Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Systém pro zadávání a odevzdávání úkolů

2023

Matyáš Hroch

Vedoucí práce:
Mgr. Radek Janoštík, Ph.D.

Studijní program: Informatika,
Specializace: Programování a vývoj software

Bibliografické údaje

Autor:	Matyáš Hroch
Název práce:	Systém pro zadávání a odevzdávání úkolů
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní program:	Informatika, Specializace: Programování a vývoj software
Vedoucí práce:	Mgr. Radek Janoštík, Ph.D.
Počet stran:	42
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Matyáš Hroch
Title:	System for creating and submitting assignments
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study program:	Computer Science, Specialization: Programming and Software Development
Supervisor:	Mgr. Radek Janoštík, Ph.D.
Page count:	42
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Tento text popisuje postup při vývoji webové aplikace pro zadávání a odevzdávání úkolů. Obsahuje porovnání již existujících řešení, stanovení konkrétních požadavků a uživatelskou dokumentaci a technický popis aplikace. Dále se věnuje možným rozšířením takto vzniklé aplikace.

Synopsis

This text describes the process of developing a web application for task management, including task assignment and submission. It includes a comparison of existing solutions, identification of specific requirements, as well as programming and user documentation. It also explores potential extensions for the developed application.

Klíčová slova: Node.js; Vue.js; JavaScript; webová aplikace; odevzdávání úkolů

Keywords: Node.js; Vue.js; JavaScript; web application; submitting assignments

Děkuji vedoucímu práce Mgr. Radku Janoštíkovi, Ph.D. za cenné rady a ještě cennější optimismus v celém průběhu vytváření práce. Stejně tak děkuji RNDr. Martinu Trnečkovi, Ph.D. za konzultace ohledně webových technologií. Dále bych rád poděkoval rodině a přítelkyni za podporu a kontrolu textu. Také děkuji svým kamarádům a kolegům za testování aplikace. Jmenovitě děkuji kolegovi Petru Špirkovi za rady ohledně nasazení testovací verze a poskytnutí domény.

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Programování je bezpochyby náročná činnost a pro její osvojení je důležitá praxe. Odtud plyne nutnost učit se na praktických úkolech. Na Katedře informatiky na Přírodovědecké fakultě Univerzity Palackého v Olomouci (dále pouze jako „naše katedra“) je poslední dobou trend takový, že pro získání zápočtu z některého programovacího jazyka je nutné splnit určitý počet úkolů. Tyto úkoly se odevzdávají přímo na hodinách nebo pomocí on-line systémů. Druhá možnost je mnohem častější a většinou časově náročnější. Je tedy nutné dobře vybrat systém, který se bude v předmětu používat, a zvážit, co vše by měl splňovat.

Například, měl by systém umožňovat učitelům vkládat automatické testy? Na první pohled je pro studenta velmi pohodlné si úkol nechat otestovat, než jej odevzdá. Z vlastní zkušenosti ale vím, že může být zprovoznění těchto testů pro studenta náročnější než vypracování úkolu samotného. Dále je nutné ze strany učitele tyto testy vymyslet a naprogramovat, což může být zbytečně zdlouhavé vzhledem k rozsahu úkolu. Jejich zavedením se dají odhalit úkoly, kde se vyskytuje syntaktická chyba nebo kde program nevrací výsledky podle zadání. Těžko se ale pomocí nich kontroluje celková kvalita kódu. Ta stále závisí na posouzení vyučujícího.

To je jen jedna z mnoha funkcionalit, které by požadovaný systém mohl mít, a je třeba u nich vždy zvážit všechny výhody, nevýhody a náročnost implementace.

1.1 Motivace

Od doby, co jsem začal na univerzitě studovat, jsem se setkal s různými způsoby odevzdávání programovacích úkolů. Žádný z nich nebyl ideální jak pro studenty, tak pro učitele. Pro mě osobně byly programovací úkoly vždy časově nejnáročnější částí studijních povinností. Nejvíce mě ale mrzelo, že jsem se musel potýkat s problémy i při jejich odevzdávání. Pamatuji si, že jsem jednou řešil odevzdání úkolu téměř dvě hodiny, což byl čas, za který se daný úkol dalo stihnout vyřešit.

To ale nebyl ten největší impuls. Ten přišel až v předmětu Tvorba webových stránek. Měl jsem totiž možnost vidět, jak časově náročné je opravování úkolů pro učitele. Také jsem si všiml, jaké má používaný systém nedostatky. Často jsem si říkal, jak by bylo těžké vytvořit systém, který by vyhovoval mně, mým kolegům z ročníku i našim učitelům více než existující řešení. Tak jsem se rozhodl, že se o to pokusím v rámci bakalářské práce.

2 Existující řešení

Jak už bylo zmíněno, v průběhu studia jsem se setkal s různými řešeními tohoto problému. Github Classroom [1], Microsoft Teams [2] a odevzdávání e-mailem byly určitě nejčastější z nich. Proto je porovnám a vyvodím z nich požadavky na optimální systém pro naši katedru. Do tohoto porovnání zahrnu i Google Classroom [3], protože je to velmi rozšířený systém a je uživatelsky přívětivý.

2.1 E-mail

Ukázalo se, že nechávat si od studentů posílat úkoly e-mailem je překvapivě dobré řešení. V rámci předmětů Jazyk C# 1 a Jazyk C# 2 jsme tento způsob praktikovali a především ze strany studenta mohu potvrdit oblíbenost tohoto způsobu. Studenti se totiž nemusí učit pracovat s žádným novým systémem a není ani nutné nic nastavovat ze strany učitele.

Má samozřejmě jasné nedostatky a určitě není optimální. Například, tento způsob neumožňuje zdrojové kódy úkolu prohlížet přímo v prohlížeči. Učitel si musí vést záznamy o prospěchu studentů sám. E-mail se dá omylem přečíst a nevyřídit, přičemž na to nebude nikdo upozorněn. Dále je zde přítomný limit velikosti příloh, nemluvě o tom, že technologie e-mail nebyla původně navržena pro posílání souborů. Dodal bych ještě, že soubory, které se v rámci programovacích úloh posílají, jsou často označeny e-mailovým klientem za spam (nevyžádanou poštu). To může opět jednoduše vést k nevyřízení e-mailu.

2.2 Microsoft Teams

Microsoft Teams je komunikační a kolaborační platforma, která umožňuje učitelům a studentům komunikovat, spolupracovat a organizovat vzdělávací procesy online. Teams nabízí prostředí pro vytváření tříd, kde lze sdílet dokumenty, rozvrhy, úkoly a další relevantní materiály. Platforma také poskytuje možnost provádět videohovory a chatování[1].

2.2.1 Výhody

Microsoft Teams je integrovaný s dalšími nástroji Microsoft 365, což usnadňuje práci s dokumenty a dalšími aplikacemi, jako je Word, Excel, PowerPoint, OneNote, ...

Má také pokročilé funkce pro organizaci tříd, správu úkolů, sledování pokroku studentů a ohodnocování jejich práce. Díky možnosti videokonferencí a chatování je komunikace mezi učiteli a studenty plynulá a efektivní.

[1] Posílání si zpráv skrze konverzační vlákno.

2.2.2 Nevýhody

Někteří uživatelé mohou mít problémy naučit se práci s pokročilou funkcionalitou platformy Teams, zejména pokud nejsou obeznámeni s ekosystémem Microsoft 365. Pak jsou tyto funkce nevyužité a jen zatěžují systém. Naproti tomu chybí jednoduše dostupný přehled součtu bodů daného studenta.

Dále má aplikace dvě verze, webovou a pro desktop, které se chovají mírně odlišně. Často tak nastává, že uživatel spoléhá na nějakou funkcionalitu, která ale v té verzi, kterou zrovna používá, chybí. Dále aplikace často vykazuje chyby, především webová verze aplikace. Dále se aplikace v mnoha chvílích (tedy nejen při svém spouštění) dlouho načítá.

Vždy pro mě osobně bylo nepříjemné například listovat soubory. Jednak se vše dlouho načítalo a rozhraní se mi nelíbilo, protože vybraný soubor překryje komponentu pro zobrazení struktury složek a souborů. Z vývojových prostředí jsem zvyklý mít tuto komponentu v postranním panelu stále viditelnou.

2.3 Github Classroom

GitHub Classroom je webová aplikace postavená na platformě GitHub. Umožňuje vyučujícím spravovat kurzy a zadávat úkoly studentům prostřednictvím GitHub repositářů. GitHub Classroom také poskytuje učitelům možnost sledovat a hodnotit práci studentů a studentům přidávat zadání přímo v rámci webové aplikace GitHub. (Ta je rozšířeným způsobem, jak využívat technologii Git k verzování a sdílení projektů. O technologii Git více v kapitole Git 4.1.8.)

2.3.1 Výhody

GitHub a systém Git jsou v praxi používanými nástroji pro vývojáře, a tak se studenti zároveň naučí alespoň základní práci s tímto prostředím. Má pokročilé funkce správy verzí, což usnadňuje sledování změn v kódu a opravování úkolů. Vyzvedl bych komentáře k jednotlivým řádkům a poměrně přehledné porovnávání různých verzí jednoho souboru. Výhodou je i integrace s nástroji pro vývojáře.

2.3.2 Nevýhody

Ačkoliv je navržený pro učitele a studenty, práce s ním není intuitivní a určitě by mohl být kladen větší důraz na základní funkcionalitu. Spousta funkcí je jen pro náročné uživatele nebo speciální případy a při každodenní práci spíše překáží. Naproti tomu denně využívané funkce nemají prostor být snadno dostupné.

GitHub Classroom je navíc vhodný spíše pro menší třídy a projekty, ale pro velké skupiny studentů může být obtížné udržovat přehled nad všemi úkoly a repozitáři. Chybí také důležité prvky, které by uživateli usnadnily práci. Konkrétně třeba (tabulka) všech studentů v jedné třídě s jejich prospěchem.

2.4 Google Classroom

Google Classroom je vzdělávací platforma od společnosti Google, která je navržena pro vyučování a organizaci učebních materiálů online. Umožňuje učitelům vytvářet třídy, zadávat úkoly, sdílet materiály a komunikovat se studenty pomocí jednoho centrálního místa.

2.4.1 Výhody

Google Classroom má jednoduché a intuitivní rozhraní, které umožňuje učitelům a studentům rychle se s ním seznámit. Učitelé mohou snadno vytvářet a spravovat třídy, úkoly i materiály. Studenti mají přehled o všech svých úkolech a materiálech na jednom místě. Google Classroom také usnadňuje komunikaci mezi učiteli a studenty. Učitelé mohou posílat zprávy a zpětnou vazbu a studenti mohou sdílet své práce a komentovat příspěvky. Opět je vše velmi přehledné a intuitivní. Google Classroom je navíc propojen s dalšími nástroji od Googlu, jako je Google Drive pro ukládání a sdílení souborů.

2.4.2 Nevýhody

Někdy může být Google Classroom považován za omezený v porovnání s jinými vzdělávacími platformami. Pro náš případ chybí klíčová funkcionalita (například zvýraznění syntaxe odevzdaného zdrojového kódu). Není zkrátka navržen pro práci s programovacími úlohami. Další nevýhodou je přílišná závislost na platformě Google. (Někteří učitelé by zkrátka nechtěli používat tuto platformu kvůli osobním preferencím. A stejné by to bylo u studentů.)

2.5 Závěry

Ze způsobu odeslání e-mailem jsem si vzal poznatek, že když použiji nějakou komponentu, se kterou už uživatelé mají zkušenost, bude to pro ně mnohem příjemnější. Napadlo mě tedy použít chatovací vlákno[2] na posílání zpráv u úkolu, které bude podobné vláknům ze známých sociálních sítí.

Dále z Google Classroom bych se rád inspiroval přímočarostí a jednoduchostí základní funkcionality a zčásti i vzhledem.

Pak pro uchovávání verzí úkolu využiji technologii Git, o které budu ještě psát v kapitole o technologiích 4.1.8, a na které stojí GitHub Classroom.

Jelikož jsem neměl s vývojem webových aplikací zkušenosti a obvykle se takového vývoje účastní celý tým vývojářů, rozhodl jsem se, že některé funkce po systému požadovat nebudu, abych udržel jednoduchost implementace. Například hodnotit úkol bude možné pouze pomocí bodů. Uživatel si bude moct zobrazit

[2]Jen pro upřesnění, pojmem „chat“ a „chatovací vlákno“ je myšleno konverzační vlákno, tedy známé grafické rozhraní používané pro komunikaci. Dále v textu používám slovo chat v různých podobách.

pouze dva soubory zároveň a získání role učitele nebude automatizováno, ale bude nutné kontaktovat administrátora.

Většina těchto řešení je realizována jako webová aplikace. I já jsem se tedy rozhodl jít touto cestou. Z předmětů Tvorba webových stránek a Webové technologie mám navíc nějaké základní povědomí, jak pracovat s obsahem on-line.

I díky těmto poznatkům jsem nyní schopen určit konkrétní požadavky na systém, který by uspokojil potřeby naší katedry. Budu se jim věnovat v následující kapitole.

3 Požadavky

Systém bude poskytovat přívětivé uživatelské rozhraní a bude realizován jako webová aplikace. Uživatelé se budou moct registrovat jakožto studenti, následně pak získat roli učitele. (Na stránce s nastavením je kontaktní e-mail). Požadavky na systém můžeme rozdělit podle role, kterou uživatel v daném kontextu bude mít.

Učitelé:

- Zakládat kurzy a jednoduše do nich studenty přidávat a zase je z nich odebírat.
- Vytvářet v kurzech zadání a připojit k nim i soubory (ty mohou sloužit například jako počáteční šablona).
- Hodnotit úkoly počtem bodů.
- Poslat zprávu všem studentům, kteří plní úkol stejného zadání.
- Zobrazit si přehled o prospěchu studentů v kurzu.

Studenti:

- Přijmout zadání.
- Odevzdávat úkoly s uchováváním všech verzí.
- Zobrazit si přehled o svém prospěchu v kurzu.

Učitelé i Studenti:

- Zobrazit si náhled odevzdaných textových souborů. Pokud se bude jednat o zdrojový kód nějakého programovacího jazyka, tak i se zvýrazněním syntaxe.
- Stáhnout nahrané soubory. Ať už se bude jednat o kteroukoliv z odevzdaných verzí nebo dodatečné soubory k zadání.
- Komunikovat mezi sebou v rámci úkolu, který student plní. To bude zahrnovat posílání zpráv, u kterých bude možné označit, kterých řádků kterého odevzdaného souboru se zpráva týká.

Dále bude vytvořeno jednoduché administrativní rozhraní pro nastavování rolí uživatelů.

4 Technický popis aplikace

V této kapitole popíšu, jakým způsobem bylo řešení implementováno. Důraz bude kladen na architekturu a účel jednotlivých technologií.

4.1 Použité technologie

Při vývoji webových aplikací existuje spousta různých přístupů, technologií a architektur, mezi kterými si musí vývojář vybrat. Musím dodat, že bez předchozích zkušeností je to velmi obtížné. V této kapitole popíšu účel a rysy těch, které jsem zvolil.

4.1.1 HTML

Hypertext Markup Language [4] je tedy dle názvu hypertextový značkovací jazyk.

Hypertext je text, který obsahuje odkazy na své části nebo na jiné textové dokumenty. Na tento typ dokumentů se při použití HTML zaměřujeme (jde o základní koncept fungování služby WWW).

HTML pak umožňuje označit části webové stránky, dát jí strukturu a význam (sémantiku). Toho HTML dosahuje pomocí *tagů* (značek, odtud označení značkovací jazyk), které obklopují obsah nebo další HTML tagy. Tímto způsobem je možné definovat na webové stránce například nadpisy, sekce, odstavce, ...

Prohlížeče a další technologie reagují na určitá místa na stránce dle tohoto popisu. Také mohou díky tomu s obsahem stránky jednodušeji manipulovat a analyzovat jej.

V minulosti se používal i pro úpravu vzhledu, ale to bylo již zcela přesunuto na CSS (viz kapitola 4.1.2). V dnešní době už by se pro úpravu vzhledu používat neměl. Slouží opravdu pouze pro popis struktury a významu částí obsahu stránky.

Bohužel se stále objevují případy, kdy lidé mylně používají specifické HTML tagy za účelem zvýraznění, nastavení velikosti či změny barvy. Například text obalený v tagu `<strong>` je ve výchozím stavu v prohlížečích zobrazován tučně (z historických důvodů). Na některých webových stránkách se této skutečnosti využívá dodnes. Přitom pro tučné písmo se má používat technologie CSS.

4.1.2 CSS

Cascading Style Sheets [5] (česky též kaskádové styly) je jazyk používaný k definování vzhledu a formátování webových stránek. Jeho hlavním úkolem je oddělit vzhled od struktury a obsahu webových stránek, což přispívá k lepší správě a údržbě.

CSS funguje tak, že pomocí *selektorů* vybírá konkrétní elementy na stránce a přiřazuje jim *vlastnosti*. Selektory mohou být například jméno třídy, identifikátor, typ elementu nebo jejich kombinace. Vlastnost je pak charakteristika daného elementu ovlivňující vzhled a rozložení. Tedy podle toho, co definujeme v HTML, můžeme v CSS vybrat určitou část, které pak nastavíme vlastnosti.

Díky CSS lze upravovat barvu, velikost písma, zarovnání, pozice, ohraničení, pozadí, animace, ... CSS zkrátka poskytuje velké množství vlastností, které umožňují dosáhnout přesného a přizpůsobeného vzhledu.

Kromě inline CSS (kdy je styl přímo definován uvnitř HTML elementu) se nejčastěji používá externí CSS soubor, který je připojen k HTML dokumentu pomocí značky <link>. Tím se zajistí oddělení stylů od obsahu a umožní snadnou změnu vzhledu webových stránek napříč celým webem.

Používání CSS ve spojení s HTML umožňuje vytvářet dobře strukturované stránky, které jsou snadno čitelné, přístupné a vizuálně přívětivé pro uživatele. Poskytuje také důležitou možnost přizpůsobení vzhledu stránek různým zařízením.

V současnosti se při vývoji webových stránek používají častěji CSS knihovny než vlastní CSS.

4.1.2.1 CSS knihovny

CSS knihovny jsou předem vytvořené sady stylů a komponent.

Používání CSS knihoven má několik výhod. Nespornou výhodou je zrychlení vývoje webových stránek. Dále je jednodušší dosáhnout konzistentního vzhledu a jednotného stylu napříč celým webem. Také je jednodušší naučit se práci s CSS knihovnou, než se samotným CSS.

Nicméně existují také některé nevýhody spojené s používáním CSS knihoven. Jednou z nevýhod je omezená flexibilita a možnost individuálního vzhledu, protože se pracuje s předem definovanými styly. Další nevýhodou může být možnost ovlivnění rychlosti načítání webových stránek v případě, kdy jsou knihovny obsáhlé.

Je důležité vyvážit výhody a nevýhody při výběru a používání CSS knihoven a zvážit vše vzhledem ke konkrétnímu projektu.

4.1.3 JavaScript

Už máme sémantiku a strukturu webové stránky pomocí HTML, pak pomocí CSS vzhled. Zbývá logika webové stránky. Programovacím jazykem, který nám ji poskytne, je JavaScript [6]. Jde o dynamický[3] prototypový programovací jazyk, který běžně běží v rámci webové stránky v prohlížeči uživatele. Může libovolně manipulovat s CSS a HTML, tudíž jde o opravdu mocný nástroj. Umožňuje nám vytvořit webovou stránku stejné funkcionality, jako by šlo o klasickou aplikaci. To vše ale znamená, že uživatel má přístup ke zdrojovému kódu, a dokonce může v průběhu vykonávání s kódem manipulovat. Na to je nutné myslet a nikdy se nespoléhat na ověření vstupů jen na straně uživatele pomocí JavaScriptu. V závislosti na projektu je pak nutné zvážit, zda není vhodné využít například klient-

[3] Umožňuje běh programu bez nutnosti kompilace a deklarace datových typů proměnných. Dynamické jazyky jsou používány pro webové aplikace, protože je zde důležitá rychlost vývoje a snadná úprava kódu.

server architekturu (viz kapitola 4.3.1) pro zachování konzistence a bezpečnosti systému.

Dnes je moderní využívat JavaScript i na tvorbu serverové části webové stránky. Vytvoříme pomocí něj program, který přijímá požadavky a odesílá nazpět odpovědi, které většinou obsahují data. Výhodou je pak především jednotná logika (komunikace mezi dvěma programovacími jazyky může být složitá na implementaci). K tomu bylo potřeba vyvinout běhové prostředí. Příkladem je Node.js [7].

4.1.4 Node.js

Jde o běhové prostředí pro JavaScript, které umožňuje vykonávat kód mimo prohlížeč. Napsáno je v jazyce C++ a je postaveno na enginu V8 od Googlu. Je vhodný pro vytváření API[4], které poskytuje data pro další aplikace, či její části. Existuje velké množství knihoven s veškerou potřebnou funkcionalitou pro vývoj backendové části aplikace v JavaScriptu.

4.1.5 NPM

NPM [8] je správce balíčků pro Node.js. Zjednodušuje práci s instalováním balíčků (knihoven, též modulů), které jsou potřeba pro vývoj a běh aplikace. Dále umožňuje spravovat tyto balíčky a jejich verze. Praxe je pak taková, že pomocí informací v konfiguračních souborech NPM vygeneruje složku node modules a její obsah. Stáhne a nainstaluje vše potřebné, a to pomocí příkazu npm install. Důležité je tedy zachovat konfigurační soubory, nikoliv složku s knihovnami. Když pak projekt, který npm využívá, chceme přesunout na jiné zařízení, složku node modules nepřesouváme, ale necháme si ji vygenerovat v novém prostředí. Vynechává se i při verzování.

4.1.6 Vue.js

Pro komunikaci se serverem, vykreslení dat a také interakci s uživatelem v dnešní době není čistý JavaScript většinou vhodnou volbou. Museli bychom vytvářet velké množství struktur a funkcí a neustále přímo přes nativní JavaScriptové funkce manipulovat s obsahem stránky. To by bylo pro oko uživatele pomalé a projekt by se jednoduše mohl stát nepřehledný, nemluvě o jeho časové náročnosti.

Největší problém by nejspíš bylo překreslování. Jakmile se změní nějaká data, chceme, aby se překreslilo vše, na co má tato změna dopad. To není zcela triviální záležitost, hlavně pokud chceme, aby se to dělo rychle. V každé webové aplikaci se snažíme dosáhnout toho, aby se změna projevila co nejrychleji, aby nebyl uživatel

[4] Api je zkratka pro aplikační rozhraní, tedy způsob, kterým spolu komunikují aplikace.

frustrován čekáním. K tomu všemu právě slouží JavaScriptové frameworky [5] jako je Vue.js [9]. Dají nám strukturu, abstrakci a reaktivitu.

Důležitou funkcionalitou, kterou Vue.js disponuje, je obousměrná vazba mezi proměnnou a jejím zobrazením. HTML elementu, jako je třeba <input> stačí jen definovat, na jakou proměnnou je navázán. Když se pak změní hodnota proměnné, změní se i zobrazená hodnota. A naopak, když se změní zobrazovaná hodnota (což prohlížeč u tohoto elementu umožňuje), změní se i hodnota proměnné.

Každý Vue.js projekt se skládá z komponent, což jsou většinou samostatné soubory, které obsahují část pro definici zobrazení pomocí HTML, část pro logiku (JavaScript s předepsanými funkcemi a pravidly specifickými pro Vue.js) a část pro stylování pomocí CSS.

4.1.7 SQLite

Jde o open-source [6] relační databázový systém. SQLite [10] uchovává informace v tabulkách a mezi tabulkami se definují asociace. Používá klasickou SQL syntax. Dále používá transakční systém pro udržení konzistentního stavu. Umožňuje vícenásobný přístup k datům. Je velmi podobný známým databázovým systémům jako je Postgresql [11] nebo MySQL [12]. Do Node.js jsem jej napojil pomocí modulu sqlite3 [13] a pracuji s ním přes ORM Sequelize (viz kapitola 4.2.6).

4.1.8 Git

Git je nástroj pro správu verzí kódu. Umožňuje sledovat změny v projektu a snadno se k nim vracet. Git je široce používaný vývojáři k organizaci a spolupráci při vývoji softwaru. Na tomto systému stojí webová aplikace GitHub [14], o které jsem se zmínil v kapitole o existujících řešeních. Já ji ve své práci také využiji.

4.2 Použité knihovny

Ve světě webových technologií, zejména v ekosystému Node.js, je k dispozici ohromné množství knihoven. S nadsázkou lze říci, že na téměř každou funkcionalitu a problém existuje specifická knihovna, která nabízí elegantní a efektivní řešení.

4.2.1 Highlight.js

Další vybranou knihovnou je Highlight.js [15], která slouží ke zvýrazňování syntaxe kódu na webových stránkách. Tato knihovna nabízí širokou škálu podpo-

[5] Framework stejně jako knihovna označuje sadu funkcí a pravidel pro použití v různých projektech. Rozdíl mezi frameworkem a knihovnou je v míře abstrakce a volnosti při práci. Framework je striktnější.

[6] Takto se označuje software, jehož zdrojový kód je veřejně dostupný a může být volně používán, upravován a šířen komunitou vývojářů.

rovaných programovacích jazyků a formátů, což se velmi hodí, jelikož budeme potřebovat zvýraznit syntaxi u odevzdaných zdrojových kódů. Highlight.js jsem začlenil do Vue.js projektu využitím pluginu. Tato integrace přináší nejen pokročilé možnosti zvýrazňování syntaxe, ale také jednoduchou správu a konfiguraci zvýraznění pro různé jazyky a styly kódu. Když se bude na webu implementovat tmavý a světlý režim, způsob zvýraznění syntaxe se tomu dá jednoduše přizpůsobit.

4.2.2 Tiny-emitter

Tiny-emitter [16] je lehká a efektivní knihovna pro správu událostí v jazyce JavaScript. Poskytuje elegantní a jednoduché rozhraní pro publikování a odběr událostí v rámci aplikace. Tato knihovna se ukázala jako užitečný nástroj pro komunikaci mezi různými komponentami, moduly a dalšími částmi kódu v projektu.

4.2.3 Tailwind CSS

Při vývoji frontendu jsem se rozhodl využít Tailwind CSS [17], což je CSS knihovna založená na principu utility-first (předdefinovaná pravidla ovlivňují většinou jen jednu vlastnost). Tailwind CSS mi umožňuje rychle a jednoduše vytvářet responzivní uživatelská rozhraní pomocí jednoduchých a opakovaně použitelných tříd.

Kvůli atomičnosti[7] je sice pořádek v CSS, ale HTML kód se může snadno stát nepřehledným. Příkladem je Daisy UI

4.2.3.1 Daisy UI

Použil jsem i Daisy UI [18], což je rozšíření pro Tailwind, které mi usnadnilo vývoj díky předdefinovaným CSS třídám a barevným kombinacím, které jsou jednoduše volitelné.

4.2.4 Express

Express [19] poskytuje vše potřebné pro zpracováváni HTTP požadavků a posílání odpovědí. Je to jednoduchý, flexibilní a robustní framework. Díky tomu se stal velmi populárním a rozšířeným mezi vývojáři. Existuje spousta rozšíření a také se snadno integruje s dalšími technologiemi v ekosystému Node.js.

4.2.5 Multer

Multer [20] je middleware (rozhraní mezi frontendem a backendem), který zajišťuje přijímání souborů od uživatele. Rozšiřuje základní framework, jako je například Express, o tuto funkcionalitu.

[7] Ve většině případů jsou Tailwind CSS třídy napsané tak, že nastavují pouze jednu vlastnost. Tomu se v kontextu CSS říká atomické.

4.2.6 Sequelize

Sequelize [21] je to ORM (Object-Relational Mapping), zajišťuje mapování objektů do relační databáze. Místo tabulky se zde používá pojem *Model*. Když se bavíme o konkrétních záznamech nějakého modelu, nazýváme jej *instancí*. Vývojář pak může pracovat s Modlely a instancemi jako s objekty v programu a změny se pak projeví do napojené databáze.

4.2.7 Simple-Git

Poskytuje jednoduché rozhraní a abstrakci nad Git příkazy, což umožňuje snadno integrovat a automatizovat operace spojené se správou verzí. Simple-Git [22] jsem využil konkrétně při implementaci verzování odevzdaných úkolů.

4.2.8 Jsonwebtoken

Díky této knihovně Jsonwebtoken [23] jsem schopný si jednoduše vytvářet takzvané tokeny, přes které se dá implementovat autentifikace uživatele, tedy uchovávání si informace o tom, jestli je uživatel přihlášený. Token je objekt zakódovaný pomocí tajného klíče (ten je uložený na serveru) do zdánlivě náhodného řetězce znaků. Pomocí tajného klíče jsem pak schopný z řetězce dekódovat uložený objekt. Dodám ještě, že má nastavitelnou dobu expirace kvůli zvýšení bezpečnosti. Podrobnosti v kapitole Autorizace a autentifikace 4.3.2.1.

4.3 Architerkura a struktura aplikace

Návrh architektury je vždy velice náročná a důležitá část vývoje jakéhokoliv systému. Naštěstí máme určité vzory a osvědčené metody, které nám návrh zjednodušují. Samozřejmostí je i spousta dnes používaných architektur pro tvorbu webových aplikací, ze kterých si stačí vybrat. Jednou z nich je klient-server.

4.3.1 Klient-server

Tato architektura je založena na rozdělení úkolů mezi dvě základní role: klienta a server. Klient je počítačový program nebo zařízení, které žádá o služby či zdroje, zatímco server je počítačový program nebo zařízení, které tyto služby či zdroje poskytuje.

Při implementaci této architektury se často používají názvy frontend a backend. Frontend je část projektu, která odpovídá implementaci klienta a backend zase serveru. Z určitého pohledu jsou tedy tato označení analogická. Jelikož budou následující odstavce i o konkrétních technologiích, tedy jde i o implementaci, budu požívat spíše označení frontend a backend.

4.3.1.1 Výhody

První (a z mého pohledu největší) výhodou je bezpečnost. Ve světě webových technologií se často musíme smířit s tím, že větší, či menší část kódu je přístupná uživateli. Jak jsem psal v kapitole o jazyce JavaScript 4.1.3, tento jazyk je dynamický a dokonce i za běhu do něj uživatel může zasahovat. Rozdělíme-li však chytře úlohy mezi kód vykonávaný na serveru a u klienta, můžeme minimalizovat riziko neoprávněného zásahu do systému. Server bude přijímat požadavky, na základě kterých bude měnit vnitřní stav systému. Pokaždé můžeme ověřit, jestli má klient právo na požadovanou akci. Veřejný kód tedy neovlivňuje přímo vnitřní stav aplikace. Podrobněji se tomu budu věnovat v kapitole Autorizace a autentifikace 4.3.2.1.

U webových aplikací, které implementují klient-server architekturu, existují různé způsoby rozdělení odpovědnosti, tedy co má na starosti frontend (klient) a co backend (server).

4.3.1.2 SPA

Single Page Application, česky jednostránková aplikace (v našem kontextu můžeme chápat jako jednostránkovou webovou aplikaci). Vyznačuje se především tím, že se obsah dynamicky generuje přímo na straně klienta.

Ze serverové strany přijdou jen data. Ty se v prohlížeči uživatele sestaví a následně zobrazí HTML a CSS. Všechny pomyslné stránky, které uživatel takto navštíví, jsou ve skutečnosti pořád ta stejná HTML stránka, která jen mění svůj obsah.

Aby byl klient (prohlížeč uživatele) schopný tato data zobrazit, musí nejprve získat ze serveru všechny potřebné soubory (frontend webové aplikace). Dotáže

se tedy serveru a stáhne celý frontend aplikace[8] Ten už si pak prohlížeč může uchovat a příště jej stahovat nemusí. Z hlediska uživatele se jedná o první navštívení webové stránky a její načtení se může jevit pomalejší právě kvůli tomu, že se musí nejprve vše potřebné stáhnout.

Důležité je, že server, který poskytne frontend, nemusí být ten samý jako server poskytující data. Tyto programy mohou běžet zcela odděleně.

Frontendová část aplikace má na starosti zobrazení uživatelského rozhraní a interakci s uživatelem. Je implementována pomocí HTML, CSS a JavaScriptu, který běží v prohlížeči klienta. Frontendová část SPA se stará o zobrazování dat a interakci s uživatelem pomocí událostí a akcí. Typicky se v SPA používají frameworky, jako je například v této práci využívaný Vue.js. Bez těchto frameworků by bylo obtížné dosáhnout potřebné funkcionality, jak bylo popsáno v kapitole o Vue.js 4.1.6.

Backendová část aplikace má na starosti poskytování datové logiky (vytvoření záznamů, jejich čtení a úpravu), zpracování požadavků klienta a komunikaci s databází nebo dalšími službami. Backend je v této práci implementován pomocí Node.js a Express.js.

Jelikož jsou frontend a backend samostatné subsystémy, mají i svoji vlastní architekturu a strukturu.

4.3.2 Backend

Kód jsem rozdělil do logických celků (souborů a složek). V kořenové složce jsou konfigurační soubory, několik pomocných scriptů (JavaScriptových souborů) a především soubor *server.js*. Po spuštění souboru server.js (pomocí příkazu *node ./server*) se načte vše potřebné a zahájí se provoz serveru. Ve složce node modules jsou pak všechny potřebné nainstalované knihovny (moduly). O tom, co je třeba nainstalovat, jsou informace uložené v konfiguračních souborech. Odtud má pak systém NPM, o kterém jsem se již zmiňoval, všechny potřebné informace. Dále je zde složka *controllers* a *models*, ke kterým se postupně dostanu v průběhu této kapitoly.

4.3.2.1 Autorizace a autentifikace

Autentifikaci realizuji pomocí knihovny Jsonwebtoken (viz kapitola 4.2.8). Konkrétněji takto:

Při úspěšném přihlášení vytvořím token, ve kterém je uloženo id [9] uživatele a pošlu jej klientovi, ten si ho uloží a při každém dalším požadavku pošle v hlavičce. Serveru tedy přijde s požadavkem i token. Než vykonám jakýkoli požadavek, který vyžaduje přihlášení, vždy dekóduji token. Pokud je token úspěšně dekódován, vím, že je uživatel přihlášen. Pokud by dekódování nebylo úspěšné

[8] Většinou se jedná o již sestavenou aplikaci, tudíž jen HTML, CSS a JavaScript. Nemusí to tak být vždy. Především během vývoje běžně běží druhý server, který poskytuje frontend klientovi.

[9] Jedinečný identifikátor uživatele

nebo by byl token po expiraci, odešlu zpět odpovídající odpověď. Pro udržení 60 minut jako doby od poslední interakce posílám téměř při každém úspěšně zpracovaném požadavku nový token, který na straně klienta nahradí ten starý.

Autorizace jde s autentifikací ruku v ruce. Tím, že jsem schopný takto uložit id uživatele, jsem také schopný určit, který uživatel požadavek poslal. To hraje zásadní roli při určování toho, zda je oprávněn danou akci žádat. U všech požadavků, které vyžadují přihlášení, kontroluji, zda je přihlášený uživatel oprávněný (autorizovaný) požadovat danou akci. Například změnit název kurzu může pouze přihlášený učitel kurzu. Samozřejmě zde máme vždy speciální případ pro uživatele s rolí *admin*. Ten může požadovat po serveru jakoukoli akci. Toho jsem docílil pomocí dekorátoru [10]. Ten jsem implementoval pomocí funkce, která jako parametr přijímá funkci, která by měla sloužit pro autorizaci uživatele (autorizační funkce). Dekorátor pak vrací novou funkci, která nejprve zkontroluje, jestli přihlášený uživatel nemá roli *admin*. Pokud ano, rovnou vyhodnotí autorizaci za úspěšně provedenou. Pokud ne, úspěch autorizace závisí na autorizační funkci (kód 1).

Zdrojový kód 1: Autentifikační dekorátor

```
exports.authorizationDecorator = (func) => {
  return async (req, res, ...args) => {
    if (
      !((await this.loggedAdmin(req, res)) ||
        (await func(req, res, ...args)))
    ) {
      res.status(403);
      throw Error("NotAuthorized");
    }
    return true;
  };
};
```

Tímto dekorátorem obalím vždy všechny autorizační funkce (viz kapitola 4.3.2.1). Když budu chtít změnit chování pro autorizaci, stačí jej upravit na jednom místě.

4.3.2.2 Modely

Ve složce *models* mám definovanou strukturu databáze pomocí knihovny Sequelize (viz kapitola 4.2.6). Definuji zde Modely (kód 2), jejich asociace (kód 3) a chování při určitých událostech. Sequelize sám vygeneruje metody pro práci s Modely a jejich instancemi na základě definovaných parametrů a asociací. Díky tomuto způsobu práce s databází jsou podrobnosti o její struktuře pro vývojáře nepodstatné[11]. Na druhou stranu důležité skutečnosti jsou dobře viditelné přímo

[10] V tomto kontextu jde o funkci, která přidává stejnou funkcionalitu různým funkcím.

[11] Například to, že pro uchování pravdivostní hodnoty SQLite používá datový typ TINYINT je jen zajímavost, kterou vývojář znát nemusí.

z kódu (kódy 2 a 3).

Zdrojový kód 2: Příklad definice Modelu, zkráceno

```
const Course = sequelize.define(
  "Course",
  {
    name: {
      type: DataTypes.STRING(nameLength),
      allowNull: false,
      validate: {
        notEmpty: true,
        notNull: true,
        len: [1, nameLength],
      },
    },
    description: {
      type: DataTypes.STRING(shortDescriptionLength),
      validate: {
        notEmpty: true,
        len: [0, shortDescriptionLength],
      },
    },
    ...
```

Zdrojový kód 3: Příklad definice asociací Modelů

```
Course.hasMany(Task, {
  as: "tasks",
  foreignKey: "courseId",
  onDelete: "CASCADE",
});
Task.belongsTo(Course, {
  as: "course",
});
```

4.3.2.3 Router

V souboru server.js je mimo jiné popsáno, které požadavky jsou směrovány na kterou konkrétní funkci. V tomto příkladu je adresa požadavku „/api/course/-delete“ a je přesměrován na funkci „routesDeleteCourse“ (kód 4).

Zdrojový kód 4: Příklad přesměrování na routovací funkci

```
app.delete(
  "/api/course/delete",
  courseController.routesDeleteCourse
);
```

4.3.2.4 Kontrolery

Ve složce controllers jsou pak soubory, z nichž každý obsahuje ucelenou funkcionalitu pro práci s nějakou entitou v systému. Většinou se jedná přímo o některý z Modelů (viz kapitola 4.3.2.2). Kontrolery obsahují především funkce. Ty jsem rozdělil do několika kategorií podle toho, jaký mají účel a jestli je lze volat vně kontroleru. Pro účely tohoto textu jsem je pojmenoval *Routovací*, *Specifické*, *Pomocné* a *Odesílací* funkce.

- Routovací funkce se exportují. Mají prefix „routes“ a jsou určené pro volání přímo po přijmutí požadavku od klienta. Předzpracují argumenty z požadavku. Zavolají Autorizační funkci. Poté buď zavolají některou Specifickou funkci, nebo přímo ve svém těle vykonají vše potřebné. Pokud vše proběhlo v pořádku, je zavolána klasická Odesílací funkce s daty pro klienta. Pokud požadavek neprojde autorizací nebo se objeví jiná výjimka, zavolá se Odesílací funkce pro odesílání chyb (objektů s informacemi o chybě). Příkladem je Routovací funkce pro požadavek na odstranění kurzu (kód 5).

Zdrojový kód 5: Příklad Routovací funkce

```
exports.routesDeleteCourse = async (req, res) => {
  const courseId = req.body.courseId;
  const result = {};

  try {
    await authDeleteCourse(req, res, courseId);
    if (await this.deleteCourse(courseId)) {
      result.mess = "Course deleted";
    } else {
      throw Error("CourseNotDeleted");
    }
    logAndSend(res, result);
  } catch (error) {
    sendError(res, error);
  }
};
```

- Specifické funkce se exportují. Jsou bez prefixu. Slouží pro volání v rámci stejného kontroleru, ale i mezi kontrolery navzájem. Vykonají vše potřebné, dle jejich účelu. Příkladem je samotné odstranění kurzu (kód 6).

Zdrojový kód 6: Příklad Specifické funkce pro odstranění kurzu

```
exports.deletecourse = async (courseId) => {
  const course = await Course.findByPk(courseId);
  course.destroy();
  return true;
};
```

- Pomocné funkce se neexportují. Slouží jen pro oddělení funkcionality z důvodů přehlednosti a usnadnění si práce. Příkladem je přidání jednoho studenta do kurzu na základě jeho e-mailu. Tuto funkci volám ve Specifické funkci pro přidávání studentů na základě jejich e-mailů (kód 7).

Zdrojový kód 7: Příklad Pomocné funkce

```
const attachStudentByEmail = async (student, course) => {
    const newStudent = await registerUser(student);
    if (!newStudent) {
      return false;
    }

    await course.addStudent(newStudent);
    return true;
};
```

- Autorizační funkce se neexportují, ověřují, jestli má uživatel právo na určitou akci. Vytváří se pomocí dekorátoru, o kterém byla zmínka v kapitole Autorizace a autentifikace 4.3.2.1. Vracejí *true* nebo *false* hodnotu. Dekorátor pak při záporné hodnotě vyvolá výjimku (kód 8).

Zdrojový kód 8: Příklad Autentifikační funkce

```
const authUpdateCourse =
  authorizationDecorator(async (req, res, courseId) => {
    return teacherOfCourse(req, res, courseId);
});
```

- Odesílací funkce slouží pro odeslání odpovědi klientovi. Nastaví správný status, odešlou hlášku o chybě v odpovídajícím formátu, ...

  Díky tomu je centralizováno veškeré odesílání odpovědí. V ukázce to využívám pro to, abych si do konzole vypsal veškeré odpovědi, které klientovi posílám (kód 9).

Zdrojový kód 9: Příklad Odesílací funkce

```
exports.logAndSend = (res, result = null, isGet = false)
=> {
  try {
    console.log(result);

    if (result && result.error) {
      this.sendError(res, result.error);
      return;
    }

    if (isGet && !result) {
      res.sendStatus(404);
```

```
      } else {
        res.send(result);
      }
    } catch (error) {
      this.sendError(error);
    }
  };
```

4.3.3 Frontend

Základem frontendové části mé aplikace je framework Vue.js.

4.3.3.1 MVVM

Vue.js je založen na architektuře MVVM (Model-View-ViewModel), která odděluje logiku aplikace od prezentační vrstvy. ViewModel obsahuje logiku a zpracovává data, zatímco View se stará o zobrazení a interakci s uživatelem. Modelem pak uvažujeme samotná data, tedy proměnné. Odpovídá tomu i syntaxe Vue.js komponent, ze kterých se projekt skládá. Každá komponenta je zvlášť soubor se sekcí pro definici zobrazení dat, chování komponenty a vzhled. Tato architektura umožňuje efektivní správu stavu aplikace.

4.3.3.2 Událostmi řízená architektura

Dále se dá frontend této aplikace považovat za takzvaně Událostmi řízený systém, jelikož interakce s uživatelem je úzce spojena s vyvoláváním událostí a reagováním na ně. Nejpoužívanějším příkladem je nejspíš kliknutí myší. Není to však jen na úrovni interakce s uživatelem, ale především na komunikaci mezi jednotlivými komponentami.

Příkladem může být odhlášení uživatele, které realizuje základní komponenta App.vue, když jiná komponenta vyhodnotí, že je uživatele nutno odhlásit, vyvolá událost odhlášení uživatele a komponenta App.vue tuto událost zachytí a reaguje na ni potřebnými kroky.

Pro vyvolávání a následné odchycení události v této práci používám jak nativně zabudovaný systém událostí, tak externí knihovnu Tiny-emitter.

4.3.3.3 Struktura projektu

Kód je opět rozdělen do souborů a složek dle jeho funkce. V kořenové složce jsou konfigurační soubory, několik pomocných scriptů (JavaScriptových souborů) a především soubor *index.html*. Ten je přístupovým bodem pro načtení souboru main.js, ve kterém se nastaví a vytvoří instance Vue.js aplikace. Dále ve složce src se nachází soubor App.js. Je základní komponentou, která spolu se soubory ve složce *router* a *views* obsahuje vše potřebné pro pomyslnou navigaci mezi jednotlivými stránkami aplikace. Stará se například i o přihlášení a odhlášení uživatele

a obsluhu chybových hlášek. Ve složce *components* se pak nachází všechny ostatní Vue.js komponenty (viz kapitola 4.1.6).

Všechny tyto soubory slouží k sestavení výsledných souborů určených pro distribuci (vygeneruje se jen html, css a javascript). Pro účely testování stačí v kořenové složce spustit příkaz *npm run dev*.

4.3.4 Responzivita

Nutno zdůraznit, že tato aplikace byla navrhována primárně pro zobrazování na monitorech stolních počítačů, obrazovkách notebooků a podobně velkých displejích, nikoliv tedy na mobilních zařízeních. Jelikož je v dnešní době téměř nemyslitelné nemít možnost zobrazení i v menším formátu, snažil jsem se komponenty vytvářet responzivní.[12] Kvůli složitosti implementace a povaze některých komponent není rozložení pro menší zařízení ideální a zdůrazňuji, že to nebylo prioritou. Nicméně v některých případech jde o jednoduchou změnu v rozložení a aplikace tím nijak neutrpěla.

[12]Reponzivní v tomto kontextu znamená, že rozložení a design se přizpůsobuje velikosti obrazovky, aniž by tím byla omezena funkcionalita či přístupnost.

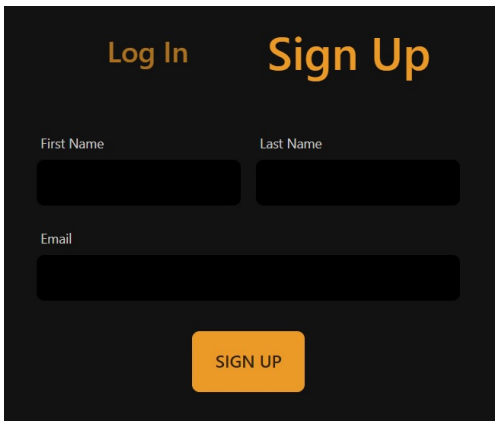
5 Uživatelská příručka

Nyní představím postupně všechny hlavní funkce aplikace. Na tomto místě je dobré poznamenat, že jsem se rozhodl udělat aplikaci v angličtině a pouze v tmavém režimu. Obě tato rozhodnutí souvisí se snadnějším vývojem a do budoucna bude obojí volitelné. Doporučovaným prohlížečem je Google Chrome [24], na kterém probíhalo primárně testování aplikace.

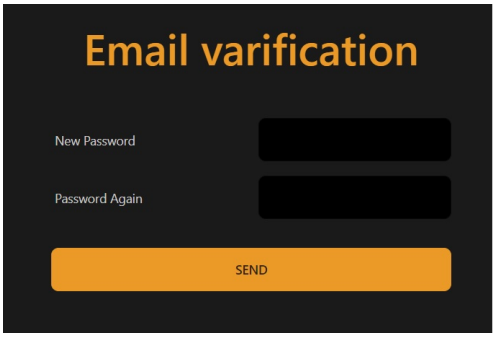
5.1 Registrace

Při registraci se zadá jméno, příjmení a e-mail v jednoduchém formuláři (obr. 1a). Proběhne kontrola e-mailu. Systém zaregistruje pouze uživatele s univerzitním e-mailem.

Na ten se pak odešle odkaz, který je platný 24 hodin. Po kliknutí na odkaz je uživatel přesměrován na obrazovku verifikace (obr. 1b), kde zadá, jaké chce mít heslo. Pokud vše proběhlo v pořádku, je uživatel přesměrován na domovskou obrazovku. Pokud uživatel nestihne odkaz využít do 24 hodin, tak je místo následného přihlášení upozorněn, že platnost odkazu již vypršela a byl mu na e-mail poslán nový odkaz. Pokud uživatel odkaz ztratí, je možné se zaregistrovat znovu, ale také až po 24 hodinách.



(a) Formulář pro registraci uživatele



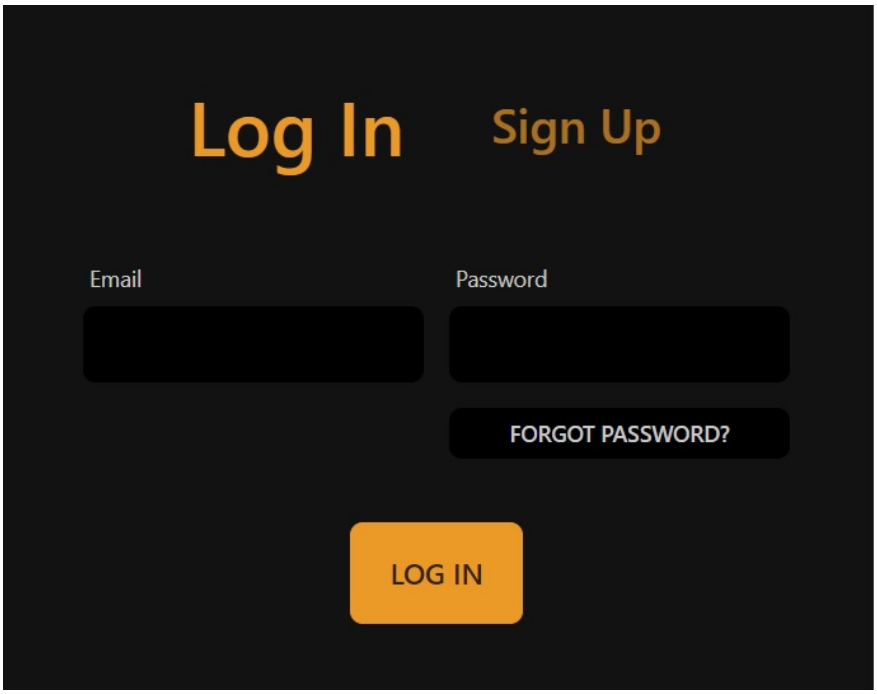
(b) Formulář pro verifikaci e-mailu uživatele

Obrázek 1: Formuláře pro Registraci a Verifikaci

Existuje ještě speciální „násilný“ způsob registrace, o kterém budu mluvit spolu s přidáváním studentů do kurzu (viz kapitola 5.5.3). Pro získání role učitele je pak nutné kontaktovat administrátora přes e-mail.

5.2 Přihlášení

Přihlášení probíhá klasicky zadáním e-mailu a hesla (obr. 2). Doba přihlášení je omezená na 2 hodiny od poslední interakce se serverem. Když je zadáno špatné heslo nebo neexistující e-mail, uživatel je vhodně upozorněn. Je zde i možnost pro resetování hesla.



Obrázek 2: Formulář pro přihlášení uživatele

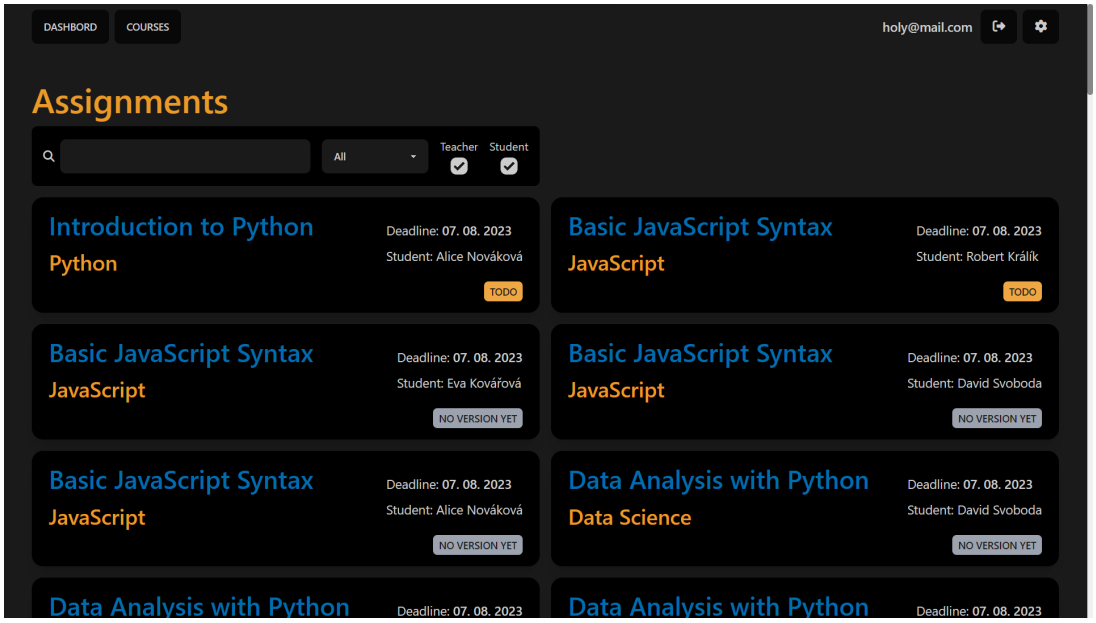
5.3 Úvodní stránka

Jsou zde zobrazeny všechny úkoly uživatele, (obr. 3). Pokud je uživatel pouze studentem, jsou zde pouze úkoly ke splnění, které student přijal v Detailu kurzu.

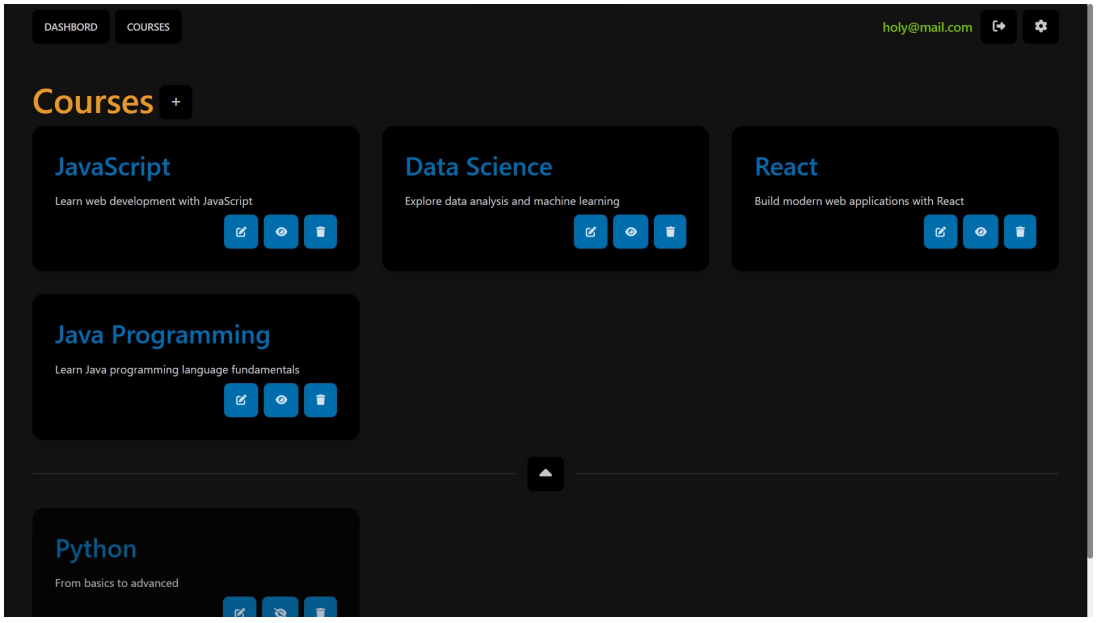
Když má uživatel roli učitele, stále může být i studentem v kurzu, který sám nezaložil. Je zde tedy jednoduchý filtr, který slouží především pro oddělení úkolů, které má uživatel vypracovat jako student a které má opravit jakožto učitel.

5.4 Kurzy

Zobrazení všech kurzů, ve kterých je uživatel ať už jako učitel nebo jako student. Pro všechny uživatele je zde možnost schovat si kurzy do části „pod čarou“. Pro učitele i možnost jejich vytvoření, editace a mazání (obr. 4). Kurzy jsou zobrazeny ve formě kartiček se základními informacemi. Po kliknutí na kartičku mimo tlačítka je uživatel přesměrován na detail Kurzu.



Obrázek 3: Stránka se všemi úkoly, které má uživatel opravit jako učitel nebo splnit jako student



Obrázek 4: Stránka se všemi kurzy které daný uživatel vytvořil nebo do nich byl přidán

5.5 Detail kurzu

Jde o nejvíce odlišnou obrazovku pro studenta a učitele. Skládá se ze tří částí pro učitele (obr. 5) a ze dvou pro studenta (obr. 6)

5.5.1 Tabulka prospěchu

Ve formě tabulky jsou zde vidět hodnocení jednotlivých úkolů, případně v jakém stavu se daný úkol nachází. Při kliknutí na buňku v tabulce se konkrétní úkol zobrazí. Učitel vidí celou tabulku, tedy prospěch všech studentů. Každý student vidí pouze svůj vlastní řádek této tabulky.

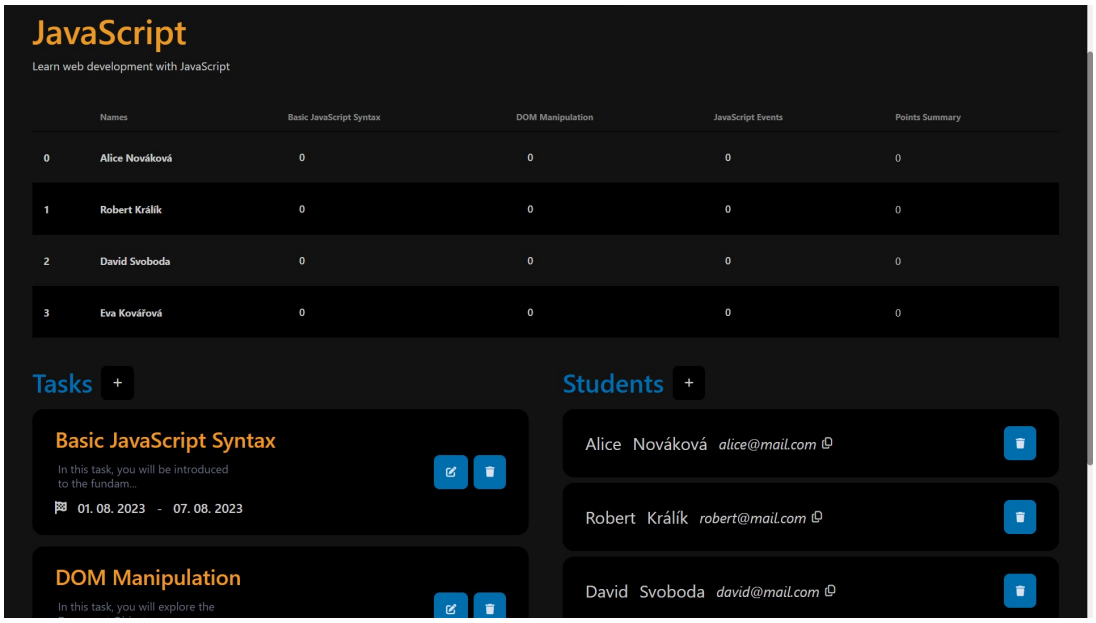
5.5.2 Seznam zadání

Přehled všech zadání, která byla v daném kurzu vytvořeny, je právě zde. Student kurzu zde může přijmout zadání a také si nechat zobrazit jeho popis. Pro učitele kurzu je pak možnost přidávání, editace, mazání a také možnost poslat zprávu do všech úkolů daného zadání.

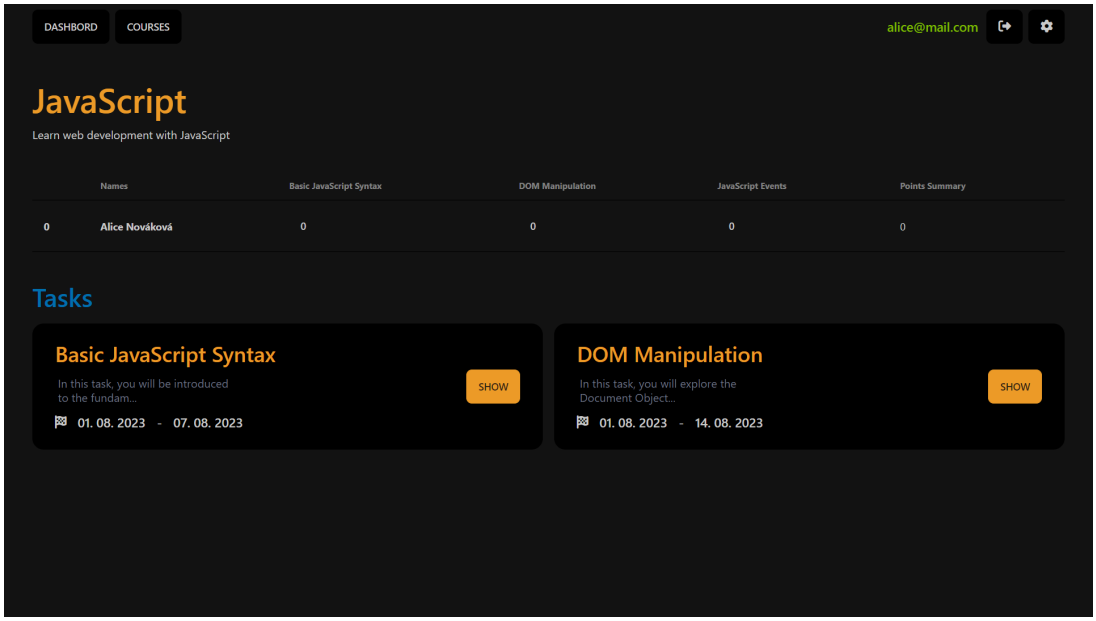
5.5.3 Seznam studentů kurzu

Učitel může přidávat studenty do kurzu a zase je z něj odebírat. Přidat studenty může učitel buďto pomocí vyhledávání ve výčtu všech zaregistrovaných uživatelů nebo může i nahrát soubor typu csv obsahujícího jména, příjmení a univerzitní e-maily studentů. Soubor musí být stejného formátu, jaký generuje IS Stag. Stačí ovšem položky *jméno*, *prijmeni* a *e-mail*.

Po nahrání tohoto souboru typu csv systém sám přidá ty studenty, kteří jsou pod daným e-mailem již zaregistrováni a ty, kteří zaregistrováni nejsou, zaregistruje (pošle jim ověřovací odkaz na e-mail) a také je do kurzu přidá. Pro studenta tento (dalo by se říci násilný) způsob zaregistrování vypadá tak, že mu jen přijde odkaz na e-mail, na který klikne. Poté zadá heslo a už je zaregistrovaný i přidaný do kurzu. Tento seznam je pro studenty zcela skrytý.



Obrázek 5: Detail kurzu z pohledu učitele



Obrázek 6: Detail kurzu z pohledu žáka

5.6 Detail Úkolu

Vlevo nahoře je název zadání, které úkolu náleží.

Student i učitel vidí vpravo nahoře počet získaných bodů, z kolika maximálních student získal. Pod ním je pro učitele k dispozici evaluace úkolu spolu s nastavením jeho stavu (statutu). Učitel má na výběr z těchto stavů: „FINISHED“, „FAILED“, „CHECKED“, „TODO“. Dále se může úkol nenacházet ani v jednom z nich. To se pak v aplikaci projevuje jako „NO VERSION YET“. Je mu umožněno si zcela libovolně vybrat, který stav bude úkol mít. Následující vysvětlení významu jednotlivých stavů považuji za doporučení či osvětlení toho, jak byly stavy myšleny ze strany autora.

- Splněný(FINISHED). Tedy úkol je splněný, ohodnocený a ani na jednu ze stran se nečeká. Je automaticky nastaven, když je úkol ohodnocen více než 0 body.
- Nesplněný(FAILED). Úkol nebyl splněn a je tedy ohodnocen 0 body. Při nastavení 0 bodů se tento stav opět sám nastaví.
- Opravený(CHECKED). To znamená, že se čeká na studentovu odpověď v Chatovacím okně či na další studentovu verzi. Tento stav se sám nenastavuje, aby nedošlo k chybnému vyhodnocení, že učitel již úkol zcela opravil.
- K Opravení(TODO). Tento stav se sám nastaví, jakmile student pošle zprávu, či odešle novou verzi.

Pro studenta je zde počet získaných bodů a především odevzdání úkolu, kde se dá nahrát buďto celá složka (struktura podsložek a souborů bude zachována), nebo přímo soubory. Protože systém vnitřně využívá Git (viz kapitola 4.1.8), není povoleno nahrávat soubory, které jsou pro tento systém typické (.gitignore, gitconfig, ...). Tyto soubory jsou automaticky vyfiltrovány (na serveru nedojde k jejich uložení). Pokaždé, když student vloží soubory nebo složku, vše se uloží a vytvoří se tím nová verze. Všechny verze se uchovávají a je možné v nich zpětně listovat.

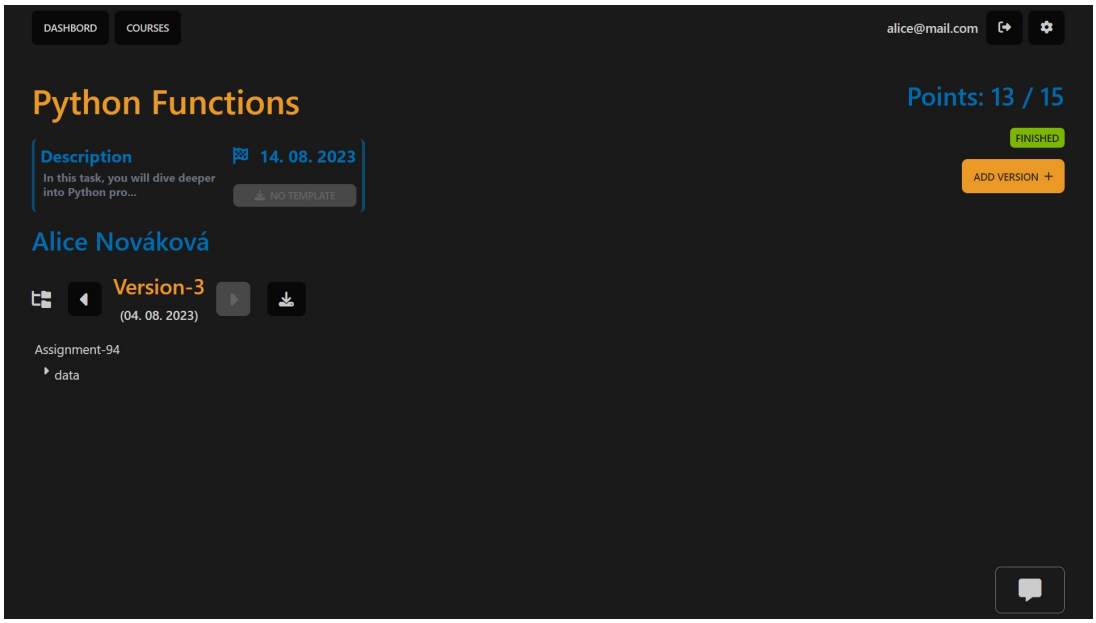
Pro názorné porovnání pohledu učitele a studenta na detail úkolu přikládám obě varianty. Pohled studenta (obr. 8) a pohled učitele (obr. 7).

5.6.1 Prohlížení souborů

Pod názvem úkolu se nachází několik tlačítek. První z nich skrývá a odkrývá rozcestník souborů a složek, které náleží vybrané verzi. Další slouží pro posouvání se časově dopředu a dozadu ve verzích. Poslední tlačítko stáhne všechny soubory verze v jako soubor typu zip.

Rozcestník jako takový pak umožňuje jednoduše prohledávat v souborech verze podobným způsobem, jako je tomu ve známých vývojových prostředích. Po vybrání konkrétního souboru se zobrazí v jednom z oken pro zobrazení souborů

Obrázek 7: Detail úkolu z pohledu učitele

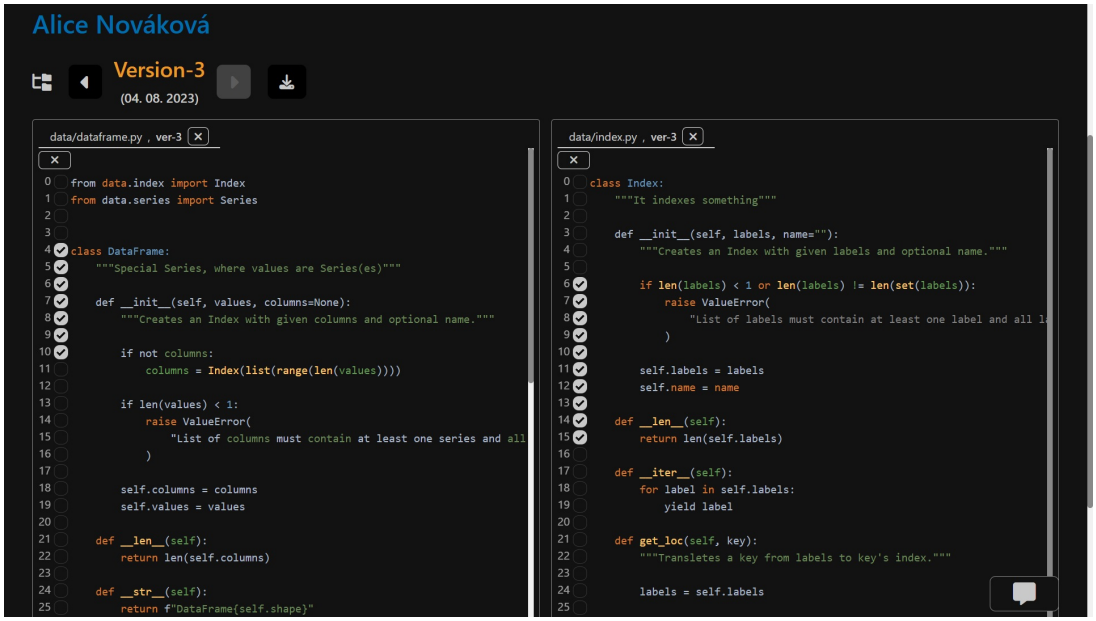


Obrázek 8: Detail úkolu z pohledu studenta

(obr. 9). Pro zjednodušení implementace jsou okna dvě, ale systém je navržený pro případné navýšení tohoto počtu.

Soubor se v okně dá zobrazit, jen pokud jde o textový soubor. Pokud by navíc šlo o zdrojový kód některého z podporovaných jazyků, barevně se zvýrazní syntaxe. Na levé straně každého řádku je checkbox[13] pro jeho označení. Tímto

[13] Jde o známý grafický prvek, který značí hodnotu ano či ne.



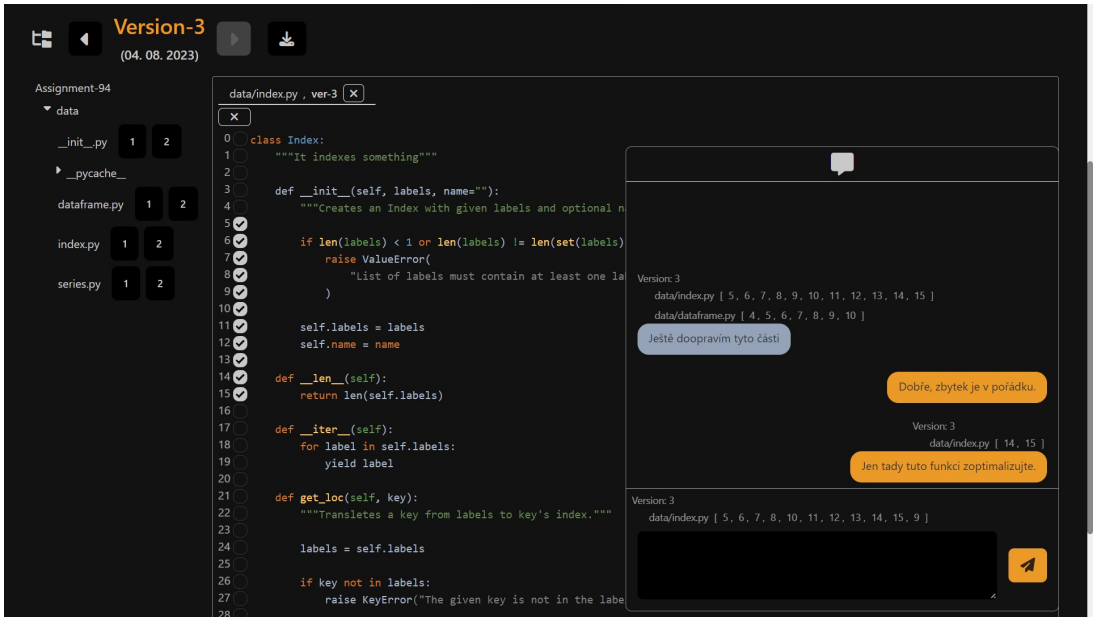
Obrázek 9: Zobrazení souborů v detailu úkolu

způsobem se dá zaznamenat, ke které části souboru se vztahuje zpráva odeslaná přes Chatovací vlákno.

5.6.2 Chatovací vlákno

Odkrývá se tlačítkem vpravo dole, kde se pak i nachází. Skrývá se pak kliknutím na horní okraj zobrazeného okna s chatovacím vláknem. Student s učitelem si mohou posílat zprávy právě díky této komponentě. Pokud při odeslání zprávy má uživatel označené řádky jakýchkoliv souborů, je tato informace uložena se zprávou jako dodatečná data. Přehled o tom, které řádky jsou momentálně označeny, má uživatel nad polem pro text nové zprávy v jakémsi „bloku referencí“. Toto zobrazení je poté nad odeslanou zprávou (obr. 10). Po kliknutí na některý soubor v bloku referencí se zobrazí příslušný soubor dané verze v zobrazovacím okně a označí se i příslušné řádky.

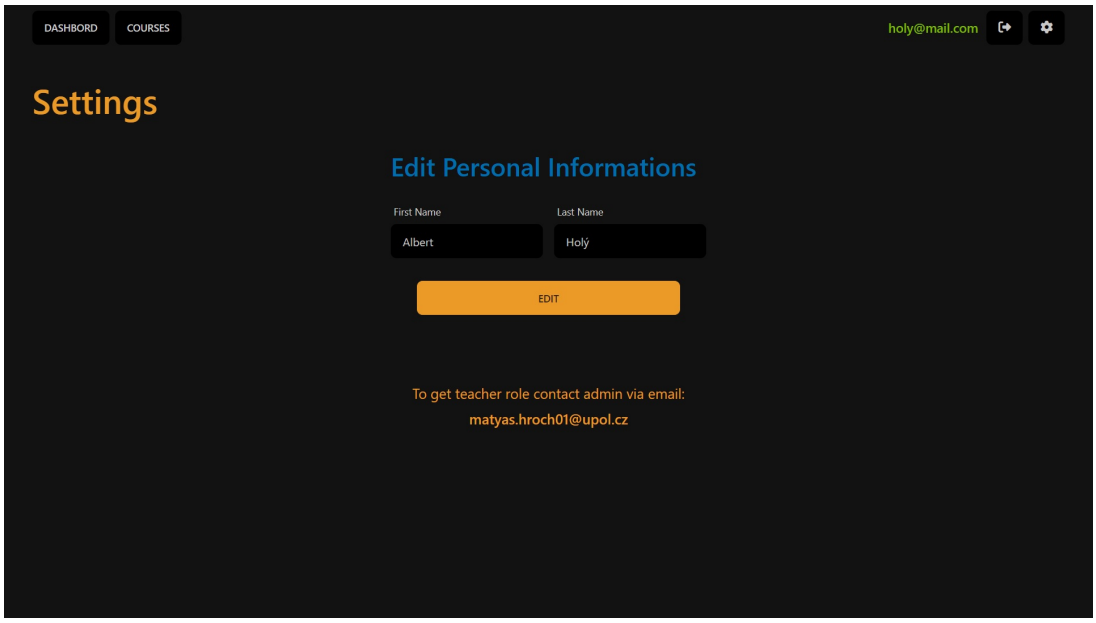
V tomto vláknu se také zobrazují všechny zprávy, které učitel poslal přes detail zadání do všech úkolů daného zadání. Všechny zprávy (včetně hromadných) jsou pak seřazeny shora dolů podle data a času vytvoření.



Obrázek 10: Chatovací vlákno s několika označenými řádky

5.6.3 Nastavení

V nastavení je možnost změny jména a příjmení (obr. 11), především kvůli možnosti opravy překlepů během registrace. Nachází se zde i kontaktní údaje na administrátora.



Obrázek 11: Nastavení osobních údajů a kontakt na administrátora

Závěr

V rámci této práce byla vytvořena webová aplikace pro zadávání a odevzdávání úkolů. Specializuje se na úkoly programovacího rázu. Umí vhodně zobrazit zdrojový kód různých programovacích jazyků a umožňuje dát zpětnou vazbu s odkazem na vybrané řádky v konkrétním souboru. Byla vytvořena pomocí moderních technologií založených na jazyce JavaScript.

Přidávání doplňků do vývojových prostředí se ukázalo být náročnější, než se původně předpokládalo. Vývojová prostředí jsou často složitá, a tak je potřeba porozumět jejich architektuře a API, aby byl doplněk vytvořen efektivně a bezchybně. Doplňky mají přístup k interním funkcím a datům vývojového prostředí, takže musí být velmi dobře navrženy a testovány, aby nedocházelo k nežádoucím konfliktům nebo bezpečnostním hrozbám. Bylo by nutné zajistit komunikaci se serverem aplikace (včetně posílání souborů) a zároveň zobrazení dat uživateli a interakci s ním. Z toho vyplývá, že vývoj doplňku se náročností blíží vývoji frontendu této aplikace. Toto rozšíření jsem zvážil a rád bych je později doplnil. Díky zvolené klient-server architektuře je k tomu část systému již přichystána. Studenti by navíc jistě ocenili možnost odevzdat úkoly přímo z vývojového prostření, ve kterém je vypracují.

Dále byla na stole otázka nahrávání automatických testů, tedy umožnit některým uživatelům vkládat kód, který bude na serveru spouštěn. Implementovat takovouto funkcionalitu se ukázalo jako příliš náročné. Muselo by se zajistit, že spuštěné testy neprovádějí žádnou činnost, kterou by ohrozily systém a že v nich není žádná chyba. Obě tyto varianty by mohly být tak nepředvídaným elementem, že způsobí pád systému nebo jeho uvedení do nekonzistentního stavu. Pak jsou zde požadavky na výkon. Kdyby měl každý student možnost testovat si své úkoly, musel by se zajistit dostatečný výkon serveru. Tyto a další problémy by bylo vhodné prostudovat, než by se rozšíření implementovalo. Tato problematika z mého pohledu vede na samostatnou práci.

Případná rozšíření v blízké době by mohla zahrnovat přepínání mezi tmavým a světlým režimem, volbu jazyka a také uložení různých preferencí uživatele. Tím je myšleno například nastavení výchozích hodnot při vytváření zadání.

Conclusions

The aim of this work was to develop a web application for task assignment and submission. It specializes on programming tasks. It is able to display the source code in various programming languages and to provide feedback with references to selected lines in particular files. This application was created by using modern technologies based on the JavaScript language.

Integration of add-ons into the development environment turned out to be even more challenging than I had previously expected. Development environment is often very complex, so it is necessary to understand the architecture and APIs properly to create the add-ons in the most efficient way and avoid any failure. Add-ons have access to internal functions and data of the development environment, so they have to be accurately designed and tested to avoid undesirable conflicts or security threats. It would be essential to ensure communication with the application server (including file uploads) as well as to display the data to users to facilitate further interaction. It follows that the add-on development is almost as demanding as to develop the frontend of this application. In the future I would like to create this extension. Due to the selected client-server architecture, a part of the system has already been prepared for this. In addition to this, students would definitely appreciate the ability to submit the tasks directly from the development environment in which they create them.

Furthermore, there was the possibility to upload automatic tests which means to allow some users to submit code that would be executed on the server. However, it turned out that it would be extremely demanding to implement such functionality. We would have to ensure that the executed tests are not engaged in any activity that could harm the system and that they do not include any failure. Both possibilities might cause system breakdown or inconsistent states. Furthermore, we have to consider the performance. If every student had the opportunity to test their tasks, we would have to ensure sufficient server performance. These and other issues ought to be thoroughly examined prior to the implementation of extension which would be the matter of further study.

Potential upcoming extensions could include for example switching between dark and light modes, language selection or the ability to save various user preferences e.g. setting basic values when the assignment is created.

A Obsah elektronických dat

Součástí práce je elektronická příloha dat v systému katedry informatiky s následující strukturou:

`text/`
: Adresář obsahuje text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce. Obsahuje také ZIP soubor obsahující všechny soubory potřebné pro bezproblémové vytvoření tohoto PDF dokumentu.

`src/`
: Složka se všemi zdrojovými kódy aplikace. Pro frontend je to složka tailwind_vue. Pro backend je to složka real_node. Podrobnosti v souboru README.txt.

`README.txt`
: Je textový soubor, který popisuje postup pro lokální spuštění aplikace, popřípadě její konfiguraci.

Literatura

[1] GitHub. *GitHub Classroom* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://classroom.github.com/⟩.

[2] Microsoft. *Microsoft Teams* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.microsoft.com/en-us/microsoft-teams/group-chat-software⟩.

[3] Google. *Google Classroom* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://classroom.google.com/⟩.

[4] Consortium, World Wide Web. *HTML* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/HTML⟩.

[5] Consortium, World Wide Web. *CSS* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/CSS⟩.

[6] Mozilla. *JavaScript* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/JavaScript⟩.

[7] Foundation, Node.js. *Node.js* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://nodejs.org/⟩.

[8] npm, Inc. *npm* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.npmjs.com/⟩.

[9] You, Evan. *Vue.js* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://vuejs.org/⟩.

[10] Consortium, SQLite. *SQLite* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.sqlite.org/⟩.

[11] PostgreSQL Global Development Group. *PostgreSQL* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.postgresql.org/⟩.

[12] Oracle Corporation. *MySQL* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.mysql.com/⟩.

[13] SQLite Consortium. *SQLite* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.sqlite.org/index.html⟩.

[14] GitHub, Inc. *GitHub* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://github.com/⟩.

[15] Highlight.js Contributors. *highlighter.js* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://highlightjs.org/⟩.

[16] Corgan, Scott. *tiny-emitter* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://github.com/scottcorgan/tiny-emitter⟩.

[17] Wathan, Adam; Reinink, Jonathan; Hemphill, David; Schoger, Steve. *Tailwind CSS* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://tailwindcss.com/⟩.

[18] DaisyJS. *Daisy UI* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://daisy.js.org/⟩.

[19] Express Contributors. *Express* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://expressjs.com/⟩.

[20] Multer Contributors. *Multer* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.npmjs.com/package/multer⟩.

[21] Sequelize Contributors. *Sequelize* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://sequelize.org/⟩.

[22] King, Steve. *simple-git* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://github.com/steveukx/git-js⟩.

[23] Auth0. *jsonwebtoken* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.npmjs.com/package/jsonwebtoken⟩.

[24] Google Chrome. *Google Chrome*. 2008. Dostupný z: ⟨https://www.google.com/chrome/⟩.

[25] Microsoft. *Microsoft 365* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.microsoft.com/microsoft-365/⟩.

[26] Microsoft. *Microsoft Word* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.microsoft.com/en-us/microsoft-365/word⟩.

[27] Microsoft. *Microsoft Excel* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.microsoft.com/en-us/microsoft-365/excel⟩.

[28] Microsoft. *Microsoft PowerPoint* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.microsoft.com/en-us/microsoft-365/powerpoint⟩.

[29] Microsoft. *Microsoft OneNote* [online]. [cit. 2023-8-4]. Dostupný z: ⟨https://www.microsoft.com/en-us/microsoft-365/onenote⟩.