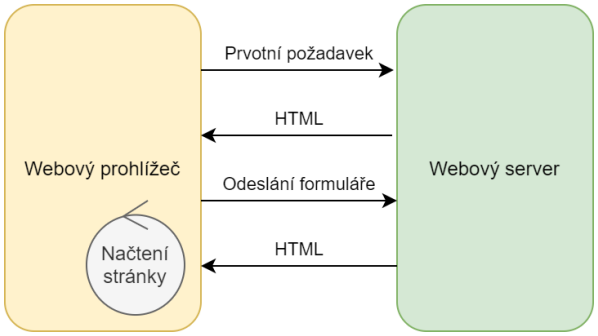
1.1.1 Vícestránková aplikace

Vícestránkové aplikace, označované zkratkou MPA, se vyznačují tím, že se většina aplikační logiky nachází na straně serveru [3, s. 9]. Server sestaví výslednou webovou stránku ve formátu HTML a následně ji odešle spolu s ostatními statickými dokumenty pomocí protokolu HTTP webovému prohlížeči. Ten již webovou stránku pouze zobrazí.

Dynamického chování vícestránkové aplikace je dosaženo reakcí serveru na vstup uživatele, kterým může být například odeslání vyplněného formuláře nebo přechod na jinou URL adresu. Server v návaznosti na takovou reakci vygeneruje zcela novou webovou stránku s aktualizovaným obsahem, kterou následně znovu zašle webovému prohlížeči k jejímu zobrazení [4, s. 5].

Použití vícestránkových aplikací je vhodné zvláště v případě jednodušších aplikací, jejichž hlavním účelem je prezentování dat. Nespornou výhodou je jejich schopnost fungovat i bez podpory skriptovacího jazyka v prohlížeči. Díky tomu jsou také mnohem snadněji optimalizovatelné pro indexování v internetových vyhledávačích [5].

Obrázek 1.1: Vícestránková aplikace MPA [6]



1.1.2 Jednostránková aplikace

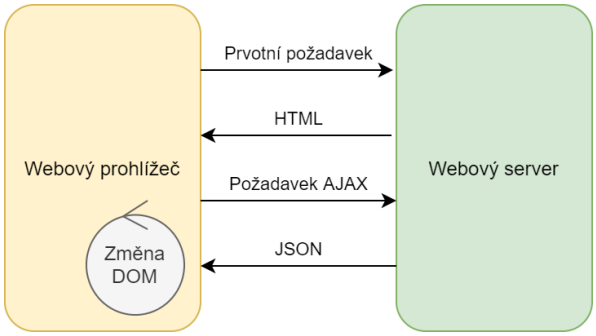
Jednostránkové aplikace, označované zkratkou SPA, se naproti tomu vyznačují přenesením části aplikační logiky na stranu klienta [3, s. 9]. Jedná se především o tu část aplikační logiky, která souvisí s uživatelským rozhraním, prezentací dat a interakcí s uživatelem [4, s. 6].

Celá klientská část jednostránkové webové aplikace je prohlížečem stažena v rámci načtení jediné webové stránky. Dynamické chování je pak zajištěno pomocí skriptovacího jazyka běžícího ve webovém prohlížeči, který na základě určité události aktualizuje strukturu současné webové stránky označovanou jako DOM model [4, s. 7]. Takovým skriptovacím jazykem může být například interpretovaný jazyk JavaScript, který v dnešní době podporuje naprostá většina webových prohlížečů [7, s. 1].

Pokud klientská část jednostránkové webové aplikace potřebuje pracovat s externími daty, komunikuje s aplikačním rozhraním serverové části aplikace například prostřednictvím technologie AJAX. Díky tomu jsou načtena pouze data, která aplikace právě potřebuje, a nemusí se opakovaně posílat a načítat celá struktura webové stránky [6].

Výsledkem takového přístupu je znatelně rychlejší odezva jednostránkových aplikací na uživatelské požadavky, nižší zatížení serveru a menší nároky na přenosovou kapacitu počítačové sítě [3, s. 10]. Použití jednostránkových aplikací je vhodné v případě, kdy aplikace neslouží primárně pro prezentování dat, nýbrž musí implementovat širokou škálu funkcionalit a současně poskytnout bohaté a přívětivé uživatelské rozhraní. Výhodou je i skutečnost, že serverová část aplikace poskytuje univerzální rozhraní, na které může být v budoucnu integrována i zcela jiná aplikace [5].

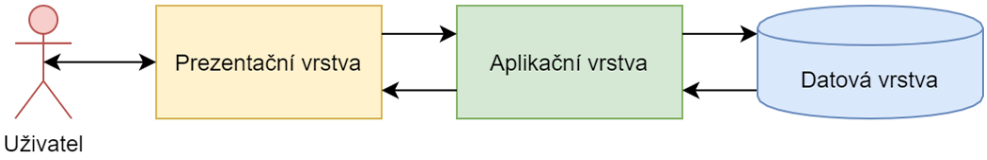
Obrázek 1.2: Jednostránková aplikace SPA [6]



1.2 Architektura aplikací

Na konceptuální úrovni lze v architektuře většiny aplikací rozlišit několik základních vrstev [8, s. 26]. V případě menších aplikací je často využíváno monolitické architektury, ve které nemusí být oddělení těchto vrstev příliš patrné [9, s. 261]. U těch větších však z důvodu lepší škálovatelnosti, spolehlivosti a údržby převládá architektura, ve které jsou jednotlivé vrstvy tvořeny oddělenými komponentami s jednoznačně definovaným rozhraním [8, s. 27]. Komponenty mohou být dokonce představovány samostatnými programy běžícími na fyzicky oddělené výpočetní infrastruktuře. Mezi tyto architektury patří i architektura třívrstvá, která vymezuje prezentační, aplikační a datovou vrstvu [10, s. 273]. Vlastnosti jmenovaných vrstev a některé s nimi související pojmy jsou v této kapitole dále diskutovány.

Obrázek 1.3: Třívrstvá architektura [11, s. 16]



1.2.1 Datová vrstva

Datová vrstva se stará o perzistentní uložení a zpřístupnění dat, se kterými celá aplikace pracuje. Nejčastěji je zajištěna pomocí vybraného typu databáze, může však být představována také souborovým systémem, webovou službou, případně i jinou aplikací [11, s. 5]. Na datovou vrstvu jsou obvykle kladeny požadavky související se zajištěním konzistence dat, zajištěním jejich předzpracování a agregace, auditování, souběžného přístupu a dalších požadovaných vlastností [12].

1.2.1.1 Relační databáze

Databáze je pojem označující soubor strukturovaných informací spolu s nástroji pro jejich spolehlivé ukládání a efektivní správu [13, s. 45]. Relační databáze jsou typem databází založených na relačním modelu, ve kterém jsou všechna data uložena ve dvourozměrných strukturách nazývaných relace. Relaci lze reprezentovat jako tabulku, kde sloupce představují atributy relace, řádky představují unikátně identifikované záznamy a konkrétní hodnoty se pak nacházejí v průniku daného řádku a sloupce [13, s. 46]. Vzájemné provázání dat napříč relacemi je zajištěno pomocí relačních vztahů [13, s. 47].

Relační databáze je řízena s využitím systému řízení báze dat označovaného zkratkou RDBMS. Ten se stará o spolehlivé a perzistentní uložení dat dle definovaného schématu a zajištění jejich integrity během prováděných operací. Kromě toho většinou implementuje i mechanismy umožňující pokročilou správu, zálohování, replikaci a mnohé další činnosti [14, s. 10]. Mezi běžně používané RDBMS patří například MySQL, SQL Server nebo Oracle.

Nejčastěji je pro dotazování nad daty uloženými v relační databázi využíván doménově specifický jazyk SQL [14, s. 33]. Jeho příkazy jsou rozděleny do několika kategorií, pomocí nichž je možné spravovat databázové objekty, vybírat, vkládat či modifikovat data nebo řídit přístup uživatelů [14, s. 40].

1.2.1.2 NoSQL databáze

NoSQL databáze je souhrnným označením pro databázové technologie, které nevyužívají relační model [15]. Své využití nalézají především v oblasti big data, kde by již běžné relační databáze nebyly schopny efektivně zvládnout daný objem dat. V poslední době se však pro svou jednoduchost a dobrou škálovatelnost začínají NoSQL databáze uplatňovat také na poli webových aplikací [16]. Dobrým příkladem je dokumentová databáze MongoDB.

1.2.2 Aplikační vrstva

Aplikační vrstva realizuje zamýšlenou logiku aplikace vyplývající přímo z problémové domény [17, s. 49]. Na základě požadavků provádí zpracování a transformace vstupních i výstupních dat [10, s. 273]. Tvoří tak prostředníka mezi datovou a prezentační vrstvou.

1.2.2.1 Aplikační server

Aplikační vrstva může být ve třívrstvé architektuře představována takzvaným aplikačním serverem. Jedná se o specializovaný program umožňující provozování sdílené aplikace. V případě webové aplikace se můžeme bavit o webovém serveru, respektive webovém aplikačním serveru [18]. Kromě samotné aplikace může aplikační server poskytovat také další služby, které jsou nezbytné k zajištění jejího provozu [19, s. 25].

1.2.2.2 Webová služba

Příkladem takových služeb mohou být webové služby. Ty umožňují vzájemnou interakci většího počtu instancí klientské části aplikace s rozhraním aplikačního serveru prostřednictvím počítačové sítě. Implementují aplikační logiku, zpracovávají transakce klientské aplikace a zprostředkovávají jí zpracovaná data [20, s. 5]. Zajišťovat mohou také autentizaci či vynucení uživatelských oprávnění. Mezi nejpoužívanější webové služby současnosti patří SOAP a REST. Zatímco SOAP je protokol zaměřený spíše na zprostředkování vzdáleného přístupu k dané službě, REST je obecný architektonický vzor pro aplikační rozhraní orientované na datové zdroje [21].

1.2.3 Prezentační vrstva

Prezentační vrstva zprostředkovává aplikací zpracovaná data v požadovaném formátu externím entitám. Dále umožňuje těmto externím entitám zpětně interagovat s běžící aplikací a poskytnout jí tak potřebná vstupní data [17, s. 49]. Externími entitami jsou nejčastěji uživatelé, kteří s aplikací interagují přes uživatelské rozhraní v podobě příkazové řádky, grafického uživatelského rozhraní nebo webové stránky. Data však mohou být zprostředkována i jiným aplikacím či systémům [11, s. 4].

1.2.3.1 Uživatelské rozhraní

Uživatelské rozhraní je součást aplikace, kterou mohou uživatelé vnímat svými smysly. Každé uživatelské rozhraní se snaží naplnit dva základní cíle, díky nimž je umožněna obousměrná komunikace uživatele s aplikací. Prvním z nich je umožnit uživateli ovládat aplikaci prostřednictvím zadaných vstupů. Druhým cílem je prezentovat výstupní data uživateli v takové podobě, která je pro něj vnímatelná a pochopitelná [22, s. 4].

Návrhu uživatelského rozhraní se věnuje takzvaný UX/UI design [23, s. 7]. Zatímco UI design pracuje spíše se vzhledem uživatelského rozhraní a rozložením jeho komponent, UX design je širší disciplína zabývající se tím, jak na uživatele celkově působí interakce s aplikací a jejím uživatelským rozhraním. [23, s. 8]. Jejich společnou snahou je navrhnout přívětivé uživatelské rozhraní, které bude snadno pochopitelné i použitelné a uživatel se v něm bude schopen v krátkém čase samostatně zorientovat.

Navrhnout dobré uživatelské rozhraní, které bude mít všechna výše uvedená pozitiva, je však velmi náročný úkol [24, s. ix]. Existují sice určité obecné principy, které mohou UX/UI vývojáři využít [23, s. 160], nicméně nelze opomenout skutečnost, že každé uživatelské rozhraní je individuální, stejně jako je individuální cílová skupina uživatelů. Proto je pro návrh dobrého uživatelského rozhraní klíčové, aby bylo konzultováno s uživateli již během raných fází vývoje [23, s. 11].

1.2.3.2 Wireframing a prototypování

Wireframing a prototypování uživatelského rozhraní jsou metody, díky kterým mají uživatelé možnost seznámit se s funkcionalitou i vzhledem uživatelského rozhraní ještě před jeho realizací [25].

Účelem první z těchto metod je vytvořit základní model nazývaný wireframe, který znázorňuje obsah a strukturu uživatelského rozhraní bez ohledu na budoucí grafický design [26, s. 171]. Pouze pomocí čar a textu definuje rozložení jednotlivých funkčních prvků a umožňuje tak uživatelům i vývojářům získat základní povědomí o tom, co bude uživatelské rozhraní umožňovat a jakým způsobem se s ním bude pracovat [27].

Velkou výhodou těchto modelů je jejich jednoduchost. Mohou tak být snadno modifikovány podle potřeb uživatele, a jsou tedy vhodné pro použití v raných fázích vývoje [26, s. 172]. Modely se také stávají součástí dokumentace a smluvních vztahů, protože jasně definují vytyčený cíl [25].

Prototypování naproti tomu co nejpřesněji modeluje skutečný vzhled výsledného uživatelského rozhraní. Kromě rozložení prvků se zabývá i jejich vzhledem a většinou umožňuje uživateli reálně si vyzkoušet interakci s aplikací. Díky prototypům lze získat velmi přesnou zpětnou vazbu, nicméně jejich tvorba a případná modifikace je již mnohonásobně pracnější [27].

1.2.3.3 Responzivní web design

Responzivní web design je způsob návrhu a tvorby webových uživatelských rozhraní, respektive webových stránek tak, aby bylo jejich zobrazení optimalizováno pro širokou škálu zařízení. S tímto konceptem přišel v roce 2010 americký programátor Ethan Marcotte [28, s. 1]. Myšlenka se poměrně rychle uchytila, jelikož v té době začal znatelně vzrůstat počet uživatelů, kteří pro zobrazování webových stránek využívali tablety nebo mobilní telefony [29].

Responzivní webová stránka je schopna přizpůsobit rozložení svého obsahu aktuálním rozměrům obrazovky daného zařízení. Takového chování je dosaženo správným využitím konstruktů jazyka CSS, který popisuje vlastnosti jednotlivých grafických prvků. Struktura webu musí být flexibilní a rozměry prvků relativní k rozměrům obrazovky. Responzivní design dále využívá CSS Media Queries, které umožňují nastavit různé vlastnosti grafických prvků pro různý rozsah velikostí obrazovek [28, s. 2].

1.3 Použité technologie

Poslední část literární rešerše má za cíl vyjmenovat a blíže popsat vybrané technologie, platformy, aplikační rámce a knihovny, které jsou přímo použité v navazujících praktických částech této práce. Odůvodnění jejich výběru se pak nachází ve třetí kapitole zabývající se návrhem aplikace administračního rozhraní a jejích součástí.

1.3.1 MySQL

MySQL je Open Source relační systém řízení báze dat. Velké oblibě se těší zvláště mezi vývojáři webových aplikací. Jeho hlavními výhodami jsou rychlost, spolehlivost a schopnost běžet na mnoha různých platformách. K jeho popularitě zajisté přispívá i skutečnost, že je v základu dostupný pod bezplatnou licencí GNU GPL [30].

V roce 2010 se při odkoupení společnosti Sun Microsystems korporací Oracle oddělila nástupnická větev MySQL označená jako MariaDB. Důvodem byla právě snaha vývojářů o zachování otevřeného kódu a bezplatné licence [31]. MariaDB je z velké části zpětně kompatibilní s MySQL [32].

Systém je primárně založen na komunikačním modelu klient/server. Roli serveru zde hraje MySQL Server, ke kterému se může prostřednictvím počítačové sítě připojit velká škála různých typů klientů [33, s. 1]. Od administračních a diagnostických nástrojů, až po aplikace využívající MySQL jako svou datovou vrstvu. Rozhraní umožňující komunikaci s MySQL je dostupné ve většině moderních programovacích jazycích [34, s. 66].

Jak již název napovídá, MySQL také podporuje jazyk SQL jako hlavní komunikační a dotazovací prostředek. Mimo základní konstrukty jazyka SQL specifikované normou ANSI/ISO obsahuje MySQL také pokročilé prvky v podobě uložených procedur, pohledů či spouští [33, s. xii].

1.3.2 REST API

Zkratka REST označuje architekturu aplikačního rozhraní navrženou pro distribuované prostředí. Webové služby poskytující toto rozhraní pak nesou přídomek RESTful [35, s. 4]. Aplikační rozhraní REST je datově orientované a představuje jednotný způsob přístupu k datovým zdrojům [36, s. 11].

Každý datový zdroj je v architektuře REST unikátně identifikován pomocí URI [35, s. 11]. Datový zdroj může poskytovat a přijímat data v nejrůznějších formátech, jako například XML nebo JSON [36, s. 12]. Stejně tak může být při komunikaci využito různých protokolů. V případě protokolu HTTP probíhá komunikace s využitím metod GET, POST, PUT a DELETE, které odpovídají takzvaným CRUD operacím pro manipulaci s datovým zdrojem.

Metoda GET slouží k získání prvku či kolekce prvků z daného datového zdroje. Metoda PUT umožňuje modifikaci již existujícího prvku. Pomocí metody POST lze do kolekce přidat nový prvek a naopak s využitím metody DELETE, jak lze dovodit z jejího názvu, je možné prvek z kolekce opět odebrat [36, s. 15].

Další vlastností REST API je bezstavová komunikace. Každý požadavek na rozhraní musí obsahovat veškeré informace nutné k jeho vykonání. I díky tomuto principu mohou být explicitně označené požadavky ukládány do mezipaměti, což má za následek nižší zatížení sítě i příslušného serveru [35, s. 4].

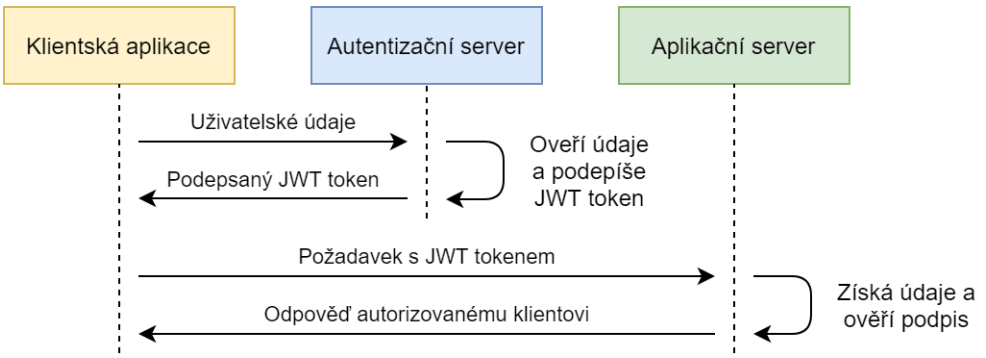
1.3.3 JSON Web Token

JSON Web Token, zkráceně JWT, je metoda pro bezpečnou výměnu informací mezi dvěma stranami. Je popsána standardem RFC 7519 [37, s. 1]. I přestože metoda umožňuje šifrování přenášených informací, jejím primárním cílem je ověření jejich pravosti pomocí digitálního podpisu. Ten může být realizován symetrickými i asymetrickými šifrovacími algoritmy [38].

JWT je zakódovaný JSON objekt, který se obvykle skládá z hlavičky, dat a podpisu. V hlavičce se nachází typ použité metody a název šifrovacího algoritmu. Následují vlastní data skládající se z výčtu vlastností a jejich hodnot [37, s. 6]. Některé názvy vlastností jsou takzvaně registrovány a mají speciální význam. Mohou například identifikovat vydavatele nebo určovat dobu platnosti tokenu [37, s. 9]. Poslední část je tvořena podpisem, což je šifrovaný hash vytvořený z hlavičky a dat [39, s. 7].

Metoda JWT je v praxi běžně používána pro autorizaci požadavků zaslaných na aplikační rozhraní [38]. Uživatel nejdříve odešle své přihlašovací údaje autentizačnímu serveru, který ověří jeho identitu a vystaví uživateli pomocí metody JWT digitálně podepsaný token. Vystavený token kromě podpisu obsahuje také identifikaci příslušného uživatele. Uživatel při zaslání požadavku na aplikační rozhraní může následně prokázat svou identitu prostým přiložením vystaveného tokenu [40].

Obrázek 1.4: Autentizace a autorizace pomocí JWT [40]



1.3.4 Node.js

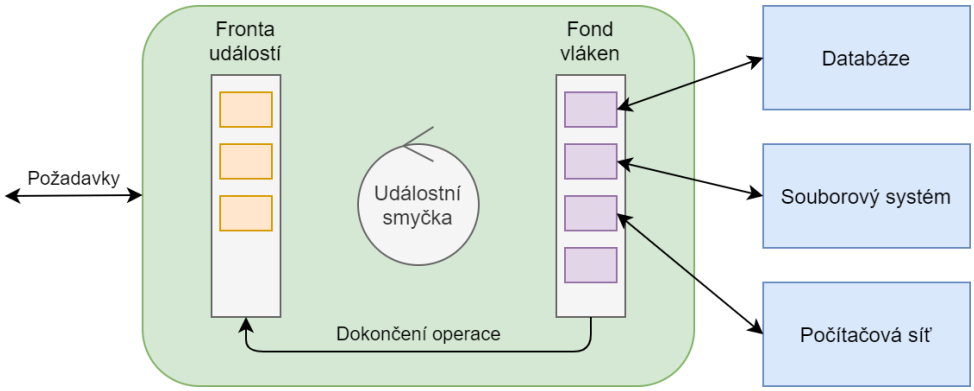
Node.js je platforma umožňující psaní modulárních a škálovatelných aplikací s využitím interpretovaného jazyka JavaScript. Přestože ji lze využít k tvorbě téměř libovolného druhu aplikací, navržena byla především pro programování síťových aplikací a aplikačních serverů [41, s. 7].

Její běhové prostředí je založeno na JS interpretu V8, který byl původně vyvinut společností Google pro webový prohlížeč Google Chrome [42]. Zatímco v případě webového prohlížeče je JS aplikace součástí načtené webové stránky, platforma Node.js umožňuje tvorbu JS aplikací běžících zcela samostatně [43].

Mimo samotný interpret je v ní obsažena také široká škála knihoven pro práci se soubory, databázemi a nejrůznějšími síťovými protokoly [41, s.8].

Platforma Node.js se vyznačuje jednovláknovou architekturou řízenou událostmi. Namísto poměrně neefektivního a komplikovaného způsobu vytváření nových vláken pro každý požadavek jsou v této architektuře všechny požadavky obslouženy jediným procesem [44]. Node.js dále využívá asynchronní neblokující vstupně výstupní operace, při kterých nedochází k zablokování procesu po dobu zpracování operace systémem. Díky tomu je možné i jedním vláknem obsloužit velké množství souběžně příchozích požadavků [45].

Obrázek 1.5: Architektura platformy Node.js [46]



1.3.5 Express.js

Express.js je minimalistický aplikační rámec postavený na platformě Node.js. Poskytuje sadu nástrojů pro tvorbu statických i dynamických webových aplikací a aplikačních rozhraní založených na protokolu HTTP [47]. Na rozdíl od základních modulů platformy Node.js řeší tento aplikační rámec také syntaktickou analýzu HTTP požadavků a cookies, směrování požadavků podle URL a HTTP metody, správu relace a mnohé další problémy, které by jinak musely být řešeny samostatně [48, s. 2].

1.3.6 Vue.js

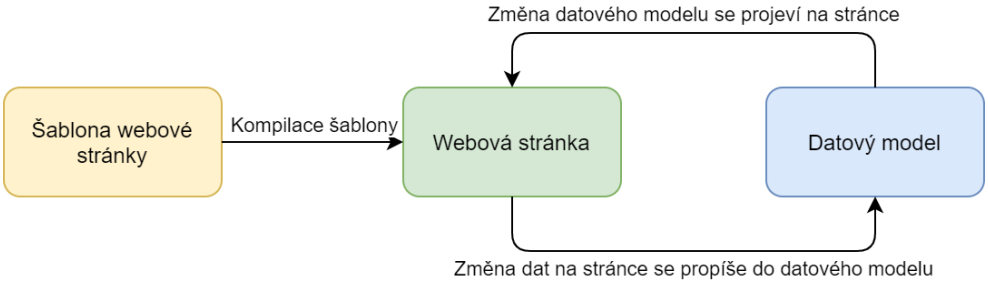
Vue.js je aplikační rámec pro vytváření webových uživatelských rozhraní pomocí jazyka JavaScript. I přesto, že je jeho minimalistické jádro zaměřeno čistě na vykreslování obsahu a chování grafických prvků na webové stránce, s využitím rozšiřujících knihoven, které jsou v aplikačním rámci obsaženy, lze vytvářet i rozsáhlé jednostránkové aplikace [49].

Aplikační rámec Vue.js umožňuje dělit obsah webové stránky do několika znovupoužitelných komponent. Každá komponenta obsahuje naprogramovanou logiku a data, která se pak za pomoci textové interpolace a speciálních

direktiv umístěných v HTML šabloně mapují na DOM model a jednotlivé prvky uživatelského rozhraní [50, s. 85].

Výhodou Vue.js je, že pro mapování využívá dvoucestné synchronizace dat. To znamená, že pokud jsou změněna data obsažená v komponentě, dojde automaticky k překreslení webové stránky. Stejně tak se tato změna zpětně propíše do datového modelu komponenty v případě, kdy uživatel zadá jiná vstupní data pomocí prvků uživatelského rozhraní. [50, s. 10].

Obrázek 1.6: Dvoucestná synchronizace dat Vue.js [51]



Dalšími důležitými součástmi Vue.js jsou Vuex a Vue-router. Vuex je návrhový vzor a současně knihovna řešící správu sdíleného stavu mezi větším počtem komponent [52]. Takovou problematiku pomáhá řešit zvláště u rozsáhlejších aplikací. Vue-router zajišťuje provázání mezi komponentami a URL. Je tak nezbytnou součástí umožňující například navigaci nebo přesměrování u plnohodnotných jednostránkových aplikací [53].

1.3.7 Bootstrap

Bootstrap je knihovna napsaná v jazyce CSS zaměřená na tvorbu responzivních webových aplikací [54]. Po svém nasazení poskytuje tato knihovna hned několik důležitých funkcionalit. V první řadě dojde k aplikování základních stylů na naprostou většinu HTML elementů. Tam, kde vlastní grafický design nehraje primární roli, je tak možné snadno vytvořit webovou stránku nebo aplikaci bez nutnosti psaní velkého množství kaskádových stylů upravujících její celkový vzhled [55].

Druhou zásadní funkcionalitou knihovny Bootstrap je její takzvaný layout a grid systém. Ten umožňuje umístit obsah webové stránky do flexibilní mřížky, která se dokáže svým rozložením plynule přizpůsobit libovolné velikosti obrazovky daného zařízení. Díky tomu je značně usnadněno nasazení responzivního web designu [56].

Poslední významnou funkcionalitou knihovny Bootstrap je předem připravená sada komponent a prvků uživatelského rozhraní [57, s. 67]. Jedná se především o tlačítka, tabulky, formuláře, navigace, dialogy a mnohé další komponenty, které mohou být snadno využity a modifikovány dle vlastních potřeb daného projektu [57, s. 67-69].

1.3.8 BootstrapVue

Jak z názvu knihovny BootstrapVue vyplývá, jedná se o počin, který kombinuje funkcionalitu i výhody knihovny Bootstrap spolu s aplikačním rámcem Vue.js [58]. Knihovna využívá výše jmenované sady komponent a prvků uživatelského rozhraní z knihovny Bootstrap a rozšiřuje ji o funkcionalitu naprogramovanou ve Vue.js. Tyto interaktivní komponenty lze následně přidat přímo do HTML šablony a jejich chování nastavit či modifikovat pomocí příslušných direktiv [57, s. 74].

1.3.9 Font Awesome

Font Awesome je knihovna poskytující sadu fontů a uživatelsky přívětivých ikonek [59]. Těmi lze doplnit například prvky uživatelského rozhraní tak, aby bylo jeho ovládání intuitivní a snadno srozumitelné.

1.3.10 Jest

Jest je aplikační rámec určený pro automatizované testování aplikací napsaných v jazyce JavaScript. Umožňuje paralelní provádění jednotkových testů, stejně jako snadný mocking funkcí, objektů nebo celých modulů. Jeho další důležitou schopností je výpočet pokrytí kódu testy včetně generování souboru s podrobnými statistikami [60].

Kapitola 2

Analýza

Analýza představuje první fázi životního cyklu softwarového produktu, jejímž účelem je získat, rozebrat a následně dále diskutovat veškerá východiska nově vznikající aplikace. Mezi tato východiska patří především požadavky zadavatele, podnikové procesy, entity a vztahy vyskytující se v problémové doméně a informace o stávajících řešeních a použitých technologiích. Výsledky analýzy budou sloužit jako podklad pro učinění správných kroků během následujících fází životního cyklu. Současně se stanou nedílnou součástí strukturované dokumentace budoucí aplikace.

2.1 Analýza požadavků

Prvotními východisky, která jsou z části uvedená již v samotném zadání této práce, jsou požadavky zadavatele na funkcionalitu a další vlastnosti vyvíjené aplikace. Požadavky popisují, jakou činnost má aplikace vykonávat, jakým způsobem ji má vykonávat a jaká omezení jsou při tom na ni kladena. Dle jednoho ze základních dělení je můžeme rozdělit na požadavky funkční a požadavky obecné.

2.1.1 Funkční požadavky

Funkční požadavky vyjadřují konkrétní schopnosti, kterými by měla aplikace disponovat. U jednotlivých požadavků je uvedena jednak zkratka a název pro jejich snadnější identifikaci, dále pak následuje jejich podrobnější popis. Tento popis uvádí, jakou činnost může uživatel prostřednictvím aplikace provést a za jakých podmínek tak může učinit.

- **F01 – Přihlášení** – nepřihlášený uživatel se může přihlásit pomocí přihlašovacích údajů a získat tak svou uživatelskou roli.
- **F02 – Správa lektorů a studentů** – administrátor může spravovat účty lektorů a studentů včetně jejich registrace do systému.

- **F03 – Správa předmětů** – administrátor může spravovat předměty a přiřazovat k nim jednotlivé lektory.
- **F04 – Přiřazení studentů** – lektor může přiřazovat studenty ke svým předmětům.
- **F05 – Správa úloh a kritérií hodnocení** – lektor může spravovat úlohy ve svých předmětech včetně kritérií jejich hodnocení.
- **F06 – Zobrazení řešení** – lektor si může v rámci svých předmětů zobrazit studenty odevzdaná řešení.
- **F07 – Ruční hodnocení odevzdaných řešení** – lektor může ve svých předmětech ručně ohodnotit nebo změnit hodnocení studenty odevzdaných řešení.
- **F08 – Zobrazení výsledků** – lektor si může zobrazit souhrnné výsledky studentů v rámci jimi odevzdaných řešení, úloh i celých předmětů.
- **F09 – Oprávnění administrátora** – administrátor může provádět všechny úkony lektora ve všech předmětech.

2.1.2 Obecné požadavky

Obecné požadavky, někdy nazývané také nefunkční nebo mimofunkční, představují popis omezení kladených na vlastnosti celého systému. Tato omezení se týkají především použitelnosti, rozšiřitelnosti, bezpečnosti, využitých technologií a dalších aspektů, kterými bude ovlivněna architektura výsledné aplikace. Stejně jako u požadavků funkčních je zde uvedena zkratka, název a podrobnější popis.

- **N01 – Webová aplikace** – aplikace je uživateli poskytována skrze počítačovou síť, klientem je webový prohlížeč. Důvodem požadavku je absence nutnosti instalace webové aplikace.
- **N02 – Single-page aplikace** – serverová část aplikace slouží pouze jako datový zdroj, aplikace je vykreslována na straně klienta. Důvodem je lepší odezva na požadavky uživatele a nižší zatížení sítě.
- **N03 – Responzivní design** – uživatelské rozhraní se samo přizpůsobí rozměrům obrazovky. Důvodem je použitelnost na všech typech zařízení.
- **N04 – Aplikační rozhraní REST** – aplikace bude získávat data přes aplikační rozhraní typu REST s využitím protokolu HTTP a datového formátu JSON. Důvodem požadavku je možnost využití REST API pro další budoucí aplikace a moduly systému SharpTest.
- **N05 – Integrace se systémem SharpTest** – aplikace musí být integrována se systémem SharpTest na úrovni datové vrstvy.

2.1.3 Uživatelské role

Uživatelská role představuje soubor činností, které je uživatel v takové roli oprávněn provést. Jednotlivé uživatelské role částečně vyplývají přímo z funkčních požadavků, přesto je však vhodné věnovat jim o něco větší pozornost a v průběhu analýzy požadavků tyto role jednoznačně vymezit.

- **Nepřihlášený uživatel** – jakýkoliv uživatel aplikace, který se zatím nepřihlásil a nebyla tak ověřena jeho identita. Kromě přihlášení nemůže nepřihlášený uživatel provést v aplikaci žádnou jinou akci.
- **Student** – přihlášený uživatel aplikace představující studenta. V rámci aplikace administračního rozhraní je studentovi zobrazena pouze informace o skutečnosti, že nemá dostatečné oprávnění pro její používání. Mimo to nemůže student provést žádnou jinou akci.
- **Lektor** – přihlášený uživatel aplikace představující lektora. Lektor má administrátorem přiřazeny určité předměty, v rámci nichž může spravovat přiřazené studenty, úlohy a kritéria jejich hodnocení. Poté, co studenti k dané úloze odevzdají prostřednictvím samostatného uživatelského rozhraní svá řešení, si je může lektor prohlédnout a ručně je ohodnotit, případně změnit již existující automatizované hodnocení.
- **Administrátor** – přihlášený uživatel aplikace představující administrátora. Administrátor se stará o registraci a následnou správu účtů lektorů a studentů. Dále spravuje také všechny předměty a přiřazuje k nim lektory, kteří budou zodpovídat za jejich obsah. Administrátoři mohou navíc provádět všechny akce lektorů, a to napříč všemi předměty.

2.1.4 Případy užití

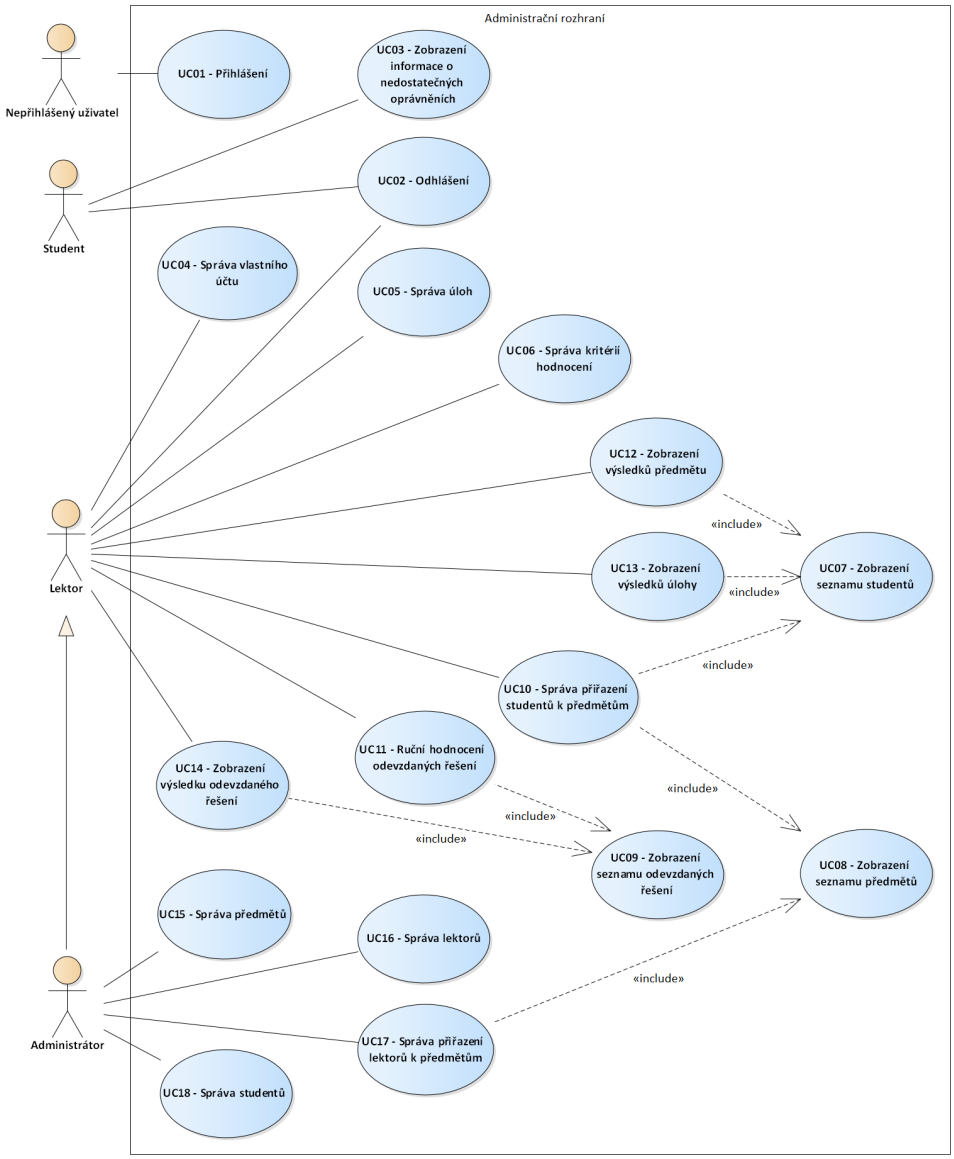
Jelikož mohou být funkční požadavky zadány na uživatelské úrovni poměrně abstraktním způsobem, je dále vhodné sestavit model případů užití. Ten podrobněji analyzuje, jaké činnosti má být uživatel či jiný účastník schopen prostřednictvím aplikace provést. Model případů užití se skládá v první řadě z výčtu a podrobného popisu účastníků a jednotlivých případů užití. Jejich vzájemné vztahy pak v druhé řadě přehledně zachytí diagram užití.

Přestože účastníky, tedy vykonavateli vybraného souboru činností, nemusejí býti pouze uživatelé, v našem případě je výčet účastníků ekvivalentní s výše uvedeným výčtem uživatelských rolí, a není tedy třeba jej v modelu případů užití duplikovat.

- **UC01 – Přihlášení** – nepřihlášený uživatel může zadat své přihlašovací údaje a po jejich ověření získat příslušnou uživatelskou roli.
- **UC02 – Odhlášení** – přihlášený uživatel se může odhlásit, čímž se stane nepřihlášeným uživatelem.

- **UC03 – Zobrazení informace o nedostatečných oprávněních** – studentovi se po přihlášení zobrazí informace o skutečnosti, že nemá dostatečná oprávnění pro využívání administračního rozhraní.
- **UC04 – Správa vlastního účtu** – lektor může v rámci vlastního účtu změnit svůj e-mail a heslo.
- **UC05 – Správa úloh** – lektor může přidávat, upravovat a odebírat úlohy v rámci daného předmětu.
- **UC06 – Správa kritérií hodnocení** – lektor může přidávat, upravovat a odebírat kritéria hodnocení dané úlohy.
- **UC07 – Zobrazení seznamu studentů** – lektor si může zobrazit seznam všech studentů registrovaných do systému.
- **UC08 – Zobrazení seznamu předmětů** – lektor si může zobrazit seznam všech předmětů, ke kterým je přiřazen.
- **UC09 – Zobrazení seznamu odevzdaných řešení** – lektor si může zobrazit seznam všech studentem odevzdaných řešení ke zvolené úloze ve svém předmětu.
- **UC10 – Správa přiřazení studentů k předmětům** – lektor může přidávat a odebírat studenty z vlastních předmětů.
- **UC11 – Ruční hodnocení odevzdaných řešení** – lektor může ručně ohodnotit studentem odevzdané řešení nebo takové hodnocení kdykoliv změnit.
- **UC12 – Zobrazení výsledků předmětu** – lektor si může zobrazit výsledky libovolného studenta ve zvoleném předmětu.
- **UC13 – Zobrazení výsledků úlohy** – lektor si může zobrazit výsledné hodnocení libovolného studenta v rámci zvolené úlohy.
- **UC14 – Zobrazení výsledku odevzdaného řešení** – lektor si může zobrazit hodnocení odevzdaného řešení.
- **UC15 – Správa předmětů** – administrátor může přidávat, upravovat a odstraňovat předměty.
- **UC16 – Správa lektorů** – administrátor může přidávat, upravovat a odstraňovat uživatelské účty lektorů.
- **UC17 – Správa přiřazení lektorů k předmětům** – administrátor může přidávat a odebírat lektory ze všech předmětů.
- **UC18 – Správa studentů** – administrátor může přidávat, upravovat a odstraňovat uživatelské účty studentů.

Obrázek 2.1: Model případů užití – diagram užití



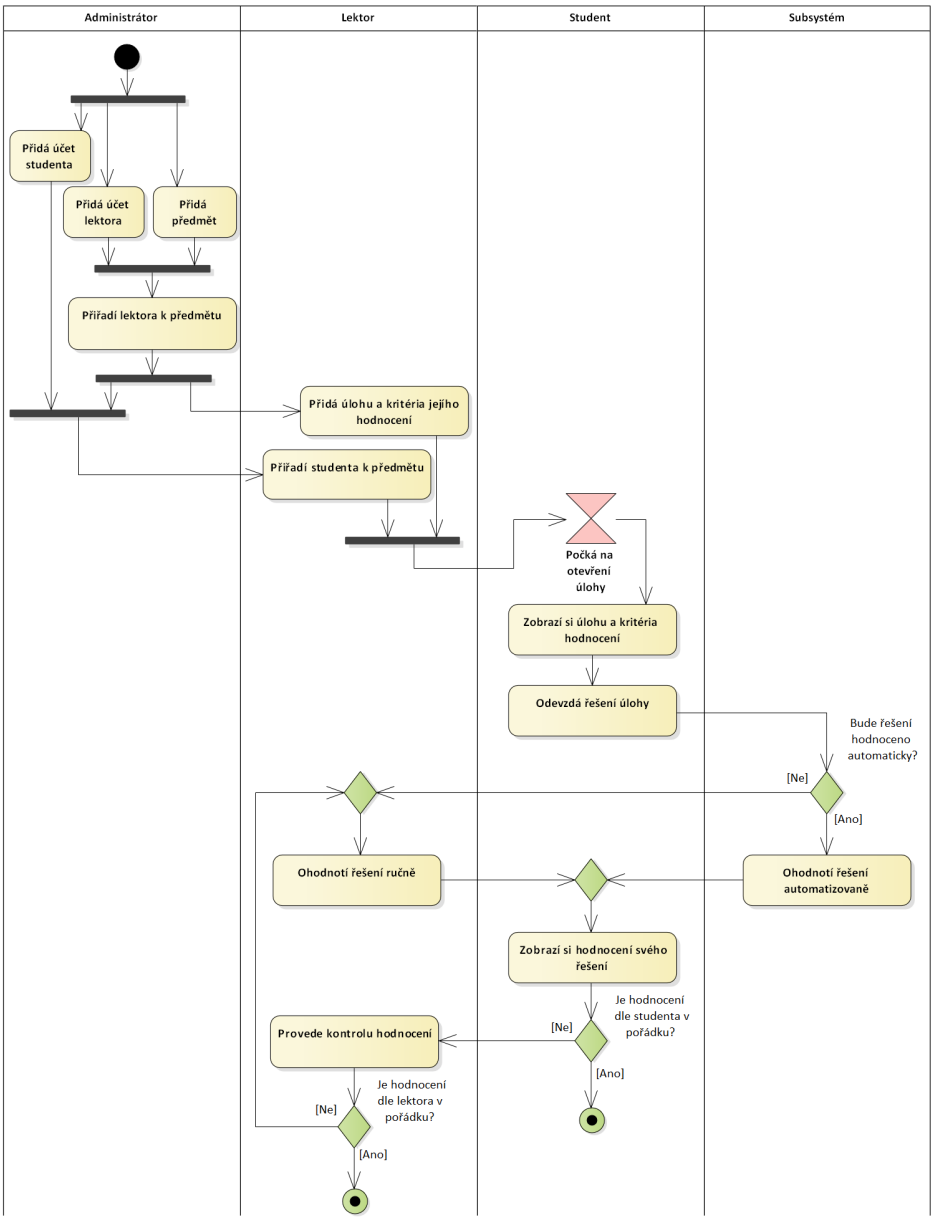
2.2 Analýza procesů

Analýza procesů si klade za cíl lepší pochopení těch činností uživatele, které budou následně vyvíjenou aplikací podporovány. Jelikož je vyvíjené administrační rozhraní z tohoto hlediska aplikací orientovanou spíše datově nežli procesně, jsou jí podporované procesy z většiny přímočaré. Pozornost si však zaslouží postup práce s celým systémem SharpTest, zvláště pak způsob hodnocení studenty odevzdaných řešení.

2.2.1 Workflow model

Workflow model sestává z posloupnosti činností a rozhodnutí, která v tomto případě představují možný průchod celou funkcionalitou systému SharpTest. K jejich přehlednému znázornění byl využit diagram aktivit. Všechny činnosti vykonávané přes administrační rozhraní jsou v diagramu dobře identifikovatelné, protože za ně zodpovídají uživatelé v roli lektora či administrátora.

Obrázek 2.2: Workflow model – diagram aktivit



Systém SharpTest obsahuje po svém nasazení vždy alespoň jeden účet administrátora. Předtím, než se systémem mohou začít pracovat lektoři a studenti, musí administrátor zaregistrovat i jejich uživatelské účty. V jeho kompetenci je také vytvoření předmětů platných pro celou instanci systému SharpTest a přiřazení lektorů zodpovědných za správu jejich obsahu.

Lektor do předmětů přidá sadu úloh, ke každé z nich pak seznam kritérií jejího hodnocení. Během přidávání úlohy lektor zvolí mimo jiné i datum a čas jejího otevření pro studenty. Součástí úloh může být také referenční řešení a testovací data určená pro případné automatizované hodnocení. Dále pak k předmětu přiřadí studenty, kteří budou zadané úlohy vypracovávat.

Student počká do doby, než bude příslušná úloha otevřena. Po otevření úlohy si zobrazí podrobný popis a kritéria hodnocení, na základě kterých vypracuje vlastní řešení. Jím vypracované řešení úlohy následně odevzdá do systému SharpTest k ohodnocení. Student má možnost odevzdat k jedné úloze více vlastních řešení, maximální počet odevzdání je však limitován.

Testovací subsystém na základě kategorie úlohy rozhodne, jakým způsobem bude odevzdané řešení hodnoceno. Pokud má být hodnocení odevzdaného řešení provedeno automatizovaně, provede jej testovací subsystém za pomoci referenčního řešení a testovacích dat. V opačném případě počká do doby, dokud lektor neohodnotí toto studentem odevzdané řešení ručně.

Po ohodnocení úlohy má student možnost prohlédnout si výsledek hodnocení a případné poznámky k jednotlivým kritériím hodnocení. Pokud dojde k názoru, že je jeho hodnocení provedeno špatně, má možnost požádat lektora o přehodnocení. Lektor má možnost zobrazit si výsledky aktuálního hodnocení a v případě, že v hodnocení odhalí nějakou chybu, může hodnocení změnit. Takto změněné hodnocení úlohy je pak znovu zobrazeno studentovi.

2.3 Analýza domény

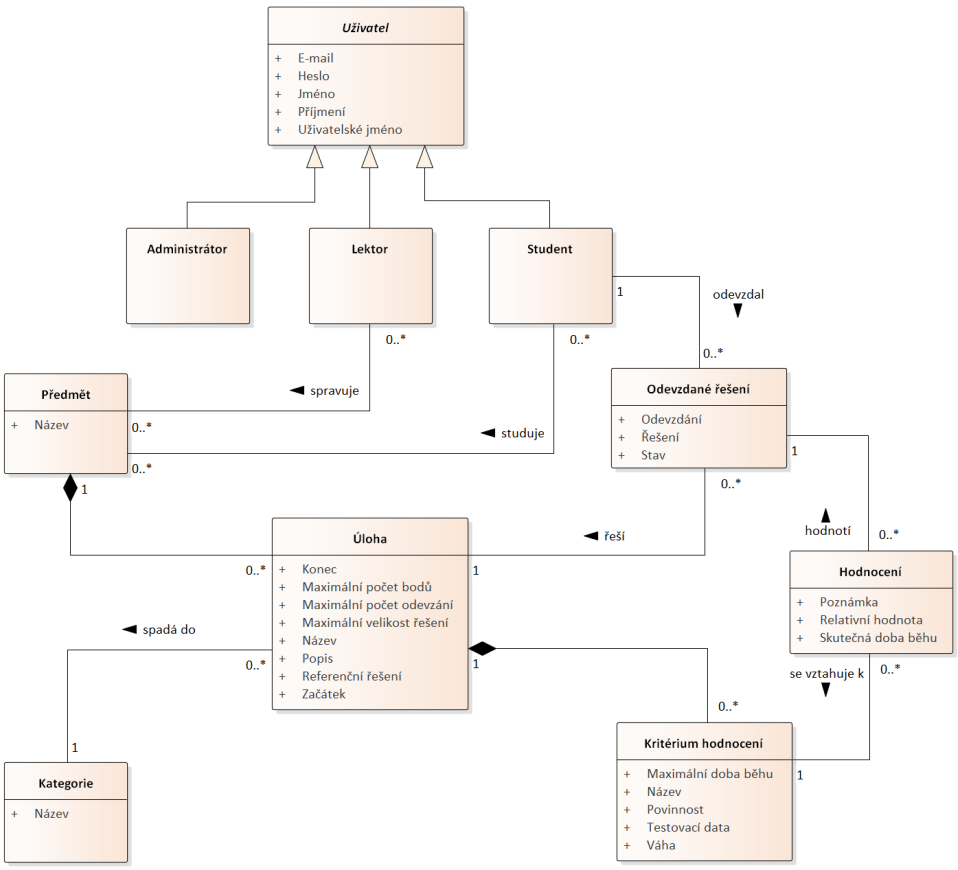
Analýza problémové domény poskytuje pohled na vyvíjenou aplikaci ze strukturálního, respektive datového hlediska. To znamená, že na rozdíl od analýzy procesů, se nezabývá posloupností činností vedoucích ke vzniku konkrétních dat, nýbrž tím, jaká je jejich struktura a vzájemná souvislost.

Její snahou je v první řadě identifikovat všechny entity nacházející se v problémové doméně. U každé entity se pak dále zabývá jejími vlastnostmi, stavy, vztahy s ostatními entitami a případnými omezeními, která jsou na ně kladena.

2.3.1 Doménový model

Doménový model je podrobným výčtem entit nacházejících se v problémové doméně. Pro zachycení jejich názvů, vlastností a vzájemných vztahů je použit diagram tříd doplněný navazujícím textovým popisem. Životní cyklus vybraných entit je dále znázorněn pomocí stavových diagramů.

Obrázek 2.3: Doménový model – diagram tříd



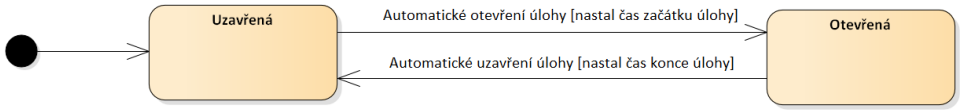
- **Uživatel** – abstraktní třída reprezentující libovolného uživatele systému. O každém uživateli je nezbytné uchovat jeho jméno, příjmení a přihlašovací údaje v podobě uživatelského jména a hesla. Volitelně lze uchovat také e-mailovou adresu. Konkrétními implementacemi této abstraktní třídy jsou odvozené třídy **Administrátor**, **Lektor** a **Student**, které zastupují jednotlivé uživatelské role popsané výše.

- **Předmět** – třída reprezentující vyučovaný předmět. O každém předmětu je uchován jeho název. K předmětu může být přiřazen libovolný počet studentů, kteří daný předmět studují a mají přístup k jeho obsahu. Stejně tak k němu může být přiřazen libovolný počet lektorů, kteří pak společně zodpovídají za jeho správu.

- **Úloha** – třída reprezentující úlohu zadanou v rámci daného předmětu. O každé úloze je nezbytné uchovat její název a maximální počet bodů získaných za její správné řešení. Dalšími nezbytnými vlastnostmi, které mají vliv na aktuální stav úlohy, jsou datum a čas začátku úlohy a da-

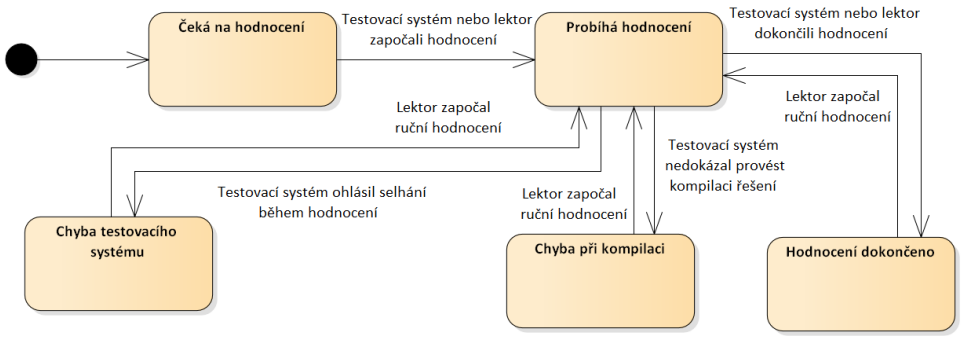
tum a čas jejího konce. Volitelně lze uchovat také podrobný textový popis zadání či údaje o omezení maximální velikosti souboru nebo maximálního počtu odevzdaných řešení. V případě, že má být řešení úlohy hodnoceno automatizovaně, musí úloha obsahovat také referenční řešení. Každá úloha je součástí právě jednoho předmětu a spadá do zvolené **Kategorie**. Dále se může úloha nacházet v otevřeném nebo uzavřeném stavu. Přechody mezi těmito stavy popisuje následující stavový diagram.

Obrázek 2.4: Stav úlohy – stavový diagram



- **Kritérium hodnocení** – třída reprezentující kritérium, podle kterého bude související úloha hodnocena. O kritériu hodnocení je nezbytné uchovat jeho název, váhu vyjadřující podíl relativního hodnocení na celkovém počtu bodů získaných z úlohy a informaci o tom, zdali je kritérium hodnocení povinné či nikoliv. V případě, že má být řešení související úlohy hodnoceno automatizovaně, je nutné uchovat testovací data nutná pro otestování daného kritéria. Volitelně lze uchovat také údaj o omezení maximální doby běhu řešení zpracovávajícího testovací data. Každé kritérium hodnocení je součástí právě jedné úlohy.
- **Odevzdané řešení** – třída reprezentující studentem odevzdané řešení vybrané úlohy. O odevzdaném řešení je nezbytné uchovat datum a čas jeho odevzdání a aktuální stav. Dále musí obsahovat samotné řešení úlohy v podobně datového souboru. Možné stavy odevzdaného řešení a přechody mezi nimi popisuje níže uvedený stavový diagram.

Obrázek 2.5: Stav odevzdaného řešení – stavový diagram



- **Hodnocení** – třída reprezentující hodnocení odevzdaného řešení vůči zvolenému kritériu hodnocení. O každém hodnocení je nezbytné uchovat relativní hodnotu vyjadřující míru splnění daného kritéria. Volitelně lze uchovat také textovou poznámku k hodnocení nebo skutečnou dobu běhu odevzdaného řešení s danými testovacími daty.

2.3.2 Doménová omezení

Vlastnosti entit a jejich vzájemné vztahy mohou podléhat dalším omezením, která není možné zachytit přímo v doménovém modelu. Tato omezení jsou přesto klíčová pro následnou implementaci aplikační logiky. Proto jsou uvedena samostatně v následujícím seznamu včetně jejich textového popisu.

- **OM01 – Začátek a konec úlohy** – datum a čas začátku úlohy musí předcházet datu a času konce úlohy.
- **OM02 – Otevřená úloha** – student může odevzdat řešení pouze k úloze, která je v otevřeném stavu.
- **OM03 – Počet odevzdání** – student může odevzdat řešení k úloze pouze v případě, že zatím nedosáhl maximálního počtu odevzdání pro tuto úlohu.
- **OM04 – Velikost souboru** – student může odevzdat řešení k úloze pouze v případě, že velikost datového souboru s řešením nepřekročila maximální velikost souboru pro tuto úlohu.
- **OM05 – Neohodnocené řešení** – student může odevzdat další řešení k úloze pouze v případě, že již bylo jeho předchozí řešení ohodnoceno.
- **OM06 – Oprávnění studenta** – student může přistupovat pouze k těm předmětům, úlohám a kritériím hodnocení, která souvisejí s jím studovanými předměty.
- **OM07 – Vlastní řešení** – student může přistupovat pouze ke svým odevzdaným řešením a k souvisejícím hodnocením.
- **OM08 – Oprávnění lektora** – lektor může spravovat pouze takové předměty, úlohy a kritéria hodnocení, která souvisejí s jím spravovanými předměty.
- **OM09 – Hodnocení řešení** – lektor může hodnotit pouze ta odevzdaná řešení, která souvisejí s úlohami v jím spravovaných předmětech.
- **OM10 – Doba běhu** – doba běhu studentem odevzdaného řešení na testovacích datech nesmí překročit maximální dobu běhu uvedenou u daného kritéria hodnocení.

Omezení se mohou pojit také se smyčkami, tedy cyklickými závislostmi mezi entitami, které jsou patrné z diagramu tříd. Každá taková smyčka představuje potenciální integritní riziko a v některých případech je nezbytné ji ošetřit samostatným integritním omezením. Uvedena je zde tedy i diskuze jednotlivých smyček.

- **Smyčka Student-Předmět-Úloha-Řešení** – představuje integritní riziko. Je nezbytné zajistit, aby mohl student odevzdat své řešení pouze k takové úloze, která je součástí jím studovaného předmětu.
- **Smyčka Řešení-Hodnocení-Kritérium-Úloha** – představuje integritní riziko. Je nezbytné zajistit, aby bylo u odevzdaného řešení hodnoceno vždy takové kritérium, které náleží k jím řešené úloze.

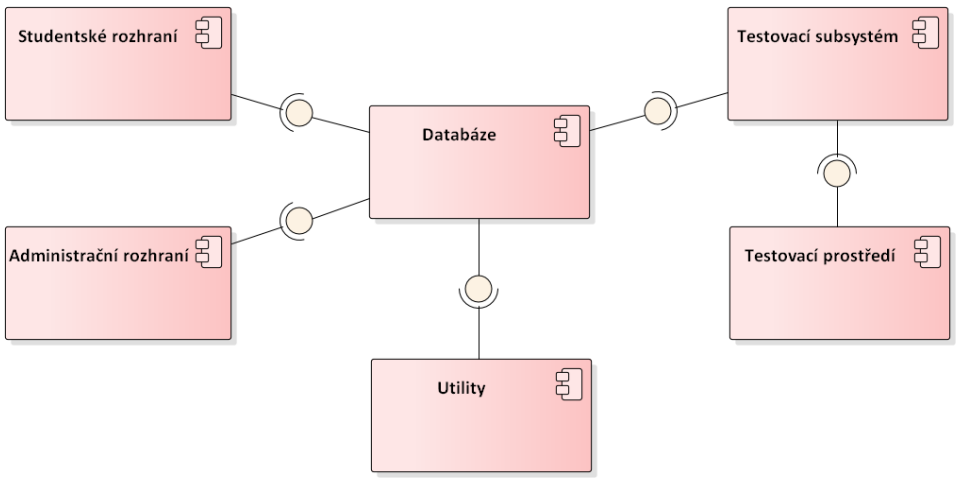
2.4 Analýza stávajícího systému SharpTest

Jedním z obecných požadavků na vyvíjené administrační rozhraní je jeho integrace se stávajícím systémem SharpTest. Aby bylo možné správně navrhnout, jakým způsobem bude administrační rozhraní integrováno, je nejdříve nezbytné pochopit a popsat architekturu stávajícího systému, jeho datovou vrstvu a některá další východiska, kterými bude tato zamýšlená integrace ovlivněna.

2.4.1 Architektura

SharpTest je modulárním systémem, který se v současnosti skládá z šesti propojených modulů. Jednotlivé moduly a způsob jejich vzájemného propojení ilustruje následující diagram komponent.

Obrázek 2.6: Architektura systému SharpTest – diagram komponent



Nejdůležitějším z modulů je centrální databáze představující datovou vrstvu systému. Jelikož právě na tuto vrstvu bude administrační rozhraní integrováno, jsou její vlastnosti podrobněji diskutovány v následující části práce.

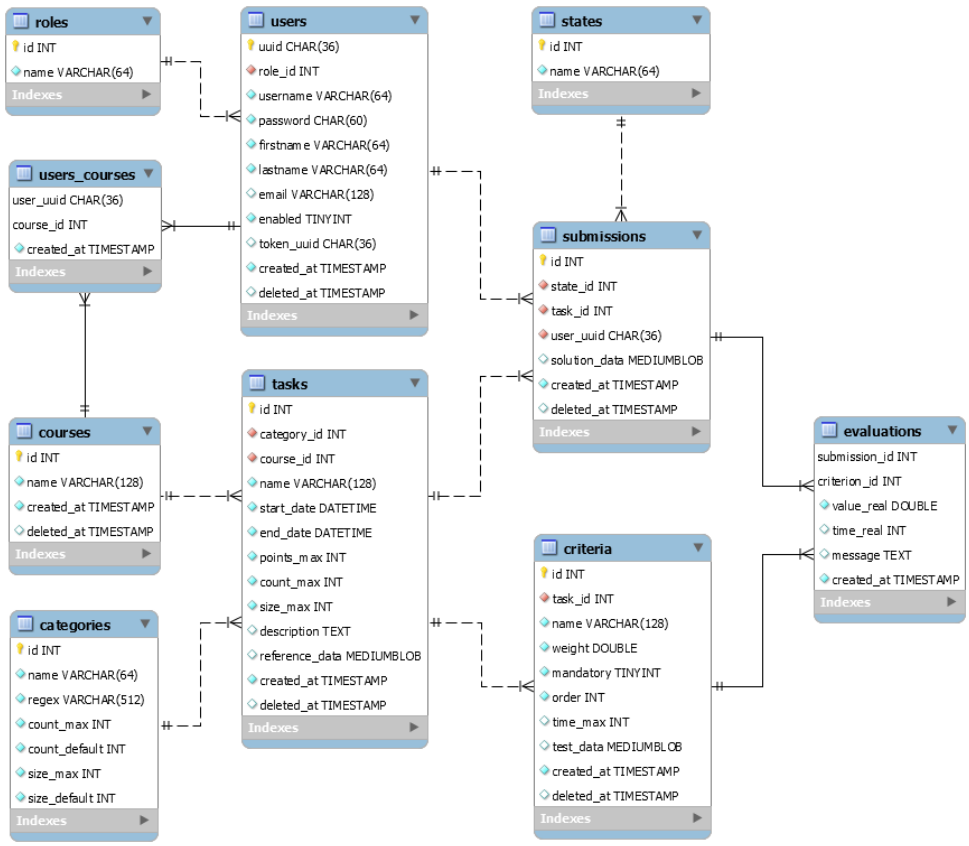
Na databázi mají být napojena dvě uživatelská rozhraní. Studentské rozhraní umožňuje studentům zobrazit si zadané úlohy, odevzdat k nim svá řešení a zjistit výsledné hodnocení. Administrační rozhraní je vyvíjeno v rámci této práce. Jeho funkcionalita byla prozatím suplována pomocí sady utilit.

Poslední dva moduly umožňují automatizované testování úloh. Testovací subsystém si z databáze načte studentem odevzdané řešení a ohodnotí ho za pomoci referenčního řešení a testovacích dat. K tomu využívá omezené testovací prostředí, ve kterém je studentem odevzdané řešení spuštěno a testováno.

2.4.2 Datová vrstva

Datová vrstva je v systému SharpTest řešena pomocí relační databáze MySQL. Uchovány jsou v ní nejen informace o uživatelích systému a zadaných úlohách, ale také studenty odevzdaná řešení a jejich hodnocení. Strukturu uložených dat zachycuje následující entitně relační diagram.

Obrázek 2.7: Databáze SharpTest – entitně relační diagram



Protože se při návrhu struktury databáze vycházelo ze stejné problémové domény, je možné si povšimnout, že až na některá specifika se jedná o transformaci doménového modelu.

Mezi tato specifika patří v tabulce `users` atribut `enabled` sloužící k dočasnému zablokování uživatele nebo atribut `token_uuid` obsahující identifikaci autorizačního tokenu. Dále je v tabulce `criteria` přidán atribut `order` umožňující vlastní řazení kritérií hodnocení.

Většina entit je doplněna o atribut `created_at`, který obsahuje časovou známku okamžiku vložení záznamu do databáze, a také o atribut `deleted_at`, pomocí něhož mohou být v databázi skryty vybrané záznamy bez nutnosti jejich nevratného fyzického odstranění.

Série atributů `count_max`, `count_default`, `size_max` a `size_default` obsažená v tabulce `categories` umožňuje určit maximální a výchozí hodnoty maximálního počtu odevzdání a maximální velikosti odevzdaného souboru pro související úlohy z této kategorie. Atribut `regex` pak obsahuje regulární výraz sloužící k filtrování souborů, jež může student k těmto úlohám odevzdat.

Posledním specifikem, které by určitě nemělo být opomenuto, jsou atributy `solution_data` z tabulky `submissions`, `reference_data` z tabulky `tasks` a `test_data` z tabulky `criteria`. Tyto atributy jsou datového typu BLOB a obsahují výhradně ZIP archiv s odevzdaným či referenčním řešením úlohy nebo s testovacími daty.

2.4.3 Výpočet výsledků

Administrační rozhraní má dle požadavků zadavatele umožňovat kromě správy studentů, předmětů a úloh také jejich hodnocení a následné poskytnutí přehledu o dosažených výsledcích. Výsledné hodnocení studenta v rámci odevzdaného řešení, úlohy či celého předmětu však není v databázi explicitně uvedeno a je nezbytné jej vypočítat agregací dílčích hodnot napříč více entitami.

2.4.3.1 Výpočet výsledků pro odevzdané řešení

Výsledné hodnocení $H_S(s)$ pro dané odevzdané řešení s se vypočte jako

$$H_S(s) = \sum_{e \in E_S(s)} \Big(e[value\_real] \cdot r(e)[weight] \cdot t(r(e))[points\_max] \Big), \quad (2.1)$$

kde
$*[x]$ je funkce, která pro entitu $*$ vrátí hodnotu jejího atributu x,
$r(x)$ je funkce, která vrátí právě jedno kritérium související s hodnocením x,
$t(x)$ je funkce, která vrátí právě jednu úlohu související s kritériem x,
$E_S(x)$ je funkce, která vrátí množinu hodnocení souvisejících s řešením x.

2.4.3.2 Výpočet výsledků pro úlohu

Výsledné hodnocení $H_T(t, u)$ pro danou úlohu t a studenta u se vypočte jako

$$H_T(t,u) = \max\left(\left\{H_S(s)\middle| s \in \left(S_T(t) \cap S_U(u)\right)\right\} \cup \left\{0\right\}\right), \tag{2.2}$$

kde
$S_T(x)$ je funkce, která vrátí množinu řešení souvisejících s úlohou x,
$S_U(x)$ je funkce, která vrátí množinu řešení souvisejících se studentem x.

2.4.3.3 Výpočet výsledků pro předmět

Výsledné hodnocení $H_C(c, u)$ pro daný předmět c a studenta u se vypočte jako

$$H_C(c,u) = \sum_{t \in T_C(c)} H_T(t,u), \tag{2.3}$$

kde
$T_C(x)$ je funkce, která vrátí množinu úloh souvisejících s předmětem x.

Kapitola 3

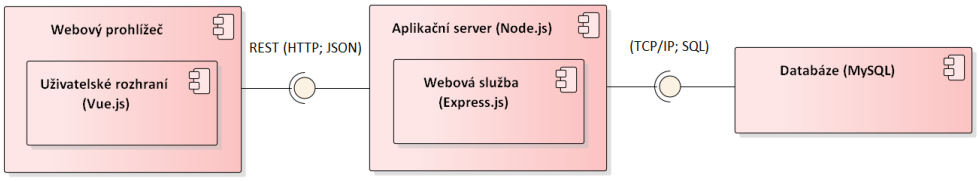
Návrh

Návrh je druhou fází životního cyklu softwarového produktu. Během této fáze jsou na základě poznatků z předchozí analytické části zvoleny odpovídající technologie a navržena architektura aplikace tak, aby byly naplněny požadavky zadavatele a potřeby uživatelů. Návrh se také zabývá strukturou a popisem některých významných částí aplikace, jako je například aplikační nebo uživatelské rozhraní.

3.1 Architektura a technologie

Aplikace administračního rozhraní bude implementována jako jednostránková webová aplikace mající třívrstvou architekturu. Vzájemné propojení vrstev spolu s dále diskutovanou volbou konkrétních technologií souhrnně popisuje níže uvedený diagram komponent 3.1.

Obrázek 3.1: Architektura aplikace – diagram komponent



Důvodem volby webové aplikace je absence nutnosti instalace a možnost snadného a okamžitého spuštění na různých zařízeních i operačních systémech. Alternativy v podobě desktopové nebo samostatné mobilní aplikace oproti tomu nepřinášejí pro daný účel použití žádnou signifikantní výhodu.

Výhodou jednostránkové aplikace je okamžitá reakce na vstupy uživatele v reálném čase. Jednostránková webová aplikace se tak svým chováním blíží nativním desktopovým aplikacím a poskytuje pro uživatele přívětivější uživatelské rozhraní, nežli je tomu v případě aplikací vícestránkových. Dále díky tomu,

že se ze serveru stahují vždy pouze aktuálně potřebná data namísto celých stránek, je navíc dosaženo menšího zatížení sítě. Nevýhoda v podobě komplikované optimalizace pro internetové vyhledávače zde, vzhledem k účelu použití jakožto uzavřené aplikace pro interní účely, nehraje důležitou roli.

Využití třívrstvé architektury zajistí snadnější dodržení principů vysoké soudržnosti a nízké provázanosti. Každá vrstva je odpovědná za specifickou sadu činností, díky čemuž je snadnější údržba výsledné aplikace i případná budoucí změna technologie na některé z vrstev. Velkou výhodou je možnost napojení prostřední aplikační vrstvy na budoucí aplikace systému SharpTest, které již nebudou muset komunikovat přímo s centrální relační databází, nýbrž budou využívat služeb a logiky již hotové aplikační vrstvy.

3.1.1 Datová vrstva

Datová vrstva vyvíjené aplikace bude představována stávající datovou vrstvou systému SharpTest, která je zajištěna pomocí relační databáze MySQL. V této databázi jsou kromě informací o jednotlivých entitách uloženy i datové soubory s řešením úloh a testovacími daty. Databáze tak do jisté míry supluje funkci souborového systému, což je umožněno obecně malou velikostí ukládaných souborů. Komunikace s databází bude probíhat prostřednictvím počítačové sítě a protokolu TCP/IP. Pro dotazování nad daty uloženými v databázi je standardně využit jazyk SQL.

Z důvodu integrace datové vrstvy na stávající součást systému SharpTest zde není možné ovlivnit volbu použitých technologií nebo způsob komunikace.

3.1.2 Aplikační vrstva

Aplikační vrstva bude zajištěna aplikačním serverem vyvinutým na platformě Node.js. Aplikační server představuje serverovou část aplikace, která bude implementovat aplikační logiku v podobě vynucení doménových omezení, validace vstupních dat a řízení přístupu k datům uloženým v databázi dle příslušných uživatelských rolí.

Součástí aplikačního serveru bude aplikační rámec Express.js, pomocí kterého budou realizovány dvě důležité funkcionality. První z nich bude funkcionalita webového serveru. Aplikační server tak bude sloužit zároveň jako webový server, který poskytne ke stažení všechny nezbytné datové soubory klientské části aplikace.

Druhou klíčovou funkcionalitou bude RESTful webová služba vystavující aplikační rozhraní typu REST. Toto rozhraní bude sloužit jakožto datový zdroj klientské části aplikace. Komunikace bude probíhat přes standardní protokol HTTP (respektive zabezpečený protokol HTTPS) a datový formát JSON. Autentizace uživatelů a autorizace požadavků na REST API bude řešena technologií JSON Web Tokens.

Důvodem využití platformy Node.js je její specifické zaměření na vývoj serverových aplikací a aplikačních serverů. Díky její jednovláknové architektuře a asynchronním vstupně výstupním operacím je možné snadno vyvíjet rozsáhlé serverové aplikace, které jsou výkonné a zároveň dobře škálovatelné. Za výhodu lze považovat i možnost vývoje v jazyce JavaScript, ve kterém bude implementována i klientská část aplikace. Obě části aplikace tak budou napsány ve stejném programovacím jazyce a pro vývojáře bude snadnější se v kódu zorientovat.

Výhodou aplikačního rámce Express.js je skutečnost, že poskytuje velmi dobrý základ pro práci s protokolem HTTP. Jednak v sobě integruje funkcionalitu webového serveru pro poskytování statického obsahu, dále pak řeší syntaktickou analýzu hlavičky i těla HTTP požadavků, jejich směrování a některé další problémy. Programování takovýchto pomocných funkcionalit by znamenalo množství zbytečné práce s malou přidanou hodnotou. Na druhou stranu je tento aplikační rámec pojat minimalisticky a na rozdíl od mnohých konkurentů neobsahuje funkcionality, které by byly nadbytečné.

Aplikační rozhraní typu REST je použito pro svou jednoduchost, snadnou rozšiřitelnost a menší nároky na objem přenesených dat oproti alternativním technologiím jako jsou SOAP nebo RPC. Vzhledem ke skutečnosti, že je vyvíjená aplikace orientovaná spíše datově a vesměs nezahrnuje žádné složité procesy, lze v tomto kontextu za výhodu REST považovat i její zaměření na datové zdroje namísto služeb či volání vzdálených procedur. Další výhodou REST API je jeho bezstavovost, díky které mohou být odpovědi na požadavky ukládány do mezipaměti.

3.1.3 Prezentační vrstva

Prezentační vrstva bude zajištěna pomocí interaktivního uživatelského rozhraní, které poběží v rámci webového prohlížeče. Toto uživatelské rozhraní bude představovat klientskou část aplikace a bude naprogramováno s využitím aplikačního rámce Vue.js v jazyce JavaScript. Plnohodnotné jednostránkové aplikace bude dosaženo pomocí rozšíření Vue-router, které prováže jednotlivé obrazovky aplikace s URL adresou. Pro správu sdíleného stavu, zvláště pak pro uchování údajů o přihlášeném uživateli a autorizačního tokenu, bude sloužit rozšíření Vuex.

Jako základ responzivního web designu, který umožní používání aplikace na všech typech zařízení nehledě na rozměry jejich obrazovky, bude využito knihovny Bootstrap. Lepší propojení aplikačního rámce Vue.js s knihovnou Bootstrap dále zajistí knihovna BootstrapVue, jejíž široká sada komponent uživatelského rozhraní bude zužitkována především při tvorbě formulářů a jejich validaci.

Uživatelská přívětivost bude podpořena sadou ikonek Font Awesome. Tyto ikonky navíc nahradí rozměrné popisky některých ovládacích prvků při zobra-

zení na malých obrazovkách. Dále bude užito knihovny Chart.js[1] pro vykreslení grafů a textového editoru TinyMCE[2] pro úpravu složitějších textů.

Důvodem použití aplikačního rámce Vue.js oproti samotnému JavaScriptu je především dvoucestná synchronizace dat a využití znovupoužitelných komponent. Těmito výhodami samozřejmě disponují i konkurenční technologie jako Angular nebo React. I oproti těm má však Vue.js výhody v podobě velmi malé velikosti, flexibilitě a snadnému nasazení.

Výhodou knihovny Bootstrap je zásadní omezení množství kaskádových stylů, které je nezbytné napsat pro základní stylování komponent uživatelského rozhraní. Jelikož v případě vyvíjené aplikace záleží především na funkcionalitě a rozvržení grafických prvků a nikoliv na specifickém grafickém designu, je základní stylování poskytnuté v rámci knihovny Bootstrap zcela dostačující. O to užitečnější je pak využití nadstavby BootstrapVue, která ušetří velké množství práce se psaním pokročilé funkcionality často využívaných komponent, jako jsou například tabulky, dialogy a mnohé formulářové prvky.

3.2 Návrh aplikačního rozhraní REST

Kromě architektury a volby použitých technologií se bude návrh dále zabývat popisem klíčových součástí aplikace. Mezi nejdůležitější z nich patří aplikační rozhraní REST, které bude oddělovat serverovou a klientskou část aplikace. Díky tomu, že bude dopředu jasně definována struktura rozhraní oddělující tyto dvě části, bude při realizaci lépe fixována jimi poskytovaná funkcionalita. I přes to, že bude klientskou aplikací využita pouze část aplikačního rozhraní, samotné rozhraní bude navrženo tak, aby bylo univerzální a mohlo sloužit i pro případné budoucí aplikace a moduly systému SharpTest.

Struktura rozhraní je dána výčtem datových zdrojů a jejich popisem pro různé druhy požadavků. Stejně důležitý je také popis omezení přístupu k datovým zdrojům vzhledem k současné uživatelské roli. Každý datový zdroj má vlastní URL adresu, jež jej jednoznačně identifikuje. Operace provedená s datovým zdrojem při zpracování konkrétního požadavku je určena použitou HTTP metodou, dle které jsou jednotlivé druhy požadavků dále děleny.

3.2.1 Požadavky typu GET

Požadavky obsahující HTTP metodu GET slouží k získání konkrétní instance dané entity nebo celé jejich kolekce. Za URL adresu každé kolekce může být doplněna cesta **/:id**, čímž z ní bude vybrána pouze jediná instance. V případě nevalidního identifikátoru je vrácen stavový kód 404.

/account Vrátí údaje o uživatelském účtu. Přístup – bez omezení.

[1] NPM repozitář: `https://www.npmjs.com/package/chart.js`.
[2] NPM repozitář: `https://www.npmjs.com/package/tinymce`.

/role Vrátí kolekci všech uživatelských rolí. Přístup – bez omezení.

/state Vrátí kolekci všech stavů odevzdaného řešení. Přístup – bez omezení.

/category Vrátí kolekci všech kategorií úloh. Přístup – bez omezení.

/student Vrátí kolekci všech studentů. Přístup – lektoři a administrátoři.

/lecturer Vrátí kolekci všech lektorů. Přístup – administrátoři.

/course Vrátí kolekci přiřazených předmětů. Přístup – bez omezení.

/course/:id/student Vrátí kolekci studentů studujících vybraný předmět. Přístup – lektoři a administrátoři.

/course/:id/student/:uuid/score Vrátí počet bodů vybraného studenta získaných z daného předmětu. Přístup – lektoři a administrátoři.

/course/:id/lecturer Vrátí kolekci lektorů vyučujících vybraný předmět. Přístup – administrátoři.

/course/:id/task Vrátí kolekci všech úloh obsažených v daném předmětu. Přístup – bez omezení.

/course/:id/total Vrátí maximální možný počet bodů, který lze získat za úlohy v daném předmětu. Přístup – lektoři a administrátoři.

/task Vrátí kolekci všech přiřazených úloh. Přístup – bez omezení.

/task/:id/data Vrátí referenční řešení dané úlohy. Přístup – lektoři a administrátoři.

/task/:id/student Vrátí kolekci všech studentů přiřazených k dané úloze. Přístup – lektoři a administrátoři.

/task/:id/student/:uuid/score Vrátí počet bodů vybraného studenta získaných z dané úlohy. Přístup – lektoři a administrátoři.

/task/:id/criterion Vrátí kolekci všech kritérií hodnocení vybrané úlohy. Přístup – bez omezení.

/task/:id/submission Vrátí kolekci všech odevzdaných řešení dané úlohy. Přístup – lektoři a administrátoři.

/criterion Vrátí kolekci všech kritérií hodnocení ke všem přiřazeným úlohám. Přístup – bez omezení.

/criterion/:id/data Vrátí testovací data dané úlohy. Přístup – lektoři a administrátoři.

/submission Vrátí kolekci všech odevzdaných řešení ke všem přiřazeným úlohám. Přístup – bez omezení, studenti pouze vlastní úlohy.

/submission/:id/data Vrátí datový soubor vybraného odevzdaného řešení. Přístup – bez omezení, studenti pouze vlastní úlohy.

/submission/:id/score Vrátí výsledné hodnocení vybraného odevzdaného řešení. Přístup – bez omezení, studenti pouze vlastní úlohy.

/submission/:id/evaluation Vrátí kolekci hodnocení daného odevzdaného řešení. Přístup – bez omezení, studenti pouze vlastní ohodnocené úlohy.

3.2.2 Požadavky typu POST

Požadavky obsahující HTTP metodu POST slouží k vložení nebo vytvoření nové instance entity do zvolené kolekce. Při vkládání instance musejí být uvedeny hodnoty všech povinných vlastností. Vlastnosti vkládané instance musejí být dále validovány, aby byla zajištěna integrita dat. V případě nevalidní hodnoty některé z vlastností je vrácen stavový kód 400.

/student Vloží novou instanci studenta. Přístup – administrátoři.

/lecturer Vloží novou instanci lektora. Přístup – administrátoři.

/course Vloží novou instanci předmětu. Přístup – administrátoři.

/course/:id/student/:uuid Přiřadí vybraného studenta k danému předmětu. Přístup – lektoři a administrátoři.

/course/:id/lecturer/:uuid Přiřadí vybraného lektora k danému předmětu. Přístup – administrátoři.

/task Vloží novou instanci úlohy. Přístup – lektoři a administrátoři.

/task/:id/data Přidá referenční řešení do vybrané úlohy. Přístup – lektoři a administrátoři.

/criterion Vloží novou instanci kritéria hodnocení. Přístup – lektoři a administrátoři.

/criterion/:id/data Přidá testovací data k vybranému kritériu hodnocení. Přístup – lektoři a administrátoři.

/submission Vloží nové odevzdané řešení. Přístup – studenti.

/submission/:id/evaluation Vloží novou instanci hodnocení k vybranému odevzdanému řešení. Přístup – lektoři a administrátoři.

3.2.3 Požadavky typu PUT

Požadavky obsahující HTTP metodu PUT slouží k upravení již stávající instance dané entity. Při upravování instance nemusejí být uvedeny všechny hodnoty všech vlastností. Stejně jako v případě vkládání jsou však uvedené vlastnosti upravované instance validovány. V případě nevalidní hodnoty vlastnosti je vrácen stavový kód 400.

/student/:id Upraví stávající instanci studenta. Přístup – administrátoři.

/lecturer/:id Upraví stávající instanci lektora. Přístup – administrátoři.

/course/:id Upraví stávající instanci předmětu. Přístup – administrátoři.

/task/:id Upraví stávající instanci úlohy. Přístup – lektoři a administrátoři.

/criterion/:id Upraví stávající instanci kritéria hodnocení. Přístup – lektoři a administrátoři.

/submission/:id Uzavře hodnocení daného odevzdaného řešení upravením jeho stavu na *Hodnocení dokončeno*. Přístup – lektoři a administrátoři.

/submission/:id/evaluation/:criterion_id Upraví hodnocení daného odevzdaného řešení. Přitom změní stav odevzdaného řešení na *Probíhá hodnocení*. Přístup – lektoři a administrátoři.

3.2.4 Požadavky typu DELETE

Požadavky obsahující HTTP metodu DELETE slouží k smazání stávající instance dané entity. V případě nevalidního identifikátoru je vrácen stavový kód 404. V souladu s databázovým návrhem nedochází při těchto požadavcích u některých entit přímo k fyzickému smazání instancí, nýbrž k jejich označení a skrytí.

/student/:id Smaže stávající instanci studenta. Přístup – administrátoři.

/lecturer/:id Smaže stávající instanci lektora. Přístup – administrátoři.

/course/:id Smaže stávající instanci předmětu. Přístup – administrátoři.

/course/:id/student/:uuid Odebere vybraného studenta z daného předmětu. Přístup – lektoři a administrátoři.

/course/:id/lecturer/:uuid Odebere vybraného lektora z daného předmětu. Přístup – administrátoři.

/task/:id Smaže stávající instanci úlohy. Přístup – lektoři a administrátoři.

/task/:id/data Smaže referenční řešení z vybrané úlohy. Přístup – lektoři a administrátoři.

/criterion/:id Smaže stávající instanci kritéria hodnocení. Přístup – lektoři a administrátoři.

/criterion/:id/data Smaže testovací data z vybraného kritéria hodnocení. Přístup – lektoři a administrátoři.

/submission/:id Smaže stávající odevzdané řešení. Přístup – lektoři a administrátoři.

/submission/:id/evaluation/:criterion_id Smaže hodnocení daného odevzdaného řešení. Přitom změní stav odevzdaného řešení na *Probíhá hodnocení*. Přístup – lektoři a administrátoři.

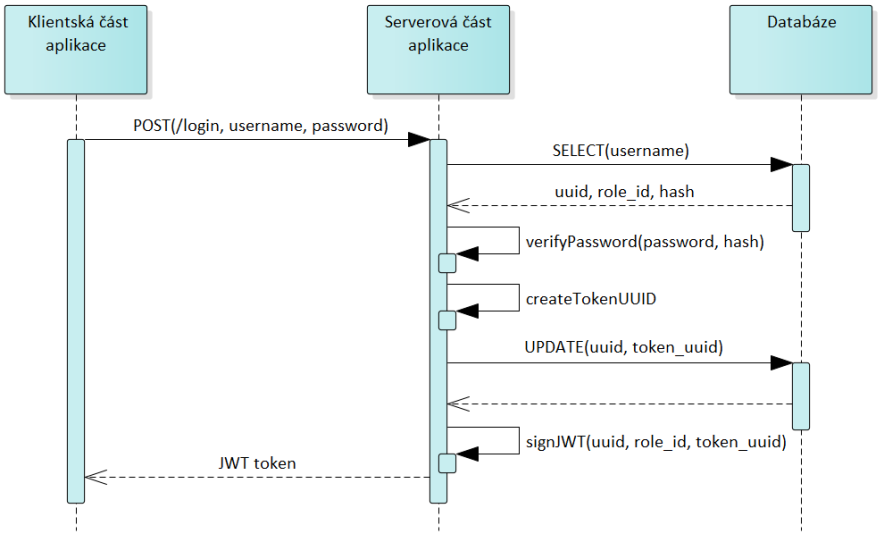
3.2.5 Autentizace a autorizace

Všechny výše jmenované požadavky vyžadují přihlášení uživatele. Proces ověření identity uživatele a přiřazení příslušné uživatelské role se nazývá autentizace a je řešen pomocí speciálního požadavku typu POST, který ve svém těle obsahuje přihlašovací údaje.

/login Přihlásí uživatele na základě poskytnutých přihlašovacích údajů.

Odpovědí na uvedený požadavek je JWT token prokazující totožnost uživatele. Způsob získání JWT tokenu popisuje sekvenční diagram 3.2.

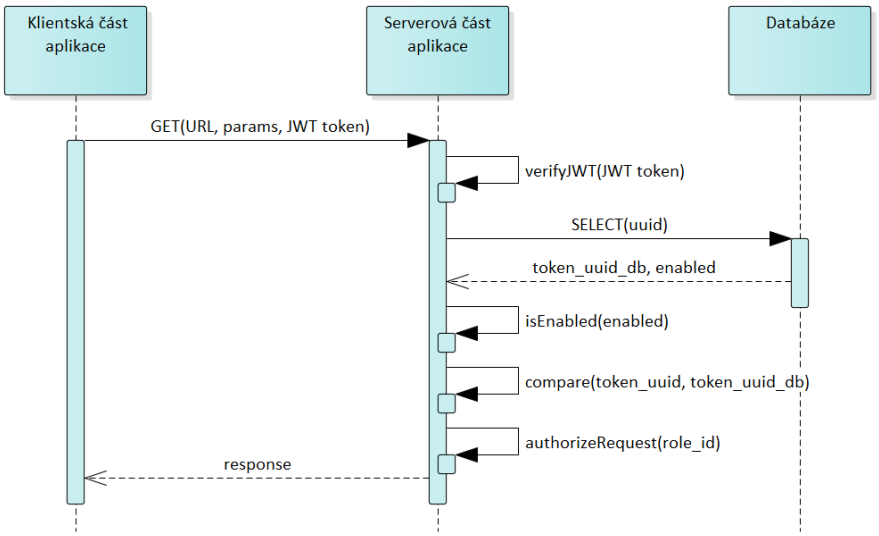
Obrázek 3.2: Autentizace – sekvenční diagram



Po odeslání požadavku POST na aplikační rozhraní si serverová část aplikace načte z databáze nezbytné údaje o uživateli, mezi které patří také hash uživatelského hesla. Tento hash porovná s hashem vytvořeným ze zaslaného hesla a v případě jejich shody pokračuje dále. V tuto chvíli serverová část aplikace vytvoří jedinečný identifikátor tokenu, který nejdříve uloží do databáze a následně jej přiloží také do nově vzniklého JWT tokenu. Vytvořený JWT token podepíše a zašle jej zpět klientské části aplikace.

JWT token slouží k autorizaci všech dalších požadavků zaslaných na aplikační rozhraní. Proces autorizace ukázkového požadavku typu GET ilustruje sekvenční diagram 3.3.

Obrázek 3.3: Autorizace – sekvenční diagram



K ukázkovému požadavku typu GET odeslanému na aplikační rozhraní je přiložen získaný JWT token. Serverová část aplikace nejdříve ověří podpis JWT tokenu. Následně z databáze načte identifikátor tokenu pro odpovídajícího uživatele a údaj o případném zablokování jeho účtu. Na základě těchto údajů ověří, že nedošlo k zablokování uživatelského účtu a že se načtený identifikátor shoduje s identifikátorem obsaženým v samotném tokenu. V tuto chvíli má serverová část aplikace ověřenou totožnost uživatele, na základě které poskytne příslušnou odpověď na zaslaný požadavek.

Pro úplnost je nutné podotknout, že je tímto způsobem autorizace bohužel negována jedna z velkých výhod JWT tokenu. Metoda jeho ověření totiž sama o sobě nevyžaduje přístup do databáze během autorizace požadavku. Přístup do databáze je zde však nezbytný z důvodu zachování možnosti okamžitého omezení nebo odepření přístupu vybraného uživatele. Tato možnost je v daném případě znatelně důležitější, nežli zátěž databáze způsobená dodatečným dotazem do databáze.

Odhlášení uživatele je řešeno pomocí speciálního požadavku typu POST, po jehož odeslání dojde k odstranění identifikátoru tokenu z databáze. V tu chvíli již nemůže být JWT token nadále používán, protože neprojde výše uvedeným autorizačním procesem.

/logout Odhlásí přihlášeného uživatele a zruší platnost JWT tokenu.

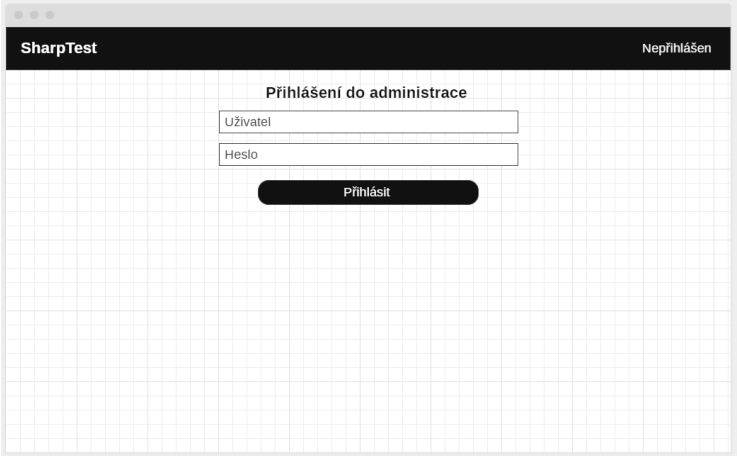
3.3 Návrh uživatelského rozhraní

Druhou důležitou součástí aplikace, jejíž strukturou se návrh dále zabývá, je uživatelské rozhraní. Pro vybrané klíčové obrazovky aplikace obsahuje návrh wireframe spolu s jeho popisem a vazbou na příslušné případy užití. Zahrnuta je také přehledová mapa zachycující přechody mezi jednotlivými obrazovkami. Díky návrhu obrazovek bude v kombinaci s jasně definovaným REST API poměrně přesně určena funkcionalita celého uživatelského rozhraní.

3.3.1 Přihlašovací obrazovka

Přihlašovací obrazovka je vstupní obrazovkou do administračního rozhraní, na kterou je přesměrován každý nepřihlášený uživatel. Nepřihlášený uživatel vyplní do formuláře své uživatelské jméno a heslo. Po kliknutí na tlačítko Přihlásit dojde k pokusu o přihlášení uživatele. V případě neúspěšného přihlášení je uživatel informován o chybně zadaných údajích a je mu dále umožněno přihlašování opakovat. Pokud se do administračního rozhraní pokusí přihlásit student, je informován o nedostatečné úrovni oprávnění pro jeho vstup. Řešené případy užití – UC01 a UC03.

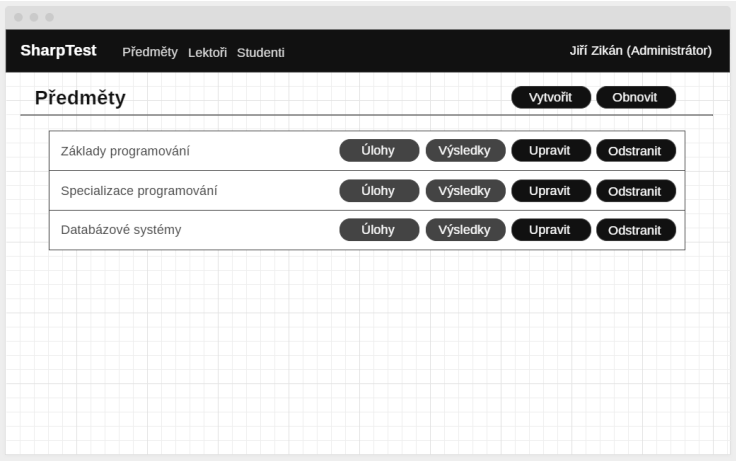
Obrázek 3.4: Přihlašovací obrazovka – wireframe



3.3.2 Seznam předmětů

Seznam předmětů slouží k zobrazení a správě předmětů. Administrátor může na této obrazovce spravovat libovolné předměty. Lektor vidí pouze výčet přiřazených předmětů, u kterých může přejít na seznam souvisejících úloh nebo si zobrazit souhrnné výsledky. Řešené případy užití – UC08 a UC15.

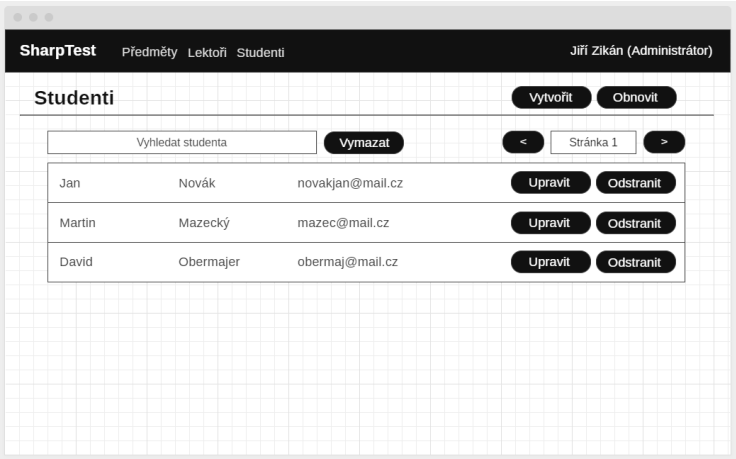
Obrázek 3.5: Seznam předmětů – wireframe



3.3.3 Seznam studentů

Seznam studentů slouží k zobrazení a správě registrovaných studentů. Administrátor může vytvářet, upravovat a mazat libovolné studenty. Lektor naproti tomu může seznam studentů pouze procházet nebo v něm vyhledávat pomocí vyhledávacího pole. Řešené případy užití – UC07 a UC18.

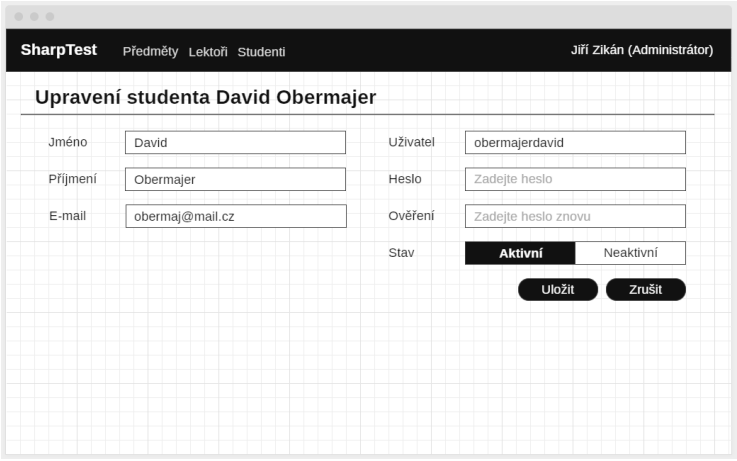
Obrázek 3.6: Seznam studentů – wireframe



3.3.4 Editor studenta

Editor studenta slouží k upravení údajů o vybraném studentovi. Mezi tyto údaje patří jméno, příjmení, e-mailová adresa a uživatelské jméno. Pomocí tohoto editoru může být studentovi také nastaveno nové heslo nebo dočasně zablokován jeho uživatelský účet. Řešené případy užití – UC18.

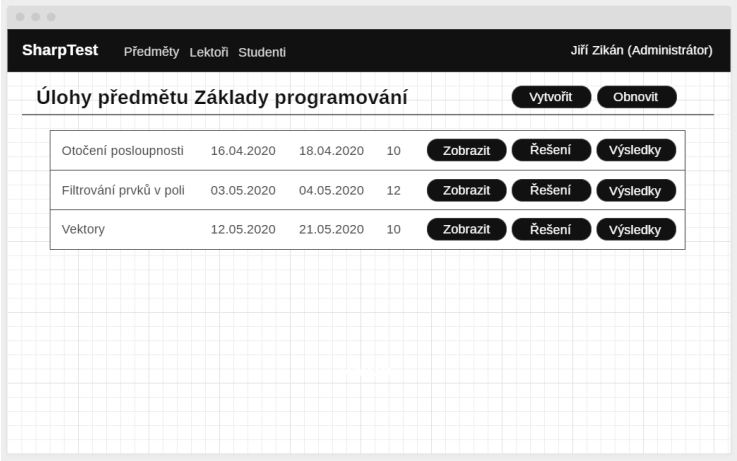
Obrázek 3.7: Editor studenta – wireframe



3.3.5 Seznam úloh

Seznam úloh slouží k zobrazení výčtu úloh souvisejících s vybraným předmětem. Lektoři i administrátoři mohou vytvářet nové úlohy a zobrazovat si detail úlohy, seznam studenty odevzdaných řešení nebo přehled výsledků. Řešené případy užití – UC05.

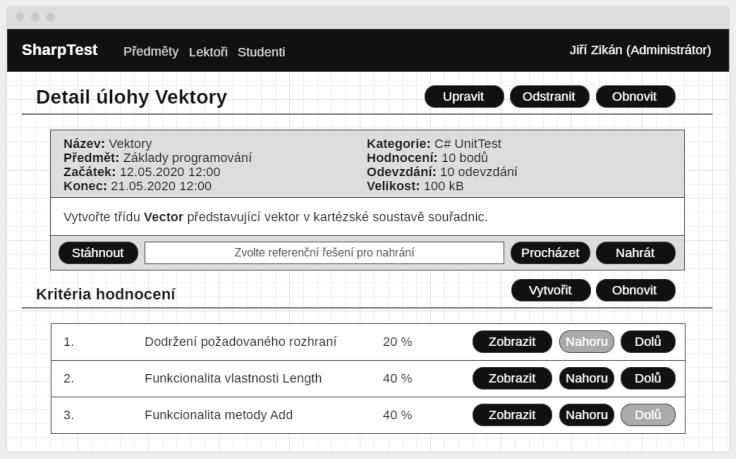
Obrázek 3.8: Seznam úloh – wireframe



3.3.6 Detail úlohy

Detail úlohy slouží k zobrazení podrobností o vybrané úloze. Lektoři i administrátoři mohou úlohu upravit, odstranit nebo k ní nahrát referenční řešení. Součástí detailu úlohy je i výčet kritérií hodnocení, který dále umožňuje vytvoření, seřazení nebo zobrazení detailu. Řešené případy užití – UC05 a UC06.

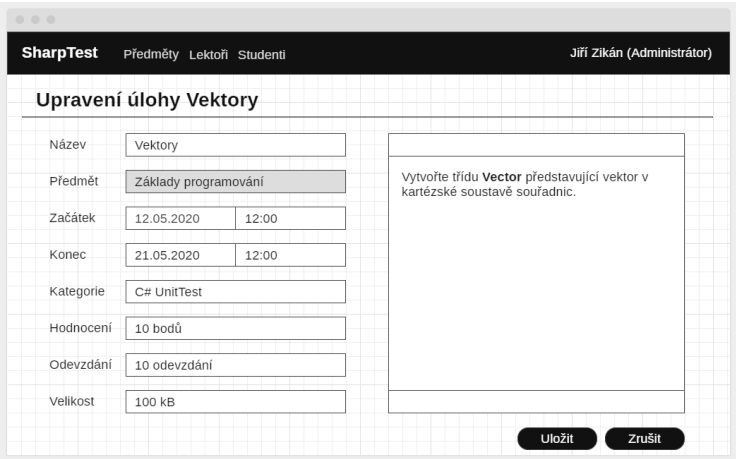
Obrázek 3.9: Detail úlohy – wireframe



3.3.7 Editor úlohy

Editor úlohy slouží k upravení údajů o vybrané úloze. Mezi ně patří název, datum a čas začátku a konce úlohy, přiřazená kategorie, počet získaných bodů, maximální možný počet odevzdání a maximální velikost souboru. Součástí je také pokročilý textový editor pro tvorbu popisu. Řešené případy užití – UC05.

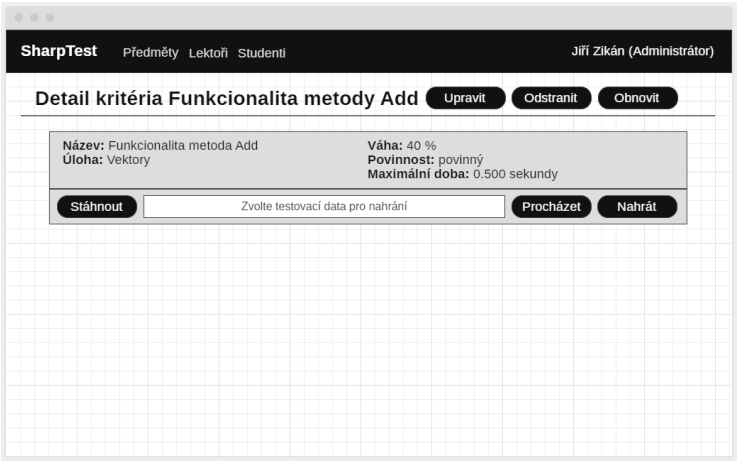
Obrázek 3.10: Editor úlohy – wireframe



3.3.8 Detail kritéria hodnocení

Detail kritéria hodnocení slouží k zobrazení veškerých podrobností o vybraném kritériu úlohy. Lektoři i administrátoři mohou kritérium hodnocení upravit či odstranit. Je zde obsažen také formulář pro stažení nebo nahrání testovacích dat. Řešené případy užití – UC06.

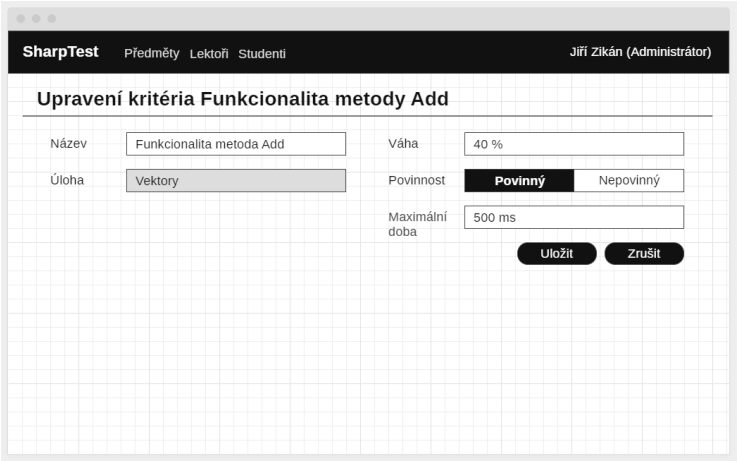
Obrázek 3.11: Detail kritéria hodnocení – wireframe



3.3.9 Editor kritéria hodnocení

Editor kritéria hodnocení slouží k upravení údajů o vybraném kritériu. Mezi ně patří název kritéria, váha určující jeho podíl na celkovém hodnocení, jeho povinnost a také případná maximální doba běhu testovacích dat. Řešené případy užití – UC06.

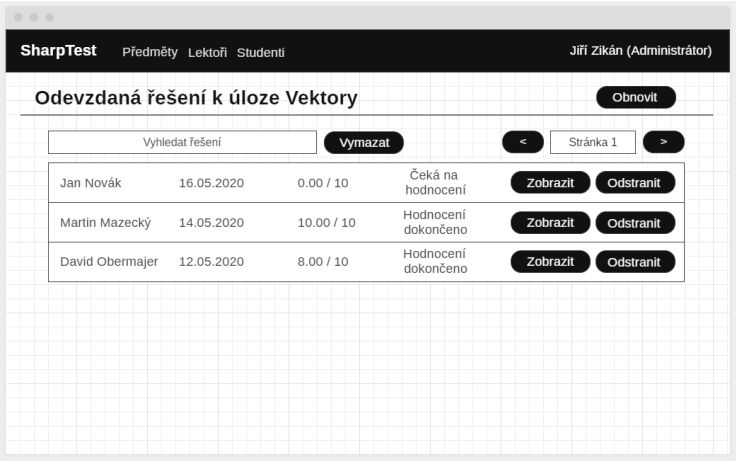
Obrázek 3.12: Editor kritéria hodnocení – wireframe



3.3.10 Seznam odevzdaných řešení

Seznam odevzdaných řešení slouží k zobrazení výčtu studenty odevzdaných řešení k vybrané úloze. Lektoři i administrátoři mohou seznam procházet, vyhledávat v něm, odstraňovat vybraná řešení nebo si zobrazovat jejich detail včetně možnosti ručního hodnocení. Řešené případy užití – UC09 a UC14.

Obrázek 3.13: Seznam odevzdaných řešení – wireframe



3.3.11 Detail odevzdaného řešení

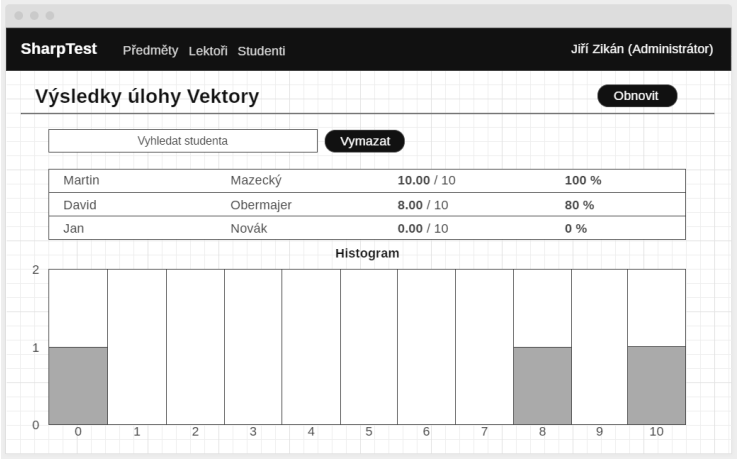
Detail odevzdaného řešení slouží k zobrazení podrobností o vybraném řešení úlohy. Lektoři i administrátoři si mohou odevzdané řešení stáhnout a zobrazit si jeho současné hodnocení. Dále mohou současné hodnocení odstranit a každé kritérium ohodnotit ručně. Řešené případy užití – UC11 a UC14.

Obrázek 3.14: Detail odevzdaného řešení – wireframe

3.3.12 Výsledky úlohy

Obrazovka s výsledky úlohy slouží k přehlednému zobrazení výsledků jednotlivých studentů v rámci dané úlohy. Kromě výčtu studentů s jejich dosaženým absolutním i relativním hodnocením obsahuje obrazovka také vyhledávací pole a histogram bodového zisku. Řešení případy užití – UC13.

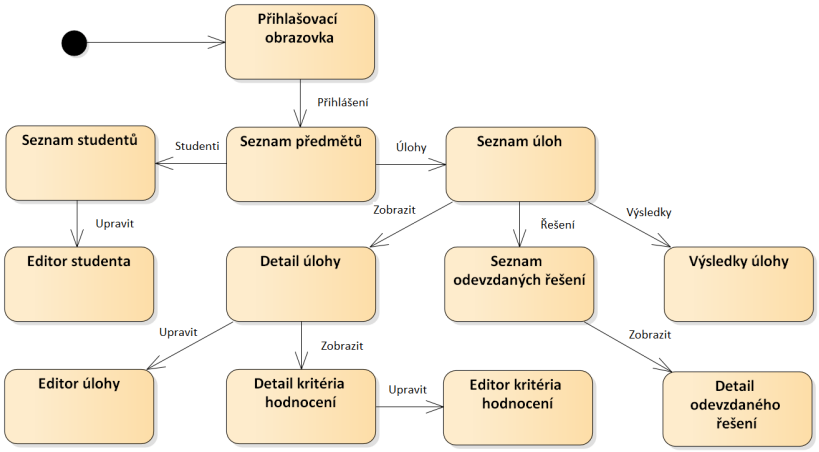
Obrázek 3.15: Výsledky úlohy – wireframe



3.3.13 Mapa obrazovek

Mapa obrazovek sestává z níže uvedeného stavového diagramu, který pro přehlednost zachycuje dopředné přechody mezi výše jmenovanými obrazovkami. K přechodu mezi obrazovkami dojde po kliknutí na příslušné navigační tlačítko, jehož název je u každého přechodu uveden.

Obrázek 3.16: Mapa obrazovek – stavový diagram



Kapitola 4

Realizace

Realizace je třetí fází životního cyklu softwarového produktu, v rámci které byla implementována serverová i klientská část administračního rozhraní tak, jak bylo analyzováno a navrženo v předcházejících kapitolách. Zdrojový kód obou implementovaných částí aplikace je nahrán na přiloženém CD. Snímky všech obrazovek realizované aplikace se nacházejí v příloze A. Obsahem samotné kapitoly je výčet použitých nástrojů a bližší popis implementačního procesu i detailů jednotlivých komponent, ze kterých obě části aplikace sestávají.

4.1 Použité nástroje

Hlavním vývojovým prostředím se během realizace obou částí aplikace stalo Visual Studio Code[3]. Pro správu jednotlivých verzí byl použit VCS systém Git[4] spolu s jeho fakultním rozšířením GitLab[5]. Jako správce JS balíčků a závislostí sloužil Node Package Manager[6]. Správa relační databáze MySQL byla řešena pomocí oficiálního nástroje MySQL Workbench[7]. K vývoji a testování aplikačního rozhraní REST byla využita aplikace Postman[8]. Stejně tak k vývoji a testování uživatelského rozhraní posloužily vývojářské nástroje integrované v prohlížeči Google Chrome[9]. Základní strukturovaná dokumentace a sada diagramů, z nichž jsou některé uvedeny i v předcházejících kapitolách, byla vytvořena pomocí nástroje Enterprise Architect od společnosti Sparx Systems[10].

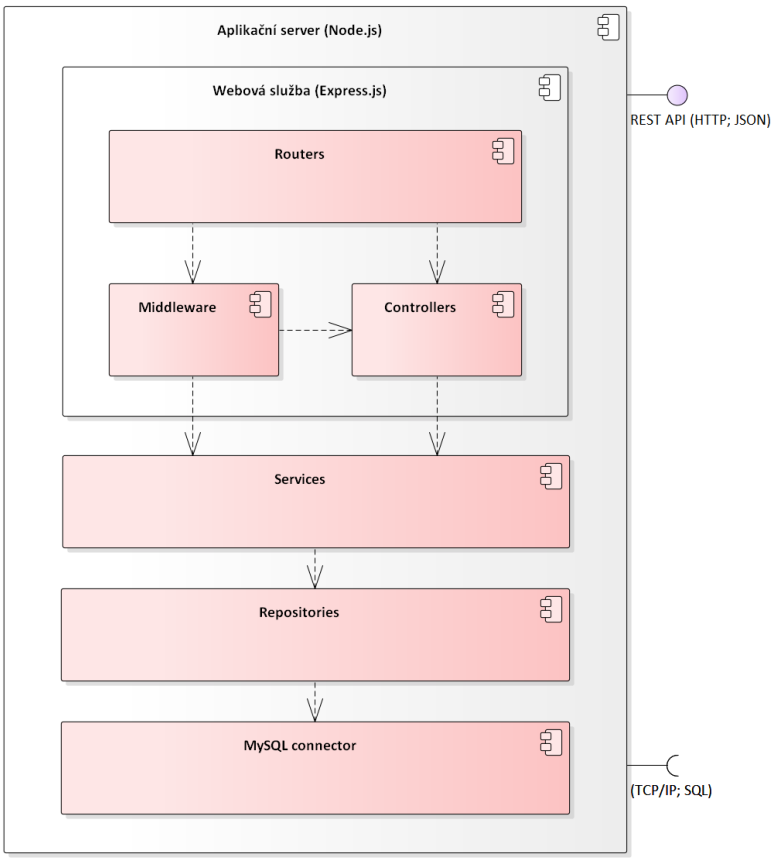
[3] https://code.visualstudio.com/
[4] https://git-scm.com/
[5] https://about.gitlab.com/
[6] https://www.npmjs.com/
[7] https://www.mysql.com/products/workbench/
[8] https://www.postman.com/
[9] https://developers.google.com/web/tools/chrome-devtools
[10] https://sparxsystems.com/

4.2 Realizace serverové části aplikace

Díky tomu, že bylo v předchozí kapitole definováno aplikační rozhraní REST oddělující obě části aplikace, by bylo teoreticky možné je bez problémů vyvíjet zároveň a nezávisle na sobě. Jelikož však na vývoji pracoval pouze autor práce, k takovému postupu nebyl žádný objektivní důvod. Jako první tedy byla realizována serverová část aplikace.

Funkcionalita serverové části aplikace je dobře fixována datovou vrstvou danou stávajícím systémem SharpTest a již zmíněným návrhem REST API. Na začátku realizace byla rozvržena struktura komponent, ze kterých serverová část aplikace sestává. Každá komponenta je tvořena sadou souvisejících tříd či metod majících zodpovědnost za jasně definovanou sadu činností. Jejich vzájemné závislosti ilustruje diagram komponent 4.1.

Obrázek 4.1: Serverová část aplikace – diagram komponent



Pomocí nástroje Node Package Manager byl založen nový projekt a vytvořen seznam závislostí. Dle uvedeného rozvržení komponent vznikla také odpovídající a níže popsaná adresářová struktura.

Obrázek 4.2: Serverová část aplikace – adresářová struktura

- `api` ..... komponenty RESTful webové služby
  - `controllers` ..... třídy komponenty `Controllers`
  - `middleware` ..... třídy komponenty `Middleware`
  - `routers` ..... definice komponenty `Routers`
- `certificates` ..... certifikáty využité pro HTTPS
- `database` ..... komponenty a konfigurace databáze
  - `repositories` ..... třídy komponenty `Repositories`
- `errors` ..... definice vlastních výjimek
- `node_modules` ..... balíčky a závislosti spravované NPM
- `services` ..... třídy komponenty `Services`
- `web` ..... soubory webového serveru
  - `public` ..... soubory poskytované webovým serverem

Následně byla vytvořena základní kostra serverové části aplikace výběrem jedné z entit, pro kterou byla implementována požadovaná funkcionalita REST API napříč všemi komponentami. Na tuto kostru byla dále nasazena autentizace a autorizace požadavků dle předcházející kapitoly. Zbytek implementace pak spočíval v postupném přidávání funkcionality pro všechny zbývající entity do doby, než poskytované REST API odpovídalo návrhu. Závěrem implementace serverové části aplikace bylo nastavení funkcionality webového serveru integrovaného do aplikačního rámce Express.js tak, aby byl připraven pro sdílení souborů klientské části aplikace.

Následující části této kapitoly se zabývají popisem implementačních detailů jednotlivých komponent serverové části aplikace.

4.2.1 MySQL connector

`MySQL connector` je základní komponenta serverové části aplikace umožňující interakci s databází MySQL. Řešena je s využitím NPM balíčku s názvem `mysql2`[11]. Pomocí konfiguračního souboru nebo proměnných prostředí jsou nastaveny údaje pro připojení do databáze. Komponenta si následně udržuje fond připojení, které dynamicky přiděluje pro zpracování zadaných dotazů.

4.2.2 Repositories

Komponenta `Repositories` je tvořena sadou tříd umožňujících vyhledávání, agregování, vkládání, upravování a odstraňování záznamů o jednotlivých entitách uložených v databázi. Každé databázové tabulce z entitně relačního diagramu 2.7 odpovídá právě jedna třída. Mezi zpracovávané záznamy se řadí také datové soubory řešení úloh a testovacích dat. Dotazy pro dané operace

[11] NPM repozitář: `https://www.npmjs.com/package/mysql2`.

jsou napsány v jazyce SQL. K jejich zpracování je využito rozhraní komponenty `MySQL connector`.

4.2.3 Services

Komponenta `Services` je tvořena sadou tříd zajišťujících aplikační logiku v podobě vynucení doménových omezení, validace vstupních dat a řízení přístupu k záznamům o jednotlivých entitách dle příslušných uživatelských rolí. Každé entitě z doménového modelu 2.3 odpovídá jedna třída. Dále je zde přítomna třída `AuthenticationService`, která dle návrhu implementuje přihlašování i odhlašování uživatelů a ověření JWT tokenu. Poslední třída nacházející se v této komponentě se nazývá `AccountService` a zajišťuje správu vlastního účtu. Pro práci s datovou vrstvou a uloženými daty je využito tříd z komponenty `Repositories`.

4.2.4 Controllers

Komponenta `Controllers` je tvořena sadou tříd zpracovávajících požadavky příchozí na vystavené REST API. Pro každý požadavek jmenovaný v návrhu REST API existuje samostatná metoda umístěná ve třídě, která svým názvem odpovídá první části cesty k datovému zdroji. Při zpracování požadavků je nejprve provedena syntaktická analýza těla požadavku a jeho parametrů. S jejich pomocí jsou zavolány příslušné metody z rozhraní komponenty `Services`, které vykonají požadované operace a vrátí výsledná data. Ta jsou nakonec zaslána jako odpověď na požadavek ve formátu JSON. V případě, že dojde během zpracování požadavku k chybě, je vrácen příslušný stavový kód.

4.2.5 Middleware

Komponenta `Middleware` obsahuje sadu tříd, které se starají o předzpracování požadavku před jeho předáním příslušné třídě komponenty `Controllers`. Třída `AuthenticationMiddleware` zajišťuje autentizaci uživatele dle zaslaného JWT tokenu a přiřazení jeho identity k dále předanému požadavku. Této identity dále využívá třída `AuthorizationMiddleware`, pomocí jejíchž metod může být vynucena požadovaná úroveň uživatelských oprávnění nutných pro zpracování zaslaného požadavku. V případě neplatného JWT tokenu nebo nedostatečné úrovně oprávnění je vrácen stavový kód 403.

4.2.6 Routers

Komponenta `Routers` je tvořená sadou továrních metod, které pro příslušnou třídu z komponenty `Controllers` vytvoří instanci třídy `Router`. Třída `Router` spadá pod aplikační rámec Express.js a pomocí jejích instancí je zajištěno směrování příchozích požadavků na odpovídající metody ze tříd komponenty `Controllers`. V rámci směrování může být před tyto metody předřazeno

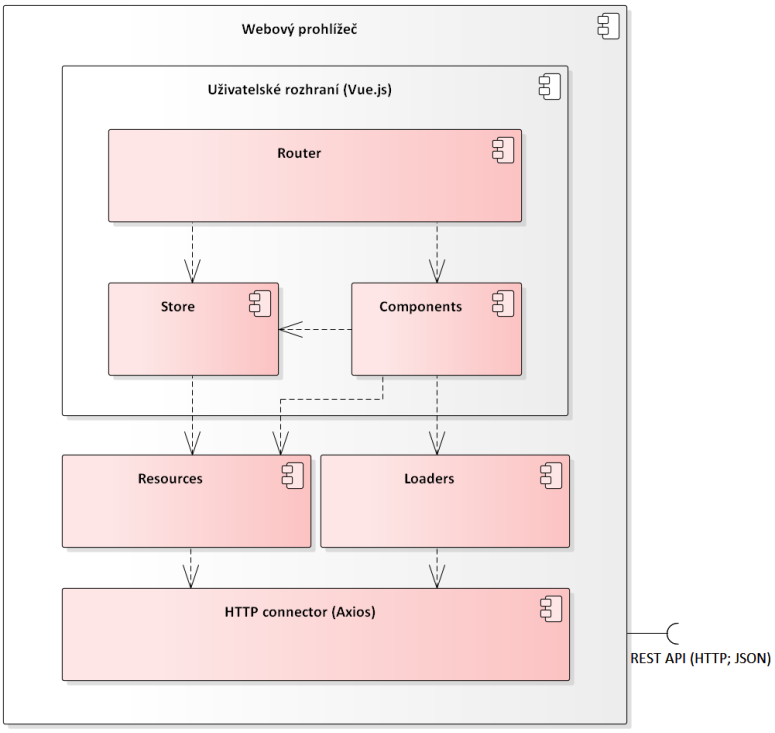
volání metod ze tříd komponenty `Middleware`, čímž je například vynuceno přihlášení uživatele nebo určitá úroveň uživatelských oprávnění nutných pro zpracování požadavku. Směrování je prováděno na základě URL adresy datového zdroje a zadané HTTP metody.

4.3 Realizace klientské části aplikace

Po úspěšné realizaci a otestování serverové části aplikace byla jako druhá realizována klientská část aplikace. Výhodou tohoto postupu byla možnost využít při vývoji jednotlivých funkcionalit uživatelského rozhraní skutečná data poskytovaná aplikačním rozhraním REST namísto pouhé simulace jeho funkce pomocí dat testovacích.

Funkcionalita klientské části aplikace byla z velké části definována již existujícím aplikačním rozhraním a podrobným návrhem klíčových obrazovek nacházejícím se v předchozí kapitole. Stejně jako v případě serverové části aplikace byla na začátku realizace rozvržena struktura komponent, ze kterých klientská část aplikace sestává. Závislosti komponent zachycuje diagram 4.3.

Obrázek 4.3: Klientská část aplikace – diagram komponent



V návaznosti na to byla vytvořena odpovídající adresářová struktura. Po jejím vytvoření byly do adresáře `libs` nahrány všechny knihovny a závislosti jmenované v návrhu aplikace.

Obrázek 4.4: Klientská část aplikace – adresářová struktura

Jako první byly realizovány všechny třídy komponenty `Resources` zprostředkovávající práci s datovými zdroji aplikačního rozhraní REST. Následně byl položen základ uživatelského rozhraní nasazením aplikačního rámce Vue.js včetně jeho rozšíření Vuex a Vue-router. Hlavní šablona uživatelského rozhraní byla rozvržena na část `NavigationComponent` představující hlavičku a navigaci webu, část `FooterComponent` představující patičku webu a takzvaný `RouterView` zastupující šablonu aktuálně načtené obrazovky. Aby nebylo nezbytně nutné načítat šablony všech obrazovek aplikace již před jejím prvním spuštěním, byla dále implementována třída `TemplateLoader` z komponenty `Loaders`, která umožnila načtení příslušných šablon až při přechodu na konkrétní obrazovku.

Z obrazovek byla nejdříve realizována přihlašovací obrazovka. Pro zajištění její funkcionality musel být implementován celý mechanismus autentizace a získání autorizačního tokenu. K uchování údajů o aktuálně přihlášeném uživateli včetně získaného JWT tokenu bylo využito komponenty `Store` a standardního uložiště `Local Storage`. Uložený JWT token byl provázán s odpovídající instancí komponenty `HTTP connector`, díky čemuž je automaticky přikládán ke všem požadavkům zasílaným přes komponentu `Resources`. Zbytek realizace spočíval v postupném přidávání šablon a implementaci funkcionality zbývajících obrazovek. Každá nová obrazovka byla vždy registrována v komponentě `Router` pod unikátní URL adresou a bylo na ni odkázáno z ostatních obrazovek dle mapy obrazovek 3.16 uvedené v návrhu.

Následující části této kapitoly se zabývají popisem implementačních detailů již zmíněných komponent klientské části aplikace.

4.3.1 HTTP connector

`HTTP connector` je komponenta umožňující komunikaci klientské části aplikace se serverovou částí aplikace prostřednictvím protokolu HTTP. Využita je jednak pro interakci s aplikačním rozhraním REST a dále také pro stahování šablon jednotlivých obrazovek. Realizována je pomocí knihovny Axios[12].

4.3.2 Resources

Komponenta `Resources` je tvořena sadou tříd zajišťujících komunikaci s aplikačním rozhraním REST. Jedná se v podstatě o protějšky ke třídám z komponenty `Controllers`, která je součástí serverové části aplikace. Pro komunikaci je využívána instance komponenty `HTTP connector`, jež je vázána na základní URL adresu REST API danou konfiguračním souborem. Jelikož je základem naprosté většiny těchto tříd zajištění funkcionality CRUD, je tato společná funkcionalita předepsána abstraktní třídou `BaseResource`.

4.3.3 Loaders

Komponenta `Loaders` je v tuto chvíli tvořena jedinou třídou `TemplateLoader` umožňující stahování HTML šablon jednotlivých obrazovek. Pro stahování je využívána samostatná instance komponenty `HTTP connector`, která je vázána na konfigurovatelnou URL adresu adresáře `templates`. Obsah tohoto adresáře je stejně jako ostatní soubory klientské části aplikace sdílen webovým serverem integrovaným v serverové části aplikace.

4.3.4 Store

`Store` je komponenta řešící správu sdíleného stavu mezi komponentami uživatelského rozhraní. Tvořena je instancí Vuex a čtyřmi továrními metodami. První tovární metoda `stateFactory` definuje seznam vlastností tvořících sdílený stav. Druhá tovární metoda `actionsFactory` definuje sadu metod, které s tímto sdíleným stavem pracují. Nedělají to však přímo, nýbrž protřednictvím takzvaných mutací definovaných tovární metodou `mutationsFactory`. Mutace jsou speciální metody určené k provedení jasně určené změny sdíleného stavu. Díky tomu mohou být změny stavu sledovány nebo zaznamenány. Poslední tovární metoda `gettersFactory` definuje sadu funkcí, které slouží k získání hodnot vypočtených z aktuálního stavu.

4.3.5 Components

Komponenta `Components` představuje jednotlivé Vue.js komponenty uživatelského rozhraní. Ty je možné rozdělit na skupinu komponent registrovaných lokálně a skupinu komponent registrovaných globálně.

[12]NPM repozitář: `https://www.npmjs.com/package/axios`.

Lokálně registrované komponenty jsou v tomto případě využity výhradně pro realizaci obrazovek klientské části aplikace. Každá obrazovka je mimo společné hlavičky a patičky tvořena právě jednou komponentou a právě jednou HTML šablonou. Komponenta je definována vlastní tovární metodou, uchovává v sobě aktuální stav obrazovky a implementuje logiku jejích uživatelských prvků. HTML šablona je ke konkrétní komponentě načtena až při přechodu na danou obrazovku.

Globálně registrované komponenty jsou využity pro implementaci a konfiguraci některých znovupoužitelných prvků, jako jsou například různé typy grafů nebo pokročilý textový editor.

4.3.6 Router

`Router` je komponenta umožňující přiřazení URL adresy k jednotlivým obrazovkám uživatelského rozhraní a následnou navigaci mezi nimi. Řešena je pomocí instance rozšiřující třídy Vue-router a jedné tovární metody. Ta definuje výčet všech obrazovek spolu s jejich názvem, URL adresou a hlavní komponentou. Výčet je doplněn také informací o nutnosti přihlášení uživatele nebo nutnosti administrátorských oprávnění pro přístup na danou obrazovku.

Kapitola 5

Testování

Testování je čtvrtou, nikoliv však poslední fází životního cyklu softwarového produktu. Kapitola se zabývá způsoby testování, kterým byla podrobena vyvíjená aplikace během své realizace i po jejím dokončení. Zmíněné testovací procesy sloužily nejen k odhalení případných chyb, ale také ověření, zdali byly splněny zadané požadavky a zdali bylo dosaženo požadované funkcionality. Součástí kapitoly je i shrnutí výsledků provedených testovacích scénářů a zhodnocení aplikace skutečným uživatelem zahrnující jeho návrhy na možná budoucí rozšíření.

5.1 Testování během vývoje

Postupné testování nově přidané funkcionality probíhalo již od počátku realizace. Během vývoje serverové části aplikace byly všechny databázové dotazy samostatně sestaveny, otestovány pomocí nástroje MySQL Workbench a teprve poté začleněny do kódu aplikace. Aplikační logika obsažená v komponentě `Services` byla testována pomocí jednotkových testů. Díky využití testovacího frameworku Jest[13] bylo možné provádět jednotkové testy automatizovaně a současně je využít i jako testy regresní. Funkcionalita vystaveného REST API byla testována sadou předem připravených požadavků prostřednictvím aplikace Postman. Tím bylo ověřeno, že aplikační rozhraní odpovídá navržené specifikaci pro všechny entity i typy požadavků.

Při vývoji klientské části aplikace bylo využíváno takzvaných smoke testů k rychlému ověření, zdali je aplikace jako celek spustitelná a zdali je uživatelské rozhraní schopné plnit své základní úkoly. Po dokončení každé obrazovky uživatelského rozhraní byla pečlivě otestována funkcionalita jednotlivých ovládacích prvků včetně validace a ošetření uživatelských vstupů. Případné problémy či chyby, které se během testování vyskytly, byly odladěny s pomocí vývojářských nástrojů integrovaných v prohlížeči Google Chrome.

[13]NPM repozitář: `https://www.npmjs.com/package/jest/`.

5.2 Testování podle scénářů

Po dokončení realizace byly sestaveny testovací scénáře, jejichž hlavním účelem je ověřit splnění funkčních požadavků a odhalit případné chyby či nedostatky, které by bránily použití realizované aplikace v zamýšleném rozsahu. Sestavení testovacích scénářů bylo provedeno na základě workflow modelu 2.2 popisujícího práci s celým systémem SharpTest. V úvahu byly brány i jednotlivé funkční požadavky, jejichž splnění je ověřováno.

Celkem byly sestaveny tři testovací scénáře pokrývající celou funkcionalitu administračního rozhraní. Jejich výčet spolu s názvem a stručným popisem je uveden níže. Plné znění testovacích scénářů se pak nachází v příloze B.

- **TS01 – Agenda administrátora** – scénář testuje přihlášení administrátora, správu lektorů a studentů, správu předmětů a přiřazení lektorů k předmětům. Testované funkční požadavky – F01, F02 a F03.
- **TS02 – Agenda lektora** – scénář testuje přihlášení lektora, přiřazení studentů k předmětům a správu úloh i kritérií jejich hodnocení. Testované funkční požadavky – F01, F04 a F05.
- **TS03 – Hodnocení a výsledky** – scénář testuje možnost zobrazení studentem odevzdaného řešení, jeho ruční hodnocení a zobrazení souhrnných výsledků. Testované funkční požadavky – F01, F06, F07 a F08.

Výsledky testování jednotlivých funkčních požadavků pomocí sestavených testovacích scénářů popisuje následující souhrnná tabulka. Pro ověření splnění funkčního požadavku F09 byly kroky testovacích scénářů TS02 a TS03 provedeny uživatelem v roli administrátora.

Tabulka 5.1: Splnění funkčních požadavků – tabulka výsledků testování

Funkční požadavek	Testováno pomocí scénáře	Výsledek
F01 – Přihlášení	TS01, TS02, TS03	splněno
F02 – Správa lektorů a studentů	TS01	splněno
F03 – Správa předmětů	TS01	splněno
F04 – Přiřazení studentů	TS02	splněno
F05 – Správa úloh a kritérií	TS02	splněno
F06 – Zobrazení řešení	TS03	splněno
F07 – Ruční hodnocení řešení	TS03	splněno
F08 – Zobrazení výsledků	TS03	splněno
F09 – Oprávnění administrátora	TS02, TS03 (admin)	splněno

Z výše uvedených výsledků testování pomocí testovacích scénářů jasně vyplývá, že všechny funkční požadavky byly úspěšně realizovány a při testování nebyly odhaleny žádné chyby.

5.3 Testování skutečným uživatelem

Posledním krokem testování, které bylo provedeno v rámci této práce, bylo testování administračního rozhraní jeho skutečným uživatelem. Tím se stal středoškolský pedagog vyučující programování na střední škole, která spolupracovala na předchozím vývoji systému SharpTest.

Během testování měl uživatel k dispozici pouze přihlašovací údaje k účtu lektora a aplikaci administračního rozhraní se základními testovacími daty. Testována tedy byla i intuitivnost ovládání uživatelského rozhraní, jelikož se v něm musel uživatel samostatně zorientovat.

Výsledek testování uživatelem je v celku pozitivní. Uživatel ocenil moderní a profesionální vzhled uživatelského rozhraní i přehlednou prezentaci výsledků včetně možnosti jejich vyhledávání. Dále ocenil také kvalitní zpracování ovládacích prvků, zvláště pak provedení pokročilejších ovládacích prvků, jako je kalendář nebo textový editor.

Jelikož aplikace obsahuje relativně velké množství funkcionalit, trvalo uživateli nějakou dobu, než si postupně prošel všechny obrazovky a v administračním rozhraní se zorientoval. I přesto tak byl schopen učinit samostatně, bez uživatelské příručky nebo jiné pomoci. Dle jeho názoru by s aplikací mohl po rychlém zaškolení pracovat i méně zkušený uživatel.

Uživatel po dokončení testování došel k závěru, že administrační rozhraní splňuje veškerou nezbytnou funkcionalitu. Aby mohla být jeho zpětná vazba využita také pro budoucí zlepšení aplikace, jmenoval níže uvedený seznam funkcionalit a rozšíření, jejichž přítomnost by do budoucna ocenil.

1. Umožnit hromadné přiřazení studentů k předmětům podle jejich třídy. Pro realizaci by bylo současně potřeba přidat informaci o třídě daného studenta, kterou v tuto chvíli systém neobsahuje.

2. Přidat k odevzdanému řešení pokročilý online editor, který by zobrazoval studenty odevzdaný kód, a to ideálně i ve více souborech. Odpadla by tak nutnost ručně stahovat a otevírat jednotlivá řešení.

3. Přidat tlačítko pro přechod na další neohodnocené řešení. V případě, kdy lektor ručně hodnotí větší množství neohodnocených řešení, by takové tlačítko umožnilo rychlejší přecházení mezi jednotlivými řešeními bez nutnosti vrátit se zpět na seznam řešení.

4. Přidat tlačítko pro kopírování úlohy do jiného předmětu. Předmět v systému je jednoroční položka. Pokud chce lektor použít stejnou úlohu i příští rok, musí její obsah ručně zkopírovat do nového předmětu.

5. Umožnit dočasné uzavření úlohy. Mohou nastat i situace, kdy je sice předem dáno datum a čas otevření i uzavření úlohy, ale v průběhu úlohy je nezbytné ji dočasně uzavřít. Bylo by vhodné přidat možnost úlohu uzavřít a následně znovu otevřít nehledě na aktuální čas.

Závěr

Cílem mé bakalářské práce bylo využít metod softwarového inženýrství a analyzovat, navrhnout a realizovat aplikaci administračního rozhraní, která bude integrována se stávajícím systémem SharpTest a umožní jeho efektivní správu. Důraz byl přitom kladen především na praktickou použitelnost výsledné aplikace a naplnění všech požadavků zadavatele.

Před začátkem vývoje administračního rozhraní byla provedena literární rešerše dostupných tuzemských i zahraničních zdrojů. Tato rešerše vymezila vybrané základní pojmy týkající se třívrstvých webových aplikací. Dále vyjmenovala a blíže popsala použité technologie, platformy, aplikační rámce a knihovny, který byly využity v průběhu realizace.

Navazující praktické části práce vycházely z jednotlivých fází životního cyklu vývoje softwarového produktu. Jako první byla provedena analýza veškerých východisek práce. Analyzovány byly funkční i obecné požadavky a uživatelské role, na základě kterých byl sestaven podrobný model případů užití. Dále byl v rámci analýzy procesů sestaven model workflow popisující postup práce s celým systémem SharpTest. Opomenuta nebyla ani klíčová analýza problémové domény zahrnující popis všech entit, jejich vlastností i vzájemných vztahů a omezení na ně kladených.

Mimo výše zmíněné skutečnosti se analytická část práce zaobírala taktéž architekturou stávajícího systému SharpTest. Popsána byla především její datová vrstva, která se měla stát součástí vyvíjené aplikace, a další specifika důležitá pro provedení zamýšlené integrace.

Na základě analyzovaných východisek byla v kapitole návrhu zvolena třívrstvá architektura a odůvodněna volba technologií pro aplikační a prezentační vrstvu. Návrh se dále zabýval specifikací, strukturou a bezpečností aplikačního rozhraní REST, které tyto dvě vrstvy odděluje. Důležitou součástí kapitoly byl také návrh přívětivého uživatelského rozhraní obsahující modely wireframe a jejich vazby na případy užití.

V předposlední kapitole práce byly jmenovány použité nástroje, byl popsán proces realizace a implementační detaily jednotlivých komponent ser-

verové části aplikace představující aplikační vrstvu a klientské části aplikace představující vrstvu prezentační. Serverová část aplikace byla implementována pomocí aplikačního rámce Express.js běžícího na platformě Node.js. Stejně tak byla klientská část aplikace založena na aplikačním rámci Vue.js. Pro zajištění responzivity uživatelského rozhraní bylo využito knihovny Bootstrap spolu s jejím rozšířením BootstrapVue.

Na závěr byly v poslední kapitole shrnuty metody testování využité k ověření, zdali byly splněny zadané požadavky a zdali bylo dosaženo požadované funkcionality. Součástí kapitoly byly také výsledky provedených testovacích scénářů a zhodnocení aplikace skutečným uživatelem včetně jeho návrhů na možná budoucí rozšíření.

Výsledkem bakalářské práce je produkčně využitelný software splňující všechny požadavky zadavatele. Na základě výše uvedených skutečností lze považovat cíl práce za splněný.

Literatura

1. MANAGEMENTMANIA.COM LLC. Webová aplikace (Web Application). In: *ManagementMania.com* [online]. 2018 [cit. 2020-06-03]. Dostupné z: `https://managementmania.com/cs/webova-aplikace-web-application`.
2. CONALLEN, Jim. *Building Web Applications with UML*. Boston: Pearson Education Incorporation, 2003. ISBN 978-0-201-73038-8.
3. BEDNARSKI, Wojciech. *Learning JavaScriptMVC*. Birmingham: Packt Publishing, 2013. ISBN 978-1-78216-020-5.
4. SCOTT, Emmit. *SPA Design and Architecture: Understanding Single Page Web Applications*. Shelter Island: Manning Publications, 2015. ISBN 978-1617292439.
5. SMITH, Steve. Choose Between Traditional Web Apps and Single Page Apps (SPAs). In: *Modern ASP.NET web applications e-book* [online]. Redmond: Microsoft Corporation, 2020 [cit. 2020-06-04]. Dostupné z: `https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps`.
6. WASSON, Mike. ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. In: *MSDN Magazine Issues* [online]. Redmond: Microsoft Corporation, 2013 [cit. 2020-06-03]. Dostupné z: `https://docs.microsoft.com/en-us/archive/msdn-magazine/2013/november/asp-net-single-page-applications-build-modern-responsive-web-apps-with-asp-net`.
7. FLANAGAN, David. *JavaScript: The Definitive Guide*. Sebastopol: O'Reilly Media, 2006. ISBN 978-0-596-10199-2.
8. VARMA, Vesudeva. *Software Architecture: A case based approach*. Delhi: Pearson Education India, 2009. ISBN 978-81-317-0749-4.

9. INGENO, Joseph. *Software Architect's Handbooks*. Birmingham: Packt Publishing, 2018. ISBN 978-1-78862-406-0.
10. TOMÁŠ, Bruckner; JIŘÍ, Voříšek; ALENA, Buchalcevová. *Tvorba informačních systémů: Principy, metodiky, architektury*. Praha: Grada Publishing, 2012. ISBN 978-80-247-4153-6.
11. ALONSO, Gustavo; CASATI, Fabio; KUNO, Harumi. *Web Services: Concepts, Architectures and Applications*. New York: Springer Publishing, 2004. ISBN 978-3-642-07888-0.
12. MANAGEMENTMANIA.COM LLC. Třívrstvá architektura (Three-tier architecture). In: *ManagementMania.com* [online]. 2015 [cit. 2020-05-24]. Dostupné z: `https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture`.
13. LACKO, Ľuboslav. *1001 tipů a triků pro SQL*. Brno: Computer Press, 2011. ISBN 978-80-251-3010-0.
14. OPPEL, Andy. *SQL bez předchozích znalostí: průvodce pro samouky*. Brno: Computer Press, 2012. ISBN 978-80-251-1707-1.
15. MONGODB INCORPORATED. What is NoSQL? In: *MongoDB Website* [online]. 2020 [cit. 2020-06-12]. Dostupné z: `https://www.mongodb.com/nosql-explained`.
16. OBASANJO, Dare. Denormalization, the NoSQL Movement and Digg. In: *Building scalable databases* [online]. 2009 [cit. 2020-06-12]. Dostupné z: `http://www.25hoursaday.com/weblog/2009/09/10/BuildingScalableDatabasesDenormalizationTheNoSQLMovementAndDigg.aspx`.
17. MAIER, Ronald; HÄDRICH, Thomas; PEINL, René. *Enterprise Knowledge Infrastructures*. Berlin: Springer Publishing, 2009. ISBN 978-3-540-89767-5.
18. MANAGEMENTMANIA.COM LLC. Aplikační server (Application server). In: *ManagementMania.com* [online]. 2017 [cit. 2020-06-01]. Dostupné z: `https://managementmania.com/cs/aplikacni-server-aps`.
19. JABLONSKI, Stefan; PETROV, Ilia; MEILER, Christian; MAYER, Udo. *Guide to Web Application and Platform Architectures*. New York: Springer Publishing, 2013. ISBN 978-3-642-05668-0.
20. PAPAZOGLOU, Michael P. *Web Services: Principles and Technology*. Harlow: Pearson Education Limited, 2008. ISBN 978-0-321-15555-9.
21. KOČUR, Vladimír. SOAP a REST služby. In: *NESS Czech Presentation* [online]. 2018 [cit. 2020-06-07]. Dostupné z: `https://www1.osu.cz/~zacek/infs2/SoapRest.pdf`.
22. GALITZ, Wilbert O. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. Indianapolis: Wiley Publishing, 2007. ISBN 978-0-470-05342-3.

23. CANZIBA, Elvis. *Hands-On UX Design for Developers: Design, prototype, and implement compelling user experiences from scratch.* Birmingham: Packt Publishing, 2018. ISBN 978-1-78862-669-9.

24. LAUESEN, Soren. *User Interface Design: A Software Engineering Perspective.* Harlow: Pearson Education Limited, 2005. ISBN 978-0-321-18143-5.

25. BABICH, Nick. Everything You Need To Know About Wireframes And Prototypes. In: *Adobe Blog* [online]. Adobe, 2017 [cit. 2020-06-06]. Dostupné z: `https://theblog.adobe.com/everything-you-need-to-know-about-wireframes-and-prototypes/`.

26. THORNSBY, Jessica. *Android UI Design.* Birmingham: Packt Publishing, 2016. ISBN 978-1-78588-742-0.

27. COSTA, Rebeca. What's the difference between wireframes and prototypes? In: *The complete guide to website wireframe design* [online]. Justinmind, 2019 [cit. 2020-06-06]. Dostupné z: `https://www.justinmind.com/blog/whats-the-difference-between-wireframes-and-prototypes/`.

28. CLARK, Jason A. *Responsive Web Design in Practice.* Maryland: Rowman a Littlefield, 2015. ISBN 978-1-4422-4368-2.

29. STATISTA INCORPORATED. Number of smartphones sold to end users worldwide from 2007 to 2020. In: *Statista.com Statistics* [online]. 2019 [cit. 2020-06-06]. Dostupné z: `https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/`.

30. ORACLE CORPORATION. What is MySQL? In: *MySQL 8.0 Reference Manual* [online]. 2020 [cit. 2020-05-26]. Dostupné z: `https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html`.

31. MAVRO, Pierre. *MariaDB High Performance.* Birmingham: Packt Publishing, 2014. ISBN 978-1-78398-160-1.

32. MARIADB CORPORATION. MariaDB versus MySQL: Compatibility. In: *MariaDB Knowledge Base* [online]. 2020 [cit. 2020-05-26]. Dostupné z: `https://mariadb.com/kb/en/mariadb-vs-mysql-compatibility/`.

33. DUBOIS, Paul. *MySQL Cookbook: Solutions for Database Developers and Administrators.* Sebastopol: O'Reilly Media, 2014. ISBN 978-1-449-37402-0.

34. SCHNEIDER, Robert D. *MySQL: oficiální průvodce tvorbou, správou a laděním databází.* Praha: Grada Publishing, 2006. ISBN 978-80-247-1516-2.

35. MASSE, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. Sebastopol: O'Reilly Media, 2011. ISBN 978-1-449-31050-9.

36. DOGLIO, Fernando. *REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development*. New York: Apress Media, 2018. ISBN 978-1-4842-3714-4.

37. JONES, Michael; BRADLEY, John; SAKIMURA, Nat. JSON Web Token (JWT). In: *Internet Engineering Task Force Request for Comments 7519* [online]. 2015 [cit. 2020-06-04]. ISSN 2070-1721. Dostupné z: `https://tools.ietf.org/html/rfc7519`.

38. AUTH0 INCORPORATED. Introduction to JSON Web Tokens. In: *JWT.IO* [online]. 2019 [cit. 2020-06-04]. Dostupné z: `https://jwt.io/`.

39. JONES, Michael; BRADLEY, John; SAKIMURA, Nat. JSON Web Signature (JWS). In: *Internet Engineering Task Force Request for Comments 7515* [online]. 2015 [cit. 2020-06-04]. ISSN 2070-1721. Dostupné z: `https://tools.ietf.org/html/rfc7515`.

40. MCLARTY, Rob. What is a JSON Web Token? In: *Rob McLarty Blog* [online]. 2020 [cit. 2020-06-04]. Dostupné z: `http://robmclarty.com/blog/what-is-a-json-web-token`.

41. HERRON, David. *Node.js Web Development: Server-side development with Node 10 made easy*. Birmingham: Packt Publishing, 2018. ISBN 978-1-78862-685-9.

42. OPENJS FOUNDATION. The V8 JavaScript Engine. In: *Nodejs.dev* [online]. 2020 [cit. 2020-06-01]. Dostupné z: `https://nodejs.dev/learn/the-v8-javascript-engine`.

43. OPENJS FOUNDATION. Differences between Node.js and the Browser. In: *Nodejs.dev* [online]. 2020 [cit. 2020-06-01]. Dostupné z: `https://nodejs.dev/learn/differences-between-nodejs-and-the-browser`.

44. OPENJS FOUNDATION. About Node.js®. In: *Nodejs.org* [online]. 2020 [cit. 2020-06-02]. Dostupné z: `https://nodejs.org/en/about/`.

45. OPENJS FOUNDATION. Introduction to Node.js. In: *Nodejs.dev* [online]. 2020 [cit. 2020-06-01]. Dostupné z: `https://nodejs.dev/learn/introduction-to-nodejs`.

46. SEN, Aritra. Node.js — Event Loop. In: *Geeks for Geeks* [online]. 2020 [cit. 2020-06-02]. Dostupné z: `https://www.geeksforgeeks.org/node-js-event-loop/`.

47. STRONGLOOP. Express: Node.js web application framework. In: *Express Home Page* [online]. 2017 [cit. 2020-06-01]. Dostupné z: `https://expressjs.com`.

48. MARDAN, Azat. *Express.js Guide: The Comprehensive Book on Express.js*. Scotts Valley: Createspace Independent Publishing Platform, 2013. ISBN 978-1494269272.

49. YOU, Evan. Introduction. In: *Vue.js Guide* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://vuejs.org/v2/guide/`.

50. FILIPOVA, Olga. *Learning Vue.js 2*. Birmingham: Packt Publishing, 2016. ISBN 978-1-78646-994-6.

51. GOOGLE LLC. Data Binding. In: *AngularJS Developer Guide* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://docs.angularjs.org/guide/databinding`.

52. YOU, Evan. What is Vuex? In: *Vuex Vue.js Home Page* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://vuex.vuejs.org/`.

53. YOU, Evan. Introduction. In: *Vue Router Vue.js Home Page* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://router.vuejs.org/`.

54. OTTO, Mark; THORNTON, Jacob. Build fast, responsive sites with Bootstrap. In: *Bootstrap Home Page* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://getbootstrap.com/`.

55. OTTO, Mark; THORNTON, Jacob. Reboot. In: *Bootstrap Documentation* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://getbootstrap.com/docs/4.5/content/reboot/`.

56. OTTO, Mark; THORNTON, Jacob. Grid system. In: *Bootstrap Documentation* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://getbootstrap.com/docs/4.5/layout/grid/`.

57. FILIPOVA, Olga. *Vue.js 2 and Bootstrap 4 Web Development*. Birmingham: Packt Publishing, 2017. ISBN 978-1-78829-092-0.

58. REGAN, Alex; MÜLLER, Jacob; SPOL. BootstrapVue. In: *BootstrapVue Home Page* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://bootstrap-vue.org/`.

59. FONTICONS INCORPORATED. Font Awesome. In: *Font Awesome Home Page* [online]. 2020 [cit. 2020-06-07]. Dostupné z: `https://fontawesome.com/`.

60. FACEBOOK INCORPORATED. Jest Home Page. In: *Jest Website* [online]. 2020 [cit. 2020-06-12]. Dostupné z: `https://jestjs.io/en/`.

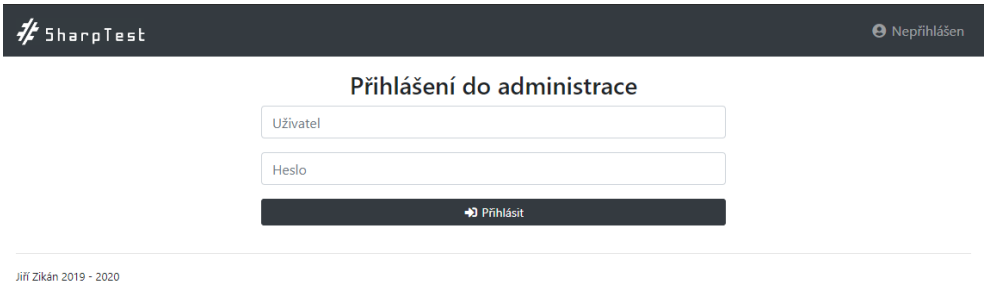
61. SOMMERVILLE, Ian. *Softwarové inženýrství*. Brno: Computer Press, 2013. ISBN 978-80-251-3826-7.

62. WIEGERS, Karl. *Požadavky na software: Od zadání k architektuře aplikace*. Brno: Computer Press, 2008. ISBN 978-80-251-1877-1.

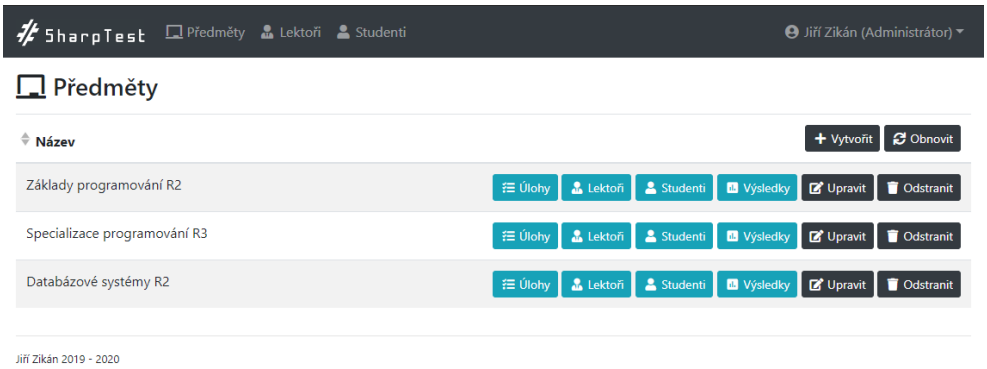
Příloha A

Obrazovky realizované aplikace

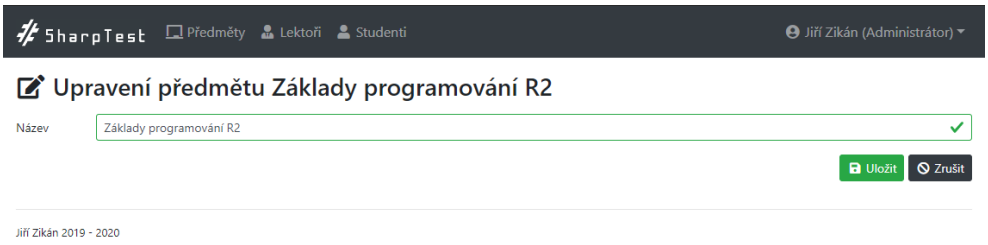
Obrázek A.1: Přihlašovací obrazovka – snímek obrazovky



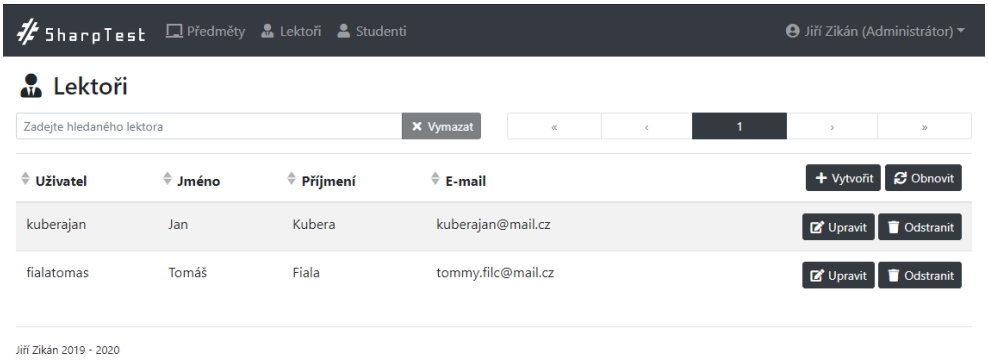
Obrázek A.2: Seznam studentů – snímek obrazovky



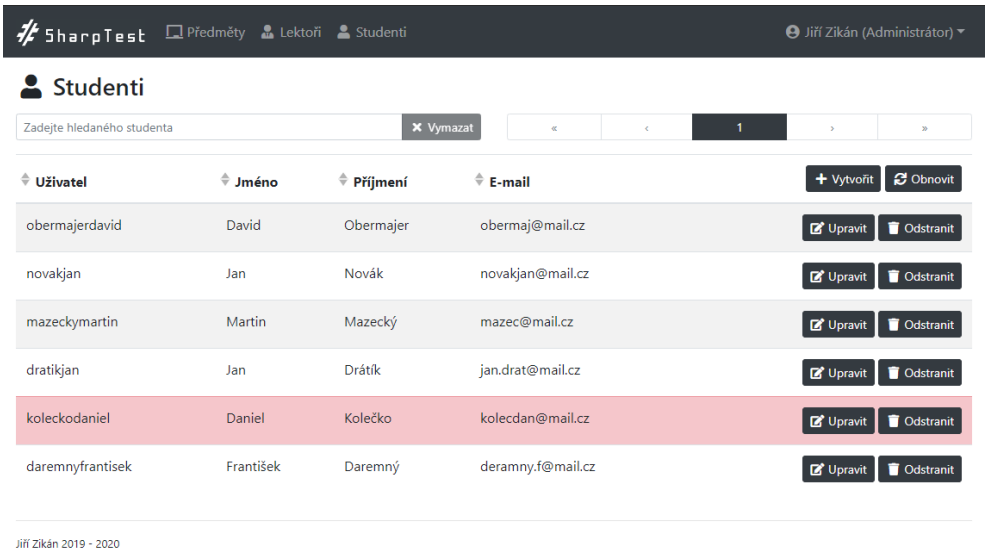
Obrázek A.3: Editor předmětu – snímek obrazovky



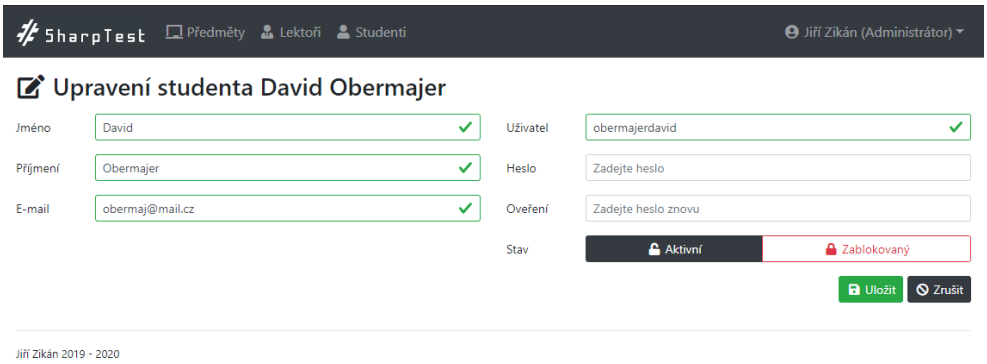
Obrázek A.4: Seznam lektorů – snímek obrazovky



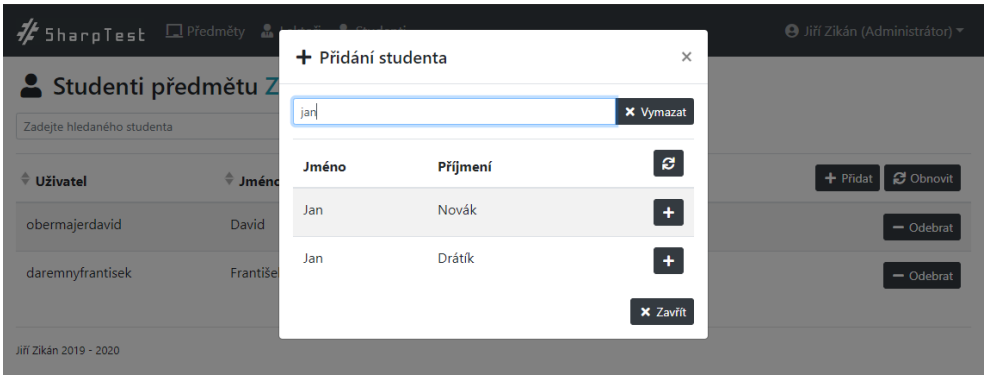
Obrázek A.5: Seznam studentů – snímek obrazovky



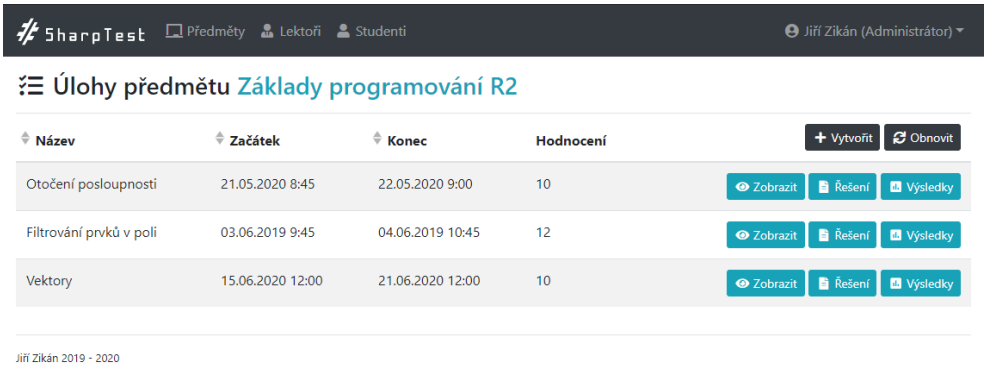
Obrázek A.6: Editor studenta – snímek obrazovky



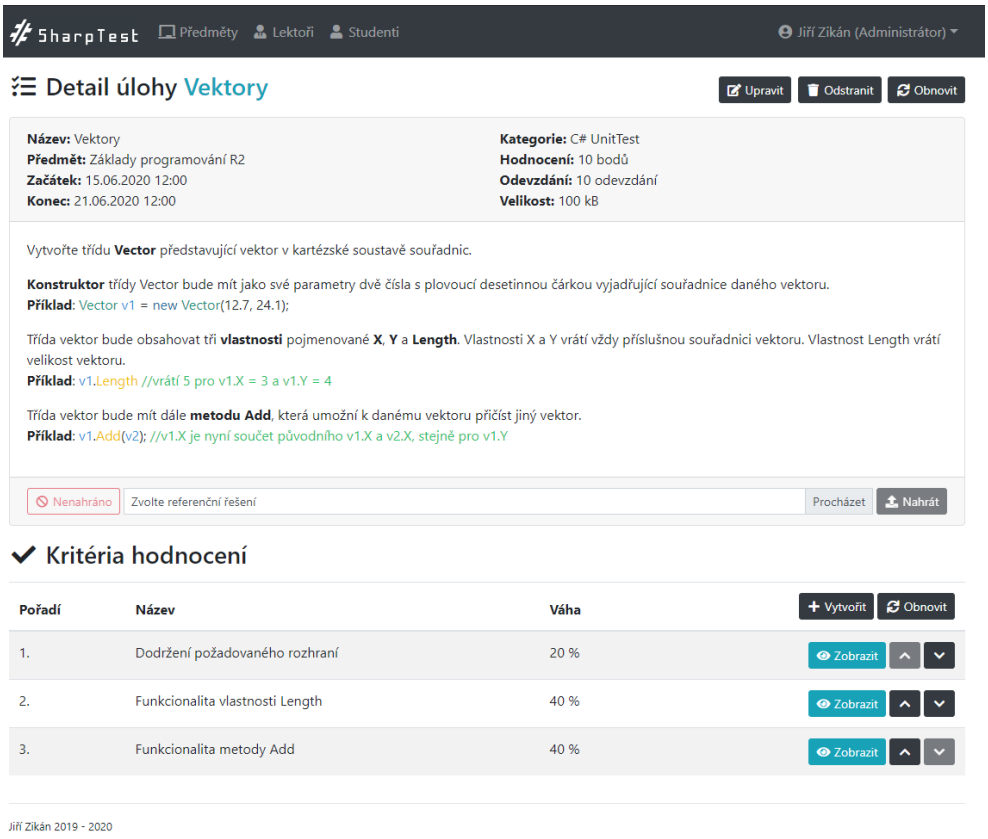
Obrázek A.7: Přiřazení studenta k předmětu – snímek obrazovky



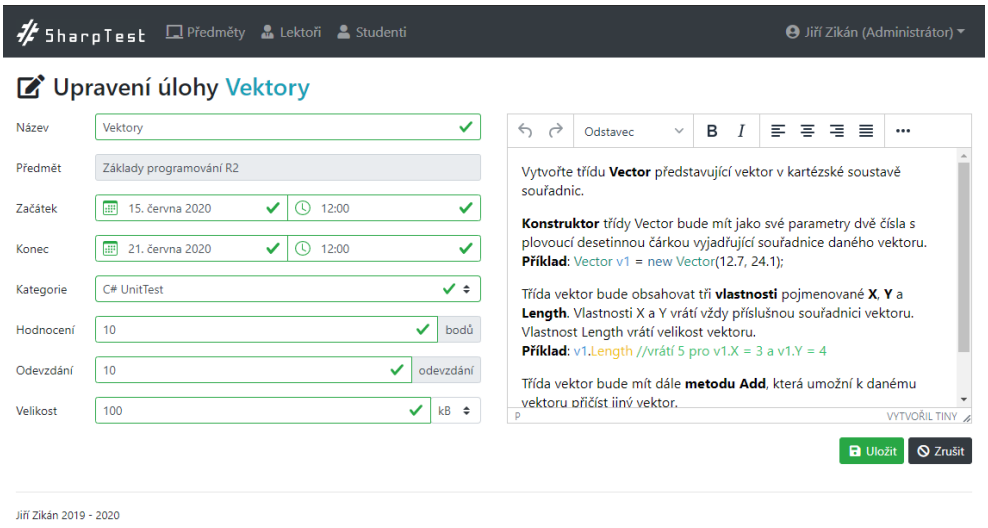
Obrázek A.8: Seznam úloh – snímek obrazovky



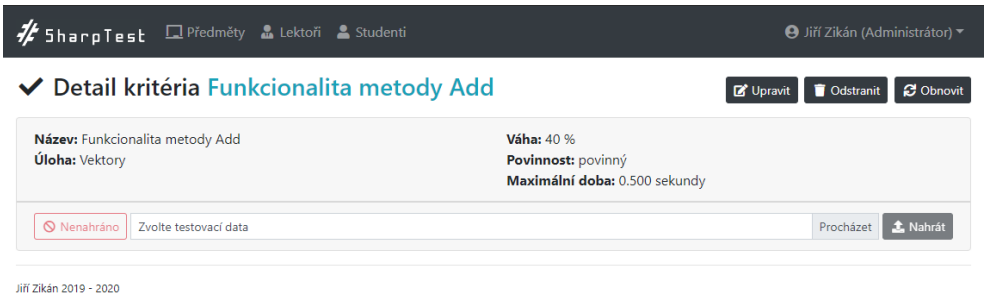
Obrázek A.9: Detail úlohy – snímek obrazovky



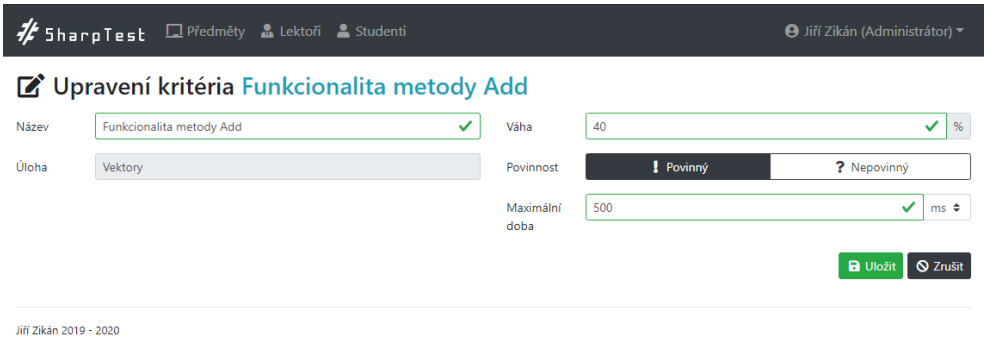
Obrázek A.10: Editor úlohy – snímek obrazovky



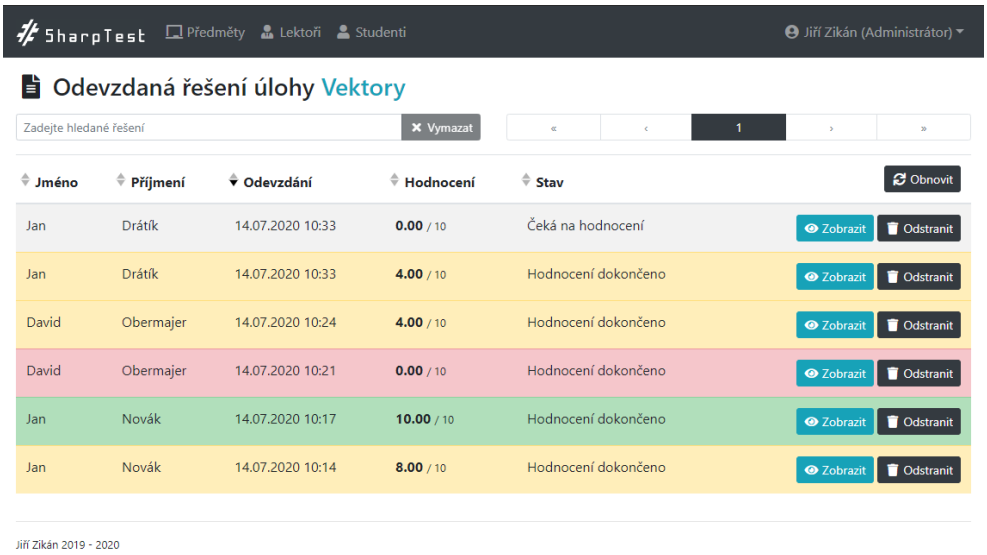
Obrázek A.11: Detail kritéria hodnocení – snímek obrazovky



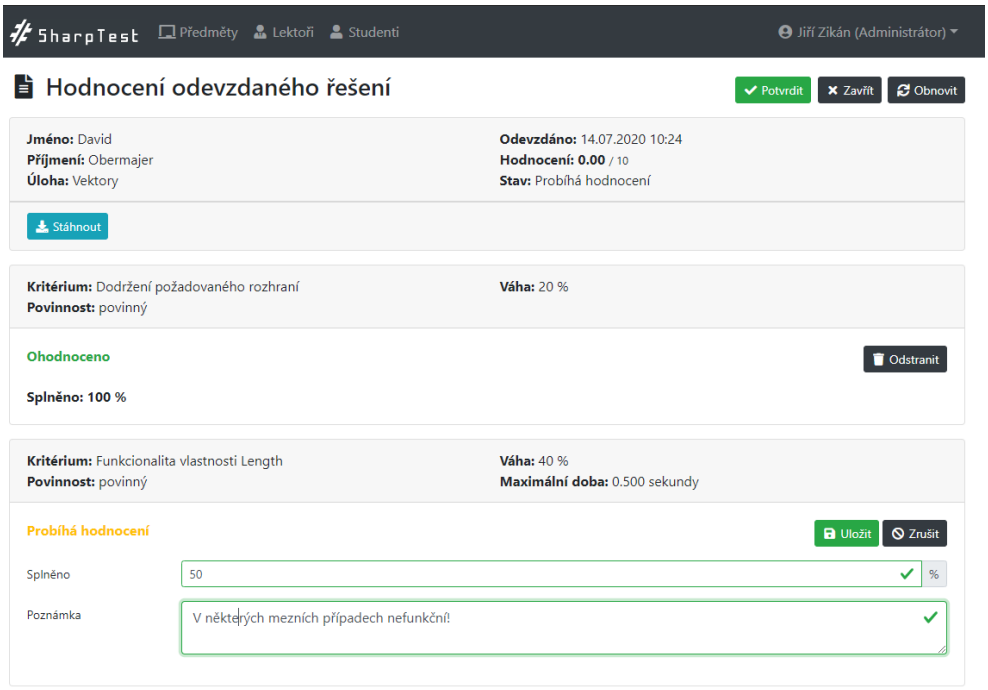
Obrázek A.12: Editor kritéria hodnocení – snímek obrazovky



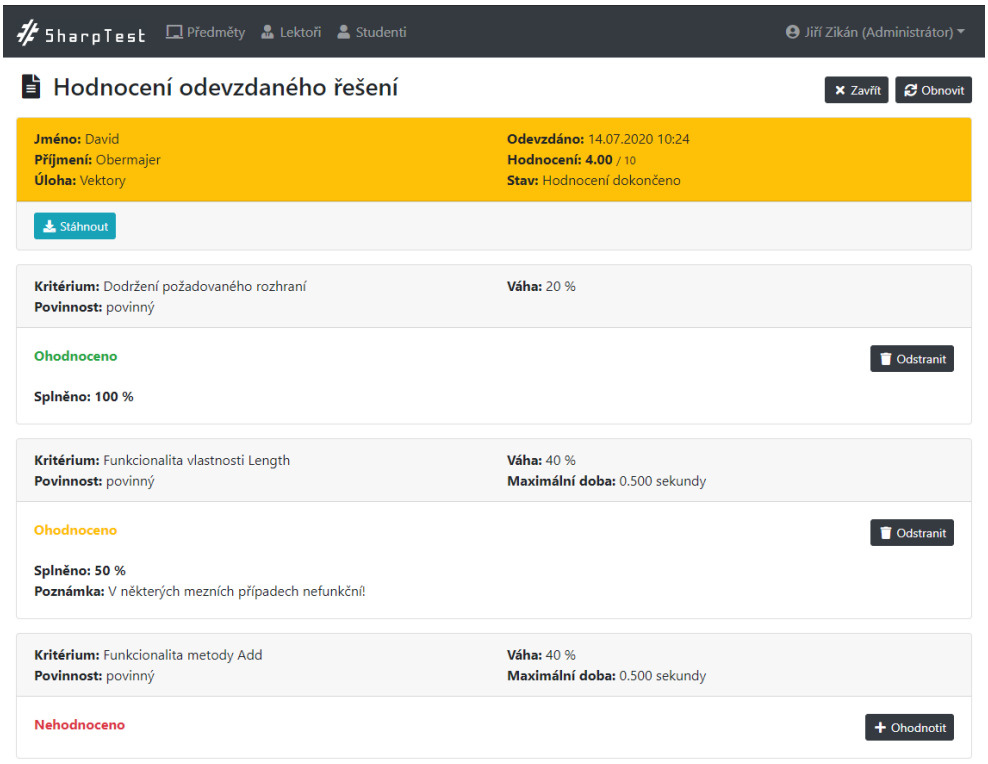
Obrázek A.13: Seznam odevzdaných řešení – snímek obrazovky



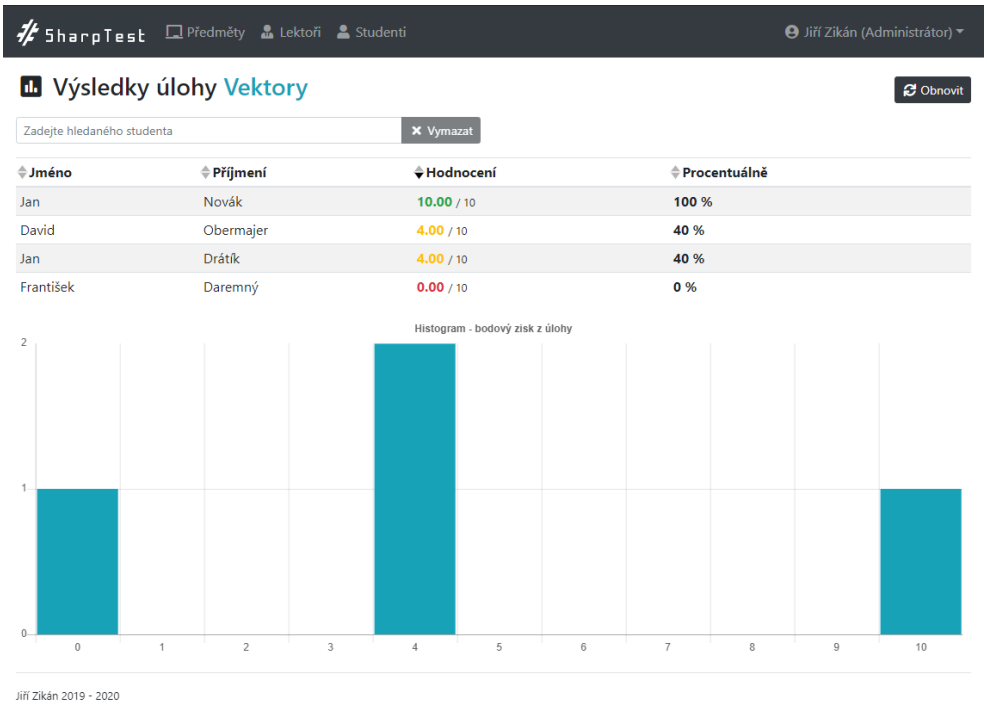
Obrázek A.14: Detail odevzdaného řešení – snímek obrazovky



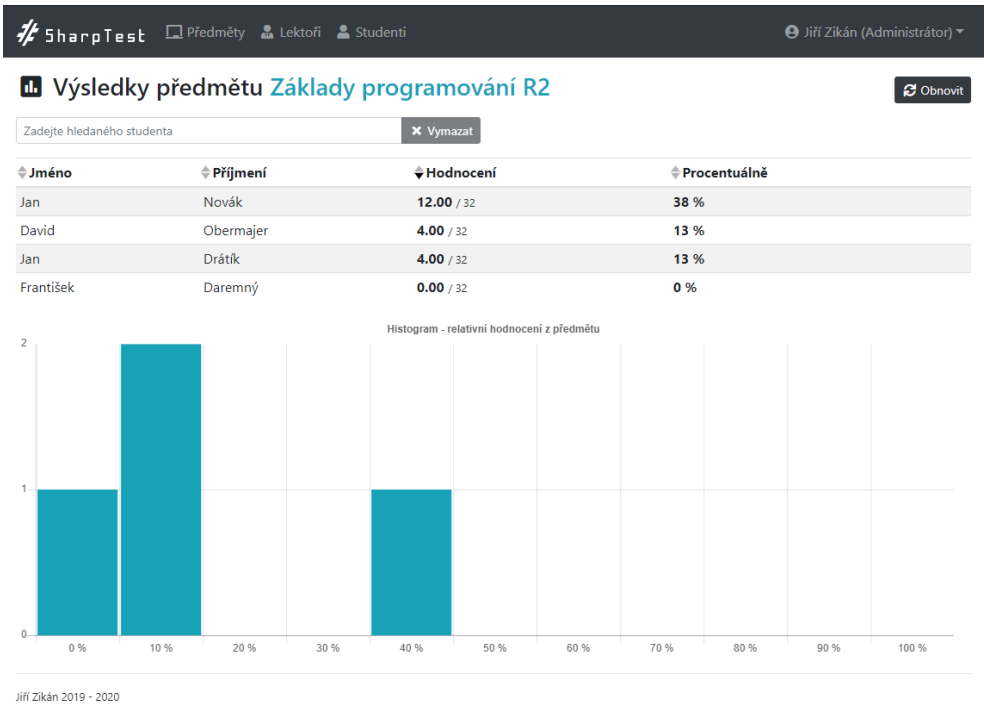
Obrázek A.15: Hodnocení odevzdaného řešení – snímek obrazovky



Obrázek A.16: Výsledky úlohy – snímek obrazovky



Obrázek A.17: Výsledky předmětu – snímek obrazovky



Příloha B

Testovací scénáře

Testovací scénáře obsažené v této příloze slouží k otestování funkcionality napříč administračním rozhraním a ke kontrole splnění funkčních požadavků. Současně však mohou jednotlivé kroky scénářů posloužit také k instruktáži budoucích uživatelů administračního rozhraní.

B.1 Agenda administrátora

Označení: TS01
Název: Agenda administrátora
Požadavky: F01, F02, F03
Popis: Scénář testuje přihlášení administrátora, správu lektorů a studentů, správu předmětů a přiřazení lektorů k předmětům.
Prerekvizity: Do systému SharpTest musí být přidán účet administrátora, jehož přihlašovací údaje jsou použity v tomto testu.
Kroky:

1. Přejděte na přihlašovací obrazovku, zadejte uživatelské jméno a heslo administrátora a klikněte na tlačítko Přihlásit. **Výstup:** Administrátor se úspěšně přihlásil a došlo k přesměrování na obrazovku se seznamem předmětů.

2. V horní navigaci nacházející se vedle loga SharpTest klikněte na tlačítko Lektoři. **Výstup:** Došlo k přesměrování na obrazovku se seznamem lektorů.

3. Klikněte na tlačítko Vytvořit nacházející se v pravém horním rohu tabulky se seznamem lektorů. **Výstup:** Došlo k otevření formuláře pro vytvoření účtu lektora.

4. Vyplňte požadované a validní údaje do textových polí Jméno, Příjmení a Uživatel. **Výstup:** Vyplněná pole jsou podbarvena zeleně. V případě,

kdy by bylo některé z polí podbarveno červeně, je nezbytné do něj zadat validní údaj.

5. Vyplňte heslo o minimální délce 6 znaků do pole Heslo a zopakujte zadání tohoto hesla do pole Ověření. **Výstup:** Vyplněná pole jsou podbarvena zeleně a došlo k odblokování tlačítka Vytvořit.

6. Klikněte na tlačítko Vytvořit nacházející se v pravém dolním rohu formuláře pro přidání lektora. **Výstup:** Došlo k uzavření formuláře a přesměrování na obrazovku se seznamem lektorů. V něm je viditelný (nebo alespoň dohledatelný) nově vytvořený lektor.

7. V horní navigaci klikněte na tlačítko Studenti. **Výstup:** Došlo k přesměrování na obrazovku se seznamem studentů.

8. Klikněte na tlačítko Vytvořit nacházející se v pravém horním rohu tabulky se seznamem studentů. **Výstup:** Došlo k otevření formuláře pro vytvoření účtu studenta.

9. Vyplňte požadované a validní údaje do textových polí Jméno, Příjmení a Uživatel. **Výstup:** Vyplněná pole jsou podbarvena zeleně.

10. Vyplňte heslo o minimální délce 6 znaků do pole Heslo a zopakujte zadání tohoto hesla do pole Ověření. **Výstup:** Vyplněná pole jsou podbarvena zeleně a došlo k odblokování tlačítka Vytvořit.

11. Klikněte na tlačítko Vytvořit nacházející se v pravém dolním rohu formuláře pro přidání studenta. **Výstup:** Došlo k uzavření formuláře a přesměrování na obrazovku se seznamem studentů. V něm je viditelný (nebo alespoň dohledatelný) nově vytvořený student.

12. V horní navigaci klikněte na tlačítko Předměty. **Výstup:** Došlo k přesměrování na obrazovku se seznamem předmětů.

13. Klikněte na tlačítko Vytvořit nacházející se v pravém horním rohu tabulky se seznamem předmětů. **Výstup:** Došlo k otevření formuláře pro vytvoření nového předmětu.

14. Vyplňte validní název předmětu do pole Název. **Výstup:** Pole Název je podbarveno zeleně. Došlo k odblokování tlačítka Vytvořit.

15. Klikněte na tlačítko Vytvořit nacházející se v pravém dolním rohu formuláře pro přidání předmětu. **Výstup:** Došlo k uzavření formuláře a přesměrování na obrazovku se seznamem předmětů. V něm je viditelný nově vytvořený předmět.

16. Klikněte na tlačítko Lektoři nacházející se na řádku u nově vytvořeného předmětu. **Výstup:** Došlo k přesměrování na obrazovku se seznamem lektorů přiřazených k novému předmětu. Tento seznam je zatím prázdný.

17. Klikněte na tlačítko Přidat nacházející se v pravém horním rohu tabulky se seznamem lektorů přiřazených k novému předmětu. **Výstup:** Došlo k otevření dialogového okna pro přidání lektora.

18. Do textového pole zadejte křestní jméno lektora, jehož účet byl vytvořen v rámci předcházejících kroků. **Výstup:** Zobrazila se tabulka se seznamem lektorů daného křestního jména, která obsahuje také nově vytvořeného lektora.

19. Klikněte na tlačítko se symbolem plus nacházející se na řádku s nově vytvořeným lektorem. **Výstup:** Nově vytvořený lektor byl přiřazen k novému předmětu a byl také skryt v rámci tabulky nacházející se v právě otevřeném dialogovém okně.

20. Klikněte na tlačítko Zavřít nacházející se v pravém dolním rohu dialogového okna. **Výstup:** Došlo k uzavření dialogového okna a v seznamu lektorů přiřazených k novému předmětu se nyní nachází nově vytvořený lektor.

21. Klikněte na jméno přihlášeného administrátora v pravém horním rohu obrazovky. **Výstup:** Zobrazí se seznam možností obsahující položky Nastavení a Odhlásit.

22. Klikněte na možnost Odhlásit z právě otevřeného seznamu. **Výstup:** Došlo k odhlášení administrátora a přesměrování na přihlašovací obrazovku.

B.2 Agenda lektora

Označení: TS02
Název: Agenda lektora
Požadavky: F01, F04, F05
Popis: Scénář testuje přihlášení lektora, přiřazení studentů k předmětům a správu úloh i kritérií jejich hodnocení.
Prerekvizity: Do systému SharpTest musí být přidán účet lektora, jehož přihlašovací údaje jsou použity v tomto testu. Dále musí být v systému přidán vybraný předmět, který je lektorem spravován. Nakonec se v systému musí nacházet také účet vybraného studenta, který zatím není přiřazen k vybranému předmětu.
Kroky:

1. Přejděte na přihlašovací obrazovku, zadejte uživatelské jméno a heslo lektora a klikněte na tlačítko Přihlásit. **Výstup:** Lektor se úspěšně přihlásil a došlo k přesměrování na obrazovku se seznamem předmětů.

2. Klikněte na tlačítko Studenti nacházející se na řádku u předem vybraného předmětu. **Výstup:** Došlo k přesměrování na obrazovku se seznamem studentů přiřazených k vybranému předmětu.

3. Klikněte na tlačítko Přidat nacházející se v pravém horním rohu tabulky se seznamem studentů přiřazených k vybranému předmětu. **Výstup:** Došlo k otevření dialogového okna pro přidání studenta.

4. Do textového pole zadejte křestní jméno předem vybraného studenta. **Výstup:** Zobrazila se tabulka se seznamem studentů daného křestního jména, která obsahuje také vybraného studenta.

5. Klikněte na tlačítko se symbolem plus nacházející se na řádku s vybraným studentem. **Výstup:** Vybraný student byl přiřazen k vybranému předmětu a byl také skryt v rámci tabulky nacházející se v právě otevřeném dialogovém okně.

6. Klikněte na tlačítko Zavřít nacházející se v pravém dolním rohu dialogového okna. **Výstup:** Došlo k uzavření dialogového okna a v seznamu studentů přiřazených k vybranému předmětu se nyní nachází vybraný student (nebo je alespoň dohledatelný).

7. V horní navigaci klikněte na tlačítko Předměty. **Výstup:** Došlo k přesměrování na obrazovku se seznamem předmětů.

8. Klikněte na tlačítko Úlohy nacházející se na řádku u předem vybraného předmětu. **Výstup:** Došlo k přesměrování na obrazovku se seznamem úloh, které náležejí vybranému předmětu.

9. Klikněte na tlačítko Vytvořit nacházející se v pravém horním rohu tabulky se seznamem úloh. **Výstup:** Došlo k otevření formuláře pro vytvoření nové úlohy.

10. Vyplňte validní název úlohy do pole Název. **Výstup:** Pole Název je podbarveno zeleně.

11. Vyplňte datum a čas začátku úlohy a datum a čas konce úlohy pomocí k tomu určených ovládacích prvků. **Výstup:** Ovládací prvky s datem a časem začátku a konce úlohy jsou podbarveny zeleně.

12. Klikněte na seznam Kategorie a zvolte v něm vhodnou kategorii úlohy. **Výstup:** Seznam Kategorie je podbarven zeleně. Do polí Odevzdání a Velikost se nakopírovaly výchozí hodnoty pro danou kategorii.

13. Vyplňte validní hodnotu vyjadřující maximální počet bodů, který lze za danou úlohu získat, do pole Hodnocení. **Výstup:** Pole Hodnocení je podbarveno zeleně. Došlo k odblokování tlačítka Vytvořit.

14. Klikněte na tlačítko Vytvořit nacházející se v pravém dolním rohu formuláře pro přidání úlohy. **Výstup:** Došlo k uzavření formuláře a přesměrování na obrazovku se seznamem úloh. V něm je viditelná nově vytvořená úloha.

15. Klikněte na tlačítko Zobrazit nacházející se na řádku u nově vytvořené úlohy. **Výstup:** Došlo k přesměrování na obrazovku s detailem úlohy.

16. Klikněte na tlačítko Vytvořit nacházející se v pravém horním rohu tabulky se seznamem kritérií hodnocení. **Výstup:** Došlo k otevření formuláře pro vytvoření nového kritéria hodnocení.

17. Vyplňte požadované a validní údaje do polí Název a Váha. **Výstup:** Vyplněná pole jsou podbarvena zeleně. V případě, kdy by bylo některé z polí podbarveno červeně, je nezbytné do něj zadat validní údaj.

18. Pomocí přepínače Povinnost zvolte, zdali se jedná o povinné kritérium či nikoliv. **Výstup:** Zvolená možnost je zvýrazněna a šedě podbarvena.

19. Klikněte na tlačítko Vytvořit nacházející se v pravém dolním rohu formuláře pro přidání kritéria hodnocení. **Výstup:** Došlo k uzavření formuláře a přesměrování na obrazovku s detailem úlohy. V seznamu kritérií je viditelné nově vytvořené kritérium.

20. Klikněte na jméno přihlášeného lektora v pravém horním rohu obrazovky. **Výstup:** Zobrazí se seznam možností obsahující položky Nastavení a Odhlásit.

21. Klikněte na možnost Odhlásit z právě otevřeného seznamu. **Výstup:** Došlo k odhlášení lektora a přesměrování na přihlašovací obrazovku.

B.3 Hodnocení a výsledky

Označení: TS03
Název: Hodnocení a výsledky
Požadavky: F01, F06, F07, F08
Popis: Scénář testuje možnost zobrazení studentem odevzdaného řešení, jeho ruční hodnocení a zobrazení souhrnných výsledků.
Prerekvizity: Do systému SharpTest musí být přidán účet lektora, jehož přihlašovací údaje jsou použity v tomto testu. Dále musí systém obsahovat alespoň jeden vybraný předmět spravovaný lektorem. Vybraný předmět musí obsahovat právě jednu úlohu včetně jednoho kritéria hodnocení. Nakonec musí být v systému přidán vybraný student, který je přiřazen k vybranému předmětu a jako jediný odevzdal řešení k úloze nacházející se v tomto předmětu.
Kroky:

1. Přejděte na přihlašovací obrazovku, zadejte uživatelské jméno a heslo lektora a klikněte na tlačítko Přihlásit. **Výstup:** Lektor se úspěšně přihlásil a došlo k přesměrování na obrazovku se seznamem předmětů.

2. Klikněte na tlačítko Úlohy nacházející se na řádku u předem vybraného předmětu. **Výstup:** Došlo k přesměrování na obrazovku se seznamem úloh, které náležejí vybranému předmětu.

3. Klikněte na tlačítko Řešení nacházející se na řádku u vybrané úlohy. **Výstup:** Došlo k přesměrování na obrazovku se seznamem odevzdaných řešení.

4. Pomocí vyhledávacího řádku vyhledejte řešení studenta odevzdané k vybrané úloze. Zadejte postupně jeho křestní jméno a případně i příjmení. **Výstup:** V tabulce se seznamem odevzdaných řešení je viditelné řešení odevzdané vybraným studentem.

5. Klikněte na tlačítko Zobrazit nacházející se na řádku u řešení odevzdaného vybraným studentem. **Výstup:** Došlo k přesměrování na obrazovku s detailem odevzdaného řešení. U kritéria hodnocení je uveden stav Nehodnoceno.

6. Klikněte na tlačítko Stáhnout nacházející se v levém dolním rohu prvního boxu. **Výstup:** Zahájilo se stahování ZIP archivu se studentem odevzdaným řešením.

7. Klikněte na tlačítko Ohodnotit nacházející se na pravé straně boxu s kritériem hodnocení. **Výstup:** Došlo k zobrazení formuláře pro ruční hodnocení daného kritéria. Stav kritéria se změnil na Probíhá hodnocení.

8. Vyplňte odpovídající a validní hodnoty do polí Splněno a Poznámka. **Výstup:** Obě pole jsou podbarvena zeleně a zároveň došlo k odblokování tlačítka Uložit.

9. Klikněte na tlačítko Uložit nacházející se v pravém horním rohu formuláře pro ruční hodnocení daného kritéria. **Výstup:** Došlo k uložení hodnocení, uzavření formuláře pro ruční hodnocení a stav kritéria se změnil na Ohodnoceno.

10. Klikněte na tlačítko Potvrdit nacházející se v pravém horním rohu obrazovky. **Výstup:** Došlo k dokončení a uzavření hodnocení. Stav řešení se změnil na Hodnocení dokončeno a horní box s detaily řešení se dle celkového hodnocení podbarvil odpovídající barvou.

11. Klikněte na tlačítko Zavřít nacházející se v pravém horním rohu obrazovky. **Výstup:** Došlo k zavření detailu odevzdání a přesměrování na obrazovku se seznamem odevzdaných řešení.

12. Klikněte na modře zvýrazněný název úlohy v nadpisu obrazovky. **Výstup:** Došlo k přesměrování na seznam úloh.

13. Klikněte na tlačítko Výsledky nacházející se na řádku u vybrané úlohy. **Výstup:** Došlo k přesměrování na obrazovku se souhrnnými výsledky dané úlohy. Ve výsledcích je uveden student, jehož odevzdané řešení bylo ohodnoceno v předcházejících krocích. Uvedené hodnocení odpovídá skutečnosti a je správně zaneseno do histogramu.

14. Klikněte na modře zvýrazněný název úlohy v nadpisu obrazovky. **Výstup:** Došlo k přesměrování na seznam úloh.

15. Klikněte na modře zvýrazněný název předmětu v nadpisu obrazovky. **Výstup:** Došlo k přesměrování na seznam předmětů.

16. Klikněte na tlačítko Výsledky nacházející se na řádku u vybraného předmětu. **Výstup:** Došlo k přesměrování na obrazovku se souhrnnými výsledky daného předmětu. Ve výsledcích je uveden student, jehož odevzdané řešení bylo ohodnoceno v předcházejících krocích. Uvedené hodnocení odpovídá skutečnosti a je správně zaneseno do histogramu.

17. Klikněte na jméno přihlášeného lektora v pravém horním rohu obrazovky. **Výstup:** Zobrazí se seznam možností obsahující položky Nastavení a Odhlásit.

18. Klikněte na možnost Odhlásit z právě otevřeného seznamu. **Výstup:** Došlo k odhlášení lektora a přesměrování na přihlašovací obrazovku.

Příloha C

Instalační manuál

Instalační manuál popisuje postup zprovoznění realizované aplikace administračního rozhraní včetně nastavení prostředí nutného pro její běh a importu testovacích dat. I přesto, že je zde uvedený postup jmenovitě určen pro provedení na čisté instalaci operačního systému Fedora 32 Workstation[14], jsou popsané principy nasazení platné i pro jiné distribuce operačního systému Linux, případně Microsoft Windows. V takovém případě je však nutné, aby čtenář přizpůsobil prováděné operace zvolenému operačnímu systému. V případě starších verzí operačních systémů může být postup náročnější kvůli nutnosti instalace novějších verzí MariaDB (10.4.13), NPM (6.14.4) a Node.js (12.16.3). Pro instalaci je nezbytné připojení k internetu.

Před instalací je nezbytné nalézt adresář `exe` nacházející se na přiloženém CD a zkopírovat jej do vybraného umístění na lokálním disku počítače. Tento adresář obsahuje spustitelnou verzi aplikace administračního rozhraní včetně databázových skriptů a testovacích dat. Instalace předpokládá právo použití administrátorských oprávnění pomocí příkazu `sudo`.

Prvním krokem instalace je otevření terminálového okna a přepnutí se do zkopírovaného adresáře `exe`.

```
cd exe # přepnutí do adresáře exe
```

Druhým krokem je instalace a následné spuštění databázového serveru relační databáze MariaDB.

```
sudo dnf install mariadb-server -y # instalace mariadb-server
sudo systemctl start mariadb # spuštení systémové služby
```

Po úspěšné instalaci a spuštění MariaDB následuje použití utility `mysql` ke spuštění tří databázových skriptů nacházejících se v adresáři `sql`. Ty vytvoří schéma databáze `sharptest`, importují testovací data a nakonec změní heslo uživatele `root` na `Abc123456`.

[14] Lze použít i Fedora 32 Workstation Live DVD bez nutnosti instalace OS.

```
sudo mysql -u root < sql/creation_script.sql # vytvoření DB
sudo mysql -u root < sql/data_insertion_script.sql # import dat
sudo mysql -u root < sql/root_password.sql # změna root hesla
```

Dále je třeba nainstalovat platformu Node.js včetně správce balíčků Node Package Manager.

```
sudo dnf install nodejs -y # instalace Node.js
sudo dnf install npm -y # instalace NPM
```

Samotné spuštění aplikace administračního rozhraní se provede pomocí následujícího příkazu.

```
npm start # spuštění aplikace administračního rozhraní
```

Po úspěšném spuštění aplikace stačí ponechat otevřené terminálové okno, otevřít webový prohlížeč (například Mozilla Firefox 75.0) a zadat do adresní řádky URL adresu administračního rozhraní.

```
URL adresa: http://localhost:8080/web
```

Pro přihlášení do administračního rozhraní jsou v testovacím prostředí předem připraveny dva uživatelské účty. První z nich je účet `zikanjiri` v roli administrátora a druhý je účet `kuberajan` v roli lektora.

```
Přihlašovací údaje:

Administrátor:
    Uživatel: zikanjiri
    Heslo: Abc123456

Lektor:
    Uživatel: kuberajan
    Heslo: Abc123456
```

Příloha D

Seznam použitých zkratek

AJAX Asynchronous JavaScript And XML

ANSI American National Standards Institute

API Application Programming Interface

BLOB Binary Large Object

CD Compact Disc

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

DOM Document Object Model

GNU GNU's Not Unix

GPL General Public License

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IP Internet Protocol

ISO International Organization for Standardization

JS JavaScript

JSON JavaScript Object Notation

JWT JSON Web Token

MPA Multi-page application

NPM Node Package Manager

RDBMS Relational Database Management System

REST Representational State Transfer

RFC Request For Comments

RPC Remote Procedure Call

RWD Responsive Web Design

SDLC Software Development Life Cycle

SOAP Simple Object Access Protocol

SPA Single-page application

SQL Structured Query Language

TCP Transmission Control Protocol

UI User Interface

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

UX User Experience

VCS Version Control System

XML Extensible Markup Language

Příloha **E**

Obsah přiloženého CD

Obrázek E.1: Obsah přiloženého CD – adresářová struktura

- `readme.txt` ..... popis obsahu CD
- `exe` ..... spustitelná forma realizované aplikace
- `src` ..... zdrojové kódy
  - `application` ..... zdrojové kódy realizované aplikace
    - `server` ..... zdrojové kódy serverové části aplikace
    - `client` ..... zdrojové kódy klientské části aplikace
  - `database` ..... zdrojové kódy databázových skriptů
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `txt` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF