FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Trainer - Webový portál pro podporu výuky programování
Student:	Bc. Ondřej Wrzecionko
Vedoucí:	Ing. Jan Matoušek
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Programování patří k důležitým kompetencím v aktualizovaných rámcových vzdělávacích programech pro základní vzdělávání i gymnázia v oblasti Informatika. Výuka programování je též důležitá na Fakultě informačních technologií ČVUT v Praze, zejména v rámci předmětů Programování a algoritmizace 1 a 2 (BI-PA1.21, BI-PA2.21). Cílem této práce je vyvinout webový portál, který by zvýšil efektivitu tohoto procesu, primárně po stránce ověřování znalostí, trénování dovedností studentů a poskytování potřebné zpětné vazby.

Pokyny k vypracování:

1) Vhodnými postupy a z různých úhlů pohledu proveďte rešerši problematiky praktické výuky programování.
2) Proveďte analýzu pro uvažovaný webový portál s přihlédnutím k potřebám programovacích předmětů prvního ročníku FIT ČVUT.
3) Navrhněte webový portál a zvolte vhodné technologie. Přihlédněte k potřebám budoucího rozvoje v rámci dalších studentských projektů.
4) Vytvořte prototyp, na kterém ověříte svůj návrh.
5) Vzniklý prototyp provozujte na infrastruktuře FIT ČVUT a využijte jej ve výuce uvedených programovacích předmětů.
6) Vhodnými postupy iterujte návrh i implementaci portálu.
7) Portál řádně otestujte a zdokumentujte pro všechny uživatelské role.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 14. února 2024 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Trainer – Webový portál pro podporu výuky programování

Bc. Ondřej Wrzecionko

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jan Matoušek

8. května 2024

Poděkování

Chtěl bych poděkovat především svému vedoucímu, Ing. Janu Matouškovi, za jeho podporu a pomoc při tvorbě této diplomové práce. Nadále bych chtěl poděkovat všem studentům a učitelům, kteří pomáhali při testování systému a v neposlední řadě rodině za její podporu při celém magisterském studiu.

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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. května 2024

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Wrzecionko, Ondřej. *Trainer – Webový portál pro podporu výuky programování.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Abstrakt

Práce se zabývá analýzou, návrhem, implementací, nasazením a testováním webového portálu pro podporu výuky programování ve dvou iteracích. Portál se skládá z frontendu implementovaného v technologii Vue.js a podpůrného backendu v technologii Spring Web. Systém umožňuje učitelům vytvářet programovací úlohy a organizovat je podle témat do lekcí, týdnů a kurzů. Studenti úlohy vyplňují a můžou svá řešení v případě potřeby konzultovat. Systém umožňuje automatické i manuální vyhodnocení úloh, přehled pokroku studentů v lekci i anonymizované promítání studentských řešení. Systém je nasazen na infrastruktuře FIT ČVUT a byl za první dva semestry používán ve 2 předmětech a 16 paralelkách celkem 768 studenty a 17 učiteli, kteří v něm vytvořili přes 350 úloh. Studenti i učitelé systém hodnotí pozitivně a v budoucnu je plánováno jeho využití i v dalších předmětech a na dalších školách.

Klíčová slova výukový portál, výuka programování, REST API, Vue.js, JavaScript, Spring Web, Kotlin, WASM, C++

Abstract

The thesis deals with analysis, design, implementation, deployment and testing of web portal to support teaching programming, done in two iterations. Portal consists of frontend implemented in Vue.js technology and supporting backend in Spring Web technology. System enables teachers to create programming tasks and organise them in lessons, weeks and courses. Students complete the tasks and if needed, they can consult their solutions. System provides automatic or manual task evaluation, summary of student progress in lessons and anonymized projection of student solutions. System is deployed on FIT CTU infrastructure and throughout first two semesters, it has been used in 2 subjects, 16 parallels by 768 students and 17 teachers, who have created over 350 tasks. Both students and teachers rate the system positively and in future, it is planned to be used in another subjects and on other schools.

Keywords education portal, teaching programming, REST API, Vue.js, JavaScript, Spring Web, Kotlin, WASM, C++

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

Programování je v dnešní době jednou ze základních součástí výukových programů na základních, středních i vysokých školách zaměřených na informatiku. Aktuální způsob výuky s sebou ale nese dva hlavní problémy: pasivitu studentů a příliš velké nároky na vyučujícího.

První problém, pasivita studentů, je často způsobený stylem výuky, a to zvláště na vysokých školách, kde je z důvodu vysokého počtu studentů ve studijních skupinách nemožné studenty aktivně zapojit do společného programování. Pro nadané studenty je tempo příliš pomalé, pomalejší studenti zase nestíhají, a tak rezignují a stávají se pasivními pozorovateli.

Druhý problém je způsoben více příčinami – kromě nedostatku učitelů jde také o náročnost přípravy nových úloh, jejich zadávání studentům a složité procesy při poskytování zpětné vazby (posílání kódu přes e-mail, problémy s kompatibilitou, bezpečnostní rizika).

Vytvoření webového portálu pro podporu výuky programování tak přinese nový, moderní a efektivní způsob, jak zapojit studenty do výuky a to tím, že umožní každému postupovat vlastním tempem s možností pokračovat doma. Webový portál také zmírní problém nároků na vyučujícího, jelikož mu usnadní vytváření nových úloh, jejich zadávání studentům a především učiteli umožní na jednom místě studentovi úlohu ohodnotit a okomentovat bez posílání e-mailů a stahování potenciálně nebezpečných souborů. Dalšími problémy, jako jsou příprava úloh pro konkrétní předměty, se práce zabývat nebude.

K učení jsem se dostal poprvé už ve čtvrtém semestru bakalářského studia, kdy jsem dostal nabídku učit předmět Programování a algoritmizace 2. V průběhu dalších semestrů jsem v učení pokračoval, jeden rok i na gymnáziu, a mohl jsem tak tyto problémy pozorovat v praxi. Když za mnou minulý rok přišel vedoucí s nápadem na tvorbu systému, který by výuku programování usnadnil, neváhal jsem dlouho a nabídku jsem přijal.

Ve své práci projdu celým procesem návrhu systému od analýzy stávajících řešení, sběru požadavků, přes návrh architektury, uživatelského rozhraní, výběr vhodných technologií, implementaci, nasazení i testování, a to ve více iteracích.

Kapitola 1

Cíl práce

Cílem diplomové práce je vytvoření webového portálu pro podporu výuky programování.

Nejprve provedu analýzu stávajících způsobů výuky programování. Na základě této analýzy stanovím požadavky pro nově vznikající systém. Následně navrhnu architekturu systému a jeho uživatelské rozhraní, zvolím vhodné technologie a naprogramuji prototyp systému, který nasadím na školní infrastrukturu. Zde prototyp i uživatelské rozhraní otestuji a na základě zjištěných chyb provedu iterativně vylepšení prototypu. Výslednou verzi prototypu pak zdokumentuji pro budoucí uživatele i programátory.

Kapitola 2

Analýza

V této kapitole nejprve zanalyzuji problém výuky programování a vyhodnotím jeho aktuální stav, především na FIT ČVUT v Praze při výuce programovacích předmětů prvního ročníku. Na základě zjištěných informací stanovím funkční a nefunkční požadavky, vůči kterým porovnám existující řešení. V závěru kapitoly pak zhodnotím, zda bude mít tvorba systému přínos a jaký.

2.1 Aktuální stav

2.1.1 Základní a střední školy

Z důvodu neustálého rozvoje digitálních technologií, ke kterému také značně přispěla koronavirová epidemie v letech 2020 až 2022, má výuka informatiky a programování na školách stále větší význam. To bylo také důvodem, proč Ministerstvo školství v roce 2021 představilo nový Rámcový vzdělávací program „Nová informatika“. Tento program představuje změnu v tom, jak se učí informatika – místo tradičního učení počítačových dovedností, jako je tvorba dokumentů, tabulek a prezentací v nástrojích Microsoft Office, se tak budou žáci už od základní školy učit programování a algoritmizaci. [1]

Na základních a středních školách výuka probíhá typicky v menších skupinách čítajících přibližně 15 žáků a je v kompetenci jednoho vyučujícího, který se řídí osnovou rámcového vzdělávacího programu a připravuje náplň hodin. Je tak především na učiteli, jak bude programovací dovednosti vyučovat.

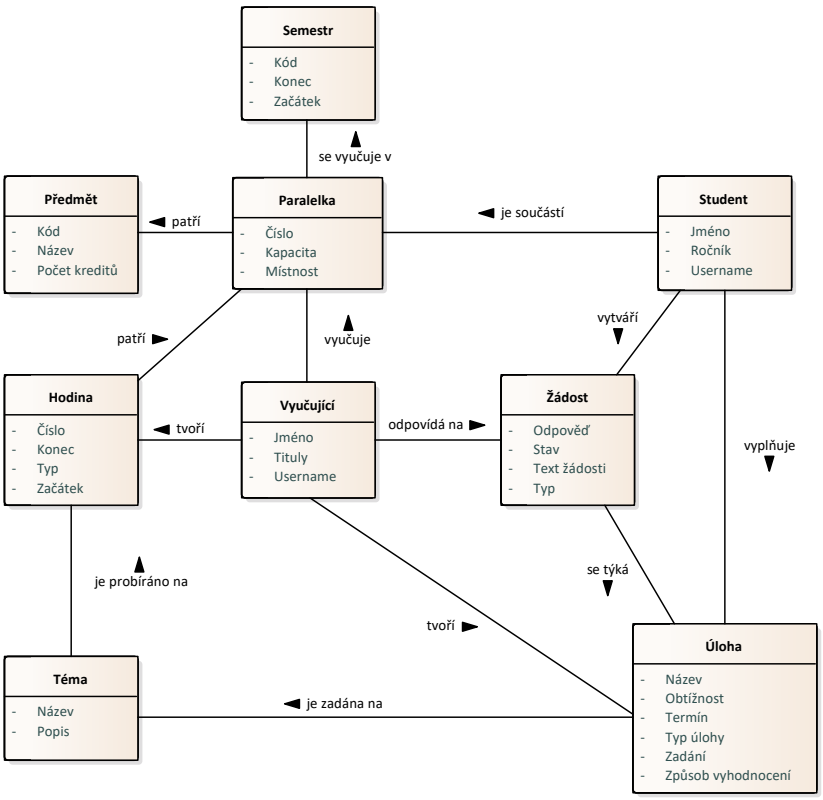
Dle Mgr. Radka Vystavěla, Ph.D. [2] je třeba dbát na čtyři hlavní věci: *aktuálnost*, tedy aby se vyučovaly současné technologie; *přitažlivost*, neboli tvoření moderního uživatelského rozhraní, programů, které jsou zajímavé a co nejblíže reálnému světu; *pokrytí literaturou*, aby měli studenti a žáci učebnici a materiály, ze kterých se můžou učit i doma a *praktické zvládnutí problematiky* – programování totiž není pouze o tom bezhlavě opisovat kód od učitele, ale o tom rozumět tomu, co člověk dělá.

Že je způsob, kterým učitel připraví hodinu, pro její průběh a dopad na žáky zásadní, jsem si mohl ověřit i v průběhu školního roku 2022 / 2023, kdy jsem vyučoval informatiku na Gymnáziu Nad Kavalírkou [3]. Při výuce podle 15 let starých materiálů byli žáci znuděni, ztráceli pozornost a vyrušovali. Když jsem vytvořil aktuální materiály, kvízy a moderní cvičebnici, aktivita v hodinách stoupla, žáci se do výuky aktivně zapojovali a hodiny je bavily.

2.1.2 Vysoké školy

Z průzkumu mezi studenty ostatních škol jsem zjistil, že na většině vysokých škol napříč republikou (FEL ČVUT, FIT VUT, MUNI, OSU, VŠB) probíhá výuka programování především v prvních ročnících ve velkých předmětech, které navštěvují povinně všichni studenti prvního ročníku, kterých bývá společně s opakujícími studenty okolo 500–1000. Výuka se tak skládá z přednášek, kde studenti poslouchají teoretický výklad učitele a programovacích cvičení – seminářů v přibližně 20členných skupinách s jedním pedagogem.

Na FIT ČVUT se vyučuje programování v prvních ročnících ve dvou velkých předmětech, Programování a algoritmizace 1 a 2 (dále jen PA1 [4], PA2 [5]). Kromě přednášek se výuka skládá i z proseminářů, kde učitel programuje a snaží se do toho zapojit i studenty. Cvičení zde probíhají narozdíl od většiny ostatních škol ve skupinách 48 studentů se dvěma pedagogy.



Obrázek 2.1: Doménový model vysokých škol

Z cvičení se ale bohužel v aktuálním formátu často stávají další přednášky a proseminář. Částečně na vině jsou studenti samotní, kteří z různých důvodů na přednášky a proseminář nechodí, částečně je to vina učitelů, ale ve velké míře je to taky nedostatek „pedagogických prostředků“ a pomůcek pro výuku.

Pro výuku v předmětech PA1, PA2 se používají dva hlavní zdroje – *e-learning* a *cvičebnice*.

E-learning obsahuje jednotlivé kapitoly a podkapitoly s učivem, rozděleny podle jednotlivých témat. Jedná se o skvělý materiál pro přečtení a rozšíření teoretických znalostí, studenti mají možnost si překopírovat jednotlivé kódy, chybí zde ale možnost, aby si studenti prakticky vyzkoušeli, jestli opravdu daným tématům rozumí. E-learning se na cvičeních většinou nepoužívá, a je doporučen pouze jako doplňkový materiál pro studenty na domácí přípravu.

Cvičebnice je databáze, ve které je pro každé cvičení uvedeno pár teoretických otázek a následně příklady, které mají pomoct studentům při jejich naprogramování pochopit dílčí problematiku.

Průběh cvičení je pak typicky následující: Učitel se frontálně zeptá na teoretické otázky, na které odpoví pár nejzdatnějších studentů, kteří se nestydí přihlásit a odpovědět nahlas před celou třídou, pak probere jednodušší příklady stylem proseminář (učitel nebo pokročilejší student programuje na tabuli lehčí příklad, ostatní pozorují) a studentům se na závěr zadají složitější příklady. Pokročilejší studenti jsou pak motivováni k vypracování těžkých, bonusových úloh, za které dostanou bonusový bod do výsledné klasifikace ze semestru.

2.2 Vyhodnocení aktuálního stavu

Aktuální stav na FIT ČVUT bohužel vede k více problémům:

- jsou studenti, kteří při odpovídání na teoretické otázky znají správnou odpověď, ale nechtějí / stydí se ji říct nahlas před učebnou plnou 47 dalších lidí, které neznají
- při společném programování lehčího příkladu hodně studentů nestíhá tempo, kterým učitel nebo pokročilejší student programuje, a uchýlí se tak k pasivnímu pozorování nebo bezmyšlenkovitému přepisování kódu, který vidí na projektoru, aniž by mu rozuměli
- při samostatném programování úloh student nedostane od učitele žádnou zpětnou vazbu ohledně kvality jeho kódu, což se následně negativně promítá do odevzdávaných řešení domácích úloh
- studentům, kteří odevzdávají těžké úlohy za bonusové body, učitel testovací data vymýšlí většinou na místě pro jejich program tak, aby ho nějak rozbil a ztrácí tak v průběhu hodiny čas, který by mohl věnovat konzultacím

Cílová skupina studentů je početná – předměty PA1 a PA2 si podle Ankety ČVUT [6] jen za minulých 6 semestrů zapsalo postupně 822, 604, 893, 614, 1032 a 772 studentů.

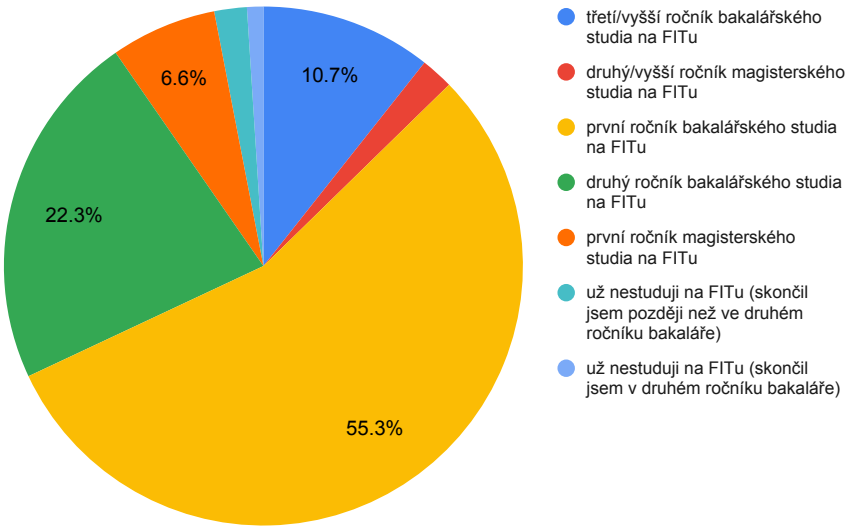
Nutno dodat, že tyto skupiny nejsou disjunktní, studenti, kteří předmět nezvládnou na první zápis, si jej můžou zapsat podruhé a pokud restartují studium, tak dokonce i potřetí nebo počtvrté.

2.2.1 Studentský dotazník

Z důvodu vysokého počtu studentů v cílové skupině jsem zvolil pro analýzu metodu *dotazníku*. Vytvořil jsem dotazník na platformě Google Forms. Tato platforma umožňuje snadné vytváření a sdílení online průzkumů včetně analýzy odpovědí v reálném čase [7]. Dotazník obsahuje otázku na ročník studenta pro lepší zařazení a následně pro každý předmět (PA1 / PA2) otázku, zda daný student předmět absolvoval úspěšně nebo neúspěšně. Následně dotazník obsahuje podmíněnou sekci pro studenty, kteří předmět na první zápis nezvládli s otázkou, kde se ztratili a co (by) jim pomohlo při druhém zápisu předmětu. Další sekce se týká obsahu předmětu – čeho se student zúčastnil, jak byla na cvičeních učena teorie a programování a na závěr je zde otázka s možností výběru funkcionalit, které by měl nový systém obsahovat. Kompletní podoba dotazníku je součástí přílohy Dotazníky.

2.2.2 Vyhodnocení dotazníku

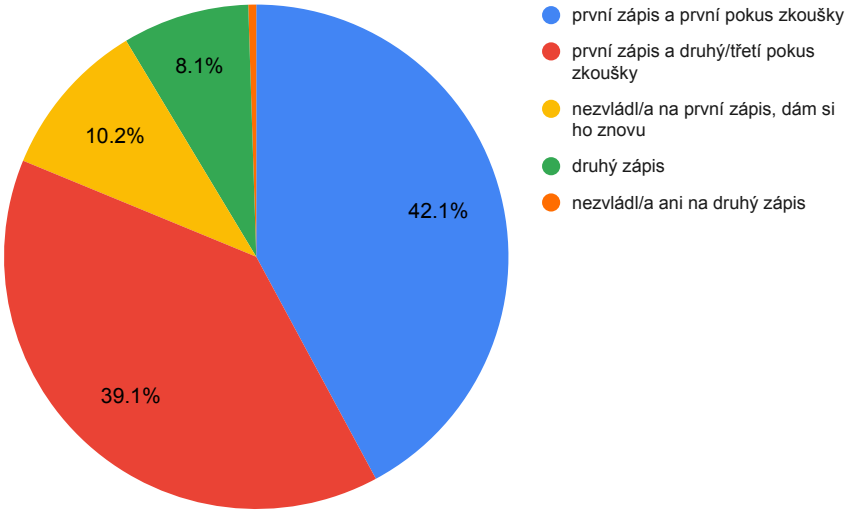
Dotazník jsem na začátku června 2023 zveřejnil na neoficiálním studentském Discordu FIT ČVUT [8], kde je přes 6 000 uživatelů, z nichž většinu tvoří aktuální studenti nebo absolventi. V průběhu června jsem nasbíral 197 odpovědí. Rozložením odpovědi odpovídaly očekávání – přibližně polovina byla od studentů prvního ročníku bakalářského studia, dalších 33 % od vyšších ročníků a zbylých 17 % od magistrů a/nebo (ne)úspěšných absolventů.



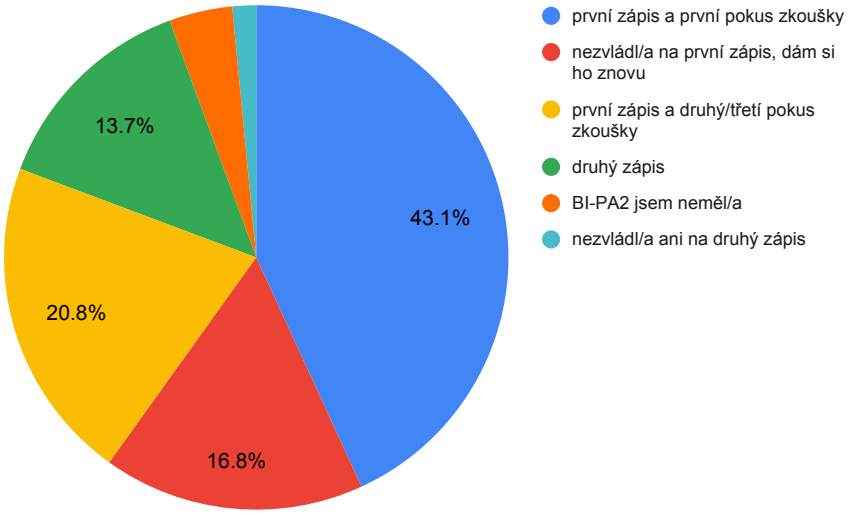
Obrázek 2.2: Přehled studentů

Předmět PA1 zvládlo na první zápis 81 % dotázaných, zatímco PA2 pouhých 64 % tázaných. Nutno podotknout, že tato procenta nereflektují skutečnou průchodnost, která je podle dat z Ankety ČVUT [6] u PA1 dlouhodobě okolo 45 % a u PA2 přibližně 40 %.

Důvodem je fakt, že je obecně problém získat kontakt na studenty, kteří školu ukončili během prvního roku studia, dochází zde tedy k určitému zkreslení.



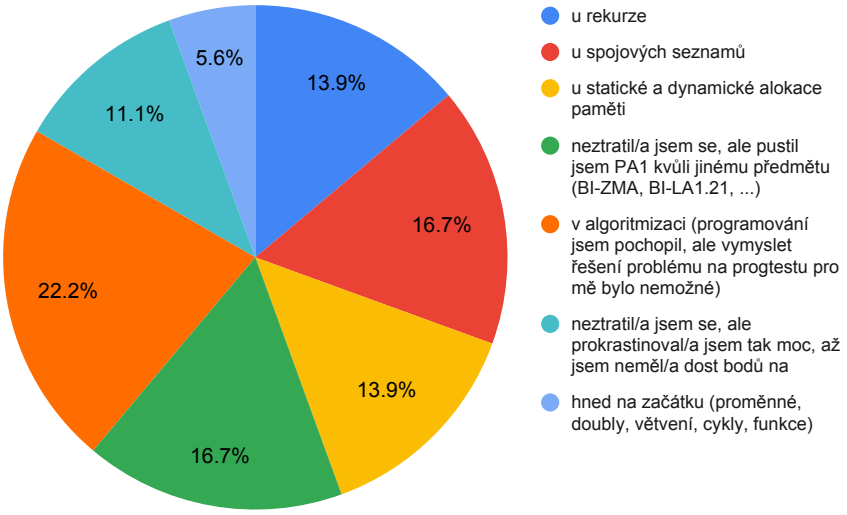
Obrázek 2.3: Graf úspěšnosti v předmětu PA1



Obrázek 2.4: Graf úspěšnosti v předmětu PA2

Mezi důvody, proč studenti předměty nezvládli, jsou nejčastěji zastoupeny následující:

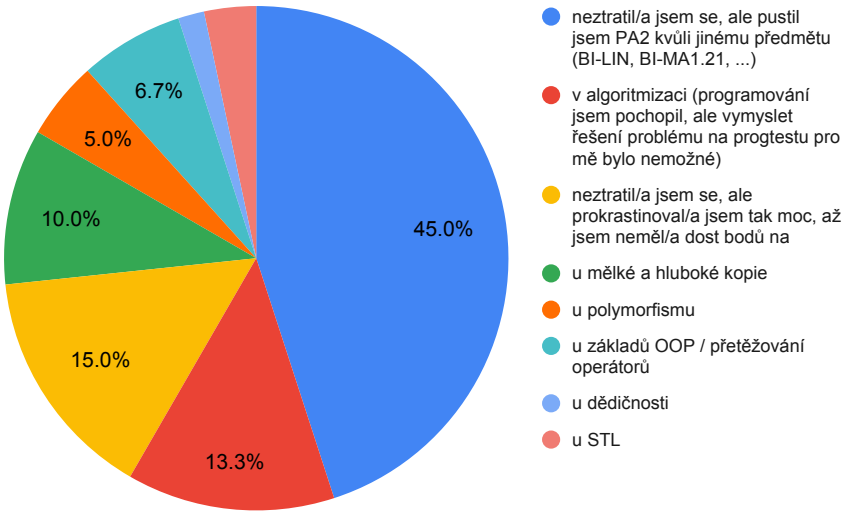
- *pustil jsem PA1/PA2 kvůli jiným předmětům* – studenti v průběhu semestru nebo zkouškového období vyhodnotili, že jim přijde PA1/PA2 jako nejtěžší předmět a že radši nezvládnou jeden předmět než více jiných: 16,7 % u PA1, 45 % u PA2
- *algoritmizace* – k tomu, aby člověk úspěšně absolvoval semestrální a zkouškové programovací úlohy nestačí pouze znát technickou stránku programovacího jazyka C / C++. Studenti musí umět také vymyslet efektivní řešení na zadaný problém: 22,2 % PA1, 13,3 % PA2
- *prokrastinace* – obecně mají studenti problém s motivací a disciplínou, a tak odkládají termíny tak dlouho, až kvůli tomu nemají dostatek bodů nebo nestihnou odevzdat semestrální práci. To může souviset i s šokem z přechodu ze střední školy, kde byli studenti zvyklí dostávat známky „za nic“ a nemuseli vynaložit téměř žádné úsilí: 11,1 % PA1, 15 % PA2
- *programování* – jednotlivá témata jsou v rámci důvodů nezvládnutí předmětu zastoupena přibližně rovnoměrně: 50,1 % PA1, 26,7 % PA2



Obrázek 2.5: Důvod neúspěchu v PA1

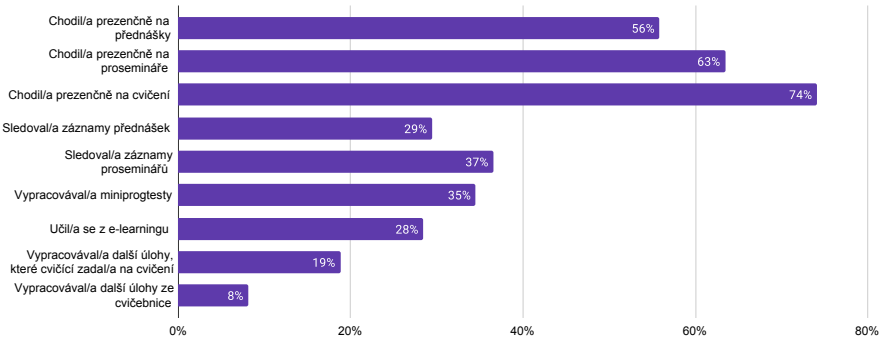
Studenti měli následně možnost v otevřené otázce vyjádřit, co (by) jim pomohlo u příštího zápisu daného předmětu. Mezi časté odpovědi patřilo zjednodušení programovací části zkoušky (snížení časového tlaku 3 hodiny na jednu zkouškovou úlohu), nebát se zeptat, výběr cvičícího, více materiálů, naučit se lépe pracovat s časem a lepší rozvržení semestrální práce v předmětu BI-PA2.

Z těchto návrhů by prakticky systém mohl pomoct se strachem se zeptat (přidáním možnosti požádat o pomoc s úlohou na dálku), lepšími materiály a/nebo poskytnutím více prostoru pro konzultaci (úlohy si studenti vyzkouší v systému, na cvičení se můžou ptát).



Obrázek 2.6: Důvod neúspěchu v PA2

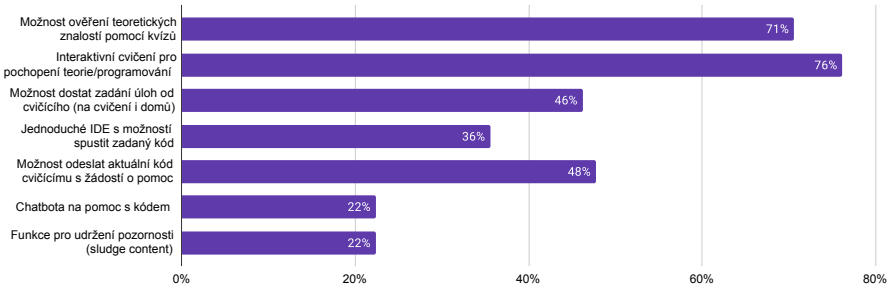
V další otázce jsem se studentů ptal na způsob přípravy v semestru a zda jim bylo poskytnuto dost materiálů. Většině studentů počet a množství materiálů v podstatě stačilo *(průměr 6,61 z 10, kde 10 je zcela dostačující)*.



Obrázek 2.7: Způsob přípravy v semestru

Z grafu způsobu přípravy v semestru lze jasně vidět důležitost cvičení, jelikož na ně chodí nejvíce studentů, ale také fakt, že bonusové úlohy a e-learning jsou používány přibližně třetinou studentů.

Poslední sekce otázek se týkala nového systému. 85 % dotázaných by nový systém uvítalo, 15 % ne. Mezi nejčastěji požadované vlastnosti patřila interaktivní cvičení (76 %), kvízy (71 %), ale také možnost požádat o pomoc (48 %) Mezi další zmíněné požadavky patří možnost požádat o code review (ohodnocení kvality studentova kódu) už v průběhu semestru, shrnutí / „cheat sheet“ ke každému tématu nebo vizualizace toho, co program dělá.



Obrázek 2.8: Nejčastěji požadované vlastnosti nového systému

Po manuálním vyhodnocení dat jsem provedl ještě zpracování pomocí Pythonu. Nejprve jsem namapoval odpovědi do matice student – způsob přípravy, ve které jsem následně s pomocí kontingenčních tabulek a Python skriptu hledal, jestli existuje nějaká závislost mezi získanými daty.

Zjistil jsem, že v datech neexistuje závislost mezi úspěšností studentů a způsobem, kterým (a zda vůbec) byla zopakována teorie na cvičeních. Taktéž neexistuje závislost mezi úspěšností a požadovanými vlastnostmi v novém systému. Zajímavostí bylo naopak to, že z dat vyplývá, že pokud někdo chodí na přednášky nebo cvičení, má menší šanci uspět u zkoušky. Toto bude ale s velkou pravděpodobností způsobeno již výše zmíněnou neúplností dat *(většina studentů, kteří nechodili na přednášky, proseminář ani cvičení nejspíše nedokončila předmět, ani nevyplňovala tento dotazník).*

Z dat však vyplývají dvě důležité závislosti: studenti, kteří se připravovali i z bonusových zdrojů *(e-learning, bonusové úlohy od cvičících / z cvičebnice, miniprogtesty)* měli větší šanci uspět v předmětu již na první zápis / termín zkoušky. Totéž platí i pro studenty, kteří chodili na cvičení, kde byli nuceni samostatně programovat *(oproti cvičením, kde pouze pasivně pozorovali).*

Potvrzuje se tak zkušenost, že bonusové materiály mají smysl a ideální formát cvičení je ten, kde studenti samostatně programují, což je také formát, který se při tvorbě systému budu snažit podporovat.

2.2.3 Učitelé

Druhou skupinou uživatelů systému jsou učitelé, kteří narozdíl od studentů již nejsou zastoupeni ve stovkách, ale pouze v desítkách. Pro učitele jsem také vytvořil dotazník na platformě Google Forms [7], na který jsem v průběhu června dostal 7 odpovědí.

Otázky v učitelském dotazníku byly vzhledem k počtu učitelů koncipovány jako otevřené. Ptal jsem se na příčiny, kde učitelé vidí, že se studenti nejčastěji ztrácejí, jak probíhá jejich cvičení a co studentům na jejich způsobu výuky vadí / co hodnotí pozitivně. V druhé sekci jsem se pak ptal, zda by učitelé využili nový systém pro podporu cvičení.

Mezi důvody, proč se studenti ztrácejí, učitelé nejčastěji uváděli šok z přechodu na vysokou školu, chybějící samostudium, neschopnost organizace času nebo podcenění náročnosti předmětu. V způsobu výuky se víceméně s drobnými obměnami opakoval průběh cvičení, který jsem popsal výše, tedy kvíz → společné programování → samostatné programování. O systém projevili zájem 3 cvičící, které jsem pak dále kontaktoval.

2.3 Stanovení požadavků

Na základě provedené analýzy jsem stanovil následující funkční a nefunkční požadavky na systém. Vůči těmto požadavkům následně porovnám stávající řešení. Požadavky jsem rozdělil do tří kategorií podle důležitosti:

- must have (F1–F5, N1–N5) – systém je **musí** splňovat a jsou pro jeho funkčnost pro všechny uživatelské role nezbytné (systém by bez nich nemohl fungovat). Tyto požadavky jsou implementovány prioritně.
- should have (F6–F8, N6) – systém by je **měl** splňovat, protože zjednodušují práci v některých scénářích, ve kterých by jinak bylo používání systému značně nepříjemné. Tyto požadavky jsou implementovány ihned po must have.
- nice to have (F9–F12, N7) jsou požadavky, které zpříjemní používání systému určitým rolím v občasných situacích. K jejich implementaci dojde, pokud zbude čas po předchozích dvou kategoriích.

2.3.1 F1 Správa studentů

Popis: Systém umožní učiteli správu studentů v rámci kurzů, které učí. Studenty lze přidávat jednotlivě, dávkově (v rámci importu) a také je jde z kurzu odebírat.

Systém také umožní učiteli nastavit kód pro připojení do kurzu. Studenti se následně do kurzu s pomocí kódu můžou dodatečně připojit.

Priorita: must-have – správa studentů je pro fungování systému nezbytná, protože bez možnosti připojení do kurzu studenti nemůžou se systémem pracovat.

2.3.2 F2 Správa lekcí, týdnů a úloh

Popis: Systém umožní učiteli vytváření lekcí v rámci kurzů, které učí. Lekce jde uspořádávat do logických celků (týdnů), upravovat, skrývat a kopírovat mezi kurzy.

Lekce obsahuje teoretickou část, která obsahuje výklad a následné ověření, zda student správně porozuměl a pochopil klíčové pojmy z přednášky. Toho lze dosáhnout například pomocí vhodně položených kvízových otázek na detaily probírané látky.

Dále lekce obsahuje praktickou část, která pomůže studentovi procvičit praktické znalosti – programování samotné. Lekce obsahuje více úloh s rostoucí úrovní obtížnosti. Úlohy můžou vyžadovat od studenta doplnění řádku kódu, funkce, vypracování celého programu nebo třeba nahrání snímku obrazovky se spuštěným programem na vlastním počítači.

Priorita: must-have – bez možnosti vytvářet lekce a úlohy by student nemohl se systémem pracovat (neměl by co vyplňovat).

2.3.3 F3 Vyplnění lekcí studentem

Popis: Systém umožní studentovi vyplnit lekce předmětů, které aktuálně studuje. Lekce se po vyplnění ukládají tak, aby byly přístupné učiteli.

U teoretické části studentovi systém postupně zobrazí otázky nebo úlohy, které má vyplnit, přičemž student na tyto otázky odpovídá a splňuje úlohy.

V praktické části student do systému zadá / nahraje kód, který je požadován dle zadání dané úlohy a systém tuto odpověď uloží.

Při vyplňování lekcí student uvidí svůj postup, kolik procent lekce už má hotovo, a které úlohy již splnil.

Priorita: must-have – vyplňování lekcí je nezbytnou součástí systému. Umožňuje uložení studentova řešení pro následné vyhodnocování a konzultaci.

2.3.4 F4 Vyhodnocení vyplněných lekcí

Popis: Systém umožní studentům vyhodnotit automaticky lekce, které vyplnili. Učitelům pak umožní vyhodnotit lekce manuálně.

Pro teoretický test je možné zadat otázky s uzavřeným výběrem odpovědí a označit správnou odpověď *(pak lze provést automatické ohodnocení)*, nebo otevřený výběr a vyhodnotit manuálně.

Praktické úlohy je možné vyhodnotit automaticky pomocí zadaných testů nebo manuálně zobrazením studentova kódu, jeho spuštěním a testem, jak reaguje na zadané vstupy a výstupy.

Priorita: must-have – možnost vyhodnotit lekce umožňuje studentovi získat zpětnou vazbu na jeho řešení, učiteli naopak poskytuje možnost hodnocení.

2.3.5 F5 OAuth přihlášení

Popis: Jak studentům, tak učitelům systém umožní přihlášení pomocí FIT nebo ČVUT OAuth. Uživatelé systému tak nebudou nuceni vytvářet zvláštní hesla a nové účty, navíc může ve spojení s požadavkem F9 Import z KOSapi rovnou po přihlášení dojít k připojení studenta do předmětů, které aktuálně studuje.

Priorita: must-have – přihlášení je nezbytné pro práci se systémem a ve spojení s OAuth zajišťuje jednoznačnou autentifikaci uživatele a bezpečnost.

2.3.6 F6 Konzultace / žádost o pomoc

Popis: V průběhu vyplňování praktických cvičení se může stát, že se student zasekne nebo si nebude vědět rady. Pro takové případy systém umožní studentovi požádat učitele o pomoc s danou úlohou. Ten v systému uvidí, s čím

má student problém, které testy mu neprošly a může studentovi odpovědět a poradit s tím, s čím má problém.

Pro detekci častých chyb lze použít také umělou inteligenci, která by automaticky při detekované časté chybě studentovi odpověděla, a k učiteli by se tak dostaly pouze specifické dotazy. Použití AI jde využít také při code review, kdy umělá inteligence zkontroluje vzhled a kvalitu kódu a předvyplní ohodnocení.

Priorita: should-have – bez možnosti konzultace přímo v systému by student musel využít alternativní způsoby, které znamenají bezpečnostní riziko (posílání kódu přes e-mail) a vyšší časovou náročnost (osobní konzultace).

2.3.7 F7 Přehled řešení studentů

Popis: Systém umožní učiteli zobrazit si přehled studentských řešení jednotlivých úloh v lekci nebo detail řešení jednoho konkrétního studenta v kurzu.

Navíc systém umožní studentovi povolit učiteli „code review“ jeho kódu *(anonymně)* na dalším cvičení. Učitel tak může snadněji prakticky poukázat na časté chyby, které studenti při vypracování dané úlohy dělají.

Priorita: should-have – přehled studentských řešení usnadňuje zápis bodů za bonusové úlohy a orientaci v systému. Bez možnosti anonymního code review by se muselo code review provádět neanonymně, což by mohlo odradit některé studenty.

2.3.8 F8 Přizpůsobení vzhledu systému

Popis: Systém umožní uživateli přizpůsobit si, jak systém vypadá. Toto přizpůsobení zahrnuje především jazyk, ve kterém bude uživatel systém používat, světlý / tmavý mód, barvu navigační lišty nebo další grafické a vizuální prvky.

Priorita: should-have – možnost přizpůsobení vzhledu systému zpříjemňuje práci se systémem, není však nezbytně nutné.

2.3.9 F9 Import z KOSapi

Popis: Systém umožní učiteli import aktuálních paralelek předmětů, které učí, včetně studentů tak, aby je nemusel ručně přepisovat.

Priorita: nice to have – jedná se o funkcionalitu, která pouze zpříjemní práci se systémem, a to pouze učitelům. Import studentů je navíc prováděn pouze na začátku semestru, tedy jednou za půl roku.

2.3.10 F10 Funkce pro udržení pozornosti (sludge content)

Popis: Do vysokých škol již nastupuje generace Z, která z důvodu nástupu nových sociálních sítí jako je Tiktok a informačního přehlcení nedokáže udržet příliš dlouho pozornost u pouze jednoho podnětu. Pro lepší udržení pozornosti by tak systém mohl obsahovat funkci, která na části obrazovky vedle aktuální lekce spustí s programováním nesouvisející videa, jako třeba Subway Surfers nebo Family Guy. [9]

Priorita: nice to have – jedná se o funkci, která není nijak potřebná pro provoz systému a používání by zpříjemnila pouze velmi úzké skupině studentů s poruchou pozornosti. Navíc si sludge content můžou studenti pustit i mimo systém v novém okně prohlížeče.

2.3.11 F11 Vizualizace kódu

Popis: Systém umožní vizualizaci kódu, který aktuálně student napsal – jaké operace provádí s jednotlivými částmi paměti *(zásobník, halda)*, které proměnné jsou aktuálně aktivní a jaké mají hodnoty. Tato vizualizace usnadní studentům pochopení toho, co jejich program dělá.

Priorita: nice to have – vizualizace umožňují snazší pochopení látky, nejsou ale potřebné pro chod systému a dají se ukázat i mimo systém.

2.3.12 F12 Porovnání s ostatními studenty

Popis: Systém umožní u jednotlivých kvízů a úloh s automatizovanými testy studentovi porovnat se s ostatními studenty *(počet zodpovězených otázek, čas zodpovězení testu, doba běhu)* a vidět jeho percentil *(kolik % studentů má méně bodů)* v rámci paralelky.

Priorita: nice to have – porovnání není potřebné pro chod systému, je spíše pro zajímavost

2.3.13 N1 API rozhraní pro budoucí rozšířitelnost

Popis: Je velká pravděpodobnost, že bude systém v budoucnu aktivně používán i v dalších semestrech, přičemž se na jeho rozvoji na potenciálních nových platformách můžou věnovat další studenti v rámci svých závěrečných prací nebo v rámci týmových projektů. Systém tedy bude implementován tak, aby byly jednotlivé části propojeny pomocí API a umožnil tak případné napojení verzí z jiných platforem.

Priorita: must have – bez API rozhraní by bylo přidání nové platformy v rámci dalších projektů velmi obtížné a náročné.

2.3.14 N2 Modulární architektura

Popis: Systém musí být navržen tak, aby využíval modulární architekturu, především pro jednotlivé typy úloh v lekci (moduly). Tato architektura umožní souběžný vývoj dalších modulů a jednoduché budoucí rozšíření o další funkcionality. Konkrétně bude souběžně s tvorbou systému probíhat vývoj kvízového modulu Matejem Paškem v rámci jeho bakalářské práce.

Priorita: must have – bez modulární architektury by bylo přidávání dalších typů úloh do systému náročné a obtížné.

2.3.15 N3 Webové rozhraní

Popis: Systém poběží na webu tak, aby nebyl žádný požadavek na instalaci nadbytečných programů uživateli.

Priorita: must have – bez webového rozhraní by byl uživatel zatížen dalšími instalacemi a navíc by bylo nutné vyvíjet systém pro každou cílovou platformu (Windows, Linux, macOS, Android, iOS).

2.3.16 N4 Responzivní rozhraní

Popis: Vzhledem k různosti prostředí, ve kterém se bude systém zobrazovat, musí být rozhraní systému responzivní, a to alespoň pro desktop, tablet a mobilní telefony.

Priorita: must have – responzivní rozhraní je nezbytně nutné pro použitelnost systému především na mobilních zařízeních, ale také na desktopech s menší velikostí obrazovky.

2.3.17 N5 Cena a provoz

Popis: Systém bude používán především v akademickém prostředí a bude používán více předměty, proto je žádoucí, aby byl vytvořen bezplatně a vhodné, aby byl provozován na akademické infrastruktuře.

Priorita: must have – fakulta nemá finanční zdroje, ze kterých by financovala placený vývoj systému. Provoz na akademické infrastruktuře navíc zajistí možnost pokračujícího vývoje dalšími studenty.

2.3.18 N6 Nasazení do kontejneru

Popis: Použití kontejnerů pro vývoj a nasazení je vhodné, jelikož aplikace není závislá na hardwaru a operačním systému, na kterém běží a navíc se dá rychle a efektivně přesunout.

Priorita: should have – bez použití kontejnerů by byl proces testování a nasazování složitější.

2.3.19 N7 Spuštění kódu u klienta

Popis: Systém umožní spuštění kódu a testů tak, aby kompilování kódů nezatěžovalo jeden centrální server a nemuselo se tak řešit umísťování požadavků na kompilaci do fronty.

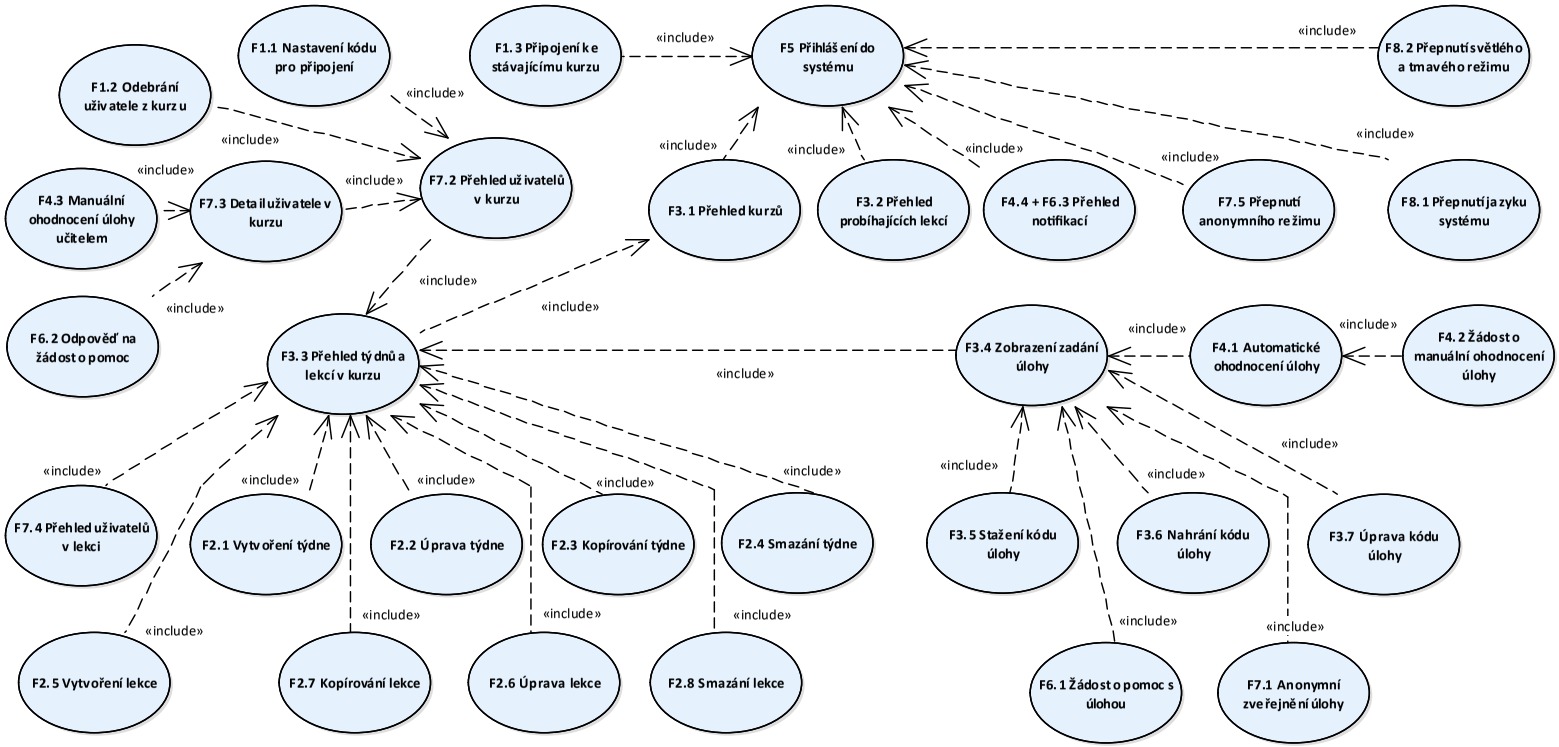
Priorita: nice to have – pokud by spouštění probíhalo na serveru, musel by program běžet pouze omezenou dobu, což lehce znepříjemňuje testování složitějších programů.

2.4 Případy užití

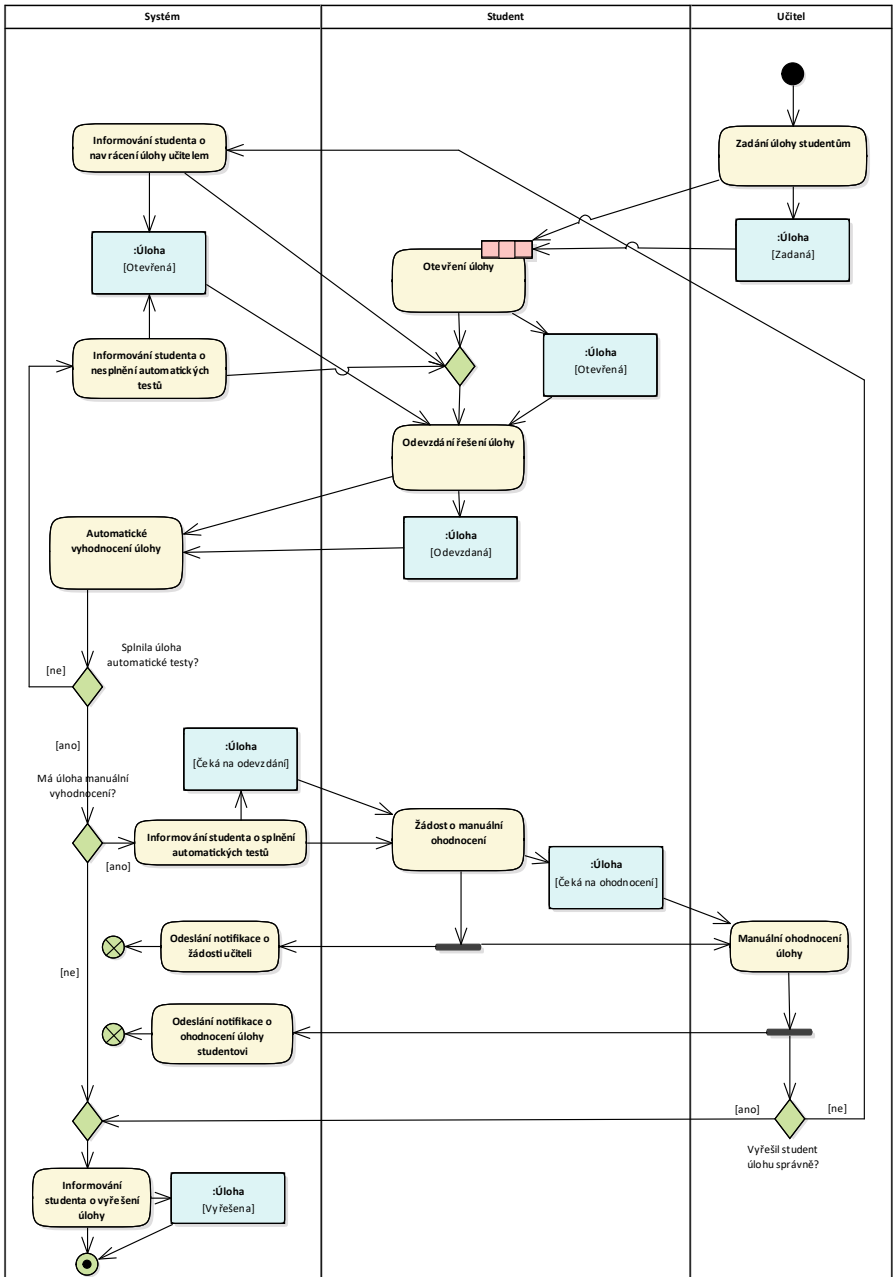
Na základě provedené analýzy a stanovených funkčních požadavků jsem identifikoval aktéry a případy užití. Kódy případů užití odpovídají funkčním požadavkům, které pokrývají. Případ užití F5.1 tedy pokrývá požadavek F5, F4.4 + F6.3 požadavky F4 a F6.

2.4.1 Aktéři

Systém budou používat dva typy uživatelů: studenti a učitelé. Pokud je v případu užití uveden jako aktér **Uživatel**, jedná se o akci, kterou může provést jak student, tak učitel (například přihlášení do systému, přehled kurzů, lekcí, odevzdání úlohy, přehled notifikací, přizpůsobení systému). **Student** se navíc může připojit ke kurzu, požádat učitele o manuální ohodnocení úlohy nebo o pomoc. **Učitel** může spravovat lekce, úlohy, odpovídat na žádosti a zobrazit přehled studentů.



Obrázek 2.9: Diagram závislostí mezi jednotlivými případy užití



Obrázek 2.10: Proces vyplnění a vyhodnocení úlohy

2.4.2 F1.1 Nastavení kódu pro připojení

Uživatelský cíl: Učitel očekává, že nastaví kód, s pomocí kterého se následně studenti budou moct připojit do kurzu daného předmětu.

1. Učitel přejde do přehledu uživatelů v předmětu, kterému chce nastavit kód podle scénáře F7.2 Přehled uživatelů v kurzu
2. Učitel stiskne tlačítko ZMĚNIT KÓD
3. Systém zobrazí dialog pro nastavení kódu
4. Učitel zadá nový kód a stiskne ZMĚNIT

2.4.3 F1.2 Odebrání uživatele z kurzu

Uživatelský cíl: Učitel očekává způsob, kterým odebere uživatele z předmětu tak, že po odebrání už nebude členem předmětu.

1. Učitel přejde do přehledu uživatelů v kurzu předmětu, ve kterém chce odebrat uživatele podle scénáře F7.2 Přehled uživatelů v kurzu
2. Systém zobrazí tabulku studentů s tlačítkem „Odebrat“ u řádku pro každého uživatele
3. Učitel klikne na toto tlačítko
4. Systém zobrazí dialog s potvrzením, zda chce učitel opravdu daného uživatele odebrat
5. Učitel stiskne ODEBRAT

2.4.4 F1.3 Připojení ke stávajícímu kurzu

Uživatelský cíl: Student se chce připojit ke kurzu předmětu (stát se jeho členem).

1. Student se přihlásí do systému podle scénáře F5.1 Přihlášení do systému
2. Systém zobrazí přehled aktuálních lekcí, přehled kurzů s výběrem semestru a tlačítko PŘIPOJIT SE KE KURZU
3. Student klikne na tlačítko PŘIPOJIT SE KE KURZU
4. Systém zobrazí dialog pro zadání připojovacího kódu
5. Student zadá kód pro připojení
6. Systém přesměruje studenta do kurzu, do kterého se právě připojil

2.4.5 F2.1 Vytvoření týdne

Uživatelský cíl: Učitel chce vytvořit nový týden v kurzu, který učí.

1. Učitel přejde do detailu kurzu, ve kterém chce přidat nový týden podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí vpravo dole tlačítko pro vytvoření nového týdne
3. Učitel stiskne toto tlačítko
4. Systém zobrazí dialog pro zadání jména, začátku a konce týdne. Pokud už kurz obsahoval nějaký týden, je předvyplněn začátek a konec podle posledního týdne v kurzu.
5. Učitel zadá údaje o kurzu a stiskne VYTVOŘIT

2.4.6 F2.2 Úprava týdne

Uživatelský cíl: Učitel chce upravit týden v kurzu, který učí.

1. Učitel přejde do detailu kurzu, ve kterém chce upravit týden podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí u každého týdne tlačítko „Upravit“
3. Učitel stiskne toto tlačítko
4. Systém zobrazí dialog pro úpravu názvu, začátku a konce týdne
5. Učitel upraví údaje o kurzu a stiskne UPRAVIT

2.4.7 F2.3 Kopírování týdne

Uživatelský cíl: Učitel chce zkopírovat týden mezi kurzy, které učí.

1. Učitel přejde do detailu kurzu, ze kterého chce zkopírovat týden podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí u každého týdne tlačítko „Zkopírovat“
3. Učitel stiskne toto tlačítko
4. Systém zobrazí dialog se seznamem kurzů, do kterých lze týden zkopírovat
5. Učitel stiskne tlačítko „Zkopírovat do kurzu“ u kurzu, do kterého chce týden zkopírovat
6. Systém učitele přesměruje do kurzu, do kterého byl týden zkopírován

2.4.8 F2.4 Smazání týdne

Uživatelský cíl: Učitel chce smazat týden z kurzu, který učí.

1. Učitel přejde do detailu kurzu, ve kterém chce smazat týden podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí u každého týdne tlačítko „Smazat“
3. Učitel stiskne toto tlačítko
4. Systém zobrazí dialog s potvrzením, zda chce učitel opravdu týden smazat
5. Učitel smazání potvrdí stiskem tlačítka SMAZAT

2.4.9 F2.5 Vytvoření lekce

Uživatelský cíl: Učitel chce vytvořit novou lekci v kurzu, který učí.

1. Učitel přejde do detailu kurzu, ve kterém chce přidat novou lekci podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí u každého týdne tlačítko „Přidat novou lekci“
3. Učitel stiskne toto tlačítko
4. Systém přesměruje učitele na obrazovku pro vytvoření nové lekce
5. Učitel zadá údaje o lekci a stiskne VYTVOŘIT

2.4.10 F2.6 Úprava lekce

Uživatelský cíl: Učitel chce upravit stávající lekci v kurzu, který učí.

1. Učitel přejde do detailu kurzu, ve kterém chce upravit lekci podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí u každé lekce tlačítko „Upravit“
3. Učitel stiskne toto tlačítko
4. Systém přesměruje učitele na obrazovku pro úpravu stávající lekce
5. Učitel zadá údaje o lekci a stiskne UPRAVIT

2.4.11 F2.7 Kopírování lekce

Uživatelský cíl: Učitel chce zkopírovat lekci mezi kurzy, které učí.

1. Učitel přejde do detailu kurzu, ze kterého chce zkopírovat lekci podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí u každé lekce tlačítko „Zkopírovat“
3. Učitel stiskne toto tlačítko
4. Systém zobrazí dialog se seznamem kurzů, do kterých lze lekci zkopírovat

5. Učitel stiskne tlačítko „Zkopírovat do kurzu“ u kurzu, do kterého chce lekci zkopírovat

6. Systém přesměruje učitele do kurzu, do kterého byla lekce zkopírována.

2.4.12 F2.8 Smazání lekce

Uživatelský cíl: Učitel chce smazat lekci z kurzu, který učí.

1. Učitel přejde do detailu kurzu, ze kterého chce smazat lekci podle scénáře F3.3 Přehled týdnů a lekcí v kurzu

2. Systém zobrazí u každé lekce tlačítko „Smazat“

3. Učitel stiskne toto tlačítko

4. Systém zobrazí dialog s potvrzením, zda chce učitel opravdu lekci smazat

5. Učitel smazání potvrdí stiskem tlačítka SMAZAT

2.4.13 F3.1 Přehled kurzů

Uživatelský cíl: Uživatel chce vidět přehled kurzů, které studuje nebo učí v daném semestru.

1. Uživatel se přihlásí do systému podle scénáře F5.1 Přihlášení do systému

2. Systém zobrazí seznam kurzů předmětů, které uživatel studuje / učí v aktuálním semestru.

3. Pokud chce uživatel vidět kurzy v minulých semestrech, vybere z rozevíracího seznamu semestr, pro který chce vidět kurzy, nebo ze seznamu vybere „Všechny“ semestry.

4. Systém zobrazí přehled kurzů předmětů, které uživatel studoval / učil ve vybraném / všech semestrech.

2.4.14 F3.2 Přehled probíhajících lekcí

Uživatelský cíl: Uživatel chce vidět aktuálně probíhající lekce v kurzech, které studuje nebo učí.

1. Uživatel se přihlásí do systému podle scénáře F5.1 Přihlášení do systému

2. Systém zobrazí seznam probíhajících lekcí v kurzech, které uživatel studuje nebo učí

2.4.15 F3.3 Přehled týdnů a lekcí v kurzu

Uživatelský cíl: Uživatel chce vidět lekce v konkrétním kurzu, který studuje nebo učí.

1. Uživatel si zobrazí přehled kurzů podle scénáře F3.1 Přehled kurzů
2. Systém u každého kurzu zobrazí tlačítko „Přejít do detailu kurzu“
3. Uživatel stiskne toto tlačítko
4. Systém uživatele přesměruje do detailu daného kurzu a zobrazí seznam týdnů a lekcí

2.4.16 F3.4 Zobrazení zadání úlohy

Uživatelský cíl: Uživatel si chce zobrazit zadání konkrétní úlohy v dané lekci v daném kurzu.

1. Uživatel si zobrazí přehled týdnů a lekcí v daném kurzu podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém u každé lekce zobrazí tlačítko „Přejít do detailu lekce“
3. Uživatel stiskne toto tlačítko
4. Systém uživatele přesměruje do detailu dané lekce a zobrazí v postranním menu seznam úloh v jednotlivých lekcích v daném týdnu
5. Uživatel vybere v menu úlohu, jejíž zadání si chce zobrazit
6. Systém uživateli zobrazí zadání dané úlohy

2.4.17 F3.5 Stažení kódu úlohy

Uživatelský cíl: Uživatel chce začít pracovat na zadání úlohy tak, že si stáhne její výchozí / svůj poslední rozpracovaný kód.

1. Uživatel si zobrazí zadání úlohy podle scénáře F3.4 Zobrazení zadání úlohy
2. Systém zobrazí pod zadáním úlohy lištu s tlačítkem „Stáhnout kód“
3. Uživatel stiskne toto tlačítko
4. Systém zobrazí kontextové menu s výběrem kódu ke stažení (výchozí kód / odevzdaný kód / referenční řešení, pokud je dostupné)
5. Uživatel vybere z menu kód, který chce stáhnout
6. Systém zobrazí prohlížečové okno pro stažení souboru s kódem

2.4.18 F3.6 Nahrání kódu úlohy

Uživatelský cíl: Uživatel chce odevzdat svůj lokální kód do systému.

1. Uživatel si zobrazí zadání úlohy podle scénáře F3.4 Zobrazení zadání úlohy
2. Systém zobrazí pod zadáním úlohy lištu s polem pro nahrání kódu
3. Uživatel nahraje kód
4. Systém kód uloží na server a zobrazí uživateli hlášku o úspěšném nahrání kódu

2.4.19 F3.7 Úprava kódu úlohy

Uživatelský cíl: Uživatel chce pracovat na úloze přímo v prohlížeči, aniž by si musel kód stahovat a nahrávat.

1. Uživatel si zobrazí zadání úlohy podle scénáře F3.4 Zobrazení zadání úlohy
2. Systém zobrazí nad zadáním vedle záložky „Zadání“ další záložky s názvy jednotlivých souborů kódu
3. Uživatel vybere záložku s názvem souboru kódu, který chce upravovat
4. Systém zobrazí editor s kódem vybraného souboru
5. Uživatel upraví kód

2.4.20 F4.1 Automatické ohodnocení úlohy

Uživatelský cíl: Uživatel chce otestovat, zda jeho řešení úlohy splňuje automatické testy.

1. Uživatel si zobrazí zadání úlohy podle scénáře F3.4 Zobrazení zadání úlohy
2. Systém zobrazí vedle zadání boční lištu s názvy a popisem jednotlivých testů. Pod testy systém zobrazí tlačítko OTESTOVAT
3. Uživatel stiskne tlačítko OTESTOVAT
4. Systém provede automatické testy a u každého názvu testu zobrazí, zda test prošel, nebo ne

2.4.21 F4.2 Žádost o manuální ohodnocení úlohy

Uživatelský cíl: Student chce, aby jeho vypracovanou úlohu, která prošla automatickými testy, manuálně ohodnotil učitel.

1. Student splní všechny automatické testy podle scénáře F4.1 Automatické ohodnocení úlohy
2. Systém zobrazí vedle tlačítka Otestovat tlačítko ODEVZDAT
3. Student stiskne tlačítko ODEVZDAT
4. Systém zobrazí dialog obsahující textové pole pro text žádosti
5. Student vyplní text žádosti a stiskne ODEVZDAT

2.4.22 F4.3 Manuální ohodnocení úlohy učitelem

Uživatelský cíl: Učitel chce manuálně ohodnotit studentovi jeho řešení úlohy.

1. a) Učitel si zobrazí detail uživatele v kurzu podle scénáře F7.3 Detail uživatele v kurzu a zobrazí si zadání úlohy podle scénáře F3.4 Zobrazení zadání úlohy
   b) Učitel zobrazí přehled uživatelů v kurzu podle scénáře F7.2 Přehled uživatelů v kurzu se zapnutým filtrem na žádosti a klikne na sloupec s názvem modulu
   c) Učitel si zobrazí přehled notifikací podle scénáře F4.4 + F6.3 Přehled notifikací a klikne na notifikaci s žádostí o ohodnocení
2. Systém místo zadání zobrazí studentův kód, v postranní liště zobrazí výsledky testů a text žádosti o ohodnocení. Pod textem zobrazí tlačítko OHODNOTIT, pokud student požádal o ohodnocení, nebo OKOMENTOVAT.
3. Učitel stiskne tlačítko OHODNOTIT / OKOMENTOVAT
4. Systém zobrazí dialog obsahující textové pole pro hodnocení a přepínač, kterým může učitel nastavit úlohu jako splněnou
5. Učitel napíše hodnocení, volitelně označí úlohu jako splněnou a stiskne OHODNOTIT / OKOMENTOVAT

2.4.23 F4.4 + F6.3 Přehled notifikací

Uživatelský cíl: Student si chce zobrazit seznam notifikací, které dostal od učitele (odpověď na žádost o pomoc / ohodnocení). Učitel si chce zobrazit seznam notifikací, které dostal od studenta (žádost o pomoc / ohodnocení).

1. Uživatel se přihlásí do systému podle scénáře F5.1 Přihlášení do systému
2. Systém zobrazí nahoře navigační lištu s tlačítkem „Přehled notifikací“. V případě nepřečtené notifikace se u tlačítka zobrazí číslo s počtem nepřečtených notifikací.
3. Uživatel stiskne tlačítko pro přehled notifikací.
4. Systém zobrazí přehled nepřečtených notifikací a tlačítko pro zobrazení všech notifikací
5. Uživatel stiskne tlačítko VŠECHNY NOTIFIKACE
6. Systém uživatele přesměruje na seznam všech notifikací

2.4.24 F5.1 Přihlášení do systému

Uživatelský cíl: Uživatel očekává, že bude přihlášen do systému, nebo bude přesměrován na FIT přihlašovací bránu, přes kterou se přihlásí do systému.

1. Systém zobrazí tlačítko PŘIHLÁŠENÍ ČVUT ÚČTEM
2. Uživatel stiskne toto tlačítko
3. Systém uživatele přesměruje na přihlašovací bránu
4. Přihlašovací brána uživatele rovnou přihlásí (existující relace v systému) nebo mu zobrazí přihlašovací formulář
5. Uživatel vyplní své přihlašovací údaje
6. Přihlašovací brána uživatele přesměruje do systému
7. Systém uživatele přihlásí a zobrazí mu úvodní stránku

2.4.25 F6.1 Žádost o pomoc s úlohou

Uživatelský cíl: Student chce požádat o pomoc s úlohou, se kterou si neví rady.

1. Student si zobrazí zadání úlohy podle scénáře F3.4 Zobrazení zadání úlohy
2. Systém zobrazí v boční liště s testy vedle tlačítka Otestovat tlačítko POŽÁDAT O POMOC
3. Student stiskne tlačítko POŽÁDAT O POMOC
4. Systém zobrazí dialog obsahující textové pole pro text žádosti
5. Student vyplní text žádosti a stiskne ODESLAT

2.4.26 F6.2 Odpověď na žádost o pomoc

Uživatelský cíl: Učitel chce studentovi odpovědět na jeho žádost o pomoc.

1. a) Učitel si zobrazí detail uživatele v kurzu podle scénáře F7.3 Detail uživatele v kurzu a zobrazí si zadání úlohy podle scénáře F3.4 Zobrazení zadání úlohy
   b) Učitel si zobrazí přehled uživatelů v kurzu podle scén. F7.2 Přehled uživatelů v kurzu se zapnutým filtrem na žádosti a klikne na sloupec s názvem modulu
   c) Učitel si zobrazí přehled notifikací podle scénáře F4.4 + F6.3 Přehled notifikací a klikne na notifikaci s žádostí o ohodnocení
2. Systém místo zadání zobrazí studentův kód, v postranní liště zobrazí výsledky testů a text žádosti o pomoc. Pod textem zobrazí tlačítko ODPOVĚDĚT.
3. Učitel stiskne tlačítko ODPOVĚDĚT
4. Systém zobrazí dialog obsahující textové pole pro odpověď na žádost o pomoc
5. Učitel napíše odpověď a stiskne ODPOVĚDĚT

2.4.27 F7.1 Anonymní zveřejnění úlohy

Uživatelský cíl: Student chce povolit anonymní zveřejnění svého řešení úlohy proto, aby ho učitel veřejně okomentoval.

1. Student si zobrazí zadání úlohy podle scénáře F3.4 Zobrazení zadání úlohy
2. Systém zobrazí pod zadáním úlohy lištu s tlačítkem „Povolit anonymizované zveřejnění“
3. Student stiskne toto tlačítko
4. Systém povolí u řešení úlohy anonymizované procházení. Toto je indikováno emotikonou oka u názvu úlohy.

2.4.28 F7.2 Přehled uživatelů v kurzu

Uživatelský cíl: Učitel chce zobrazit seznam uživatelů v kurzu.

1. Učitel zobrazí seznam týdnů a lekcí v kurzu podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí nad seznamem úloh a lekcí tlačítko UŽIVATELÉ
3. Učitel stiskne toto tlačítko
4. Systém zobrazí seznam všech uživatelů (učitelů i studentů) v daném kurzu. V seznamu lze vyhledávat podle jména, usernamu, role nebo pokroku studenta a jde ho řadit kliknutím na název sloupce.

2.4.29 F7.3 Detail uživatele v kurzu

Uživatelský cíl: Učitel chce zobrazit detail konkrétního uživatele v kurzu.

1. Učitel si zobrazí přehled uživatelů v kurzu dle scénáře F7.2 Přehled uživatelů v kurzu
2. Systém zobrazí v řádku u každého uživatele tlačítko „Detail uživatele“
3. Učitel stiskne toto tlačítko u uživatele, jehož detail chce zobrazit
4. Systém zobrazí pohled na seznam týdnů a lekcí tak, jak by jej viděl daný uživatel (včetně pokroku v jednotlivých lekcích a možnosti přejít do detailu lekce). V postranním menu je seznam uživatelů, ve kterém se dá přecházet mezi studenty.

2.4.30 F7.4 Přehled uživatelů v lekci

Uživatelský cíl: Učitel chce zobrazit detail všech uživatelů v konkrétní lekci.

1. Učitel zobrazí seznam týdnů a lekcí v kurzu podle scénáře F3.3 Přehled týdnů a lekcí v kurzu
2. Systém zobrazí v řádku u každé lekce tlačítko „Přehled studentů a modulů“
3. Učitel stiskne toto tlačítko u lekce, jejíž přehled chce zobrazit
4. Systém zobrazí tabulku s jednotlivými studenty v řádcích a modulech ve sloupcích, kde v každém poli tabulky je indikace, jestli student daný modul splnil / splnil částečně / nesplnil, povolil anonymizované zveřejnění nebo požádal o pomoc / ohodnocení. Nad tabulkou je filtr, který umožňuje hledat podle splnění / anonymizovaného zveřejnění / žádosti.

2.4.31 F7.5 Přepnutí anonymního režimu

Uživatelský cíl: Učitel nechce při promítání na projektoru ukazovat jména studentů v přehledech lekcí a nechce ukazovat řešení, která nemají povolené anonymizované zveřejnění.

1. Učitel se přihlásí do systému podle scénáře F5.1 Přihlášení do systému
2. Systém zobrazí navigační lištu s rozbalovacím menu u názvu uživatele. V rozbalovacím menu je přepínač anonymního módu.
3. Učitel stiskne přepínač anonymního módu
4. Systém přejde do anonymního módu. V seznamech studentů se místo jejich jmen ukazují emoji, v pohledu na studentská řešení se neukazují řešení, u nichž nebylo povoleno anonymizované zveřejnění.

2.4.32 F8.1 Přepnutí jazyku systému

Uživatelský cíl: Uživatel chce používat systém v angličtině nebo jiném jazyce.

1. Učitel se přihlásí do systému podle scénáře F5.1 Přihlášení do systému
2. Systém zobrazí navigační lištu s rozbalovacím menu u názvu uživatele. V rozbalovacím menu je přepínač jazyka.
3. Učitel stiskne přepínač jazyka
4. Systém se přepne do požadovaného jazyka

2.4.33 F8.2 Přepnutí světlého a tmavého režimu

Uživatelský cíl: Uživatel chce používat systém ve světlém / tmavém módu.

1. Učitel se přihlásí do systému podle scénáře F5.1 Přihlášení do systému
2. Systém zobrazí navigační lištu s rozbalovacím menu u názvu uživatele. V rozbalovacím menu je přepínač světlého a tmavého módu.
3. Učitel stiskne přepínač světlého a tmavého módu
4. Systém se přepne do požadovaného módu

2.5 Existující řešení

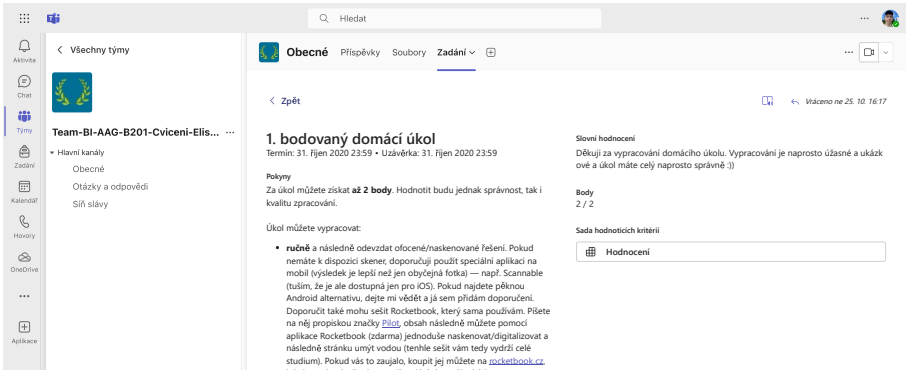
Abych zjistil, jestli má tvorba systému smysl, provedl jsem analýzu existujících systémů, které slouží k zadávání úloh, poskytování zpětné vazby (směrem od učitele k žákovi) a praktickému ověřování (programovacích) znalostí.

Při analýze jsem neuvažoval stávající řešení sloužící k teoretickému ověřování znalostí pomocí kvízů (požadavek F3 Vyplnění lekcí studentem), jelikož se analýzou tohoto požadavku a tvorbou kvízového modulu zabývá Matej Pašek ve své bakalářské práci.

2.5.1 Microsoft Teams

Prvním analyzovaným systémem je Microsoft Teams [10] (dále už jen Teams). Ty jsou dostupné na většině platforem a v omezené míře i jako webová aplikace. Teams se začaly používat v průběhu koronavirového období, ale jejich popularita i po zrušení epidemických omezení a návratu studentů do škol neklesla a stále se aktivně používají na základních, středních i vysokých školách.

Teams umožňují správu týmů (F1), které se typicky používají pro jeden předmět a jednu třídu. V rámci týmu je možné poskytovat studentům materiály a zadávat domácí úkoly, které lze následně bodově ohodnotit (F4.2 + F4.3). Přihlášení (F5) je možné s pomocí Microsoft účtu, který má zřízen každý student i učitel FITu a do něhož se přihlašuje pomocí ČVUT OAuth. Pokud student potřebuje s něčím pomoct, může využít soukromý chat s učitelem (F6).

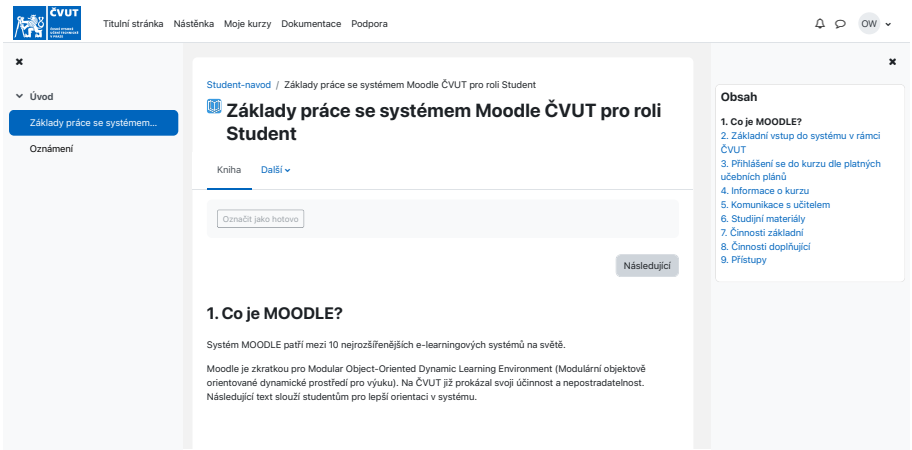


Obrázek 2.11: Ohodnocený úkol v systému Microsoft Teams

Byť jsou Teams mocným nástrojem, který se dá pro výuku programování používat, neumožňují automatické vyhodnocování odevzdaných úloh (F4.1), což značně znepříjemňuje učitelům práci. Způsob konzultace úlohy, se kterou si student neví rady, je zde také značně omezen. Teams navíc neumožňují ani spouštění kódu (N7) nebo jakoukoliv práci s kódem.

2.5.2 Moodle

Další alternativou je systém Moodle [11]. Moodle lze provozovat jako webovou aplikaci a na ČVUT se používá pro podporu výuky většiny předmětů.



Obrázek 2.12: Lekce v systému Moodle

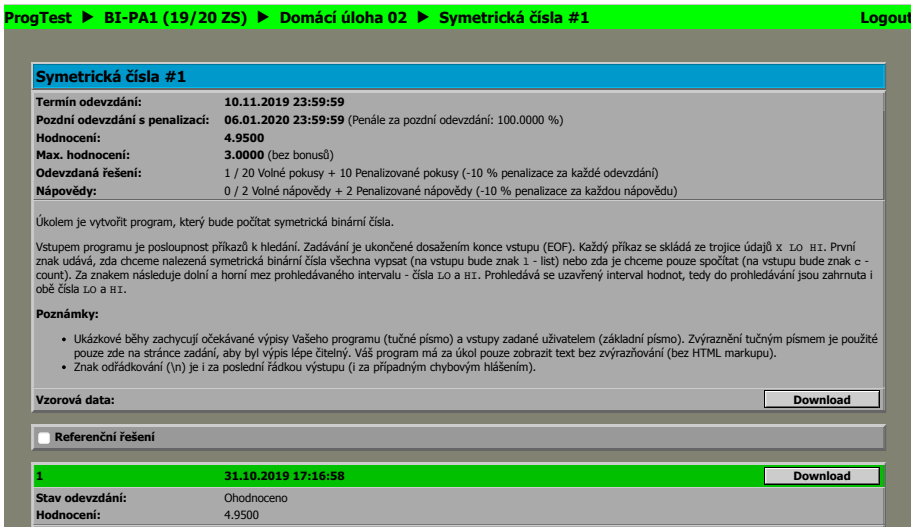
Jedná se přímo o e-learningový systém, podporuje tedy správu kurzů (F1), ve kterých může učitel připravit materiály pro studenty, lze se do něj přihlásit pomocí ČVUT OAuth (F5), umožňuje odevzdávání úkolů (F3), které lze hodnotit (F4) a navíc oproti Teamsům dokonce poskytuje přehled studentů ve třídě, jejich počet bodů a výpočet známek na základě získaných bodů (F7).

Jelikož jde o obecný systém, opět zde chybí podpora automatizovaného vyhodnocování úloh nebo spouštění kódu (F4.1). Moodle ale narozdíl od Teams podporuje modulární architekturu, je tedy možné, že by šlo tento systém rozšířit o modul, který by programovací úlohy podporoval. Vzhledem ke komplexitě, neatraktivnímu uživatelskému rozhraní a absenci konzultace odevzdávaných úloh by to ale stejně nejspíše nevedlo k dobrému výsledku.

2.5.3 ProgTest

Dalším systémem, který se na FITu používá, je ProgTest [12]. Tento systém se používá k automatizovanému vyhodnocování domácích úkolů z programování, mimojiné i v předmětech PA1 a PA2.

Tento systém slouží především k ověřování znalostí, neobsahuje tedy výklad, úlohy s neomezeným počtem odevzdání na procvičení nebo možnost konzultace neúspěšného řešení (F6). Umožňuje přihlášení přes ČVUT OAuth (F5), vyplňování a automatické vyhodnocování domácích úloh (F3 + F4), porovnání s ostatními studenty (F12), ale také například propojení s KOSapi (F9). Učiteli poskytuje také rozhraní pro správu studentů a přehled jejich pokroku v jednotlivých úlohách (F7).



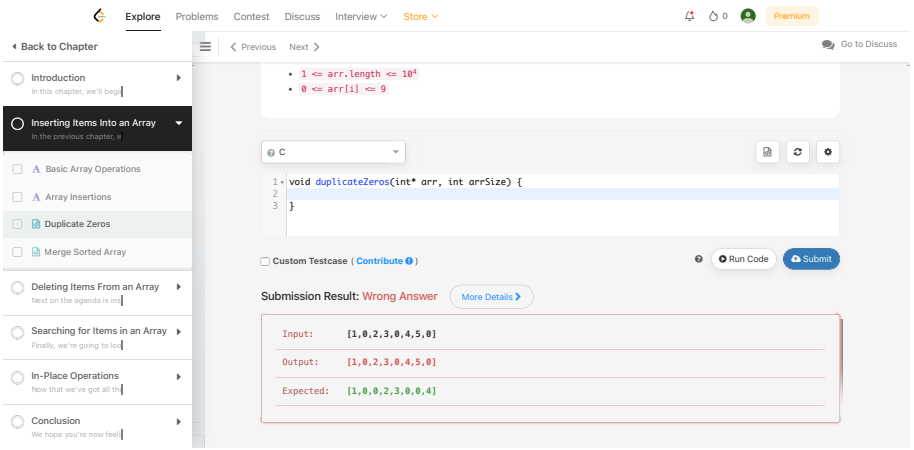
Obrázek 2.13: Zadání úlohy v systému ProgTest

Jedna z věcí, za kterou je ProgTest dlouhodobě kritizován studenty, je jeho zastaralé uživatelské rozhraní a ne příliš užitečné nápovědy. Kód je navíc spouštěn na serveru v jedné frontě, což vede k dlouhým kompilačním časům v situaci, kdy se o kompilaci pokusí více studentů najednou.

2.5.4 LeetCode

Kromě systémů, které jsou používány na FITu, existuje i řada online e-learningů, které lze použít k samostudiu programování, jako je LeetCode [13].

LeetCode se skládá z kurzů obsahujících texty s vysvětlením daného tématu následované krátkými programovacími úlohami pro ověření, zda student pochopil zadanou problematiku. Úlohy lze automaticky vyhodnotit (F3 + F4). V průběhu vyplňování lekce student vidí svůj rostoucí pokrok v lekci. Pokud se u nějaké úlohy student zasekne, má možnost se zeptat na fóru ostatních studentů a dostat radu.

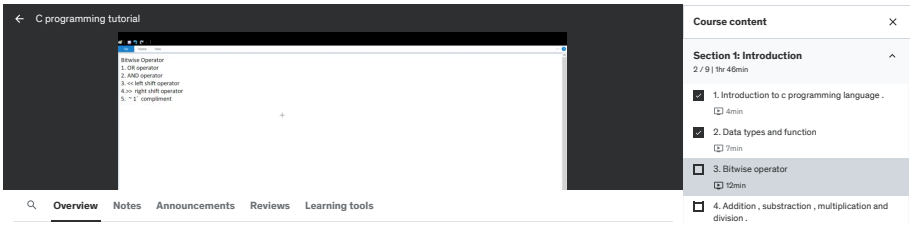


Obrázek 2.14: Úloha v systému LeetCode

LeetCode je moderní a praktický systém, bohužel ale není přizpůsoben pro práci ve třídě a pro použití ve školní výuce. LeetCode neumožňuje vytváření kurzů se správou uživatelů (F1), napojení na vlastní přihlašovací systém jako je OAuth (F5) ani provoz na vlastní infrastruktuře (N5). Neexistuje zde role učitele, možnost ohodnotit úlohy manuálně (F4.2 + F4.3) nebo zobrazit přehled studentů a jejich řešení v rámci lekce (F7).

2.5.5 udemy

Jedním z dalších e-learningů je portál udemy [14]. Ten je narozdíl od LeetCodu zaměřený spíše na výuková videa. Každý kurz obsahuje na sebe navazující výuková videa, která jsou obohacena o doplňkové materiály.

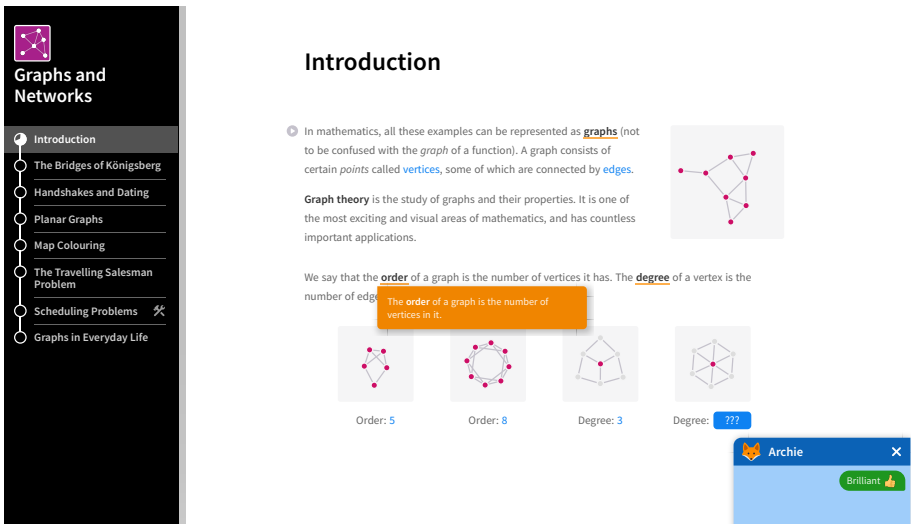


Obrázek 2.15: Kurz v systému udemy

Tento portál tedy neumožňuje programovací úlohy se spouštěním a vyhodnocováním kódu, není zde dokonce ani možnost požádat o pomoc (F4, F6). Stejně jako v případě systému LeetCode, ani udemy není určen pro práci ve třídě a školní výuku – není zde možnost správy kurzů, přihlášení přes OAuth ani pohled učitele (F1, F5, F7).

2.5.6 Mathigon

Posledním analyzovaným e-learningem je Mathigon [15]. Mathigon je systém zaměřený spíše na výuku matematiky a například grafové teorie, má ale velmi přívětivé a příjemné uživatelské rozhraní. V jednotlivých lekcích postupně vyplňujete různé doplňovačky, přesouváte interaktivní prvky, přičemž při správné odpovědi dostanete hned zpětnou vazbu a vidíte svůj postup v lekci. Pokud si s něčím nevíte rady, je vám k dispozici chatbot Archie.



Obrázek 2.16: Interaktivní vyplňování v e-learningu Mathigon

Tento systém opět neumožňuje vytváření vlastních lekcí (F2) a nepodporuje práci ve třídě (F1) ani přihlášení přes OAuth (F5), umožňuje ale velmi plynulý a příjemný průchod lekcí (F3) s vizualizacemi jednotlivých témat (F11).

2.6 Shrnutí

Po zanalyzování výše popsaných existujících řešení jsem vytvořil matici požadavek – řešení. ✓ znamená, že požadavek je zcela splněn, • znamená částečné splnění, - znamená, že požadavek není splněn vůbec. Tučně jsou zvýrazněny must have požadavky, nice to have jsou zvýrazněny kurzívou.

	MS Teams	Moodle	ProgTest	LeetCode	udemy	Mathigon
F1	✓	✓	✓	-	-	-
F2	✓	✓	✓	-	-	-
F3	✓	✓	✓	✓	✓	✓
F4	•	•	✓	✓	-	-
F5	✓	✓	✓	-	-	-
F6	✓	✓	-	✓	-	-
F7	-	-	-	-	-	-
F8	✓	•	-	-	•	✓
F9	✓	✓	✓	-	-	-
F10	-	-	-	-	-	-
F11	-	-	-	-	-	✓
F12	-	-	✓	✓	-	-
N1	✓	✓	-	✓	✓	✓
N2	-	✓	-	✓	-	✓
N3	•	✓	✓	✓	✓	✓
N4	✓	✓	-	✓	✓	✓
N5	✓	✓	✓	-	-	✓
N6	-	✓	-	-	-	✓
N7	-	-	-	-	-	-

Tabulka 2.1: Matice systém – splnění požadavku.

Z matice je zřejmé, že žádný systém **nesplňuje** všechny must have požadavky – u Microsoft Teams a Moodlu není zcela splněn požadavek na vyhodnocování vyplněných lekcí (F4) nebo přehled studentských řešení (F7), u e-learningových kurzů zase chybí možnost správy lekcí (F2), prohlížení studentových řešení nebo provoz na vlastní infrastruktuře (N5).

Vytvoření nového systému, který by tyto požadavky splňoval, tedy **má smysl**, a to jako samostatný celek – žádný ze stávajících systémů neumožňuje jednoduché rozšíření tak, aby chybějící požadavky začal podporovat.

Kapitola 3

Návrh

V této kapitole provedu návrh systému samotného. Nejprve navrhnu architekturu systému jako celku i jeho jednotlivých částí a pro každou část zvolím vhodnou technologii. V závěru pak určím způsob vývoje.

3.1 Architektura systému

Systém by měl na základě nefunkčních požadavků poskytovat webové rozhraní a API pro případná budoucí rozšíření. Zároveň je zde požadavek na modulární architekturu.

Jelikož systém pracuje s daty, které je potřeba ukládat, jednou z jeho součástí bude databáze. Tato data budou poskytována přes API, systém by měl obsahovat tedy i backend, který zajistí komunikaci s databází a poskytování API. Poslední ze základních částí systému je frontend, který poskytuje webové rozhraní pro uživatele.

3.1.1 Možné architektury

Existují tři základní způsoby, kterými lze databázi, backend, API a frontend poskytovat.

Prvním řešením je vytvořit systém skládající se ze dvou komponent – databáze a webového serveru zajišťující zároveň backend, propojení s databází, API i frontend. Takový server by mohl být napsán například v programovacím jazyce PHP s využitím frameworku Symfony. Modularizace by pak byla zajištěna vhodnou architekturou webového serveru.

Druhým řešením je rozdělit systém na tři komponenty – databázi, backend poskytující API a webový frontend využívající toto API. Možnou technologií pro takový backend by mohl být například Spring Web v programovacím jazyce Java, pro webový frontend například React v jazyce JavaScript. Modularizace by byla opět zajištěna vhodným rozdělením architektury backendu a frontendu.

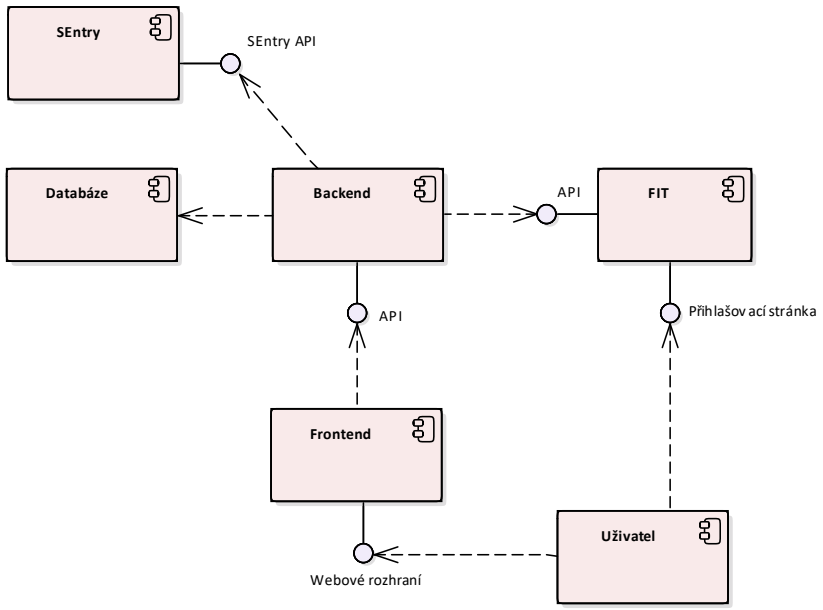
Třetím řešením je rozdělit komponenty z předchozího řešení podle jednotlivých modulů. Možné technologie zůstávají stejné, rozdělením by ale vznikla databáze, backend poskytující API a webový frontend pro každý modul. Jednotlivé moduly by pak mohly mezi sebou komunikovat s pomocí jejich API.

3.1.2 Zvolená architektura

První z výše uvedených řešení vede k velkému a komplexnímu systému, který vyžaduje precizní architekturu. Při zvolení nesprávné architektury může snadno dojít k nepřehlednému kódu a většímu výskytu chyb. Zároveň takovéto řešení typicky vede k server-side aplikaci a obtížnějšímu využití moderních technologií, jako je AJAX.

Druhé z uvedených řešení umožňuje rozumné rozdělení systému na jednotlivé části, kde jedna komponenta poskytuje API a druhá webové rozhraní pro uživatele. S tímto rozdělením mám zároveň nejvíce zkušeností.

Třetí řešení je vhodné pro rozsáhlé systémy v případě, kdy se předpokládá, že API jednotlivých modulů budou volána a využívána zvlášť. Tato architektura může být užitečná pro systémy vyžadující vyšší úroveň bezpečnosti, propojené s dalšími systémy, což pro moji práci neplatí.



Obrázek 3.1: Diagram komponent systému

Po zvážení těchto možností jsem zvolil druhé řešení. Systém se tedy skládá z databáze, backendu poskytujícího API a frontendu poskytujícího webového rozhraní. Rozdělení na moduly je řešeno uvnitř jednotlivých komponent vhodnou architekturou.

Systém bude také využívat fakultní API pro přihlášení pomocí OAuth a katederní Sentry [16] pro automatické nahlašování chyb přímo z backendu.

3.2 Datové úložiště

Na základě zjištěných požadavků musí systém ukládat jednotlivé lekce, týdny, kurzy, studenty, učitele, data související s vyplňováním a hodnocením úloh nebo žádostmi o pomoc. Zároveň by si systém měl pamatovat informace o přihlášení nebo uživatelská nastavení (tmavý režim, zvolený jazyk systému).

3.2.1 Cookies

Pro jednotlivé typy dat v systému jsou vhodné různé technologie pro jejich uložení. Informace o přihlášení nebo uživatelských preferencích se typicky ukládají jako soubor v prohlížeči uživatele, tzv. cookie. [17]

3.2.2 Binární data

Dále budou v systému uloženy studentská odevzdání jednotlivých úloh nebo vzorová data k jednotlivým lekcím a úlohám. Ta můžou být nejen textová (kód, textový komentář), ale také binární (archiv s testovacími daty, obrázek k lekci, studentův obrázek).

Binární data můžou být uložena v databázi jako binární objekt nebo jako soubor přímo na backendu. Vzhledem k tomu, že budou binární data získávána přímo uživatelem (načtení obrázku v prohlížeči, stažení souboru s testovacími daty), bude nejvhodnější řešení je ukládat jako soubory přímo na backendu.

3.2.3 Textová a objektová data

Textová a objektová data bývají typicky uložena v relační nebo NoSQL databázi. Relační databáze umožňují pevné schéma, indexaci dat, rychlé vyhledávání, NoSQL databáze zase volnější strukturu nebo lepší škálovatelnost. Oba dva typy databází obsahují objektová rozšíření, která umožňují ukládat objekty. Jelikož mají data systému pevně danou strukturu a jejich množství je relativně malé (tisíce studentů, jeden student vyřeší stovky úloh), pro tato data je vhodné využít relační databázi.

Mezi nejpoužívanější opensource relační databáze patří PostgreSQL [18], MariaDB [19] a MySQL [20]. Z komerčních databází jsou nejčastěji používané Oracle Database [21] a Microsoft SQL Server [22].

Vzhledem k povaze systému a požadavku na cenu nemá smysl uvažovat o komerčních databázích, které oproti opensource řešením poskytují větší bezpečnost a pokročilé nástroje pro administraci.

Mezi jednotlivými opensource relačními databázemi jsou pouze drobné rozdíly v syntaxi dotazovacího jazyka a způsobu ukládání dat. Z trojice výše uvedených opensource databází jsem zvolil MySQL, jelikož s ní mám největší zkušenosti.

3.2.4 Shrnutí

V systému tedy budu využívat cookies na frontendu pro ukládání informací o přihlášení a uživatelských preferencí, soubory na backendu pro binární data (archiv s testovacími daty, obrázky k lekcím) a relační databázi MySQL pro ukládání strukturovaných a textových dat (informace o lekcích, týdnech, úlohách, studentech, učitelích).

3.3 Architektura API

API slouží ke komunikaci mezi jednotlivými částmi systému. Architektura API určuje význam a podobu používaných zpráv. Mezi nejčastěji používané architektury patří SOAP, REST a GraphQL. [23]

3.3.1 SOAP

SOAP, z anglického Simple Object Access Protocol, zpřístupňuje data v XML (eXtended Markup Language) formátu a je silně standardizovaný. Byl vytvořen Microsoftem v roce 1999. SOAP zpráva se skládá z počáteční a koncové značky, těla s požadavkem nebo odpovědí, nepovinné hlavičky a informací o případných chybách. SOAP API je napsáno v WSDL (Web Service Description Language) popisem endpointů (= adresa, se kterou lze komunikovat) a popisem všech akcí, které lze provést. Výhodou SOAPu je jeho nezávislost na platformě a programovacím jazyce, vestavěné zpracování chyb a různá bezpečnostní rozšíření – proto se často používá pro přenos dat u bank a korporátů. Velkou nevýhodou je nutnost použití formátu XML, který způsobuje, že přenášené datové zprávy jsou velké. [23]

Použití architektury SOAP by z uživatelského hlediska způsobilo pomalejší komunikaci s API z důvodu většího množství přenášených dat a pomalejšímu parsování. Z hlediska programování je pak návrh API v architektuře SOAP pomalejší a XML (oproti formátu JSON) nemá nativní podporu v prohlížeči.

3.3.2 REST

REST, z anglického REpresentational State Transfer, je samopopisná architektura API pocházející z roku 2000. REST podporuje různé formáty, nejčastěji XML a JSON. REST není silně standardizovaný, musí ale splňovat určité požadavky, mezi které patří architektura klient-server, umožňující nezávislý vývoj jak klienta, tak serveru a bezestavovost, kdy veškerá data potřebná k zpracování požadavku jsou obsažena v požadavku samotném, server tedy nemusí ukládat dodatečná data.

Komunikace s REST API probíhá pomocí volání endpointů, které reprezentují jednotlivé zdroje. Operace s těmito zdroji jsou pak prováděny pomocí metod protokolu HTTP. Velkou výhodou RESTu je jeho samopopisnost, kdy při použití HATEOAS (Hypertext As The Engine of Application State) je u každého požadavku uvedena sada metadat odkazujících na informace, jak API používat. Nevýhodou HATEOAS je, že způsobuje nárůst velikosti přenášených zpráv. [23]

Použití RESTu s formátem JSON bez HATEOAS by vedlo k rychlejší komunikaci s API a snadnému parsování díky nativní podpoře formátu JSON v jazyce JavaScript. Návrh API v architektuře REST je navíc časově méně náročný než v architektuře SOAP.

3.3.3 GraphQL

GraphQL [24] je architektura umožňující provedení přesného požadavku. Použití GraphQL je vhodné, pokud data obsahují hodně komplexních entit, které se k sobě vztahují. Návrh GraphQL spočívá ve vytvoření schématu všech dotazů,

které lze provést a všech datových typů, které můžou být vráceny v odpovědi. To zajišťuje, že server dokáže na požadavek vždy odpovědět a data jsou vrácena přesně v podobě, která je požadována. Nevýhodou je ale složitost zpracování požadavků a také množství práce před začátkem vývoje. [23]

Výhodou GraphQL je možnost získat všechna data v jednom dotazu, což ale způsobuje pomalejší komunikaci s API. Z programátorského hlediska je navíc GraphQL náročnější na naučení a jeho návrh vyžaduje více času.

3.3.4 Zvolená architektura

Po analýze možných architektur jsem pro systém vybral architekturu REST, která nejlépe odpovídá jeho potřebám: je přehledná, jednoduchá a snadná na použití, nevyžaduje velké zabezpečení ani standardizované rozhraní, nevykonává komplexní požadavky a nepracuje s velkým množstvím dat. S touto architekturou se navíc setkávají studenti bakalářského studia v předmětu Technologie Java, takže je vhodná i pro snazší pochopení systému studenty, kteří budou na práci na systému v budoucnu navazovat.

3.4 Technologie pro backend

Pro backend, který poskytuje REST API, jsem měl na výběr z různých frameworků pro tvorbu webových aplikací. Mezi nejčastěji používané patří Slim, Nette, Symfony, Laravel, ASP.NET, Ktor a Spring. Tyto frameworky jsem rozdělil podle programovacích jazyků, ve kterých jsou napsané.

3.4.1 PHP

PHP je dynamicky typovaný jazyk určený primárně pro vývoj webových aplikací. [25] Vzhledem ke své rozšířenosti je podporován většinou webových hostingů a pro nasazení webu napsaném v tomto jazyce stačí HTTP server, jako je Apache [26] nebo Nginx [27]. V tomto jazyce jsou dostupné frameworky Slim, Nette, Symfony a Laravel.

Framework Slim [28] je specifický tím, že sám o sobě obsahuje pouze základní funkce a třídy pro zpracování HTTP požadavků a routování. Pro práci s databází, souborovým systémem či autentizaci je nutné využít externí knihovny nebo si napsat vlastní nástroje. Případný vývoj backendu v tomto frameworku by tak znamenal hodně práce s hledáním vhodných knihoven a s tím související potenciální bezpečnostní hrozby.

Další frameworky Nette [29], Symfony [30] a Laravel [31] se již skládají ze sady „komponent“, které umožňují práci s databází a souborovým systémem, generování HTML kódu s pomocí šablon nebo zpracovávání chyb.

S programovacím jazykem PHP a frameworky Nette a Symfony mám zkušenosti ze střední školy, kdy jsem v nich psal jednoduché webové aplikace (backend i frontend). Jakmile jsem ale potřeboval napsat komplexnější backend, vzhledem k slabé typové kontrole a absenci vlastností jazyka PHP, jako jsou šablony nebo anotace začal vznikat velmi komplexní kód, který nebyl udržitelný a obsahoval mnoho chyb.

3.4.2 C#

C# [32] je staticky typovaný jazyk vyvíjený firmou Microsoft. Oproti PHP se jedná o kompilovaný jazyk, vzniklý kód je tedy třeba nejprve zkompilovat a až následně nasadit, což vede k rychlejšímu běhu aplikace.

V jazyce C# je napsán framework ASP.NET. Ten je silně propojen s operačním systémem Windows, Microsoft ale poskytuje Docker kontejner, díky kterému lze framework spustit i na jiných platformách.

S jazykem C# ani frameworkem ASP.NET nemám žádné zkušenosti.

3.4.3 Java a Kotlin

Java [33] je silně typovaný jazyk vyvíjený firmou Oracle. Jedná se sice o kompilovaný jazyk, kompilace ale narozdíl od C# neprobíhá do strojového kódu, ale do Java „bytecode“. Spuštění pak probíhá ve speciálním virtuálním stroji – JVM. To vede k širší kompatibilitě za cenu mírně pomalejšího běhu aplikace.

V programovacím jazyce Java je napsán framework Spring [34], který obsahuje sadu komponent pro práci s autentizací, routováním i databází.

Kotlin [35] je programovací jazyk vyvinutý firmou Jetbrains. Je plně kompatibilní s Javou, veškerý kód lze tedy přeložit do Java bytecode. Hlavní výhodou programovacího jazyka Kotlin je jeho jednoduchost a množství tzv. DSL (doménově specifických jazyků), které umožňují psát kód přehledně a stručně.

Pro Kotlin existuje také framework Ktor [36]. Ten ale neobsahuje tolik komponent, jako Spring [34], který je s Kotlinem také plně kompatibilní.

S jazyky Java a Kotlin mám dlouholeté zkušenosti už ze střední školy. S frameworkem Spring jsem se pak seznámil v 2. ročníku bakalářského studia v předmětu Technologie Java a využil jsem ho pro tvorbu backendu pro další navazující semestrální práce. V tomto frameworku jsem také napsal backend pro svou bakalářskou práci.

3.4.4 Zvolená technologie

Při finální volbě technologie jsem uvažoval svoje zkušenosti s danými programovacími jazyky a frameworky, rozsah projektu, ale také možnost navazující práce na systému bakalářskými studenty.

Po zvážení všech těchto kritérií jsem se rozhodl pro využití frameworku Spring – mám s ním největší zkušenosti, je ideální pro projekt středního rozsahu a tento framework je navíc vyučován v bakalářském předmětu Technologie Java povinném pro studenty specializace Softwarové inženýrství.

Jako programovací jazyk jsem zvolil Kotlin, jelikož s ním mám dlouholeté zkušenosti, je přehledný, moderní, a plně kompatibilní s Javou. Pro studenty, kteří by se tento jazyk chtěli naučit detailně, je navíc na fakultě k dispozici volitelný předmět Programování v jazyku Kotlin, kterým jsem sám prošel.

3.5 Architektura backendu

Jako architekturu backendu jsem zvolil třívrstvou architekturu, skládající se z prezentační vrstvy poskytující výše popsané API, datové vrstvy získávající data z databáze a business vrstvy, která získá informace z datové vrstvy a předá je do prezentační vrstvy.

3.6 Frontend

Cílem frontendu je komunikace s backendem s pomocí REST API a prezentace získaných dat uživateli. Pro vývoj frontendu se používá dynamicky typovaný programovací jazyk JavaScript [37] a jeho typová nadstavba TypeScript [38]. Mezi nejčastější frameworky pro tvorbu frontendu patří Angular, React a Vue.js.

Ani s jedním z těchto frameworků nemám předchozí zkušenosti. S JavaScriptem mám dlouholeté zkušenosti – využíval jsem ho při tvorbě jednoduchých webových skriptů v průběhu studia na střední škole. S TypeScriptem žádné předchozí zkušenosti nemám.

3.6.1 Angular

Angular [39] byl vyvinut v roce 2010 společností Google. Je napsán v jazyce TypeScript. Má komplexní strukturu projektu, skládající se ze služeb, komponent a modulů. Tato komplexní struktura společně s jazykem TypeScript způsobuje, že je Angular vhodnější pro rozsáhlé projekty, ale složitější na naučení. [40]

3.6.2 React

React [41] byl vyvinut v roce 2013 společností Facebook. Je napsán v jazyce JavaScript a obsahuje sadu komponent včetně prvků pro tvorbu webových rozhraní. Jelikož se jedná pouze o knihovnu, je nejjednodušší na naučení. Na rozdíl od Angularu nevyužívá React skutečný, ale virtuální DOM (Document Object Model), což umožňuje rychlejší překreslování stránky. [40]

3.6.3 Vue.js

Vue.js [42] byl vyvinut v roce 2014 bývalým zaměstnancem Googlu, Evan You. Využívá jak JavaScript, tak TypeScript, a skládá se z jednotlivých komponent. Je jednodušší než Angular, ale složitější než React. Stejně jako React i Vue.js využívá virtuální DOM pro rychlejší překreslování stránky. [40]

3.6.4 Zvolená technologie

Jelikož jsem s ani jedním z výše uvedených frameworků dosud nepracoval, uvažoval jsem při výběru frameworku především jeho jednoduchost pro naučení vůči rozsahu projektu. Abych se s jednotlivými frameworky lépe seznámil, prošel jsem si v každém tutoriál na tvorbu jednoduché aplikace pro seznam úkolů.

Po projití tutoriálů a zvážení všech kritérií jsem zvolil framework Vue.js – je jednoduchý na naučení, ideální pro projekt středního rozsahu a jeho struktura mi navíc přišla nejpodobnější struktuře technologií Jetpack Compose a SwiftUI, které znám z vývoje Android a iOS aplikací. Vue.js také podporuje znovupoužitelné komponenty, což usnadní vývoj modulární architektury.

Jednoduchost frameworku je navíc důležitá i pro návaznou práci na systému – na FITu se totiž žádný frontendový framework v povinných ani volitelných předmětech nevyučuje.

Jako programovací jazyk jsem zvolil JavaScript, jelikož jsem s ním už měl předchozí zkušenosti.

3.7 Vyhodnocování kódu

Při procesu vývoje studenti potřebují kód editovat (1), zkompilovat *(a slinkovat)* (2), spustit (3) a odeslat výsledky (4).

Pro editaci kódu existují dvě základní možnosti: online editor a klasický (offline) editor. Výhoda online editoru je především v tom, že se kód pravidelně ukládá přímo na server a není zde tedy potřeba manuálního nahrávání kódu. Nevýhodou je absence pokročilejších možností úpravy kódu a syntax highlightingu, kterou disponuje většina offline editorů.

Pro kompilaci, spouštění kódu a odeslání výsledků existují dvě možnosti: vzdálená (na serveru) a lokální (u studenta).

3.7.1 Server-side

Při kompilaci a spouštění na serveru je zajištěno stejné prostředí pro všechny uživatele a je zde jistota, že odeslané výsledky nebudou zfalšovány. Dílčí nevýhodou je ale zatížení serveru především při spouštění kódu, kdy je nutno zavést limit délky běhu programu nebo počtu odevzdání, a i tak může jedno odevzdání zatížit systém na jednotky sekund. Při uvažovaném použití na cvičení by tak v případě, že úlohu odevzdá najednou 50 studentů, čekal poslední student na vyhodnocení až 15 minut.

3.7.2 Client-side

Druhou možností je kompilace a spouštění kódu lokálně s následným nahráním na server. Velkou výhodou tohoto řešení je rychlost a také nezatěžování jednoho centrálního serveru. Pokud je program pomalý, může běžet libovolně dlouhou dobu, protože zatěžuje pouze počítač studenta.

Nevýhodou je bezpečnost, kdy nelze zaručit, že výsledky nebudou zfalšovány. Vzhledem k tomu, že na cvičení se ale uděluje pouze nepatrný počet bodů a to pouze za manuálně hodnocené bonusové úlohy za aktivitu, nejedná se o problém. Vyhodnocení kódu má totiž k dispozici i učitel, který si při manuálním hodnocení může korektnost studentova řešení ověřit.

Nahrání na server může probíhat přímo z kódu pomocí knihovny pro komunikaci s API. Takové nahrávání výsledků ale znamená velmi vysoké riziko zfalšování studentem, navíc jsou v takovém případě testovací vstupy a výstupy přímo dostupné studentovi. Alternativním řešením je použití WebAssembly [43].

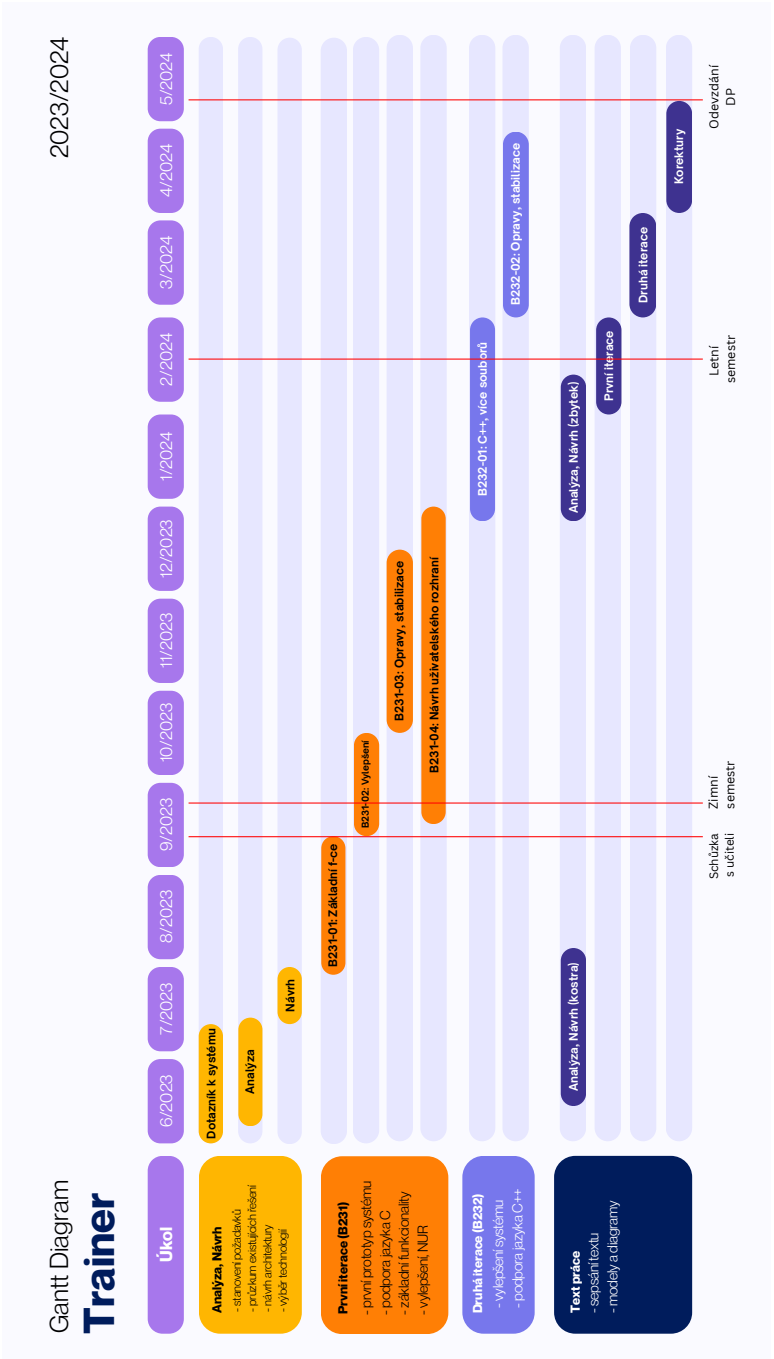
Při použití technologie WebAssembly dochází ke kompilaci a spouštění kódu přímo v prohlížeči. Student tak nemá přímý přístup k testovacímu programu a nahrávání výsledků na API probíhá z frontendu, riziko zfalšování je tedy menší.

3.7.3 Vybrané řešení

Po zvážení všech výhod a nevýhod způsobu vyhodnocování kódu jsem se rozhodl pro lokální (client-side) vyhodnocování kódu s využitím WebAssembly. Student tedy nahraje svůj kód na frontend, ten ho zkompiluje, následně spustí a nahraje výsledky včetně kódu na API.

3.8 Plán vývoje

Po dokončení analýzy a návrhu v červenci 2023 jsem určil plán vývoje systému. Systém plánuji využít ve dvou předmětech, PA1 a PA2, které poběží postupně v zimním a letním semestru. Rozhodl jsem se proto pro vývoj ve dvou iteracích.



Obrázek 3.2: Ganttův diagram plánu práce na systému

1. iterace

První iterace (v diagramu B231-0X) zahrnuje funkcionality potřebné pro použití v předmětu PA1. V PA1 se vyučují základní koncepty programování v jazyce C, systém tedy ještě nemusí podporovat jazyk C++ a kompilaci více souborů – v PA1 probíhá vývoj vždy pouze v jednom souboru.

Na vývoj první iterace je velmi omezený čas, jelikož první schůzka vyučujících předmětu PA1, kde systém plánuji prezentovat, proběhne již 13. 9. 2023, semestr pak začíná 25. 9. 2023. Implementace většiny funkcionalit první iterace tak musí proběhnout v průběhu srpna a září. Dokončení zbylých funkcionalit se ještě dá tolerovat během první poloviny října, kdy se v PA1 studenti teprve seznamují s jazykem C a nevyužívají se pokročilejší mechanismy, jako je dynamická alokace paměti.

V průběhu zimního semestru se také budu účastnit magisterského předmětu NI-NUR (Návrh uživatelského rozhraní), ve kterém budu v týmu analyzovat vzniklé uživatelské rozhraní. Výstupy z tohoto předmětu pak budu moct použít v závěru semestru (prosinec 2023) pro zlepšení uživatelského rozhraní systému.

2. iterace

Druhá iterace (v diagramu B232-0X) zahrnuje vylepšení nutná pro předmět PA2. Oproti PA1 se v PA2 vyučuje jazyk C++, který vyžaduje podporu výjimek a kompilaci více souborů.

Pro vývoj druhé iterace bude čas během zkouškového období zimního semestru a prvních tří týdnů letního semestru (leden a únor 2024). V březnu 2024 pak bude prostor na opravy a drobná vylepšení. Od dubna pak předmět PA2 přejde do konzultačního módu, kdy už systém nebude aktivně využíván při výuce a já se budu moct více věnovat psaní textu této práce.

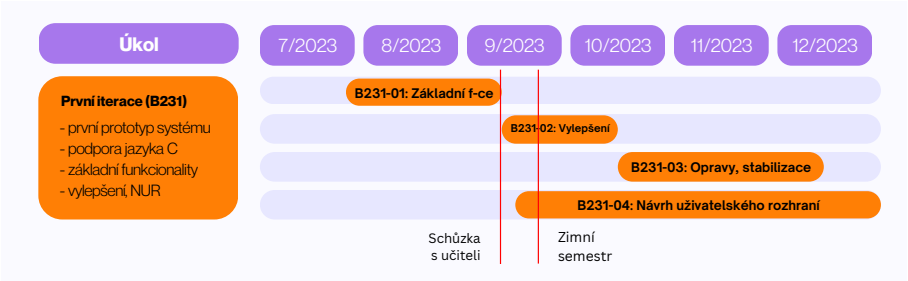
Kapitola 4

První iterace

V této kapitole provedu první návrh databáze, API a uživatelského rozhraní. Následně popíšu implementaci první iterace backendu a frontendu a výzvy s ní související. Na závěr první iteraci nasadím na školní infrastrukturu, kde ji otestuji a zhodnotím nejzávažnější problémy k opravení do druhé iterace.

4.1 Popis

První iterace (B231) probíhala v průběhu prázdnin a výukové části zimního semestru (od srpna do prosince 2023). Cílem iterace bylo nejprve vytvoření funkčního prototypu systému, který by mohl být 13. 9. prezentován na schůzce vyučujícím PA1 (B231-01), jeho následné uvedení do provozu (B231-02, září až říjen) a stabilizace (B231-03, říjen až listopad). Druhým cílem pak bylo tento prototyp řádně otestovat, zjistit zpětnou vazbu od uživatelů a také analyzovat vzniklé uživatelské rozhraní v rámci předmětu NI-NUR (B231-04, říjen – prosinec).



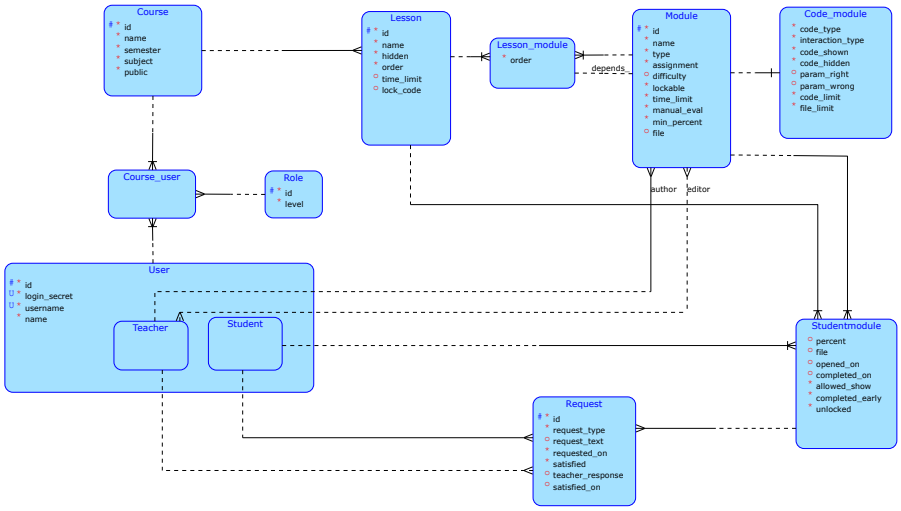
Obrázek 4.1: Časový plán první iterace

4.2 Databáze

Pro první iteraci systému jsem ještě nepočítal s organizací do týdnů, lekce jsou organizovány pouze atributem pořadí lekce v rámci kurzu. Kurz je součástí běhu předmětu s kódem v konkrétním semestru a odpovídá paralelce (studijní skupině), která má učitele a studenty.

V lekcích jsou jednotlivé úlohy (moduly), které lze skrývat, zamykat a nastavit jim včasné vyplnění nebo manuální ohodnocení učitelem. Studenti moduly vyplňují, můžou povolit jejich anonymizované zveřejnění a pokud potřebují pomoc, vytvoří žádost, na kterou učitel odpoví.

Pro kódové moduly je navíc vytvořena pomocná entita, která obsahuje typ kódového modulu, způsob interakce, jednotlivé kódy a parametry pro spouštění programu.



Obrázek 4.2: Prvotní konceptuální model databáze

4.3 Návrh API

Z konceptuálního modelu vyplývá, že systém bude vyžadovat zdroj pro kurzy (/courses), lekce (/lessons), moduly (/modules), kódový modul (/code) a uživatele (/auth).

4.3.1 Zdroj /auth

Na uživateli bude přístupný zdroj /auth, který umožní přihlášení uživatele (POST /auth) pomocí kódu získaného z OAuth. Zároveň umožní získání nastavení pro OAuth pomocí GET /auth/settings.

4.3.2 Zdroj /courses

Systém umožní získat seznam kurzů aktuálně přihlášeného uživatele (GET /) a detail konkrétního kurzu včetně všech lekcí, které obsahuje (GET /id). Zároveň umožní *(prozatím fixním uživatelům)* vytváření nových kurzů daného předmětu (POST), úpravu kurzu konkrétního předmětu (PATCH /id) a jeho smazání (DELETE /id). Učiteli umožní nastavení kódu pro připojení do kurzu (PUT /id/secret), studentovi připojení do kurzu se zadaným kódem (PUT /me).

Dále systém umožní získat seznam všech uživatelů v kurzu (GET /id/users), všech učitelů v kurzu (GET /id/users/teachers) a detail uživatele v kurzu (GET /id/users/userId). Pevně určeným učitelům umožní přidávání uživatelů do kurzu (POST /id/users) a jejich odstranění (DELETE /id/users/userId).

4.3.3 Zdroj /lessons

Systém umožní uživateli zjistit aktuálně probíhající lekce (GET /current) a detail konkrétní lekce (GET /id). Také studentovi umožní odemknout všechny moduly v dané lekci (POST /id/code). Učiteli pak umožní vytvářet lekce (POST /), upravovat stávající lekce (PATCH /id), mazat je (DELETE /id) a kopírovat (POST /id/courses/courseId).

Dále učiteli umožní získat seznam studentů v lekci (GET /id/users), detail konkrétního studenta v lekci (GET /id/users/userId) a resetovat studentův pokrok v lekci (DELETE /id/users/userId).

Učitel bude taky moct zkopírovat modul do lekce (POST /id/modules), přidat stávající modul do lekce (PUT /id/modules/moduleId) nebo modul z lekce odebrat (DELETE /id/modules/moduleId).

V neposlední řadě systém umožní studentovi nahrání jeho řešení daného modulu (PUT /id/modules/moduleId/data/me) a vytvoření či úpravu žádosti (PUT /id/modules/moduleId/requests/me). Učitel pak bude moct zobrazit si studentovo řešení (GET /id/modules/moduleId/data/userId) a odpovídat na jeho žádosti (PUT /id/modules/moduleId/requests/userId).

4.3.4 Zdroj /modules

Systém umožní učiteli získat seznam všech modulů (GET /) a detail konkrétního modulu (GET /id). Učitel bude také moduly vytvářet (POST /), upravovat (PATCH /id) a mazat (DELETE /id).

4.3.5 Zdroj /code

Systém umožní učiteli vytvářet kódový modul (POST /), upravit ho (PATCH /id) a mazat (DELETE /id). Student si bude moct zobrazit detail a informace kódového modulu (GET /id).

4.3.6 Zdroj /notifications

Systém umožní jak studentovi, tak učiteli, zjistit poslední (ne)vyřešené žádosti o pomoc a ohodnocení pomocí takzvaných notifikací. U notifikací půjde zjistit jejich přehled (GET /) a označit je jako přečtené (DELETE /).

4.4 Backend

Pro backend jsem v předešlé kapitole (Technologie pro backend) zvolil technologii Spring Web [34] a programovací jazyk Kotlin [35]. Při implementaci jsem vycházel z nadstavby pro základní CRUD operace, kterou jsem naprogramoval v rámci bakalářské práce [44]. Tato nadstavba ve spolupráci s knihovnami Springu a vlastnostmi jazyka Kotlin umožňuje rychlou a přehlednou implementaci jednotlivých vrstev.

4.4.1 Datová vrstva

Datová vrstva obsahuje dva balíčky, Entity a Repository, které jsou implementované pomocí frameworku Spring JPA [45].

Každá entita reprezentuje jednu tabulku v databázi a odpovídá datové třídě (`data class`) v jazyce Kotlin. Definice sloupců probíhá s pomocí anotace `@Column` na jednotlivých třídních atributech. Vazby mezi tabulkami a integritní omezení jsou definovány dalšími anotacemi (např. `@OneToMany`, `@JoinColumn`).

```
@Entity
data class Lesson(
    @Column(nullable = false) val name: String,
    @Column(nullable = false, columnDefinition = "boolean
    default false")
    val hidden: Boolean,
    @Column(name = "`order`", nullable = false) val order: Int,

    @Column(nullable = false, columnDefinition = "LONGTEXT")
    @Lob val description: String,
    @Column(nullable = true) val lockCode: String?,
    @Column(nullable = true) val timeLimit: Timestamp?,

    @ManyToOne
    @OnDelete(action = OnDeleteAction.CASCADE)
    @JoinColumn(name = "course_id", nullable = false)
    val course: Course,

    @OneToMany(mappedBy = "lesson")
    val modules: List<LessonModule>,

    override val id: Int = 0
) : IEntity(id)
```

Výpis kódu 1: Ukázka definice třídy `Lesson` reprezentující lekci. Lze si všimnout různých datových typů pro jméno, indikaci skryté lekce, nebo třeba časový limit pro včasné splnění úloh v lekci. Parametrem `nullable` se ve spolupráci s nullable typem `Timestamp?` je zaručena nullabilita typu. Popis lekce může být velmi dlouhý, proto je jeho typ `LONGTEXT` a je označen anotací `@Lob`. Lekce má N:1 vazbu na kurz, proto je zde použita anotace `@ManyToOne` a `@JoinColumn` s definicí akce `CASCADE` při smazání kurzu. Obdobně má vazbu 1:M k modulům přes pomocnou entitu `LessonModule`, proto je zde použita anotace `@OneToMany`.

Každá entita pak implementuje metody `canView` a `canEdit`, které určují, zda-li může uživatel danou entitu vidět (Read operace v CRUD) a upravovat (Create, Update, Delete operace v CRUD). Tato implementace umožňuje snadnější kontrolu práv, jelikož lze sdílet logiku oprávnění mezi jednotlivými entitami (například metoda `canView` u lekce volá interně `course.canView`).

```
override fun canView(user: User?)
    = (!hidden && course.canView(user)) || (hidden && user !=
    null && course.canEdit(user))

override fun canEdit(user: User)
    = course.canEdit(user)
```

Výpis kódu 2: Ukázka metod `canView` a `canEdit` ve třídě `Lesson` reprezentující lekci. Uživatel může lekci vidět, pokud není skrytá a může vidět kurz, ve kterém je (= je jeho členem nebo je kurz veřejný). Skryté lekce vidí pouze uživatel s právy na úpravu kurzu (= učitel).

Balíček Repository obsahuje definici rozhraní pro jednotlivé repozitáře. Spring JPA s pomocí těchto definic vygeneruje konkrétní implementaci s dotazy pro danou databázi. Každý repozitář z důvodu kompatibility s dalšími vrstvami dědí ze šablony `IRepository<T: Entity>`, která s využitím JPA vygeneruje metody pro jednotlivé základní CRUD operace.

V případě potřeby hledání podle jiných kritérií, než je identifikátor entity, je využita anotace `@Query` a dotaz napsaný v JPQL [46] (Jakarta Persistence Query Language).

```
@Repository
interface CourseRepository: IRepository<Course> {
    @Query("SELECT c FROM Course c WHERE c.secret = :secret")
    fun getBySecret(secret: String) : Course
}
```

Výpis kódu 3: Ukázka rozhraní `CourseRepository` pro definici repozitáře pro kurz. Je zde s pomocí anotace `@Query` a JPQL definována metoda pro vyhledání kurzu podle připojovacího kódu (`getBySecret`).

4.4.2 Business vrstva

Business vrstva se skládá ze dvou balíčků: DTO a Service. DTO slouží ke komunikaci mezi prezentační vrstvou (Controllery) a Service. DTO jsou implementovány jako datové třídy (`data class`) a podle jednotlivých operací se dělí na čtyři typy: `FindDTO` a `GetDTO` sloužící k získání dat z business směrem k prezentační vrstvě, `CreateDTO` a `UpdateDTO` k přenosu dat z prezentační do business vrstvy.

Oproti mé bakalářské práci [44] zde došlo k rozdělení původního jednoho „čtecího“ `ReadDTO` na dva typy. `FindDTO` se používá v seznamu, například při získání všech lekcí v daném kurzu a obsahuje pouze základní informace (jméno, identifikátor) a kontextuální informace v rámci seznamu (pořadí v rámci kurzu). `GetDTO` se používá při získání detailu jedné entity a obsahuje všechna data, v případě lekce (`LessonGetDTO`) tedy obsahuje kromě základních informací ještě seznam všech modulů (`List<ModuleFindDTO>`). Toto rozdělení má výhodu v menším objemu přenášených dat a zabraňuje rekurzi (`Lesson FindDTO` má atribut `course`, který je typu `CourseFindDTO`, neobsahuje tedy seznam lekcí, který by vedl k rekurzi při pokusu o výpis).

Chování „zápisových“ DTO zůstalo stejné, tedy `CreateDTO` obsahuje povinně všechny atributy a slouží k vytváření, `UpdateDTO` má atributy nepovinné, slouží k úpravě a pokud není atribut uveden, tak není změněn *(s výjimkou nullable polí, tam musí dojít ke změně vždy)*.

```
data class LessonFindDTO(override val id: Int, val name: String,
val hidden: Boolean, val order: Int, val lockCode: String?, val
timeLimit: Timestamp?, val progress: Int?) : IFindDTO

data class LessonGetDTO(override val id: Int, val course:
CourseFindDTO, val name: String, val hidden: Boolean, val
lockCode: String?, val timeLimit: Timestamp?, val modules:
List<ModuleFindDTO>) : IGetDTO

data class LessonCreateDTO(val courseId: Int, val name: String,
val hidden: Boolean, val order: Int, val lockCode: String?, val
timeLimit: Timestamp?) : ICreateDTO

data class LessonUpdateDTO(val name: String?, val hidden:
Boolean?, val order: Int?, val lockCode: String?, val timeLimit:
Timestamp?) : IUpdateDTO
```

Výpis kódu 4: Ukázka DTO pro lekci (Lesson). `LessonFindDTO` obsahuje kromě základních dat taky informaci o pokroku uživatele v lekci (`progress`), `LessonGetDTO` obsahuje navíc seznam modulů (`modules`). Obě čtecí DTO obsahují `FindDTO` pro kurz, aby nedošlo k zacyklení při výpisu. `LessonCreateDTO` má navíc parametr pro ID kurzu, které se nastavuje pouze při vytváření lekce. Oproti `LessonUpdateDTO` jsou atributy pro jméno lekce (`name`), indikátor skrytí lekce před studenty (`hidden`) a pořadí v rámci kurzu (`order`) nenulové.

Balíček Service obsahuje třídy, které slouží k zpracování dat z datové vrstvy a jejich převedení do prezentační vrstvy. Všechny Service dědí ze základní šablony `IServiceBase<T>`, která obsahuje metody pro získání uživatele podle jeho id (`getUser`) a zpracování výjimek v předaném bloku pracující s datovou vrstvou (`tryCatch`).

Stejně jako v bakalářské práci [44] jsem zde vytvořil šablonu `IServiceBase` se základními implementacemi CRUD metod. Oproti bakalářské práci jsem ale v jednotlivých metodách často potřeboval dodatečnou logiku *(přidání uživatele do kurzu, který právě vytvořil; ověření data poslední úpravy modulu při jeho úpravě)*, kvůli časovému tlaku jsem ale šablonu ponechal a jednotlivé metody jsem přetížil.

Kromě základních metod ještě Service obsahují další ne-CRUD metody. Ty jsou typicky implementovány následovně: nejprve proběhne získání entity, se kterou pracuji (metoda `getReferenceById` na repozitáři) a kontrola oprávnění (metody `canView` a `canEdit` na entitě). Následně jsou provedeny požadované operace (s využitím základních nebo vlastních metod injectnutých repozitářů) a je vráceno výsledné DTO.

```
fun cloneLesson(id: Int, courseId: Int, user: UserFindDTO?)
= tryCatch {
    val lesson = repository.getReferenceById(id)
    val newCourse = courseRepository.getReferenceById(courseId)

    // Authorisation
    val userEntity = getUser(user)
    if (!lesson.canEdit(userEntity) ||
    !newCourse.canEdit(userEntity))
        throw ResponseStatusException(HttpStatus.FORBIDDEN)

    // Step 1: create the lesson (hidden, unlocked)
    val newLesson = saveAndFlush(Lesson(
        lesson.name, true, lesson.order, null, null, newCourse,
        listOf()
    ))

    // Step 2: copy the modules
    lessonModuleRepository.saveAllAndFlush(lesson.modules.map {
        lm -> LessonModule(newLesson, lm.module, lm.order,
        lm.dependsOn) // Keep the order, depends and module
    })

    // Step 3: return lesson with modules
    newLesson.toGetDTO(userEntity)
}
```

Výpis kódu 5: Ukázka metody pro kopírování lekce mezi kurzy v `LessonService`. Nejprve se zkontroluje, zda má uživatel práva k úpravě kopírované lekce a kurzu, do kterého se kopíruje. Pak se vytvoří nová lekce v daném kurzu, zkopírují se jednotlivé moduly a vrátí se nově vytvořená lekce.

4.4.3 Prezentační vrstva

Prezentační vrstva je tvořena jediným balíčkem Controller. Všechny třídy z tohoto balíčku dědí ze základní třídy `IControllerAuth`. Tato třída obsahuje stejně jako v bakalářské práci [44] metodu `authenticate`, která zkontroluje přítomnost hlavičky s přihlašovacím klíčem a zavolá požadovanou akci s aktuálně přihlášeným uživatelem.

Třídy z balíčku Controller zpřístupňují jednotlivé metody Service v podobě jednotlivých HTTP metod na REST API. Toho je docíleno využitím anotace `@RestController` na každé třídě a následně anotacemi `@GetMapping`, `@PostMapping`, `@PutMapping`, `@PatchMapping` a `@DeleteMapping` podle jednotlivých HTTP metod. Jednotlivé parametry z URL jsou popsány v hlavičce metody pomocí `@PathVariable`, data v těle požadavku anotací `@RequestBody`.

Pro Controllery, které zpřístupňují jednotlivé CRUD metody, stejně jako v bakalářské práci využívám šablonu `IControllerImpl<T, F, G, C, U>`, která poskytuje implementaci CRUD metod. Jejich viditelnost lze ovlivnit anotací `@VisibilitySettings` s nastavením hodnoty u příslušné metody na `NONE`.

Při implementaci ne-CRUD metod je postup následující: nejprve proběhne autentifikace pomocí metody `authenticate` a následně je zavolána odpovídající metoda Service a vrácen její výsledek.

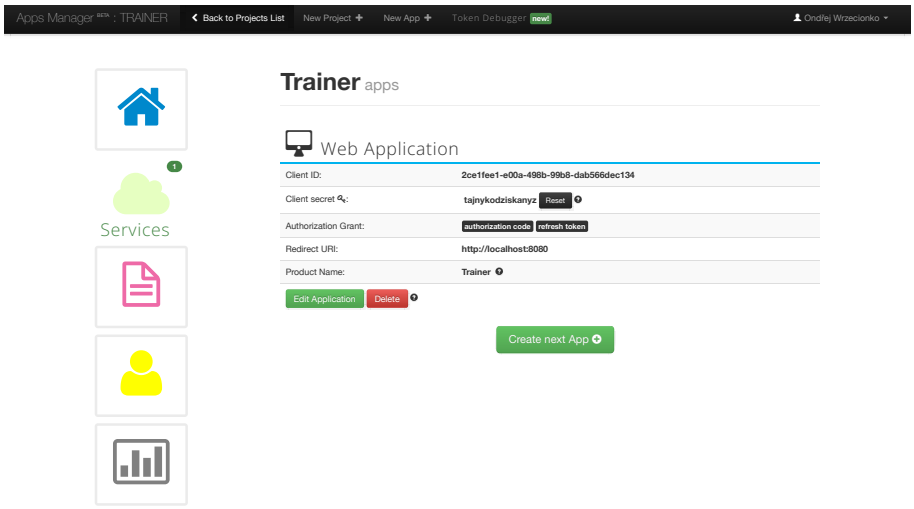
```
@PostMapping("/{id}/courses/{courseId}")
fun cloneLesson(@PathVariable id: Int, @PathVariable courseId:
Int, request: HttpServletRequest)
    = authenticate(request, VisibilitySettings.LOGGED) { user ->
    service.cloneLesson(id, courseId, user) }
```

Výpis kódu 6: Ukázka metody pro kopírování lekce mezi kurzy v `LessonController`. Anotací `@PostMapping` s odpovídající URL je definováno, že metoda bude přístupná jako POST /id/courses/courseId. Hodnota identifikátoru lekce i kurzu je získána s využitím anotace `@PathVariable` a dále je zde injectnutý HTTP požadavek pro potřeby autentifikace. Po zavolání metody se provede autentifikace a se získaným uživatelem a identifikátory je následně zavolána metoda pro zkopírování lekce na `LessonService`.

4.4.4 Přihlašování

Pro přístup k jednotlivým REST metodám je vyžadován přihlašovací kód (login secret). Tento kód frontend získá během procesu přihlašování. Během analýzy byl stanoven požadavek na přihlášení přes FIT OAuth, proto jsem se prozatím rozhodl podporovat přihlašování pouze přes OAuth.

Pro získání údajů pro OAuth je třeba nejprve získat identifikátor klienta (`clientId`), kód (`clientSecret`) a URL adresu pro přesměrování (`redirect Uri`). Tyto údaje lze získat po zaregistrování aplikace v FIT Apps Manageru dostupném na adrese `https://auth.fit.cvut.cz/manager/`.



Obrázek 4.3: Získání údajů pro OAuth v Apps Manager

Získané údaje je následně potřeba nastavit v konfiguračním souboru frameworku Spring `application.properties`.

```
auth.clientId=2ce1fee1-e00a-498b-99b8-dab566dec134
auth.clientSecret=tajnykodziskanyz
auth.redirectUri=http://localhost:8080/login
```

Výpis kódu 7: Konfigurace pro OAuth

Backend pak identifikátor klienta a adresu pro přesměrování poskytuje frontendu přes REST API na adrese /auth/settings. Ten na základě těchto údajů získá kód, který zašle zpět backendu. Backend kód odešle na FIT autorizační endpoint a získá access token, s pomocí něhož získá uživatelské jméno přihlášeného uživatele. Na základě tohoto jména je vyhledán nebo vytvořen uživatel, který je vrácen.

```
private fun fitAuth (code: String) : String = tryCatch {
    val body  = networkService.fitAuthRequest(code, clientId,
    clientSecret, redirectUri)
    val token = Gson().fromJson(body, FitToken::class.java)

    val body2 =
    networkService.fitTokenInfoRequest(token.access_token)
    val decoded = Gson().fromJson(body2,
    FitTokenInfo::class.java)
    decoded.user_name
}

// Class for token
private data class FitToken (val access_token: String, val
token_type: String, val refresh_token: String, val expires_in:
Int, val scope: String)

// Class for user information
private data class FitTokenInfo (val aud: List<String>, val exp:
Int, val user_name: String, val authorities: List<String>, val
client_id: String, val scope: List<String>)
```

Výpis kódu 8: Ukázka získání uživatelského jména z OAuth kódu. Nejprve je získán access token na základě kódu, následně jsou dalším požadavkem získány informace o uživateli, včetně jeho uživatelského jména.

4.4.5 Ukládání souborů

Jak již bylo zmíněno v návrhu, v systému budou ukládána binární data, jako soubory s testovacími daty nebo obrázky k lekcím / úlohám. Pro jejich ukládání je nutné využívat HTTP metodu POST, a to kvůli omezení prohlížeče, který neumožňuje odesílat soubory v jiných metodách (PUT / PATCH).

V Controlleru je přijímání souborů indikováno v anotaci `@PostMapping` specifikací `consumes = [MediaType.MULTIPART_FORM_DATA_VALUE]`. Parametr se souborem je pak popsán anotací `@RequestPart("jmeno")` a má v Kotlinu typ `MultipartFile`.

```
val uploadsPath = Paths.get(ModuleService.UPLOADS_PATH_IMAGES)
if (!uploadsPath.exists())
    uploadsPath.createDirectories()

val extension = when (image.contentType) {
    "image/jpeg" -> "jpeg"
    "image/png" -> "png"
    else -> throw
    ResponseStatusException(HttpStatus.UNSUPPORTED_MEDIA_TYPE)
}

val fileName = "${ModuleService.generateFileName()}.$extension"
val destinationFile = uploadsPath
    .resolve(Paths.get(fileName))
    .normalize().toAbsolutePath()

image.inputStream.use { inputStream -> Files.copy(
    inputStream, destinationFile,
    StandardCopyOption.REPLACE_EXISTING
) }

return "/images/${fileName}"
```

Výpis kódu 9: Ukázka ukládání obrázku v `ImageService`. Nejprve je vytvořena cesta k ukládání obrázku, následně je určena přípona podle typu obrázku, vygenerováno unikátní jméno a obrázek je zkopírován na disk. Metoda pak vrací cestu k nově vytvořenému obrázku.

4.4.6 Nahlašování chyb

Abych měl v průběhu vývoje snadnější a rychlejší přístup k chybám, které na backendu nastanou, rozhodl jsem se při návrhu architektury systému využít službu Sentry [16]. Ta umožňuje automatické nahlašování chyb, jejich monitorování podle verze, ale také monitorování výkonu aplikace a počtu provedených požadavků na jednotlivých REST zdrojích.

```
sentry.dsn=https://publickey@sentry.ksi.fit.cvut.cz/7
sentry.traces-sample-rate=1.0
sentry.environment=test
sentry.release=progtrain-backend@testlocal
```

Výpis kódu 10: Konfigurace pro Sentry

Integrace Sentry ve frameworku Spring [47] spočívá pouze v přidání balíčku `io.sentry:sentry-spring-boot-starter-jakarta` do závislostí projektu a následném přidání konfigurace pro Sentry do konfiguračního souboru Springu `application.properties`.

Při přidání Sentry do projektu dochází k automatickému nahlašování nezachycených chyb a monitorování jednotlivých REST zdrojů. Aby docházelo k nahlašování zachycených výjimek jsem ještě musel přidat kód pro odeslání výjimky do `catch` bloku funkce `tryCatch`.

```
protected fun <X> tryCatch (block: IRepository<T>.() -> X) =
    try { repository.block() } // ...
    catch (xcc: Exception) {
        Sentry.captureException(xcc);
        throw ResponseStatusException(HttpStatus.BAD_REQUEST)
    }
```

Výpis kódu 11: Zachycení výjimky s pomocí Sentry

4.5 Uživatelské rozhraní

Na základě případů užití z analýzy jsem v nástroji Figma [48] vytvořil prvotní návrh uživatelského rozhraní. Jeho kvalita odpovídá časovému tlaku na vytvoření návrhu i prototypu celého systému do začátku semestru.

Uživatelské rozhraní se skládá ze studentské a učitelské části. Obě rozhraní obsahují obrazovky Přihlášení, Přehled kurzů, Detail kurzu, Detail lekce a Seznam notifikací. Učitelské rozhraní navíc obsahuje Přehled uživatelů, Přehled studentů v lekci, Úpravu lekce a Úpravu modulu.

4.5.1 Výběr knihovny

Pro implementaci uživatelského rozhraní jsem vybíral mezi dostupnými knihovnami pro framework Vue.js, ve kterém jsem se v kapitole Frontend rozhodl frontend implementovat. Mezi nejčastěji používané patří Vuetify 3 [49], Quasar [50] a Bootstrap Vue [51].

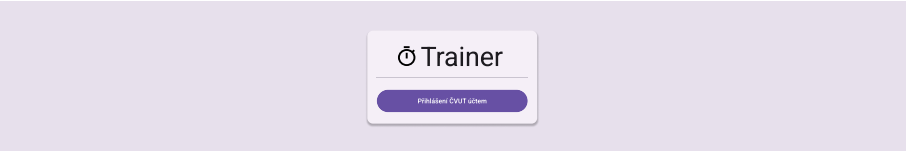
Po prohlédnutí jednotlivých knihoven jsem vyloučil Bootstrap Vue, která je kompatibilní pouze se zastaralou verzí Vue 2, byť jsem již s knihovnou Bootstrap měl předchozí zkušenosti. Vybíral jsem tedy mezi Vuetify 3 a Quasarem. Po prohlédnutí obou knihoven jsem se rozhodl pro Vuetify 3 ze dvou hlavních důvodů: komponenty ve Vuetify jsou tvořeny v souladu s pravidly Material Design [52], který se mi osobně líbí více a Vuetify 3 je oproti Quasaru jednodušší na používání a snadnější na pochopení pro nové vývojáře, což je vhodné vzhledem k povaze projektu a jeho budoucímu rozvoji.

Knihovna Vuetify 3 obsahuje komponenty pro celkové rozložení aplikace (navigační lišta, obsah, patička, karty) i jednotlivé obrazovky (tlačítka, textová pole, nebo tabulky). Ve Figmě navíc existuje knihovna s komponentami pro Vuetify 3 [53], kterou jsem použil proto, aby byl návrh co nejvěrnější.

Kompletní návrh vzniklého uživatelského rozhraní včetně souboru pro nástroj Figma, stejně jako podobu výsledného uživatelského rozhraní 1. iterace naleznete v přílohách.

4.5.2 Přihlášení

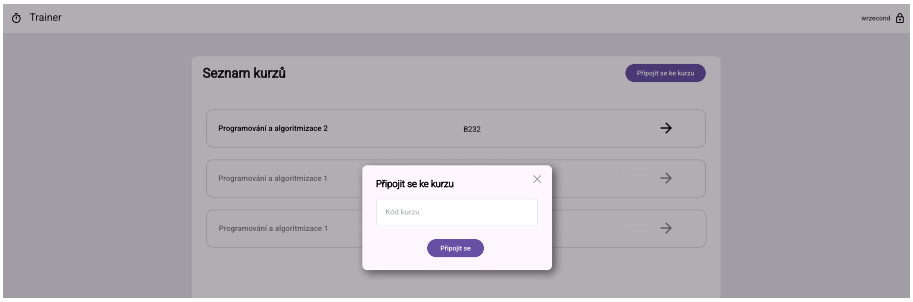
Přihlašovací obrazovka obsahuje logo a název systému a tlačítko pro přihlášení přes OAuth.



Obrázek 4.4: Přihlašovací obrazovka

4.5.3 Přehled kurzů

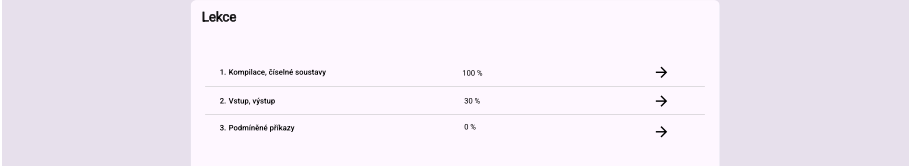
Obrazovka obsahuje seznam všech kurzů daného uživatele. U každého kurzu je uvedeno jeho jméno, semestr a je zde možnost přejít do detailu kurzu. Archivované kurzy jsou v seznamu uvedeny šedou barvou. Nad seznamem se vyskytuje tlačítko Připojit se ke kurzu, které umožní připojení studenta do nového kurzu na základě kódu, který mu poskytne učitel.



Obrázek 4.5: Seznam kurzů s aktivním dialogem pro připojení do nového kurzu

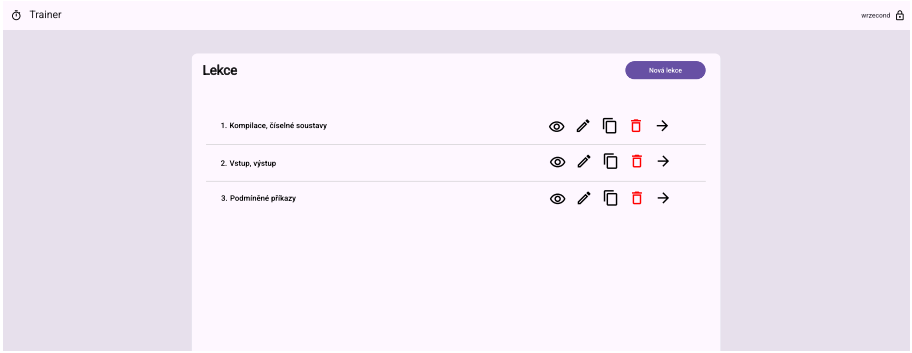
4.5.4 Detail kurzu

Detail kurzu se liší pro studenta a pro učitele. Student vidí seznam lekcí a u každé lekce kromě jejího názvu vidí svůj pokrok v dané lekci.



Obrázek 4.6: Detail kurzu (studentův pohled)

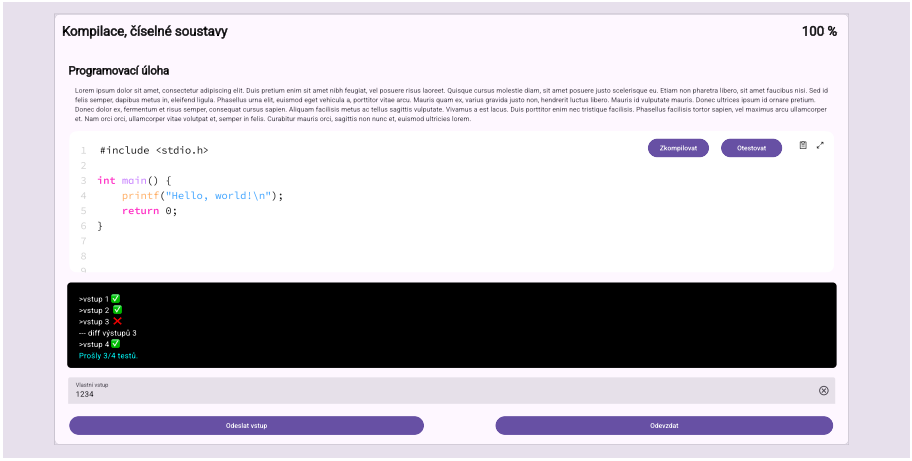
Učitel místo pokroku vidí seznam tlačítek pro úpravu, smazání, zkopírování lekce nebo přehled pokroků studentů v dané lekci. Nad seznamem lekcí vidí navíc tlačítko pro přidání nové lekce.



Obrázek 4.7: Detail kurzu (učitelský pohled)

4.5.5 Detail lekce

V detailu lekce jsou zobrazeny pod sebou všechny moduly. Každý modul se vykreslí jinak podle jeho typu – u textového modulu se zobrazí pouze text, u kódového modulu se zobrazí editor pro psaní kódu, tlačítka pro kompilaci, spuštění testů nebo odevzdání učiteli, textové pole pro vlastní vstup a „konzole“ s výstupem programu a výsledky testů, u assignment modulu zase formulář pro nahrání souboru studentského řešení.



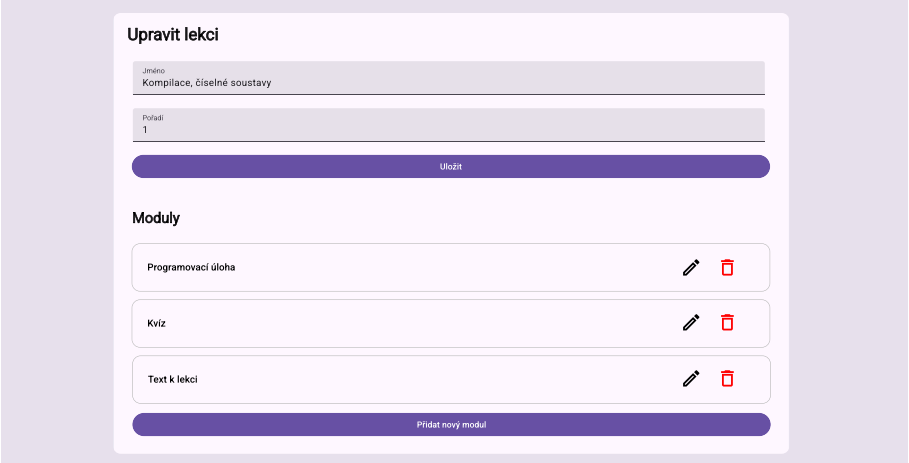
Obrázek 4.8: Detail lekce

4.5.6 Úprava modulu a lekce

Možnost upravovat lekce (a moduly) mají pouze učitelé daného kurzu. V rozhraní pro úpravu modulu je možné upravovat jeho název, pořadí v rámci lekce, zadání a v případě kódového modulu testovací kód.

Obrázek 4.9: Úprava modulu

V případě úpravy lekce je zde textové pole pro úpravu názvu, pořadí v rámci kurzu a rozhraní pro úpravu modulů. Jedná se o seznam modulů s proklikem na úpravu jednotlivých modulů nebo odstranění modulu. Pod seznamem modulů je proklik na vytvoření nového modulu v dané lekci.



Obrázek 4.10: Úprava lekce

4.6 Frontend

Pro implementaci frontendu jsem v kapitole Frontend vybral technologii Vue.js 3 [42] a programovací jazyk JavaScript [37]. Frontend používá MVVM architekturu, která je pro Vue.js typická. Na úrovni Modelu jsou jednotlivé JavaScriptové objekty. Jelikož je Javascript dynamicky typovaný jazyk, nemají objekty pevně danou strukturu, a nevytvářel jsem pro ně tedy žádné třídy. Úroveň View je tvořena HTML, které vygeneruje Vue na základě specifikované šablony v tagu `<template>` jednotlivých komponent. ViewModel je pak tvořen samotným JavaScriptovým kódem v tagu `<script setup>` u jednotlivých komponent.

Frontend se skládá ze tří částí: statických zdrojů ve složce `public/` (1), pomocných tříd pro zajištění routování, komunikace s backendem, navigace, kompilace a spouštění kódu ve složkách `plugins/`, `router/`, `service/` (2) a samotných Vue komponent ve složkách `components/` a `views/` (3). Části (2) a (3) rozeberu v následujících podkapitolách.

4.6.1 Navigace

Pro navigaci v rámci frontendu využívám Vue Router [54]. Základním pojmem v navigaci pomocí Vue Routeru je cesta (`route`). Každá cesta se skládá z adresy včetně parametrů (`path`), názvu (`name`) a komponenty, která se má vykreslit (`component`).

```
{
  path: '/lessons/:lesson',
  name: 'lesson-detail',
  component: () => import('../views/LessonDetailView.vue')
}
```

Výpis kódu 12: Definice cesty pro detail lekce

Uvnitř frontendu pak probíhá navigace mezi cestami buď přes komponentu `RouterLink` s atributem `to`, nebo přes metodu `push` na instanci Routeru, kterou lze získat funkcí `useRouter`.

Uživatel by měl také vědět, na které stránce se nachází. Toho jsem docílil pomocí proměnné `appState`, která je definovaná na úrovni celého frontendu. Tato proměnná obsahuje pole `navigation`, ve kterém je seznam jednotlivých cest a jejich názvy. V hlavičce stránky se pak zobrazuje drobečková navigace.

Obrázek 4.11: Drobečková navigace v hlavičce stránky

Jak atribut `to`, tak metoda `push` nebo pole `navigation` vyžadují cestu ve formátu obsahujícím název cesty, parametry cesty a jméno, které se má zobrazit uživateli.

Vytvořil jsem proto třídu `NavigationElement` s těmito atributy a metodou `routerPath()`, která vrátí obsah třídy v podobě potřebné pro navigaci. Komponentu pro hlavičku stránky jsem vhodně upravil, aby tuto třídu používala.

Pro každou cestu jsem vytvořil třídu, která dědí z této základní třídy `NavigationElement` a která na základě předaných entit vhodně nastaví název a parametry cesty i jméno, které zobrazit uživateli.

```
export class LessonDetail extends NavigationElement {
    constructor(lesson) {
        super(lesson.name, 'lesson-detail', {
            lesson: lesson.id
        })
    }
}
```

Výpis kódu 13: Ukázka třídy `LessonDetail`, která umožňuje navigaci do detailu lekce. Parametrem konstruktoru je objekt lekce, uživateli se v hlavičce zobrazí název lekce a do cesty routeru je vhodně předán název a parametr id lekce.

```
<router-link :to="new Nav.LessonDetail(lesson).routerPath()">
    <v-btn variant="text" icon="mdi-arrow-right"></v-btn>
</router-link>
```

Výpis kódu 14: Ukázka navigace do detailu lekce s využitím komponenty `RouterLink` a metody `routerPath`.

```
appState.value.navigation = [
    new Nav.CourseList(),
    new Nav.CourseDetail(lesson.course),
    new Nav.LessonDetail(lesson)
]
```

Výpis kódu 15: Při načtení komponenty `LessonDetail` je ve funkci `onMounted` nastavena drobečková navigace v hlavičce stránky.

4.6.2 Komunikace s backendem

Pro komunikaci s backendem je využita knihovna Axios [55]. Axios umožňuje volání HTTP metod včetně jednoduchého rozhraní pro nahrávání souborů a odesílání HTTP formulářových dat.

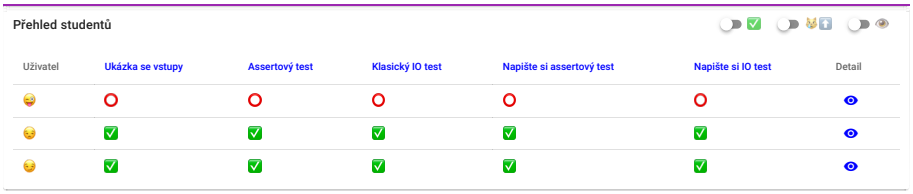
Při načtení instance axiosu nastavím URL adresu pro API (`axios.baseURL`) a hlavičku `LoginSecret` obsahující přihlašovací klíč, která je společná pro všechny požadavky. Následně definuji klienta (`apiClient`) jako objekt, který obsahuje funkce pro volání jednotlivých endpointů. Tyto funkce pak volám ve všech komponentách.

```
const apiClient = {
    // ...
    async lessonDetail(id) {
        const response = await axios.get(`/lessons/${id}`)
        return response.data
    },
    async putModuleFile(id, file) {
        const formData = new FormData()
        formData.append("file", file)
        const response = await axios.post(`/modules/${id}/file`,
        formData, {
            headers: { 'Content-Type': 'multipart/form-data' }
        })
        return response.data
    },
    async getModuleFile(id) {
        const response = await axios.get(`/modules/${id}/file`,
        { responseType: 'arraybuffer' })
        return response.data
    },
    // ...
}
export default apiClient
```

Výpis kódu 16: Ukázka nastavení klienta pro komunikaci s backendem. Funkce `lessonDetail` pouze vrátí JSON data, která získá pomocí GET požadavku. Funkce `putModuleFile` nahraje uživatelské řešení modulu jako soubor ve formuláři, `getModuleFile` zase tento soubor umožní získat (binární data).

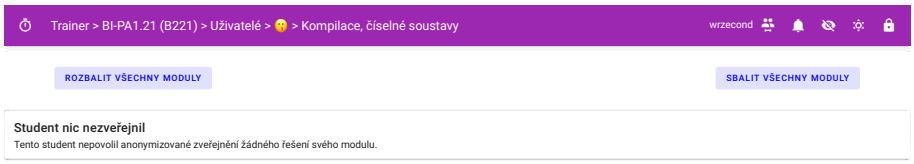
4.6.3 Anonymní mód

Jedním z dalších požadavků byla možnost promítání anonymizovaných řešení studentů učitelem. Pro tento požadavek jsem vytvořil pro učitele takzvaný anonymní mód. Po přepnutí do anonymního módu skrze tlačítko oka v navigační liště se ve všech seznamech, drobečkové navigaci a výsledcích kvízů místo jmen studentů ukazují emoji.



Obrázek 4.12: Seznam řešení studentů s aktivní anonymizací

V detailu lekce jsou pak viditelné pouze ty úlohy, u kterých student souhlasil s anonymizovaným zveřejněním, případně hlášení, že student nepovolil zveřejnění žádného řešení.



Obrázek 4.13: Detail řešení studenta s aktivní anonymizací

Anonymní mód je implementován jako příznak `anonymous` v uživatelském úložišti (`userStore`), který využívají komponenty pro studentovo jméno / seznam modulů v detailu lekce.

```
<v-card v-for="module in lesson.modules.filter((mod) =>
!userStore.anonymous || !props.user || mod.allowedShow)"
:key="module.id" class="mx-16 my-4">
  <ModuleDetail :lesson="lesson" :module="module"
  :user="props.user" :reload="reload" />
</v-card>
```

Výpis kódu 17: Anonymní mód v detailu lekce (`LessonDetail`). V seznamu modulů jsou v případě aktivního anonymního módu (`userStore.anonymous`) a pohledu na detail studentova řešení (`props.user`) vyfiltrovány ty moduly, u kterých student nesouhlasil se zveřejněním (`mod.allowedShow`).

4.6.4 Úložiště

V návrhu jsem popisoval, že mezi data, které systém ukládá, patří i informace o přihlášení a uživatelská nastavení. Tato data jsou ukládána v prohlížeči klienta ve formě cookies a toto uložení je zprostředkováno právě frontendem.

```
export const useUserStore = defineStore('user', {
    state: () => { return { user: null, darkMode: false } },
    getters: { isLoggedIn: (state) => state.user !== null },
    actions: {
        setUser(user) { this.user = user },
        toggleDarkMode() { this.darkMode = !this.darkMode },
        logout() { this.setUser(null) }
    }, persist: true
})
```

Výpis kódu 18: Ukázka úložiště pro informace o přihlášení a uživatelská nastavení. To je definováno pomocí `defineStore` s výchozími daty, getterem pro zjištění, zda je uživatel přihlášen a akcemi na přepnutí světlého / tmavého režimu a přihlášení / odhlášení uživatele. Na konci definice je nastavení perzistentního ukládání.

Ukládání řídí knihovna Pinia [56], která umožňuje definici úložiště dat pod daným identifikátorem s předem určenou strukturou, akcemi, které lze s daty provádět a především s možností data ukládat trvale (do prohlížeče klienta).

Jak informace o přihlášení, tak uživatelská nastavení (světlý / tmavý mód) jsou uložena v perzistentním úložišti `userStore`, které jsem definoval právě pomocí Pinia.

4.6.5 Přihlašování

Jak již bylo zmíněno, pro komunikaci s backendem potřebuje frontend přihlašovací klíč (`loginSecret`). Ten získá v procesu přihlášení a je uložen v uživatelském úložišti (`userStore`) v podobě cookies.

O přihlášení se stará komponenta `LoginForm`. Ta při načtení z backendu stáhne identifikátor klienta (`clientId`) a adresu pro přesměrování (`redirectUri`). Po stisknutí přihlašovacího tlačítka je pak uživatel přesměrován na FIT OAuth se získanými parametry.

Pokud přihlášení proběhlo úspěšně, bude uživatel přesměrován zpět na stránku s kódem (`code`) v URL adrese. Komponenta `LoginForm` tento kód odešle na backend a pokud je přihlášení úspěšné, získá zpět `loginSecret` s informací o uživateli, které uloží do `userStore` a přesměruje ho na seznam kurzů.

```
onMounted(async () => {
  const urlParams = new URLSearchParams(window.location.search)
  const codeParam = urlParams.get('code')

  if (!codeParam) {
    settings.value = await api.getOAuthSettings()
    loading.value = false
    return
  }

  api.login({ code: codeParam })
      .then((userLogin) => {
        let store = useUserStore()
        store.setUser(userLogin)
        router.push(new CourseList().routerPath())
      })
      .catch(() => window.location.href = '/')
})
```

Výpis kódu 19: Ukázka kódu pro přihlášení. Pokud je v URL parametr `code`, je uživatel přihlášen a přesměrován na úvodní stránku. Jinak se získají nastavení a zobrazí se přihlašovací stránka.

4.6.6 Modulární architektura

Jedním z požadavků na systém je modulární architektura. Ta je podporována jednak celosystémově využitím Vue komponent, ale také na úrovni jednotlivých typů úloh.

Systém aktuálně podporuje 4 typy modulů: textový (TEXT), zadání (ASSIGNMENT), kódový (CODE) a kvízový (QUIZ). Modulární architektura je zajištěna využitím oddělených komponent pro jednotlivé typy modulů, a to jak při zobrazení, tak při editaci.

Pro přidání nového typu modulu by tak stačilo pouze přidat tento typ jako možnost vedle stávajících v parametru `type` a přidat zobrazení komponenty pro nový typ do detailu modulu (`ModuleDetail.vue`) a úpravy modulu (`ModuleCreateEdit.vue`).

```
<v-card-item v-else-if="module.type === 'TEXT'">
  <TextModule :module="module" :teacher="props.user" />
</v-card-item>
<v-card-item v-else-if="module.type === 'ASSIGNMENT'">
  <AssignmentModule :module="module" :lesson="lesson.id"
  :teacher="props.user" />
</v-card-item>
<v-card-item v-else-if="module.type === 'CODE'">
  <CodeModule :module="module" :lesson="lesson.id"
  :teacher="props.user" />
</v-card-item>
```

Výpis kódu 20: Ukázka modulární architektury při zobrazení detailu modulu. Na základě konkrétního typu modulu je použita odpovídající komponenta.

```
<div v-else-if="type === 'CODE'">
  <CodeEditModule :moduleId="moduleId" />
</div>
<div v-else-if="type === 'TEXT' || type === 'ASSIGNMENT'">
  <!-- Nothing -->
</div>
```

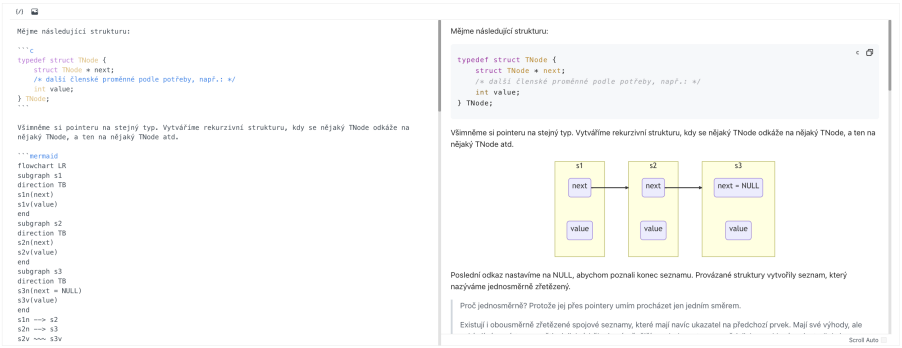
Výpis kódu 21: Ukázka modulární architektury při úpravě modulu. V případě kódového modulu je zobrazena dodatečná komponenta pro nastavení, u textového / assignment modulu žádná dodatečná nastavení nejsou.

Modulární architektura je pak také zajištěna na úrovni dat, kdy jsou pro jednotlivé moduly jejich nastavení a data ukládána ve specifických tabulkách (např. tabulka `code_module` pro kódový modul).

Následuje popis jednotlivých modulů. Kvízový modul zde není popsán, jelikož na jeho vývoji pracuje souběžně s touto prací Matej Pašek v rámci své bakalářské práce. Mnou navrhnutá modulární architektura umožňuje jednoduché začlenění tohoto kvízového modulu po boku zde popsaných modulů.

4.6.7 Textový modul

Textový modul umožňuje zobrazit uživateli text obsahující např. zadání úlohy, popis lekce nebo vysvětlení látky. Pro uvažovaný obsah textového modulu nestačí jednoduché textové pole, jelikož se v textu můžou vyskytovat různé diagramy, obrázky a ukázky kódu. Učitel by měl mít také možnost text upravovat, formátovat, a to ideálně v co nejjednodušším formátu.



Obrázek 4.14: Textový modul s ukázkou kódu a diagramem v učitelském rozhraní. Vlevo je editor, vpravo náhled, jak modul uvidí student.

Na základě požadavků jsem se pro textový modul rozhodl využít formát Markdown [57]. Ten umožňuje jednoduché formátování textu, vkládání obrázků, odkazů, kódu i diagramů. Tento formát lze následně jednoduše vyrenderovat jako HTML a vložit do webové stránky.

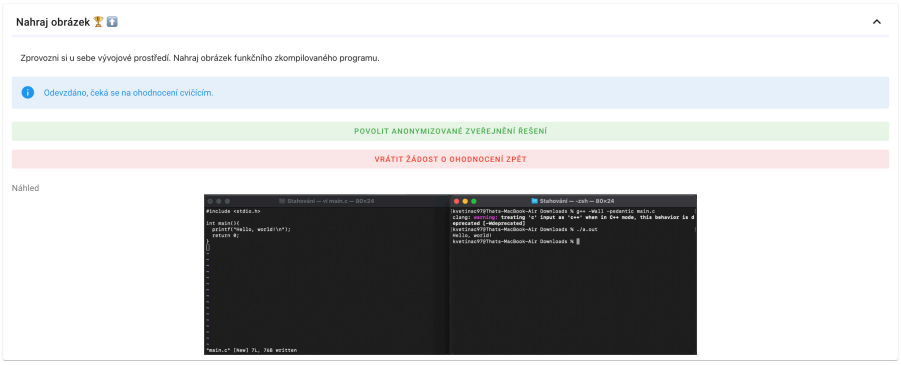
Pro Markdown jsem zvolil editor MdEditorV3 [58], který umožňuje všechny požadované vlastnosti: základní formátování, vkládání kódu, tabulek, vzorců, diagramů a nahrávání obrázků. Tento editor je navíc přímo dostupný jako dvojice komponent pro framework Vue.js: kromě komponenty editoru (`MdEditor`) i jako komponenta `MdPreview` pro vygenerování HTML z Markdownu.

```
const onUploadImg = async (images, callback) => {
  const results = await Promise.all(
    images.map((image) => moduleApi.postImage(image)))
  callback(results.map(
    (imagePath) => axios.defaults.baseURL + imagePath))
}

<MdEditor language="en-US" preview-theme="github"
  :theme="store.darkMode ? 'dark' : 'light'"
  :no-iconfont="true" :scroll-auto="false" class="px-4"
  :toolbars="['code', 'image']" onUploadImg="onUploadImg" />
```

Výpis kódu 22: Využití komponenty pro editor v textovém modulu. Pokud je do editoru nahrán obrázek, zavolá se funkce `onUploadImg`, která nahraje obrázky na backend a vrátí odkaz na nahrané obrázky.

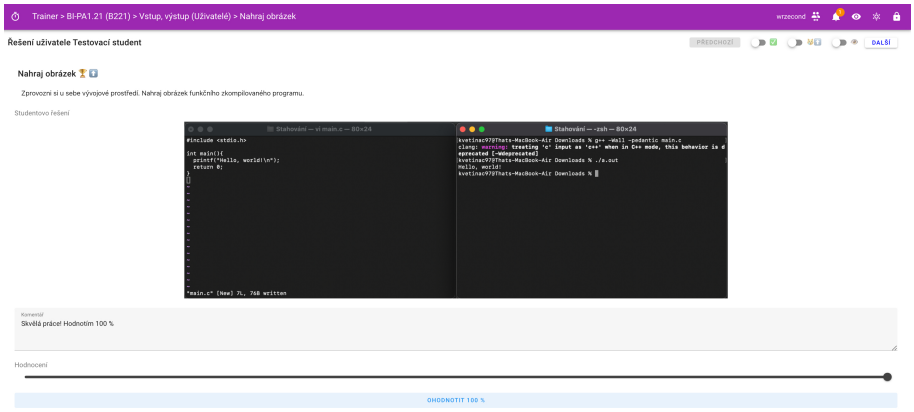
4.6.8 Assignment modul

Assignment modul slouží k zadání úloh, ve kterých učitel manuálně hodnotí obrázek odevzdaný studentem. V budoucnu by šlo tento modul rozšířit o nahrávání jakéhokoliv typu souboru, vzhledem k požadavkům předmětů PA1 a PA2 byla prozatím implementována pouze verze s nahráváním obrázků.



Obrázek 4.15: Assignment modul po odevzdání studentem

Assignment modul obsahuje zadání, formulář pro nahrání studentova obrázku a tlačítko pro odevzdání. Po odevzdání již student nemůže upravovat nahraný obrázek, pokud odevzdání nezruší. Učitel může studentův obrázek zobrazit a ohodnotit pomocí posuvníku na škále 0 až 100 %.



Obrázek 4.16: Assignment modul z pohledu učitele

Při implementaci assignment modulu byla výzvou komprese nahraných obrázků. Vzhledem k počtu studentů a paralelek byl nastaven limit 1 MB na nahraný obrázek, přičemž běžný obrázek vyfocený z mobilního telefonu, nebo některé větší snímky obrazovky tento limit jednoduše překročí.

Využil jsem proto knihovnu CompressorJS [59], která umožňuje kompresi se zadanou kvalitou a změnu velikosti v případě příliš velkého vstupního obrázku. Pokud je výsledný obrázek i tak větší než 1 MB, uživateli se zobrazí chybové hlášení.

```
new Compressor(file, {
    quality: 0.8,
    maxWidth: 1920, maxHeight: 1080, resize: "none",
    convertTypes: "image/png", convertSize: MAX_UPLOAD_SIZE,
    success: (resultImage) => { /* ... */ },
    error: (err) => { error.value = err }
})
```

Výpis kódu 23: Komprese obrázku v assignment modulu

4.6.9 WASM kompilace a běhové prostředí

Jak již bylo zmíněno v návrhu, pro vyhodnocování kódu jsem vybral variantu lokálního vyhodnocení s využitím technologie WebAssembly [43]. Jedná se o binární instrukční formát, do kterého lze zkompilovat kód různých programovacích jazyků tak, aby byl následně spouštěn v prohlížeči.

V předmětech PA1 a PA2 se používají dva programovací jazyky, C a C++. Oba dva z těchto jazyků lze zkompilovat do WASM tak, aby byly následně spouštěny v prohlížeči.

Oficiální doporučený způsob, jak C/C++ kód do WASM kompilovat, je s pomocí kompilátoru Emscripten [60]. Ten má ale zásadní nevýhodu: jedná se o program, který běží pouze na Windows, macOS nebo Linuxu, což není vyhovující. Pro potřeby systému by byly třeba kompilátor a linker, které by také běžely v prohlížeči.

Ty mi poskytl vedoucí práce, který do WASM zkompiloval kompilátor (Clang) i linker. Dodal mi také pro kompilátor potřebnou knihovnu libc, upravenou speciálně pro WASM [61].

Poslední věc potřebná pro lokální vyhodnocování kódu je WASI (WebAssembly System Interface) [62]. Jedná se o knihovnu, která umožňuje spouštět programy ve WASM, pracovat s jejich vstupem a výstupem a kontrolovat jejich návratovou hodnotu. Existuje více různých implementací WASI, od vedoucího mi byla doporučena `@runno/wasi` [63].

Na frontendu jsem kompilátor (`clang.wasm`), linker (`wasm-ld.wasm`) a archiv s knihovnami (`sysroot.tar`) umístil do složky `public/`, aby je mohl prohlížeč klienta stáhnout.

Veškeré funkce pro práci s WASI jsem umístil do `plugins/code.js`. Jedná se o funkce `Code.compile` pro kompilaci C/C++ kódu do objektových souborů, `Code.link` pro jejich slinkování do WASM programu a `Code.run` pro spuštění WASM programu s kódem.

Aby kompilace a linkování fungovalo, musel jsem při kompilaci includovat přiložené knihovny extrahované z archivu `sysroot` ve funkci `prepareFS` pomocí flagu `-isystem`. Při linkování se pak musí přilinkovat C knihovna (`-lc`) a runtime knihovny pro WASM.

Při prvním pokusu o kompilaci jsem přišel na dva problémy s knihovní implementací WASI. Prvním bylo nesprávné použití návratových kódů při práci se souborovým systémem, což při kompilaci vedlo k hlášce „Capabilities insufficient“. Druhým pak bylo neefektivní asynchronní spouštění zkompilovaného a slinkovaného studentova programu ve WASM.

Oba tyto problémy jsem vyřešil úpravou v knihovně WASI. První problém jsem vyřešil náhradou kódu `ENOTCAPABLE` za `ENOENT` v případě nenalezení souboru ve `wasi.js`. Druhý problém jsem vyřešil změnou způsobu, jak se do JavaScript workeru předává studentův program ke spuštění.

V původní verzi se předávala URL souboru, který se následně pomocí JavaScript fetch API [64] stáhnul. U malých souborů tak docházelo k zbytečnému zdržení při stahování přes prohlížeč. Místo URL souboru jsem do workeru předal přímo studentův program, což vedlo k mnohonásobnému zrychlení. Následně jsem vytvořil patch soubor pomocí `npx patch-package @runno/wasi`, aby byly změny aplikovány i pro nové instalace.

```
const clangCommonArgs = ['-isysroot', '/', '-isystem',
  '/include/c++/v1', '-isystem', '/include', '-isystem',
  '/lib/clang/16/include', '-fno-builtin', '-D__TRAINER__']

let fs = await prepareFS(code, tester)
let [stdout, stderr] = ["", ""]
let context = new WASIContext({
    args: [ 'clang++', ...clangCommonArgs, '-c', TESTER_FILE,
    '-o', OUTPUT_OBJ ],
    env: {},
    fs: fs,
    stdout: (s) => { stdout += s },
    stderr: (s) => { stderr += s },
})

let wasi = new WASI(context)
wasi.init(await WebAssembly.instantiate(await fetchClang(), {
    wasi_snapshot_preview1: wasi.getImports("preview1"),
    wasi_unstable: wasi.getImports("unstable")
}))
let result = await wasi.start()
return [stdout, stderr, result.exitCode === 0 ? result.fs :
null]
```

Výpis kódu 24: Ukázka kódu pro kompilaci programu do WASM. Nejprve je ve funkci `prepareFS` stažen a rozbalen archiv s knihovnami (`sysroot.2.tar`), do kterého jsou přidány soubory studentova (`code`) a testovacího (`tester`) programu. Následně je vytvořen kontext pro WASI, který ukládá výstup kompilátoru do proměnných `stdout` a `stderr`. S tímto kontextem je vytvořena instance WASI, která je inicializována a spuštěna s kompilátorem. Výsledek běhu je v případě úspěchu vrácen volající funkci.

Ve WASI knihovně i při způsobu kompilace byly v průběhu semestru provedeny změny pro kontrolu práce s pamětí. Pro jejich nasazení jsem nejprve nahradil archiv s knihovnami novou verzí, kterou mi dodal vedoucí práce. Tyto změny spočívaly v přidání exportu pro počet volání funkce `malloc` a `free` do knihovny `libc`, což umožnilo kontrolu uvolňování paměti ve studentských programech. Zároveň obsahovaly opravu chyby v knihovně `libcrt`, která neuvolňovala paměť alokovanou pro argumenty funkce `main`.

Následně jsem přidal do WASI knihovny kód, který vyplní při startu programu zásobník byty `0xdeadbeef`. Při standardním chování je totiž ve WASM paměť programu vynulovaná a nedochází k problémům s neinicializovanými proměnnými. Vyplnění zásobníku tak vede k nesprávnému chování studentova programu a selhání testů.

```
let context = new WASIContext({
    args: ['wasm-ld', '-z', `stack-size=${stackSize}`,
    `-L${libdir}`, crt1, OUTPUT_OBJ, '-lc', '-lc++', '-lc++abi',
    '-L/lib/clang/16/lib/wasi', '-lclang_rt.builtins-wasm32',
    '-o', WASM_FILE], env: {}, fs: fs,
    stdout: (s) => { stdout += s },
    stderr: (s) => { stderr += s },
})
// ...
return [stdout, stderr, result.exitCode === 0 ?
result.fs[WASM_FILE].content : null]
```

Výpis kódu 25: Ukázka linkování ve WASM. Linkování probíhá v souborovém systému získaném z úspěšné kompilace. Výstupní soubor je slinkován do výsledného programu, který je v případě úspěchu vrácen v binární podobě.

```
let context = new WASIContext({
    args: ['./a.out', parameter], env: {}, fs: {},
    stdout: (s) => { appendOutput(s) },
    stderr: (s) => { appendOutput(s) },
})

let workerHost = new WASIWorkerHost(wasmFileContent, context);
workerHost.start()
    .then((result) => { finish(result) })
    .catch((err) => { finish() })
return workerHost
```

Výpis kódu 26: Asynchronní spouštění WASM programu v Javascript workeru. Oproti kompilaci, která běží bez použití workeru, je zde navrácen objekt `WASIWorkerHost`, který slouží k zasílání vstupu a ukončení programu. Výsledek běhu programu a výstup jsou předány do callback funkcí `finish` a `appendOutput`.

4.6.10 Kódový modul

Lokální vyhodnocování kódu je využito v kódovém modulu. Existují různé způsoby, jak program spouštět a vyhodnocovat. Po diskuzi s učiteli předmětu a analýze požadavků jsem identifikoval 5 způsobů spouštění a vyhodnocování odpovídající 5 typům kódového modulu, které jsem následně implementoval.

První způsob je **SHOWCASE** – Ukázka kódu. Učitel nahraje kód a ukázkové vstupy. Studenti následně kód vidí (nemůžou ho upravovat) a můžou ho spustit s ukázkovými nebo vlastními vstupy.

Druhým způsobem je **TEST_IO** – Klasický I/O test. Učitel nahraje vstupy a odpovídající referenční výstupy. Cílem studentů je napsat program, který má pro daný testovací vstup stejný výstup jako je ten referenční.

Třetí je **TEST_ASSERT** – Assertový test. Učitel nahraje testovací program, studenti napíšou kód, který je otestován testovacím programem.

Čtvrtý a pátý způsob jsou podobné. Učitel napíše sadu funkcí, z nichž některé fungují špatně a některé správně. Student pak musí napsat testovací vstupy a výstupy (4. typ **WRITE_IO**) nebo testovací program (5. typ **WRITE_ASSERT**), které korektně odhalí správné a nesprávné funkce.

Mezi jednotlivými typy kódového modulu jsem identifikoval, že se zde nachází kód viditelný studentem (`codeShown`) a skrytý kód, typicky testovací program (`codeHidden`). Následně učitel nebo student podle typu modulu můžou nahrát testovací vstupy. V posledních dvou způsobech pak ještě učitel definuje, které funkce jsou správné a které špatné.

Rozhodl jsem se tedy udělat jednotné rozhraní pro úpravu modulu, které dynamicky podle vybraného typu modulu zobrazuje / skrývá editaci jednotlivých kódů / dat.

Rozhraní pro zobrazení kódového modulu jsem už kvůli rozdílnému vzhledu (seznam s výběrem testovacích dat, pole pro nahrání testovacích dat, konzole pro zadání vstupu) jako jednotné nenavrhnul.

```
const HIDDEN_LABELS = {'TEST_ASSERT': 'Asserty', 'WRITE_ASSERT':
'Testovaný kód', 'WRITE_IO': 'Testovaný kód'}
const SHOWN_LABELS = {'SHOWCASE': 'Kód', 'TEST_ASSERT': 'Výchozí
kód', 'TEST_IO': 'Výchozí kód', 'WRITE_ASSERT': 'Výchozí
asserty'}
const FILE_LABELS = {
  'SHOWCASE': 'Ukázkové vstupy (.in) v archivu .tar',
  'TEST_IO': 'Testovací vstupy (.in, .ref) v archivu .tar',
  'WRITE_IO': 'Ukázkové vstupy (.in, .ref) v archivu .tar'
}
const WRONG_LABELS = {'WRITE_IO': 'Špatné programy',
'WRITE_ASSERT': 'Špatné programy'}
const RIGHT_LABELS = {'TEST_ASSERT': 'Možné sady testů',
'WRITE_IO': 'Správné programy', 'WRITE_ASSERT': 'Správné
programy'}
```

Výpis kódu 27: Definice popisků pro úpravu jednotlivých polí podle typu modulu. Pokud popisek není definován, pole se neukazuje.

Pro každý typ kódového modulu je navržena samostatná komponenta, která vychází ze společné šablony `CodeModuleBase`, ve které je definován kód pro kompilaci, linkování nebo nahrávání výsledků společný pro všechny typy.

Způsob spouštění se liší podle typu kódového modulu. Například u SHOWCASE se spouští program jednou s konkrétním vstupem. U TEST_IO nebo WRITE_IO se spouští program pro každý test v každé sadě testů zvlášť, u TEST_ASSERT nebo WRITE_ASSERT se program spustí 1x pro každou sadu.

```
let total = 0, passed = 0
for (const param of params.value) {
  const checkResult = (result) => {
    passed += checkTestOutput(param, result)
    total += 1
    if (total === params.value.length)
      submitResults(passed, total)
  }

  const workerHost = await Code.run(runtime.value.wasmFile, ()
  => {}, checkResult, [param])
  runtime.value.workerHosts.push(workerHost)
}
```

Výpis kódu 28: Ukázka spouštění kódu pro typ TEST_ASSERT. Pro každou testovací sadu je program spuštěn s daným parametrem. Po doběhnutí programu se zkontroluje návratový kód programu a v případě úspěchu test prošel danou testovací sadou. Po doběhnutí testů jsou výsledky odeslány na backend.

```
const workerHost = await Code.run(runtime.value.wasmFile,
appendOutput, (wasmResult) => {
  if (checkTestOutput(testName, wasmResult, outputValue.value,
  tests.value[testSet][testName], print))
    result.success += 1
  else return callback()

  if (queue.length)
    runQueue(queue, testSet, result, appendOutput, outputValue,
    callback)
  else callback()
})
runtime.value.workerHosts.push(workerHost)
workerHost.pushStdin(input).then(() => workerHost.pushEOF())
```

Výpis kódu 29: Ukázka spouštění kódu pro typ TEST_IO. Nejprve je každý vstup v dané testovací sadě vložen do fronty. Vstupy jsou pak spouštěny jeden po druhém, vstup je do nich předán metodou `pushStdin` na `WASIWorkerHost`. Při neúspěchu je běh testovací sady přerušen. Po doběhnutí sady je zavolána funkce `callback`, která po doběhnutí všech sad odešle výsledky na backend.

4.7 Nasazení

Aby mohl být systém využíván studenty, musel jsem ho jako celek (databázi, backend i frontend) nasadit na školní infrastrukturu. K nasazení jsem využil školní platformu Cloud FIT, na které vedoucí práce zařídil virtuální server s adresami `https://trainer.ksi.fit.cvut.cz` pro ostrou verzi a `https://dev.trainer.ksi.fit.cvut.cz` pro vývoj.

4.7.1 Kontejnerizace

Aby byl celý proces vývoje, testování i nasazení co nejsnadnější, rozhodl jsem se použít nástroj Docker [65], který umožňuje vývoj jednotlivých částí v kontejnerech, které jsou nezávislé na operačním systému a stroji, na kterém běží. Tyto části jsou následně orchestrované s pomocí nástroje Docker Compose.

Docker Compose jsem nastavil s pomocí souboru `docker-compose.yml`, který obsahuje konfiguraci jednotlivých částí a způsob, kterým je propojit. Pro databázi jsem použil již existující obraz `mysql`, kterému jsem nastavil port 3306, proměnné prostředí pro nastavení výchozí databáze a využil jsem inicializační skripty pro nastavení výchozích dat v databázi.

Obraz pro backend i frontend je sestaven s pomocí souborů `Dockerfile` (zvlášť pro backend i frontend) ze základního obrazu pro Javu / Node.js pomocí příkazů, které zkompilují kód (build-stage) a následně ho spustí (production-stage).

```
backend:
  build: ./backend
  depends_on:
    db:
      condition: service_healthy
  env_file:
    - .test.env
  restart: always
  volumes:
    - ./uploads:/app/uploads
```

Výpis kódu 30: Ukázka souboru pro konfiguraci Docker Compose, konkrétně části pro nastavení backendu. Obraz se sestaví ze složky `backend` a bude spuštěn až po spuštění databáze. Konfigurace bude přečtena ze souboru `test.env` a do kontejneru bude napojena složka `uploads`.

Jelikož je výsledkem kompilace frontendu pouze složka s HTML, CSS a JS soubory, kterou nelze spustit, využil jsem na spuštění webového serveru pro frontend Nginx [27]. Nginx funguje jako webový server, ale také jako proxy, která umožňuje přístup k backendu na adrese `/api`.

Použití proxy pro přístup k backendu bylo nutné ze dvou důvodů. Prvním je, že virtuální server samotný je za fakultní proxy, a veškerá komunikace je nutně přesměrována na port 80. Rozlišení frontendu a backendu jiným portem tedy nepřipadá v úvahu.

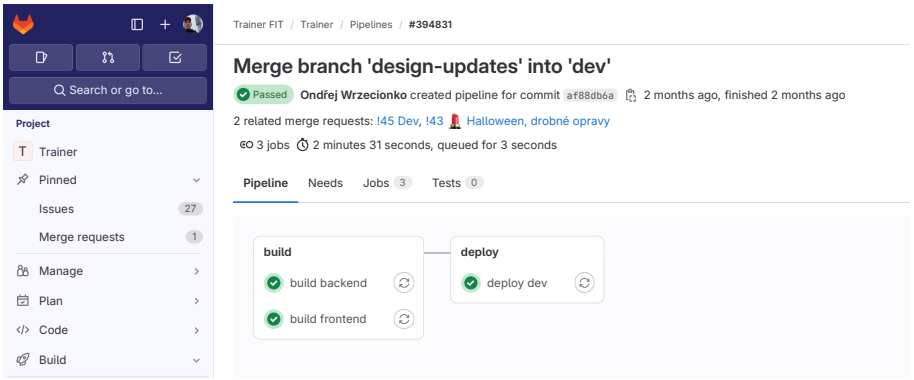
Druhým důvodem je WASM, které vyžaduje takzvaný „Cross Origin Isolation“ [66]. To znemožňuje využití zdrojů (stylů, skriptů a obrázků) z jiných originů. Jelikož backend poskytuje obrázky, musí k nim být z prohlížeče přistupováno na stejném originu, a tedy i na stejném portu.

```
server {
    listen       80;
    server_name  localhost;
    location /api/ {
        proxy_pass  http://backend:8080/;
    }
    location / {
      root   /app;
      index  index.html;
      add_header Cross-Origin-Embedder-Policy require-corp;
      add_header Cross-Origin-Opener-Policy same-origin;
      try_files $uri $uri/ /index.html;
    }
}
```

Výpis kódu 31: Ukázka souboru pro konfiguraci Nginx proxy poskytující frontend a přesměrování na backend (`/api`).

4.7.2 CI/CD

Vývoj systému probíhá na školním Gitlabu, který umožňuje také Continuos Integration a Continuos Deployment, které lze nastavit v konfiguračním souboru `.gitlab-ci.yml`. Vedoucí práce nastavil tento skript tak, že po každém commitu do větve `dev` nebo při vytvoření tagu začínající na dev proběhne nasazení do kontejnerů na testovací server. Při vytvoření tagu začínající na `Bxxx` (např. B231-01) proběhne nasazení do kontejnerů na produkci. Toto nastavení velmi usnadnilo nasazování nových verzí, které tak probíhalo i ve spojení s využitím kontejnerů automaticky bez nutnosti přenahrávat soubory na backend / frontend.



Obrázek 4.17: Ukázka CD po provedení merge do větve dev

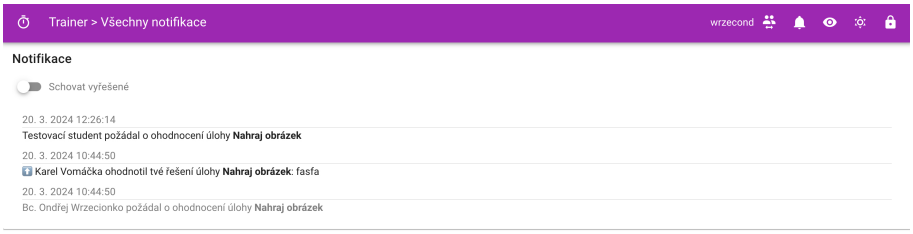
4.8 Testování

Vzhledem k časovému tlaku jsem v první iteraci neprováděl jednotkové ani integrační testy. Probíhalo tedy především uživatelské testování, a to „za pochodu“ během hodin PA1, které jsme společně s vedoucím práce měli každé úterý od 7:30 do 9:00 a od 9:15 do 10:45. Kromě našich dvou paralelek systém ještě využívala další dvojice cvičících.

4.8.1 Komunikace se studenty

Jak se studenty, tak se cvičícími, probíhala v průběhu celého zimního semestru intenzivní komunikace s cílem ověřit, zda prototyp obsahuje všechny požadované vlastnosti. Mnoho důležitých vlastností systém ještě v průběhu prvních týdnů semestru neměl. Ve druhém týdnu semestru například proběhla migrace dat kódového modulu (v původní verzi totiž neexistovala entita pro kódový modul `CodeModule` a veškerá data kódového modulu byla uložena jako JSON v atributu `data` u entity `Module`).

Ve 3. týdnu byly do systému přidány notifikace na ohodnocení úlohy nebo odpověď na žádost o pomoc. Před přidáním notifikací museli studenti i učitelé pravidelně kontrolovat stránky jednotlivých lekcí, což bylo neúnosné.



Obrázek 4.18: Seznam notifikací

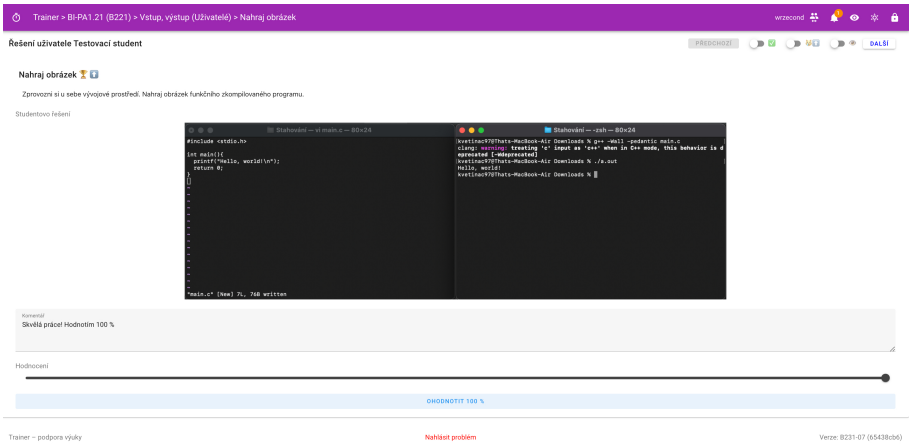
Během 4. týdne bylo přidáno zobrazení pro přehled všech studentských řešení konkrétního modulu v lekci. Toto zobrazení značně usnadnilo průběh hodiny, kdy učitel mohl zobrazovat jednotlivá anonymně zveřejněná řešení studentů za sebou bez navigace zpět na přehled studentů.

Zároveň individuální zobrazení umožnilo jednodušší odpovídání učitelů studentům na žádosti po prokliknutí z notifikace (učitel vidí pouze jeden modul, předtím musel modul najít na stránce). Kromě tohoto zobrazení byl přidán také filtr, který umožnil jak z individuálního procházení, tak z tabulky v přehledu studentů vyfiltrovat úspěšná řešení / žádosti / zveřejněná řešení.

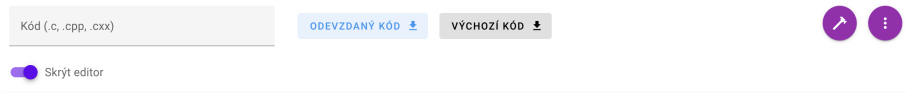
V 6. týdnu jsem do systému na požadavek studentů přidal možnost nahrát kód ze souboru s možností schovat editor. Někteří studenti totiž kód ze systému vždy stáhli do vývojového prostředí a následně ho překopírovali do editoru. Přidání pole pro nahrání souboru jim tak zjednodušilo práci a navíc zpřehlednilo stránku detailu lekce.

Během poloviny semestru se prototyp ustálil a probíhaly pouze drobné opravy. Posledními vylepšeními, které jsem implementoval, byl halloweenský a vánoční režim.

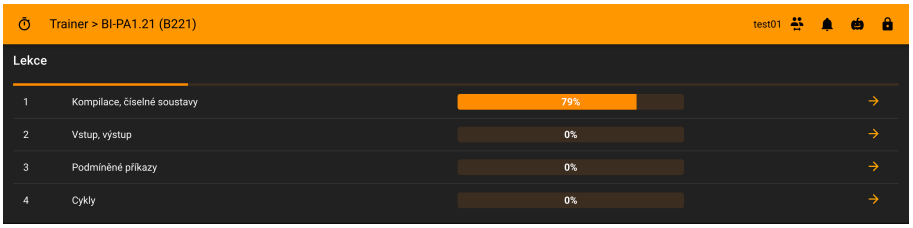
Ty fungují tak, že v konkrétní časovou dobu (31. října pro halloween, celý prosinec pro vánoční režim) se všem uživatelům změní barva navigační lišty, v přehledu studentů se místo emoji obličejů ukazují tématické emoji a v případě vánočního režimu se na stránce objeví animace sněhu.



Obrázek 4.19: Zobrazení studentského řešení konkrétního modulu v lekci



Obrázek 4.20: Pole pro nahrání existujícího souboru s kódem



Obrázek 4.21: Halloweenský režim

4.8.2 Studentský dotazník

Jelikož výuka probíhala v celkem 5 paralelkách, z nichž jsem učil pouze 2, rozhodl jsem se v polovině semestru vytvořit pro studenty na platformě Google Forms [7] dotazník, ve kterém jsem se ptal na jejich názor na Trainer, co je na systému nejvíce trápí / co se jim líbí nebo co jim chybí. Na dotazník zodpovědělo 37 studentů (z 240) rovnoměrně rozdělených mezi jednotlivé paralelky.

V dotazníku systém obdržel průměrné hodnocení 8.56 (1 je nejhorší, 10 nejlepší). Mezi negativními vlastnostmi systému převažovaly výtky na uživatelské rozhraní a chybějící možnost zobrazení referenčního řešení. Z pozitivních vlastností se nejčastěji objevovala přehlednost, usnadnění komunikace s učiteli a možnost pracovat na úlohách vlastním tempem. Přínos systému v lepším pochopení látky předmětu PA1 studenti hodnotili známkou 8.13 z 10.

4.8.3 Zjištěné problémy

Z hlediska funkčnosti jsem v systému neodhalil žádné závažné chyby, které by znemožňovaly jeho použití, zato jsem ale přišel na hodně chyb, které značně znepříjemnily uživatelskou zkušenost. Jako jedna z největších slabin se v průběhu testování ukázalo **uživatelské rozhraní**, a to jak pro studenty, tak pro učitele.

Student se musel při každém otevření systému prokliknout do Detailu kurzu a najít v seznamu lekcí aktuální lekci. To by šlo zjednodušit zobrazením aktuálně probíhajících lekcí, nebo alespoň rozčleněním lekcí do týdnů s možností rozbalit a skrýt dané týdny. Největším problémem byla obrazovka Detailu lekce. Například lekce pro 4. cvičení „Cykly“ obsahovala 20 modulů, ve kterých nebyla žádná možnost orientace, a student tak musel při každém otevření lekce narolovat konkrétní modul.

Pro učitele byla největším problémem chybějící možnost přepínání mezi jednotlivými pohledy, především mezi úpravou lekce / modulu a detailem lekce. Při vytváření a testování úloh tak bylo jediným řešením mít dva okna prohlížeče, jedno pro úpravu, a jedno pro testování. Zároveň zvláště při úpravě kódového modulu byla stránka pro úpravu velmi dlouhá (obsahovala všechna editační pole pod sebou) a při uložení úpravy se celá stránka načetla znovu, čímž musel učitel znova narolovat kód, který právě upravoval.

Kromě uživatelského rozhraní v systému spíše chyběly určité funkcionality, na kterých jsme byli s vedoucím domluveni už od začátku, že budou implementovány v rámci druhé iterace. Jednalo se o podporu jazyka C++ a vícesouborové kódové moduly (pro předmět PA2), nový způsob testování kódového modulu, překlad systému do angličtiny a již zmiňované členění lekcí do týdnů.

4.8.4 Uživatelské rozhraní

Od poloviny zimního semestru jsme v předmětu Návrh uživatelského rozhraní v rámci čtyřčlenného týmu (já, Hoang Nam Tran, Alois Kouba, Jorge Zuñiga) analyzovali stávající uživatelské rozhraní společně s výstupy ze studentského dotazníku. Vzhledem k velikosti systému jsme uvažovali **pouze** rozhraní pro **studenty**.

Výsledkem společné práce bylo vytvoření nového návrhu uživatelského rozhraní pro studenty v nástroji Figma [48], klikatelného prototypu pro uživatelské testování bez napojení na reálný backend, jeho usability testování osmi bakalářskými studenty a rozčlenění odhalených problémů pomocí 10 Nielsenových heuristik [67]. Figma soubor s vzniklým návrhem, použité testovací scénáře a heuristická analýza jsou obsahem příloh.

Vzniklý návrh uživatelského rozhraní jsem následně s drobnými změnami sám implementoval v rámci druhé iterace.

4.9 Zhodnocení iterace

V první iteraci jsem naprogramoval funkční prototyp systému, který jsem odprezentoval vyučujícím PA1 a uvedl do ostrého provozu v 5 paralelkách předmětu PA1 čítajících přibližně 240 studentů. Během semestru jsem průběžně reagoval na požadavky studentů i učitelů a doimplementoval jsem funkčnosti jako notifikace, pole pro nahrání existujícího souboru s kódem a speciální halloweenský a vánoční režim.

V závěru semestru jsem pak s týmem v předmětu NI-NUR analyzoval problémy stávajícího uživatelského rozhraní, navrhl nové rozhraní, otestoval ho a zhodnotil možná budoucí vylepšení.

První iterace svůj cíl z mého pohledu úspěšně splnila, což se projevilo i na pozitivních ohlasech z řad studentů a vyučujících. Systém značně studentům usnadnil pochopení látky předmětu PA1, konzultaci s učiteli, učitelům zase zjednodušil přípravu cvičení a umožnil jim lepší poskytnutí zpětné vazby.

Kapitola 5

Druhá iterace

V této kapitole zaktualizuji návrh uživatelského rozhraní, databáze a API na základě zpětné vazby z první iterace. Taktéž provedu návrh a implementaci druhé iterace backendu a frontendu, které řádně otestuji a opravím odhalené problémy. Na závěr vzniklý systém zdokumentuji.

5.1 Popis

Druhá iterace probíhala v průběhu zkouškového období zimního semestru a výukové části letního semestru (leden až duben 2024). Cílem iterace bylo především vylepšení uživatelského rozhraní prototypu z první iterace, přidání podpory C++ a možnost vícesouborových úloh v kódovém modulu (B232-01). Druhým cílem pak bylo systém řádně otestovat a zdokumentovat pro budoucí vývojáře (B232-02).

Obrázek 5.1: Časový plán druhé iterace

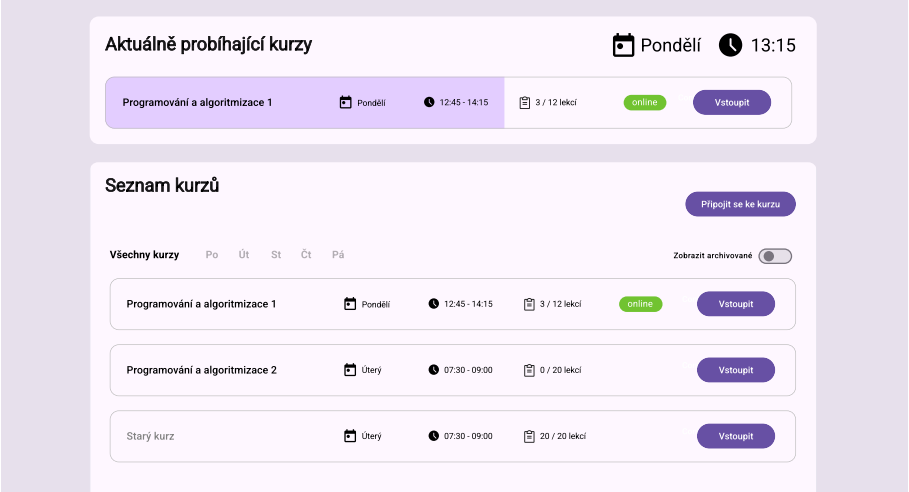
5.2 Uživatelské rozhraní

Jak jsem již zmínil v závěru předchozí kapitoly, uživatelské rozhraní bylo jedním z největších problémů první iterace. V rámci předmětu NI-NUR jsme proto navrhli nové uživatelské rozhraní pro studenty a vytvořili klikatelný prototyp bez napojení na backend.

V průběhu zkouškového období zimního semestru jsem vzniklé rozhraní implementoval, napojil na stávající backend a nasadil. Zároveň jsem na základě zpětné vazby od učitelů týkající se učitelského rozhraní provedl redesign uživatelského rozhraní pro učitele, které jsem také implementoval do systému.

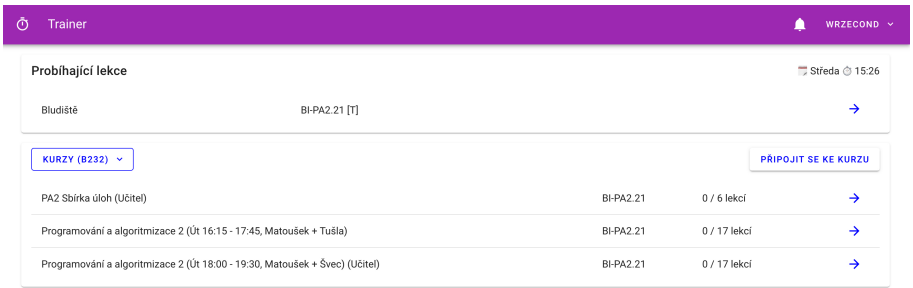
5.2.1 Přehled kurzů

Na stránce přehledu kurzů přibyl nahoře seznam s aktuálně probíhajícími kurzy. Tento seznam má za úkol uživatelům usnadnit přístup do aktuálně probíhajících kurzů. V průběhu usability testování se ukázalo, že spíše než aktuálně probíhající kurzy je pro studenty důležité vidět aktuálně probíhající **lekce**, proto byly kurzy v seznamu nahrazeny lekcemi, návrh ale zůstal stejný.



Obrázek 5.2: Původní návrh seznamu kurzů

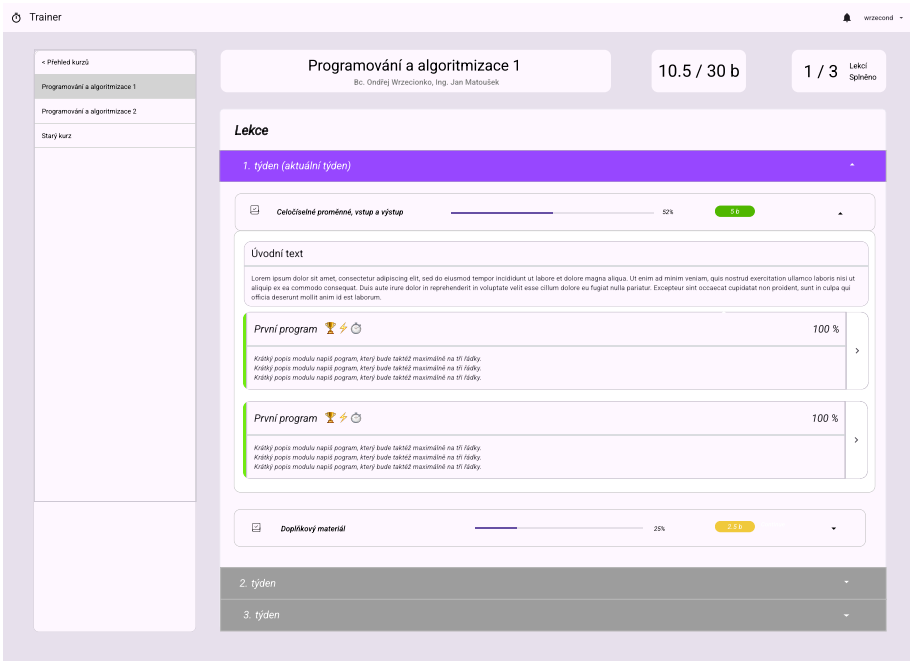
V seznamu kurzů se již nezobrazují všechny kurzy, ale pouze ty aktivní (v tomto semestru). Archivní kurzy jsou dostupné pomocí přepínače „Zobrazit archivované“. Během testování měli všichni uživatelé problém najít tento přepínač, proto byl nahrazen rozbalovací lištou s výběrem semestru (Nielsen 1: Viditelnost stavu systému [67]), která nahradila nadpis „Seznam kurzů“.



Obrázek 5.3: Seznam kurzů a přehled aktuálně probíhajících lekcí

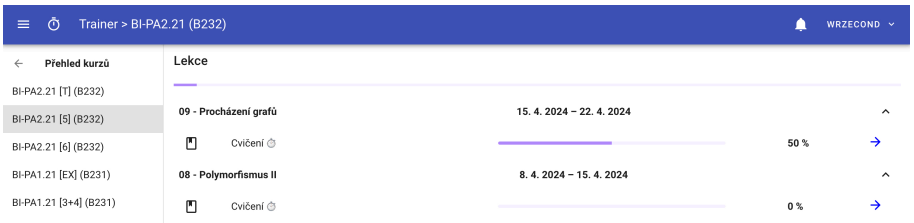
5.2.2 Detail kurzu

Obrazovka pro detail kurzu byla značně rozšířena. Byla přidána levá navigace pro snadné přepínání mezi kurzy, informační panel s názvem kurzu a počtem splněných lekcí a seznam lekcí byl rozšířen na seznam týdnů. Jednotlivé týdny jsou rozbalovací a každý z nich obsahuje seznam lekcí. Lekce je také rozbalovací a obsahuje pro každý modul úvodní text s informacemi.



Obrázek 5.4: Návrh detailu kurzu s týdny

Design seznamu týdnů, lekcí a modulů byl pro testující značně zmatený a nepřehledný (Nielsen 8: Minimalistický design [67]), proto jsem se vrátil k původnímu jednoduchému designu, do kterého byla akorát přidána jedna úroveň týdnů. Detaily jednotlivých modulů se uživatelům zobrazí až po otevření lekce. Horní panel s názvem kurzu jsem pro zjednodušení vzhledu systému také odstranil.

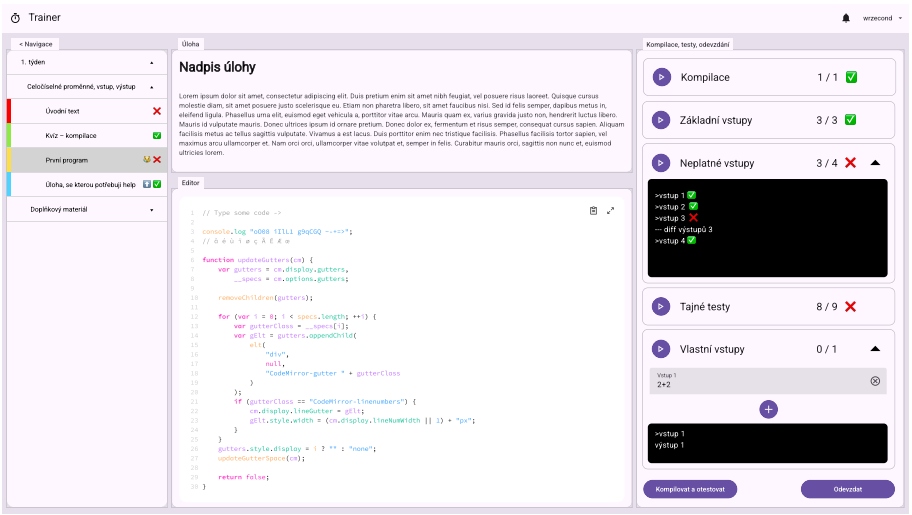


Obrázek 5.5: Zjednodušený detail kurzu

5.2.3 Detail lekce / modulu

Obrazovka detailu lekce byla kompletně přepracována, a to především z důvodu, že design 1. iterace nebyl udržitelný právě v případech, kdy měla lekce velké množství modulů.

V druhé iteraci jsem tedy obrazovku upravil tak, aby zobrazovala vždy pouze jeden modul nebo popis lekce. Zároveň jsem přidal levý navigační panel pro orientaci mezi lekcemi a moduly v daném týdnu s přehledem stavu vyplnění jednotlivých modulů pomocí bočního pruhu a pro kódový modul zároveň pravý panel s výsledky jednotlivých testů a možností zkompilovat / odevzdat kód nebo požádat o pomoc. Uprostřed zůstalo zadání a editor kódu.



Obrázek 5.6: Detail modulu v lekci (návrh)

Toto rozhraní se v průběhu usability testování osvědčilo, proto jsem ho příliš neupravoval. Kvůli podpoře vícesouborových modulů a většímu prostoru jsem akorát sloučil záložku zadání a jednotlivých editorů do jednoho okna s možností přepínání. Pod toto okno jsem umístil panel s možností nahrát kód, stáhnout výchozí / odevzdaný kód a povolit anonymizované zveřejnění.

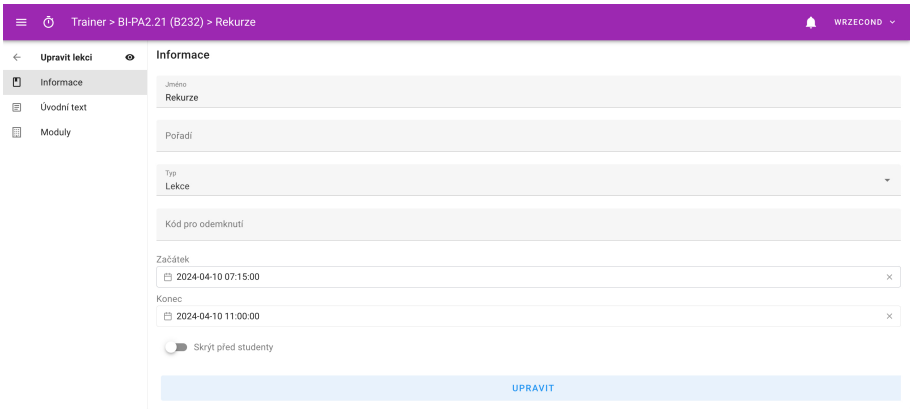
5.2.4 Úprava lekce

Hlavní rozdíl v uživatelském rozhraní pro úpravu lekce a modulů spočíval v rozdělení obsahu na jednotlivé záložky, mezi kterými se lze navigovat pomocí postranní navigace. Toto umožňuje větší přehlednost a možnost úpravy souborů / testů, které byly nově přidány do kódového modulu.

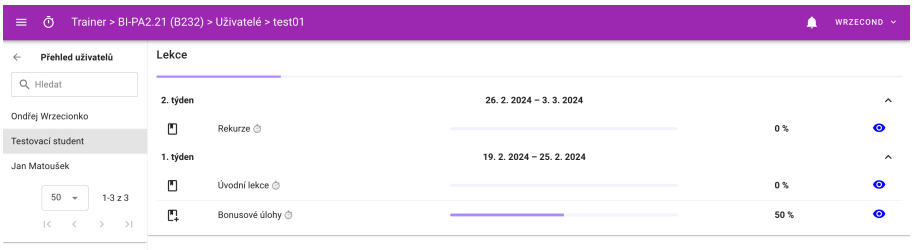
5.2.5 Přehled studentů

Jak v přehledu všech studentů v daném kurzu, tak v přehledu jednotlivých studentských řešení daného modulu v lekci jsem také přidal postranní navigaci pro snadnější orientaci mezi studenty / řešeními.

Obrázek 5.7: Detail modulu v lekci (implementace)



Obrázek 5.8: Úprava lekce (implementace)



Obrázek 5.9: Přehled studentů v kurzu – detail studenta

5.2.6 Vylepšení použitelnosti

Kromě redesignu stávajících obrazovek jsem se zaměřil také na zlepšení použitelnosti systému na základě problémů objevených při usability testování. Přidal jsem oznámení o akcích, jako je povolení/zrušení anonymního zveřejnění, nahrání kódu, odevzdání kódu, odeslání žádosti, úprava lekce/modulu, smazání, změna pořadí lekcí v týdnu a dalších.

Pro učitele jsem také přidal řadu tlačítek pro snažší navigaci v systému, jako třeba proklik mezi úpravou a zobrazením lekce nebo modulu nebo proklik na přehled všech studentských řešení daného modulu z detailu modulu.

5.3 Databáze

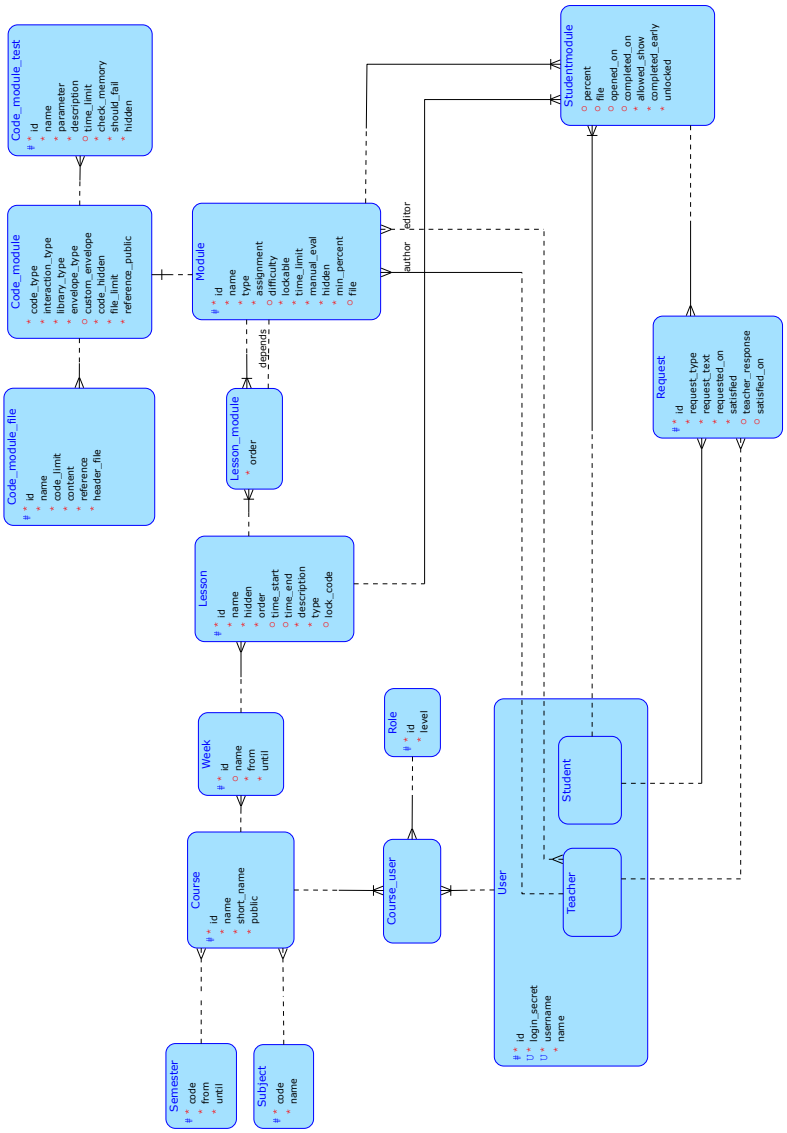
Pro druhou iteraci jsem aktualizoval konceptuální model na základě plánovaných změn. Nahradil jsem atributy semestr a předmět v kurzu za samostatné entity. Přidal jsem entitu pro týdny, která rozdělila původní 1:M vazbu kurzu a lekce na dvě vazby (kurz – týden, týden – lekce) pro snazší členění. Lekci jsem přidal atribut popis a typ lekce pro snadnější členění a pole time_limit jsem nahradil time_end symbolizující konec lekce. V modulu došlo k přidání atributu hidden, který umožní skrýt moduly a text žádostí je nově povinný. Poslední velkou změnou bylo přidání testů a souborů ke kódovému modulu v rámci přípravy na vícesouborový kódový modul.

Vzhledem k netriviálním změnám v databázi mezi 1. a 2. iterací (entita pro semestr, předmět, týden, testy a soubory v kódovém modulu) jsem musel provést migraci databáze. Pro migraci jsem se rozhodl napsat jednoduchý skript v jazyce PHP.

```php
foreach ($codeSt->fetchAll(PDO::FETCH_OBJ) as $code) {
    foreach (explode(", ", $code->param_right) as $paramRight)
        $createCodeTest->execute([
            'id' => $CODE_TEST_START_ID++,
            'name' => "Test $paramRight",
            'param' => intval($paramRight),
            'shouldFail' => 0, 'cmId' => $code->id
        ]);
    $createCodeFile->execute([
        'id' => $CODE_FILE_START_ID++, 'cmId' => $code->id,
        'codeLimit' => $code->code_limit,
        'content' => $code->code_shown,
    ]);
}
```

Výpis kódu 32: Migrace kódového modulu z 1. do 2. iterace

Tento skript nejprve pro každý kurz nahradí atribut `semestr` a `subject` za entitu. Pokud již semestr / předmět s daným kódem existuje, je použit, jinak je vytvořen. Následně se pro každý kurz vytvoří týden, který je nastaven všem lekcím v daném týdnu. Posledním krokem je pak přidání výchozího souboru se jménem `student.cpp` ke každému kódovému modulu, a vytvoření testů s parametry odpovídajícími původním polím `param_right` a `param_wrong`.



Obrázek 5.10: Druhý konceptuální model databáze

5.4 Návrh API

V návrhu REST API došlo především ke přidání nového zdroje /weeks pro týdny a k drobným úpravám v dalších zdrojích.

5.4.1 Zdroj /weeks

Systém umožní učiteli vytvářet nové týdny (POST /), upravovat stávající týdny (PATCH /id) a mazat je (DELETE /id). Dále poskytne metodu pro úpravu pořadí lekcí v kurzu (PATCH /id/lessons) a umožní zkopírovat týden do jiného kurzu (POST /id/courses/newCourseId).

5.4.2 Úpravy zdrojů

Do zdroje /courses byla přidána metoda pro získání detailu týdne se zadaným identifikátorem v konkrétním kurzu (GET /courses/courseId/weeks/weekId). Zdroj /lessons umožní získat detail týdne, ve kterém je zadaná lekce (GET /lessonId/week).

Zdroj /code při detailu kódového modulu (GET /id) nově zobrazí také všechny testy a soubory daného modulu. Také nově umožní mazat podle identifikátoru jednotlivé soubory (DELETE /files/fileId) a testy (DELETE /tests/testId).

5.5 Backend

V implementaci datové vrstvy (balíčků Entity a Repository), DTO a Controllerů došlo k přidání nových entit pro týden (`Week`, `WeekRepository` a všechny typy `WeekDTO`), soubory (`CodeModuleFile`) a testy (`CodeModuleTest`) v kódovém modulu a přidání / změn atributů ve stávajících entitách.

Přidal jsem také třídy `FileService` a `ImageService`, do kterých jsem přesunul veškerou logiku práce se soubory a obrázky.

Zásadních změn se dočkal balíček Service, který jsem kompletně přepracoval tak, aby reflektoval aktuální potřeby a nedocházelo k opakování kódu. K opakování kódu docházelo především ve dvou případech: při převodu tříd z balíčků Entity / DTO a během získávání entit a kontroly práv v každé metodě.

5.5.1 Převod mezi Entity a DTO

První problém spočíval v definici rozhraní `IService`. Každá Service totiž musela obsahovat implementaci metod pro převod mezi entitou a DTO. Jelikož byla ale jednotlivá DTO vnořená v ostatních (např. `CourseGetDTO` obsahuje `SubjectFindDTO`, `SemesterFindDTO`, `WeekGetDTO` i `LessonFindDTO`), byly tyto metody obsaženy ve více třídách. Duplicitní kód pak vedl k menší přehlednosti, větší náchylnosti k chybám a nutnosti při změně Entity nebo DTO změnit převod na více místech.

Řešením bylo vytvoření pomocné třídy `ConverterService`, která obsahuje všechny metody pro převod na jednom místě, a nedochází tak k duplicitám.

```
@Service
class ConverterService /* (...) */ {
    fun toFindDTO(subject: Subject?) = subject?.run {
        SubjectFindDTO(id, name, code)
    }
    fun toGetDTO(week: Week, user: User? = null) = week.run {
        WeekGetDTO(id, name, from, until, toFindDTO(course),
            lessons.filter { it.canView(user) }
                   .sortedBy { it.order }
                   .map { toFindDTO(it, user) })
    }
    fun toEntity(week: WeekCreateDTO, course: Course)
    = week.run { Week(name, from, until, course, emptyList()) }
    fun merge(week: Week, dto: WeekUpdateDTO) = week.run {
        Week(dto.name ?: name, dto.from ?: from, dto.until ?:
        until, course, lessons, id)
    }
}
```

Výpis kódu 33: Ukázka třídy `ConverterService` pro převod Entit a DTO

5.5.2 Kontrola přístupových práv

Druhý problém jsem nastínil již v první iteraci. V implementaci každé metody Service proběhne typicky zisk entity z databáze na základě jejího identifikátoru, kontrola oprávnění pomocí `canView` / `canEdit` a následné provedení akce s danou entitou. Jelikož k zisku entity a následné kontrole dochází téměř v každé metodě, kód se tímto stává velmi dlouhým a nepřehledným.

Tento problém jsem vyřešil přidáním dvou metod – `checkViewAccess` a `checkEditAccess`. Tyto metody získají entitu i uživatele, zkontrolují oprávnění a následně zavolají akci se získanou entitou / uživatelem.

```
// IServiceBase.kt
protected fun <X> checkViewAccess (entity: T, dto: UserFindDTO?,
block: (T, User) -> X) = tryCatch {
    val user = getUser(dto)
    if (!entity.canView(user)) throw
    ResponseStatusException(HttpStatus.FORBIDDEN)
    block(entity, user)
}

// LessonService.kt
override fun getByID(id: Int, dto: UserFindDTO?)
    = checkViewAccess(getEntityById(id), dto) { lesson, user ->
        converter.toGetDTO(lesson, user)
    }
```

Výpis kódu 34: Využití metody `checkViewAccess` pro kontrolu přístupu k lekci

5.6 Frontend

Během druhé iterace prošel frontend zásadními změnami, především proto, že hlavní dva pilíře této iterace byly uživatelské rozhraní a vícesouborové kódové moduly. Došlo tak k rozdělení na jednotlivé komponenty a implementaci nového uživatelského rozhraní, přidání překladů a rozšíření kódového modulu.

5.6.1 Uživatelské rozhraní

Jelikož je Vue.js hodně založené na komponentách, dal jsem si při implementaci nového uživatelského rozhraní na rozdělení na komponenty záležet. Složku `components/`, ve které byly původně v podstatě všechny komponenty kromě kódového modulu, jsem tak rozdělil na podsložky `course/` pro komponenty týkající se rozhraní seznamu kurzů a detailu kurzu, `lesson/` pro detail a úpravu lekce a modulu a `custom/` pro speciální komponenty využité napříč celým frontendem. Ve složce `modules/` pak nadále zůstaly komponenty kódového modulu.

Postranní menu

Oproti první iteraci jsem začal používat postranní menu, ať už pro seznam kurzů v detailu kurzu, seznam lekcí a modulů v detailu lekce, nebo pro jednotlivé záložky při úpravě lekce a modulu. Postranní menu jde do Vuetify layoutu přidat pomocí komponenty `VNavigationDrawer`.

```
<template>
  <v-navigation-drawer v-model="appState.leftDrawer">
    <LessonDetailLeftDrawer :lesson="$route.params.lesson"
    :module="$route.params.module" :user="$route.params.user" />
  </v-navigation-drawer>
  <LessonModuleDetail :lesson="$route.params.lesson"
    :module="$route.params.module" :user="$route.params.user" />
</template>
```

Výpis kódu 35: Definice komponenty `LessonModuleDetailView` pro detail lekce a modulu. S pomocí `v-navigation-drawer` je definováno postranní menu, ve kterém je komponenta `LessonDetailLeftDrawer`.

Tato komponenta ale ve výchozím nastavení ukáže postranní menu v desktop rozložení **vždy** a při mobilním rozložení naopak menu skryje. Jediný způsob, jak menu na mobilu zobrazit, je gestem potažení doleva, které ale na nejnovějších verzích Androidu a iOS odpovídá gestu pro navigaci o stránku zpět.

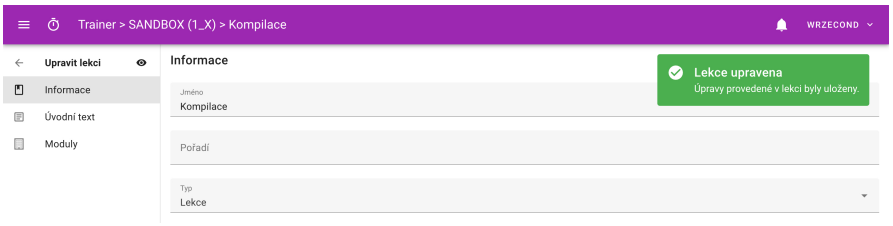
Aby šlo menu schovávat, musel jsem přidat do horní lišty tlačítko, které je řízeno proměnnou `appState.leftDrawer`. Aby se ale tlačítko neukazovalo na stránkách, kde postranní menu není k dispozici, musel jsem naimplementovat ještě dodatečnou logiku. Pokud je proměnná ve stavu `undefined`, menu není k dispozici. Jinak `false` menu skryje a `true` menu ukáže.

```
const appState = inject('appState')
appState.value.leftDrawer = true
onBeforeRouteLeave((to) => {
  if (['lesson-detail', /* ... */].includes(to.name)) return
  appState.value.leftDrawer = undefined
})
```

Výpis kódu 36: Logika pro přepínání postranního menu v detailu lekce a modulu. Při načtení komponenty se menu vždy ukáže (nastaví na `true`), při navigaci pryč z komponenty se vypne (hodnota `undefined`).

Notifikace

Jednou z důležitých vlastností pro použitelnost systému je, aby měl uživatel potvrzení o akcích, které provádí. Mezi takové akce patří zveřejnění anonymizovaného řešení, odevzdání řešení učiteli, nahrání souboru s kódem, uložení úprav, smazání lekce apod. Toho lze velmi dobře zacílit použitím „dočasných“ notifikací (nezaměňovat s notifikacemi o ohodnocení úlohy, které se zobrazují v navigační liště).



Obrázek 5.11: Notifikace o uložení lekce

Jelikož Vuetify neobsahuje žádnou komponentu, která by přímo dočasné notifikace zobrazovala, naimplementoval jsem vlastní komponentu `Notification Snackbar`. Ta obsahuje jednotlivé notifikace, které jsou implementovány pomocí komponenty `HideableAlert`, která jen rozšiřuje Vuetify komponentu `VAlert` tak, že se automaticky schová po `timeout` milisekundách.

```
<div class="notification-list" style="position: fixed; top:
64px; right: 0; cursor: pointer; z-index: 9999">
  <HideableAlert :notification="notification"
    :on-click="() => clickNotification(notification, ix)"
    :on-hide="() => hideNotification(notification)"
    :timeout="5000" :key="notification.title"
    v-for="(notification, ix) in appState.notifications" />
</div>
```

Výpis kódu 37: Komponenta `NotificationSnackbar` pro zobrazení dočasných notifikací. Při kliknutí je notifikace schována a pokud obsahovala odkaz, je uživatel přesměrován.

Komponenty

Stejně jako na backendu, i na frontendu došlo během implementace 1. iterace k častému opakování kódu. Příčinou bylo především nedostatečné znovuvyužívání komponent.

Během druhé iterace jsem tuto chybu napravil. Jedna komponenta je tak používána například pro mazání lekcí i týdnů (`CourseDetailDeleteDialog`), panel pro filtrování v přehledu studentských řešení (`LessonUserFilterPanel`), detail lekce i modulu (`LessonModuleDetail`) nebo panel pro nahrání souboru v kódovém / assignment modulu (`ModuleBottomOverlay`).

Výše zmíněné případy sjednotily ale typicky pouze jednu nebo dvě komponenty. K nejvýraznějšímu opakování kódu docházelo při načítání.

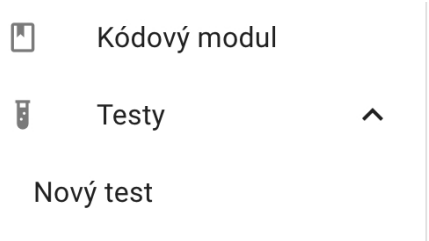
Typicky totiž každá komponenta, která něco načítala (ať už seznam kurzů, detail kurzu, lekce, modulu) obsahovala proměnnou pro načítaná data a chybu. V `<template>` se pak typicky nacházela logika typu „pokud `data === null`, ukaž načítání, pokud je `error !== null`, ukaž chybu, jinak ukaž data“. Tuto logiku jsem extrahoval do samostatné komponenty `LoadingScreen`.

Této komponentě tak stačí pouze předat proměnnou (`ref`) s načítanými daty, chybou a zadefinovat obsah tabulky (`table`), karty (`card`) nebo celé komponenty (`items`).

Úprava modulu

Jednou z dalších výzev, kterým jsem v průběhu implementace čelil, bylo postranní menu při úpravě modulu. Oproti úpravě lekce, kde jsou záložky vždy stejné (Informace, Úvodní text, Moduly) zde totiž záložky závisí na typu modulu. Kromě základních záložek (Informace, Zadání) má totiž kódový modul ještě specifické záložky (Testy, Výchozí kód, ...), které navíc obsahují podzáložky.

Záložky se při úpravě kódového modulu definují do seznamu záložek (proměnná `moduleEditTabList`). Každá záložka má kromě identifikátoru, jména a ikony ještě seznam podzáložek. Tyto podzáložky mají už pouze jméno a identifikátor, víceúrovňové podzáložky jsem se rozhodl nepodporovat. Logika schovávání podzáložek je v komponentě `ModuleCreateEditMenu`.



Obrázek 5.12: Postranní menu při úpravě kódového modulu

Komponentu pro úpravu modulu (`ModuleCreateEdit`) tvoří formulář s tlačítkem pro uložení modulu, ve kterém je obsah určen aktivní záložkou (proměnnou `moduleEditItem`).

```
<template> <!-- LoadingScreen.vue -->
  <slot name="items" v-if="items && !error">
    <v-card-item>  <slot name="prepend" />
      <slot name="content">
        <v-table> <tbody><slot name="table" /></tbody>
        </v-table>
      </slot>
    </v-card-item>
  </slot>
  <v-card-item v-else-if="error">
    <slot name="error"> {{ getErrorMessage(t, error) }} </slot>
  </v-card-item>
  <v-card-item class="pa-8 flex justify-center" v-else>
    <slot name="loading"> ... </slot>
  </v-card-item>
</template>

<!-- CourseList.vue -->
<LoadingScreen :items="courses" :error="error">
  <template v-slot:table>
    <CourseListRow :course="course"
        v-for="course in courses" :key="course.id" />
    <tr v-if="!courses.length">
      <td>{{ t('$vuetify.course_list_empty') }}</td>
    </tr>
  </template>
</LoadingScreen>
```

Výpis kódu 38: Definice komponenty `LoadingScreen` a její využití v komponentě pro seznam kurzů `CourseList`

```
<v-form>
  <ModuleCreateEditPartModule v-if="moduleEditItem.id ===
  MODULE_EDIT_ITEM_INFO" :lesson="lesson" />
  <TextEditor :editable="true" v-else-if="moduleEditItem.id ===
  MODULE_EDIT_ITEM_TEXT" v-model="moduleData.assignment" />

  <QuizEditModule :moduleId="moduleId"
    v-if="moduleData.type === 'QUIZ'" />
  <CodeEditModule :moduleId="moduleId"
    v-if="moduleData.type === 'CODE'" />
  <!-- TextModule / AssignmentModule have no content -->

  <v-btn type="submit" :disabled="!submitAllowed()">...</v-btn>
</v-form>
```

Výpis kódu 39: Ukázka komponenty pro úpravu modulu (`ModuleCreateEdit`).

5.6.2 Překlady

Požadavek na podporu více jazyků byl na systém již od začátku, proto jsem se ho rozhodl implementovat, a to pro dva jazyky – češtinu a angličtinu. Vuetify překlad nativně podporuje, pro jeho aktivaci stačí přidat do definice parametr `locale` s aktuálním překladem.

Jelikož výchozí překlad `cs` a `en` obsahuje pouze základní texty (upravit, zavřít), definoval jsem vlastní překlady `customCs` a `customEn`. Definice překladů jsou v souboru `plugins/translations.js`.

```
export const customCs = {
    ...cs,
    tooltip_lesson_timed: "Včasné vyřešení do {0}",
    code_module: {
        title: "Kódový modul",
    }
    // ...
}
```

Výpis kódu 40: Definice překladu pro Vuetify. Jedná se o běžný Javascriptový objekt, který pro každý klíč obsahuje hodnotu nebo vnořený objekt.

Pro použití překladu uvnitř kódu stačí v jakékoliv komponentě použít funkci `useLocale()`, která vrátí instanci locale včetně překladové funkce `t`. Při zavolání `t("klíč", param1, param2)` funkce vrátí překlad pro zadaný klíč, přičemž tato funkce umí pracovat i s parametry.

Každý klíč musí začínat `"$vuetify."` následované cestou v objektu (pro vnořený objekt se používá tečka). Například pro výše uvedený překlad by byl první klíč `"$vuetify.tooltip_lesson_timed"`, zatímco druhý klíč by byl `"$vuetify.code_module.title"`.

5.6.3 Kódový modul

Pro druhou iteraci systému jsem provedl významnou aktualizaci kódového modulu, která vycházela ze zkušeností s používáním během první iterace. Mezi zásadní nedostatky patřila nemožnost pracovat s velkými vstupy, chybějící popis a nastavení u jednotlivých testů nebo již zmiňovaná podpora více souborů.

Velké vstupy

Pro první iteraci bylo zvoleno přesměrovávání vstupu a výstupu do/z programu s pomocí WASI. Takové přesměrování odpovídá způsobu, jakým se běžně se vstupem a výstupem v jazycích C a C++ pracuje. Při úlohách pracujících s velkými vstupy se ale objevil problém, protože se vstup a výstup mezi programem a „testovacím prostředím“ v Javascriptu posílal přes JavaScript Worker. Ten se při větších datech (přes 1 MB) zasekl. Další nevýhodou tohoto přístupu byla nutnost manuálně nahrávat archivy s testovacími vstupy a referenčními výstupy k úloze, což prakticky znemožňovalo testy náhodnými daty a omezovalo množství dat, které se do programu mohlo maximálně poslat.

Vzhledem k těmto omezením už v průběhu první iterace učitelé začali obcházet testování vstupu a výstupu pomocí modulu typu `TEST_ASSERT`. Funkce pro práci se vstupem a výstupem byly předefinovány na práci s in-memory soubory a testovací program tak mohl přímo generovat vstup a pracovat se studentovým výstupem.

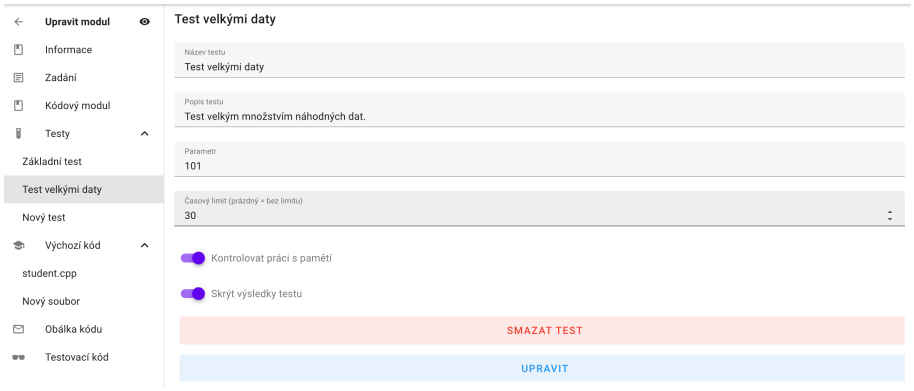
```
FILE * test_stdin, test_stdout;

template<class... Args> static int test_scanf(const char * fmt,
Args... args) {
    return fscanf(test_stdin, fmt, args...);
}
template<class... Args> static int test_printf(const char * fmt,
Args... args) {
    return fprintf(test_stdout, fmt, args...);
}

#define scanf test_scanf
#define printf test_printf

namespace CStudent {
    #include "tested.cpp"
    #undef scanf
    #undef printf
}

void generateOutput (const char * in, size_t szIn, char *
outbuf, size_t szOut) {
    test_stdin = fmemopen(const_cast<char *>(in), szIn, "r");
    test_stdout = fmemopen(outbuf, szOut + 10, "w");
    CStudent::main();
    fclose(test_stdin); fclose(test_stdout);
    test_stdin = test_stdout = 0;
}
```

Výpis kódu 41: Řešení testování velkých vstupů v kódovém modulu. V testovacím souboru jsou definovány in-memory soubory `test_stdin` a `test_stdout`, pro které jsou předefinované funkce `scanf` a `printf`. Funkce `generateOutput` pak umožňuje vygenerovat výstup programu pro zadaný vstup do předem alokovaného řetězce.

Po analýze a diskuzi s učiteli předmětu jsem rozhodl, že vstup i výstup bude do programu posílán vždy výše uvedeným způsobem. Kromě možnosti testovat libovolně velké vstupy a kontrolovat například řádky výstupu nezávisle na pořadí má tento přístup výhodu také ve značném zjednodušení implementace pro jednotlivé typy modulu – způsob kompilace a linkování je stejný. Typ kódového modulu `WRITE_IO`, u kterého by tento přístup použít nešel, byl zrušen – v průběhu semestru byl použit jen jednou a jeho priorita byla nízká.

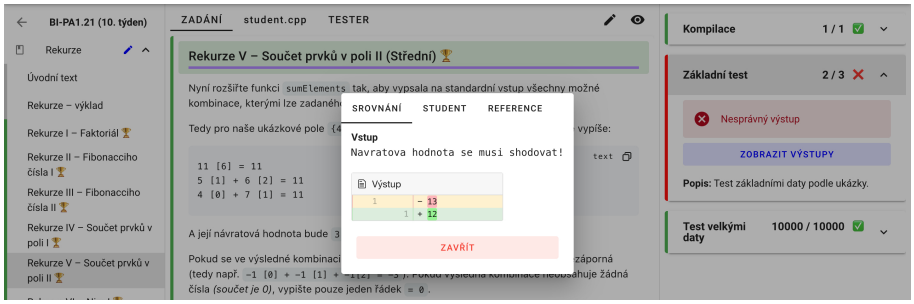
Testování

Během první iterace byly testy určeny nahranými soubory v případě I/O testů, nebo parametry (čísly) zadanými v rozhraní pro úpravu kódového modulu. Při testování pak především u modulů typu `TEST_ASSERT` student viděl pouze hlášku Test *[číslo]* prošel/neprošel bez informace o tom, co je testováno. Pro testy jsem proto ve druhé iteraci vytvořil samostatnou entitu. Učitel tedy může pro danou úlohu nastavit jednotlivé testy, včetně názvu, popisu, časového limitu pro běh nebo nastavení kontroly paměti.



Obrázek 5.13: Učitelské rozhraní pro úpravu testu

V detailu kódového modulu jsem pak přidal pravé postranní menu, ve kterém se ukazují všechny testy – už před spuštěním programu tak student vidí, co je testováno a v případě chyby si může zobrazit přehledný rozdíl mezi očekávaným a jeho výstupem.



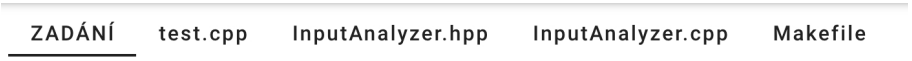
Obrázek 5.14: Postranní panel s testy a dialogové okno pro rozdíl výstupů

Vícesouborové moduly

V první iteraci byl kompilován a linkován vždy pouze jeden soubor, který obsahoval direktivu `#include` pro vložení druhého souboru. Student upravoval vždy jeden soubor s kódem, učitel podle typu kódového modulu nejvýše dva soubory (výchozí studentův kód, testovací kód). Pro potřeby výuky programovacího jazyka C v předmětu PA1 to bylo dostačující.

Pro druhou iteraci byla plánována podpora pro jazyk C++, kde je práce s hlavičkovými soubory především pro výuku objektově orientovaného programování nezbytná. Provedl jsem tedy analýzu detailních potřeb předmětu PA2 a na jejím základě jsem provedl implementaci druhé iterace kódového modulu.

Podobně jako má kódový modul definované testy, stejně má definované i soubory. Studenti si tedy soubory nevytváří libovolně, ale mají pevně určeno, že v modulu bude například soubor `test.cpp` pro jejich vlastní testy, `CString.hpp` jako hlavičkový soubor pro třídu `CString` a `CString.cpp` jako soubor pro implementaci této třídy. Každý soubor má limit na délku kódu, výchozí obsah, referenční řešení a nastavení, zda se má kompilovat.



Obrázek 5.15: Navigace mezi soubory v kódovém modulu

Aby šlo vícesouborové moduly rozumně testovat, je každý studentův kód a referenční řešení vloženo do tzv. „obálky“. Obálka umožňuje předefinovat funkce pro práci se vstupem a výstupem, omezit hlavičkové soubory a třídy, se kterými bude moct student pracovat nebo definovat hlavičku funkce, kterou má studentův kód otestovat. Učitel má na výběr z výchozích obálek (C/C++ hlavičky, C hlavičky a přesměrování vstupu, C++ hlavičky se zakázáním STL), nebo může definovat obálku vlastní.

```
struct Node {
    int val;
    struct Node * next;
};

int listSum(struct Node * node);
```

```
#include <assert.h>
namespace __STUDENT_NAMESPACE__ {
    #line 1
    __STUDENT_FILE__
}
```

Výpis kódu 42: Ukázka obálky kódu pro modul typu `WRITE_ASSERT`, kde student testuje funkci `listSum` počítající součet hodnot prvků ve spojovém seznamu. V obálce jsou využity dvě „proměnné“ – `__STUDENT_NAMESPACE__` (`CStudent` v případě kódu studenta, `CRef` v případě referenčního řešení) a `__STUDENT_FILE__` (studentův kód / reference).

Druhým prvkem, který usnadňuje práci při testování, je knihovna, která obsahuje pomocné funkce pro vygenerování výstupu zadané funkce a porovnání dvou čísel nebo řetězců. Knihovnu jsem naimplementoval ve dvou verzích – pro jazyk C (základní, rychlá) a pro jazyk C++ (pomalejší, pokročilejší funkce, jako porovnávání dvou tříd operátorem =).

```
class CTester {
public:
    static std::istringstream test_stdin;
    static std::ostringstream test_stdout;
    static bool assertBool ( bool condition, const std::string &
    input, const std::string & output = "", const std::string &
    ref = "" );
    static bool assertEqual ( const std::string & input, const
    std::string & output = "", const std::string & ref = "" );
    template<typename T>
    static bool assertEqual ( const std::string & input, const T
    & out, const T & ref );
    static void outputPrint ( const std::string & input, const
    std::string & output );
    static bool compareLines ( const std::string & l, const
    std::string & r );
};

template<typename T, typename R, typename... Args>
std::string generateOutput ( const std::string & in, T function,
R & retRef, Args... args);
template<typename T, typename... Args>
std::string generateOutput ( const std::string & in, T function,
Args... args);
```

Výpis kódu 43: Hlavičkový soubor C++ knihovny, kterou jsem vytvořil pro potřeby testování studentských kódů.

Funkce `assertBool` a `assertEqual` fungují podobně jako C assert – v případě nesplněné podmínky / nerovnajících se argumentů ukončí program. Oproti assertu ale tyto funkce navíc pomocí WASM exportů (viz další kapitola) umožní zobrazení vstupu, výstupu a ref. řešení, na kterém program selhal.

Funkce `compareLines` umožní porovnat řádky dvou textů nezávisle na jejich pořadí. Do WASM exportu se uloží buď informace o různém počtu řádků, nebo první řádek, na kterém se výstup a ref. řešení liší.

Funkce `generateOutput` zavolá předanou funkci se zadanými argumenty s přesměrovaným vstupem a výstupem. Vstup je čten z parametru `in`, výstup je návratová hodnota funkce.

Při kompilaci jsou pak všechny studentovy soubory vloženy do souborového systému. Nehlavičkové soubory jsou vloženy do obálky a zkompilovány do příslušných objektových souborů. Ty jsou v dalším kroku slinkovány (společně s objektovým souborem knihovny) a výsledný program je spuštěn s jednotlivými parametry podle vytvořených testů.

5.6.4 WASM

V první iteraci bylo automatické hodnocení programu plně v režii jednotlivých komponent `CodeModule`. Při testování I/O byl porovnán výstup programu spuštěného s jednotlivými vstupy. U typu `TEST_ASSERT` bylo doběhnutí programu s nulovým návratovým kódem považováno za splnění testu.

Pro druhou iteraci jsem se rozhodl o přesunutí tohoto hodnocení na stranu C/C++ programů pomocí tzv. WASM exportů. Informaci o počtu provedených a počtu úspěšných testů, velikosti alokované a uvolněné paměti a případný vstup / výstup, na kterém testy selhaly, tak poskytují přímo statické metody třídy `CTester` v C/C++ knihovně.

```
__attribute__((export_name("trainer_input")))
const char * testInput() {
    return CTester::m_Input;
}
```

Výpis kódu 44: S pomocí atributu kompilátoru (`__attribute__`) je návratová hodnota funkce `testInput` (testovací vstup programu) exportována do WASM pod názvem `trainer_input`. Obdobným způsobem je exportován i počet provedených a úspěšných testů, výstup programu a referenční výstup.

V komponentě `CodeModuleBase` jsou po doběhnutí programu získány jednotlivé exporty a podle typu kódového modulu a parametrů jednotlivých testů je rozhodnuto o splnění či nesplnění testu. V případě nesplnění student získá dodatečnou informaci o počtu nesplněných testů a vstupu, na kterém program selhal.

```
const checkTestOutput = (test, wasmResult, idx) => {
  test.total = wasmResult?.testsTotal ?? 1
  if (!wasmResult || wasmResult.exitCode > 128) {
    return t('$vuetify.code_module.error_ret_code_not_ok',
        wasmResult.exitCode)
  }
  test.passed = wasmResult.testsPassed

  if (wasmResult.testsPassed !== wasmResult.testsTotal)
    return t('$vuetify.code_module.error_invalid_output')

  if (test.checkMemory && wasmResult.freeCalls !==
  wasmResult.allocCalls)
    return t('$vuetify.code_module.error_memory_not_freed',
    wasmResult.freeCalls, wasmResult.allocCalls)

  return null // success
}
```

Výpis kódu 45: Ukázka kódu pro vyhodnocení výsledku testu (zjednodušeno pro typy `TEST_ASSERT` a `TEST_IO`). Nejprve je zkontrolován návratový kód, následně počet splněných a provedených testů a nakonec uvolněná paměť.

Aby kompilace C++ fungovala korektně, musel jsem do `clangCommonArgs` přidat přepínače pro experimentální podporu výjimek (`-fexceptions`, `-fwasmexceptions`) a nastavení verze standardu C++ 20. Při linkování jsem přidal přepínače pro podporu C++ a výjimek (`-lc++`, `-lc++abi`, `-lunwind`).

Během testování nového způsobu kompilace, linkování a spouštění se objevil problém, kdy kvůli více kompilovaným souborům kompilace již netrvala stovky milisekund, ale jednotky sekund. Jelikož kompilace probíhala na hlavním vlákně, docházelo během kompilace k zamrznutí prohlížeče.

Řešením tohoto problému bylo přesunout kompilaci do Javascript workeru. To však způsobilo, že kompilace buď vůbec nedoběhla, nebo doběhla za desítky sekund. Problémem byla úprava, kterou jsem provedl v první iteraci, kdy jsem nahradil předávání spouštěného souboru ve Workeru (program s kódem) pomocí URL za předávání konkrétního programu. V případě kompilace byl totiž tímto programem kompilátor, který měl přibližně 40 MB. Knihovnu WASI jsem tedy rozšířil tak, aby umožňovala jak předání URL adresy pro velké soubory (kompilátor), tak předání binární reprezentace programu pro malé soubory (studentův program).

Druhou úpravou v knihovně WASI bylo zpracování exportů z doběhnutého programu a jejich přidání do `wasmResult`. U celočíselných proměnných jako počet testů / velikost paměti šlo jen o zkopírování hodnoty proměnné. U řetězců (vstup, výstup, reference) jsem ale musel implementovat dodatečnou logiku – hodnota vrácené proměnné totiž obsahovala pouze ukazatel na místo v paměti, kde řetězec začíná.

Pro export řetězcových proměnných jsem tedy naimplementoval pomocnou funkci `readStr`, která čte řetězec z paměti po jednotlivých znacích, dokud nedosáhne nuly, která C řetězec ukončuje.

```
const readStr = (pos) => {
    let readPos = pos
    let charlimit = 8192
    let memview = new Uint8Array(this.memory.buffer)
    while ( --charlimit > 0 ) {
        const chr = memview[readPos++]
        if (chr === 0) break;
    }
    return new TextDecoder().decode(memview.subarray(pos,
    readPos))
}

const sti = () => this.instance.exports.trainer_input ?
    readStr(this.instance.exports.trainer_input()) : ""

// start program, check for exceptions
return { /* ... */, input: sti() }
```

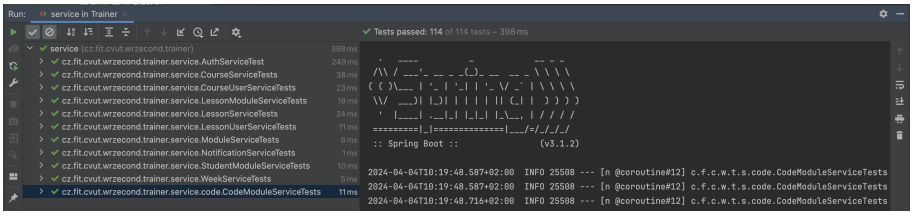
Výpis kódu 46: Nově přidané funkce `readStr`, `sti` a další ve WASI knihovně. V případě vstupu je čten textový řetězec z paměti, u číselných hodnot je přímo čtena hodnota. Hodnota neuvedených exportů je nula / prázdný řetězec.

5.7 Testování

Oproti první iteraci jsem měl během letního semestru mnohem více času a prostoru pro testování, proto jsem důkladně provedl jak jednotkové, tak integrační testování. Obdobně jako v první iteraci, i nadále pokračovala komunikace se studenty, nově ale také s učiteli – systém během semestru používalo 12 nových učitelů v celkem 11 paralelkách (s 528 studenty).

5.7.1 Jednotkové testování

Jednotkové testy patří mezi tzv. automatizované testy. Stačí je tedy jednou napsat a v případě správně nakonfigurovaného CI se po každé změně v kódu spustí znovu. Jednotkové testy testují určitou „jednotku“ kódu, typicky jednu třídu a její konkrétní metody. Vzhledem k povaze projektu jsem se tedy rozhodl takto otestovat na backendu balíček Service, který obsahuje veškerou aplikační logiku. Frontend, který obsahuje logiku zobrazení dat získaných z backendu, otestuji uživatelským testováním.



Obrázek 5.16: Úspěšně provedené jednotkové testy

Pro testování Service jsem použil framework Kotest [68]. Pro každou testovanou třídu `Service` je vytvořena odpovídající třída s testy `ServiceTest`, která dědí z `StringSpec`. Jednotlivé testy jsou pak definovány pomocí DSL (doménově specifického jazyka).

```
@SpringBootTest(classes = [CourseService::class])
class CourseServiceTests(
    @MockBean val cr: CourseRepository,
    @MockBean(answer = Answers.CALLS_REAL_METHODS)
    val converterService: ConverterService,
): StringSpec({ /* ... */ })
```

Výpis kódu 47: Ukázka definice třídy s testy pro `CourseService`

Abych testoval opravdu pouze chování požadované třídy, musím všechny její závislosti tzv. namockovat. Typicky se pro mockování v Kotlinu používá framework MockK [69]. Při jeho použití se ale vyskytl problém, kdy MockK nedokázal správně nainjectovat závislosti pro `ConverterService`, kterou má většina Service jako atribut. Použil jsem proto verzi frameworku Mockito [70] pro Kotlin, ve které se všechny namockované třídy definují pomocí anotací `@MockBean`.

```
beforeTest { reset(cr) }
"getOne" {
    given(cr.getReferenceById(course.id)).willReturn(course)
    service.getByID(course.id, userDto) shouldBe gdto
    verify(cr).getReferenceById(course.id)
}
"getOne_404" {
    given(cr.getReferenceById(anyInt())).willThrow(...)
    val e = shouldThrow<ResponseStatusException> {
        service.getByID(666, userDto)
    }
    e.statusCode shouldBe HttpStatus.NOT_FOUND
}
```

Výpis kódu 48: Ukázka jednotkových testů pro `CourseService` (zjednodušeno). Před každým testem jsou mocky resetovány, v jednotlivém testu jsou mocky nastaveny, následně je zkontrolován pomocí `shouldBe` výsledek, a je zkontrolováno, zda byly zavolány požadované metody. Místo výsledku může být zkontrolováno vyhození výjimky pomocí `shouldThrow`.

Během jednotkového testování jsem odhalil některé chyby v kontrole oprávnění při přístupu k jednotlivým entitám (student viděl připojovací kód ke kurzu, mohl si zobrazit editaci modulu nebo lekce a zjistit tak kód pro odemčení úloh), započítávání skrytých lekcí do celkového počtu lekcí v kurzu a špatně nastavenou nullabilitu data začátku / konce lekce nebo připojovacího kódu ke kurzu v UpdateDTO, kvůli kterým nešlo smazat začátek / konec lekce.

5.7.2 Integrační testování

Integrační testování umožňuje otestovat část systému jako celek, který na dané požadavky poskytuje správné odpovědi. To je vhodné pro REST API poskytované backendem.

Pro integrační testování používám nástroj Postman [71]. Ten umožňuje organizovat jednotlivé požadavky na API do složek (např. kurzy, lekce, moduly) a pro každou ze složek i celý projekt nastavit používané proměnné, které lze následně používat kdekoliv v požadavcích, i v testech.

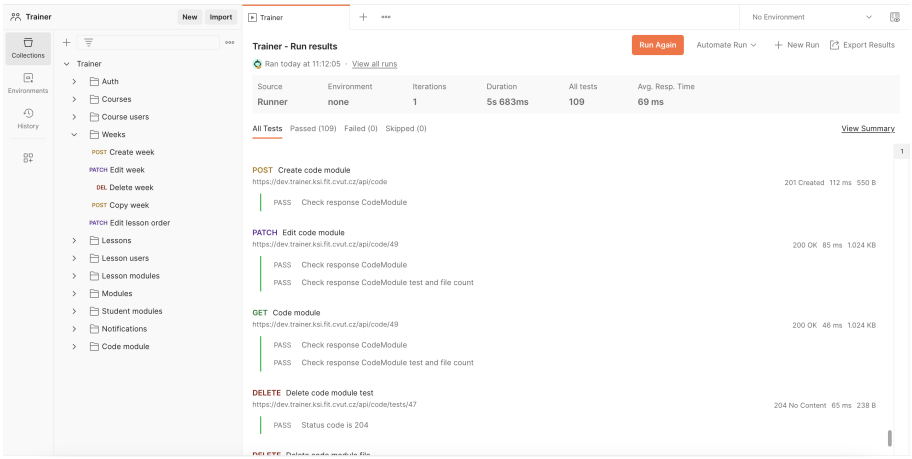
Při testování používám tři globální proměnné, které je třeba nastavit před testováním – `API_URL` pro adresu API serveru a přihlašovací klíče pro studenta (`STUDENT_SECRET`) / učitele (`TEACHER_SECRET`). Před spuštěním každého požadavku je pak nastaven do proměnné `LOGIN_SECRET` odpovídající klíč podle toho, jestli má být proveden z pohledu studenta, nebo učitele.

Kromě globálních proměnných pak používám proměnné také pro uložení identifikátoru entity (kurzu, lekce, modulu, ...), se kterou dále pracuji nebo uložení identifikátoru právě vytvořené entity. Typická posloupnost u klasických CRUD operací je tedy získat všechny entity (GET /), uložit si identifikátor první entity (`getId`), provést GET /getId, následně vytvořit novou entitu (POST /), uložit si její identifikátor (`createId`), upravit ji (PATCH /createId) a smazat ji (DELETE /createId).

```
pm.test("Result course matches", () => {
    const courseId = pm.collectionVariables.get("courseId");
    pm.expect(pm.response.json().id).to.equal(courseId);
    pm.expect(pm.response.json().name).to.equal("Test");
    pm.expect(pm.response.json().subject.code).to.equal("TEST");
    pm.expect(pm.response.json().semester.code).to.equal("A");
    pm.expect(pm.response.json().role).to.equal("STUDENT");

    const week = pm.response.json().weeks[0]
    pm.collectionVariables.set("weekId", week.id);
    pm.expect(week.name).to.equal("TestWeek");
    pm.expect(week.course.id).to.equal(courseId);
});

pm.test("Student can't see course secret", () => {
    pm.expect(pm.response.json().secret).to.be.null;
});
```

Výpis kódu 49: Ukázka testů pro metodu GET /courses/courseId. Odpověď musí obsahovat kurz v předmětu TEST v semestru s kódem A, ve kterém je uživatel student. Jeho identifikátor se zároveň musí shodovat s hodnotou proměnné `courseId`. Následně je zkontrolováno i jméno prvního týdne a identifikátor tohoto týdne je nastaven do proměnné `weekId` pro kontrolu v dalších požadavcích. V druhém testu kontroluji, že je vrácený připojovací klíč do kurzu prázdný, a student ho tedy opravdu nevidí (i když je nastaven).

Během integračních testů jsem odhalil bezpečnostní chybu, díky které mohl libovolný učitel přesunout jakoukoliv lekci v systému (i z předmětů, které neučí) do týdne v předmětu, který učí. Přišel jsem také na chybu, která znemožňovala kopírování kódového modulu do nové lekce. Kromě těchto větších chyb se jednalo už pouze o drobné úpravy v DTO.



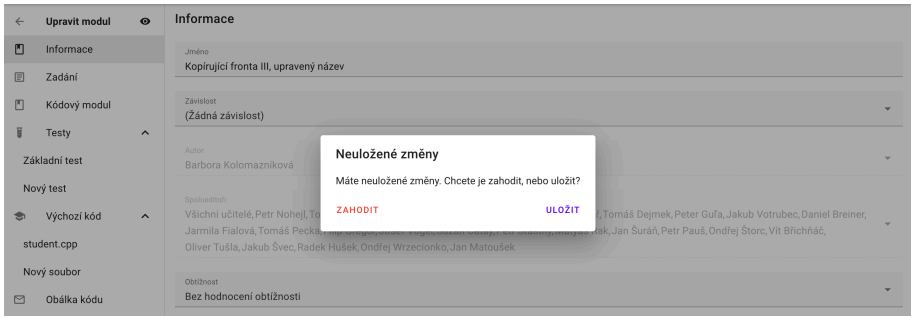
Obrázek 5.17: Úspěšně provedené integrační testy

5.7.3 Uživatelské testování

Uživatelské testování v druhé iteraci probíhalo oproti první iteraci nejen se studenty, ale i s učiteli. Intenzivní komunikace s oběma typy uživatelů v průběhu semestru pomohla odhalit nejen problémy, na které se nepřišlo během testování návrhu uživatelského rozhraní, ale také další chyby vzniklé při implementaci nového rozhraní a vícesouborových modulů.

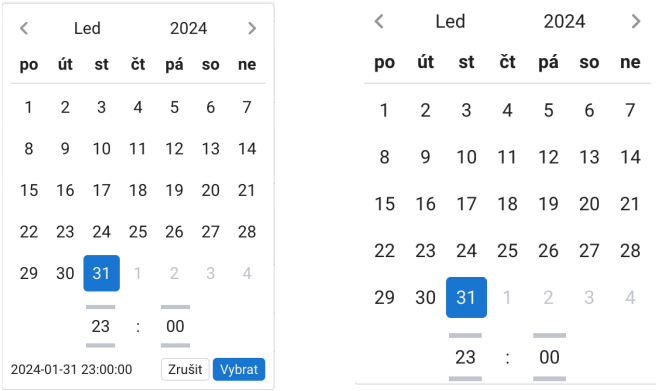
Aby byla komunikace s uživateli snadnější, založil vedoucí práce v rámci skupiny na Gitlabu repozitář pro zadávání zpětné vazby přístupný pro všechny studenty i učitele FIT. Zatímco studenti preferovali pro zpětnou vazbu osobní komunikaci, učitelé tento repozitář využili a poskytli tak velmi užitečnou zpětnou vazbu.

Jedním z odhalených nedostatků uživatelského rozhraní byl chybějící dialog při opuštění stránky bez uložení úprav. Mohlo se tak stát, že učitel omylem při úpravě lekce / modulu přišel o svoji práci. Do komponenty pro úpravu lekce a modulu jsem proto přidal zobrazení tohoto dialogu pro zamezení těchto situací.



Obrázek 5.18: Dialog pro potvrzení opuštění stránky bez uložených změn

Do repozitáře učitelé také hlásili problémy s používáním datepickeru při vybírání začátku a konce lekce. Ten ve výchozím nastavení totiž požadoval, aby po zadání času uživatel stiskl na tlačítko „Vybrat“.



Obrázek 5.19: Staré (vlevo) a nové (vpravo) rozhraní datepickeru

Pokud byl datepicker uzavřen kliknutím jinde na stránce, čas nebyl uložen. Podobný problém nastal v situaci, kdy učitel nakopíroval datum a čas přímo do textového pole. Řešením bylo změnit mód datepickeru v nastavení komponenty, kterou jsem využíval – odstranil jsem tlačítko pro potvrzení.

Dalším implementovaným vylepšením uživatelského rozhraní pro učitele bylo sjednocení skrývání lekcí a modulů. Lekce se totiž skrývaly, zatímco moduly se zveřejňovaly, což vytvářelo nejednoznačnost. Rozhodl jsem tedy o sjednocení rozhraní na skrývání.

Se skrýváním lekcí a modulů souvisel problém, kdy učitel vytvořil novou lekci / modul (skryté), které následně neviděl. Učitelé to hlásili jako chybu při vytváření lekce a modulu, ve skutečnosti ale spočíval problém v tom, že se skryté lekce a moduly v anonymním módu neukazují. Přidal jsem proto do anonymního módu indikaci počtu skrytých lekcí a modulů.

Obrázek 5.20: Indikace skrytých modulů a lekcí (emotikona ducha)

Ve studentském uživatelském rozhraní studenti a učitelé nahlásili nedostatky jako nemožnost scrollovat textem chyby při kompilaci, špatné zobrazení některých prvků uživatelského rozhraní na mobilních zařízeních nebo chybějící popisky některých tlačítek.

Mezi implementované nápady uživatelů patřilo také inteligentní nahrávání souborů s kódem ve vícesouborovém modulu. Pokud má modul pouze jeden soubor, je do něj vložen obsah nahraného souboru. V případě více souborů se musí shodovat konec jména nahraného souboru (např. soubory `ref-test.cpp`, `test.cpp` i `mycustomtest.cpp` budou nahrány do souboru `test.cpp`). V praxi si tak student může stáhnout výchozí kód jako archiv, rozbalit jej, upravovat, a následně nahrát zpět vybrané soubory bez nutnosti překlikávat mezi záložkami s jednotlivými soubory.

5.7.4 Studentský dotazník

Stejně jako v zimním, i v letním semestru jsem vytvořil v polovině semestru pro studenty s pomocí Google Forms [7] dotazník, kde jsem se ptal, jak se jim systém líbí, co je na systému štve, a jestli jim pomohl k lepšímu pochopení látky PA2. Oproti dotazníku ze zimního semestru jsem tentokrát ale přidal ještě podmíněnou sekci pro studenty, kteří již systém používali v zimním semestru s porovnáním „staré“ a „nové“ verze. Na dotazník odpovědělo 41 studentů (z 528), z nichž byla většina (87,8 %) z pěti paralelek (č. 3 až 7).

Hodnocení napříč paralelkami se příliš nelišilo, s výjimkou paralelky č. 3, která hodnotila především uživatelské rozhraní velmi negativně (4.6 oproti celkovému průměru 7.4 / 10, 10 je nejlepší). Zeptal jsem se proto učitelů této paralelky, jak systém používali a v čem byl problém.

Problémem bylo, že v této paralelce byly používány především kvízy (na kterých nepracuji já, ale Matej Pašek), které na začátku semestru neměly vyladěné rozhraní pro dlouhé otázky. To korespondovalo i s textovými komentáři v otázkách, co konkrétně se studentům nelíbilo – na základě negativní zkušenosti s kvízy na prvních cvičeních tak u této paralelky převládl negativní postoj k systému. V celkovém hodnocení systém obdržel průměrné hodnocení 7.80, uživatelské rozhraní 7.39, přínos Traineru ve výuce PA2 byl hodnocen známkou 7.22 / 10.

Mezi negativními vlastnostmi systému se kromě výše zmíněných kvízů vyskytovaly problémy s kompilací a lehce přehlcené uživatelské rozhraní. Problémy s kompilací byly z větší části opraveny a souvisely s úlohami s velkým množstvím kompilovaných souborů. Možným budoucím řešením by bylo znovuvyužití JavaScript workerů při kompilaci. Zjednodušení uživatelského rozhraní může v budoucnu řešit další tým v předmětu NI-NUR.

Z pozitivních vlastností opět vynikala přehlednost, koncept systému samotného, možnost procvičovat si úlohy se zpětnou vazbou, možnost dostat na cvičení zpětnou vazbu skrze anonymní zveřejnění kódu nebo přehledná struktura systému.

Na podmíněnou sekci s porovnáním první a druhé iterace systému zodpovědělo 15 studentů, kteří systém již používali v minulém semestru. Vylepšení hodnotili na škále 1 (první iterace byla lepší) až 10 (druhá iterace byla lepší) známkou 8. První iterace byla podle studentů přehlednější a jednodušší na používání, druhá iterace zase poskytla detailnější popis provedených testů a chyb.

5.8 Dokumentace

Jelikož budou na vývoj systému v budoucnu navazovat další studenti v rámci svých závěrečných prací a předmětů Softwarový týmový projekt 1/2, projekt jsem řádně zdokumentoval, a to jak z hlediska kódu, tak z hlediska uživatelského.

5.8.1 Kód

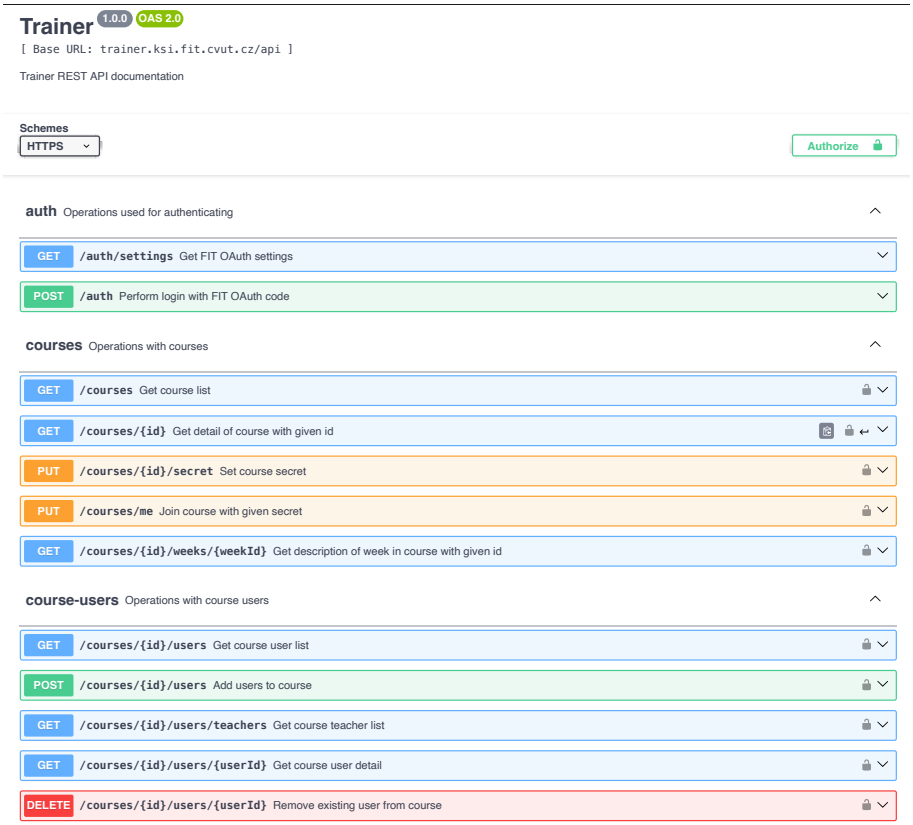
V celém kódu, a to jak na backendu, tak na frontendu, jsou využity dokumentační komentáře u hlaviček metod i jednořádkové komentáře na místech, kde je složitější logika. Návrh, architektura systému a implementace jsou popsány v této diplomové práci, přičemž jednotlivé diagramy, návrh uživatelského rozhraní, návrh REST API, kód, jednotkové i integrační testy jsou součástí společného repozitáře používaného pro vývoj systému. Vše výše jmenované je také obsaženo v příloze této závěrečné práce.

5.8.2 REST API

Pro tvorbu dokumentace REST API jsem využil nástroj Swagger [72]. Dokumentace je dostupná v přílohách jako soubor `swagger.yaml`, který lze do editoru vložit (záložka File – Import file). Po vložení lze procházet popis jednotlivých zdrojů, možné metody, parametry i popis návratových kódů. Přímo z nástroje lze také požadavky zkoušet.

```kotlin
/**
 * Find user's lessons that are currently running
 * @property userDto user to search lessons for
 * @return list of lessons running at given time
 */
fun currentLessons(userDto: UserFindDTO?) = tryCatch {
    val user = getUser(userDto)
    val now = Timestamp.from(Instant.now())
    repository.findUserLessonsAtTime(user, now)
        .map { converter.toGetDTO(it, user) }
}
```

Výpis kódu 50: Dokumentační komentáře v kódu

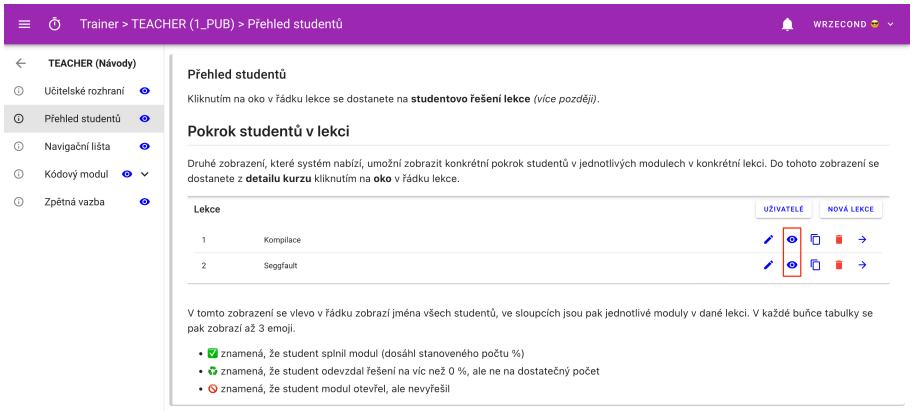


Obrázek 5.21: Dokumentace REST API v nástroji Swagger

5.8.3 Uživatelská

Jednou z nedílných součástí systému je také jeho dokumentace, a to jak pro studenty, tak pro učitele. Dokumentace je dostupná přímo jako kurz pro studenty (veřejný) a učitele ve speciálním semestru 1_PUB.

Výhodou tohoto přístupu k dokumentaci je možnost úpravy lekcí libovolným učitelem bez nutnosti znovu kompilovat backend / frontend, nebo měnit konfigurační soubory backendu. Nevýhodou je složitější přenos dokumentace mezi instancemi systému (lokální testovací / vývojářská verze / ostrá verze), kdy musí přenos proběhnout zkopírováním příslušných lekcí, týdnů a modulů v databázi.



Obrázek 5.22: Uživatelská dokumentace pro učitele

Možnou alternativou tohoto přístupu do budoucna by bylo mít dokumentaci na Gitlabu, buď ve formě veřejné Wiki, nebo v Markdownu s následnou generací.

5.9 Zhodnocení iterace

V druhé iteraci jsem nejprve implementoval nové, kompletně přepracované uživatelské rozhraní pro studenty i učitele, překlady a přidal jsem do kódového modulu podporu pro C++ a možnost vícesouborových úloh. Druhou iteraci jsem v průběhu semestru důkladně otestoval jednotkovými, integračními i uživatelskými testy a na závěr ji zdokumentoval, a to jak pro budoucí programátory, tak pro uživatele. Během semestru systém využívalo 12 učitelů a 528 studentů předmětu PA2 v celkem 11 paralelkách.

Druhou iteraci hodnotím za sebe opět kladně, jelikož splnila své cíle. Studenti i učitelé druhou iteraci hodnotili také pozitivně a systém tak nadále naplňuje svou vizi pomáhat studentům k lepšímu pochopení látky, učitelům k jednodušší přípravě cvičení a snadnější komunikaci se studenty.

Závěr

Ve své práci jsem se zabýval tvorbou webového portálu pro podporu výuky programování se zaměřením na potřeby předmětů PA1 a PA2.

Nejprve jsem provedl analýzu výuky programování na středních a vysokých školách, ze které jsem odhalil hlavní problémy a stanovil funkční a nefunkční požadavky na systém, který by tyto problémy pomohl řešit. Při následné analýze jsem zjistil, že žádný ze stávajících systémů pro podporu výuky tyto požadavky nesplňuje. Navrhl jsem proto systém tvořený z MySQL databáze, backendu s použitím technologie Spring Web v programovacím jazyce Kotlin a frontendu ve frameworku Vue.js v programovacím jazyce JavaScript. Technologie byly zvoleny tak, aby na ně mohli v budoucnu navázat další studenti v rámci svých studentských projektů.

Navržený systém jsem implementoval ve dvou iteracích. V průběhu první iterace jsem navrhl prvotní řešení lokální kompilace v prohlížeči, anonymní mód a modulární architekturu. Toto řešení jsem nasadil na školní infrastrukturu, kde jsem ho uživatelsky otestoval. Na základě výstupů testování jsem v druhé iteraci provedl kompletní redesign uživatelského rozhraní včetně dalších vylepšení, jako přehlednější výsledky testů, podpora kompilace více souborů a překlad do angličtiny. Systém jsem otestoval pomocí jednotkových, integračních i uživatelských testů a výsledné řešení jsem zdokumentoval jak pro programátory, tak pro uživatele.

Za první dva semestry systém použilo 768 studentů, kteří v 16 paralelkách řešili celkem 350 úloh. V dotaznících, které jsem studentům a učitelům rozeslal během poloviny zimního i letního semestru byl systém hodnocen průměrně 8.18 / 10 (1 = nejhorší, 10 = nejlepší), mnoha pozitivními komentáři a přínos systému hodnotili studenti známkou 7.68 / 10. Vzhledem k pozitivním reakcím bude v následujících semestrech práce na systému pokračovat prostřednictvím týmových projektů i závěrečných prací.

Možná vylepšení

V průběhu implementace první i druhé iterace přišlo od studentů i učitelů mnoho podnětů a návrhů na další funkce, které v systému chybí. Zároveň jsem nejen z časových důvodů nestihl implementovat všechny nice to have požadavky z analýzy. V následujícím seznamu uvádím možná budoucí vylepšení podle jejich priority.

Nejvyšší priortita

Do této kategorie spadá Správa kurzů, Správa modulů a Kategorizace modulů. Tato vylepšení měla tak vysokou prioritu, že na nich začal pracovat tým v předmětu SP1 už v letním semestru, souběžně s vývojem diplomové práce. Plánované nasazení těchto změn je začátek zkouškového období.

Správa kurzů: aktuálně správu kurzů provádí vedoucí práce přímo v databázi. Na začátku semestru zjistí, kteří učitelé plánují systém využívat, a pro každého učitele vytvoří „pískoviště“ (soukromý kurz pro zkoušení úloh) a kurzy pro paralelky, které učí. Řešením je vytvoření role administrátora systému, který bude moct kurzy vytvářet přímo v uživatelském rozhraní, případně role garanta předmětu, kdy garant bude moct vytvářet libovolně kurzy pro daný předmět.

Správa modulů: při přidávání modulů do lekce se učiteli zobrazí dialogové okno, do kterého se načtou všechny moduly z databáze a filtrování probíhá čistě na straně frontendu podle názvu modulu a autora. Aktuálně jsou v systému stovky modulů, do budoucna ale tento přístup není udržitelný a pro vyučující není takové přidávání modulů přehledné.

Kategorizace modulů: souvisí se správou modulů. Aktuálně neexistuje způsob, jak dělit (a následně filtrovat) moduly. Možným řešením by bylo přidání štítků, podle kterých by šlo následně moduly dělit a také lépe popsat.

Další vylepšení

Následující vylepšení už nejsou nezbytně nutné, ale uživatelům zpříjemní práci se systémem, případně do něj přináší nové funkcionality.

Zjednodušení uživatelského rozhraní: na základě závěrečného dotazníku mezi studenty a rozhovorů s učiteli vyplývá, že uživatelské rozhraní sice obsahuje všechny potřebné obrazovky, je ale kvůli velkému množství prvků nepřehledné. Jednotlivé obrazovky by tedy potřebovaly zjednodušit a zpřehlednit.

Synchronizace s KOSapi: aktuálně provádí import studentů do jednotlivých paralelek učitel s pomocí manuálního exportu z KOSu. S využitím KOSapi by mohl systém zjistit, které předměty má v daném semestru student zapsané, a přímo ho do předmětu přidat.

Synchronizace kurzů s Gitlabem: jednotlivé kurzy by nově šlo definovat v Git repozitáři. Učitelé by tak mohli jednotlivé texty k lekcím, výchozí kódy i asserty nahrát přímo do Gitlab repozitáře, ze kterého by se úlohy synchronizovaly do systému. Úlohy by tak mohli chystat přímo ve vývojovém prostředí bez nutnosti kopírování kódů.

Vizualizace studentova kódu: přidání nástroje, který by umožnil vizualizovat např. závislosti v Makefile, #include direktivách, nebo paměť programu.

Metriky kvality studentova kódu: systém by mohl provádět analýzu studentova kódu a uvést metriky jako cyklomatickou složitost, počet řádek kódu a jednotlivých funkcí nebo počet funkcí, podobně jako to dělá třeba Progtest.

Lepší code review: aktuálně probíhá komunikace studenta s učitelem pouze přes textové pole u žádosti / odevzdání. Systém by mohl umožnit psaní komentářů přímo k jednotlivým řádkům kódu, nebo zobrazení historie komunikace.

Grafický výstup programu: jelikož je kód kompilován do WASM, je zde možnost využít funkce prohlížeče, jako je třeba kreslení na canvas. C a C++ knihovnu by šlo rozšířit o vhodné funkce, které by umožnily grafický výstup programu.

Interaktivita: po spuštění kódu student nemá žádnou možnost interakce s výsledným programem. Kromě výsledků testů by tak systém mohl umožnit i interaktivní spuštění se zadáváním vstupů přímo od uživatele.

Optimalizace a refactoring backendu: některé funkce na backendu provádí složité SQL dotazy, které v budoucnu mohou vést k dlouhé prodlevě u požadavků. Tyto funkce by šlo optimalizovat tak, aby pracovaly efektivně i pro velká data.

Podpora pro další programovací jazyky: systém má potenciál použití i v dalších předmětech, ve kterých se vyučují jiné programovací jazyky. Pro jejich podporu by ale bylo třeba zprovoznit kompilaci do WASM (která už pro některé jazyky existuje).

Bibliografie

1. ČT24. *Do škol přichází „revoluce“ v informatice. Word už stačit nebude, žáci mají umět pracovat s daty i programovat* [online]. 2021-02-15. [cit. 2024-02-27]. Dostupné z: https://ct24.ceskatelevize.cz/clanek/domaci/do-skol-prichazi-revoluce-v-informatice-word-uz-stacit-nebude-zaci-maji-umet-pracovat-s-daty-i-progr-38035.
2. VYSTAVĚL, Radek. *Moderní výuka programování* [online]. 2008-09-25. [cit. 2024-02-27]. Dostupné z: https://clanky.rvp.cz/clanek/2646/MODERNI-VYUKA-PROGRAMOVANI.html.
3. *Gymnázium Nad Kavalírkou* [online]. [cit. 2024-04-23]. Dostupné z: https://kavalirka.cz.
4. *Programování a algoritmizace 1* [online]. Bílá kniha [cit. 2024-04-23]. Dostupné z: https://bk.fit.cvut.cz/cz/predmety/00/00/00/00/00/00/06/53/35/p6533506.html.
5. *Programování a algoritmizace 2* [online]. Bílá kniha [cit. 2024-04-23]. Dostupné z: https://bk.fit.cvut.cz/cz/predmety/00/00/00/00/00/00/06/69/20/p6692006.html.
6. *Anketa ČVUT* [online]. [cit. 2024-04-23]. Dostupné z: https://anketa.cvut.cz.
7. *Formuláře Google: Nástroj pro tvorbu online formulářů* [online]. Google Workspace, 2024 [cit. 2024-04-23]. Dostupné z: https://www.google.com/forms/about/.
8. *Neoficiální Discord server FIT ČVUT v Praze* [online]. [cit. 2024-04-23]. Dostupné z: http://discord.fit.cvut.cz.
9. NOLAN, Beatrice. *Why the 'Sludge Content' TikTok Trend has Gen Z hooked* [online]. Business Insider, 2023-05-28 [cit. 2024-02-27]. Dostupné z: https://www.businessinsider.com/sludge-content-gen-z-hooked-tiktok-2023-4.
10. MICROSOFT TEAMS. *Videokonference, schůzky, volání* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-teams/group-chat-software/.
11. MOODLE PTY LTD. *Titulní stránka* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://moodle.org.

12. *ProgTest* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://progtest.fit.cvut.cz.
13. LEETCODE. *The World's Leading Online Programming Learning Platform* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://leetcode.com.
14. UDEMY. *Online Courses - Learn Anything, On Your Schedule* [online]. 2024. [cit. 2024-04-23]. Dostupné z: https://www.udemy.com.
15. MATHIGON. *The Mathematical Playground* [online]. 2023. [cit. 2024-04-23]. Dostupné z: https://mathigon.org.
16. FUNCTIONAL SOFTWARE, INC. *Application Performance Monitoring & Error Tracking Software* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://sentry.io.
17. MOZILLA FOUNDATION. *Using HTTP cookies – HTTP* [online]. MDN web docs, 2024-02-23 [cit. 2024-03-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.
18. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL: The world's most advanced open source database* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://www.postgresql.org/.
19. MARIADB FOUNDATION. *MariaDB Server: The open source relational database* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://mariadb.org/.
20. ORACLE. *MySQL* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://www.mysql.com/.
21. ORACLE. *Database* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://www.oracle.com/database/.
22. MICROSOFT. *SQL Server 2022 — Microsoft* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://www.microsoft.com/en-us/sql-server/sql-server-2022.
23. ALTEXSOFT. *Comparing SOAP vs REST vs GraphQL vs RPC API* [online]. 2020-05-29. [cit. 2024-04-24]. Dostupné z: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/.
24. THE GRAPHQL FOUNDATION. *GraphQL — A query language for your API* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://graphql.org/.
25. THE PHP GROUP. *PHP: Hypertext Preprocessor* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://www.php.net/.
26. THE APACHE SOFTWARE FOUNDATION. *The Apache HTTP Server Project* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://httpd.apache.org.
27. NGINX, INC. *Nginx* [online]. [cit. 2024-04-24]. Dostupné z: https://nginx.org/.
28. LOCKHART, Josh; SMITH, Andrew; ALLEN, Rob; BÉRUBÉ, Pierre et al. *Slim Framework* [online]. [cit. 2024-04-24]. Dostupné z: https://www.slimframework.com.
29. GRUDL, David. *Nette - pohodlný a bezpečný vývoj webových aplikací v PHP* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://nette.org.

30. *Symfony, High Performance PHP Framework for Web Development* [online]. [cit. 2024-04-24]. Dostupné z: https://symfony.com.

31. LARAVEL HOLDINGS INC. *Laravel - The PHP Framework For Web Artisans* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://laravel.com.

32. MICROSOFT. *A tour of C# - Overview* [online]. Microsoft Learn, 2024 [cit. 2024-04-24]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

33. ORACLE. *What is Java and why do I need it?* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://www.java.com/en/download/help/whatis_java.html.

34. BROADCOM, INC. *Spring — Web Applications* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://spring.io/web-applications.

35. JETBRAINS. *Kotlin Programming Language* [online]. [cit. 2024-04-24]. Dostupné z: https://kotlinlang.org/.

36. JETBRAINS. *Ktor: Build Asynchronous Servers and Clients in Kotlin* [online]. Ktor Framework [cit. 2024-04-24]. Dostupné z: https://ktor.io.

37. MOZILLA FOUNDATION. *JavaScript* [online]. MDN web docs, 2024-03-05 [cit. 2024-04-24]. Dostupné z: https://developer.mozilla.org/en/JavaScript.

38. MICROSOFT. *TypeScript: JavaScript With Syntax For Types* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://www.typescriptlang.org.

39. GOOGLE. *Angular* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://angular.io.

40. TATVASOFT SOFTWARE DEVELOPMENT COMPANY. *Angular Vs React Vs Vue: Which One To Choose* [online]. 2024-01-30. [cit. 2024-04-25]. Dostupné z: https://www.tatvasoft.com/blog/angular-vs-react-vs-vue/.

41. META OPEN SOURCE. *React* [online]. 2024. [cit. 2024-04-24]. Dostupné z: https://react.dev.

42. YOU, Evan. *Vue.js - The Progressive JavaScript Framework* [online]. 2024. [cit. 2024-04-25]. Dostupné z: https://vuejs.org.

43. MOZILLA FOUNDATION. *WebAssembly* [online]. MDN web docs, 2024-04-16 [cit. 2024-04-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/WebAssembly.

44. WRZECIONKO, Ondřej. *Zpěvník pro náboženská shromáždění – mobilní aplikace*. 2022. České vysoké učení technické v Praze, Fakulta informačních technologií.

45. BROADCOM, INC. *Spring Data JPA* [online]. 2024. [cit. 2024-04-26]. Dostupné z: https://spring.io/projects/spring-data-jpa.

46. BAELDUNG. *Spring Data JPA @Query — Baeldung* [online]. 2024-04-12. [cit. 2024-04-26]. Dostupné z: https://www.baeldung.com/spring-data-jpa-query.

47. FUNCTIONAL SOFTWARE, INC. *Sentry for Spring Boot* [online]. [cit. 2024-04-26]. Dostupné z: `https://docs.sentry.io/platforms/java/guides/spring-boot/`.

48. FIGMA, INC. *Figma: The Collaborative Interface Design Tool* [online]. [cit. 2024-04-28]. Dostupné z: `https://www.figma.com`.

49. VUETIFY. *A Vue Component Framework* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://vuetifyjs.com`.

50. PULSARDEV SRL. *Quasar Framework* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://quasar.dev`.

51. BOOTSTRAPVUE CORE TEAM. *BootstrapVue* [online]. [cit. 2024-04-28]. Dostupné z: `https://bootstrap-vue.org`.

52. GOOGLE. *Material Design* [online]. [cit. 2024-04-28]. Dostupné z: `https://m3.material.io`.

53. VUETIFY. *Official Vuetify 3 UI Kit Figma* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://store.vuetifyjs.com/products/vuetify-ui-kit-figma`.

54. YOU, Evan. *The official Router for Vue.js* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://router.vuejs.org`.

55. ZABRISKIE, Matt. *Axios* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://axios-http.com`.

56. MOROTE, Eduardo San Martin. *The intuitive store for Vue.js* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://pinia.vuejs.org`.

57. CONE, Matt. *Getting Started — Markdown Guide* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://www.markdownguide.org/getting-started/`.

58. IMZBF. *MdEditorV3 Documentation* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://imzbf.github.io/md-editor-v3/`.

59. FENGYUAN, Chen. *Compressor.js* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://imzbf.github.io/md-editor-v3/`.

60. EMSCRIPTEN CONTRIBUTORS. *Emscripten documentation* [online]. 2015. [cit. 2024-04-28]. Dostupné z: `https://emscripten.org`.

61. MATOUŠEK, Jan. *WASM artifacts* [online]. 2024. [cit. 2024-05-06]. Dostupné z: `https://gitlab.fit.cvut.cz/trainer-fit/wasm-artifacts`.

62. WASI SUBGROUP. *Introduction — WASI.dev* [online]. [cit. 2024-04-28]. Dostupné z: `https://wasi.dev`.

63. TAYLOR, Ben. *Runno* [online]. [cit. 2024-04-28]. Dostupné z: `https://runno.dev/wasi`.

64. MOZILLA FOUNDATION. *Using the Fetch API* [online]. MDN web docs, 2023-08-18 [cit. 2024-04-28]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch`.

65. DOCKER INC. *Accelerated Container Application Development* [online]. 2024. [cit. 2024-04-28]. Dostupné z: `https://www.docker.com/`.

66. MOZILLA FOUNDATION. *Window: crossOriginIsolated property* [online]. 2024-04-26. [cit. 2024-04-28]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/API/Window/crossOriginIsolated`.

67. NIELSEN, Jakob. *10 Usability Heuristics for User Interface Design* [online]. Nielsen Norman Group, 2024-01-30 [cit. 2024-03-22]. Dostupné z: `https://www.nngroup.com/articles/ten-usability-heuristics/`.

68. KOTEST TEAM. *Kotest* [online]. 2024. [cit. 2024-04-29]. Dostupné z: `https://kotest.io`.

69. *MockK — mocking library for Kotlin* [online]. [cit. 2024-04-29]. Dostupné z: `https://mockk.io`.

70. *Mockito framework site* [online]. [cit. 2024-04-29]. Dostupné z: `https://site.mockito.org`.

71. POSTMAN, INC. *Postman API Platform* [online]. 2024. [cit. 2024-04-29]. Dostupné z: `https://www.postman.com/product/what-is-postman/`.

72. SMARTBEAR SOFTWARE. *Swagger Editor* [online]. 2024. [cit. 2024-05-02]. Dostupné z: `https://swagger.io/tools/swagger-editor/`.

Příloha **A**

Seznam zkratek

AI umělá inteligence

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CD Continuos Deployment

CI Continuos Integration

CRUD Create Read Update Delete

ČVUT České vysoké učení technické v Praze

DOM Document Object Model

DSL Domain Specific Language

DTO Data Transfer Object

FEL Fakulta elektrotechnická

FIT Fakulta informačních technologií

GraphQL Graph Query Language

HATEOAS Hypertext As The Engine of Application State

HTML HyperText Markup Language

I/O vstup a výstup

ID identifikátor

JPA Java Persistence API

JPQL Java Persistence Query Language

JS JavaScript

JSON JavaScript Object Notation

JVM Java Virtual Machine

MUNI Masarykova univerzita

MVVM Model View ViewModel

NoSQL Not only SQL

NUR Návrh uživatelského rozhraní

OAuth OpenAuth

OSU Ostravská univerzita

PA1 Programování a algoritmizace 1

PA2 Programování a algoritmizace 2

REST Representational State Transfer

SOAP Simple Object Access Protocol

STL Standard Template Library

UI uživatelské rozhraní

URL Uniform Resource Locator

VUT Vysoké učení technické v Brně

VŠB Vysoká škola Báňská – Technická univerzita Ostrava

WASI Web Assembly System Interface

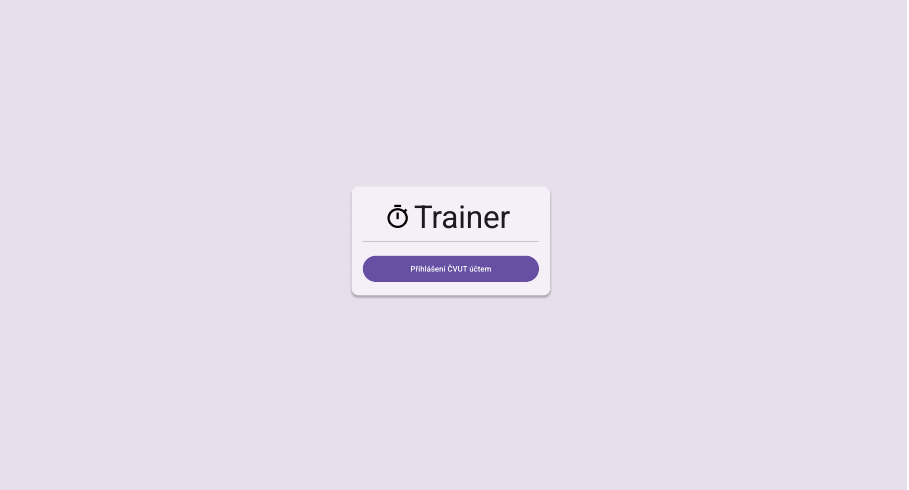
WASM Web Assembly

WSDL Web Service Description Language

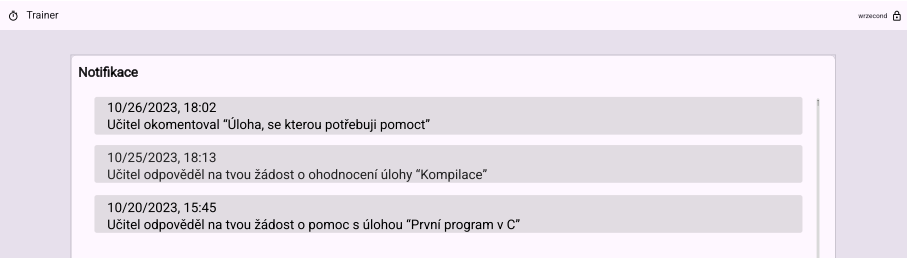
XML eXtended Markup Language

Příloha B

Návrh uživatelského rozhraní

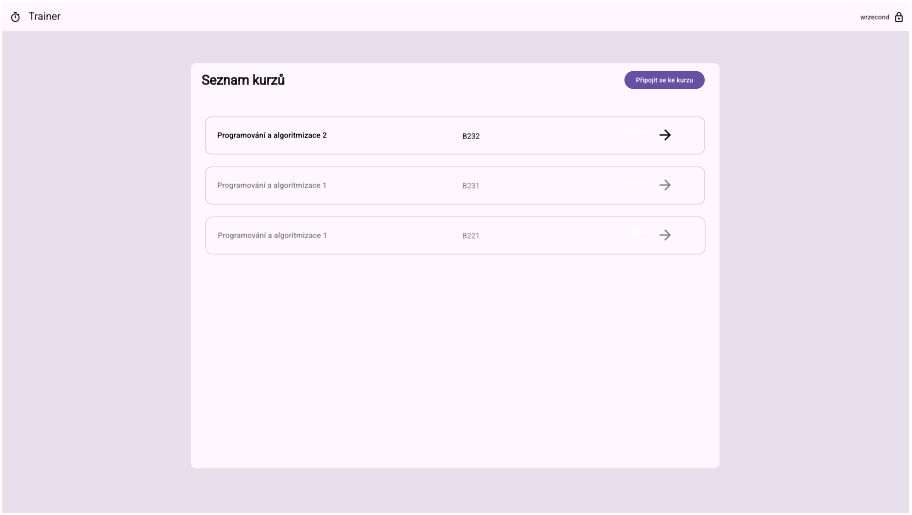


Obrázek B.1: Přihlašovací obrazovka

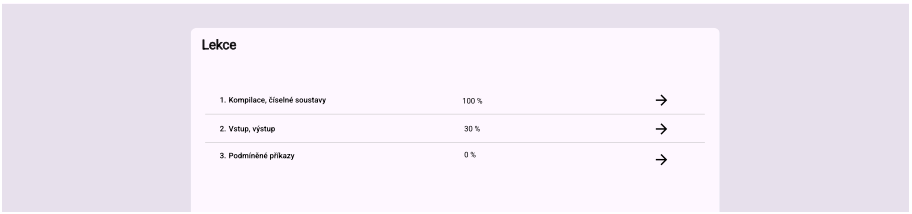


Obrázek B.2: Seznam notifikací

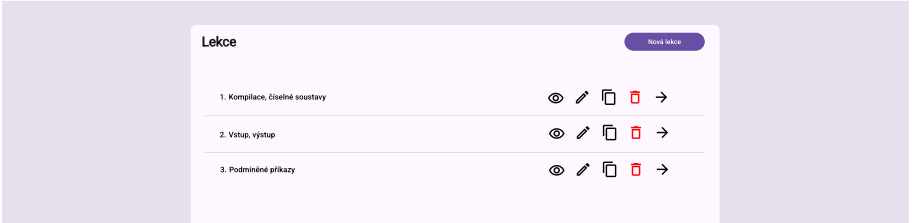
První iterace



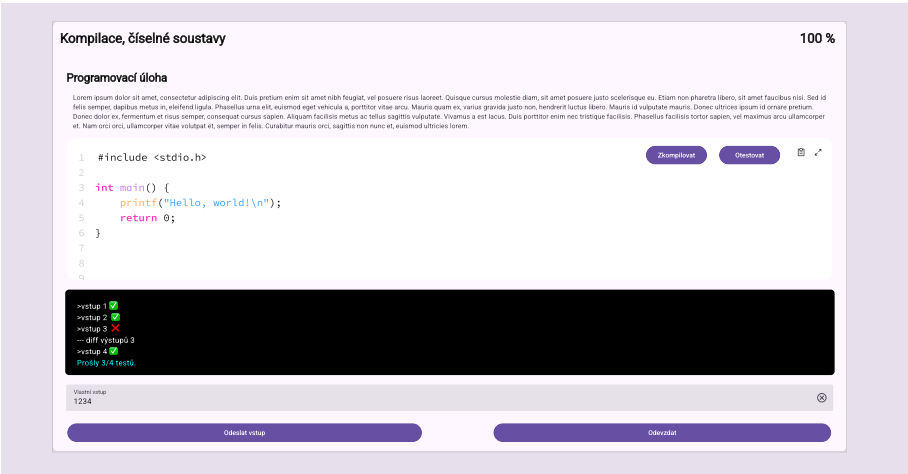
Obrázek B.3: Seznam kurzů



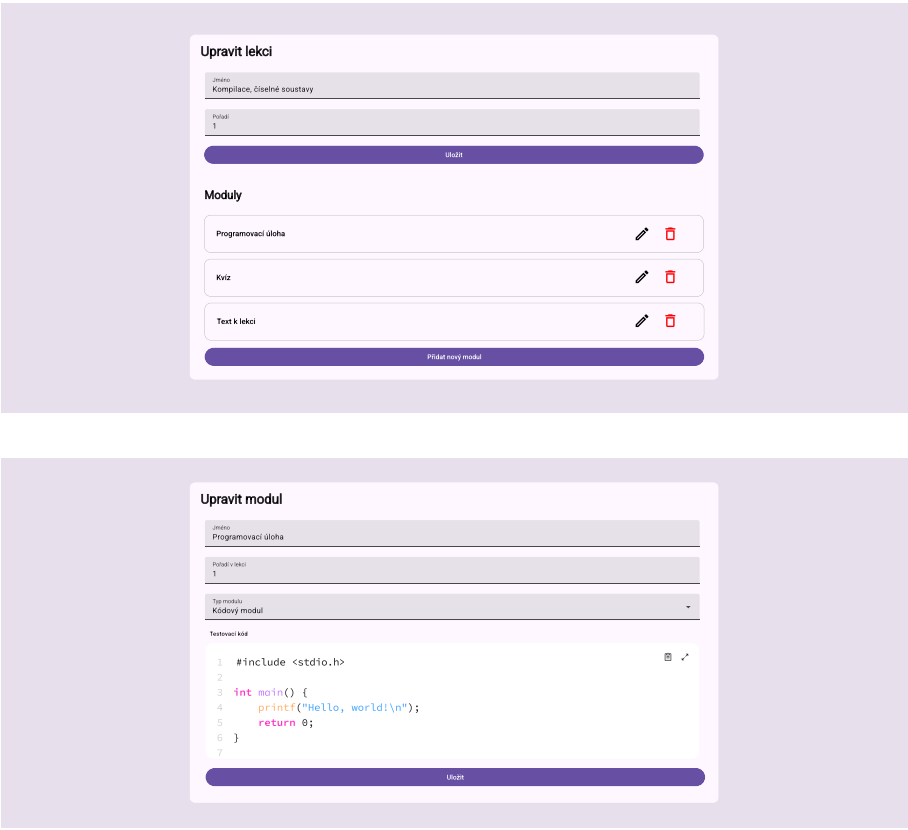
Obrázek B.4: Detail kurzu (student)



Obrázek B.5: Detail kurzu (učitel)

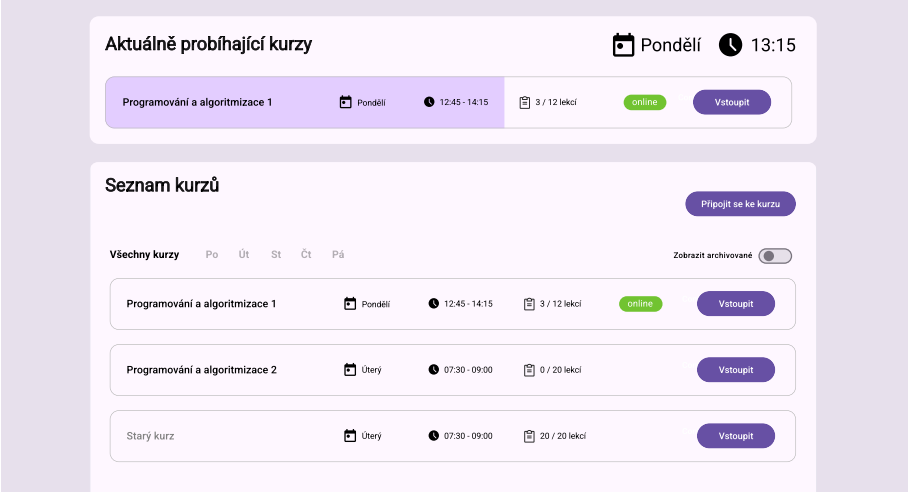


Obrázek B.6: Detail lekce

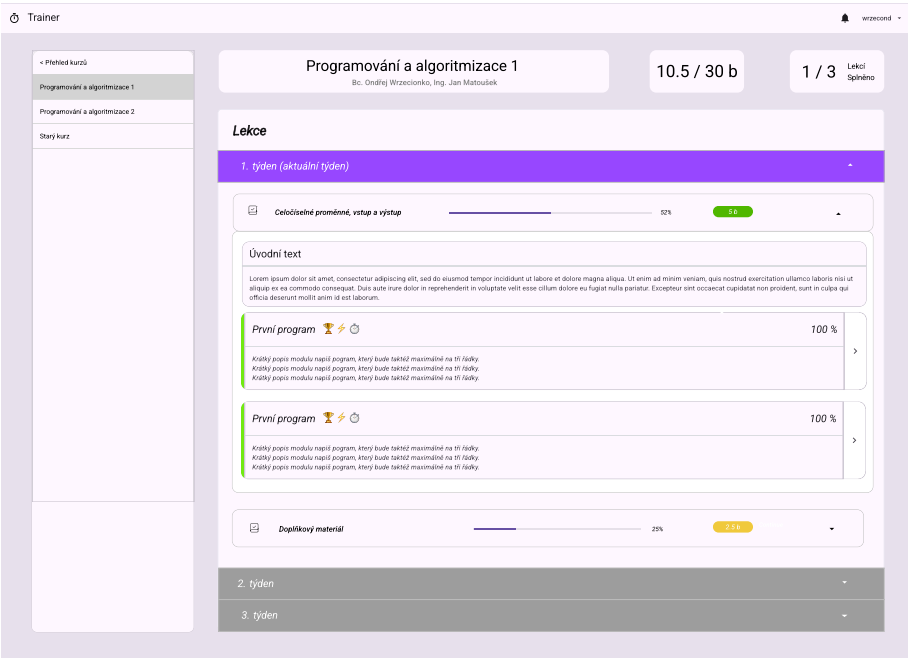


Obrázek B.7: Úprava lekce a úprava modulu

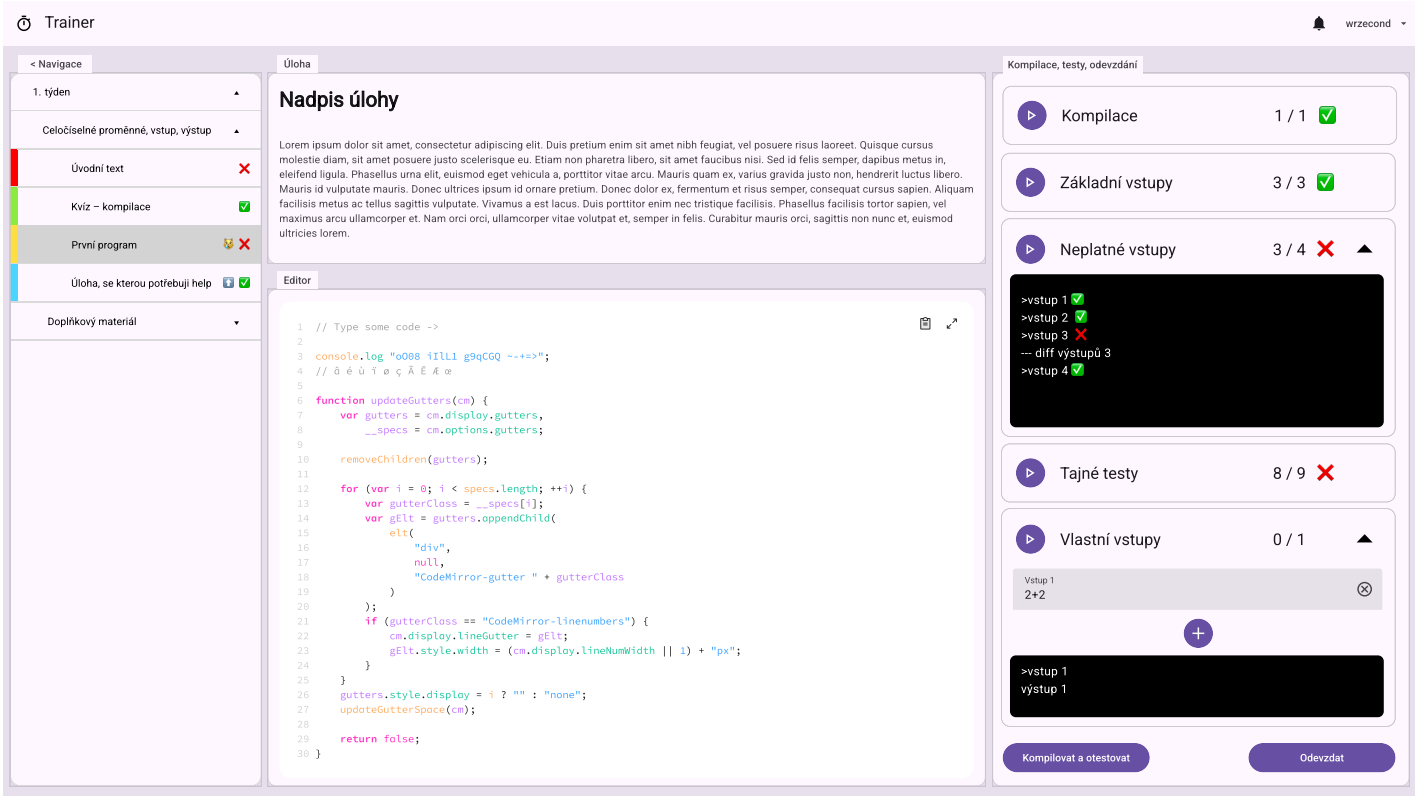
Druhá iterace



Obrázek B.8: Seznam kurzů



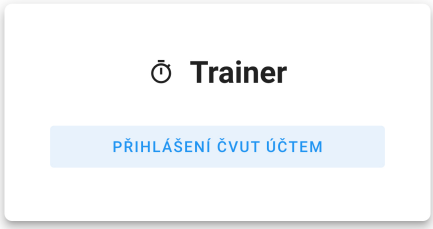
Obrázek B.9: Detail kurzu



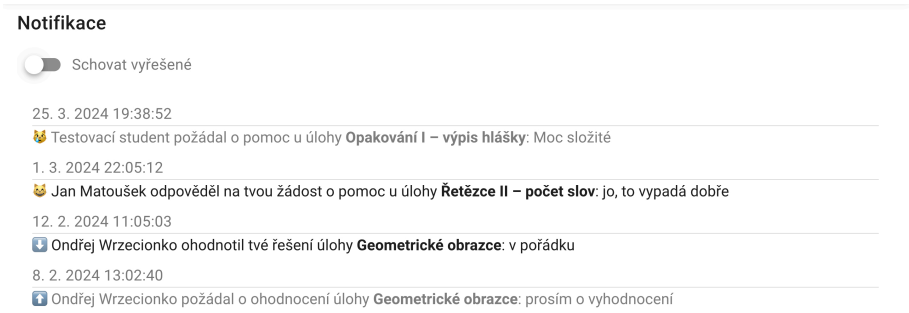
Obrázek B.10: Detail lekce

Příloha C

Uživatelské rozhraní

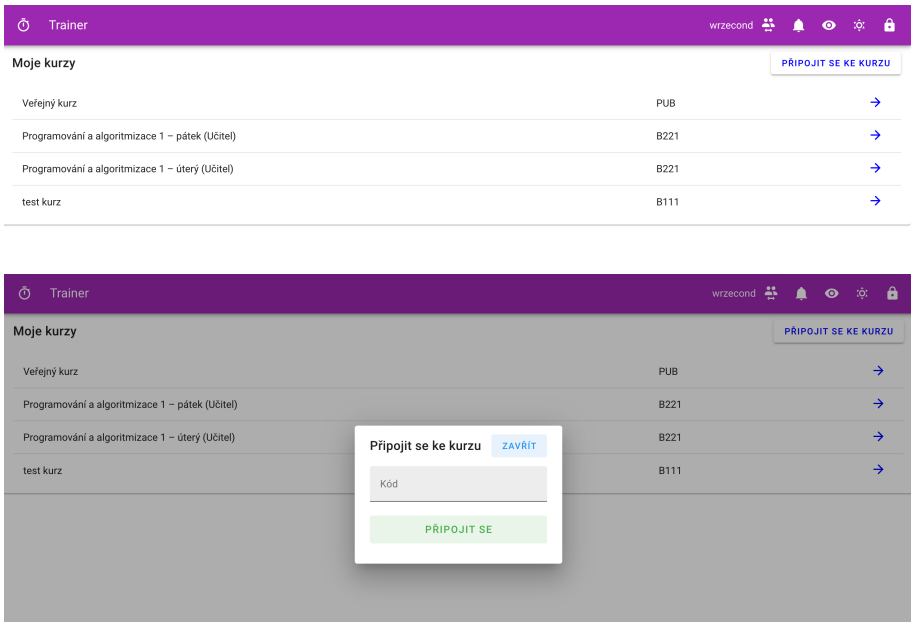


Obrázek C.1: Přihlašovací obrazovka

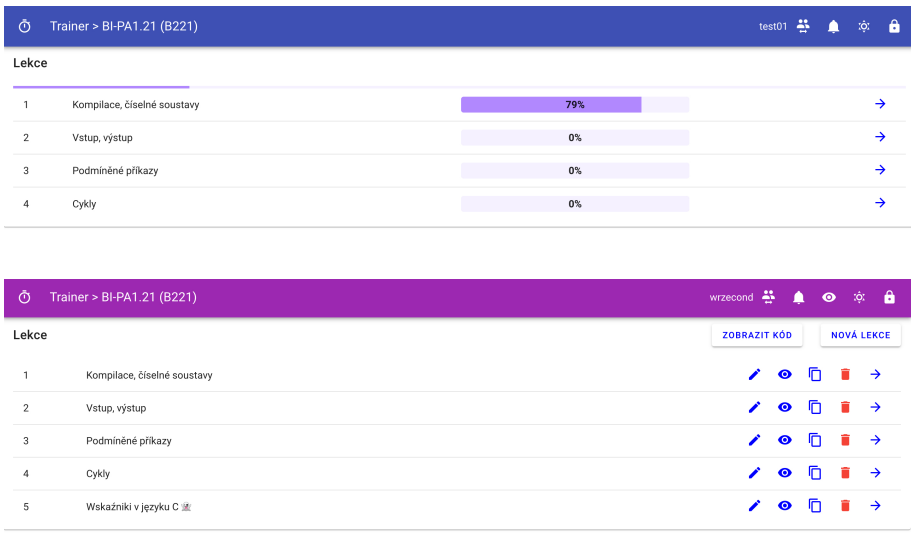


Obrázek C.2: Seznam notifikací

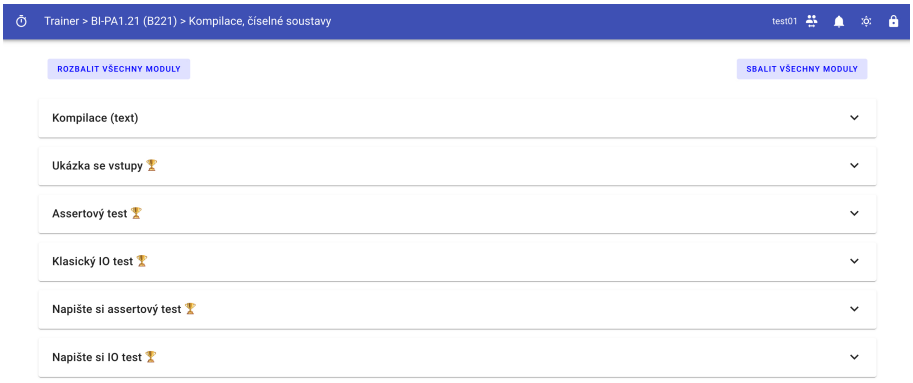
První iterace



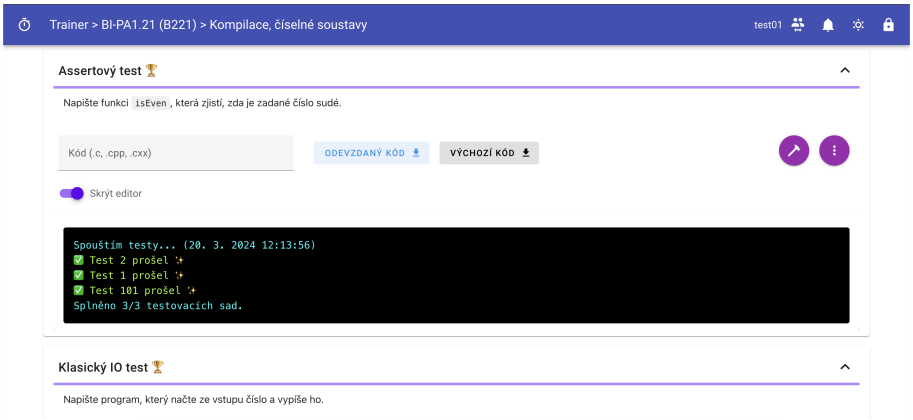
Obrázek C.3: Seznam kurzů, dialog pro připojení do nového kurzu



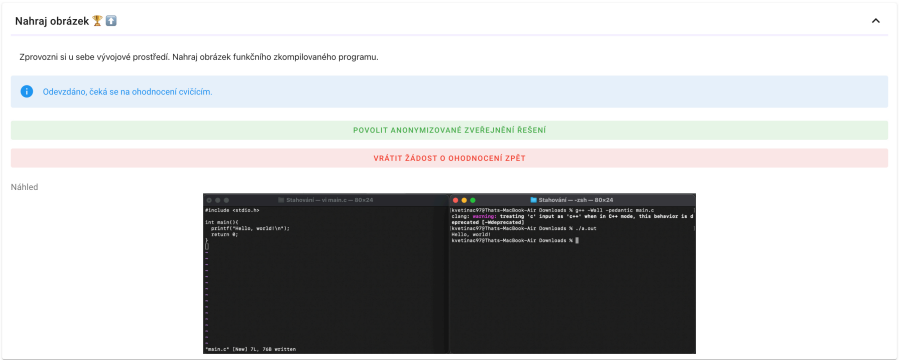
Obrázek C.4: Detail kurzu (nahoře student, dole učitel)



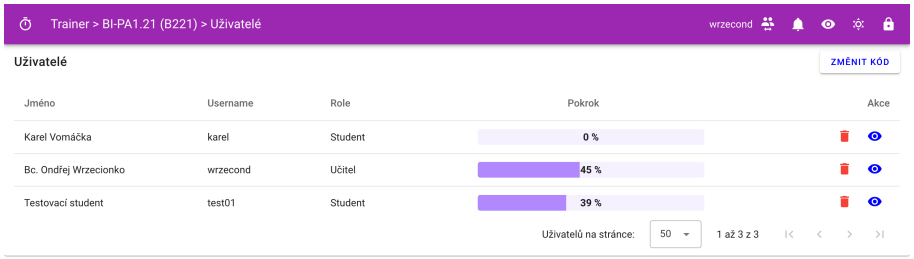
Obrázek C.5: Detail lekce – sbalené moduly



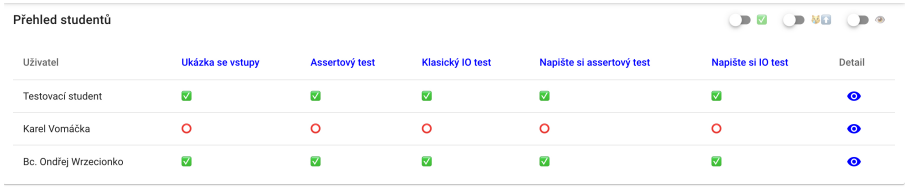
Obrázek C.6: Detail lekce – kódový modul



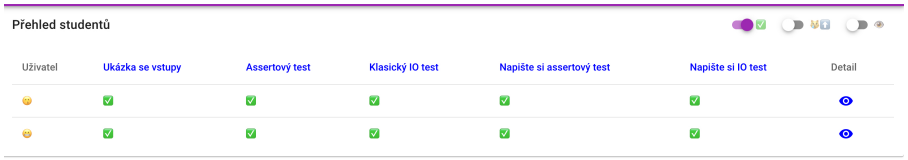
Obrázek C.7: Detail lekce – assignment modul



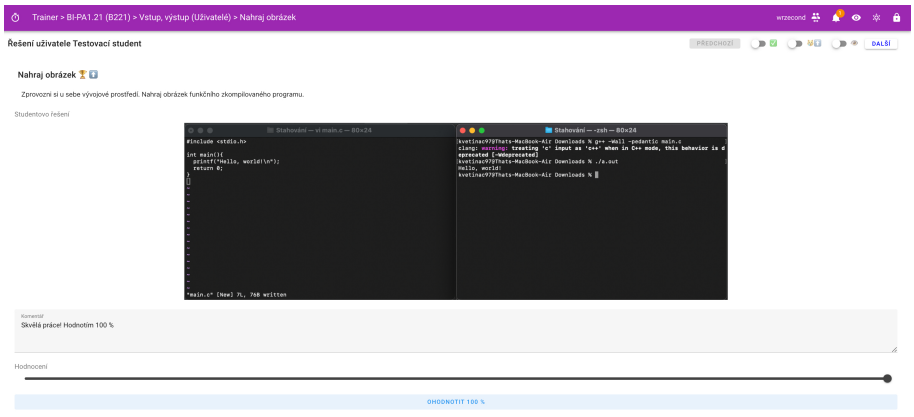
Obrázek C.8: Přehled uživatelů v kurzu



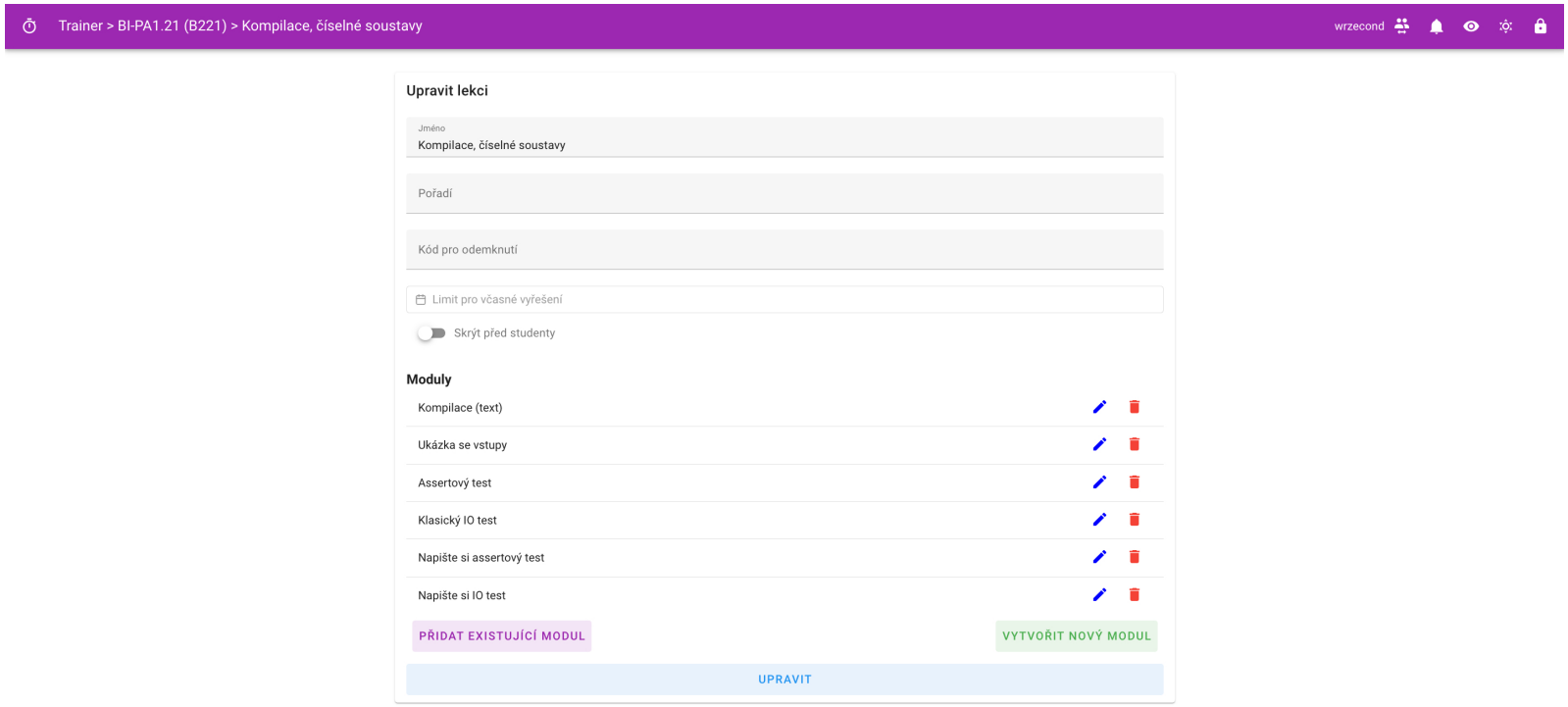
Obrázek C.9: Přehled studentů v lekci



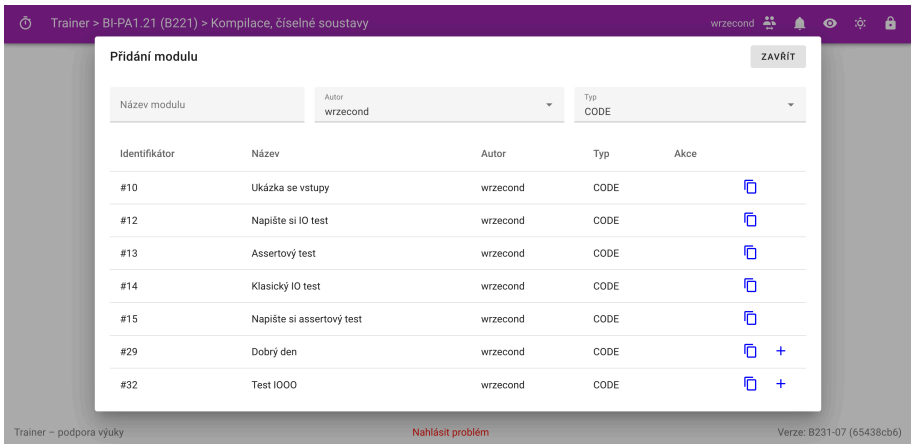
Obrázek C.10: Přehled studentů v lekci (anonymní mód, filtr)



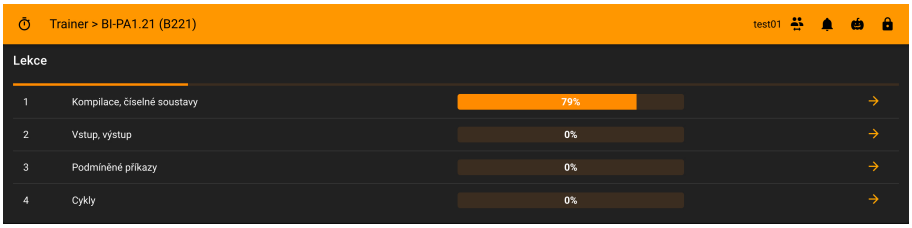
Obrázek C.11: Pohled na studentské řešení



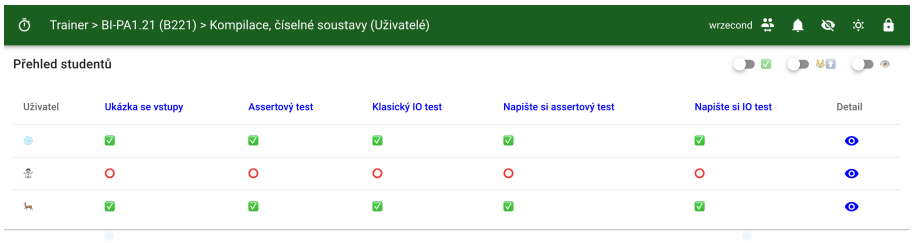
Obrázek C.12: Úprava lekce (učitel)



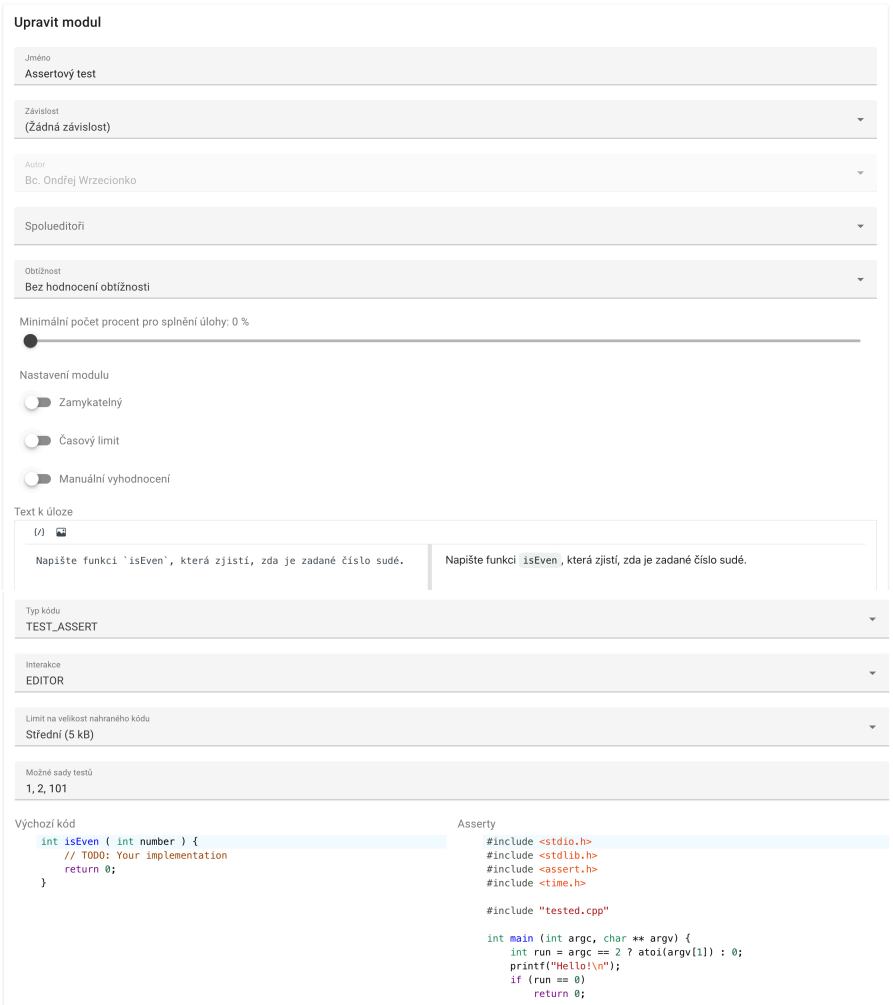
Obrázek C.13: Dialog pro přidání modulu do lekce (učitel)



Obrázek C.14: Detail kurzu (Halloweenský mód)

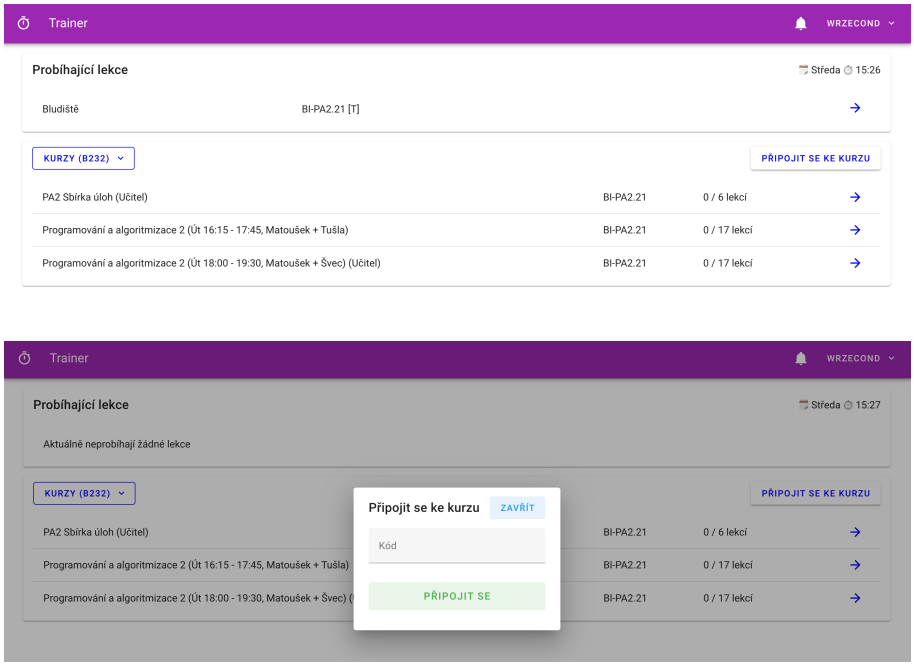


Obrázek C.15: Přehled studentů (Vánoční mód)

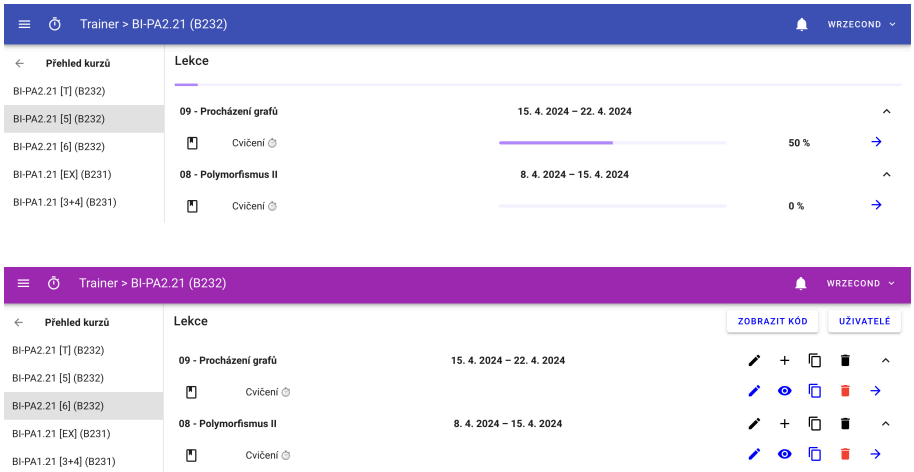


Obrázek C.16: Úprava modulu

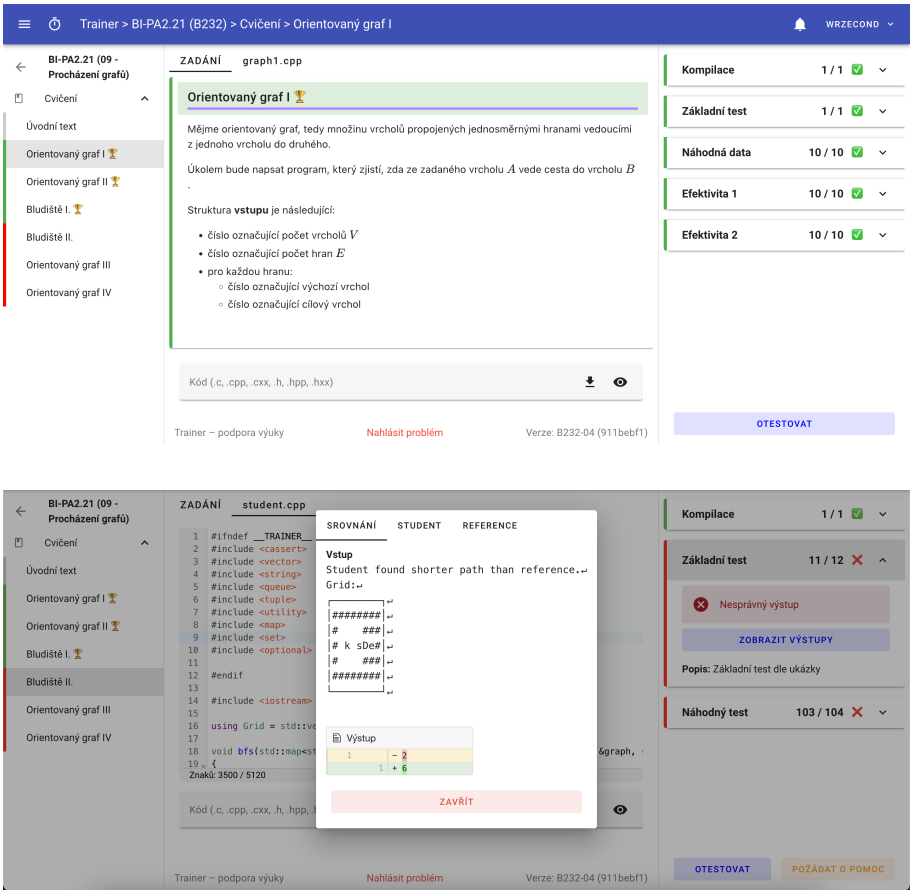
Druhá iterace



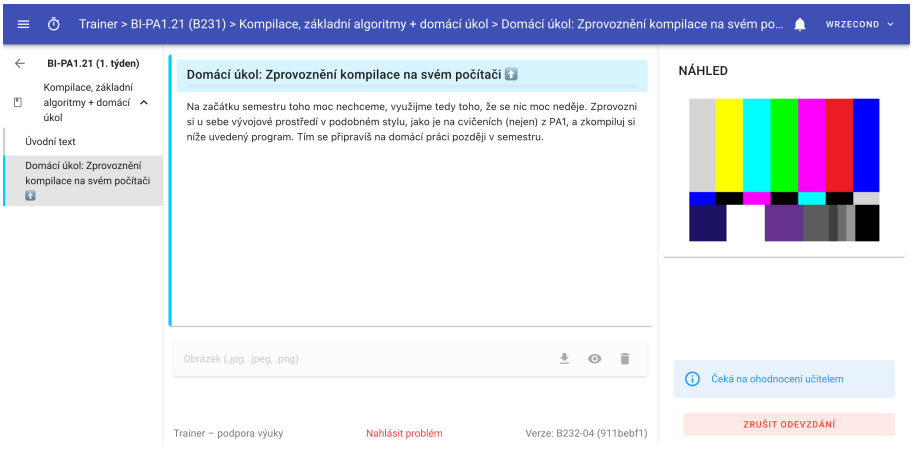
Obrázek C.17: Seznam kurzů, dialog pro připojení do nového kurzu



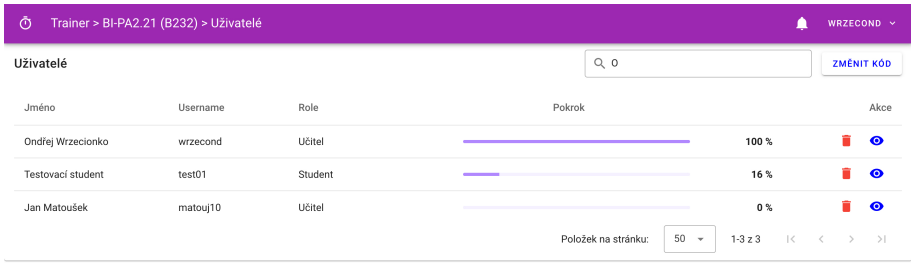
Obrázek C.18: Detail kurzu (nahoře student, dole učitel)



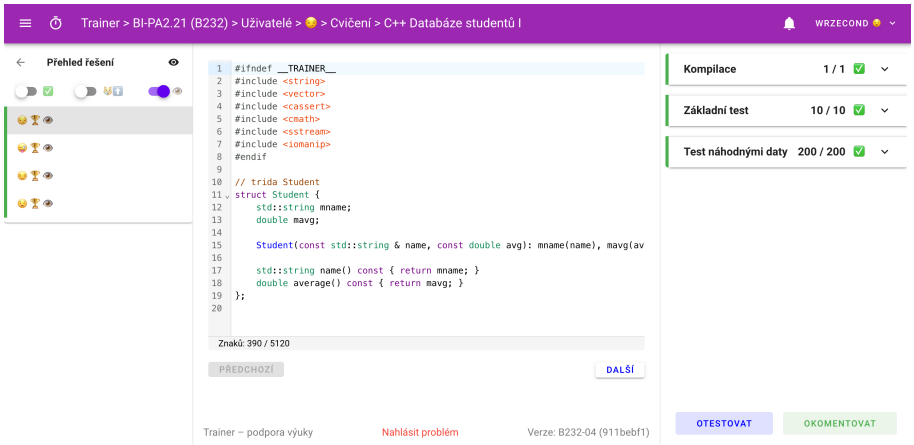
Obrázek C.19: Detail lekce – úspěšné a neúspěšné řešení



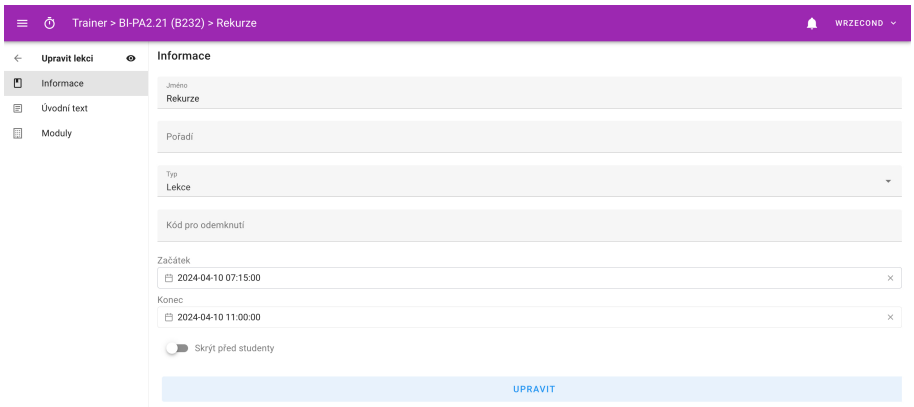
Obrázek C.20: Detail lekce – assignment modul



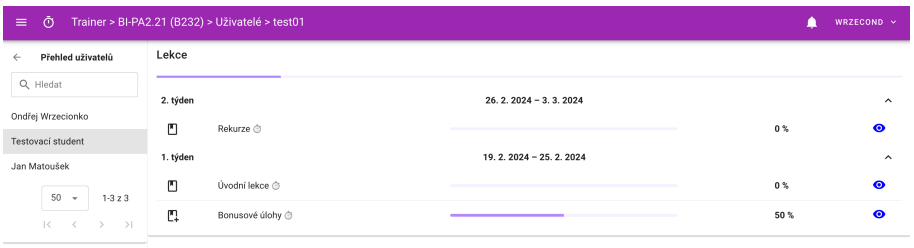
Obrázek C.21: Přehled uživatelů v kurzu



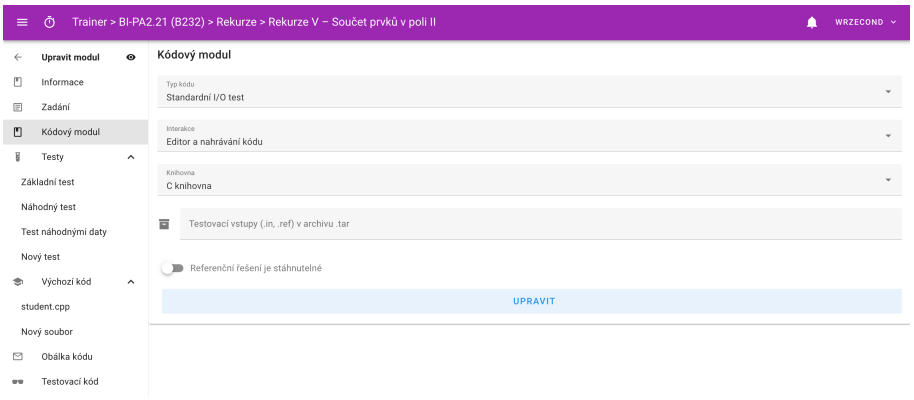
Obrázek C.22: Pohled na studentské řešení



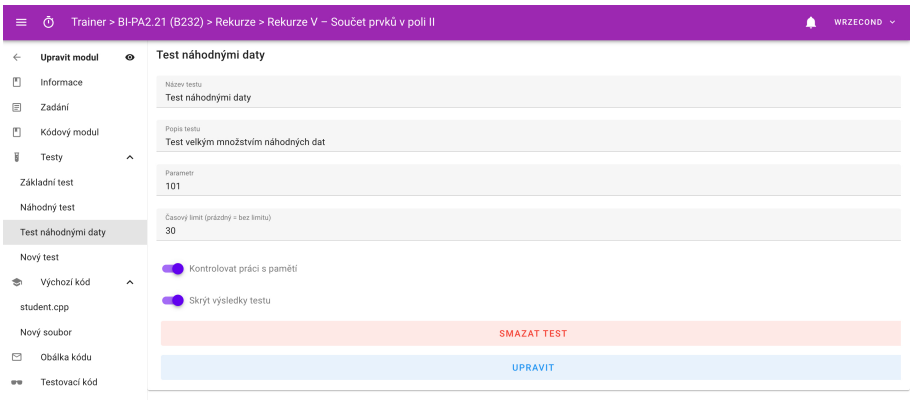
Obrázek C.23: Úprava lekce



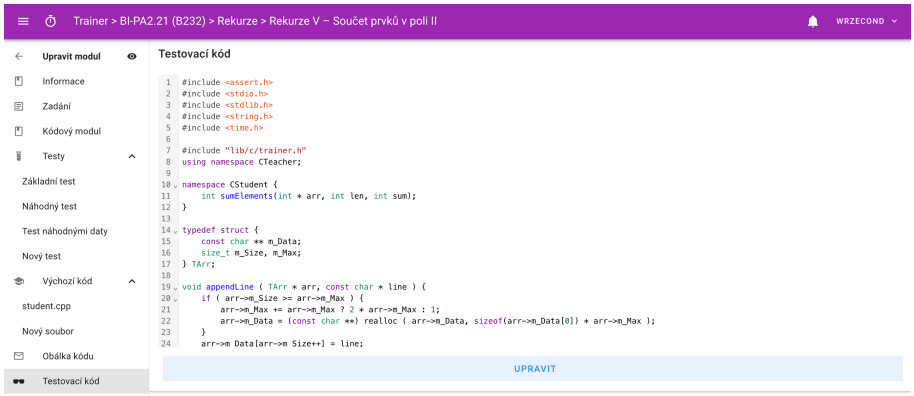
Obrázek C.24: Pohled na studenta v kurzu



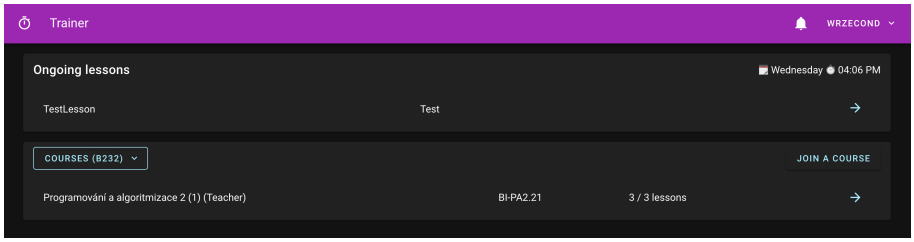
Obrázek C.25: Úprava kódového modulu – informace



Obrázek C.26: Úprava testu v kódovém modulu



Obrázek C.27: Úprava testovacího kódu v kódovém modulu

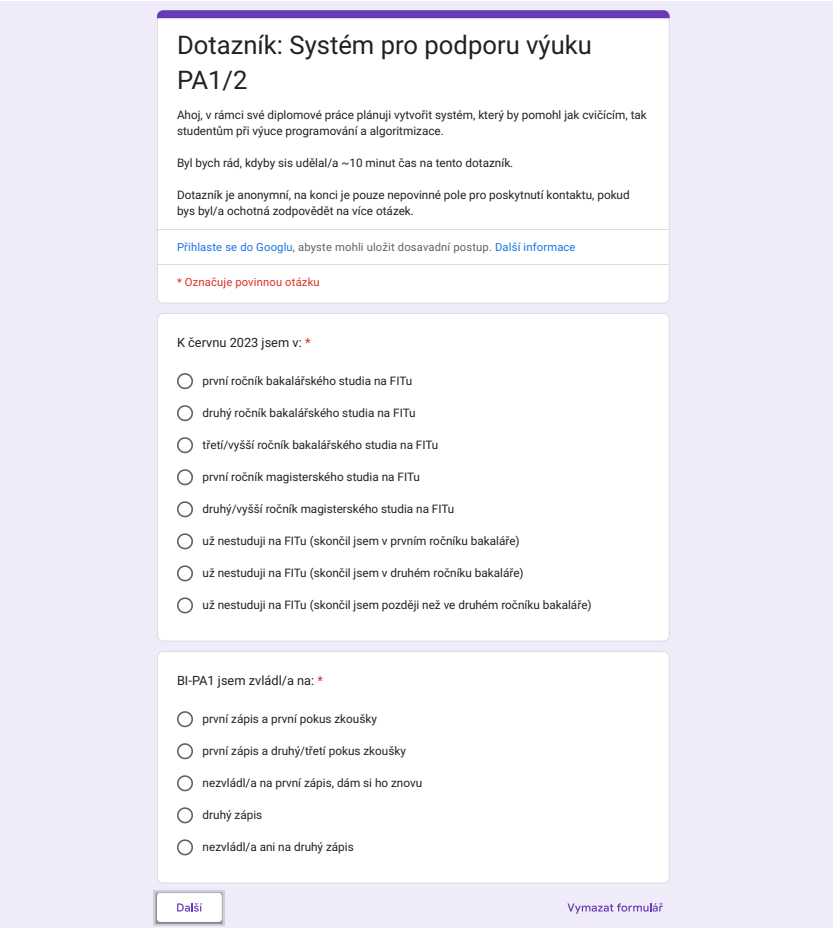


Obrázek C.28: Seznam kurzů v angličtině a tmavém režimu

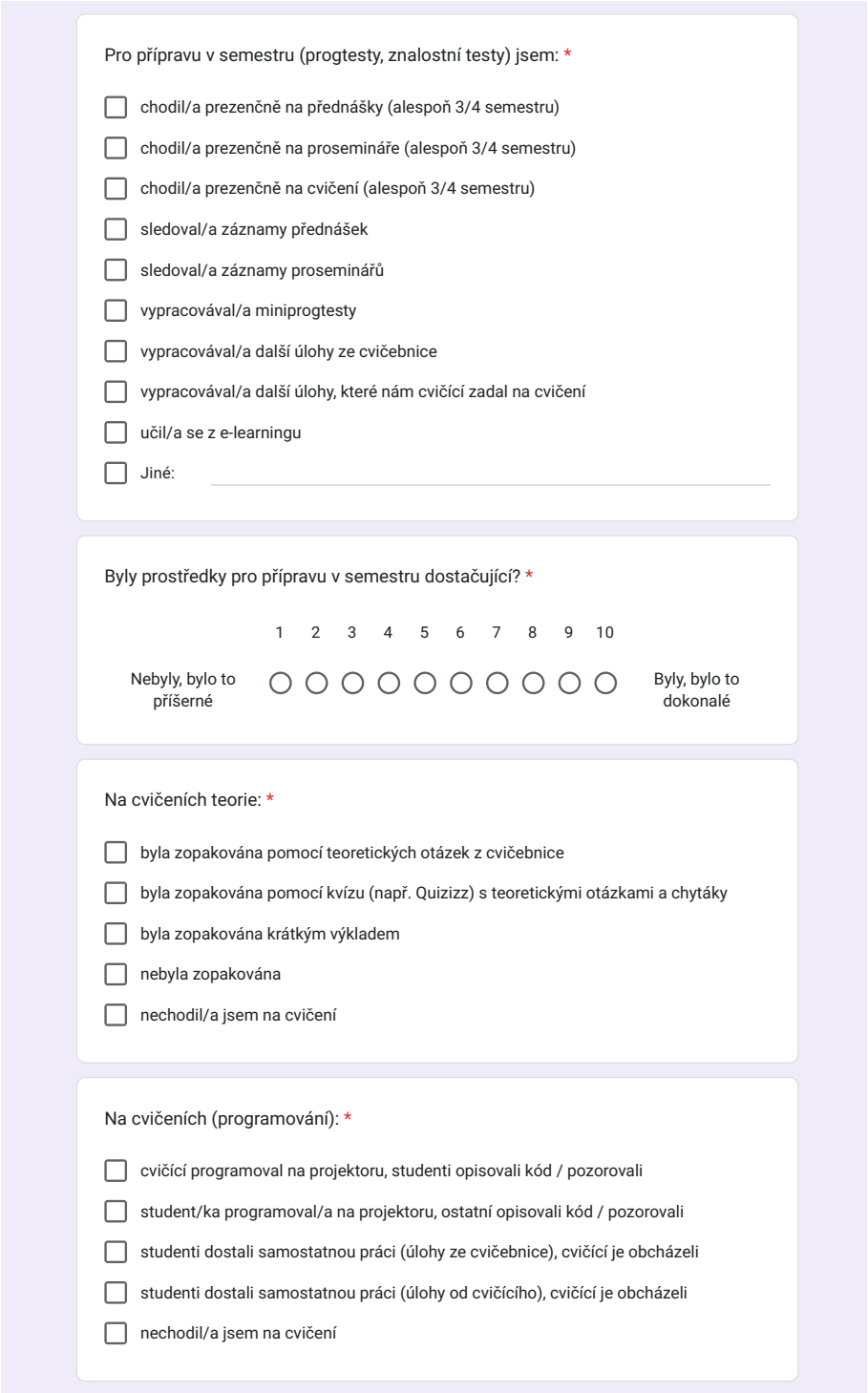
Příloha D

Dotazníky

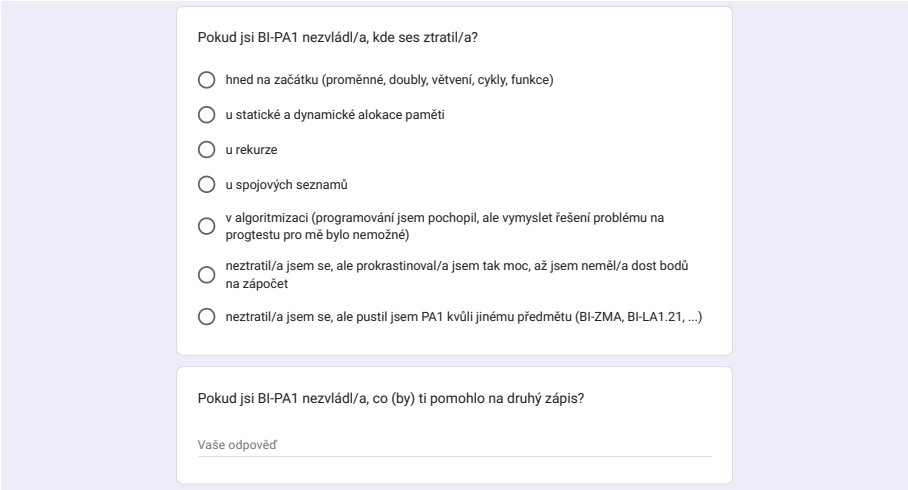
Studentský dotazník ohledně systému (červen 2023)



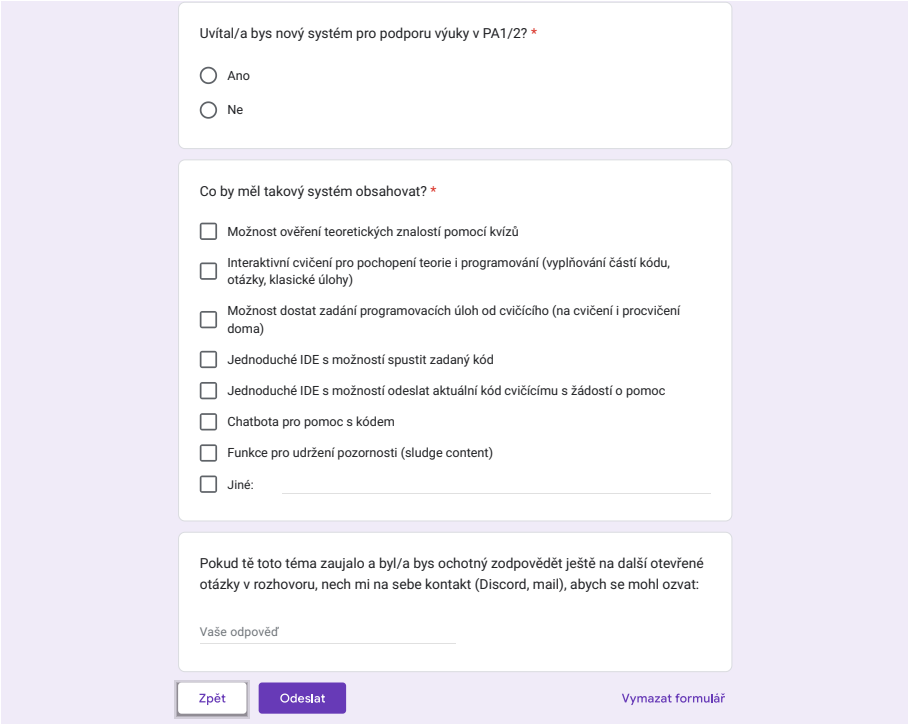
Obrázek D.1: Úvodní otázka o ročníku studia a úspěšnosti PA1



Obrázek D.2: Otázky o podobě přípravy v semestru a cvičeních

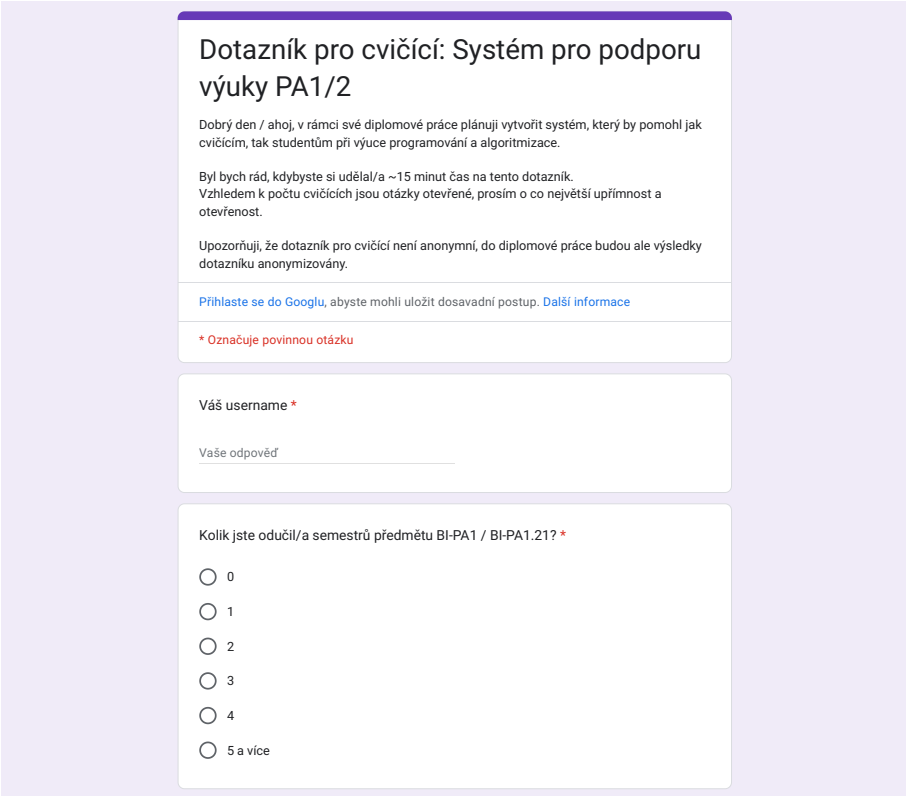


Obrázek D.3: Podmíněná sekce s otázkami, kde se student ztratil

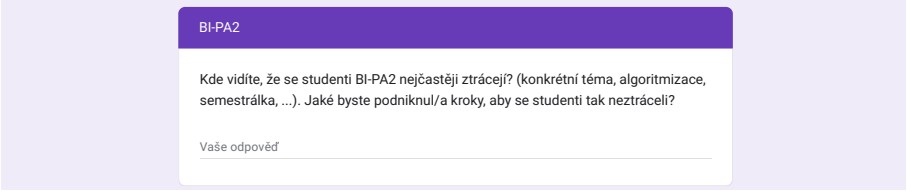


Obrázek D.4: Otázky týkající se nového systému

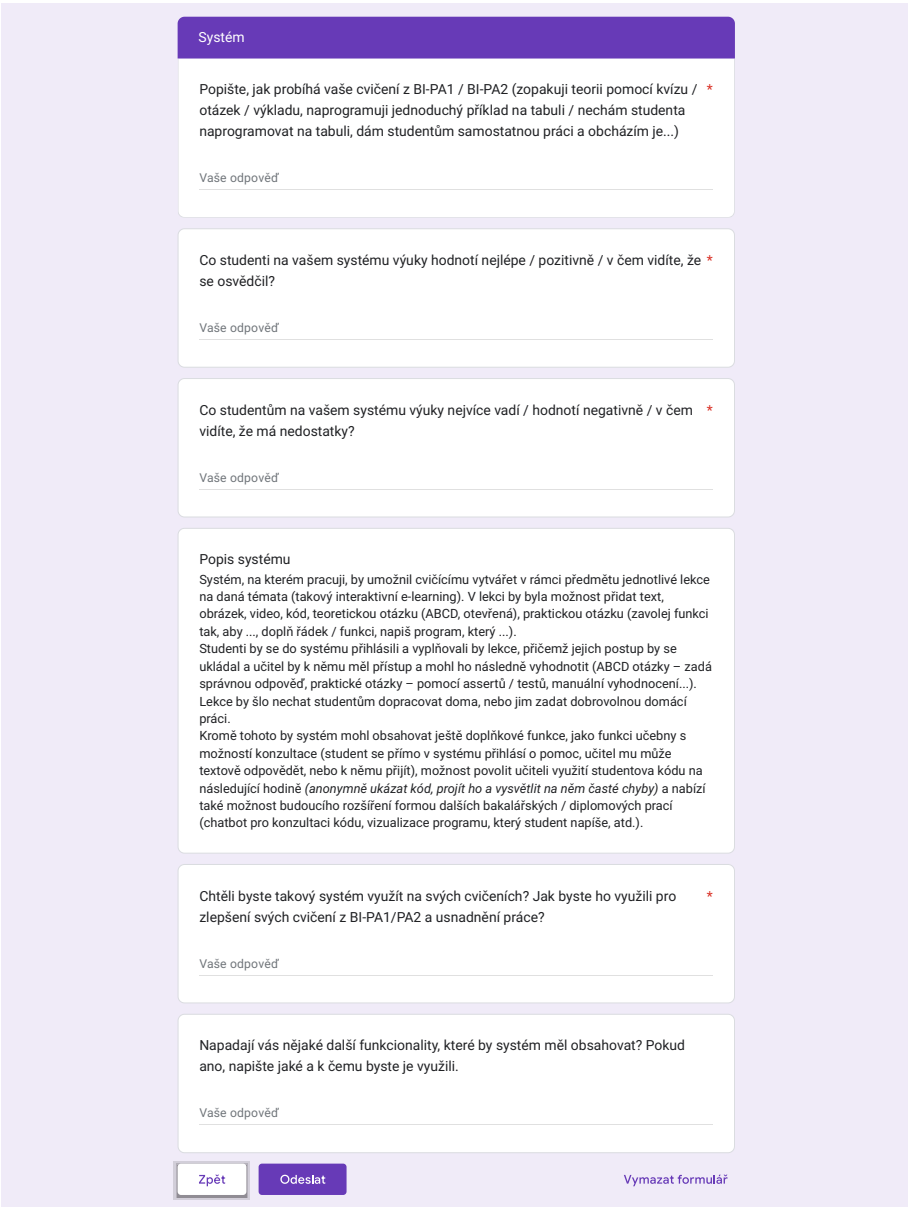
Učitelský dotazník ohledně systému (červen 2023)



Obrázek D.5: Úvodní otázka na jméno a počet odučených semestrů

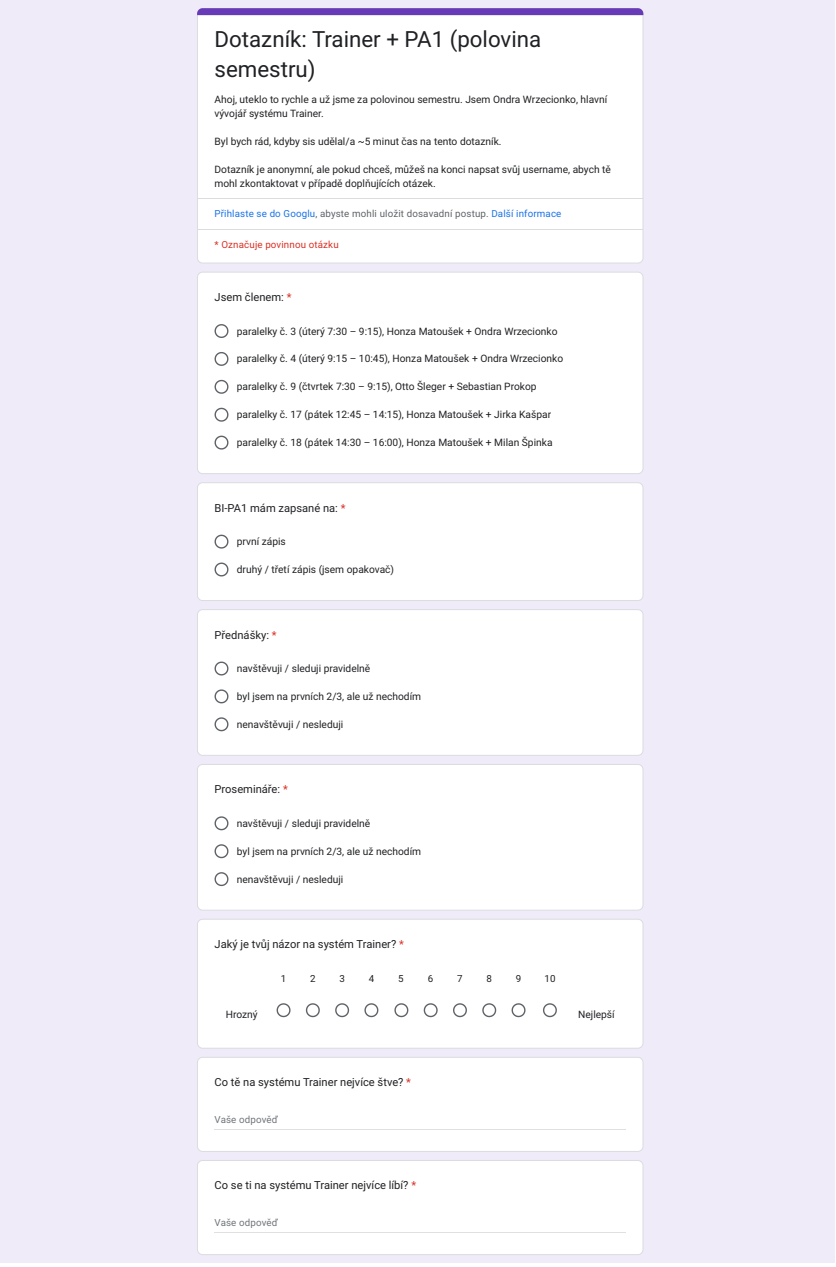


Obrázek D.6: Podmíněná otázka, kde učitel viděl, že se studenti nejvíce ztráceli



Obrázek D.7: Otázky o způsobu výuky a novém systému

Dotazník spokojenosti se systémem (listopad 2023)



Dotazník: Trainer + PA1 (polovina semestru)

Ahoj, uteklo to rychle a už jsme za polovinou semestru. Jsem Ondra Wrzecionko, hlavní vývojář systému Trainer.

Byl bych rád, kdyby sis udělal/a ~5 minut čas na tento dotazník.

Dotazník je anonymní, ale pokud chceš, můžeš na konci napsat svůj username, abych tě mohl zkontaktovat v případě doplňujících otázek.

Přihlaste se do Googlu, abyste mohli uložit dosavadní postup. Další informace

* Označuje povinnou otázku

Jsem členem: *

- paralelky č. 3 (úterý 7:30 – 9:15), Honza Matoušek + Ondra Wrzecionko
- paralelky č. 4 (úterý 9:15 – 10:45), Honza Matoušek + Ondra Wrzecionko
- paralelky č. 9 (čtvrtek 7:30 – 9:15), Otto Šleger + Sebastian Prokop
- paralelky č. 17 (pátek 12:45 – 14:15), Honza Matoušek + Jirka Kašpar
- paralelky č. 18 (pátek 14:30 – 16:00), Honza Matoušek + Milan Špinka

BI-PA1 mám zapsané na: *

- první zápis
- druhý / třetí zápis (jsem opakovač)

Přednášky: *

- navštěvuji / sleduji pravidelně
- byl jsem na prvních 2/3, ale už nechodím
- nenavštěvuji / nesleduji

Proseminář: *

- navštěvuji / sleduji pravidelně
- byl jsem na prvních 2/3, ale už nechodím
- nenavštěvuji / nesleduji

Jaký je tvůj názor na systém Trainer? *

	1	2	3	4	5	6	7	8	9	10	
Hrozný	○	○	○	○	○	○	○	○	○	○	Nejlepší

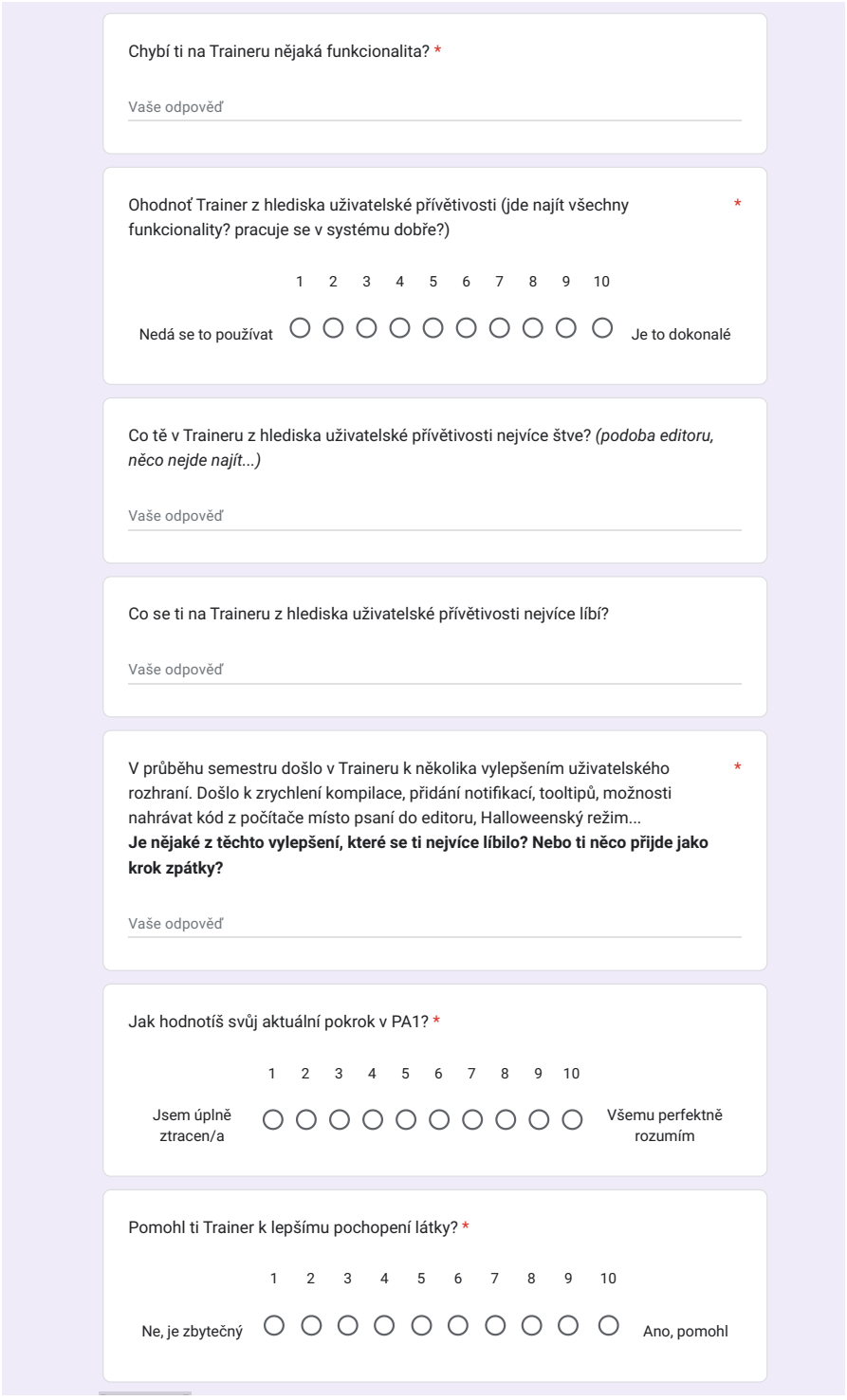
Co tě na systému Trainer nejvíce štve? *

Vaše odpověď

Co se ti na systému Trainer nejvíce líbí? *

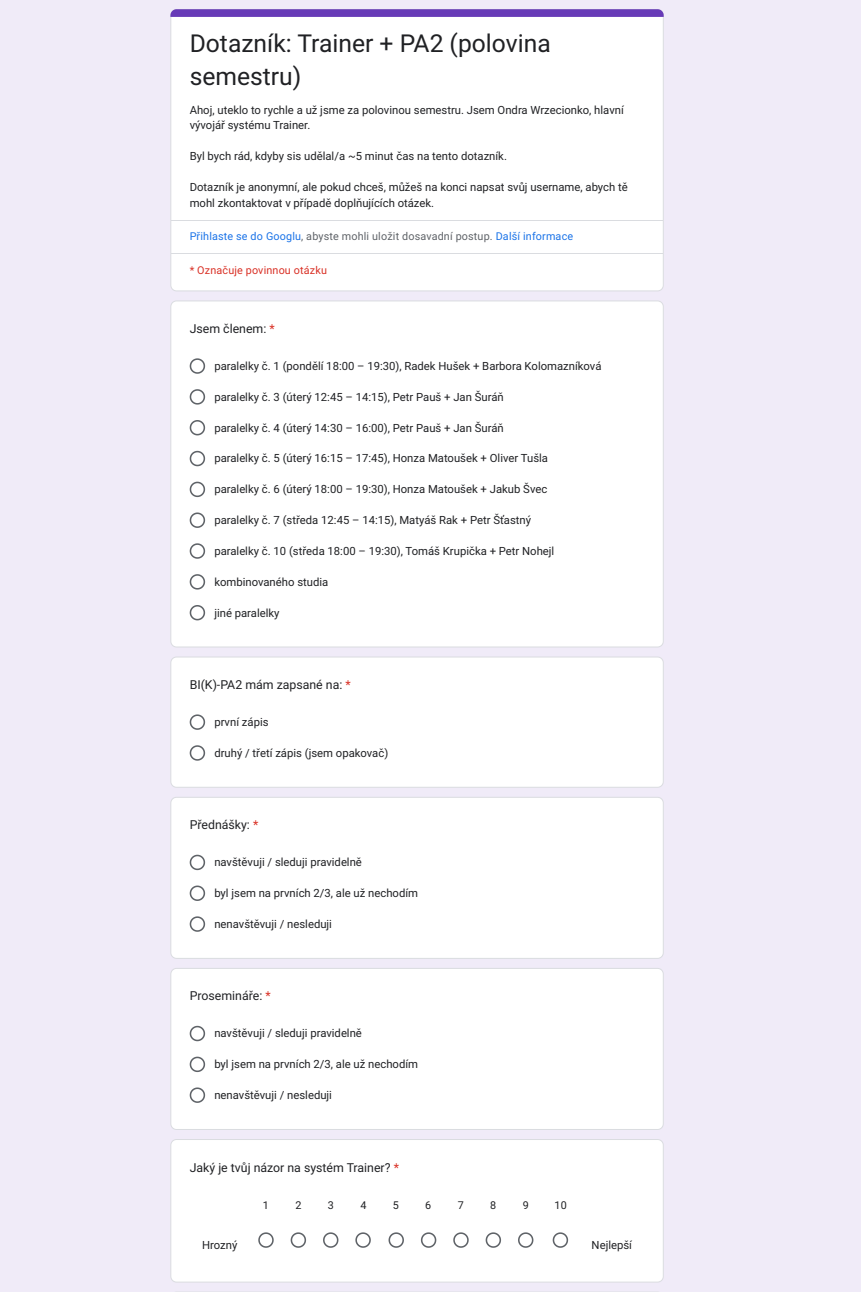
Vaše odpověď

Obrázek D.8: Otázky pro zařazení studenta, celkový pohled na systém

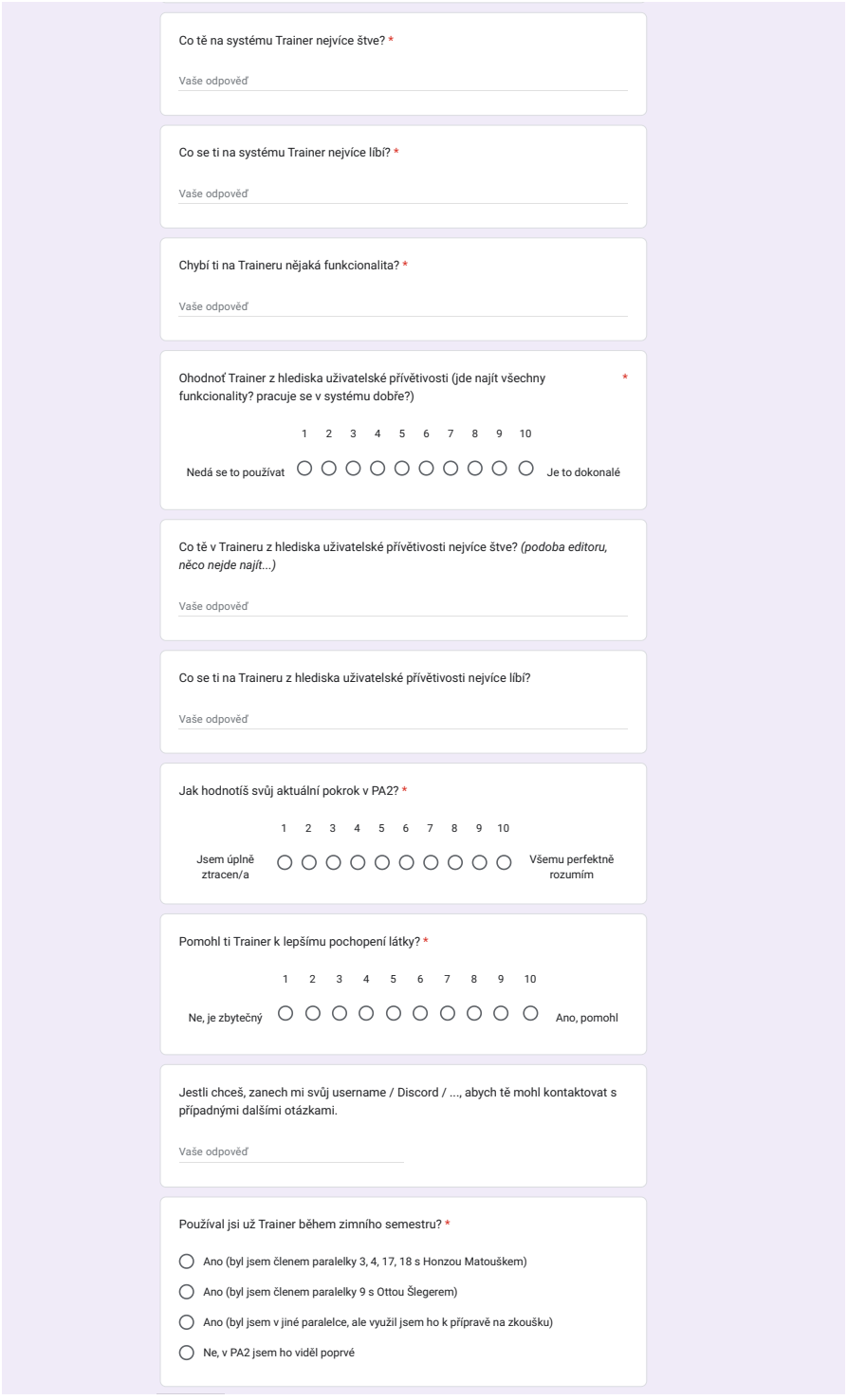


Obrázek D.9: Otázky na uživatelské rozhraní a možná vylepšení

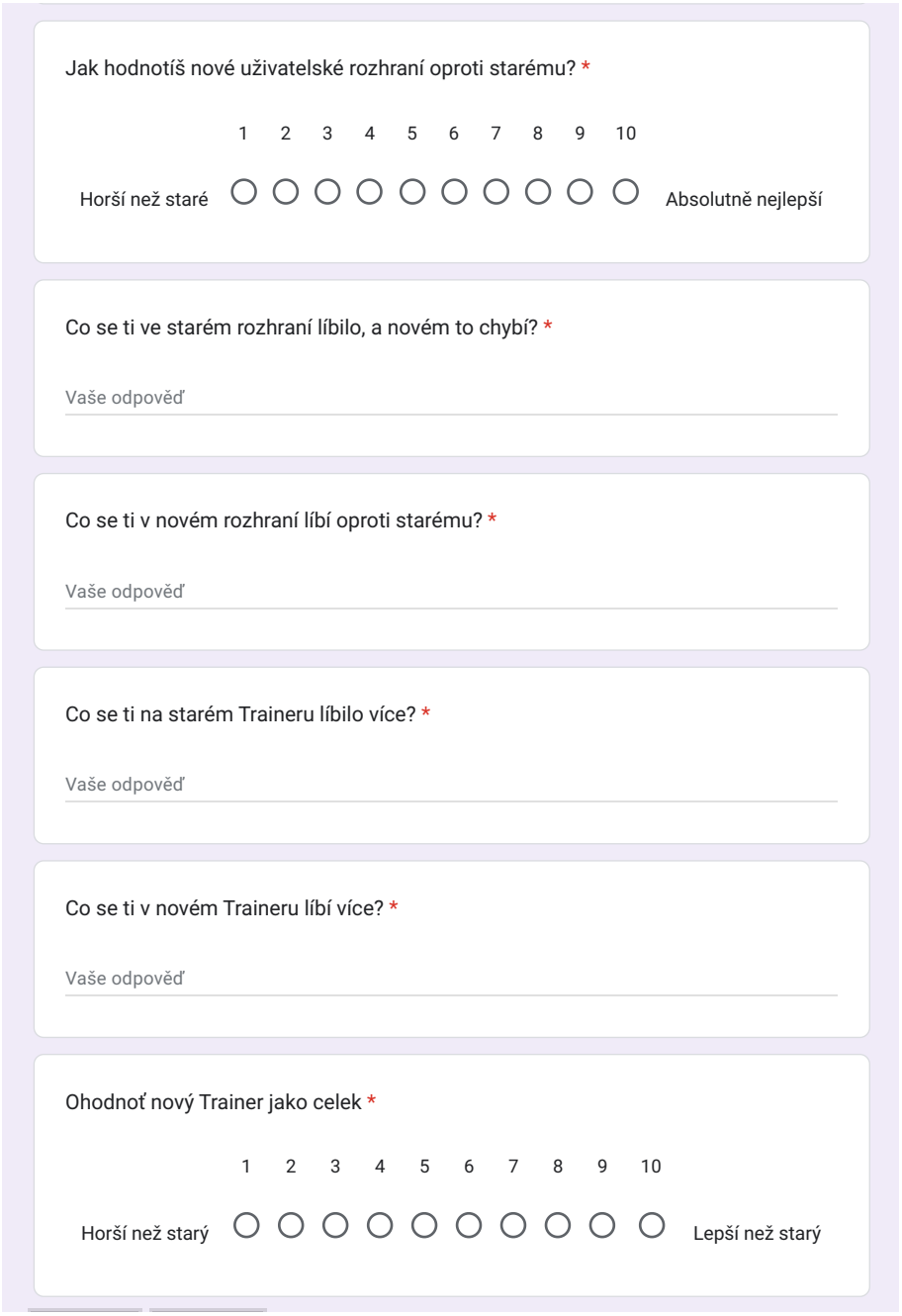
Dotazník spokojenosti se systémem (duben 2024)



Obrázek D.10: Otázky pro zařazení studenta, celkový pohled na systém



Obrázek D.11: Otázky na UI a předchozí zkušenost se systémem



Obrázek D.12: Podmíněná sekce s porovnáním starého a nového rozhraní

Příloha E

Obsah příloh

- `readme.txt` ..... stručný popis příloh
- `documentation.pdf` ..... uživatelská dokumentace systému
- `design`
  - `ea` ..... Enterprise Architect projekt a diagramy
  - `figma` ..... návrh uživatelského rozhraní v nástroji Figma
  - `forms` ..... výsledky studentských a učitelských dotazníků
  - `model` ..... konceptuální model databáze
  - `nur` ..... testovací scénáře a heuristická analýza
  - `postman` ..... integrační testy pro nástroj Postman
  - `swagger.yaml` ..... návrh REST API pro nástroj Swagger
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `trainer`
    - `backend` ..... zdrojové kódy backendu
    - `db` ..... skripty pro vytvoření, naplnění a migraci databáze
    - `frontend` ..... zdrojové kódy frontendu
    - `docker-compose.yml` ..... soubor pro konfiguraci Docker Compose
    - `test.env` ..... konfigurační soubor pro Docker image backendu
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF