FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	dbs.fit.cvut.cz testy - frontend pro studenty
Student:	Dana Suchomelová
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

V předmětu BI-DBS ČVUT FIT, katedry Softwarového inženýrství, je třeba testovat studenty v průběhu semestru i u zkoušky.
Cílem této práce je navázat na aktuální stav vývoje backendu a na zkušenosti z předmětu BI-SP1 a BI-SP2 a realizovat plně použitelný a přívětivý frontend z pohledu studenta pro realizaci testů a jejich doprovodných procesů.

Postupujte v těchto krocích:
1. Analyzujte stávající frontend testů z pohledu studenta a dále aktuální stav mikroslužeb backendu poskytující funkcionalitu k vámi řešené části. Dále analyzujte předešlé práce věnující se tomuto tématu.
2. Na základě analýzy a vlastních zkušeností navrhněte uživatelské rozhraní z pohledu studenta věnující se psaní a kontrole výsledků testů a demo testů a dále i patřičný návrh k budoucímu vývoji.
3. Dle návrhů implementujte část psaní a kontroly výsledků testů.
4. Navrhněte a realizujte vhodné sady testů.
5. Pokuste se opravit zjištěné nedostatky z testování.
6. V průběhu vývoje řádně dokumentujte převážně z důvodu zajištění návaznosti z pohledu dalších částí nově vznikajícího frontendu.
7. Navrhněte budoucí směr rozvoje, shrňte dosažené výsledky.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 9. února 2023 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

dbs.fit.cvut.cz testy – frontend pro studenty

Dana Suchomelová

Katedra Softwarového inženýrství
Vedoucí práce: Ing. Jiří Hunka

11. května 2023

Poděkování

V první řadě bych chtěla poděkovat vedoucímu práce, Ing. Jiřímu Hunkovi, který mi během práce poskytoval opravdu cennou zpětnou vazbu, zejména při vytváření návrhů. Dále bych chtěla poděkovat týmu backendu (Jakub Pavličko, Tomáš Douba a Bc. Radoslav Hašek), který se mnou měl trpělivost při řešení problémů s backendovou částí. Velké díky patří také týmu SP (Radim Květ a Vojtěch Březina), který mi poskytl část imlementace frontendu. V neposlední řadě bych ráda poděkovala blízkým, Samuelovi Švalichovi a Janu Trojákovi, kteří mi při vytváření práce byli velkou oporou. Děkuji také všem testerům, kteří mi poskytli zpětnou vazbu, rodině za podporu a převážně pak mamince, která provedla korekturu textu.

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 11. května 2023 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Suchomelová, Dana. *dbs.fit.cvut.cz testy – frontend pro studenty.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Tato práce se zabývá návrhem a implementací nového frontendu studentských testů pro portál dbs.fit.cvut.cz. Portál je využíván pro podporu předmětu Databázové systémy na FIT ČVUT v Praze. V analýze jsou identifikovány hlavní nedostatky současného řešení studentských testů a stanoveny požadavky na frontend nový. Na základě analýzy vznikly nové návrhy, které sloužily jako předloha pro implementaci. Portál byl realizován pomocí frameworků Vue.js a Quasar. Dokončená část implementace (přehled testů, psaní testu a náhled opraveného testu) byla uživatelsky otestována a na základě výsledku testů byly navrhnuty změny pro budoucí vývoj. Kromě návrhů a implementace vznikla v rámci této práce také nová dokumentace pro vývoj.

Klíčová slova webová aplikace, frontend, uživatelské rozhraní, studentské testy, databázové systémy, Vue.js, Quasar, vývojářská dokumentace

Abstract

This thesis focuses on the design and implementation of a new frontend for student tests on the dbs.fit.cvut.cz portal. The portal is used to support the Database Systems course at the Faculty of Information Technology, Czech Technical University in Prague. The analysis identifies the main shortcomings of the current student test solution and establishes requirements for the new frontend. Based on the analysis, new designs were created, serving as a template for implementation. The portal was developed using the Vue.js and Quasar frameworks. A completed part of the implementation (test overview, writing test, and overview of corrected test) underwent user testing, and based on the test results, changes were proposed for future development. In addition to the design and implementation, this thesis also includes the creation of new documentation for development purposes.

Keywords web application, frontend, user interface, student tests, database systems, Vue.js, Quasar, development documentation

Obsah

Seznam obrázků

Úvod

Zkoušení a testování studentů je běžným procesem na vysokých školách. Dříve se jednalo spíše o písemné testy a ústní zkoušky, ale díky rozvíjejícím se technologiím je v dnešní době stále častější psaní testů přes webové aplikace. Stejně tak tomu je i u předmětu Databázové systémy (BI-DBS), který je vyučován na Fakultě informačních technologií (FIT) na ČVUT v Praze. V průběhu absolvování předmětu BI-DBS studenti využívají portál dbs.fit.cvut.cz (dále jen portál DBS). Píší zde testy, vytvářejí semestrální práci, a jsou zde hodnoceni vyučujícími. Portál DBS obsahuje mnoho nedostatků a vyžaduje náročnou údržbu. Proto se pomocí novějších technologií začala vyvíjet zcela nová aplikace.

Cílem této práce je v první řadě analyzovat současné řešení frontendu psaní testů a zobrazování jejich výsledků a nalézt případné nedostatky. Dále pak analyzovat předešlé práce věnující se podobné problematice, a také prozkoumat aktuální stav backendu portálu. Poté navázat na tuto analýzu a navrhnout přívětivé uživatelské rozhraní studentských testů s ohledem na nalezené nedostatky. Dalším cílem je naimplementovat část psaní testů a jejich zobrazení a hotovou část implementace uživatelsky otestovat. Dále pak podle výsledku testování navrhnout možné změny pro budoucí rozvoj. Cílem práce je také vytvořit dokumentaci pro vývojáře a během práce dokumentovat.

Zvolené téma jsem si vybrala z toho důvodu, že jsem se problematikou studentských testů zabývala už během studia. Motivací bylo vytvořit přehledné a intuitivní uživatelské rozhraní, které zpříjemní průchod předmětu mnoha studentům. Benefitovat z nového frontendu mohou také vyučující, kteří nyní musí řešit problémy spojené s psaním testů a chybami pramenících z omylů studentů. Nová dokumentace pro vývojáře usnadní příchod novým studentům do projetu portálu DBS a urychlí proces počátku vývoje.

Kapitola 1

Analýza

Dříve než přistoupím k analýze samotné, představím čtenáři zvolenou metodiku vývoje. Postupoval jsem v souladu s kroky definovanými v zadání práce. Tyto kroky lze shrnout do následujících fází: analýza a sběr požadavků, návrh, implementace a testování. Tento postup odpovídá Vodopádovému modelu, který předepisuje dokončení jedné fáze před zahájením další. [1]

V této kapitole seznámím čtenáře s předmětem Databázové systémy a také portálem, který předmět využívá. Dále provedu analýzu současného stavu části portálu, zabývající se studentskými testy. Prozkoumám také konkurenční aplikace a již vzniklé práce, věnující se podobné problematice. Dále provedu analýzu aktuálního stavu mikroslužeb, stejně jako aktuálního stavu frontendu. Na základě analýzy pak stanovím požadavky na nový frontend studentských testů.

1.1 Databázové systémy

V této podkapitole čtenáře uvedu do kontextu práce. Seznámím je tedy s předmětem Databázové systémy, portálem DBS, vývojem v rámci předmětu Softwarový týmový projekt (BI-SP1 a BI-SP2) a stručně nastíním důvody k vytvoření nového projektu.

O předmětu

BI-DBS (Databázové systémy) je povinný předmět Fakulty informačních technologií (FIT), který si studenti zapisují zpravidla v 2. semestru bakalářského studia na ČVUT v Praze. Tento předmět studenty naučí navrhovat menší databáze pomocí konceptuálních modelů a seznámí je se základy jazyků relační algebry (RA) a SQL.

Portál DBS je systém, který byl vytvořen za účelem podpory předmětů vyučujících databázové systémy. Doposud byl však využíván pouze pro potřeby předmětu BI-DBS. V portálu DBS studenti vytvářejí a odevzdávají semestrální

práce, píší testy a kontrolují si svoje výsledky. Učitelé využívají portál DBS pro správu testů, semestrálních prací a také pro hodnocení studentů. [2]

Vývoj v rámci SP1 a SP2

Na portálu DBS pracují převážně studenti předmětů Softwarový týmový projekt (dále jen SP1 a SP2). Jedná se o předměty oboru Softwarového inženýrství, kde si studenti vybírají projekty, na kterých se budou podílet. Jedním z projektů je právě portál DBS. [2]. Studenti zde pracují v týmech, v nichž jsou také hodnoceni. Každý tým si volí vedoucího, který má na starosti souhru celého týmu. Řízení projektu nastavil ve své BP Ing. Oldřich Malec. [2] Tento projekt probíhá pod vedením Ing. Jiřího Hunky (dále jen vedoucí projektu), který má jasné vize a aktivně usiluje o posunutí projektu vpřed. FE (frontend) portálu má na starosti již zmíněný Ing. Oldřich Malec. BE (backend) portálu probíhá pod dohledem Ing. Andriiho Plyskache.

Sama jsem v rámci SP1 a SP2 měla příležitost na portálu DBS pracovat. Společně s týmem jsme se věnovali testovému modulu a jeho problematice z pohledu studentů. To bylo však předtím, než se začal vytvářet zcela nový projekt.

Rozhodnutí o vytvoření nového projektu

Někteří studenti, podílející se na vývoji portálu, nemají dostatečné zkušenosti s implementací rozsáhlejších aplikací. Zároveň se na projektu během vývoje vystřídalo velké množství studentů. Tyto faktory jsou důvodem, proč se nyní potýkáme s nekvalitním kódem, který je nutné neustále opravovat a přepisovat. Probíhalo mnoho refaktoringů zdrojového kódu, avšak s vidinou možnosti použití novějších technologií bylo rozhodnuto o vytvoření zcela nové aplikace. Dříve zmíněný Ing. Andrii Plyskach, vedoucí týmů BE, ve své diplomové práci vytvořil novou architekturu portálu DBS, kde se rozhodl monolitní architekturu nahradit mikroslužbami. [3] Co je mikroslužba a jaký je aktuální stav BE více popíši v kapitole 1.5. Bylo rozhodnuto o vytvoření *klient–server* aplikace. Což znamenalo FE (klientskou část) kompletně oddělit od BE (serverové části). Aktuální stav projektu FE je opět popsán v samostatné podkapitole 1.5. Framework nového FE a další knihovny zvolené pro nový projekt jsou více rozebrány v sekci Technologie pro implementaci 2.4.

Testy a jejich typy

Jelikož se má práce zabývá studentskými testy, představím čtenáři jejich typy. Studentské testy v portálu DBS jsou rozděleny na 3 typy: demotest, semestrální test a zkouška. **Demotest** je zkušební test, který slouží k procvičování, a za který student nezíská žádné body. Jedná se o test, který si student může zkoušet opakovaně, a tím si procvičit praktickou část probírané látky. **Semestrální test** je bodově hodnocený test v semestru a jeho splnění je nutné

k tomu, aby student na konci semestru mohl skládat **zkoušku**, opět bodově hodnocenou. Po splnění zkoušky student může zažádat o zkoušku ústní, a získat tak bonusové body.

Aby nevznikla nedorozumění v oblasti pojmenování testů, určím zde jasně definované pojmy. Pokud je v práci zmíněn **test**, vždy je myšlen test obecně, tedy jakýkoliv ze tří zmíněných typů (demotest, semestrální test, zkouška). Jednotlivé typy testů budou vždy označovány zadefinovanými pojmy, bez výjimky.

1.2 Analýza UI současného stavu

Jelikož jsem sama předmětem Databázové systémy prošla, a poté se na portálu DBS dokonce podílela, byla pro mě analýza o to snazší. Uživatelské rozhraní studentských testů jsem rozdělila do 3 částí: zobrazení testů studenta, náhled opraveného testu a psaní testu. V každé části popíši přístup řešení a s ním spojené klady a zápory.

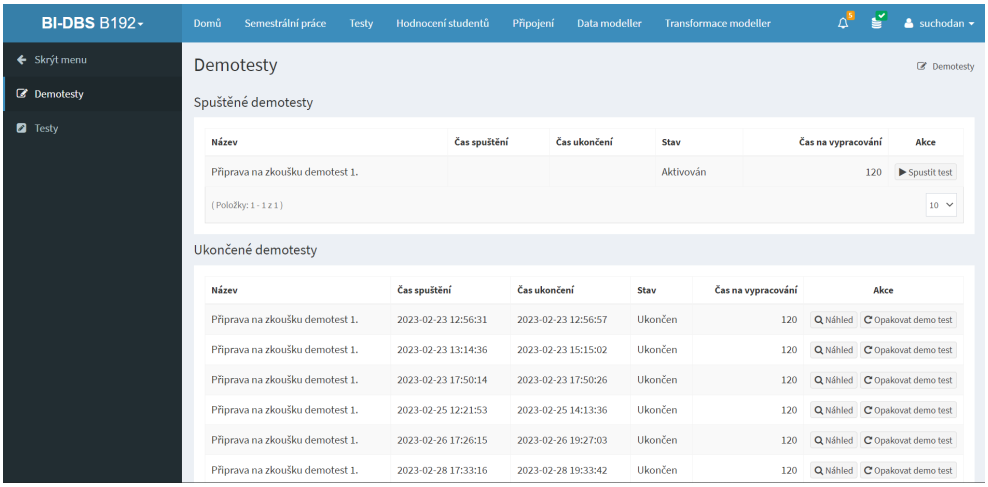
1.2.1 Zobrazení testů studenta

V současném portálu je FE testů rozdělen na 2 stránky: stránka zobrazující demotesty a stránka zobrazující semestrální testy společně se zkouškami. Mezi těmito stránkami se student přepíná pomocí bočního menu. Obě tyto stránky zobrazují nejdříve tabulku aktivních testů ke spuštění, a poté tabulku testů ukončených. Z těchto tabulek lze provést akce: SPUSTIT AKTIVNÍ TEST, PŘEJÍT NA NÁHLED OPRAVENÉHO TESTU, anebo OPAKOVAT DEMOTEST. Na obrázku 1.1 můžete vidět stránku s demotesty.

Pokud je studentovi přiřazen aktivní semestrální test nebo zkouška, systém je zabezpečen tak, že student nesmí opustit záložku testů. Jediná akce, kterou může vykonat je SPUSTIT onen aktivní test. Toto zabezpečení trvá i při samotném psaní testu. Cílem je zamezit podvádění studentů. Teprve po odevzdání testu je mu umožněno volně se pohybovat po portálu.

Klady

- Testy jsou zobrazeny formou tabulky, což je přehledné a intuitivní.
- Jsou rozlišeny testy spuštěné od ukončených.
- U tlačítek akcí jsou ikonky, což přispívá ke snadnější orientaci.
- Jsou zobrazeny všechny potřebné informace.
- Systém je zabezpečen v případě přiřazeného semestrálního testu nebo zkoušky.



Obrázek 1.1: Zobrazení demotestů studenta

[4]

Zápory

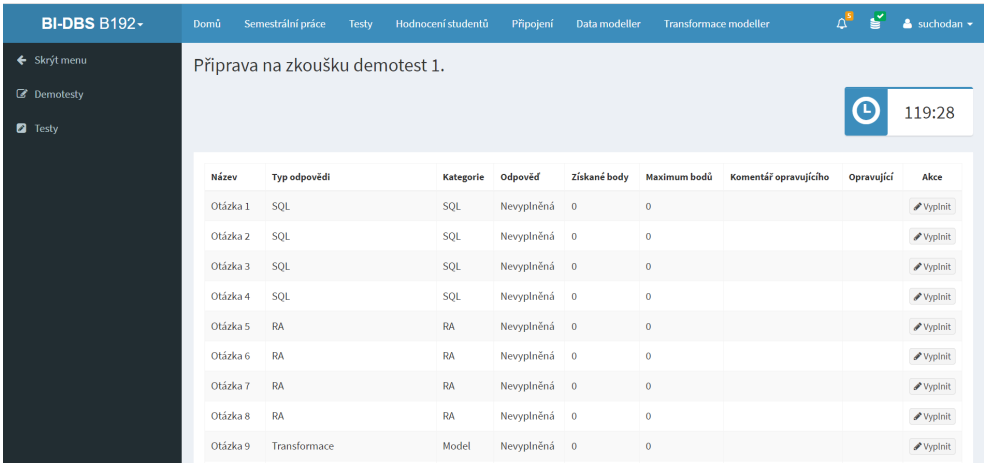
- Seřazení ukončených testů je od nejstarších po nejnovější a nelze ho změnit.
- V bočním menu jsou odděleny demotesty od hodnocených testů, a je potřeba se mezi stránkami přepínat.
- V tabulkách nelze řadit ani filtrovat.
- V tabulce ukončených semestrálních testů a zkoušek nelze jednoduše určit, zda student v testu uspěl (tedy zda splnil minimální počet bodů).
- Tlačítka akce jsou poměrně malá a tabulka nepodporuje kliknutí celého řádku.

Jedním z hlavních nedostatků současného UI je rozdělení různých typů testů na dříve zmíněné 2 stránky. Student tak nemá všechny testy pohromadě a musí přecházet mezi stránkami. V návrzích tedy přijdu s alternativním řešením. Také se zaměřím na rozlišení stavů testů. Jelikož tabulky spuštěných a ukončených testů sdělují identické informace, není potřeba testy rozdělovat do 2 separátních tabulek.

1.2.2 Psaní testu

Uživatelské rozhraní psaní testů (různých typů) se kromě tabulky při spuštění testu neliší, a tak představím UI pouze na příkladu demotestu.

Po spuštění demotestu se studentovi zobrazí souhrn testových otázek. Jak můžete vidět na obrázku 1.2, opět zde byla využita tabulka. Pomocí akce VYPLNIT student může přejít na vyplnění odpovědi. Každá otázka má vlastní stránku, na které se mu zobrazí zadání, úkol a odpověď.



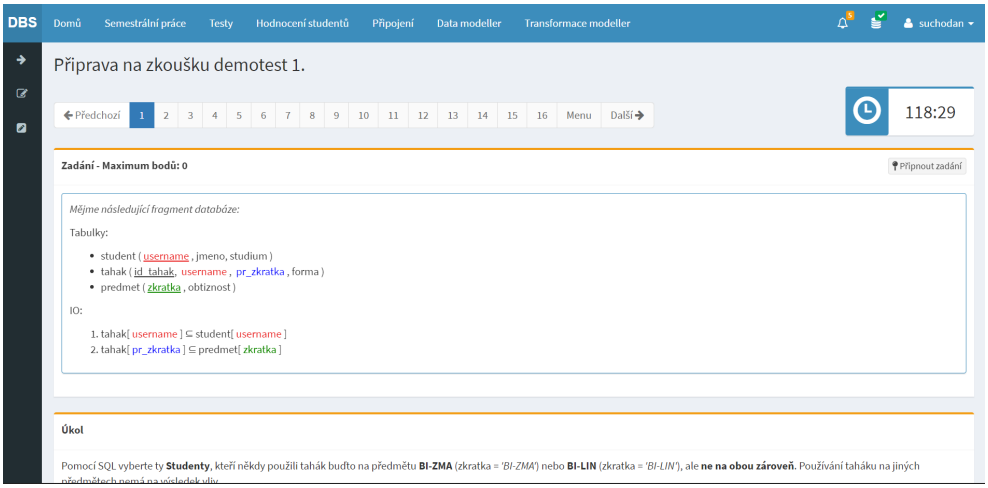
Obrázek 1.2: Souhrnná tabulka otázek při psaní demotestu

[4]

Zadání může mít formu textu, obrázku, modelu, transformace, normalizace, nebo databázového připojení (to se studentovi nezobrazuje). Vždy se jedná o nějaký popis problému, nad kterým bude student pracovat. [5] Jedna otázka může mít více zadání, a v takovém případě se jednotlivá zadání zobrazují pod sebe. Praktickou funkcionalitou zadání je jeho připnutí na obrazovku. Box se zadáními se zafixuje na jedno místo a když se student na obrazovce přesune k vyplňování odpovědi, zadání má stále na očích. V současném stavu ale není realizováno správně, a zadání se zafixuje doprostřed obrazovky. Prostor na vyplňování odpovědi je tedy úzký, a uživatele může rušit.

Úkol u otázky může na rozdíl od zadání být pouze jeden. Jedná se o instrukce, které popisují, co by měl student udělat v rámci konkrétního zadání. [5] Zobrazení zadání a úkolu v současném portálu DBS si můžete prohlédnout na obrázku 1.3.

Každá otázka má typ odpovědi a kategorii. Kategorie určuje, co je tématem dané otázky. **Typ odpovědi** pak udává, jakým způsobem bude student odpovídat. Systém podporuje následující typy odpovědi: text, check box, radio list, diagram, RA, SQL, transformaci a normalizaci. Dále v krátkosti popíšu jednotlivé typy a jejich současná řešení.



Obrázek 1.3: Psaní testu — zadání a úkol

[4]

Text
Textové pole slouží pro zadání textové odpovědi studenta. Většinou se používá pro prověření teoretických znalostí studentů.

Check box
Jedná se o zaškrtávací pole, ve kterém student vybírá libovolný počet možností. Student může být dotazován jak na teoretické, tak i na praktické otázky. Pro každou možnost je zobrazen jeden řádek.

Radio list
Radio list má podobný princip jako check box, s tím rozdílem, že v radio listu je povinně vždy jen jedna správná odpověď a student tedy zaškrtává pouze jedno políčko.

Diagram
Typem diagram je student prověřován, zda si osvojil látku konceptuálních modelů relačních databází. Pomocí kreslítka systému modelují studenti *entity* a *relace* mezi nimi. Komponentu kreslítka ve svých pracích vytvářeli Bc. Jiří Slavotínek [6] a Ing. Tomáš Fedor [7].

RA
U typu odpovědi RA se studenti dotazují nad relační databází. Dotazy píší do upraveného textového pole. K dispozici mají tlačítka s operátory, které se při kliknutí propíší do textového pole. Kromě toho zde mají k dispozici seznam

entit, nad kterými pracují. Pokud kliknou na *entitu*, název se jim opět propíše do textového pole. Tato funkcionalita má za úkol snížit chybovost v názvech. Student svoji odpověď může ověřit. Při ověření je student pomocí notifikací upozorněn, pokud byla při vyhodnocování dotazu identifikována chyba.

SQL
U typu odpovědi SQL se studenti opět dotazují nad relační databází. Obrázek 1.4 ukazuje zobrazení tohoto typu odpovědi, které je velice podobné jako u předchozího typu RA. Student se dotazuje v textovém poli, po levici vidí seznam *entit* a opět je při dotazování upozorněn notifikacemi o nečekaných událostech dotazu. Místo operátorů zde studenti naleznou odkazy na dokumentaci Postgresu a Oraclu, které mohou během testu využít.

Transformace
U transformace konceptuálního modelu na relační má student za úkol vytvořit relační zápis podle konceptuálního modelu tak, aby popisoval stejné schéma. Transformaci studenti opět zapisují do textového pole, jsou jim k dispozici potřebné operátory, a také jsou upozorňováni, pokud v zápisu mají nečekanou chybu syntaxe. Tentokrát ne pomocí notifikace, ale pomocí zbarveného boxu přímo u odpovědi.

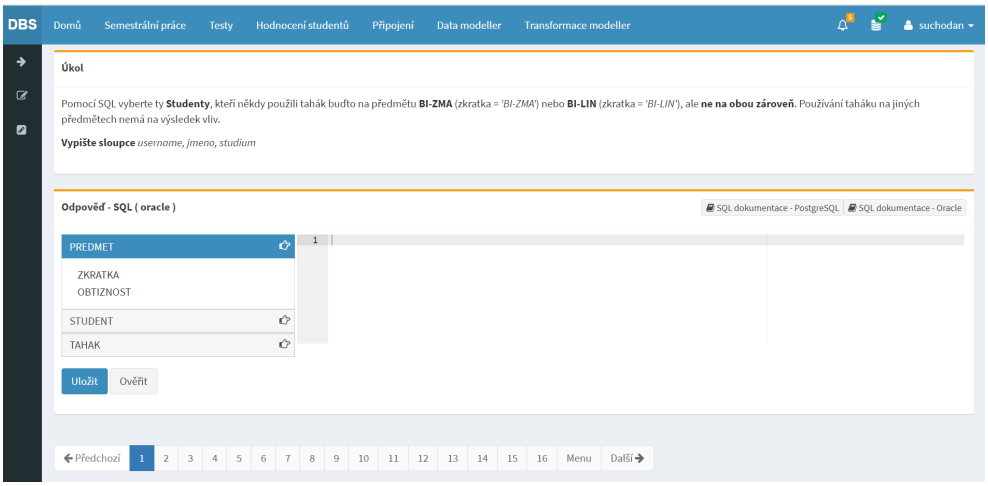
Normalizace
Studenti jsou dotazování na funkční závislosti, normální formu, tranzitivní uzávěr... Je to velice komplexní typ odpovědi, který se skládá z textových otázek a zaškrtávacích polí. Problematikou nejen uživatelského rozhraní odpovědi typu normalizace se ve své BP věnoval Ing. Marek Erben. Více o realizaci si můžete přečíst zde. [8]

V současném portálu nejsou odpovědi ukládány automaticky, ale student je musí ukládat manuálně pomocí tlačítka. Tento přístup bohužel hraje proti studentům, kteří často zapomenou změněnou odpověď uložit, a nakonec odevzdají test s neaktuální verzí odpovědi. Na tento problém se detailněji zaměřím v návrzích nového UI.

Během celého průběhu testu (tedy od souhrnné tabulky, přes vyplňování odpovědí, až po odevzdání) je studentovi zobrazen odpočet minut, který udává zbývající čas do konce testu. Na každé stránce s otázkou je také navigační menu, které slouží pro přechod mezi otázkami. Toto menu však není příliš přehledné.

Jak bylo zmíněno v předchozí části 1.2.1, během vyplňování semestrálního testu a zkoušky je systém zabezpečen proti podvádění. Pokud student právě vyplňuje semestrální test nebo zkoušku, nesmí opustit test a pohybovat se v portálu, dokud test neukončí.

Student může test ukončit předčasně. Toto ukončení probíhá ze stránky souhrnu otázek. Po odevzdání testu se test okamžitě ukončí a odešle se k vyhodnocení. Pokud student test neodevzdá předčasně, test je ukončen po uplynutí časového limitu automaticky.



Obrázek 1.4: Psaní testu – SQL odpověď

[4]

Klady psaní testu

- Odpočet času, který znázorňuje zbývající počet minut.
- Zadání se dá připnout.
- Rozložení stránky u každé otázky je konzistentní.
- V souhrnu všech testových otázek je ukázáno, zda je odpověď vyplněná, nebo nevyplněná.

Zápory psaní testu

- Odevzdání testu není ošetřeno proti mylným akcím. Pokud tedy student nechtěně klikne na tlačítko UKONČIT TEST, test se odešle k vyhodnocení a již není návratu.
- Navigace otázek není vždy viditelná, stejně jako odpočet času.
- Student se přepíná mezi jednotlivými stránkami otázek a navigace mu nedává moc dobrou představu o tom, kde se v testu nachází.

- Připnutí zadání nefunguje správně.

- U odpovědi typu RA a SQL je student při chybě informován pomocí notifikace. Tyto notifikace se mu skládají pod sebe a při větším počtu nemusejí být přehledné.

Možná dokonce největším nedostatkem současného systému v části studentských testů je ukončování testů. Pokud student stiskne tlačítko UKONČIT TEST, otázky jdou okamžitě k automatické opravě, a student, ačkoli test odevzdal neúmyslně, se nemá možnost k testu vrátit. Často se pak studenti obracejí na vyučující a domlouvají se na náhradním termínu opakování testu. Také vedoucí projektu vnímá tento nedostatek jako stěžejní, a proto se na něj u navrhování zaměřím.

1.2.3 Náhled opraveného testu

Jelikož se náhledy opravených testů liší podle typů testů, popíši je odděleně. Nejdříve představím **náhled demotestu**. Všechny demotesty jsou vyhodnocovány automaticky, a tedy ihned po ukončení si student může zobrazit své výsledky. Náhled opraveného demotestu se zobrazuje v modálním okně. Nachází se zde souhrn testových otázek formou tabulky. Tato tabulka však nesděluje žádné informace o úspěšnosti. Z tabulky lze přejít na detail jednotlivých otázek, který se objevuje v dalším modálním okně. Náhled opravené otázky vypadá podobně jako v samotném psaní demotestu – zadání, úkol, odpověď. Později se opět dostanu k jednotlivým typům odpovědi a jejich opraveným podobám, nejdříve ale popíši náhledy semestrálních testů a zkoušek.

Otázka ze **semestrálního testu** nebo **zkoušky** může být vyhodnocena automaticky (v případě, že byla shodná s některou z referenčních odpovědí nebo jinou studentskou odpovědí opravenou učitelem), anebo je opravena učitelem ručně. Opravení semestrálního testu a zkoušky tedy může trvat delší dobu. Současný systém nyní nepodporuje zobrazení opravených odpovědí semestrálních testů a zkoušek. Studentovi je zobrazena pouze tabulka jednotlivých otázek opět v modálním okně. Z tabulky dokáže vyčíst, jak byl v otázce úspěšný (tj. kolik bodů získal), jestli byla odpověď vyhodnocena automaticky nebo učitelem, popř. kterým a případný komentář. Nemůže si však zobrazit svou odpověď, což je také jedna z věcí, která se v novém portálu změní.

Dále rozeberu jednotlivé **typy odpovědí** a jejich zobrazení v opraveném **demotestu**. U všech typů se odpověď zobrazuje v režimu *readonly*, aby student nemohl danou odpověď upravit.

Text

V demotestu je studentovi zobrazeno textové pole s jeho odpovědí a procentuální shoda s odpovědí správnou.

Check box
V check boxu je každý řádek zabarven (červenou nebo zelenou) barvou podle toho, zda student na danou možnost odpověděl správně. V případě kompletně bezchybné odpovědi jsou všechny řádky zbarveny zelenou barvou.

Radio list
Radio list, podobně jako check box, zobrazuje zbarvené řádky. V případě správné odpovědi jsou opět všechny řádky zelené. Pokud student odpověděl chybně, červeně se zbarví dvojice řádků: studentova odpověď, která byla chybně zaškrtnutá a správná odpověď, která zaškrtnutá nebyla.

Diagram
Studentovi je zobrazeno kreslítko s jeho odpovědí. V diagramu se červenou barvou zabarví ty části, které jsou chybné. Mohou to být *relace*, *entity* nebo *atributy*.

RA
Studentovi je zobrazen jeho dotaz. Pod jeho odpovědí se nachází zbarvené boxy s informacemi o správnosti navrácených dat. Student je informován, pokud při vyhodnocení došlo k chybě, pokud dotazovaná *relace* nebyla nalezená, a dále je informován o počtu a správnosti názvů sloupců a řádků.

SQL
Opravená odpověď typu SQL je zobrazena stejně jako předchozí RA.

Transformace
Zde je studentovi zobrazena jeho odpověď a rozbor chyb, který porovnává studentovu odpověď s nejpodobnější referenční odpovědí. Tento rozbor ukazuje informace o *entitách*, *atributech* a *relacích* (např. kolik chybí *entit*).

Normalizace
U typu odpovědi normalizace opět odkážu na BP Ing. Marka Erbena [8].

Klady náhledu opraveného testu

- U opravených odpovědí je často využito barevné zvýraznění, které je intuitivní.
- Z přehledu tabulky otázek semestrálního testu a zkoušky lze snadno vyčíst hlavní informace, které studenta zajímají.

Zápory náhledu opraveného testu

- Student si nemůže zobrazit opravené odpovědi semestrálního testu a zkoušky.
- Náhledy testů se zobrazují v modálních oknech, které se vrství přes sebe.
- V náhledu otázky demotestu není vždy zobrazena procentuální úspěšnost. Zobrazena je pouze u některých typů odpovědí.
- V náhledu otázky demotestu, pokud je zadání dlouhé, studentova odpověď není vidět na první pohled. Přitom je to hlavní věc, která studenta zajímá.
- U demotestu studentovi není sděleno, jak byl úspěšný v celém testu.

Ze zmíněných kladů a záporů jde na první pohled poznat, že zápory v této části převládají. Proto v návrhu bude oproti stávajícímu řešení mnoho změn.

1.3 Analýza konkurence

V této podkapitole se budu zabývat analýzou konkurenčních aplikací, s cílem inspirovat se ideálními řešeními problémů uživatelského rozhraní. Zaměřím se na psaní testu a zobrazování výsledků testu.

Moodle

Moodle je veřejně otevřený software pro správu výuky. Používají ho například školy pro podporu výuky, a to jak střední školy, tak i vysoké. Také zde probíhají různé elektronické kurzy. [9]

S moodle testy jsem se seznámila již na střední škole a psaní testů mi v této aplikaci vždy vyhovovalo. A to především díky rozložení stránky při vyplňování testu, které jsem pro tuto práci dále analyzovala.

Klasické rozložení moodle testu je na dva sloupce. Užší sloupec slouží k navigaci otázek v testu. Je zafixovaný, a uživatel má tedy stále přehled o tom, kterou otázku vyplňuje. V tomto sloupci je také odpočet času na vyplnění testu. V širším sloupci je pak obsah zadání a odpovědi. Tímto rozložením stánky se budu inspirovat při vytváření návrhů.

Grades

Systém grades.fit.cvut.cz je webová aplikace FIT ČVUT v Praze, která slouží k hodnocení studentů. [10] Hodnocení je rozděleno na jednotlivé předměty a bodují se zde testy, semestrální práce a další náplň výuky. Je zde zobrazován poměr počtu dosažených bodů k počtu maximálních bodů (např. 2/5). Tento systém zvýrazňuje počet získaných bodů barvou, která znázorňuje úspěšnost.

Jedná se o procentuálně vypočítanou barvu na škále mezi oranžovou a zelenou barvou. Toto barevné znázornění dle mého názoru podává informaci o úspěšnosti jasně a přehledně. Podobným konceptem se budu řídit i v návrzích nového UI pro zobrazení studentských výsledků.

1.4 Práce věnující se související problematice

Refaktoringem testového modulu současného portálu DBS se zabývalo mnoho studentů ve svých BP a DP. Novou strukturu testového modulu položil ve své BP Ing. Andrii Plyskach. V této práci se autor zabýval nedostatky dřívějšího systému, které v novém návrhu odstranil. Například se věnoval problematice podvádění studentů během testu a zkoušky, kterou vyřešil pomocí logování IP adres každého studenta. Další změnou bylo přidání nového typu zadání: databáze. Tento typ se nezobrazuje studentům, ale obsažené připojení slouží k opravování odpovědí typu RA a SQL. [11]

Na tuto bakalářskou práci navazuje hned několik dalších prací se stejnou problematikou – refaktoring testového modulu.

Refaktoring testů I

Bc. Martin Hanzl ve své BP analyzoval zobrazování testových otázek studentům, s cílem nalézt lepší řešení. Autor nejdříve do hloubky rozebral dřívější systém a podle analýzy stanovil požadavky. Poté vytvořil papírové modely nového FE testové části. Testy v bočním menu rozdělil nově podle aktivních a ukončených. V nich dále rozlišuje demotesty, semestrální testy a zkoušky. Dohromady tak vzniklo pro studenty 6 stránek týkajících se testů. Další změnou je umožnění zobrazení detailu opraveného semestrálního testu a zkoušky. Autor zůstal u dřívějšího provedení a náhledy opět jako u demotestů vyřešil pomocí modálních oken. [12]

Autor se snažil eliminovat některé z nedostatků, na které jsem sama v analýze poukázala. Dle mého názoru však nebyly vyřešeny ideálně. Především rozdělení testů na 6 stránek nevnímám jako přijatelné řešení. Proto při navrhování nebudu z této práce vycházet a přijdu s vlastním řešením této problematiky.

Refaktoring testů II

Tvorbou testů z pohledu učitele se zabýval ve své BP Bc. Pavel Jordán. Autor měl navázat na BP Ing. Andriiho Plyskache a pro tento nový BE vytvořit frontend. U vytváření testů a testových otázek je novinkou například přiřazování štítků, které slouží pro třídění jednotlivých otázek. Problematika UI testů vyučujících však nepokrývá téma mé práce, a tak ji více nebudu rozbírat. [13]

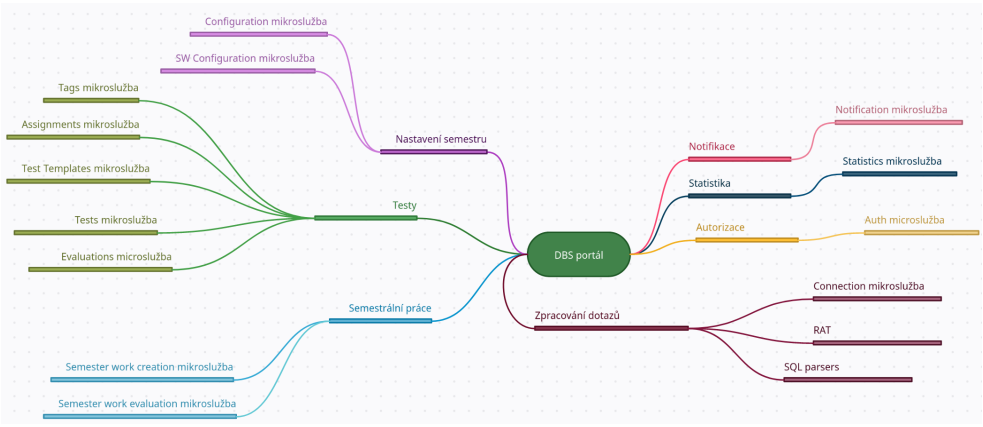
Frontend manuální opravy otázek

Na práci Ing. Andriiho Plyskache také navázal Bc. Tomáš Chalupa, který navrhoval nový FE pro manuální opravu testových odpovědí. Autor se zaměřil na responzivní UI, ve kterém si uživatel může přizpůsobit rozložení opravování otázky podle preferencí. Dále se zabýval UI jednotlivých typů odpovědí. Tato část uživatelského rozhraní však neobsahovala žádné radikální změny. [14]

1.5 Aktuální stav mikroslužeb

Jak bylo nastíněno v podkapitole 1.1, pro BE portálu byla vytvořena zcela nová architektura. Byla vyvinuta za účelem odstranění nedostatků klasické monolitické architektury, jako například špatná udržovatelnost nebo pomalý vývoj. Nový systém je rozdělen do **mikroslužeb** (také mikroservis). Základní myšlenkou mikroslužeb je rozdělení aplikace na menší služby (mikroslužby), které mají přesně definovanou roli a běží v samostatném procesu. To znamená, že každá mikroslužba je vyvíjena, upravována a nasazována samostatně. To přinese výhody především do budoucna, až bude údržba aplikace opět náročnější. Bez problémů pak bude možné část systému vyjmout, upravit a opět začlenit do běžící aplikace. Podrobnější popis si můžete přečíst v práci Ing. Andriiho Plyskache [3]. Rozdělení portálu DBS do mikroslužeb můžete vidět na obrázku 1.5.

Nový portál využívá REST API. Je to způsob, jak komunikovat mezi klientskou částí (FE) a serverovou (BE) pomocí webových protokolů jako HTTP. Pomocí požadavků jako GET, POST, PUT a DELETE (popř. PATCH) je pak v klientské části možno manipulovat s daty ze serverové části. [15]



Obrázek 1.5: Rozvržení mikroslužeb

[16]

Jak ilustruje obrázek 1.5, testová část systému je rozdělena na mikroslužby: Evaluations, Tests, Test Templates, Assignments a Tags. FE studentských testů bude komunikovat téměř se všemi vyjmenovanými mikroslužbami s výjimkou mikroslužby Tags, která bude potřeba především u vytváření testů učiteli. V rámci realizace testové části BE vznikají 3 práce.

- Jakub Pavličko – vyhodnocení testů (mikroslužba Evaluations).
- Tomáš Douba – tvorba otázek a štítků (mikroslužba Tags a Assignments).
- Bc. Radoslav Hašek – testy a testové šablony (mikroslužba Tests a Templates).

Ještě než jsem začala s vývojem, Jakub Pavličko, Tomáš Douba a Bc. Radoslav Hašek (dále jen tým BE) mi poskytli dokumentaci API. To mi umožnilo pochopit, jakým způsobem budu pracovat s daty, a mohla jsem si podle definic připravit rozhraní na FE. Týmu BE jsem z mé strany poskytla popis uživatelského rozhraní studentských testů, aby se mohli ujistit, že mají pokryté všechny požadavky. Souběžný vývoj FE a BE má bohužel tu nevýhodu, že jednotlivé části jsou na sobě závislé. Proto jsem i u svého vývoje musela čekat, než mi poskytnou alespoň první funkční celek. Do té doby jsem pracovala pouze s mockovanými daty. Souběžný vývoj s sebou však nese i výhody, a to například rychlé reakce na případné chyby. Vždy, když jsem narazila na nějakou chybu, tým BE poměrně pohotově vše opravil.

Jelikož se jedná o doposud nezaběhnutý projekt, během vývoje nastalo několik problému. Problémy při rozběhnutí projektu BE více popíši v kapitole realizace 3.

1.6 Aktuální stav frontendu

Jelikož jsem implementaci začleňovala do již vytvořeného projektu, bylo potřeba provést analýzu aktuálního stavu projektu FE. Když jsem začínala s implementací, projekt byl krátce po vytvoření a obsahoval pouze základní nedokončenou konfiguraci knihoven. Během předchozího vývoje vzniklo mnoho větví, avšak kód z nich se nikdy nedostal do větve hlavní (*master*). Po zvážení možností jsem si vytvořila novou větev z větve nejaktuálnější a začala s implementací.

Souběžně začínal nový běh SP1, a začali se konat každotýdenní schůzky. Schůzek jsem se také účastnila, abych měla přehled o aktuálním dění v projektu. Projekt byl doposud psán pomocí *Options API*. S Vue.js 3 se ale naskytla možnost použití novějšího *Composition API*. Rozdíly mezi přístupy si můžete přečíst zde [17]. Po debatě bylo rozhodnuto přepsat projekt do *Composition API*, a to především z důvodů novější technologie, u které bude

zaručena podpora s případnou novější verzí Vue.js. Přepis projektu si vzal na starosti jeden z týmů FE.

Problém byl v tom, že SP tým přepsal větev *master*, která neobsahovala nejaktuálnější verzi. Během přepisu se také změnila celá struktura projektu. Tím pádem vznikal paralelní vývoj více větví s různými technologiemi a odlišnou strukturou. Proto bylo potřeba na schůzce vyjasnit stavy zmiňovaných větví a domluvit nezbytné změny k aktualizaci hlavní větve. Nakonec tedy *master* obsahoval nejaktuálnější verzi v *Composition API*, která obsahovala funkční konfiguraci knihoven, a která mohla sloužit jako základ pro další vývoj. Tím byl navrácen pořádek v oblasti větví.

1.7 Analýza požadavků

Sběr požadavků tvoří základní předpoklad pro návrh, vývoj a testování systému. Nejdříve čtenáře seznámím s kategorizací samotnou, a dále pak představím nasbírané požadavky.

1.7.1 Kategorizace požadavků

Požadavky lze kategorizovat podle různých vlastností. V této podkapitole představím základní kategorizace.

Funkční a nefunkční požadavky

Funkční požadavky jsou požadavky na funkčnost systému, tedy to, co systém umí a jak se chová. Tyto požadavky popisují vlastnosti, chování a funkce nutné pro splnění potřeb zadavatele. Například: *"Systém musí umožňovat uživatelům vytvářet účty"*.

Nefunkční požadavky jsou požadavky na vlastnosti systému, které nejsou přímo spojeny s jeho funkčností. Tyto požadavky se obvykle týkají výkonu, spolehlivosti, bezpečnosti a dalších. Například: *"Systém musí být dostupný minimálně 99% času"*. [18]

Kategorizace FURPS

Název **FURPS** je zkratka získaná z počátečních písmen dalších kategorií. Tato kategorizace rozděluje požadavky na následující kategorie:

- **F**unctionality – funkčnost
- **U**sability – použitelnost
- **R**eliability – spolehlivost
- **P**erformance – výkonnost

- **S**upportability – podpora a rozšiřitelnost

[18]

Kategorizace MoSCoW

Kategorizace MoSCoW udává prioritu požadavků.

- **M**ust have – nejdůležitější požadavky, které musí být splněny.
- **S**hould have – požadavky, které jsou důležité, ale nejsou pro aplikaci stěžejní.
- **C**ould have – pokud jsou požadavky z předchozích dvou kategorií splněny, může se začít přemýšlet o požadavcích z této kategorie.
- **W**on't have – v této kategorii se můžou stanovit požadavky, které nebudou realizovány.

[18]

1.7.2 Sběr požadavků

Název: Zobrazení přehledu testů studenta
Zkratka: F1
Popis: Systém umožní zobrazení přehledu testů studenta pomocí tabulky. V tabulce budou zobrazeny všechny typy testů. V jediné tabulce budou zobrazeny aktivní testy i testy již ukončené a opravené. V tomto přehledu student uvidí základní informace o daných testech: název, čas spuštění, typ testu, úspěšnost, čas na vypracování a akce.
Kategorizace: Funkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Střední

Název: Řazení tabulky testů
Zkratka: F2
Popis: Testy studenta musí být automaticky seřazeny podle data spuštění. Nejdříve se zobrazí testy aktivní, a dále pak testy od nejnovějšího času spuštění. Podle tohoto sloupce by mělo být možné řadit.
Kategorizace: Funkční požadavek, Should have
Priorita: Střední
Předpokládaná obtížnost: Střední

Název: Formát zobrazení data spuštění
Zkratka: F3
Popis: Je požadováno, aby datum spuštění v tabulce testů bylo zobrazeno ve

formátu *timeago*. Například tedy *před 2 hodinami / před 5 měsíci*.
Kategorizace: Funkční požadavek, Could have
Priorita: Nízká
Předpokládaná obtížnost: Nízká

Název: Zobrazení úspěšnosti testu
Zkratka: F4
Popis: Úspěšnost demotestu, semestrálního testu a zkoušky by měla být zobrazena jinou formou. U demotestu by se měly zobrazovat procenta úspěšnosti. U semestrálního testu a zkoušky je požadováno zobrazit bodový zisk takovým způsobem, aby bylo jasné, zda student uspěl nebo ne.
Kategorizace: Funkční požadavek, Should have
Priorita: Střední
Předpokládaná obtížnost: Nízká

Název: Spuštění testu
Zkratka: F5
Popis: Studentovi je umožněno spustit si aktivní demotest, semestrální test nebo zkoušku.
Kategorizace: Funkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Nízká

Název: Zobrazení otázek při psaní testu
Zkratka: F6
Popis: Během psaní testu je požadováno, aby student neustále viděl souhrn testových otázek. V tomto souhrnu jsou informace o otázce: pořadí otázky v testu, typ odpovědi, stav odpovědi, maximální počet bodů (pouze v semestrálním testu a ve zkoušce). Je potřeba zvýraznit informaci o stavu odpovědi tak, aby na první pohled bylo zřejmé, zda je odpověď uložená.
Kategorizace: Funkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Vysoká

Název: Vyplnění a uložení odpovědi v testu
Zkratka: F7
Popis: Systém umožní studentovi vyplnit odpověď podle typu odpovědi. Systém podporuje tyto typy odpovědí: text, check box, radio list, diagram, RA, SQL, transformace a normalizace. Student může svou odpověď uložit a u typu RA a SQL může ověřit svůj dotaz.
Kategorizace: Funkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Vysoká

Název: Ukončení testu
Zkratka: F8
Popis: Student může odevzdat test předčasně. Pokud tak neučiní, test je automaticky ukončen při vypršení časového limitu.
Kategorizace: Funkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Nízká

Název: Potvrzení ukončení testu
Zkratka: F9
Popis: Pokud student sám ukončí test před vypršením limitu, je vyžadováno, aby student musel ukončení testu potvrdit. Při tomto potvrzení bude student upozorněn, pokud bude mít v testu nevyplněné nebo neuložené odpovědi.
Kategorizace: Funkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Střední

Název: Obsah při zobrazení opraveného testu
Zkratka: F10
Popis: Při zobrazení opravené otázky v demotestu se má zobrazit zadání, úkol a odpověď. U semestrálního testu a zkoušky se má zobrazovat pouze úkol a odpověď.
Kategorizace: Funkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Nízká

Název: Pohyb po portálu při psaní testu
Zkratka: F11
Popis: Pokud student vyplňuje test, systém mu neumožní opustit záložku testů. Teprve po ukončení testu se může opět volně pohybovat v portálu.
Kategorizace: Funkční požadavek, Should have
Priorita: Střední
Předpokládaná obtížnost: Střední

Název: Zobrazení detailu testu ihned po ukončení
Zkratka: F12
Popis: Detail opraveného demotestu si student může zobrazit ihned po ukončení demotestu, avšak detail semestrálního testu a zkoušky je studentovi zpřístupněn teprve po uplynutí limitu na daný termín. Bude se tak předcházet poskytování opravených odpovědí mezi studenty ještě během testu.
Kategorizace: Funkční požadavek, Should have
Priorita: Střední
Předpokládaná obtížnost: Nízká

Název: Přihlášení na ústní zkoušku
Zkratka: F13
Popis: Pokud student úspěšně absolvoval zkoušku, může se přihlásit na zkoušku ústní. Pokud se přihlásil, ale rozmyslel si to, může se také zpětně odhlásit. Systém umožňuje přepínání mezi přihlášením a nepřihlášením.
Kategorizace: Funkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Nízká

Název: Volba technologií
Zkratka: N1
Popis: Testová část FE musí být začleněna do nového projektu, a musí tedy využívat stejné technologie.
Kategorizace: Nefunkční požadavek, Must have
Priorita: Vysoká
Předpokládaná obtížnost: Nízká

Název: Responzivita
Zkratka: N2
Popis: Webová aplikace by měla být responzivní, a uživatelské prostředí by mělo být přizpůsobeno také zobrazení v mobilním zařízení.
Kategorizace: Nefunkční požadavek, Should have
Priorita: Střední
Předpokládaná obtížnost: Střední

Název: Vícejazyčnost
Zkratka: N3
Popis: Systém podporuje český a anglický jazyk. Proto i testová část musí obsahovat překlady pro tyto jazyky.
Kategorizace: Nefunkční požadavek, Should have
Priorita: Střední
Předpokládaná obtížnost: Nízká

Název: Údržba a rozšiřitelnost
Zkratka: N4
Popis: Frontend bude používat znovupoužitelné komponenty. Tyto komponenty by měly být přizpůsobeny potřebám celého systému a měly by být lehce rozšiřitelné.
Kategorizace: Nefunkční požadavek, Should have
Priorita: Střední
Předpokládaná obtížnost: Střední

1.8 Výstup analýzy

Zde shrnu nejdůležitější poznatky z této kapitoly. Po důkladné analýze frontendu testové části pro studenty jsem zjistila klady a zápory současného stavu. Identifikovala jsem tedy nejzávažnější nedostatky a v návaznosti na tuto analýzu jsem stanovila požadavky na nový FE testů. Hlavním požadavkem je možnost zobrazení opraveného semestrálního testu a zkoušky studentem, což v současném softwaru není možné. Stěžejním problémem v současném stavu je odevzdávání testů. Odevzdání nyní probíhá bez potvrzení studentem, a často tak dochází k omylům ze strany studenta. V rámci analýzy jsem také zjistila aktuální stav BE a FE, bohužel oba projekty nejsou naplno funkční, a tedy se tyto aspekty projeví také v realizační části.

Kapitola 2

Návrh

V této kapitole nejdříve představím volbu technologií pro modelování wireframů. Poté popíši zásady pro vytváření uživatelského prostředí, a to pomocí Nielsonovy heuristiky. Poté čtenáře seznámím s novými návrhy frontendu, a také s průběhem jejich vytváření. Podrobně rozeberu změny jednotlivých verzí, až nakonec představím konečné návrhy UI.

2.1 Volba technologií pro návrh wireframů

Důležitým krokem procesu tvorby frontendu je modelování uživatelského rozhraní pomocí wireframů. Tento model určuje, co a jak bude stránka zobrazovat. Výhodou je možnost se na stránku podívat a určit nedostatky a nevhodně zvolené grafické prvky. Wireframy lze upravovat až do chvíle, kdy je návrh dostačující. Pro vývojáře je pak snazší a rychlejší implementovat přesně podle této předlohy. [19]

Balsamiq wireframes

Balsamiq wireframes je nástroj pro modelování wireframů. Je to placená aplikace, která však nabízí 30denní verzi zdarma. Systém obsahuje celou řadu předem připravených prvků, které uživatel může využít. Jedná o Lo-Fi prototyp, který je oproti Hi-Fi méně přesný. Uživatel tak má spíše abstraktní představu o následné stránce. [20]

S tímto systémem jsem se poprvé seznámila v SP1, kde jsme společně s týmem vytvářeli modely pro testovou část studentů. Práce v tomto systému byla jednoduchá, ale mě osobně nevyhovoval černobílý nepřesný vzhled modelů.

Figma

Figma je novějším nástrojem pro navrhování wireframů. Je to cloud-based nástroj, což znamená, že návrhy nejsou vytvářeny offline lokálně na počítači, jako je to u většiny podobných softwarů, ale úpravy probíhají online. Do počítače jsou většinou ukládány až hotové projekty. [21]

Návrhy ve Figmě jsou velice přesné, jedná se o Hi-Fi prototyp, který působí realističtěji [20]. Hotové návrhy je dokonce možné si splést s reálnou aplikací. Detailní zobrazení prvků ale přináší nevýhodu oproti dříve zmíněnému Balsamiqu, a tím je delší doba vytváření návrhů. Návrhář se tedy musí rozhodnout, zda mu za déle strávený čas u wireframů stojí detailnější a realističtější výsledek.

Velkou výhodou Figmy jsou znovupoužitelné komponenty, které si návrhář může vytvořit. Pokud je jeden prvek využíván na více místech, vložíte jen vytvořenou komponentu, a pokud budete provádět úpravy nad komponentou, změní se i na použitých místech. Základní verzi lze využívat zdarma, což je velkou výhodou, ale k dispozici je také rozšířená placená verze. Další výhodou Figmy je snadná spolupráce v týmu. Je umožněno snadné sdílení, a také na projektu může pracovat více lidí najednou.

S tímto nástrojem jsem se seznámila při návrhu osobní stránky. Velice se mi zalíbil reálný vzhled i celý koncept znovupoužití komponent.

Vybraný nástroj

Přesný návrh wireframů podle mě velmi usnadní proces vývoje, a proto jsem se rozhodla návrhy vytvořit ve Figmě. Jedná se o nástroj, který je v dnešní době velmi využívaný, a pro člověka, který se zajímá o frontend a UI je dobré ovládat moderní dostupné nástroje. I proto jsem se zvolila právě Figmu.

2.2 Heuristika Jakoba Nielsena pro návrh uživatelského rozhraní

Vzhled webových stránek nebo aplikací má větší dopad na uživatele, než se může zdát. Nejde jen o to, zda se uživateli aplikace líbí, ale také o fakt, zda je pro něj jednoduché se v aplikaci orientovat a zda je pro něj používání snadné a intuitivní. Jacob Nielsen (1994) v jeho heuristice [22] popisuje 10 obecných zásad pro návrh dobrého uživatelského rozhraní.

- Uživatelé by měli být neustále informováni o tom, co se děje.
- Měla by se použít mluva uživatelům vlastní.
- Uživatelé mohou omylem provést nechtěnou akci, proto by v celém systému mělo být umožněno přerušit akci a navrátit se zpět.

- Uživatelské rozhraní by mělo být konzistentní.
- Návrh by měl předcházet chybám a problémům.
- Uživatelé by si neměli muset pamatovat informace z jiných částí systému. Potřebné informace by měl mít uživatel vždy k dispozici na dané stránce.
- Používání zkratek může urychlit práci zkušeným uživatelům.
- Minimalistický design je přehledný. Uživateli by se neměli zobrazovat informace navíc.
- Chybové hlášky by měly být napsány srozumitelně a ne pomocí kódu.
- Dokumentace a jiné pomocné nástroje mohou sloužit jako nívod na použití.

Některé body jsou natolik primitivní, až by se mohlo zdát, že jsou samozřejmé. Nicméně právě uvědomění si těch samozřejmých principů může být velice přínosné. S heuristikou jsem se seznámila až v průběhu vytváření návrhů. Zpětně jsem se tedy snažila nalézt nedostatky, na které by heuristika poukazovala. Zjistila jsem, že vytvořené návrhy byly v souladu s principy a tím jsem si ujistila, že návrhy budou v závěru kvalitní.

2.3 Vytváření návrhů

Jak už bylo zmíněno v předchozích kapitolách, testovou částí a především UI testů jsem se zabývala už během výuky předmětů SP1 a SP2. V práci představím některé návrhy z této doby. Tyto návrhy nemusí být v souladu s nasbíranými požadavky v této práci. Tyto modely jsem však do práce zahrnula z toho důvodu, aby bylo znatelné, z čeho jsem vycházela a zároveň, jak moc jsem se posunula.

Návrhy vznikající za cílem mé BP jsem vytvářela po verzích. Tedy vytvořila jsem návrh, konzultovala s vedoucím práce a společně jsme identifikovali nevýhody daného řešení a nastínili řešení více vyhovující. Takto jsem pracovala v iteracích, dokud jsem nedospěla k finálním návrhům. Již proces vytváření návrhů by se tedy dal označit za určitý způsob testování.

Přesto, že je v zadání stanoveno vytvoření frontendu pro testovou část studentů, můj úkol byl daleko rozsáhlejší. Frontend studentských testů je jednou z prvních částí aplikace, která vzniká. A proto mnohá rozhodnutí, která jsem uskutečnila, nebyla pouze pro část testovou, ale pro celý budoucí FE.

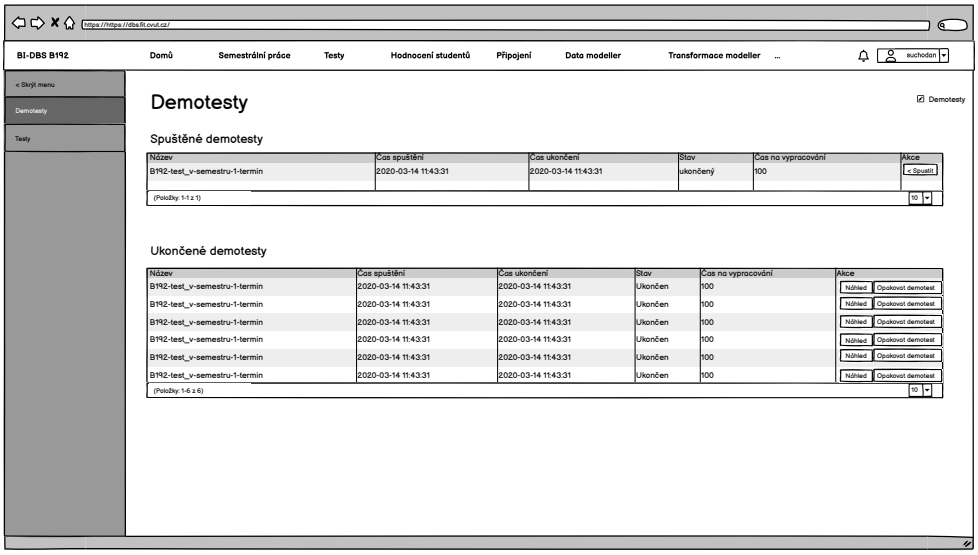
V rámci této BP vzniklo přes 60 obrazovek návrhů. Z důvodu takového rozsahu jsem se rozhodla v práci uvést pouze zástupce reprezentující největší změny. Všechny návrhy si pak můžete prohlédnout v přiloženém médiu, nebo po dobu podpory Figmy také **zde v odkazu**.

2.3.1 Zobrazení studentských testů

1. verze zobrazení studentských testů byla vytvořena už v rámci předmětů SP1 a SP2. Onen návrh byl vytvořen v dříve zmíněném Balsamiqu. Ostatní verze už byly navrženy za cílem obsahu BP.

1. verze

Jak můžete vidět na obrázku 2.1, tato verze spočívala pouze v překreslení současného stavu. Během návrhů jsme s týmem v této části nebyli vedeni k žádným změnám, a tak jsme považovali nynější stav za dostačující. Jedinou novinkou bylo zrušení modálních oken pro náhledy testů. To více popíši v sekci o zobrazení opravených testů.

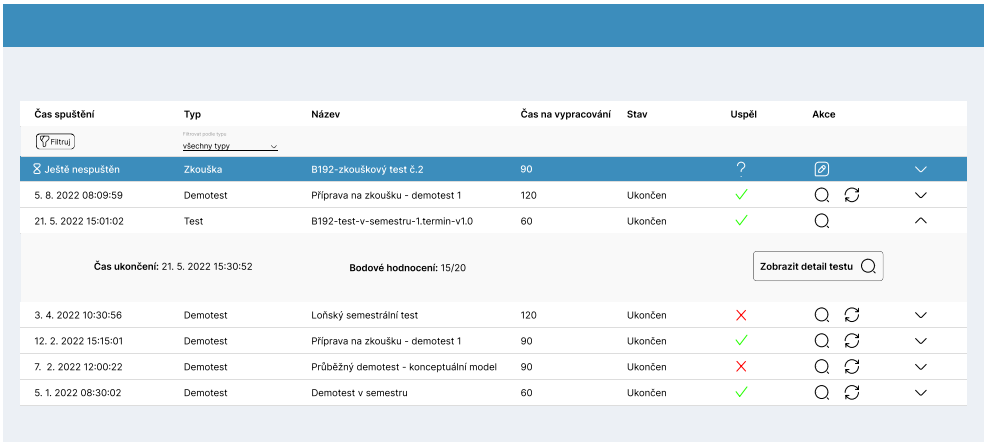


Obrázek 2.1: Tabulka demotestů v Enterprise Architect 1. verze

Když jsem pokračovala v návrzích, po zvolení tématu BP, překreslila jsem stejný návrh do Figmy. Přestože jde o stejný návrh, Figma díky barevnému realistickému vzhledu působí lépe.

2. verze

Po konzultaci s vedoucím práce jsem se rozhodla vydat jiným směrem. Byl zvolen přístup jediné stránky, kde se zobrazí všechny typy testů (demotesty, semestrální testy i zkoušky). Pomocí vyhledávání a filtrování v tabulce si student jednoduše může zobrazit přesně to, co požaduje. Navíc vše bude na jednom místě, a nebude potřeba se překlikávat pomocí bočního menu. Aktivní testy ke spuštění jsou odlišeny podbarvením celého řádku pro upozornění uživatele.



Obrázek 2.2: Tabulka testů 2. verze

Ve sloupcích tabulky jsem se pomocí barevných ikonek rozhodla zobrazit informaci, zda student v testu uspěl. Kliknutím na celý řádek se řádek rozbalí a student zde najde podrobnější informace o testu: Čas ukončení, bodové hodnocení, popř. body za ústní zkoušku u typu zkouška. Návrh 2. verze můžete vidět na obrázku 2.2.

Idea návrhu byla dobrá, ale posléze jsem zjistila, že podrobnějších informací o testu není tolik, a rozbalování řádku tedy za tímto cílem nemá smysl.

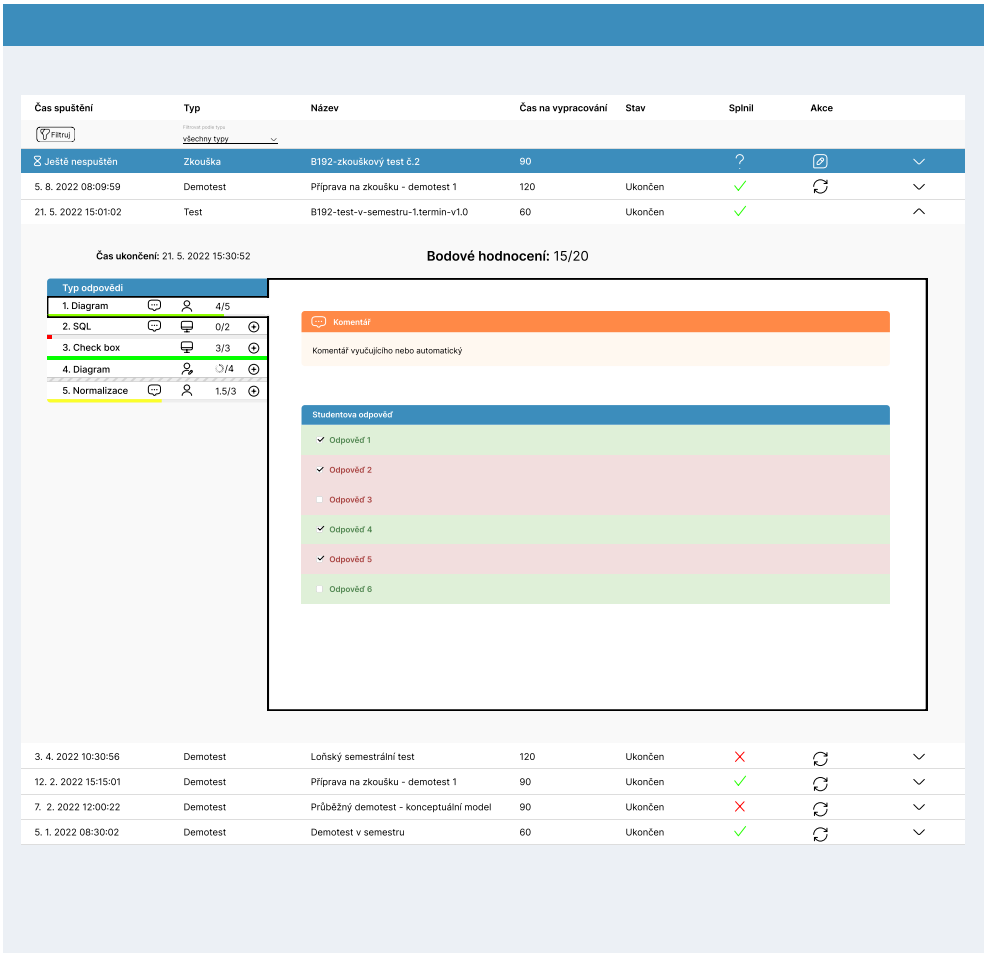
3. verze

V další verzi (obrázek 2.3) jsem se vydala podobným směrem. Celkový koncept rozbalení řádku je stejný, pouze v této verzi se řádek rozbaluje za účelem zobrazení celého detailu testu. Výhodou by bylo to, že by se celý přehled testů i náhled testu zobrazoval v jedné stránce. Tento návrh opět nebyl vyhovující, a to z důvodu nepřehlednosti. Kvůli zobrazení opravených testů, které vám představím dále v textu 2.3.1, jsem se dostala do situace, kde se rozbalovala tabulka v tabulce. A to především na mobilních zařízeních není přehledné.

4. verze

V tomto návrhu jsem se oprostila od rozbalování řádků. Náhledy opravených testů jsem se rozhodla zobrazovat po rozkliknutí řádku na nové stránce. Návrh můžete vidět na obrázku 2.4

Kromě rozložení stránek jsem začala řešit detailnější věci, jako například bodové hodnocení. U demotestu se studentovi bude zobrazovat procentuální úspěšnost, zatímco u semestrálních testů a zkoušek bodové hodnocení ve formátu *počet získaných bodů / počet maximálních bodů*. Zvolila jsem podbarvení textu barvou vypočítanou z procentuální úspěšnosti na škále od červené,

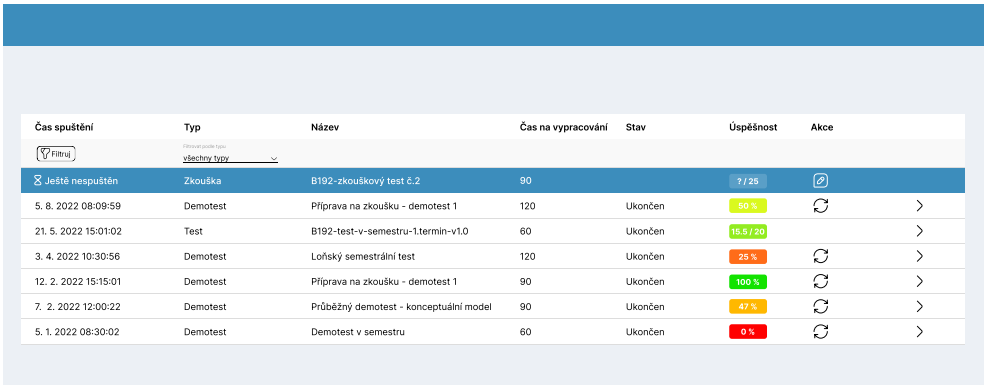


Obrázek 2.3: Tabulka testů 3. verze

přes žlutou, až k zelené. V tomto řešení jsem později nalezla nedostatek, a to ten, že studenta hlavně zajímá informace, jestli uspěl nebo neuspěl, což z mého návrhu nebylo jasně poznat. A tak se dostáváme k finálnímu návrhu.

Finální návrh

Problém z předchozího návrhu, týkající se zobrazení úspěšnosti, jsem vyřešila následovným způsobem. U **semestrálního testu** a **zkoušky** se text podbarví zelenou barvou, v případě splnění minimálního počtu bodů, a v případě neúspěchu pak barvou červenou. Podbarvení může být ještě šedé, a to v případě, že test prozatím nebyl spuštěn nebo ohodnocen, a tedy nelze určit splnění bodových požadavků. U **demotestu** se zobrazí text černé barvy, který udává procentuální úspěšnost. Nakonec jsem nezvolila barevnou vari-

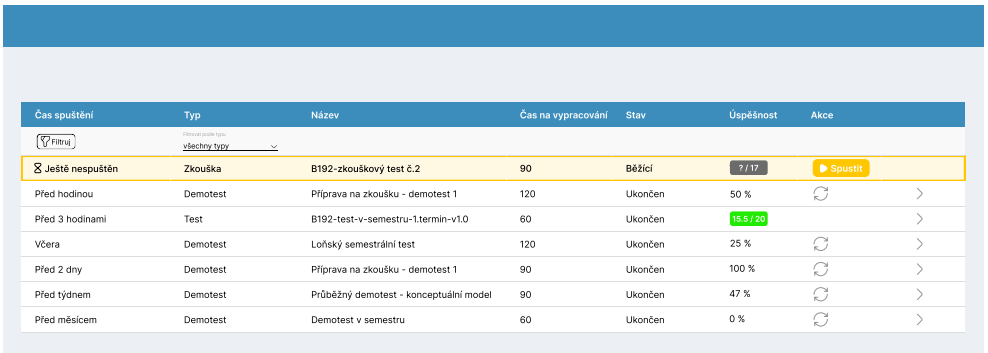


Obrázek 2.4: Tabulka testů 4. verze

antu, aby barev nebylo příliž. Barevné zvýraznění pouze bodovaných testů umožní přitáhnout pozornost na výsledky důležitých testů.

Do nového portálu jsem se také rozhodla přinést další barvu pro zvýraznění těch nejdůležitějších informací. Žlutá barva je dostatečně výrazná a v kombinaci s modrou krásně vyzdvihne důležité akce. Takto zabarvený je celý řádek aktivního testu a akce SPUSTIT TEST, popř. PŘIHLÁSIT SE NA ÚSTNÍ ZKOUŠKU. Tento návrh můžete vidět na obrázku 2.5

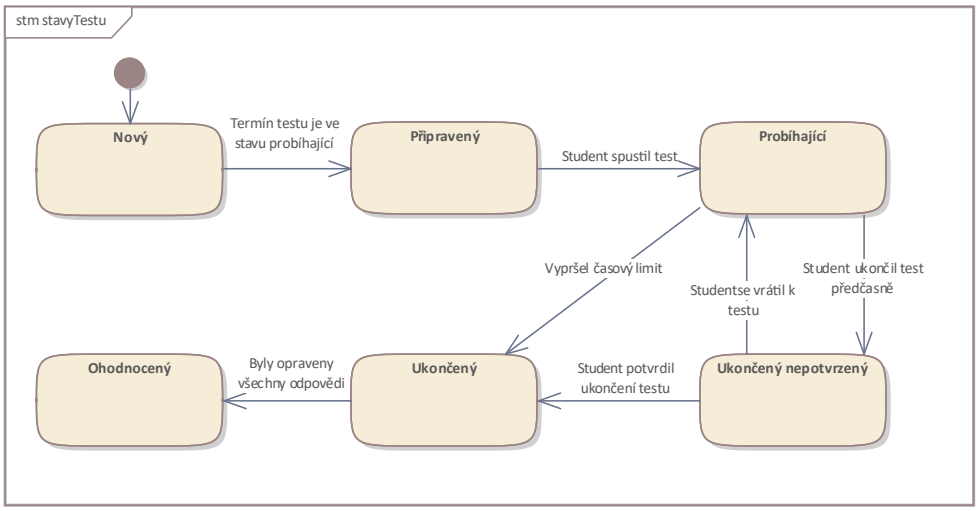
Pokud je studentovi přiřazen semestrální test nebo zkouška, řádky ostatních testů jsou *disabled* a nelze tedy ani opakovat demotest, ani přejít na náhledy testů.



Obrázek 2.5: Tabulka testů finální verze

Po dokončení zkoušky se na místě akce zobrazí tlačítko na přihlašování k ústní zkoušce. Tlačítko je *disabled* do té doby, než je test kompletně opraven. Poté se opětovným stisknutím tlačítka může přihlašovat a odhlašovat. Všechny možnosti různých stavů této tabulky naleznete v přiloženém médiu.

Na obrázku 2.6 můžete vidět stavový diagram znázorňující jednotlivé stavy testu.



Obrázek 2.6: Stavy testu

2.3.2 Náhled opravených testů

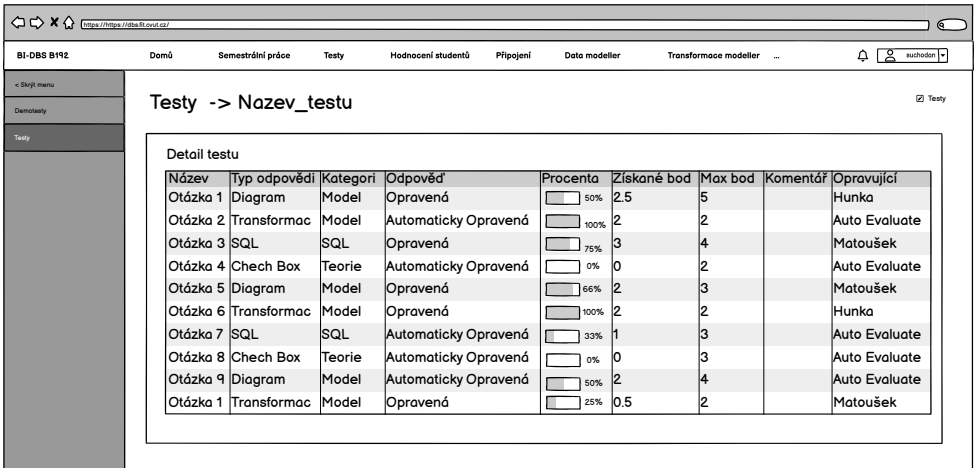
Touto částí jsem se opět zabývala již v době, kdy jsem na portálu pracovala v rámci předmětů SP1 a SP2. Na rozdíl od zobrazování testů jsem se už tehdy více ponořila do problematiky přehlednosti a intuitivního používání, a pod vedením vedoucího projektu jsem přišla s novými návrhy, které popíši ve verzi 1 a 2.

1. verze

Novinkou oproti současnému stavu je kompletní zrušení modálních oken. Místo toho se detail testu otevírá na nové stránce.

Tabulka náhledu **demotestu** je totožná se současným stavem, tedy nachází se zde sloupce: název otázky, typ odpovědi, kategorie a akce. Každá otázka jde nově kliknutím rozbalit a pod daným řádkem se zobrazí detail demotestu (tedy zadání, úkol a odpověď).

Tabulka **semestrálního testu** a **zkoušky** se taktéž otevírá v novém okně. Zde se do tabulky přidal nový sloupec, který graficky ilustruje procentuální úspěšnost dané otázky. Zatím opět nelze zobrazit detail semestrálního testu, ani zkoušky. Obrázek 2.7



Obrázek 2.7: Náhled opraveného semestrálního testu 1. verze

2. verze

Už během SP předmětů jsme společně s vedoucím projektu uznali, že předchozí řešení nebylo vyhovující. Také bylo definitivně rozhodnuto, že se studentům bude zobrazovat detail semestrálních testů a zkoušek. Přišla jsem tedy s novým návrhem, který umožňuje konzistentní zobrazení všech typů testů. Tento návrh vykreslí jak tabulku otázek, tak i zobrazení detailu otázky na jednu stránku. Toto řešení má přinést lepší přehlednost otázek bez použití modálních oken nebo rozbalování řádků.

Stránka je rozdělena na 2 pomyslné části. Nalevo se zobrazuje tabulka s informacemi o jednotlivých otázkách a napravo je velký box pro obsah aktivní otázky. Výchozí stav zobrazuje detail 1. otázky v testu.

Místo sloupců: název otázky, typ odpovědi a kategorie je studentům zobrazeno pouze pořadí otázky společně s typem odpovědi. Informace ze současné tabulky jsem se snažila snížit na potřebné minimum tak, aby tabulka vlevo mohla být co nejmenší, a v pravé části tak bylo dostatečné místo na detail testu. Další stěžejní informací v tabulce je opravující dané otázky. Tato informace je zobrazena pomocí ikonek. Otázka může být vyhodnocena automaticky, a v tom případě student uvidí ikonu počítače. Pokud otázku opravil vyučující, zobrazí se mu ikona uživatele, a pomocí *tooltipu* (zobrazení textu při najetí na element) poté jméno učitele. U náhledu semestrálního testu a zkoušky může být navíc oproti demotestu ikonka komentáře, která studenta upozorní, že odpověď obsahuje komentář.

Zobrazení úspěšnosti se liší podle typu testu. U demotestu se zobrazují procenta úspěšnosti (např. 50%), a u semestrálního testu a zkoušky pak počet získaných bodů a počet maximálních bodů (např. 2/3). Úspěšnost jsem se roz-

hodla zobrazit také pomocí progress baru. Ten je zabarven podle procentuální úspěšnosti na barevné škále od červené, přes žlutou, až po zelenou barvu. Tedy při 100% správné odpovědi je zobrazen zelený plný pruh. Pokud byla odpověď správná z 50%, pruh má poloviční šířku a žlutou barvu, a při kompletně špatné odpovědi je nejužší červený proužek. Takto navržený design má studenta na první pohled informovat o úspěšnosti.

V hlavičce pravé části vedle tabulky jsou zopakované informace jako název otázky, počet bodů a opravující. Níže už je zobrazena opravená odpověď studenta, popřípadě komentář učitele. Znázorněním studentské odpovědi a dalšími části pravého boxu se budu více zabývat v dalších verzích.

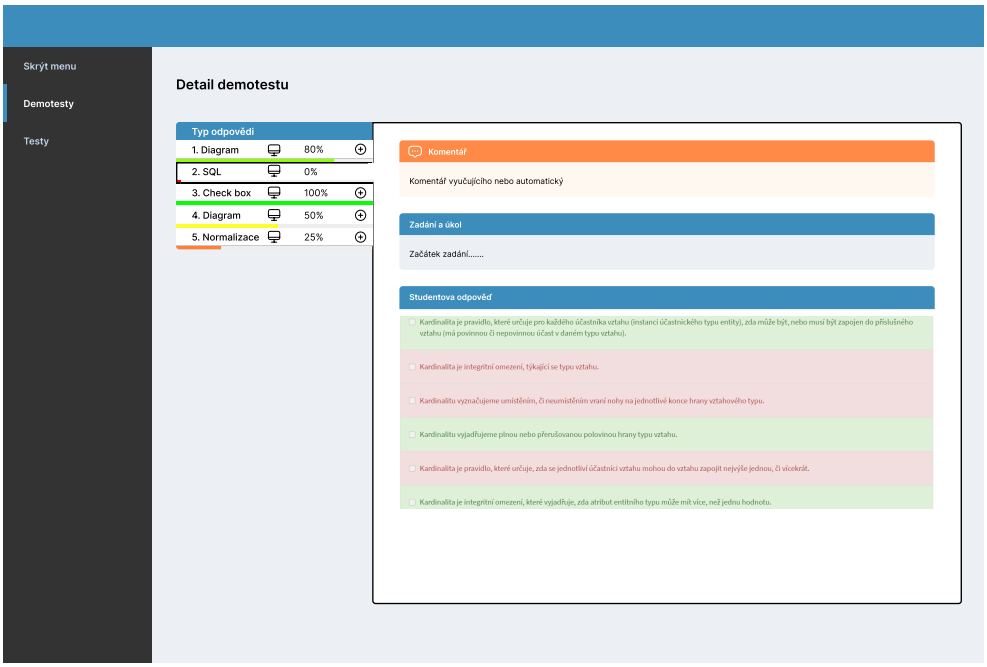
Obrázek 2.8: Náhled opraveného testu 2. verze

Tento návrh poskytl solidní základ pro finální verzi, a to především v rozvržení stránky. Návrh 2. verze si můžete prohlédnout na obrázku 2.8. Obsahuje ale mnoho nedostatků, které jsem v dalších verzích odstranila.

3. verze

Hlavní myšlenkou další verze bylo ještě více zredukovat tabulku v levé části, aby prostor pro zobrazení odpovědi byl co největší. Z řádku tabulky jsem odstranila progress bar s úspěšností, a ve sloupci zůstaly pouze procenta nebo body (podle typu testu), které nezabírají tolik místa. Progress bar je zobrazen pod řádkem dané otázky jako tenký proužek zabarvený stejným způsobem, jako v předchozí verzi. Opět se mění jeho šířka, a to maximálně na šířku celého řádku (v případě 100% úspěšnosti). Na obrázku 2.9 je vidět náhled demotestu.

Poprvé jsem začala řešit situaci, kdy některá z otázek zatím není opravená. Takový řádek je podbarven šedou barvou, progress bar je zatím prázdný a místo bodů je studentovi zobrazena pomlčka. Všechny tyto známky mají studentovi naznačit, že otázka je ještě v opravě.



Obrázek 2.9: Náhled demotestu 3. verze

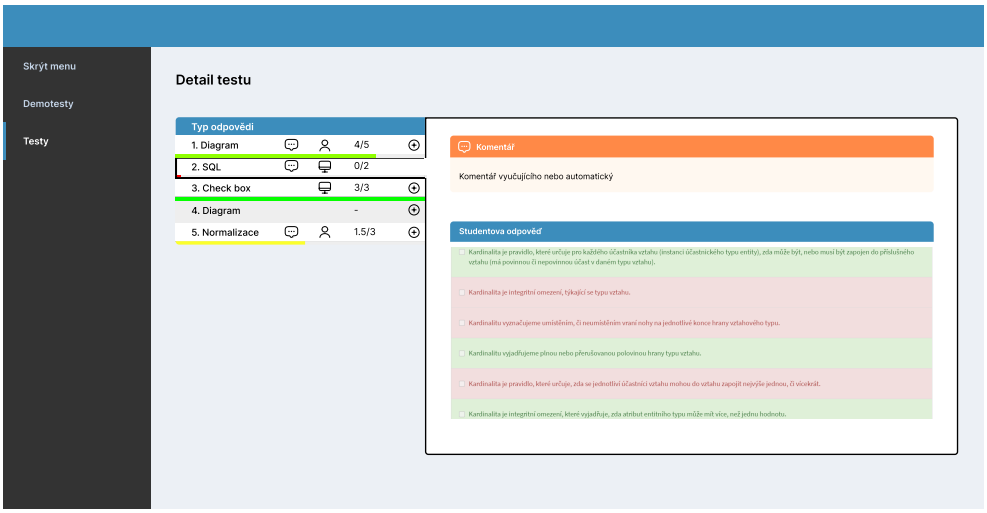
Změny jsou také v pravé části, a to hned v hlavičce. Odstranila jsem duplicitní informace (opravujícího, body a typ odpovědi), které student může nalézt v tabulce. Náhled bodovaného testu je na obrázku 2.10.

Jasněji jsme si s vedoucím práce definovali, co se studentům bude/nebude zobrazovat u demotestů a semestrálních testů / zkoušek. Konečný verdikt je takový, že u **demotestu** se zobrazuje zadání, úkol a odpověď. Zadání a úkol se zobrazí ve sbalené podobě, aby student viděl primárně svoji odpověď. V **semestrálních testech** a **zkouškách** se zobrazuje pouze studentova odpověď, popřípadě komentář vyučujícího.

Jednotlivé části, jako zadání, úkol, odpověď studenta a komentář, jsou v boxech. Především komentář učitele je zvýrazněn barvou, protože znázorňuje důležitou informaci.

4. verze

V další verzi jsem upravila znázornění doposud neopravené otázky. Šedé podbarvení by mohlo evokovat *disabled* řádek, a tak jsem přišla s novým návrhem. To, že otázka nebyla opravena automaticky, znamená, že bude určitě opravena učitelem (jelikož neprošla automatickou opravou). Ikona opravujícího tedy bude ikona osoby s tužtičkou. Místo dosažených bodů bude pouze ikona načítání. Progress bar je zbarven šedě proužkovaně. Tato kombinace jasně



Obrázek 2.10: Náhled semestrálního testu 3. verze

dává najevo fakt, že otázka není opravená. Zároveň jde poznat, že se student na odpověď může podívat.

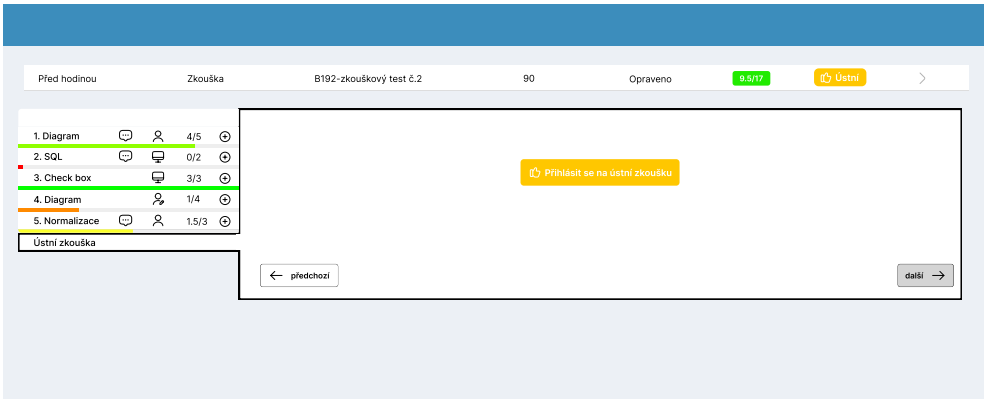
Obrázek 2.11: Náhled semestrálního testu 4. verze

Dále jsem si uvědomila že v předchozích verzích studentovi nezobrazuji žádné informace o testu jako celku, a tedy podle stránky náhledu nezjistí, na který test se dívá. Proto jsem nad tabulkou zobrazila alespoň název aktivního testu, jak je vidět na obrázku 2.11. Zobrazením jednotlivých typů odpovědí jsem se zabývala až v samotném závěru návrhu této části, a tak je představím až ve finální verzi.

Finální návrh

Zobrazení informací o samotném testu bylo v předchozích verzích nedostačující. Proto jsem se rozhodla ve vrchní části náhledu testu zobrazit řádek z tabulky přehledu všech testů. Tímto řešením jsem umožnila studentovi mít všechny potřebné informace pohromadě, a docílila tak konzistentního návrhu.

U typu zkouška se přidává do tabulky otázek jeden řádek pro ústní zkoušku. Po rozkliknutí se uživateli umožní přihlásit se / odhlásit se na ústní zkoušku, pokud jsou všechny otázky opravené (obrázek 2.12). V opačném případě je student informován o tom, že na ústní zkoušku se může přihlásit teprve po opravení testu.

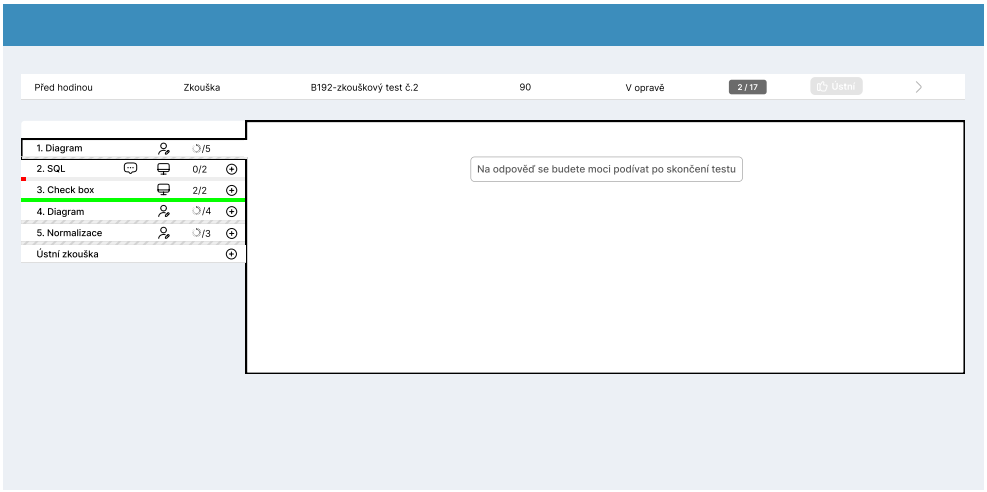


Obrázek 2.12: Náhled zkoušky finální verze – přihlášení na ústní zkoušku

Zaměřila jsem se také na zatím nezmiňovaný aspekt, a to zobrazení detailu testu a otázek přímo po odevzdání testu. Pokud student odevzdá test před vypršením limitu, ostatní studenti stále mohou pracovat na svých testech. Snažíme se předejít podvodům, a tak studentům bude umožněno podívat se na jejich odpovědi až v momentě, kdy test skončil pro všechny studenty. Do té doby je o tomto omezení student informován v detailu otázky v pravé části. Obrázek 2.13.

Zobrazení opravené odpovědi podle typu odpovědi

Zobrazení jednotlivých typů opravených odpovědí je velmi podobné jako v současném systému, a tedy je zde nebudu představovat. Jsou však obsaženy v příloženém médiu.



Obrázek 2.13: Náhled zkoušky finální verze – nelze zobrazit odpověď

2.3.3 Psaní testů

Psaním testů se v předmětech SP zabývali zbylí členové našeho týmu. Tato část prošla během návrhu obrovskými změnami, které nyní čtenářům popíši opět v jednotlivých verzích.

1. verze

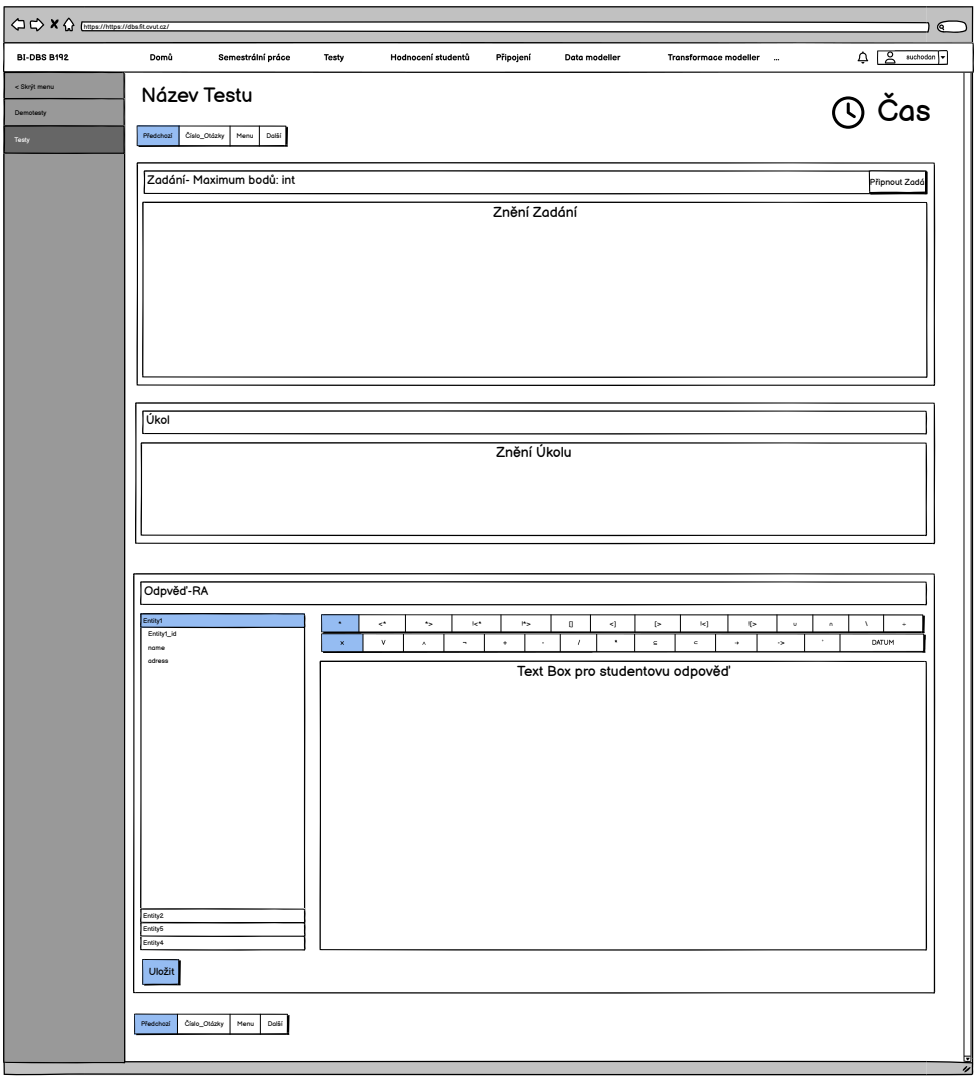
Jak jsem zmínila v předešlém odstavci, prvotní návrhy psaní testů, které byly vytvořeny v předmětech SP1 a SP2, nejsou mým dílem. Převzala jsem je pouze pro ukázku toho, odkud jsem začala.

Návrh odpovídá současnému stavu systému. Tedy po spuštění testu se zobrazí souhrnná tabulka otázek. Odtud lze přejít na vyplnění každé otázky, u které je zobrazeno zadání, úkol a odpověď, lišící se podle typu odpovědi. Studentovi je zobrazen odpočet času a na každé otázce je také navigace testu. Na obrázku 2.14 je vidět vyplňování odpovědi typu RA.

2. verze

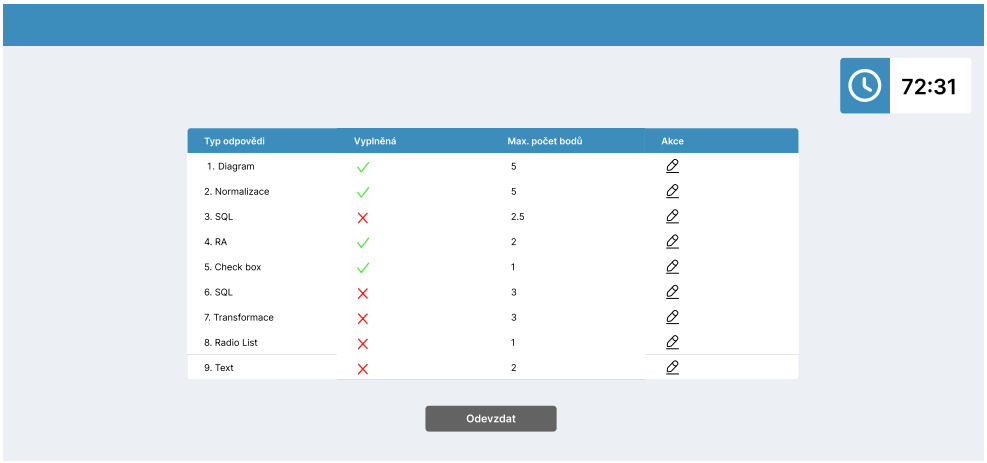
V druhé verzi jsem nejdříve zredukovala sloupce tabulky otázek. Stejně jako v předchozích náhledech testů se místo názvu otázky, typu odpovědi a kategorie zobrazuje číslo otázky a typ odpovědi společně. Barevnými ikonami je zvýrazněna informace, zda je otázka vyplněná, nebo ne, protože pro studenty je daná informace důležitá. Dále je zobrazen maximální počet bodů a akce pro přechod na vyplnění otázky (obrázek 2.15).

Vyplňování otázek je pak velice podobné předchozí verzi. Opět zde najdete odpočet času a navigaci, zadání, úkol a odpověď.

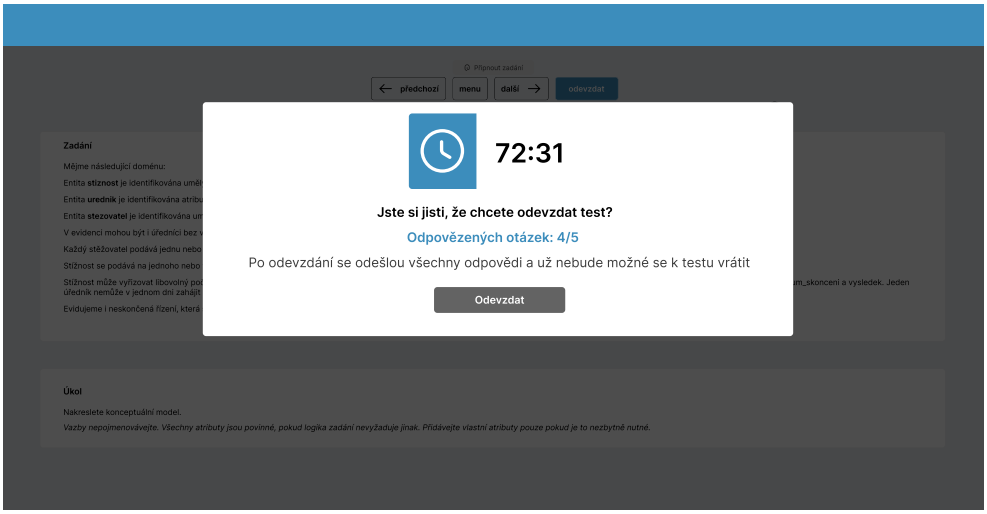


Obrázek 2.14: Psaní testu 1. verze – RA odpověď

Už v analýze jsem nastínila, že je velký problém s odevzdáváním testů. Proto jsem přišla s řešením, které po stisknutí tlačítka ODEVZDAT zobrazí modální okno, kde bude student znovu dotázán, zda opravdu chce test odevzdat. Rozhodla jsem se studenty informovat i o tom, kolik otázek mají zodpovězených, aby náhodou neodevzdali test s nevyplněnými odpověďmi. Student může test odevzdat, nebo se vrátit k vyplňování testu. Modální okno můžete vidět na obrázku 2.16



Obrázek 2.15: Psaní testu 2. verze – tabulka otázek

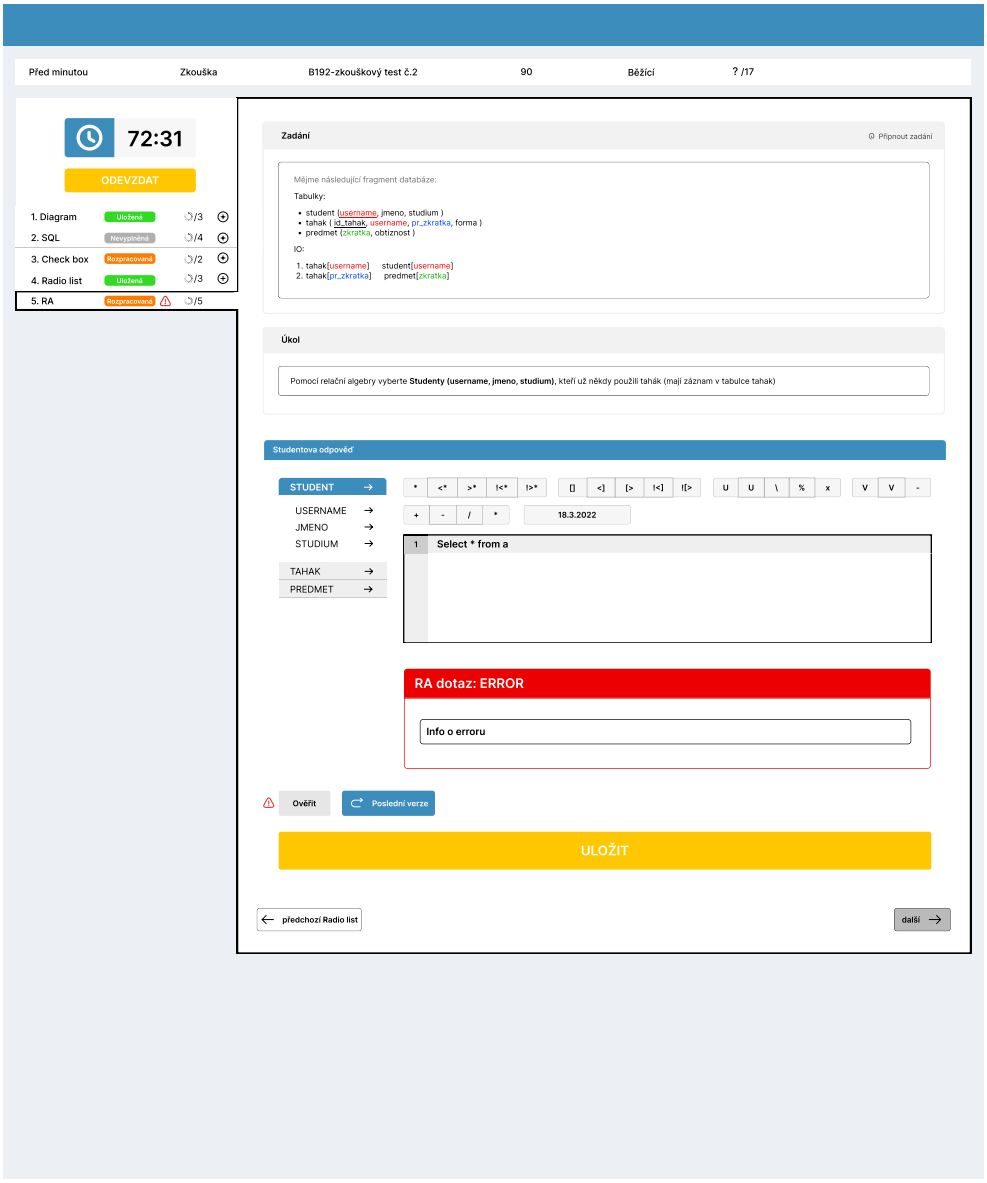


Obrázek 2.16: Psaní testu 2. verze – potvrzení před ukončením testu

Finální návrh

Částí psaní testu jsem se zabývala až jako poslední, což se nakonec ukázalo být výhodou. Když jsme s vedoucím práce procházeli wireframy pro náhledy opravených testů, uvědomili jsme si, že tento návrh, který byl pečlivě a detailně upravován podle našich potřeb je vlastně ideálním řešením také pro psaní testů. Nebyl tedy důvod vymýšlet jiné řešení, pouze bylo potřeba upravit návrhy náhledu testů. Ve vrchní části je opět pruh s informacemi o spuštěném testu. Rozložení na levou tabulku s otázkami a pravou část s obsahem lze

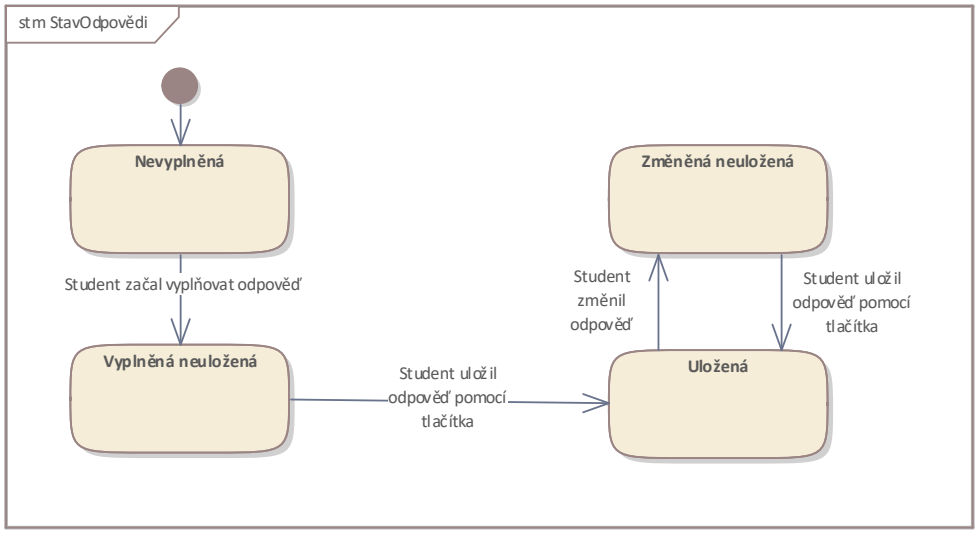
jednoduše aplikovat i zde. Navíc tím bude docíleno konzistentního návrhu frontendu, který bude pro studenty příjemný. Toto rozložení můžete vidět na obrázku 2.17



Obrázek 2.17: Psaní testu finální verze

Levá tabulka obsahuje pořadí a typ otázky, stav odpovědi a bodové hodnocení. Je jasné, že během psaní testu otázka není ohodnocená, proto na místě dosažených bodů je ikonka načítání. Studenta ale v testu zajímá informace, za kolik je daná otázka bodů. A třeba se pak více zaměří na otázky za více bodů.

Nad samotnou tabulkou se zobrazuje odpočet času a výrazné žluté tlačítko pro odevzdání testu. Tato levá část bude bude podporovat smart scrollování, a tedy bude neustále na očích.



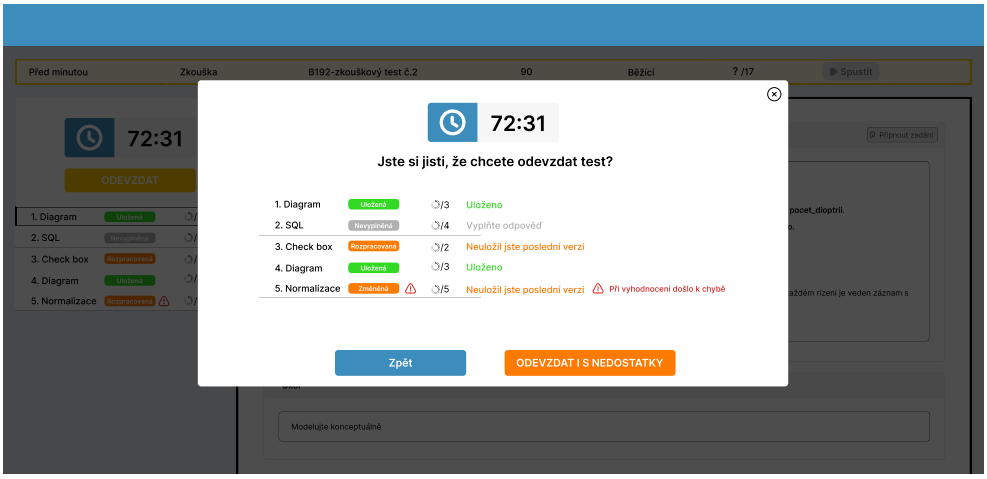
Obrázek 2.18: Stavy odpovědí

Nyní vysvětlím jednotlivé stavy odpovědi. Prozatím není naimplementováno průběžné ukládání studentské odpovědi, a proto student musí své odpovědi ukládat ručně. V předchozích verzích jsem u otázky rozlišovala pouze to, jestli je vyplněná nebo ne. Studentova odpověď ale může mít více stavů, které jsem se také rozhodla studentům znázornit. V levé tabulce otázek je tato informace zobrazena pomocí barevného obdélníčku s názvem stavu. Jak je vidět na stavovém diagramu na obrázku 2.18, odpověď je nejdříve NEVYPLNĚNÁ. V tom případě je zobrazen šedý rámeček. Pokud student začne pracovat na své odpovědi, ale odpověď neuloží, nachází se ve stavu VYPLNĚNÁ NEULOŽENÁ. Oranžová barva obdélníku má studenta upozornit, že by měl svou odpověď uložit. Pokud rozpracovanou odpověď uloží, stav odpovědi je ULOŽENÁ a obdélník zezelená. V posledním stavu se odpověď může objevit, pokud už byla uložená, ale student provedl nějaké změny, které opět neuložil. Tento stav (ZMĚNĚNÁ NEULOŽENÁ) pro studenty ale znamená stejnou informaci jako stav 2. Proto budou oba stavy neuložených odpovědí zvýrazněny stejnou oranžovou barvou. Během psaní testu by pak studenti měli docílit toho, že jsou všechny obdélníčky zelené, a tedy, že je vše uloženo. Na FE budou stavy odpovědí znázorněny jako NEVYPLNĚNÁ, ROZPRACOVANÁ, ULOŽENÁ, ZMĚNĚNÁ.

V pravé části je pak stejně jako v současném stavu nejdříve zadání, pak úkol, a nakonec odpověď. Pod odpovědí je opět výrazné tlačítko uložit, kterým

studenti ukládají svou odpověď. Kromě žluté barvy jsem tlačítko zvýraznila jeho šířkou, která se táhne přes celý box.

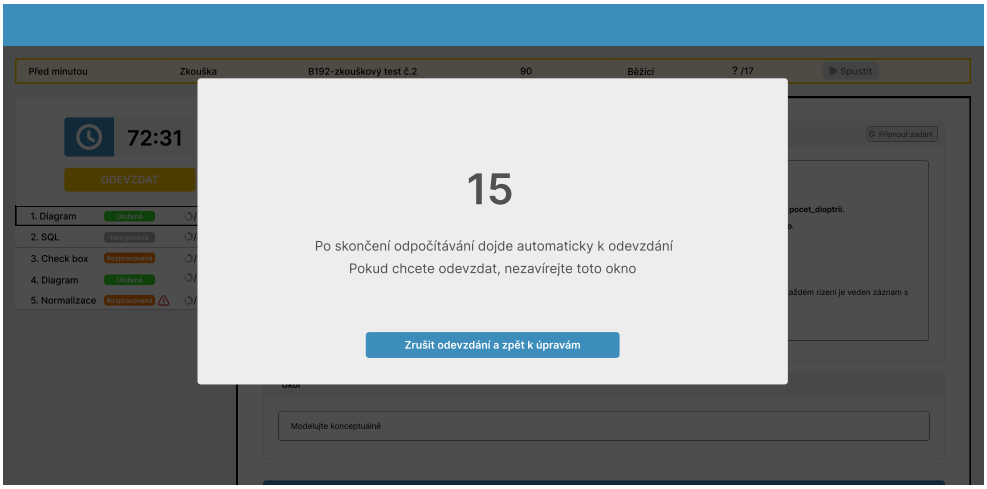
U typů odpovědí RA a SQL je zde také tlačítko ověřit, které provede daný dotaz. Reakce na provedení dotazu se nyní zobrazuje notifikacemi. Ideální by ale bylo dostat tuto informaci přímo k studentově odpovědi. Inspirovala jsem se tedy dotazováním u semestrálních prací, kde se pod odpovědí při provedení dotazu zobrazí buď zelený obdélník SUCCESS, nebo červený ERROR. Pokud při dotazu dojde k chybě, obdélník jde rozbalit a student zde najde informace o chybě. Kromě tohoto zobrazení chyby je studenovi zobrazen červený vykřičník. A to jak u tlačítka ověřit, tak i v levé tabulce na řádku dané otázky. Tento symbol je na obou místech z důvodu toho, aby student věděl, s čím si ho má v tabulce spojit. U studentské odpovědi se také nachází tlačítko, které umožní studentovi obnovit předchozí verzi odpovědi, popř. vrátit se k novější verzi. To je také novou funkcionalitou frontendu. Návrh je na obrázku 2.17



Obrázek 2.19: Psaní testu finální verze – první potvrzení při odevzdání

Stejně jako v předchozí verzi, i zde je studentovi po odevzdání testu umožněno navrátit se zpět k testu. V této verzi je však navrženo ještě s větší péčí, aby student neodevzdal test omylem, nebo s neuloženými odpověďmi. V modálním okně (obrázek 2.19) jsem se rozhodla zobrazit celou tabulku otázek. Především barevně zvýrazněné stavy odpovědí mají studenta upozornit na případné neuložení odpovědi. Na konci řádku tabulky je studentovi ještě slovně popsáno, proč je odpověď v daném stavu, a co by měl udělat, aby vše bylo v pořádku. Student se může buď vrátit k vyplňování testu, nebo test odevzdat. Na žádost vedoucího projektu jsem i po potvrzení odevzdání přidala další možnost úniku. I přesto, že byl student upozorněn na nedostatky, a přestože i po dotázání odevzdal test, stále má možnost vrátit se zpět. V modálním okně se mu po potvrzení odevzdání zobrazí odpočítávání s textem, který ho

informuje, že po skončení odpočítávání se test automaticky ukončí. Během 15 vteřin odpočtu se student může vrátit k rozpracovanému testu. Při této dvojité kontrole by student opravdu neměl ukončit test, aniž by tak zamýšlel. Modální okno s odpočtem je na obrázku 2.20

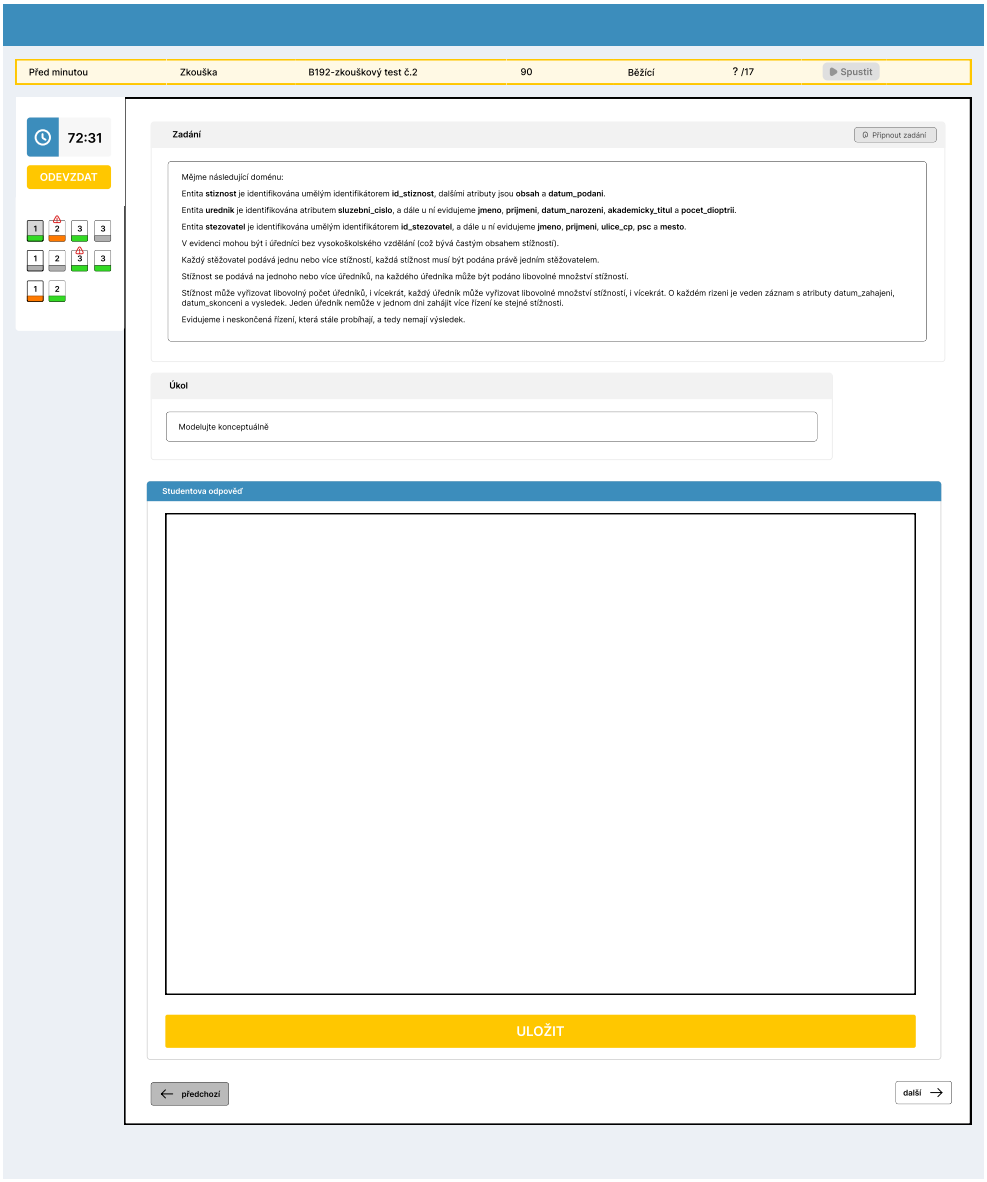


Obrázek 2.20: Psaní testu finální verze – druhé potvrzení při odevzdání

Snažím se, aby nové UI bylo co nejpříjemnější na používání. A tak by studenti během používání portálu neměli narazit na žádné překážky. Studenti většinou píší testy ve škole na obrazovkách s velkým rozlišením. Při zvláštních příležitostech (např. koronavirus) studenti vyplňují test na svých zařízeních z domu, stejně tak při zkušebních demotestech. Tato zařízení už mohou mít rozlišení menší. V takovém případě by se mohlo stát, že levá tabulka rozložení stránky by mohla zabrat tolik místa, že by box pro obsah nebyl dostatečně velký a vyplňování odpovědí by pro studenta nebylo přívětivé. Proto jsem přišla s alternativním návrhem, ve kterém se levá tabulka otázek společně s odpočtem času a tlačítkem na odevzdání zobrazí v nejmenší možné šířce. Jednotlivé otázky budou reprezentovány jako čtverečky s číslem otázky. Šedivě je zvýrazněna rozkliknutá otázka, spodní polovina je zabarvena podle stavu, ve kterém je odpověď (NEVYPLNĚNÁ, ULOŽENÁ, ROZPRACOVANÁ). Toto rozložení umožní studentům mít neustále na očích důležité informace a přitom jim nenaruší odpovídání na otázky, jak můžete vidět na obrázku 2.21

2.4 Technologie pro implementaci

V následující podkapitole popíši použité technologie pro implementaci. Většina byla předem určena základem projektu, ale podílela jsem se například na výběru knihovny Quasar a použití Composition API společně s Vue.js 3.



Obrázek 2.21: Psaní testu – zúžení tabulky

Vue.js 3

Vue.js je javascriptový framework pro vývoj uživatelského rozhraní a pro snadné sestavení Single Page Aplication. Vue aplikace je založena na opakovaně použitelných komponentách, které se do sebe vnořují, což umožnuje rychlý a efektivní vývoj a přehledný strukturovaný kód. Vue také sleduje změny stavu javascriptu, a pokud taková změna nastane, efektivně je aktua-

lizován DOM.

Vue.js 3 je nejnovější dostupná verze tohoto frameworku. Nabízí možnost využití *Composition API*, což je způsob psaní komponent pomocí funkcí namísto objektů, jako to bylo v *Options API* v předchozích verzích. *Composition API* usnadňuje psaní komponent, zápis je kratší a přehlednější, a tak i lépe čitelný. [23]

Pro potřeby porozumění implementační části zde popíši alespoň základní principy Vue. Jak byl zmíněno v předchozích odstavcích, klíčovým konceptem pro Vue jsou komponenty, které se do sebe mohou vnořovat. ***Props*** jsou vstupní data, která jsou předávána z rodičovské komponenty do komponenty vnořené. Ve vnořené komponentě jsou tyto *props* přístupné jako *atributy*, které mají název a definovaný typ. Opačný přístup, tedy pokud chci komunikovat z vnořené komponenty do rodiče, využívá principu ***emit***. Jedná se o událost, která informuje rodičovskou komponentu o nějaké změně. Této události se dají také přiřadit data, která se předají. V rodičovské komponentě se tato událost zachytí pomocí posluchače oné události [24].

Využívanou vlastností v komponentách jsou ***sloty***. *Slot* slouží pro zobrazení dynamického obsahu z rodičovské komponenty do komponenty vnořené. *Slot* se definuje ve vnořené komponentě, a v rodičovské komponentě je přiřazen obsah, který slot zobrazí. Takto se znovupoužitelné komponenty dají upravovat a plnit obsahem zvenčí, což umožňuje ještě více využít opakování obecných komponent.

Typescript

V Javascriptu nemají proměnné, ani parametry funkcí informace o typech, a proto vývojář může opomenout nějaký případ a mohou vznikat errory v runtimu. Typescript je naopak typově založeý jazyk, který tyto chyby detekuje již v buildtimu a upozorní tak vývojáře. Jednoduše řečeno Typescript je Javascript s datovými typy. Vue projekt podporuje oba zmíněné jazyky a pro portál byl zvolen Typescript.

Základními datovými typy jsou: *number, string, boolean, object, array, any, null a undefined*. Typescript také umožňuje vytvářet vlastní datové typy nejčastěji pomocí *interfaců*. [25], [26]

Quasar

Quasar je grafická knihovna, která uživatelům poskytuje graficky zpracované komponenty (např. tabulka, tlačítko, layout stránky...). Ty lze samozřejmě upravovat podle individuálních potřeb. Všechny komponenty jsou responzivní, podporují dark mode a chovají se tak, jak by uživatel očekával. Použití grafické knihovny urychluje vývoj a také pomůže docílit konzistentního designu celé aplikace. [27]

I18n

Tento nástroj slouží pro překlady. Jelikož je předmět BI-DBS vyučován i pro zahraniční studenty, je potřeba aby aplikace podporovala více jazyků. Momentálně je v systému možné zvolit češtinu nebo angličtinu. [28]

Router

Vue router umožňuje sestavit single page aplikaci, což znamená, že u přechodu mezi stránkami nedochází k načítání celého obsahu, ale pouze toho nového.

Aby směrování pomocí routeru fungovalo, musí se nejdříve zadefinovat jednotlivé routy. Každý takový route má přiřazenou komponentu, která se při přesměrování načítá.[20]

Es-lint

Tento nástroj slouží ke kontrole a formátování textu kódu. Podle předepsaných pravidel nejdříve zkontroluje kód, popřípadě ho opraví. Pomáhá udržovat konzistentní kód v celém projektu, a tak usnadňuje jeho čitelnost. [29]

Pinia

Pinia je stavová knihovna pro Vue aplikace. Hlavní stavy informací jako nastavený jazyk, přihlášený uživatel apod. by bylo těžké posílat přes celou aplikaci. Pinia slouží k uchování těchto stavů, které se dají zjistit z jakékoliv části kódu. [30]

Axios

Další použitou knihovnou je Axios. Jedná se o knihovnu, která má na starosti práci s HTTP požadavky. Umožňuje vývojářům snadno odesílat požadavky a zpracovávat odpovědi v různých formátech. [31]

Docker

Docker je open-source platforma pro vývoj a spouštění aplikací, která umožňuje izolovat aplikace do tzv. kontejnerů. Tímto způsobem je možné, aby více kontejnerů běželo současně a nezávisle na sobě na jednom stroji. Pomocí virtualizace je vytvořeno kompletní prostředí pro vývoj, testování a nasazování aplikací. [32]

Postman

Postman je platforma pro vývoj, používání a testování API. Umožňuje snadno vytvářet a odesílat HTTP požadavky a testovat tak jednotlivé endpointy. Pomocí workspaců lze snadno sdílet práci mezi členy týmu. [33]

Vuepress

Vuepress je generátor statických stránek optimalizovaný především pro vytváření technických dokumentací. Dokonce samotná dokumentace Vue.js byla vytvořena pomocí tohoto nástroje. Velkou výhodou je předem poskytnutý motiv, díky kterému jsou stránky graficky přívětivé. Motiv lze také samozřejmě upravit. Stránky ve Vuepress jsou vytvářeny v Markdown souborech. Každý takový soubor je nejdříve zkompilován do HTML a poté zpracován jako Vue komponenta. To dovoluje přímo přidávat také Vue komponenty. [34]

2.5 Shrnutí návrhu

Návrhy byly vytvářeny ve Figmě v iteracích a postupně konzultovány s vedoucím projektu. To mi umožnilo detailně se zaměřit na jednotlivé části a postupně je zdokonalovat. Výsledné návrhy jsou dle mého názoru velkým pokrokem UI současného stavu studentských testů. Tyto návrhy budou sloužit při implementaci jako předloha. Dále jsem představila technologie pro realizační část.

Kapitola 3

Realizace

V této kapitole nejdříve popíši rozběhnutí prostředí frontendu a backendu, a zaměřím se na problémy, které během této fáze nastaly. Dále seznámím čtenáře s hlavními částmi implementace a představím některé vytvořené komponenty. Nakonec shrnu dosažené výsledky.

3.1 Rozběhnutí prostředí

Rozběhnutí projektu FE bylo snadné a nenastaly u něj žádné problémy. Stačilo si projekt z repozitáře Gitlabu naklonovat lokálně a poté pomocí Dockeru spustit. K tomu jsem zvolila desktopovou verzi aplikace Docker Desktop.

Zprovoznění BE pro mě bylo náročnější, jelikož jsem s ním doposud neměla žádné zkušenosti. Při rozběhnutí projektu jsem se naštěstí mohla řídit návodem v dokumentaci od Ing. Andriho Plyskach. Přesto však nastaly problémy, které dále více rozvedu.

První problém nastal při spuštění projektu. Ukázalo se, že některé části BE nebyly plně funkční. Po diskuzi s BE týmem jsem zjistila, že stačilo část modulu odstranit. Poté už spuštění proběhlo v pořádku. Tento problém jsem se však nejdříve snažila vyřešit sama, a až poté jsem se obrátila na tým BE.

Další problém vznikl z důvodu konfigurace prostředí na OS Windows. Jelikož jsou kontejnery BE linuxového typu, na Windows mi procesy trvaly nepřiměřeně dlouhou dobu. Například instalace knihoven v kontejneru, která za normálních okolností trvá přibližně minutu, běžela cca 15 minut. Neobvykle dlouhou dobu zabralo také vyhodnocování požadavků, jež bylo potřeba k vytváření dat. Řešením těchto problémů bylo přesunutí projektu BE na Windows Subsystem For Linux (WSL). Problémy však zpomalily postup vývoje o pár dní. Navíc se ve WSL objevil další problém spojený s právy v gitu.

Doposud se jednalo o problémy, které pouze zpomalily vývoj. Následující problém mi však kompletně znemožnil práci. Při spuštění sekvence požadavků pro vytvoření dat se data nevytvořila. Problém byl u požadavku **Import semestru**, který překročil maximální dobu na zpracování požadavku 30 vteřin.

Zajímavé na tom bylo, že požadavek vracel kód 200 a trvalo mi tedy delší dobu, než jsem přišla na to, že problém byl právě v importu semestru. Někdy tento problém vyřešil restart počítače a nová konfigurace kontejnerů, jindy však problém přetrvával několik dní. Pokusila jsem se řešit tento problém úpravou parametru stanovujícího maximální dobu pro provedení požadavku, avšak kontejner na tuto změnu nereagoval. Konfigurace kontejnerů dockeru není mojí specializací, a tak jsem se obrátila na kolegu Jana Trojáka, který má k tématu bližší vztah. Po důkladném zkoumání situace a několika neúspěšných pokusech o řešení mi doporučil přesunout projekt BE do virtuálního stroje s OS Linux. Toto řešení problém odstranilo a umožnilo mi tak pokračovat ve vývoji FE. Proces zkoumání a odstranění problému mi však znemožnilo vývoj na 5–6 dní. Tento problém časem nastal opakovaně. Také se objevil u jiných vývojářů a po prozkoumání BE se zjistilo, že je chyba na straně BE. Jelikož nebyl BE v rozumném čase schopný vyřešit tento problém globálně, bylo potřeba lokálně provést *merge* s větví s řešením problému.

3.2 Vytváření dat

Když jsem začala s implementací, projekt BE ještě nebyl připravený na použití. První funkční verzi mi tým BE poskytl až v průběhu implementace. Díky definici API jsem však znala rozhraní dat, podle kterých jsem si mohla vytvořit data mockovací. Jelikož jsem věděla, že v nedaleké době budu moci komunikovat s BE, a mockování je tedy velmi krátkodobé řešení, rozhodla jsem nezvolit žádný mockovací nástroj. Pomocí definovaných rozhraní (*interfaců*) jsem si ve scriptu komponent vytvořila pouze proměnné, které jsem využívala jako mockovací data.

Předpokladem pro spuštění studentského testu je jeho existence. Vytváření termínů, testů a testových otázek je součástí FE testů z pohledu učitele. Tato část FE však prozatím nebyla naimlementována, a proto bylo potřeba data vytvořit manuálně. Data jsem vytvářela pomocí aplikace Postman, kterou jsem popsala v předchozí kapitole 2.4.

Bc. Radoslav Hašek, který vyvíjel BE testů a testových šablon, mě přizval do týmu v Postmanu a poskytl mi kolekce požadavků týkající se testů a testových šablon. Později jsem si z těchto požadavků vytvořila vlastní kolekci (sekvenci požadavků), která provedla sekvenci po6adavk; a vytvořila tak potřebná data pro studentské testy. Tato kolekce mi ulehčila práci tím, že nebylo nutné provádět desítky požadavků manuálně. Pomocí kolekce jsem byla schopná data dostat vždy do stejného stavu.

3.3 Implementace

V této části popíši implementační část práce. Během implementace jsem spolupracovala s jedním ze zmíněných SP týmů (dále jen SP tým) a zastupovala

jsem roli zadávájícího. V následujícím textu bude jasně zmíněno, pokud byla některá část převzata od tohoto t7mu. Především však představím mnou vytvořenou část implementace.

3.3.1 Wrapper pro tabulku

Framework Quasar poskytuje předem definované komponenty tabulky *QTable* nebo *QMarkupTable*. Věděla jsem, že tabulka bude využita na více místech portálu. Aby tak nevznikal duplicitní kód, a tabulka se nevytvářela pokaždé znovu, rozhodla jsem se vytvořit komponentu wrapperu tabulky. Vytvoření wrapperu také umožní rozšířit tabulku o jiné funcionality, které tabulka Quasaru nepodporuje. Pro wrapper jsem zvolila komponentu *QTable*, která je podle dokumentace komplexnější než *QMarkupTable*.

Wrapper v počáteční fázi podporoval pouze základní vlastnosti, během vývoje se však průběžně rozšiřoval o pokročilejší funkcionality. Také proběhlo několik refaktoringů pro zlepšení kvality kódu. Nyní představím vlastnosti, které wrapper podporuje.

- **Řádky tabulky**: Tabulka Quasaru přijímá řádky jako pole objektů. Kvůli Typescriptu však bylo potřeba vyřešit datový typ objektu. Rozhodla jsem se vytvořit *interface* pro obecný řádek. Všechny konkrétní řádky z něj poté dědí, což mi umožní přiřadit jeho typ do obecného řádku. Následně jsem musela vymyslet, co bude spojovat řádky libovolných tabulek, abych mohla *interface* obecného řádku definovat. Jelikož většina řádků bude obsahovat akce, *interface* obsahuje pouze jeden nepovinný *atribut*, a to pole Akcí (více popsané v dalším bodě). Konkrétní řádek má tedy typ potomka obecného řádku, a tím pádem obsahuje i nepovinný atribut akcí. *Interface* konkrétního řádku pak může obsahovat další libovolné hodnoty. Prozatím se každý nový typ řádku musí přiřadit do Union typu *TableRowType*, do budoucna by šlo však vymyslet vhodnější řešení.

- **Akce**: Akce jsou v tabulkách využívány poměrně často, a proto bylo potřeba vymyslet komplexní řešení. Každá akce je znázorněna pomocí tlačítka. Pro akci jsem zadefinovala nový *interface*, kterému náleží tyto *atributy*: barva, ikona, barva ikony, velikost a nápis pro *tooltip*. Těmito atributy uživatel volí grafické vlastnosti tlačítka. Dalším atributem je funkce, která určuje, co se provede při stisknutí tlačítka. Dalším atributem je podmínka, která definuje, kdy je daná akce přiřazena řádku. Posledními atributy je určena podmínka a *tooltip* pro tlačítko v *disabled* módu. Wrapperu je pomocí *props* předáno pole akcí a podle vlastností řádku jsou pak řádku přiřazeny vyhovující akce splňující podmínku. Přiřazení akcí probíhá teprve po načtení dat řádků, což jsem vyřešila pomocí *watcheru*. *Interface* akce je řešen pomocí generického typu, a

```
export interface TableRow {
    actions?: Array<Action>
}

export interface Action<T extends TableRow = TableRowType> {
    icon: string,
    label: string,
    bgColor: string,
    iconColor: string,
    action: (item: T) => void,
    condition: (item: T) => boolean,
    size: string,
    disabledCondition?: (item: T) => boolean,
    disabledLabel?: string
}
```

Obrázek 3.1: Definování interfaces pro řádek a akci

to z důvodu typu proměnných ve funkcích. Definice řádku a akce je v ukázce 3.1.

- **Sloupce tabulky**: Využití sloupce je stejné jako v Quasar tabulce. Při předávání sloupců pomocí *props* pak stačí dodržet rozhraní definované v Quasar tabulce. U sloupce lze určit, zda má podporovat řazení, případně pak zadefinovat funkci pro řazení vlastní.
- **Stránkování a řazení**: Stránkování a řazení je řešeno na backendu. To umožňuje efektivně načítat pouze zobrazovaná data. Jelikož se data načítají mimo wrapper, bylo potřeba vyřešit komunikaci mezi komponentami. Rodičovská komponenta pomocí *props* předá wrapperu stránkování (počet řádků na stránku, řazení...) Ve wrapperu je pak stránkování řešeno přes *v-model*. Pokud se stránkování či řazení změní, je rodičovská komponenta upozorněna pomocí *emitu*. V tom případě musí být načtena nová data. Prozatím wrapper nepodporuje filtrování, je však připraven na jeho začlenění.

Dále jsem se rozhodla přidat možnosti pro úpravu tabulky. Jedná se o vlastnosti, které se určují bool typem v props wrapperu. Tedy mohou se vypínat nebo zapínat.

- **Hlavička tabulky**: Hlavičkou tabulky se myslí řádek s názvy sloupců. Přepínačem se určuje, zda se hlavička zobrazovat má, či nikoli.

- **Spodní lišta**: Pomocí přepínače lze obdobně stanovit, zda se zobrazí spodní lišta tabulky. Ve spodní liště jsou zobrazeny prvky pro stránkování.
- **Hustota řádků**: Přepínačem dense lze určit zda mají být řádky klasické nebo hustější. Při druhé variantě řádky zabírají méně místa jak na šířku, tak i na výšku.
- **Hover efekt**: Další přepínač určuje, zda se při najetí myší zvýrazní řádek.
- **Načítání**: Při načítání dat je potřeba dát uživateli najevo, že probíhá načítání. Proto je pomocí *props* předáván přepínač, a pokud probíhá načítání, v tabulce je tento stav znázorněn.

Dále jsem přidávala pokročilejší vlastnosti, konkrétně jsem umožnila v tabulce zobrazovat vlastní obsah v podobě komponent. Ve wraperu jsem zadefinovala *sloty* pro různé části tabulky. Kvůli těmto *slotům* však bylo potřeba přeuspořádat strukturu komponenty wrapperu. Při použití je v rodičovské komponentě potřeba přiřadit *slotu* se správným jménem vlastní obsah.

- **Sloty pro sloupce**: Výchozím formátem políčka tabulky je hodnota políčka. Quasar sice umožňuje přiřadit sloupci vlastní formát, může se však jednat pouze o modifikaci textu. Ve wrapperu jsem proto vytvořila *slot* pro každý sloupec, a vývojáři jsem tak umožnila v políčku zobrazit libovolný obsah.. V tabulce studentských testů se například ve sloupci úspěšnost zobrazuje komponenta *q-chip*, která zdůrazňuje bodové hodnocení. Jméno slotu se mění dynamicky podle názvu sloupce. Ukázka využití je na obrázku 3.2
- **Vlastní první řádek**: Při psaní testu bylo potřeba v záhlaví tabulky zobrazit odpočet času a tlačítko na ukončení testu. Proto jsem ve wrapperu zadefinovala *slot* **customTopRow**. Vývojář si tak v záhlaví může zobrazit libovolnou komponentu, či dokonce více komponent.
- **Řádky navíc**: V tabulce zobrazení detailu testu bylo podle návrhů potřeba zobrazit progress bar pod každým řádkem. Jelikož má být wrapper obecný, rozhodla jsem se tento problém vyřešit pomocí přepínače **extraRows** a *slotu*. Pokud vývojář potřebuje přidat podobnou vlastnost, může zvolit přidání speciálních řádků, ve kterých si může zobrazit libovolný obsah. Během vytváření tohoto slotu jsem se potýkala s problémem rozhraní řádků, když jsem chtěla do řádku vložit další element. Výsledné řešení je takové, že slot je uvnitř elementu *td*, který je až pod řádkem samotným.
- **Stylování řádku**: Tato možnost povoluje graficky upravovat řádky podle zadané funkce. Funkce předaná přes *props* se aplikuje na každý

```
<template #row.points="slotProp">
    <div v-if="slotProp.row.term_type === TermType.Demo">
        {{ pointsFormat(slotProp.row) }}
    </div>
    <q-chip
        v-else
        :color="setChipColor(slotProp.row)"
        text-color="white"
    >
        {{ pointsFormat(slotProp.row) }}
    </q-chip>
</template>
```

Obrázek 3.2: Ukázka kódu – přiřazení obsahu slotu sloupce v tabulce

řádek, a podle ní se pak zvolí grafické úpravy řádku. Takto si vývojář může upravovat vzhled podle vlastních potřeb, a každý řádek může mít jiný vzhled.

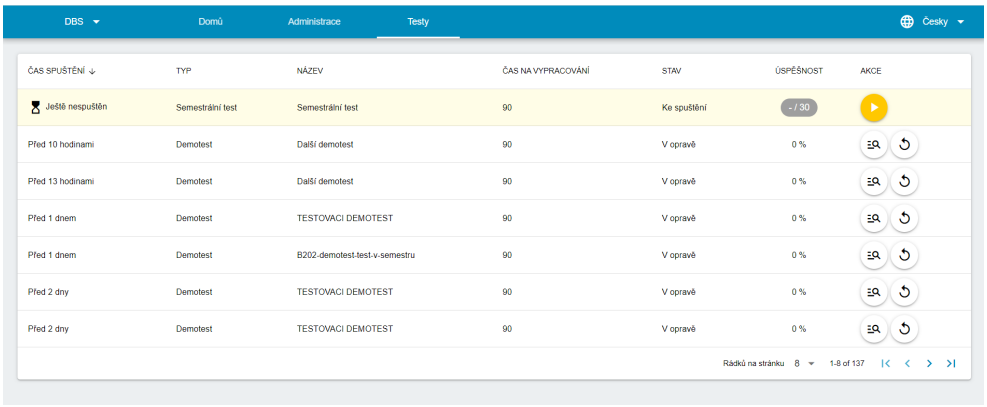
Prozatím wrapper tabulky obsahuje vlastnosti potřebné pro studentské testy. Wrapper je však navrhnut tak, aby bylo snadné začlenit další funkcionality v případě, že by nebyly dostačující.

3.3.2 Využití wrapperu pro tabulku testů

Hlavní stránkou studentských testů je stránka, která zobrazuje **tabulku testů studenta**. Tuto stránku si můžete připomenout v návrhu 2.5. Pro tuto stránku jsem vytvořila komponentu *StudentTestsTablePage*, která využívala wrapper tabulky. Řádek z této tabulky (tedy řádek jednoho testu) měl být studentovi zobrazen i při psaní a náhledu testu (jak lze vidět v návrzích 2.17, 2.13). To by však znamenalo vytvořit novou tabulku (pouze s jedním řádkem) s velice podobnými vlastnostmi. Tyto tabulky se měli lišit v následujících vlastnostech: zobrazení záhlaví, filtrování, posledního řádku se stránkováním a samozřejmě daty. Proto vznikla komponenta *StudentTestsTable*, která slouží přímo pro zobrazení tabulky studentských testů (jak pro celou tabulku testů, tak pro zobrazení samostatného řádku). Na základě přijímaných *props* jsou opět upravovány vlastnosti tabulky. Použitím sdílené komponenty bylo možné definovat společné vlastnosti (sloupce, akce...) obou tabulek pouze jednou.

Při vytváření tabulky studentských testů jsem řešila problém formátu sloupce *Čas spuštění*. Požadavky stanovují zobrazit čas spuštění ve formátu *před x hodinami*. Pro tuto funkcionalitu jsem využila plugin *timeago.js*, který ze *stringu* času vytvoří právě požadovaný formát. Problém však nastal v překladech. Jelikož v českém jazyce je jiný slovosled než v anglickém a zároveň

jednotky času se mění, nestačil pouze statický překlad, ale je využit překlad s parametry. Nová realizace přehledu studentských testů je znázorněna na obrázku 3.3.



Obrázek 3.3: Tabulka studentských testů

3.3.3 Psaní a náhled testu

Pro stránky psaní a náhledu testu vznikly komponenty *WritingTestPage* a *TestDetailPage*. Jelikož obě tyto stránky měli zobrazovat velmi podobné rozhraní (jak lze vidět v návrzích 2.17 a 2.13), rozhodla jsem se vytvořit komponentu *TestLayout*, která je použita v obou těchto stránkách. Společná komponenta, která řeší jak psaní, tak i náhled umožnila eliminovat duplicitní kód. Zároveň do budoucna usnadní údržbu, jelikož změny v jedné komponentě se projeví ve všech stránkách, které ji využívají. Toto zefektivnění kódu však přineslo několik problémů. Hlavní problémy a jejich řešení popíši v dalších odstavcích.

Prvním problémem byla **tabulka otázek**, která není pro obě části identická. Tento problém jsem vyřešila tak, že vlastnosti pro tabulku se určují v rodičovských komponentách (*WritiongTestPage* a *TestDetailPage*), a jsou předávány pomocí *props*. Dobrý návrh wrapperu *QTableExtension* mi dovolil zadefinovat *sloty* jak pro psaní testu, tak pro náhled bez rozdílu. Podle sloupců z *props* se pak přiřadily pouze *sloty* potřebné. Kromě *slotů* pro sloupce jsem zde zadefinovala i *slot* pro první řádek tabulky. První řádek v psaní testu obsahuje komponentu odpočtu času a tlačítko na odevzdání testu. Komponenta pro odpočet času mi byla poskytnuta od SP týmu.

Zároveň jsem řešila **rozhraní řádků**. Při psaní i náhledu je potřebné znát informace o otázce a odpovědi. U náhledu navíc může obsahovat ohodnocení odpovědi a u psaní je naopak potřeba zaznamenat stav odpovědi 2.18. Opět jsem zde využila dědičnosti *interfaců*. To mi dovolilo vytvořit *inter-*

face ***QuestionRow*** se společnými atributy, a poté potomka pro psaní a potomka pro náhled s atributy navíc. Takto zadefinované rozhraní mi dovolilo v *TestLayout* pracovat s řádky stejným způsobem. Rozhraní řádků je vytvářeno v rodičovských komponentách (*WritiongTestPage* a *TestDetailPage*) a předáváno pomocí props.

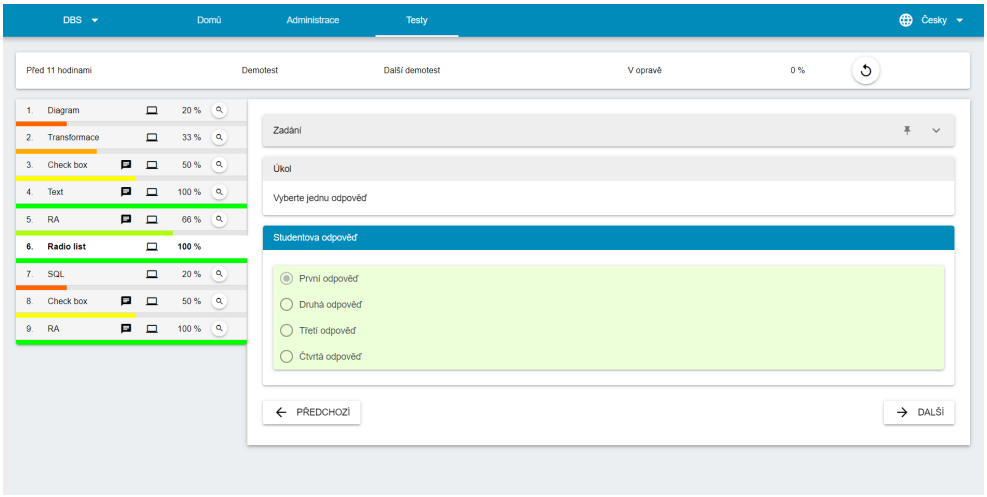
V komponentě TestLayout jsem také vyřešila **zvláštní případy detailu aktivní otázky v náhledu** opraveného testu. Jedná se o případ znemožnění náhledu otázky z důvodu probíhajícího termínu. Přestože obsah pravého boxu (zadání, úkol...) se řeší v komponentě (*QuestionView* popsaná v dalším odstavci), tento obsah jsem se rozhodla zařadit do komponenty *TestLayout*, A to z toho důvodu, že se jedná o zvláštní případ, který nezobrazuje klasický obsah pravého boxu. Komponenta zvláštního případu je zobrazena na základě podmínek. Je připravena také podmínka pro další zvláštní případ přihlášení k ústní zkoušce. Ta však není prozatím není použita.

Obě stránky (psaní a náhled testu) zobrazují zadání, úkol a studentovu odpověď (třebaže v jiné formě). Základ pro tuto část mi poskytl tým SP. Jednalo se o komponentu *QuestionView*, která měla zobrazovat zmíněné části. Jelikož dle mého názoru byla komponenta málo rozčleněna do komponent menších a nebyl čas, abych čekala na jejich opravení, použila jsem ji jen jako základ a provedla úpravy. Kód zobrazující **zadání** jsem vložila do nové komponenty a upravila připnutí zadání tak, aby měl stanovenou výšku a nezabíral celou obrazovku. Přidala jsem také funkci sbalení zadání, která byla vyžadováno v náhledu testu. Také kód zobrazující **otázku** jsem přesunula do vlastní komponenty. A v této části jsem vyřešila komunikaci s BE. Přidala jsem také komponentu pro komentář, která se zobrazuje pouze v náhledu testu. Na obrázku 3.4 si můžete prohlédnout realizaci náhledu opraveného demotestu. Psaní testu pak znázorňuje obrázek 3.5.

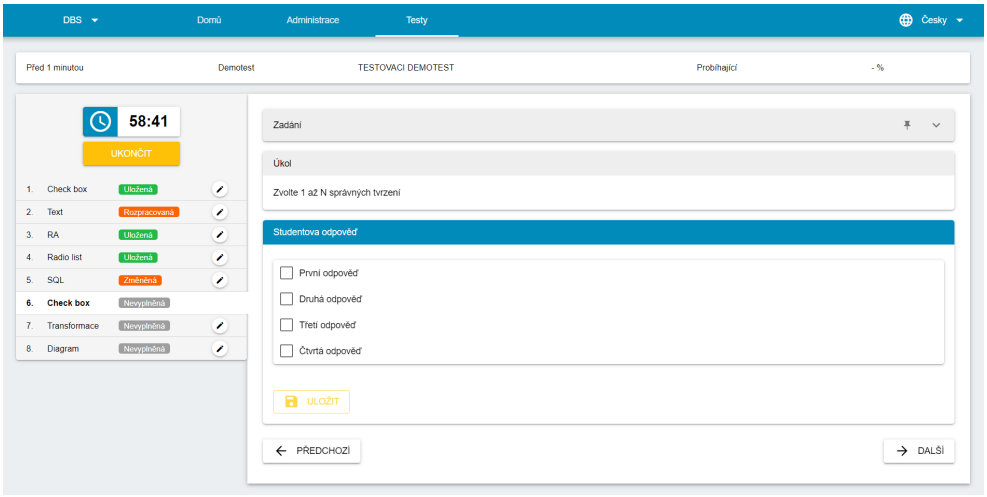
3.3.3.1 Studentova odpověď

Jak bylo zmíněno v předchozím odstavci, studentova odpověď je zobrazována z komponenty *QuestionView*. Pro každý **typ odpovědi** byla vytvořena samostatná komponenta. Tato komponenta opět řeší zobrazení odpovědi jak pro psaní testu, tak pro náhled. Tato vlastnost je pak určena pomocí *boolean* přepínače v *props*. Pomocí *props* komponenta přijímá také hodnotu studentovi odpovědi. Zde byl problém s typem této hodnoty. Typ hodnoty odpovědi totiž určuje typ odpovědi. Např. komponenta odpovědi typu Check box má pracovat pouze s *polem čísel*. Proto jsem zde využila *type assertion*, které mi dovolilo předat hodnotu se správným typem. Uvnitř komponenty je hodnota aktualizována pomocí *v-model* a pomocí *emit* předána zpět rodičovské komponentě *QuestionView*.

Při psaní testu jsem řešila také problém s **vytvářením odpovědí**. Prvotní myšlenkou bylo při spuštění testu vytvořit pro každou otázku prázdnou odpověď. Při uložení odpovědi studentem by se odpověď pouze aktualizovala.



Obrázek 3.4: Náhled opraveného demotestu



Obrázek 3.5: Psaní demotestu

Takto jsem vytváření otázek také naimplementovala. Bohužel BE byl navržen tak, že nebylo možné prázdnou odpověď vytvořit. Proto jsem musela odpověď vytvořit až u prvního uložení studentem. To mi však znesnadnilo práci. Nakonec jsem se rozhodla vytvořit si inicializační odpoveď s hodnotou *null* pro všechny typy odpovědí, která je využita pouze na FE. Pří předání odpovědi do komponenty odpovědi je pak snadné zjistit, zda už odpověď existuje.

V rámci této práce jsem vytvořila komponenty pro následující typy odpovědí: **text, check box, radio list a transformace**. Komponenta trans-

```
<template>
    <q-card class="column full-width q-ma-sm">
        <q-checkbox
            v-for="option in questionOptions"
            :key="option.id"
            v-model="checkedOptions"
            :label="option.label"
            :val="option.id"
            :color="writing ? 'primary' : 'grey'"
            :class=" writing ? 'option'
                : styleEvaluatedOption(option)"
            :disable="!writing"
            :model-value="checkedOptions"
            @update:model-value="updateAnswer()"
        />
    </q-card>
</template>
```

Obrázek 3.6: Ukázka kódu – odpověď typu Check box

formace v módu psaní textu nepodporuje všechny funkcionality. Do budoucna by měla být napojena na Tralex. Tralex slouží pro opravu otázek typu transformace, také ale obsahuje upravený editor, který studentovi našeptává názvy entit. Než se však integruje do nového portálu, bude zde pouze obyčejné textové pole. Dále mi byly poskytnuty od SP týmu komponenty pro typy odpovědí: diagram, RA a SQL. Většina typů odpovědí je však stále z části namockovaná.

V ukázce kódu 3.6 můžete vidět *template* komponenty pro **Check box odpověď**. Můžete si všimnout, že třídy jsou přiřazovány podle funkce *styleEvaluatedOption*. Tyto třídy udávají grafické úpravy řádku podle správnosti odpovědi.

3.4 Vývojářská dokumentace

Jak jsem zmínila v předchozí kapitole, dokumentaci jsem vytvořila pomocí nástroje Vuepress. Vuepress dokumentaci lze vytvořit 2 způsoby. Lze použít generátor, který sestaví projekt automaticky, nebo lze projekt sestavit manuálně. Nakonec jsem zvolila manuální možnost, a to z důvodu chyby při použití generátoru. Chyba byla zřejmě spojená s použitím generátoru s *yarn*. Rozdíl mezi automatickým sestavením a manuálním si můžete nastudovat v dokumentaci Vuepressu. [34]. Pro dokumentaci jsem zvolila následující strukturu:

- **Návody:** v této sekci jsou obsaženy návody, např. jak si spustit projekt.
- **Knihovny:** zde jsou popsány frameworky a knihovny využívané na FE. Také zde naleznete ukázky použití knihoven.
- **Časté problémy:** v následující části jsou popsány časté problémy a jejich řešení. Než se student při problému na někoho obrátí, měl by prozkoumat dokumentaci, zda už se někdo s podobným problémem nesetkal.
- **Knihovna komponent:** v projektu by se měly co nejvíce využívat již vytvořené komponenty. V této části dokumentace je seznam již vytvořených obecných komponent a jejich základní použití. Ideální průchod v praxi by byl: Pokud někdo potřebuje nějakou komponentu, podívá se do dokumentace, zda už náhodou vytvořena nebyla. Pokud byla, prozkoumá její použití, pokud nebyla, komponentu vytvoří a přidá do dokumentace.
- **Moduly:** v této části je prostor pro jakýkoliv relevantní obsah pro jednotlivé moduly (testy, semestrální práce, atd). Mohou zde být odkazy na návrhy, poznatky k mikroslužbám BE...
- **Psaní dokumentace:** v poslední sekci jsou informace k rozšiřování dokumentace. Je zde vysvětlena struktura projektu dokumentace a návod na lokální rozjetí.

Aby čas strávený rozšiřováním dokumentace byl minimální, rozhodla jsem se rozdělit kód pouze do souborů podle zmíněných kategorií. Není potřeba vyhledávat v desítkách souborů a importovat menší části. Tato struktura umožní rychlou orientaci v kódu a redukuje tak čas strávený dokumentací.

V průběhu vývoje jsem doplňovala jednotlivé části dokumentace. Po nasazení, se kterým mi pomohl Bc. Max Hejda, byla dokumentace představena také studentům SP1. Jelikož dokumentace slouží právě pro ně, mají možnost dokumentaci libovolně upravovat a rozšiřovat. Dokumentace je tedy již reálně využívaná a je dostupná na odkazu: dbs.pages.fit/dokumentace-fe/.

3.5 Shrnutí realizace

Realizační část byla velmi náročná, a to především kvůli problémům spojeným s backendem. Tyto problémy zdržely vývoj v řádech dnů až týdnů. Kromě problémů s rozjetím projektu jsem se potýkala také s překážkou neaktualizování definice API ze strany BE, a to i přes opakované upozornění na tento problém. I přes tyto překážky se mi však podařilo naimplemetovat velkou část studentských testů.

- Je možné zobrazit si přehled studentských tesů.
- Lze spustit test a následně ho vyplnit.
- Důležité je, že je funkční zobrazování stavů odpovědí a eliminuje se tedy odevzdání testů s neuloženými odpověďmi.
- Funkční je také ukončení testu pomocí dvojité kontroly.
- Dále je možné zobrazit si náhled ukončeného testu s opravenými odpověďmi.
- Prozatím testy podporují následující typy odpovědí: text, check box, radio list, transformaci, SQL, RA a diagram.
- Přihlašování na zkoušku je prozatím možné pouze z tabulky testů. Tedy chybí přidání řádku pro ústní zkoušku v náhledu zkoušky.
- Je také znemožněno opustit test při psaní testu.

Velká část FE komunikuje s BE, pouze následující části pracují s mockovacími daty: zadání, bodové hodnocení v náhledu, komentáře, a některé typy odpovědí. Velkým přínosem pro celý portál je dle mého názoru wrapper pro tabulku a také vytvořená dokumentace. Spolupráce s SP týmem v konečném důsledku práci spíše zpomalila, jelikož bylo potřeba je seznámit s projektem a návrhy, společně se sesynchronizovat a domlouvat postup práce. Tým mi však poskytl několik komponent, které jsem po úpravách zařadila do své části implementace. Výhodou spolupráce také je, že tým už se orientuje v projektu, specificky v testové části, a bude schopen ve vývoji pokračovat dál (pokud se rozhodnou pokračovat ve vývoji portálu v předmětu BI-SP2).

Pro výsledek je také důležité zmínit, že ačkoli je vytvořená část spustitelná, nemá smysl ji rozjíždět, pokud v zařízení nejsou lokálně vytvořená data pro zobrazení.

Kapitola 4

Testování a budoucí rozvoj

V této kapitole čtenáře seznámím s testováním. Nejdříve popíši výběr testů, stejně jako skupiny testových uživatelů, a poté představím nedostatky, které z testů vyplynuly. Dále stanovím budoucí směr rozvoje.

4.1 Výběr testů

Jelikož implementace nebyla zcela dokončena, bylo potřeba vhodně zvolit testy. Velkou částí této práce byly návrhy pro nový frontend, které přinesly kompletně nové rozhraní. Proto jsem pro testování zvolila **uživatelské testování (usability testing)**, které by mělo ukázat, zda byl návrh kvalitní, a je pro uživatele rozhraní přívětivé a intuitivní. Zvažovala jsem také testování výkonu uživatelského rozhraní a akceptační testy, u obou typů testování jsem však usoudila, že z důvodu nekompletního frontendu prozatím testy postrádají smysl.

4.2 Uživatelské testy

Uživatelské testování (neboli usability testing) je častým způsobem ověřování **přívětivosti** uživatelského rozhraní. Jedná se o testování, které vyžaduje účast testovaných uživatelů (respondentů). Během testování jsou respondentům zadávány úkoly na provedení podle připravených scénářů. Zadavatel pozoruje reakce, chování a zpětnou vazbu respondentů a zapisuje si poznámky. Cílem testování je bližší pochopení chování a preferencí cílových uživatelů a především identifikování problémů v návrzích UI. Respondenti by měli být vybírání jako realističtí uživatelé. Buď by se tedy mělo jednat o uživatele, kteří již aplikaci používají nebo alternativně o uživatele s podobnými potřebami. [35]

4.2.1 Skupiny respondentů

Pro testování jsem zvolila 2 skupiny respondentů. První skupina zahrnovala **stávající studenty FIT ČVUT**, kteří již předmět BI-DBS absolvovali. Tito respondenti mohli poskytnout porovnání se současným portálem DBS. Jelikož nový portál DBS budou používat studenti, kteří se starým portálem nikdy nepracovali, pro druhou testovanou skupinu respondentů jsem zvolila **studenty jiných vysokých škol**, kteří předmět BI-DBS neabsolvovali. Tato skupina mi mohla poskytnout zpětnou vazbu z pohledu běžných vysokoškolských studentů. A přiblížila se tak reálné skuině uživatelů. Pro identifikování většiny problémů produktu Nielsen Norman Group doporučuje testovat 5 respondentů z každé skupiny. [35] Z důvodu předčasného testování před kompletním dokončením celého frontendu jsem však zvolila celkový počet respondentů 5 (2 studenty, kteří již absolvovali BI-DBS, a 3 studenty jiných škol).

4.2.2 Průběh testování

Jelikož je pro používání vytvořené části nového portálu nutná konfigurace prostředí (rozchození BE a vytvoření dat), respondenti prováděli úkony na mém zařízení. První skupina byla testována prezenční formou a se studenty z jiných VŠ jsem se spojila prostřednictvím aplikace Anydesk. Tato aplikace umožňuje vzdálené připojení k zařízení, a tedy i jeho ovládání.[36] Jednotlivé testování poté probíhalo následovně. Zjistila jsem základní informace o respondentovi. Slovně jsem představila portál DBS a seznámila respondenta s průběhem testování. Respondenti byli upozorněni na fakt, že některé části používají mockovací data, a že se nejedná o kompletně dokončený frontend. Byli také vyzváni k popisování myšlenkových pochodů během testování slovně nahlas. V průběhu testování jsem si zaznamenávala poznámky s chováním respondentů a jejich připomínky.

Úkol pro respondenty byl následující: **Zopakujte naposledy napsaný demotest a poté si zobrazte dosažené výsledky.** Tento úkol jsem zvolila, jelikož bude často využívaný v praxi studentů. Jedná se o komplexní zadání, jehož testování odhalí schopnost orientace v testové části frontendu a porozumění průchodu psaní testu, tak i náhledu. Dalším úkolem bylo **zobrazit si náhled semestrálního testu**. V tomto náhledu uživateli není umožněn náhled otázek z důvodu stále probíhajícího termínu. V textu nyní popíši nalezené nedostatky a shrnu výsledky testování. Text bude členěn dle testovacích skupin.

4.3 Výsledky testování

4.3.1 1. testovací skupina

Jak bylo zmíněno výše, jednalo se o současné studenty FIT ČVUT, kteří již předmět BI-DBS absolvovali, a tedy byli seznámeni s portálem DBS. V této skupině byli testování 2 respondenti. Respondenti byly po dokončení úkolů dotázáni, zda během úkolů narazili na nějaké nejasnosti a zda nemají další připomínky.

Nedostatky a návrhy pro zlepšení

1. Tlačítka pro navigaci mezi otázkami v psaní i náhledu testu. U první a poslední otázky, kdy nelze akce jedním směrem provést, se zobrazují a jsou *disabled*. Respondenti navrhli tlačítka v těchto případech odstranit.[**Priorita:** nízká, **Obtížnost:** nízká]

2. Chybí *tooltip* u ikony komentáře a opravujícího, pokud je otázka stále v opravě. *Tooltip* je potřebný převážně pro zdůraznění informace, že je otázka v opravě. [**Priorita:** vysoká, **Obtížnost:** nízká]

3. V tabulce testů v *tooltipu* času spuštění se zobrazuje pouze datum. Respondenti navrhli zobrazit datum i přesný čas. [**Priorita:** střední, **Obtížnost:** nízká]

4. Nebylo zcela jasné, jak se z náhledu demotestu vrátit zpět na tabulku testů. Jeden z respondentů téměř stiskl tlačítko pro opakování demotestu. Navigace by se však měla vyřešit v rámci celého portálu. [**Priorita:** střední, **Obtížnost:** střední]

5. U psaní demotestu respondenti navrhli nezobrazovat v tabulce procenta, jelikož jim tato informace nic nepřináší. [**Priorita:** střední, **Obtížnost:** nízká]

6. Při načítání tabulky se ve spodní části automaticky od Quasaru zobrazuje vykřičník (jelikož prozatím tabulka neobsahovala data). Tento vykřičník by se při načítání zobrazovat neměl. [**Priorita:** střední, **Obtížnost:** nízká]

7. U připnutého zadání by se měl scrollovat pouze obsah zadání, aby byla neustále vidět hlavička (připnutí, sbalení). [**Priorita:** vysoká, **Obtížnost:** střední]

8. V tabulce testů nefunguje stránkování při zobrazení všech testů. Tedy tlačítlo ALL. [**Priorita:** vysoká, **Obtížnost:** střední]

9. V případě znemožnění náhledu otázek z důvodu stále probíhajícího termínu jeden z respondentů navrhl náhled znemožnit už z tabulky testů. [**Priorita:** nízká, **Obtížnost:** nízká]

10. U jednoho z respondentů proběhlo chybné uložení některých odpovědi. Způsobeno to bylo tím, že ještě než se odpověď uložila na BE, respondent přešel na jinou otázku. [**Priorita:** vysoká, **Obtížnost:** vysoká]

11. Ikony v tabulce otázek při psaní testu a náhledu testu (ikony +) evokují přidání položky. [**Priorita:** nízká, **Obtížnost:** nízká]

Body 1–5 vzešly z obou testování, tedy na tyto nedostatky poukázali oba respondenti. Většinou se jednalo o pouze malé nedostatky, které často byli spíše doporučením pro lepší řešení. Další nedostatky a nápady pro zlepšení (6–11) byly vyřčeny vždy pouze jedním respondentem.

Porovnání se stávajícím portálem

Už během samotných úkolů respondenti projevily pozitivní ohlas na různé části. Po dokončení testování a diskuzi na téma nedostatky jsem požádala respondenty, aby porovnali moji práci se stávajícím portálem a zdůraznili klady a zápory nové testové části. Oba respondenti jednoznačně preferovali novější verzi a argumentovali mnoha fakty. V následujících bodech shrnu hlavní poznatky z porovnání.

- Jako studenti, kteří již pracovali v současném portálu, respondenti velice ocenily barevné znázornění stavů odpovědí. Znázornění je podle nich přehledné a oproti současnému portálu, kde se tato informace nachází pouze v tabulce otázek (kam se musí překlikávat), je to podle nich obrovský pokrok.

- Stejně tak pozitivně ohodnotili proces odevzdávání testu. Ocenily především to, že při odevzdání byli dotázáni, zda opravdu chtějí odevzdat.

- Snad největším kladem, který zdůraznily oproti současnému portálu jsou náhledy testů. Líbilo se jim, že mají vše na jednom místě, a nemusejí se proklikávat mezi modálními okny, jako v současném portálu.

- Jeden z respondentů projevil sympatie ke konzistenci rozložení stránky při psaní a náhledu testu.

- Respondenti také označily novou testovou část za graficky atraktivnější.

4.3.2 2. testovací skupina

Druhá skupina obsahovala studenty z vysokých škol jiných než FIT ČVUT, a tedy se nesetkali se současným portálem DBS. Pro tuto skupinu jsem zvolila 3 respondenty. Zvolila jsem jich více než v předchozí skupině, a to z toho důvodu, že tito respondenti jsou blíže skupině, která bude nový portál v budoucnu reálně využívat. Dříve, než jsem však začala s testováním 2. skupiny, částečně jsem opravila nedostatek č.11 z 1. skupiny, při kterém došlo k chybnému uložení některých odpovědí. Tento problém jsem shledala natolik závažným, že bylo potřeba ho opravit před testováním další skupiny. Prozatím jsem přidala znázornění načítání ukládání k tlačítku ULOŽIT. Pro kompletní vyřešení problému bude potřeba během procesu ukládání znemožnit přechod na jinou otázku.

Nedostatky a návrhy pro zlepšení

Několik nedostatků bylo identických s již zmíněnými v 1. testovací skupině. Jedná se o nedostatky: 1, 2, 5, 11. Dále popíši nově nalezené nedostatky 2. testovací skupiny.

1. Jeden z respondentů navrhl při konci odpočtu času zabarvit odpočet červenou barvou, aby byl student upozorněn na blížící se konec testu. [**Priorita:** nízká, **Obtížnost:** nízká]

2. Nebyl zcela jasný nápis v *tooltipu* NÁHLED. Vhodnější by podle respondenta byl nápis: ZOBRAZIT VÝSLEDKY. [**Priorita:** nízká, **Obtížnost:** nízká]

3. Dokonce dva z respondentů nezávisle na sobě navrhli přidání možnosti označení otázky vlaječkou. To by mělo sloužit k označení otázky, ke které by se chtěl student později vrátit. Tento princip znají z platformy Moodle. [**Priorita:** nízká, **Obtížnost:** vysoká]

4. Jedním respondentem nebyl pochopen stav: ZMĚNĚNÁ. Přestože byl stav zbarven oranžově, respondent se domníval, že je pouze informován o změnění odpovědi, a pokračoval v testu v domnění, že je odpověď uložená. S odpovědí ve stavu ZMĚNĚNÁ test také ukončil. Respondent navrhl stavy zúžit na 3 a stav ZMĚNĚNÁ nahradit stavem ROZPRACOVANÁ. [**Priorita:** střední, **Obtížnost:** nízká]

5. Stejný respondent také nepochopil při odevzdání testu tlačítko ODEVZDAT S NEDOSTATKY. Domníval se, že nedostatky se týkají aspektu správnosti odpovědi. [**Priorita:** střední, **Obtížnost:** nízká]

6. V tabulce testů se nezobrazuje číslo aktuální stránky tabulky. [**Priorita:** střední, **Obtížnost:** střední]

7. Jeden z respondentů nepochopil řádek odpovědi, který je teprve v opravě. Tento problém však bude vyřešen při eliminování již dříve zmíněného problému s chybějícím *tooltipem* (V OPRAVĚ) na ikonách opravujícího. [**Priorita:** vysoká, **Obtížnost:** nízká]

Porovnání s jinými platformami pro psaní testů

Poté, co mi respondent sdělil připomínky a nejasnosti, opět jsme přešli k diskuzi celkového dojmu. Tentokrát jsem respondenty požádala o porovnání s jinými platformami pro psaní testů, se kterými se setkali. Všichni respondenti používali pouze platformu Moodle. Poznatky s porovnání opět představím v bodech.

- Opět byla vyzdvižena grafická stránka, která byla podle respondentů atraktivnější.
- Respondenti ocenily možnost náhledu všech již napsaných testů. Tato možnost jim v platformě Moodle chyběla.
- Opět respondenti kladně reagovali na odpočet času při odevzdávání.
- Společně s odpočtem ocenily také celkově dvojitou kontrolu při odevzdání. Měli pocit, že mají dostatečnou možnost se k testu vrátit.

4.3.3 Zhodnocení testování

Testování hodnotím velice kladně. Přestože se může zdát, že nedostatků je mnoho, po detailnějším prozkoumání je znatelné, že se jedná o velice drobné a snadno napravitelné detaily. Důležité je, že během testování nenastaly vážnější problémy, kdy by byl respondent v portálu ztracen. Jediné zaváhání jsem pozorovala u respondenta, který zcela nepochopil poslední stav odpovědi. Podstatné však je, že všichni respondenti zhodnotily uživatelské prostředí jako přehledné a snadné pro použití. Respondenti nejvíce ocenily proces ukončování testů, znázornění stavů odpovědí a také rozložení stránek jako celek. Tyto klady byly zmiňovány nejčastěji. Testování podle mého názoru dopadlo uspokojivě především z důvodu důkladných návrhů, které byly pečlivě upravovány v iteracích. Jak jsem zmínila v kapitole 2.3, už během návrhů probíhal určitý druh testování s vedoucím práce. Především to je tedy důvodem, proč se konečná realizace jeví jako úspěšně navržená. Více nedostatků se samozřejmě nalezne při testování po kompletním dokončení frontendu. Vzhledem k úspěšnému otestování hlavní části studentských testů však nepředpokládám žádné kritické nedostatky.

4.4 Budoucí rozvoj

Jelikož testování odhalilo nemalé množství nedostatků, které však nebyly obtížné na opravu, rozhodla jsem se některé z nich vyřešit okamžitě po testování. Z 1. skupiny testování jsem opravila nedostatky: 1, 2, 3, 5, 6, 7, 8, 10 a 11. Tedy z první skupiny nebyly vyřešené pouze dva nedostatky. Z 2. skupiny poté byly opraveny nedostatky: 2 a 7.

Jak už vyplynulo z kapitoly realizace, FE testové části pro studenty není zcela dokončen. Většina FE už komunikuje s BE a pracuje s reálnými daty, ale malá část stále pracuje s daty namockovanými. Je tedy potřeba tuto část přepojit na BE. Prozatím také nebyla realizována zúžená verze tabulky v psaní testu z návrhu 2.21. Do budoucna by se dalo pořemýšlet také o možnosti přizpůsobení rozhraní pravého boxu. Zadání a úkol někdy zabírají zbytečně velký prostor na výšku, při zobrazení ve sloupcích vedle sebe by se tento problém vyřešil.

Nejvíce je však potřeba se zaměřit na dokončení komponent jednotlivých typů odpovědí, které jsou prozatím částečně namockované, a nepodporují všechny předpokládané funkce. Samozřejmě je také potřeba přidat odpověď typu normalizace. Dále je potřeba frontend napojit na autorizační část, která vznikala souběžně s touto prací. Prozatím jsem také neřešila progtestovou image, která je vyžadována při psaní semestrálních testů a zkoušek.

Závěr

Požadavkem této práce bylo analyzovat uživatelské rozhraní testové části současného portálu DBS. Testová část studentů zahrnuje psaní testů a kontrolu výsledků testů různých typů. Abych zvážila veškeré aspekty nově vznikajícího systému, bylo potřeba analyzovat také současný stav mikroslužeb, projektu FE, stejně jako předešlé práce, zabývající se podobným tématem. Identifikovala jsem problémové oblasti stávajícího řešení, a podle analýzy jsem stanovila požadavky na FE nový.

Jedním z hlavních cílů bylo zohlednit nalezené nedostatky a stanovené požadavky, a navrhnout přívětivé uživatelské rozhraní studentských testů. Při navrhování bylo potřeba dbát na fakt, že se jedná o první komplexnější část FE portálu, a bude udávat budoucí směr vývoje celého systému. Ve finálních návrzích se mi podařilo vytvořit konzistentní rozhraní, které by mělo být pro uživatele intuitivní.

Následně jsem pokračovala s implementací studentských testů. Hotová realizace umožňuje zobrazení tabulky přehledu studentských testů. Umožňuje také spuštění testu, jeho vyplnění a ukončení přes dvojité potvrzení. Dále poskytuje možnost zobrazení výsledků testů opravených, nebo testů v opravě. Během implementace bohužel nastala řada problémů, která ovlivnila postup vývoje, a proto implementace nebyla zcela dokončena. Byla však vytvořena hlavní část, kterou bylo možné uživatelsky otestovat. Výsledky testování jednoznačně potvrdily, že nový FE nabízí přehledné rozhraní, ve kterém se uživatelé dokáží orientovat. Toto přínosné zlepšení je částečně důsledkem velmi důkladného návrhu, který jsem prováděla. Společně s implementací také vznikla nová dokumentace celého projektu FE, která je již reálně využívána studenty podílejícími se na portálu DBS.

Bibliografie

1. TEAM, Adobe Communications. *Waterfall Methodology: A Complete Guide* [online]. Adobe Experience Cloud Blog, 2022. [cit. 2023-04-27]. Dostupné z: `https://business.adobe.com/blog/basics/waterfall`.
2. MALEC, Oldřich. *Řízení projektu a infrastruktury portálu pro podporu výuky předmětu BI-DBS*. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2017. Bakalářská práce.
3. PLYSKACH, Andrii. *Modernizace a migrace DBS portálu*. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2023. Diplomová práce.
4. FIT, ČVUT [online]. BI-DBS, 2023. [cit. 2023-05-11]. Dostupné z: `https://dbs.fit.cvut.cz/`.
5. PLYSKACH, Andrii. *Slovník pojmů a procesů z domény* [online]. redmine dbs.fit.cvut.cz, 2022. [cit. 2023-04-17]. Dostupné z: `https://dbs.fit.cvut.cz/redmine/projects/dbs-bp-testy/wiki/Slovn%C3%ADk_pojm%C5%AF_a_proces%C5%AF_z_dom%C3%A9ny`.
6. SLAVOTÍNEK, Jiří. *Webová komponenta na kreslení ER diagramů* [online]. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2016. [cit. 2023-04-23]. Dostupné z: `https://projects.fit.cvut.cz/theses/2135`. Bakalářská práce.
7. FEDOR, Tomáš. *ER diagrams web component II* [online]. Czech Technical University in Prague, Faculty of Information Technology, Prague, 2017. [cit. 2023-04-23]. Dostupné z: `https://projects.fit.cvut.cz/theses/2244`. Master's thesis.
8. ERBEN, Marek. *Automatické generování a oprava otázek na normalizaci databáze pro předmět BI-DBS*. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2018. Bakalářská práce.

9. MOODLE. *Making quality online education accessible for all* [online]. Moodle, 2023. [cit. 2023-04-23]. Dostupné z: `https://moodle.com/about/`.
10. EVOLUTION-TEAM. *Fit KLASIFIKACE* [online]. FIT klasifikace, 2013. [cit. 2023-04-05]. Dostupné z: `https://grades.fit.cvut.cz/`.
11. PLYSKACH, Andrii. *Refaktoring testové části backendu portálu dbs. fit. cvut. cz*. České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2020. Bakalářská práce.
12. HANZL, Martin. *dbs.fit.cvut.cz - Refaktoring testů I.* České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2021. Bakalářská práce.
13. JORDÁN, Pavel. *dbs.fit.cvut.cz - Refaktoring testů II.* České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2022. Bakalářská práce.
14. CHALUPA, Tomáš. *Frontend manuální korektury otázek v portálu dbs.fit.cvut.cz.* České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2022. Bakalářská práce.
15. IBM. *What is REST API?* [online]. ibm.com, 2023. [cit. 2023-04-20]. Dostupné z: `https://www.ibm.com/topics/rest-apis`.
16. PLYSKACH, Andrii. *Popis microslužeb* [online]. redmine dbs.fit.cvut.cz, 2022. [cit. 2023-04-12]. Dostupné z: `https://dbs.fit.cvut.cz/redmine/projects/dbs-bp-testy/wiki/Popis_microslu%C5%BEeb`.
17. ALLOTEY, Charles. *Options API vs Composition API - Vue School Articles* [online]. vueschool.io, 2023. [cit. 2023-05-10]. Dostupné z: `https://vueschool.io/articles/vuejs-tutorials/options-api-vs-composition-api/`.
18. MLEJNEK, Jiří. *Analýza a sběr požadavků.* SI1 přednáška 3., 2018. Dostupné také z: `https://moodle-vyuka.cvut.cz/pluginfile.php/532096/mod_resource/content/7/03.prednaska.pdf`.
19. A.S., UNIFER alfa. *Drátěný model webu – na rozložení záleží* [online]. Unifer, 2022. [cit. 2023-04-13]. Dostupné z: `https://unifer.cz/drateny-model-webu-na-rozlozeni-zalezi/`.
20. ČERMÁK, Jakub. *Nový frontend synchronizačního middleware Timer2Ticket.* České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2021. Bakalářská práce.
21. JŮN, Šimon. *Proč je figma dar z nebes?* [online]. Simonjun, 2022. [cit. 2023-04-23]. Dostupné z: `https://www.simonjun.cz/blog/proc-je-figma-dar-z-nebes`.

22. GROUP, Nielsen Norman. *10 Usability Heuristics for User Interface Design* [online]. Nielson Norman Group, 2023. [cit. 2023-03-26]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/.

23. YOU, Evan. *Introduction* [online]. Introduction — Vue.js, 2014. [cit. 2023-03-21]. Dostupné z: https://vuejs.org/guide/introduction.html.

24. VUE.JS. *Props — Vue.js* [online]. vuejs.org, 2023. [cit. 2023-05-10]. Dostupné z: https://vuejs.org/guide/components/props.html.

25. VUE.JS. *Using Vue with TypeScript — Vue.js* [online]. vuejs.org, 2022. [cit. 2023-04-23]. Dostupné z: https://vuejs.org/guide/typescript/overview.html.

26. W3SCHOOLS. *TypeScript Introduction* [online]. www.w3schools.com, 2022. [cit. 2023-04-23]. Dostupné z: https://www.w3schools.com/typescript/typescript_intro.php.

27. STOENESCU, Razvan. *Why Quasar?* [online]. Quasar Framework, 2015. [cit. 2023-04-23]. Dostupné z: https://quasar.dev/introduction-to-quasar.

28. TEAM, Vue.js. *Introduction — Vue I18n* [online]. kazupon.github.io, 2023. [cit. 2023-05-10]. Dostupné z: https://kazupon.github.io/vue-i18n/introduction.html.

29. TEAM, ESLint. *Getting Started with ESLint - ESLint - Pluggable JavaScript Linter* [online]. eslint.org, 2023. [cit. 2023-05-10]. Dostupné z: https://eslint.org/docs/latest/use/getting-started.

30. MOROTE, Eduardo San Martin. *Pinia* [online]. pinia.vuejs.org, 2019. [cit. 2023-05-06]. Dostupné z: https://pinia.vuejs.org/.

31. AXIOS. *Getting Started —Axios Docs* [online]. axios-http.com, 2023. [cit. 2023-05-06]. Dostupné z: https://axios-http.com/docs/intro.

32. INC, Docker. *Docker overview*. Docker Documentation, 2023. Dostupné také z: https://docs.docker.com/get-started/overview/.

33. POSTMAN, Inc. *What is Postman?* [online]. Postman API Platform, 2023. [cit. 2023-04-16]. Dostupné z: https://www.postman.com/product/what-is-postman/.

34. TEAM, Vuepress. *Introduction — VuePress* [online]. vuepress.vuejs.org, 2021. [cit. 2023-05-10]. Dostupné z: https://vuepress.vuejs.org/guide/#how-it-works.

35. MORAN, Kate. *Usability Testing 101* [online]. Nielsen Norman Group, Nielsen Norman Group, 2019 [cit. 2023-05-05]. Dostupné z: https://www.nngroup.com/articles/usability-testing-101/.

36. GMBH, AnyDesk Software. *Innovative and Reliable: Our Features* [online]. AnyDesk, 2023. [cit. 2023-05-06]. Dostupné z: `https://anydesk.com/en/features`.

Příloha **A**

Seznam použitých zkratek

BE Backend

BI-DBS / DBS Databázové systémy

BP Bakalářská práce

ČVUT České vysoké učení technické

DP Diplomová práce

FE Frontend

FIT Fakulta informačních technologií

RA Relační algegra

SP1 / BI-SP1 Softwarový týmový projekt 1

SP2 / BI-SP2 Softwarový týmový projekt 2

SQL Structured query language

UI User interface

Příloha B

Obsah přiloženého média

- `README.md` ..... stručný popis obsahu přiloženého média
- `text`
  - `BPsuchodan.pdf` ..... text práce ve formátu PDF
  - `BPsuchodan.zip` ..... text práce ve formátu ZIP
- `wireframy`
  - `navrhyStare.pdf` ..... exportované návrhy z Figmy (starší verze)
  - `navrhyNove.pdf` ..... exportované návrhy z Figmy (novější verze)