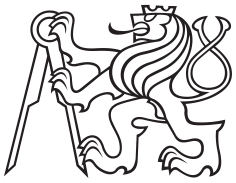
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Mobilní a webová aplikace pro testování znalostí uživatelů

Sebastian Voráč

Vedoucí: Ing. Ivo Malý, Ph.D.
Obor: Softwarové Inženýrství
Studijní program: Web a multimedia
Březen 2017

Poděkování

Chtěl bych poděkovat svým přátelům za podporu, především mému spolubydlícímu, který mě celý čas usilovně podporoval a snášel mé noční studium - mnohdy na úkor i svého spánku. Dále bych chtěl poděkovat rodině, která mě též podpořila jak jen bylo v jejich možnostech. Poděkování patří všem vyučujícím za předání svých zkušeností a užitečných informací, které hojně využívám při vykonávání zaměstnání i v běžném životě. V neposlední řadě děkuji mému vedoucímu práce Ing. Ivovi Malému, Ph.D. za jeho velice cenné rady a zpětnou vazbu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 8. března 2017

Abstrakt

Obsahem této bakalářské práce je návrh a implementace webové a mobilní aplikace určené k testování znalostí studentů. Webová aplikace umožňuje správu učitelů, předmětů, testových otázek a zobrazení výsledků ostrých testů uživatelů mobilní aplikace. Mobilní aplikace poskytuje studentům možnost stahovat vytvořené předměty, ze kterých lze následně testovat své znalosti za pomoci cvičných i ostrých testů. Dále nabízí možnost spravovat výsledky svých testů.

Klíčová slova: vývoj webových aplikací, vývoj mobilních aplikací, testování znalostí studentů, React.js, Node.js, React Native, Android

Vedoucí: Ing. Ivo Malý, Ph.D.
Praha 2, Karlovo náměstí 13, E-425

Abstract

The content of this bachelor thesis is the design and implementation of a web and mobile application designed to test students' knowledge. The web application allows teachers to manage subjects, teachers, test questions and display the test results of mobile application users. Mobile application for students allows to download subjects and then test the knowledge with both mock test and test. It also offers the ability to manage their test results.

Keywords: web application development, mobile application development, student knowledge testing, React.js, Node.js, React Native, Android

Title translation: Mobile and web application for testing user knowledge

Obsah

Obrázky

Tabulky

ctuthesis t1606152353

Kapitola 1

Úvod

Vzhledem ke zvyšující se poptávce webových a mobilních aplikací se zvyšuje i jejich nabídka. Je velice důležité nejen udržet krok s konkurencí, ale v ideálním případě mít po technologické stránce navrch. Při vytváření aplikace je cílem zanechat z používání aplikace co nejlepší uživatelský dojem, který se projeví na celkové spokojenosti zákazníka. Mobilní zařízení jsou společně s tablety čím dál tím více využívány. Webové aplikace jsou ovšem stále důležité. Chceme-li, aby se náš produkt uživatelům příjemně používal, velice záleží na konkrétních úkonech, které budou uživatelé při používání našeho produktu vykonávat. Pokud budou například vytvářet data, pak je vhodnější tuto operaci provádět z webového prohlížeče na osobním počítači či notebooku. Pro testování si svých vědomostí ve volné chvilce na cestách, se uživatelům bude hodit spíše mobilní aplikace. Při vývoji našeho softwaru bychom měli zvážit pro jakou platformu náš software vytvořit, aby měli uživatelé z jeho užívání co nejlepší výslednou uživatelskou zkušenost.

Vývoj aplikací podle přesných požadavků klientů je však náročný a mnohdy zdlouhavý. Výstup z vývoje ve většině případů není při tvorbě našich dalších aplikací znovupoužitelný - kvůli autorským právům vytvořené aplikace, které vlastní naši klienti. Když se navíc podíváme na trh s mobilními aplikacemi, po aplikacích řešících stejný druh problému, pak můžeme nalézt mnoho aplikací jejichž nabízená funkcionalita je prakticky totožná. Ze své podstaty to dává smysl, když řeší stejný problém. Dané aplikace se mnohdy liší pouze vzhledem, konkrétní cílovou skupinou a konkrétním obsahem. Vezměme si například aplikaci pro testování znalostí uživatelů. Aplikace na trhu řešící tuto problematiku jsou zaměřeny na konkrétní testovaná témata - tím pádem jsou zaměřeny pouze na konkrétní skupinu uživatelů. Aplikace disponují fixní sadou otázek, u kterých často není prováděna ani jejich aktualizace.

Kdybychom dokázali vytvořit aplikaci pro testování znalostí uživatelů, která by nebyla závislá na konkrétním testovacím tématu, a tedy ani na cílové skupině uživatelů. Pak by byl náš produkt vhodný pro všechny uživatele, kteří problém testování uživatelů řeší. Otevřely by se nám tím nové možnosti variability aplikace. Produkt bychom mohli rozdělit na webovou a mobilní aplikaci. Testovaná témata spolu se zněním jejich otázek a odpovědí by bylo možné jednoduše spravovat z prostředí webové aplikace. Testování uživatelů by probíhalo z aplikace v mobilním zařízení testovaných uživatelů. Cílem této práce je takový software vytvořit.

1.1 Motivace

Rozsáhlá technologická vyspělost společnosti Facebook dala za vnik novým moderním technologiím. Jednou z těchto moderních technologií je i javascriptová knihovna React.js, která byla z počátku využívána pro interní vývoj společnosti. V roce 2012 byla pomocí této knihovny postavena celá klientská část aplikace Instagram.com a o rok později Facebook poskytl knihovnu k volném použití jako open-source. To nám dovoluje ji využít zcela k našim potřebám. Mohli bychom využívat starší technologie, ale je zbytečné vymýšlet kolo, když máme k dispozici vůz. Práce s knihovnou React.js již vyžaduje pokročilejší porozumění webovým technologiím a první pokusy o implementaci nemusí být vůbec jednoduché. Toto prvotní úsilí na sebe však nenechá dlouho čekat. Velice brzy se projeví, proč je znalost této knihovny, již po pár letech od její zpřístupnění, tak žádaná.

Uživatelé mobilních aplikací ovšem používají rozdílné platformy – nejčastěji operační systém Android a operační systém iOS. Pokud se tedy nechceme připravit o žádnou skupinu těchto uživatelů, pak musí být aplikace vyvíjena pro oba systémy. Ve světě vývoje mobilních aplikací to však znamená vyvinout dvě na sobě naprosto nezávislé aplikace. Z toho vyplývá dvojnásobné množství výdajů potřebných k vyvinutí koncového produktu. S příchodem nových technologií máme i druhou možnost. Vytvořit aplikaci, která bude multiplatformní - tedy logika a základní struktura aplikace bude nezávislá na konkrétní platformě. Jinými slovy kód aplikace bude umožňovat jeho kompilaci do nativní aplikace dané platformy.

Funkcionalita knihovny React.js byla v roce 2015 modifikována a rozšířena pro vytvoření ještě o něco komplexnějšího nástroje s názvem React Native. Tento framework je určen právě pro zmíněnou tvorbu kompletních mobilních aplikací nezávislých na platformě. Kód napsaný za pomocí frameworku React Native se totiž překládá do nativního kódu platformy Android respektive iOS.

Kapitola 2

Analýza

2.1 Vize projektu

V této vizi si vymezíme cíle projektu a stanovíme cílovou skupinu, pro kterou budeme náš software primárně vytvářet. Vymezení cílů projektu a cílové skupiny nám umožňuje vzájemně porovnat více (konkurenčních) projektů - ať už realizovaných či nerealizovaných. Usnadňuje nám také rychlou orientaci v řešené problematice. V neposlední řadě nám bude vize sloužit jako pevný bod, ke kterému se můžeme při vývoji projektu vracet.

Cíl projektu

Cílem tohoto projektu je analyzovat trh s mobilními aplikacemi, které se zabývají testováním uživatelů za pomoci testovacích otázek. Na základě této analýzy bychom měli být schopni vyvodit vhodné požadavky, kterými by měla vyvíjená aplikace disponovat. Následně je třeba zvolit adekvátní technologie určené pro vývoj webové i mobilní aplikace. U použitých technologií budou nastíněny jejich hlavní vlastnosti a důvod použití. Díky poznatkům získaných při analýze konkurence a s ohledem na použité technologie a jejich možnosti, budou navrhnuty a modelovány potřebné požadavky spolu s případy užití systému obou aplikací. Dále je zapotřebí dané modelované požadavky přenést do návrhu aplikace. Návrh aplikace bude obsahovat strukturu a architekturu aplikace a její grafický design v podobě prototypů. Pro dosažení zpětné

vazby nezávislých uživatelů bude na vytvořené prototypy aplikace aplikováno testování s uživateli. Výsledky testování by měli dát dostatečné poznatky vedoucí ke zlepšení návrhu prototypu aplikace. Návrh aplikací bude za pomoci diagramů reflektovat strukturu jednotlivých částí, ze kterých se aplikace skládá. V neposlední řadě budou obě aplikace dekomponovány na menší podproblémy, jejichž řešení bude postupně implementováno. Na závěr je potřeba provést řadu testování prostřednictvím adekvátních testovacích nástrojů. Výsledné testování obou aplikací prozradí případné nedostatky, které budou před vydáním aplikací eliminovány.

Cílová skupina

Výběr cílové skupiny je pro nás velice důležitý. Dává nám lepší představu o tom, jak bude projekt sloužit konkrétním lidem, v konkrétním prostředí. Jako cílovou skupinu si vybereme školní prostředí, jelikož se v něm tvůrce projektu pohybuje a je mu tak blízké a dobře známé. Uživatelé, kteří budou v naší aplikaci vytvářet otázky a testovat koncové uživatele budou učitelé. Naopak koncoví uživatelé, kteří budou testováni, budou studenti. Vymezením cílové skupiny se stávají požadavky na tvorbu aplikace méně abstraktními. Výběr cílové skupiny ovšem neznamená limitaci využití našeho produktu pouze na danou skupinu uživatelů. Pokud však projekt nemá cílovou skupinu, pak má jeho návrh a následná implementace za snahu vyhovět všem různým typům uživatelů. Je tedy snaha o to dělat aplikaci co možná nejvíce univerzální, a tím se řeší mnohdy zbytečné, jindy nežádoucí kompromisy, které je potřeba při vývoji aplikace vytvářet. Při testování s uživateli se mnohdy zjistí, že aplikace obsahující nespočet kompromisů, tak aby vyhovovala všem, nevyhovuje vlastně vůbec nikomu. Proto je třeba vyvarovat se kompromisům a snaze o univerzálnost projektu a je třeba se zaměřit na jednu až dvě cílové skupiny, pro které budu projekt primárně vytvářet. Pokud se projekt osvědčí, mohu ho kdykoliv upravovat dle požadavků a přání koncových uživatelů jiné cílové skupiny.

Cílová platforma

Framework React Native nabízí možnost tvorby aplikace, která lze přeložit do nativní aplikace pro operační systém Android i operační systém iOS. Tyto dva operační systémy však mají odlišné konvence a standardy v oblasti designu - vzhledu a rozvržení jednotlivých elementů. Proto bude mít výběr platformy podobný důvod, jako výběr cílové skupiny. Nebudeme dělat žádné kompromisy v tom, kterými pokyny se v návrhu řídit. Vybereme si jednu

konkrétní platformu a zaměříme se na její oficiálně doporučené designové pokyny k návrhu vzhledu a rozmístění komponent.

Za cílovou platformu si zvolíme operační systém Android, a to ze dvou důvodů. Prvním důvodem je omezení, které stanovila společnost Apple. Omezení spočívá v nemožnosti vývoje pro platformu iOS, na jiném zařízení než na macOS v prostředí xCode [1]. Druhým důvodem je, že Android nabízí podstatně větší možnosti emulace mobilního zařízení v osobním počítači, který budeme hodně využívat při vývoji mobilní aplikace. Budeme se tedy řídit tzv. Android Design Guidelines [2].

2.2 Analýza konkurence

Chceme-li vytvořit nějaký produkt nebo nabízet nějaké služby, musíme nejdříve úspěšně analyzovat konkurenci. Na základě detailní analýzy konkurence můžeme vyvodit hned několik důsledků. Můžeme například zjistit jakým způsobem konkurence pracuje, jaké jsou jejich postupy, v čem jsou silní či naopak slabí. Nejdůležitější poznatek, který můžeme z výsledků testování získat je zjištění, jestli je náš projekt, respektive vyvíjený software konkurence schopný a má v nějakém ohledu oproti konkurenci navrch. V opačném případě uživatelé upřednostní konkurenční produkt, tím náš projekt na trhu propadne.

V našem případě se jedná o aplikaci, která testuje znalosti uživatelů pomocí sady otázek a správných odpovědí formou cvičného a ostrého testu. Budeme tedy hledat podobné aplikace. Po vybrání adekvátní množiny testovacích subjektů budeme aplikace jednu po druhé testovat na základně určitých stanovených kritérií. Z výsledků testování vytvoříme zhodnocení shrnující nejdůležitější poznatky, které z testování aplikací vyplynuly, a jenž bychom měli zanést do návrhu naší aplikace.

2.2.1 Kritéria testování konkurence

Abychom mohli aplikace testovat a následně je adekvátně porovnat, musíme vymezit jednoznačná kritéria, na jejichž základě bude evaluace jednotlivých aplikací probíhat. Hodnocení by jinak bylo příliš netransparentní a bylo by obtížné z něj vyvodit reálné poznatky.

Mezi naše kritéria pro testování aplikací jako první zařadíme grafické rozhraní aplikace. Při testování se díváme, jestli mají grafické prvky vhodnou velikost a odsazení dle oficiálních standardů pro design aplikace, jestli k sobě prvky vzájemně barevně ladí a zdali použitá grafika odpovídá cílové skupině uživatelů. Obecně například platí, že by se neměly míchat různé styly písem a mělo by být použito pouze pár k sobě se hodících barev. Vyhodnotíme také výsledný grafický dojem z aplikace.

Dále budeme hodnotit uživatelskou zkušenost s aplikací. Otestujeme, jestli se uživatel dokáže v aplikaci orientovat a rozpoznat, jakou akci musí provést, aby dosáhl cíleného dílčího kroku v aplikaci. Uživatel by měl od aplikace dostávat dostatečnou zpětnou vazbu, která by ho měla informovat o úspěchu či neúspěchu jeho akce v aplikaci. Žádaná vlastnost je i asociace barev s konkrétními akcemi uživatele dle konvencí. Například červená barva je asociována s mazáním dat.

Aplikace by měla zabraňovat uživateli udělat zbytečné chyby. Pokud nějaké chyby nastanou, měl by být uživatel schopný se z nich zotavit. Dalším požadavkem na dobrou aplikaci je její rychlost. Aplikace by měla být dostatečně rychlá na to, aby uživatel zbytečně nečekal na to, až bude moci nějakou akci provést.

Na základě těchto stanovených kritérií, které vycházejí z obecně platných standardů testování mobilních aplikací, budeme jednotlivé testované subjekty evaluovat. Očekáváme, že nám výsledky testování přinesou inspiraci a několik podnětů, na základě kterých navrhneme a implementujeme konkurenci schopnou aplikaci.

2.2.2 Testované subjekty

Aplikace uvedené v tabulce 2.1 byly zvoleny jako testovací subjekty. Následně byly jedna po druhé testovány na mobilním zařízení. Testování aplikací proběhlo na základě daných kritérií, které jsme v dokumentu specifikovali.

2.2.3 Výsledky testování

Mnoho testovaných aplikací zobrazovalo reklamy. Některé z aplikací vykazovaly grafický design adekvátní výrazně mladší generaci, než byla před-

Avition exam	English Test
TestBank	Grammer Up
Driving Theory	Firefighter
Test English	AP Psychology
Life in UK	Theory Genius
Smart CNA	Zbraně kvalitně

Tabulka 2.1: Tabulka testovaných subjektů

pokládaná cílová skupina uživatelů. Často měla aplikace mnoho přídavné funkcionality, které však ve většině případů udělaly aplikaci nepřehlednou. Tím bylo znemožněno příjemné použití aplikace k hlavním účelům, pro které byla aplikace vytvořena. Často chyběla zpětná vazba uživateli o tom, jak dopadla právě provedená akce. Vyskytlo se i několik případů pádů celé aplikace při jejím rutinním používání. Mnoho testovaných subjektů vyžadovalo k jejich používání registraci.

Definovaná kritéria pro testování zvládly úspěšně aplikace FireFighter, Smart CNA a Zbraně kvalitně. Při návrhu a následném vývoji mobilní aplikace se jimi budeme inspirovat.

2.2.4 Vyvození poznatků

Povaha naší cílené aplikace vyžaduje koncentraci uživatele. Reklamy odvádějí uživatelovu pozornost, což může vést ke zkreslení výsledků jeho znalostí v testu. Měli bychom se tedy zcela jistě nasazení reklam do aplikace vyhnout a veškeré finance z aplikace získat už při jejím prodeji.

Co by však mohlo uživatele odradit v naprostém počátku používání aplikace je registrace. Procesy některých registrací byly příliš zdlouhavé. V mnoha případech nepřidávala aplikaci možnost, respektive nutnost registrace a následného přihlášení žádnou přidanou hodnotu. Vytvoříme tedy aplikaci, která bude připravena ihned k použití bez zbytečné registrace.

Design aplikace hraje důležitou roli. Kvůli špatnému designu může být uživatel odrazen od používání aplikace nebo dokonce již od její instalace. Navrhneme tedy jednoduchý elegantní design, ve kterém se uživatel snadno zorientuje. Použijeme barevné kombinace, vygenerované online barevnými generátory jako je například nástroj Paletton.[3]

Testované subjekty nabízejí rozdílné dílčí doplňkové funkcionality a odlišují se grafickým vzhledem, v principu však fungují velice podobně. Můžeme nalézt dokonce téměř identické aplikace lišící se pouze tématikou testování a vzhledem. Neustále jsou tedy vytvářeny různé aplikace, které řeší stejný problém - testování uživatelů za pomocí otázek, a tak sdílí mnoho společné funkcionality. Tento fakt je pro nás velice důležitý. Představme si aplikaci, která by byla řešením na celou problematiku testování uživatelů za pomoci otázek a nebyla by závislá na předmětu testování, a tím pádem ani na cílové skupině testovaných uživatelů. Přesně takovou aplikaci se pokusíme v rámci našeho projektu vytvořit.

2.3 Modelování požadavků

2.3.1 Funkční požadavky webové aplikace

1. Systém bude umožňovat vytváření uživatelů, kteří budou moci po jejich přihlášení vytvářet další uživatele.
2. Systém bude umožňovat zaslání přihlašovacích údajů vytvořenému uživateli na email.
3. Přihlášený uživatel bude moci vytvářet editovat a mazat témata, a jednotlivá témata přiřadit jednomu z uživatelů aplikace.
4. Přihlášený uživatel bude moci vytvořit a smazat otázky a k nim odpovídající odpovědi k jednomu z vytvořených témat.
5. Systém bude umožňovat vytvořit otázky dvou typů. První typ otázky bude mít právě jednu správnou odpověď. Druhý typ otázky bude mít libovolný počet správných odpovědí, maximálně však rovný počtu celkových odpovědí. Maximální počet odpovědí se stanoví při návrhu aplikace.
6. Přihlášený uživatel bude mít možnost si typ otázky kdykoliv rozmyslet a změnit otázku z jednoho typu na typ opačný, a to při zachování již vyplněných dat.
7. Systém bude ke každému vytvořenému předmětu generovat jeho kód na základě kterého si uživatelé mobilní aplikace předmět spolu se všemi jeho otázkami a odpovědmi stáhnou do jejich mobilního zařízení.
8. Přihlášení uživatelé si budou moci zobrazit výsledky ostrých testů uživatelů mobilní aplikace.

Chrome	Firefox	Safari	Edge
46.0	43.0	8.0	14.0

Tabulka 2.2: Verze webových prohlížečů

9. Přihlášený uživatel uvidí data, která souvisí s jeho osobou, graficky odlišené.

10. Systém bude umožňovat přihlášeným uživatelům, kteří jsou zároveň administrátoři, vygenerovat nové heslo. Heslo se automaticky odešle prostřednictvím systému na email uživatele, kterému bylo obnoveno.

11. Přihlášený uživatel, který je zároveň administrátor bude moci z aplikace smazat již neaktuální uživatele. Nebude však moci smazat vlastníka aplikace a sebe samotného.

2.3.2 Kvalitativní požadavky webové aplikace

1. Aplikace bude funkční a optimalizovaná v prohlížečích Firefox, Chrome, Safari a Edge v minimálních verzích vyznačených v tabulce 2.2

2. Aplikace bude disponovat vysokou rychlostí, kterou zajistí povaha jednostránkové aplikace spolu s asynchronní načítání dat ze serveru.

3. Aplikaci bude snadné používat a její používání povede k snadnému osvojení si jednotlivých úkonů. Úkony prováděné v aplikaci budou intuitivní a transparentní.

4. Aplikace bude optimalizovaná pro většinu obvykle používaných monitorů, tedy monitorů od šířky 1366px do šířky 2560px.

5. Plné používání aplikace budou umožňovat i tabletové zařízení - tedy od šířky 768px výše. Na tabletových zařízeních však nemusí být design aplikace vždy dotažen do detailů.

6. Aplikace bude odolná proti základním typům útoků na webové aplikace jako je například SQL injection a Cross-site scripting (XSS).

2.3.3 Funkční požadavky mobilní aplikace

1. Uživatelé systému budou moci přidávat a mazat předměty. Systém automaticky s přidáním předmětu přidá i jeho testové otázky spolu s jejich

odpovědmi. Všechny tyto data zůstanou uchována v paměti telefonu uživatele.

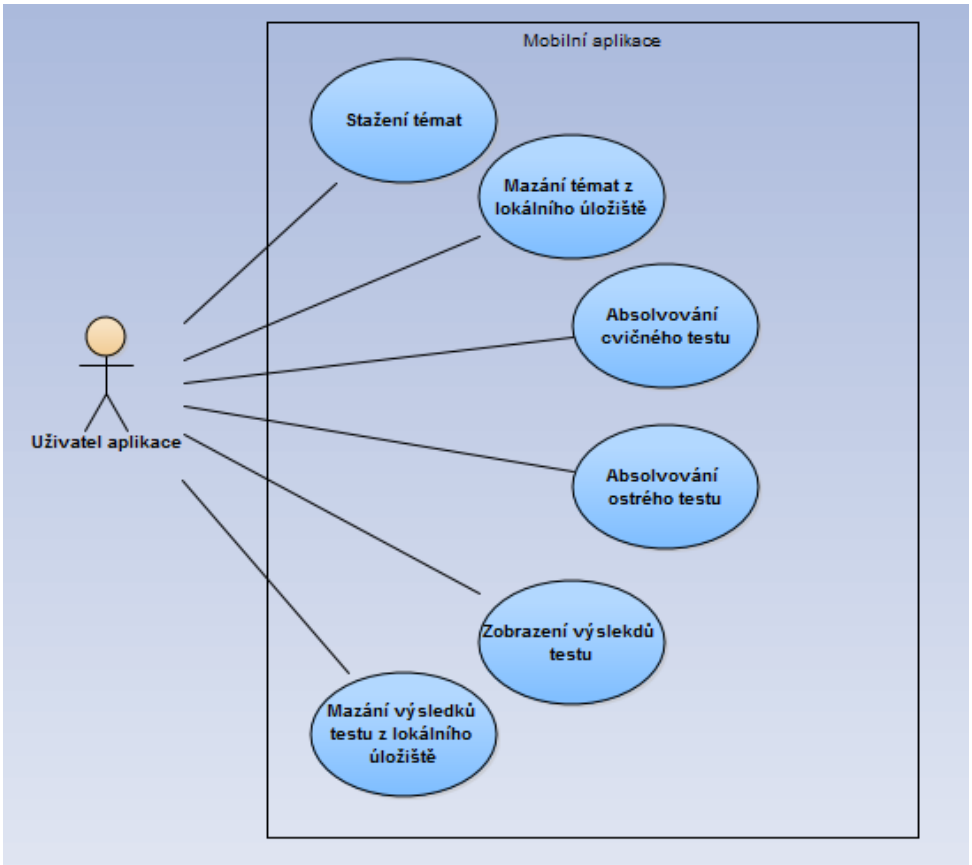
2. Uživatel bude moci jakýkoliv přidaný předmět smazat. Systém s odstraněním předmětu odstraní i jeho asociované otázky z paměti telefonu uživatele.
3. Systém umožní uživatelům procvičit si otázky formou cvičného a ostrého testu.
4. Systém nabídne uživatelům po dokončení cvičného respektive ostrého testu přehled o úspěšnosti testu.
5. Systém bude výsledky testů uchovávat v paměti uživatele telefonu.
6. Uživatel si bude moci zpětně zobrazit a případně smazat svoje lokální instance výsledků dokončených testů.
7. Po dokončení ostrého testu uživatelem zašle systém výsledky testu do webové aplikace, kde budou zobrazeny přihlášeným uživatelům webové aplikace.

2.3.4 Kvalitativní požadavky mobilní aplikace

1. Aplikace bude fungovat na verzi systému Android 4.4.2 a vyšší.
2. Aplikace nepřesáhne velikost 25 MB při iniciálním stavu komponenty aplikace, tedy při neuložených datech aplikace v interní paměti telefonu.
3. Aplikace bude vyžadovat internetové připojení pouze při přidávání nového předmětu a při absolvování ostrého testu. Ostatní operace v aplikaci budou na použití internetového připojení nezávislé.

2.3.5 Společné kvalitativní požadavky aplikací

1. Obě aplikace budou mít čistý přehledný kód formátovaný podle oficiálních standardů. Kód bude členěn do souborů a modulů. Jednotlivé názvy proměnných a funkcí budou samovysvětlující, psané konvencí camelCase, kde jen to bude možné.
2. Aplikace budou jednoduché na údržbu i pro jejich budoucí rozšíření o přídavnou funkcionalitu.
3. Aplikace budou splňovat aktuální trendy moderního vývoje webových respektive mobilních aplikací.



Obrázek 2.1: Use case diagram mobilní aplikace

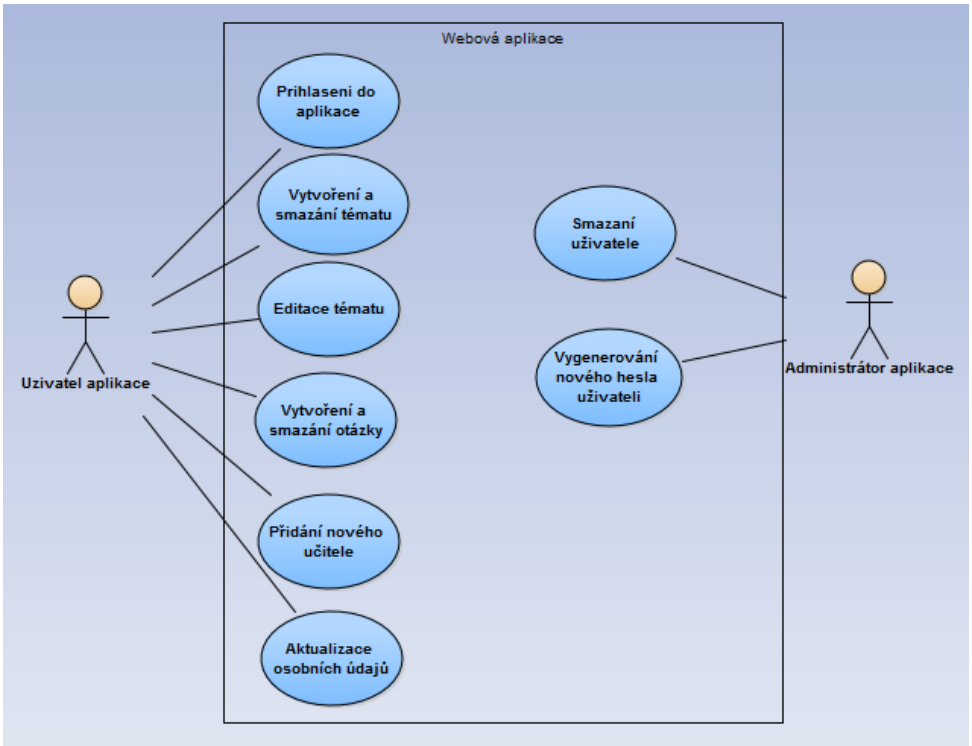
2.4 Uživatelské role a případy užití

Aplikace bude rozlišovat následující role:

- **Přihlášený uživatel webové aplikace** - uživatel, který se prokázal správnými přihlašovacími údaji.

  Administrátor webové aplikace - uživatel, který má stejné možnosti a oprávnění jako přihlášený uživatel webové aplikace. Navíc má možnost mazání uživatelů z aplikace a generování nových uživatelských hesel.
- **Uživatel mobilní aplikace** - každý uživatel, který si stáhne mobilní aplikaci.
- **Vlastník** - člověk, který obě aplikace spravuje a vlastní.

Uživatelské role a případy užití jsou zachyceny na diagramu 2.1 a 2.2



Obrázek 2.2: Use case diagram webové aplikace

2.4.1 Přihlášený uživatel webové aplikace

Přihlášený uživatel webové aplikace bude moci vytvářet uživatele stejného typu - kolegu, kterému přiřadí jméno, uživatelské jméno a email. Uživatelské jméno a email musí být v systému unikátní. Na zadaný email uživatele jsou zaslány jeho přihlašovací údaje, konkrétně uživatelské jméno a systémem vygenerované heslo. Aplikace bude zobrazovat aktuální uživatele aplikace včetně jejich veřejných údajů.

Přihlášený uživatel webové aplikace uvidí data spojena s jeho osobou graficky odlišené, respektive zvýrazněné. Graficky odlišen bude jeho účet v seznamu uživatelů, témata aktuálně vyučována jeho osobou a výsledky uživatelů mobilní aplikace z jím vyučovaných témat. Pokud se změní učitel tématu, pro které již existují ve webové aplikaci výsledky, pak zůstane grafická asociace předešlých výsledků s předešlým učitelem. Jinými slovy až nové výsledky se budou graficky asociovat s novým učitelem tématu.

Přihlášený uživatel webové aplikace může všechny své osobní údaje aktualizovat. Tedy jméno, uživatelské jméno, email a heslo lze změnit ovšem stále za předpokladu, že je uživatelské jméno a email v systému unikátní.

Dále bude moci vytvářet témata, u kterých je potřeba zadat jejich název a vybrat učitele tématu ze seznamu uživatelů systému. Všechny témata lze každým z uživatelů smazat nebo editovat jejich název či samotného učitele tématu. Vytvořené téma je ihned po jeho tvorbě bez jakéhokoliv načítání vidět v seznamu témat, které zobrazují název tématu, jeho učitele a kód tématu vygenerovaný systémem. Tento kód mohou uživatelé webové aplikace dále distribuovat uživatelům mobilní aplikace. Distribuce kódu však již probíhá mimo systém.

Poté bude mít možnost vytvořit ke zvolenému tématu otázku s jednou možnou odpovědí a otázku s více nebo žádnou možnou odpovědí. Přímo u tvorby otázky bude uživatel vytvářet a označovat správné odpovědi na dané otázky. Počet odpovědí bude moci v průběhu tvorby otázky kdykoliv dynamicky měnit. Během tvorby otázky bude moci kdykoliv změnit i její typ, na což aplikace adekvátně zareaguje změnou grafického rozhraní a elementů pro zadávání opačného typu otázky. Uživatel si bude moci přehledně zobrazit otázky z různých témat, díky filtru témat v podobě výběrového boxu na stránce s otázkami. Otázky bude moci uživatel smazat.

V neposlední řadě si uživatelé webové aplikace mohou zobrazit výsledky ostrých testů uživatelů mobilní aplikace, tedy název tématu, který byl testován, úspěšnost jejich správně zodpovězených otázek ku počtu špatně zodpovězených otázek, procentuální úspěšnost a datum dokončení testu. Tyto jednotlivé výsledky bude moci uživatel ze systému odstranit.

2.4.2 Administrátor webové aplikace

Administrátor webové aplikace může v aplikaci dělat totéž, co přihlášený uživatel webové aplikace. Navíc může zažádat o obnovu hesla uživateli, který své heslo zapomněl. Systém na jeho žádost automaticky vygeneruje nové heslo pro daného uživatele a nové přihlašovací údaje zašle danému uživateli na jeho email.

2.4.3 Uživatel mobilní aplikace

Uživatel mobilní aplikace - bude moci přidávat předměty prostřednictvím kódu předmětu, který byl uživatelům sdělen jinými dorozumívacími prostředky. Přidané předměty bude moci kdykoliv smazat.

Uživatelé mobilní aplikace si mohou procvičit otázky formou cvičného testu, který jim hned po zodpovězení otázky sdělí její správné odpovědi. Budou moci absolvovat i test ostrý, který jim nebude dávat žádnou zpětnou vazbu v průběhu testu. Oba testy však uživateli zobrazí informaci o celkovém výsledku testu. Výsledky testů budou obsahovat totožné atributy, které jsou uvedeny v popisu výsledků uživatelské role **Přihlášený uživatel webové aplikace**.

2.4.4 Vlastník

Vlastník - má všechny vlastnosti administrátora webové aplikace a dokáže pomocí příkazu v konzolovém řádku založit iniciálního uživatele, respektive administrátora webové aplikace.

2.5 Analýza platforem a vývojových nástrojů webové aplikace

V této kapitole analyzujeme dostupné platformy a vývojové nástroje pro tvorbu webové aplikace. Následně provedeme výběr těch nejvhodnějších technologií, které budou splňovat funkční a kvalitativní požadavky webové aplikace.

2.5.1 Nativní aplikace

U mobilní aplikace chceme docílit příjemného a rychlého používaní. Chceme docílit toho, aby uživatel nemusel čekat na načítání částí aplikace. Proto byla upřednostněna nativní aplikace pro konkrétní platformu, před aplikací webovou. Díky tomu, že je nativní aplikace vytvořena pro konkrétní operační systém, dokáže využívat hardwarových schopností telefonu. Oproti webové aplikaci často potřebuje pouze minimální nebo dokonce žádné internetové připojení – šetří tedy mobilní data a je zpravidla rychlejší na používání.

2.5.2 Webová aplikace

Pro dosažení nezávislosti aplikace na daném tématu, respektive na konkrétní cílové skupině uživatelů, bude potřeba jednotlivé testovací témata vytvářet a spravovat. Správa témat zahrnuje vyplňování otázek a odpovědí, kterých může být i velice mnoho. S touto operací budeme při vývoji aplikace počítat jako s velice časově náročnou. Dáme si záležet, abychom tvorbu otázek a odpovědí udělali co možná nejrychlejší a nejintuitivnější. Pro zadávání textových dat do aplikace se jako médium jeví nejrychleji osobní počítač. Proto bude aplikace pro vytváření a správu otázek vytvořena za pomocí webových technologií.

2.5.3 Izomorfní aplikace

Při vývoji webových aplikací představuje hranice mezi klientskou a serverovou stranou aplikace zeď, na jejíž druhé straně je naprosto jiný svět. Typicky chceme data z prezentační vrstvy klientské strany webové aplikace dostat do logické vrstvy webové aplikace. Ta je musí adekvátně zpracovat a poslat modelové vrstvě, který zpracovaná data uloží do databáze. Případně chceme naopak data z databáze zobrazit a tak aplikovat obrácený postup. Pokud do celého procesu zapojíme ještě mobilní aplikaci, pak může složitost komunikace a zpracování dat nepříjemně narůst.

My jsme však při výběru technologií s tímto faktem počítali. Proto jsme vybrali pro implementaci klientské a serverové části webové aplikace technologie, které využívají stejný programovací jazyk - Javascript. Aby toho však nebylo málo, tak i implementace mobilní aplikace proběhne za pomoci Javascriptu. Takové aplikace se označují za izomorfní. Izomorfní aplikace nám dává značné možnosti. Můžeme například přejímat kousky kódu z nějaké části naší aplikace a sdílet jej do jiné části. Tuto schopnost oceníme především při práci s daty. Je tím zároveň splněn dobrý princip znovupoužitelnosti kódu z objektově orientovaného programování.

2.5.4 Klientská část webové aplikace

HTML 5

HTML [4] Byl primárně navržen, aby popisoval dokumenty z hlediska sémantiky. Jeho postupná modifikace v průběhu let však způsobila, že se pomocí HTML začali popisovat různé typy dokumentů. HTML je definován metajazykem SGML a až do jeho verze 5 byl překládán prohlížeči z důvodů kompatibility jako SGML dokument. Jeho pátá verze z roku 2014 je verzí aktuálně nejnovější a přináší mimo jiné použití nových API, jako je například Drag and Drop API. Pomocí jazyka HTML 5 budeme popisovat logickou strukturu dokumentu webové aplikace.

CSS 3

CSS [5] je jazyk pro popis způsobů zobrazení HTML, XML a XHTML elementů. Za vytvořením CSS stojí též komunita W3C. Tento jazyk pro kaskádové styly se skládá ze selektorů, které vybírají množinu elementů a definují jim atributy. Atributy následně určují nejrůznější vizuální vlastnosti. Třetí verze jazyka CSS je plně zpětně kompatibilní a přináší se sebou mimo jiné nové možnosti v oblasti využití grafické karty za účelem akcelerace výpočtů, jako je například hardwarová akcelerace při použití 3D transformací. Pomocí kaskádových stylů vytvoříme vizuální část webové aplikace.

Javascript ES6

Jazyk Javascript od společnosti Netscape se svezl na tehdejším úspěšném programovacím jazyku Java a jeho název použil při vytvoření jména nového multiplatformního jazyka [6]. Ačkoli Javascript spadá do rodiny jazyků C, C++, Java a je objektově orientovaný, jeho sémantika je velice odlišná. Namísto tříd používal až do své verze ES6 prototypovou dědičnost (která ovšem s verzí ES6 zůstává zachována). Javascript dlouhé roky sloužil především k manipulaci s interaktivními prvky na stránce. Byl tedy spouštěn na klientské straně, kde patřičně modifikoval uživatelův DOM [7]. V posledních několika letech se však Javascript začal používat pro celou škálu technických řešení. To vše především díky vzniku mnoha javascriptových frameworků. Za zmínku stojí i stále se rozšiřující zájem o používání Javascriptu na serveru. Rozšířený

zájem komunity způsobila především sada rozhraní a knihoven s názvem Node.js [8].

Jazyk Javascript je primárním jazykem naší webové a mobilní aplikace. Díky podpůrným nástrojům můžeme využívat jeho novou specifikaci ES6, která ještě nebyla do webových prohlížečů oficiálně implementována. Podpůrné nástroje však implementují polyfilly, které specifikaci ES6 překládají do starší verze jazyka Javascript již aktuálně podporované webovými prohlížeči.

jQuery

jQuery je javascriptová knihovna, která si klade za cíl usnadnit uživateli zápis Javascriptu. Knihovna jQuery definuje přehlednější, lépe pochopitelnou a zapamatovatelnou syntaxi. Má kvalitní dokumentaci a rozsáhlou komunitu uživatelů. Díky těmto aspektům budeme knihovnu jQuery používat, a to především na práci s klientskou DOM manipulací.

React.js

React.js nová moderní javascriptová knihovna pro webových aplikací. Tato knihovna se zaměřuje pouze na tvorbu uživatelských rozhraní. Díky své specifičnosti dokázala přinést velice robustní a propracované řešení na většinu situací, které s návrhem a implementací uživatelského rozhraní souvisejí. Tím se liší například od relativně oblíbeného frameworku Angular.js, který se snaží díky své povaze aplikačního rámce obsáhnout řešení pro komplexní oblast problémů. To se odráží na výsledné kvalitě některých řešení, které nejsou z nejrůznějších důvodů, dle názoru specialistů, úplně ideální. Nedotažená řešení se stávají obětí nejrůznějších internetových článků, které na tyto nedostatky upozorňují [9].

Knihovna React.js je stěžejní technologií, kterou použijeme pro implementaci webové aplikace. Splní naše požadavky na rychlost, jelikož modifikuje jen tu část aplikace, která se změnila. Nemusíme tedy čekat na zaslání celé nové stránky, jako je tomu například u PHP, ale veškerá interakce uživatele s aplikací bude probíhat pouze za pomoci asynchronních požadavků na server. Taková aplikace se anglicky nazývá Single Page Application [10].

Sass

Sass [11] je preprocesor jazyka CSS. Sass je ve své podstatě rozšíření jazyka CSS, ne ovšem ve smyslu nových atributů, které lze použít, ale ve smyslu syntaktickém. Zpřehledňuje a usnadňuje zápis jednotlivých stylů například pomocí zanořování selektorů do sebe. Nabízí toho ovšem mnohem více, jako je třeba definice a používání proměnných. K jeho kompilaci do jazyka CSS je zapotřebí využití externích nástrojů, jako je například Webpack.

Yarn

Je poměrně nový správce javascriptových balíčků, nahrazující staršího správce balíčků npm. Balíčky obsahují většinou nějakou knihovnu, kterou je možno si do projektu nainstalovat a využívat jejich funkcí. O proti npm je Yarn výrazně rychlejší, nabízí více možností a lepší zabezpečení. Pomocí Yarnu budeme instalovat a spravovat balíčky obou našich aplikací. To nám dá nad aplikacemi a využívanými moduly značný přehled a kontrolu.

Webpack 2.0

Webpack se stal jedním z nejdůležitějších nástrojů pro moderní vývoj webových aplikací. Je to primárně balíček modulů pro vývoj Javascriptu. Lze pomocí něj však transformovat i naše soubory, které se využívají na klientské straně webové aplikace. V neposlední řadě dobře zpracovává balíčky instalované správcem balíčků Yarn.

V našem případě využijeme Webpack především pro usnadnění a urychlení vývoje prostřednictvím tzv. hot reload funkce, díky které se obnoví jen změněná část stránky a výsledek změny je okamžitě vidět bez dodatečného čekání a načítání stránky. Dále Webpack použijeme pro kompilaci Sass souborů a zavedení nástrojů určených pro kompilaci nové specifikace javascriptu ES6 do jeho starší verze.

Twitter bootstrap

Populární HTML, CSS a Javascriptový framework pro tvorbu responzivních webových aplikací. Díky němu dosáhneme ve webové aplikaci škálovatelného designu, který se bude přizpůsobovat velikosti zobrazovacího zařízení.

2.5.5 Serverová část webové aplikace

Node.js

Náší serverovou část aplikace budeme implmenetovat sadě rozhraní a knihoven obalující implementaci Javascriptu Node.js. Tato sada dovoluje jednoduše spouštět javascriptový kód i mimo webový prohlížeč. Node.js byl vydán v roce 2009, stal se velice oblíbený a zastínil konkurenční CommonJS. Nyní se považuje za standardním řešení pro serverové vykonávání Javascriptu [12]. Použijeme ho ke zpracování požadavků na serveru, abychom dosáhli izomorfizmu aplikace.

MongoDB

Při výběru databáze jsme se zaměřili na to, s jakými daty budeme pracovat. Data budou dynamicky se měnící. Jelikož budeme v obou aplikacích využívat javascriptových objektů, tak by bylo ideální, kdyby data měli obdobnou strukturu. Proto upřednostníme noSQL databázi před databází relační. Jako konkrétní databázi zvolíme databázi MongoDB a to především díky dobrým referencím vývojářů. MongoDB umožňuje snadnou tvorbu a integraci dat.

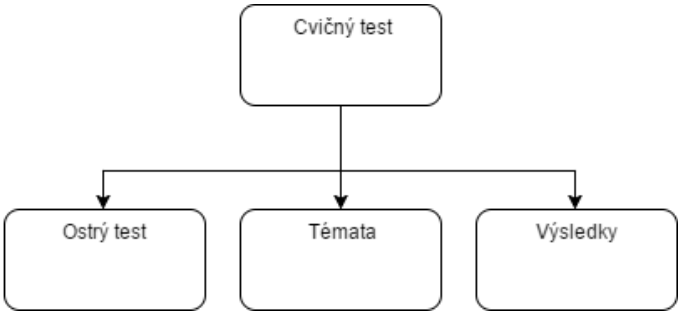
Kapitola 3

Návrh

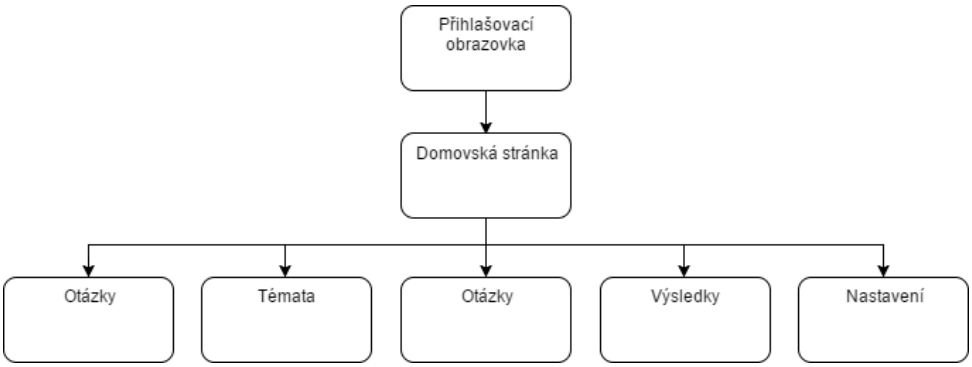
Provedli jsme nezbytnou detailní analýzu řešeného problému, konkurence, vývojových nástrojů a požadavků aplikace, abychom mohli vytvořit adekvátní návrh aplikace. Na základě návrhu bude provedena implementace obou aplikací vycházející z požadavků zmíněných v analýze. V této kapitole se zaměříme na návrh struktury a architektury aplikace. Vytvoříme grafický prototyp aplikace, který necháme otestovat uživatelským testováním. Na základě výsledků uživatelského testování prototypy aplikace náležitě upravíme a navrhneme prezentační vrstvu naší aplikace. Na závěr vytvoříme analytický diagram tříd, který bude sloužit jako předloha pro návrh struktury databáze.

3.1 Struktura aplikace

Návrh struktury aplikace nám dává představu o tom, jakým způsobem lze aplikaci procházet. Jinými slovy popisujeme, z jaké obrazovky se můžeme kam dostat. Při návrhu jednotlivých obrazovek, kterými bude uživatel procházet, byla snaha o to, aby byly intuitivní a co nejjednodušší. Uživatel se tím pádem nebude v aplikaci zbytečně zanořovat. Mělo by to vést ke zrychlení používání aplikace, tedy o náš požadavek. Přihlášený uživatel webové aplikace bude mít z domovské stránky ihned k dispozici sekci učitelů, témat, předmětů, otázek, výsledků a osobního nastavení. Uživatel mobilní aplikace bude mít ihned po spuštění aplikace otevřenou záložku cvičného testu a bude moci přejít přímo do sekce ostrého testu, předmětů a svých výsledků. Návrh struktury obou aplikací je nastíněn na obrázcích 3.1 a 3.2, které byly vytvořeny prostřednictvím aplikace draw.io [13].



Obrázek 3.1: Struktura webové aplikace.



Obrázek 3.2: Struktura webové aplikace.

3.2 Prototypování

Na základě požadavků byly vytvořeny prototypy webové a mobilní aplikace pomocí online nástroje NinjaMock [14]. Z toho vyplynula základní představa o tom, jak by aplikace mohla vypadat. Tato představa však musí být otestována nezávislými uživateli, kteří nám mohou dát zpětnou vazbu. Zpětnou vazbu následně zohledníme při úpravách obou prototypů.

Prototypy mobilní aplikace zachycuje obrázek 7.1. a prototypy webové aplikace jsou zobrazeny na obrázku 7.2.

3.3 Testování s uživateli

Oba prototypy aplikace jsme nechali podrobit uživatelskému testování. Uživatelé měli na každou obrazovku jasně stanovený cíl, kterého měli dosáhnout. Testováno bylo 8 uživatelů. Každý měl na průchod testem stanovených 15

minut. Jelikož jsou prototypy pouze v podobě tiskové grafiky, proběhlo tudíž testování uživatelů metodou Wizard of Oz [15].

3.3.1 Výsledky testování

Z výsledků testování jsme získali zpětnou vazbu, která vedoucí k odstranění chyb a nedokonalostí, kterých jsme se při tvorbě prototypů dopustili. Uživatelé jsou v tabulce označeni písmenem u a přiřazeným číslem. Na základě výsledků testování prototypů webové aplikace jsme zjednodušili proces vytváření otázek. Obdobný test byl proveden pro prototyp mobilní aplikace, na jehož základě jsme zjednodušili orientaci v aplikaci.

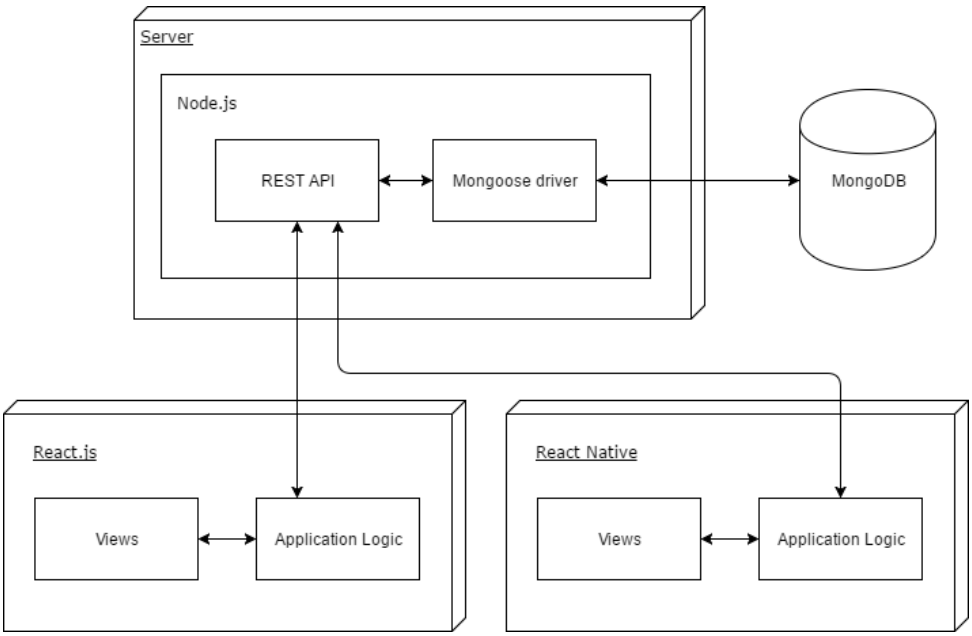
Akce	u1	u2	u3	u4	u5	u6	u7
Orientace v aplikaci je intuitivní	ano	ano	ano	ano	ano	ano	ano
Proces vytváření otázek je pochopitelný	ano	ne	ne	ano	ne	ne	ano
Aplikace reaguje očekávaně	ano	ne	ano	ano	ano	ano	ano
Tabulky s daty jsou přehledné	ano	ano	ano	ano	ano	ano	ano

Tabulka zachycující testování prototypů webové aplikace.

3.4 Architektura aplikace

Při návrhu architektury webové aplikace jsme řešili primárně problém hostování aplikace. Serverová část totiž bude napsána v Node.js. České hostingové společnosti však tento typ aplikací nepodporují. Nabízely se tedy dvě možnosti jak aplikaci úspěšně umístit na internet. První možnost byla pronájem VPS neboli virtuálního privátního serveru. Zahrnovalo by to ale kompletní konfiguraci serveru od továrního nastavení. Využili jsme tedy raději možnost hostování aplikace na serveru společnosti Heroku. Ta má již pro tento účel vytvořený kontejner na Node.js aplikace. Byli jsme tedy ušetřeni náročné konfigurace serveru od továrního nastavení.

Další problém byl v otázce přístupu mobilní a webové aplikace ke společnému médiu za účelem výměny dat. Na základě doporučení vedoucího práce a odborníka na danou problematiku bylo nakonec zvoleno aplikační rozhraní REST [16]. Díky němu můžeme provádět CRUD operace, neboli operace vytvoření, čtení, aktualizace a mazání dat. Aplikační rozhraní REST bude



Obrázek 3.3: Architektura aplikace.

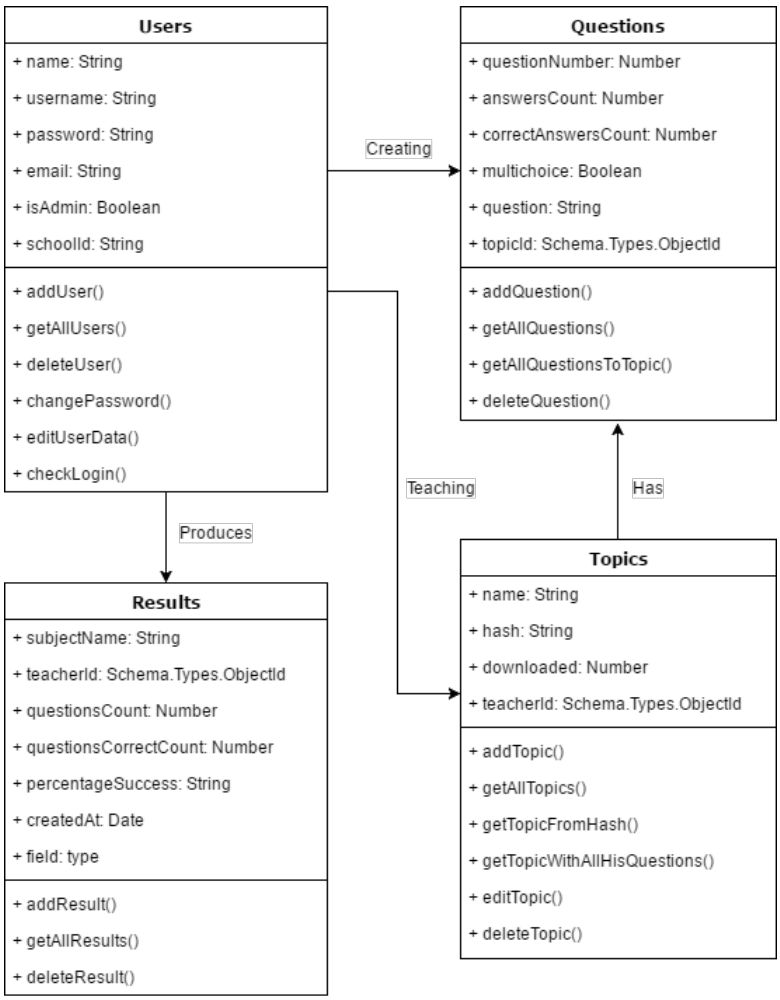
umístěno na serveru, respektive jeho implementace bude provedena pomocí serverového Node.js. Rozhraní REST bude tedy umožňovat výměnu dat mezi databází MongoDB a oběma aplikacemi.

Data budou z grafických rozhraní aplikací předávána do jejich řídící logiky, odkud budou odesílány na aplikační rozhraní REST. Aplikační rozhraní zavolá příslušné metody modelů a ty následně v databázi provedou dané úkony, popřípadě vrátí výsledek opačným procesem zpět. Při zobrazování dat z databáze zašle logická vrstva aplikace požadavek na aplikační rozhraní REST, které získá z databáze požadované data a vrátí je zpět logické vrstvě, která zašle získaná data do pohledové vrstvy k jejich zobrazení uživateli.

Obrázek 3.3 popisuje architekturu aplikace.

3.5 Analytický diagram tříd

Do analytického diagramu tříd jsme zachytili entity, které v naší aplikaci figurují a naznačili jsme vazby mezi nimi. Analytický diagram tříd nám bude sloužit při návrhu modelové vrstvy aplikace. Měl by také sloužit k rychlé orientaci ve vazbách a atributech modelovaných objektů, do kterých ukládáme data. Pro názvy entit, atributů a relací jsme v diagramu použili anglická



Obrázek 3.4: Analytický diagram tříd.

slova.

Analytický diagram tříd je zachycen na obrázku 3.4

3.6 Datová vrstva

Jelikož je naše databáze objektová, nemůžeme v ní vytvořit klasické vazby známé z relačních databází. Tedy vazby, které využívají primární a cizí klíče. Entity naznačené v analytickém diagramu tříd se staly modely a mají podobu javascriptových objektů. Naše modely využívají pro výměnu dat formát s názvem JSON [17]. JSON z velké části nahradil výměnu dat předešlého XML,

především díky své jednoduché struktuře a javascriptovému zápisu. Data v JSON zápisu mají strukturu klíč:hodnota.

Naše modely, na místo cizích a primárních klíčů, využívají atributy k uchovávání si jednoznačného identifikátoru modelu, se kterým mají vazbu. Tento model, jehož identifikátor je uložen v atributu jiného modelu, o vazbě standardně neví. Databáze MongoDB slouží k zachycení surových dat v textových souborech uskupené ve formátu JSON. Umožňuje nám to mít volnou ruku při návrhu struktury modelů databáze. S touto volností zároveň přichází riziko toho, že návrh modelů naší databáze nebude efektivní.

Kapitola 4

Implementace

V této kapitole popisujeme implementaci navržených aplikací. Nejdříve nakonfigurujeme vývojové prostředí a implementujeme prezentační vrstvu webové i mobilní aplikace. Poté následuje propojení naší aplikace s databází. Jednotlivé databázové modely implementujeme dle návrhu datové vrstvy. Následně vytvoříme logiku aplikace. Začneme s implementací aplikačního rozhraní REST. Poté zprovozníme odesílání a příjem dat z prezentační vrstvy do vrstvy datové a naopak. Načtená data uživateli adekvátně zobrazujeme a pro jeho lepší prožitek z používání aplikace implementujeme stavy komponent knihovny React.js. Nastíníme implementaci hlavních funkcionalit webové aplikace, jako je například tvorba testových otázek a odesílání emailů. Následně provedeme privatizaci webové aplikace. Omezíme tedy její používání pouze pro přihlášené uživatele, pro které vzápětí aplikaci personalizujeme. Dále nastíníme implementaci hlavních funkcionalit mobilní aplikace, jako je správa předmětů, testování a vyhodnocování otázek pomocí ostrých a cvičných testů. V neposlední řadě v mobilní aplikaci implementujeme perzistentní úložiště pro uchovávání stažených předmětů a dosažených výsledků uživatele i po ukončení aplikace.

4.1 Konfigurace prostředí pro vývoj webové aplikace

Nainstalovali jsme si program Node.js kvůli možnosti využití správce balíčků npm. Skrze npm jsme nainstalovali náš cílený správce balíčků Yarn.

Pochopitelně byla potřeba nainstalovat i balíčky jako je React.js, Webpack 2, Bootstrap a další potřebné spolu se všemi jejich závislostmi.

Konfigurace vývojového a produkčního prostředí byla provedena v konfiguračních souborech webpack.dev.config.js a webpack.prod.config.js balíčku modulů Webpack. V těchto souborech jsme definovali vstupní cesty souborů určených ke kompilaci, pluginy a loadery, taktéž potřebné ke kompilaci, včetně umístění souborů, které budou výstupem kompilace.

Vytvořili jsme si účet v aplikaci Heroku. Vybrali Node.js kontejner a nainstalovali databázi MongoDB. Pomocí správce balíčku Yarn a balíčků modulů Webpack jsme vyprodukovali build aplikace, který jsme díky systému správy verzí Git nahráli na server Heroku. Git nám tedy zajistil i pravidelné zálohy aktuálního stavu kódu aplikace.

4.2 Konfigurace prostředí pro vývoj mobilní aplikace

Při konfiguraci prostředí pro vývoj mobilní aplikace jsme se plně řídili návodem uvedeným v oficiální dokumentaci React Native [18]. Místo doporučeného npm jsme však využili již zavedený Yarn. Po instalaci a nastavení softwaru Android Studio jsme instalovali emulátor Android aplikací Genymotion [19], který funkce Android studia rozšiřuje. Vyvíjenou aplikaci jsme tedy mohli po každé změně ihned v emulovaném zařízení zobrazit a námi provedené změny otestovat.

4.3 Implementace serveru

Vytvořili jsme server napsaný v Node.js. Server rozlišuje zda-li k němu přistupujeme z vývojového nebo produkčního prostředí. Pokud je do něj přistupováno z vývojového prostředí, pak se zapojí tzv. Webpack Hot Middleware [20]. Tím je zajištěno, aby se po uložení našeho kódu změna projevila ve webovém prohlížeči okamžitě, bez nutnosti obnovování stránky. Hlavní použitý modul serveru je framework Express [21], který práci s Node.js velice usnadňuje.

4.4 Prezentační vrstva

Do defaultní metody render postupně implementujeme HTML kód ve formě zápisu JSX. Ten nám dovoluje psát značkovací jazyk HTML do našeho javascriptovéo kódu, což je opačný přístup než zaujímá většina šablonovacích jazyků.

4.4.1 Prezentační vrstva webové aplikace

Nejdříve bylo vytvořeno základní rozvržení domovské stránky, tedy pozice, velikost a vzhled navigační lišty a umístění obsahu stránky. Vytvořili jsme jednotlivé komponenty odpovídající obrazovkám v návrhu aplikace. Do těchto komponent jsme implementovali tabulku pro zobrazování dat a ovládací prvky sloužící k vyvolání konkrétní akce uživatelem. Komponentu, ve které probíhá tvorba otázek, jsme dekomponovali do menších komponent. Dále jsme vytvořili úvodní stránku s přihlášením do aplikace a domovskou stránku. Ta se bude objevovat bezprostředně po přihlášení do aplikace.

Pro umístění formulářů s uživatelskými vstupy jsme zvolili vyskakovací okna. Ty jsme implementovali pomocí frameworku bootstrap, který tzv. modal okna nabízí ve svých komponentách [22]. Pro dosažení responsivity na požadovaném rozsahu monitorů jsme použili opět framework Bootstrap, konkrétně jeho Grid system [23]. Ten stránku poměrově rozděluje do řádků a dvanácti sloupečků. Na závěr jsme vytvořili vzhled emailové obou emailových šablon.

Prezentační vrstva mobilní aplikace je zobrazena na obrázku 7.4.

Část prezentační vrstvy webové aplikace je znázorněna na obrázku 7.9. Obrazovky prezentační vrstvy mobilní aplikace jsou znázorněny na obrázcích 7.5 až 7.8.

4.4.2 Prezentační vrstva mobilní aplikace

Vytvořili jsme si základní React Native elementy dle prototypu mobilní aplikace. Poté jsme k nim přidávali styly pomocí defaultní komponenty

StyleSheet. Díky této komponentě jsme za pomoci jejího atributu Flexbox provedli základní rozložení elementů na stránkách. Komponenta StyleSheet nám umožnila přiřadit styly elementům obdobným způsobem, jako umožňuje jazyk CSS ve vývoji webových aplikací. V komponentě StyleSheet jsme tedy použili množinu atributů, které lze použít v klasickém CSS. Pomlčková notace atributů používaná v jazyku CSS byla však v komponentě StyleSheet vyměněna za camelCase notaci.

Pro přepínání jednotlivých záložek jsme využili komponentu Scrollable Tab View [24]. Funkcionality posouvání obsahu po stránce při situaci, kdy se nevejde na jednu obrazovku, bylo dosaženo za pomoci React Native komponenty ScrollView.

4.5 Datová vrstva

Tato kapitola popisuje náš postup připojení se k databázi a tvorbu databázových modelů, ve kterých budeme uchovávat data aplikace. Pro usnadnění práce s databází MongoDB jsme využili modul Mongoose pro elegantní modelování MongoDB objektů v Node.js [25]. Mongoose tedy zaobaluje naší jednoduchou databázi do komplexnější vrstvy.

4.5.1 Připojení k databázi

Ve webové aplikaci jsme si vytvořili funkci pro připojení se k databázi. MongoDB poskytuje pro účely připojení se k databázi standardní URI připojení, které má podobu textového řetězce. V textovém řetězci je obsažen název databáze, uživatelské jméno a heslo pro připojení se do databáze. Naprosto totožné funkce jsme implementovali i v aplikaci mobilní. Tím se začaly projevovat výhody izomorfní aplikace a správnost výběru použitých technologií.

4.5.2 Tvorba modelů

Jelikož je databáze MongoDB ze své podstaty pouze úložiště textových dat, nerozlišuje datové typy. Abychom naši databázi rozšířili o možnost datových typů a o možnost kontroly unikátnosti záznamu, využili jsme schéma

poskytované modulem Mongoose. Na serveru, na kterém je umístěna webová aplikace, jsme si vytvořili složku models, do které jsme postupně přidávali jednotlivé modely. Každý model byl reprezentován jedním javascriptovým souborem, ve kterém se definuje pomocí Mongoose schéma daného modelu. Dále jsme v modelech na základě požadavků obou aplikací implementovali jednotlivé funkce provádějící operace s daty a operace s databází MongoDB.

4.6 Vrstva aplikační logiky

Vrstva aplikační logiky má na starosti propojení prezentační vrstvy s vrstvou datovou. Je zde implementována veškerá logika aplikace. V následující kapitole popisujeme postup implementace aplikačního rozhraní REST. Je také popsána tvorba logiky, která má na starosti operaci s daty. Implementací React stavů jsme uživateli zpříjemnili práci s daty a celkový dojem z používání aplikace. Následně byla provedena implementace specifická pro mobilní a webovou aplikaci.

4.6.1 Aplikační rozhraní REST

V souboru obsahující náš server jsme implementovali koncové body URL. Tyto body zachycují požadavky typu POST a požadavky typu GET na jednotlivých URL adresách, které jsme si stanovili. Koncové body přijímající požadavky typu POST, mají URL adresu ve formátu api/topic/add, kde topic je konkrétní název podstránky a add je konkrétní operace, kterou chceme provést. U požadavků typu GET je URL adresa definována standardně ve formátu /api/topics, kde topics jsou data modelu, které chceme získat.

Následně v těchto koncových bodech voláme exportované metody našich modelů a posíláme jim jako parametry data, která jsme přijali od našich aplikací. Všechny chybové hlášení i hlášení o úspěšně provedených operacích, které vzniknou v modelech, necháváme probublat koncovými body aplikačního rozhraní REST. Hlášení jsou následně aplikačním rozhraním zpracována a aplikacím je zpětně zaslána informace o úspěšnosti požadovaných operací.

4.6.2 Ukládání dat do databáze

Pro ukládání dat do databáze jsme využili technologie AJAX, což je technologie pro zasílání asynchronního požadavku na server. Zjednodušený zápis metody nám umožnila knihovna jQuery. Formuláři, zobrazenému ve vyskakovacím okně, jsme nejdříve zabránili jeho defaultnímu synchronnímu odeslání. Následně jsme na formulář navěsili funkci asynchronního odeslání požadavku na server a uložili si obsah formuláře pomocí jQuery DOM selektorů. V asynchronním požadavku jsme definovali jednotlivé koncové body totožné s koncovými body na našem serveru. Dále jsme definovali data uložená z formulářových polí a požadavek odeslali na již implementované aplikační rozhraní REST.

4.6.3 Zobrazování dat z databáze

AJAX umožňuje odesílání i přijímaní požadavků podle deklarace typu POST respektive GET. Při implementaci byla ovšem zjištěna i alternativa funkce k funkci AJAX, a to novější funkce Fetch. Ta se řídí podobnými principy. Její zápis je však jednodušší. Pro načítání dat, které poskytuje aplikační rozhraní REST, byla tato funkce Fetch použita. Ta nám zprostředkovala aktuální data poskytovaná rozhraním. Tyto data jsme si v naší aplikaci uložili do lokálních proměnných (později do stavů komponent) a na patřičných místech jsme je v aplikaci vykreslili.

4.6.4 Zobrazování informačních hlášení

Při ukládání, modifikaci a případném žádaní o konkrétní data ze serveru, nám aplikační rozhraní vrací informace o výsledků námi žádaných operací. Tento fakt jsme použili pro implementaci hlášení, které uživatele informují o výsledku požadovaných operací. Při implementace informačních hlášení byly použity vyskakující upozornění frameworku Bootstrap [26].

4.6.5 Implementace React stavů

Pro změnu stavu komponenty aplikace React používá tzv. State Lifecycle [27]. Díky tomu máme k dispozici objekt stavů aplikace stejně jako definované

funkce zpětného volání životního cyklu komponenty. Ve funkci zpětného volání componentDidMount, která se automaticky zavolá po vykreslení komponenty načteme aktuální data ze serveru funkcí Fetch. Získaná data si uložíme do objektu stavu komponent aplikace. Následně data v komponentě vykreslujeme v metodě defaultní metodě render, která slouží pro vykreslení HTML obsahu komponenty z vyhodnoceného javascriptového kódu napsaném ve formátu JSX.

Stavy aplikace mají jednu dobrou vlastnost. Pokud jsou jejich hodnoty logickou vrstvou změněny, pak se změny projeví i v již načtené stránce, a to automatickým překreslením pouze těch HTML značek, které změněné hodnoty obalují. Tuto schopnost jsme napříč webovou i mobilní aplikací hojně využili. Všichni učitelé, předměty a otázky tak v adekvátní tabulce přibudou bezprostředně po jejich vytvoření. Obdobně je tomu u i jejich smazání.

4.6.6 Logická vrstva webové aplikace

V této podkapitole jsme implementovali možnost přepínání jednotlivých podstránek webové aplikace v reakci na uživatelem přepínané záložky v navigačním menu. Dále byla na základě stanovených požadavků implementována logika všech podstránek. Provedli jsme privatizaci aplikace a omezili tak použití aplikace pouze na přihlášené uživatele. Prostředí aplikace jsme přihlášeným uživatelům dostatečně personalizovali. Na závěr implementace logiky webové aplikace jsme vytvořili systém pro odesílání elektronické pošty.

Přepínaní podstránek

Abychom docílili jednostránkové aplikace, využili jsme modul React Router [28], který nám umožňuje vykreslovat komponenty na základě změny URL adresy. Kliknutím na odkaz na stránce změníme URL v prohlížeči. Na tuto akci zareaguje modul React Router a vykreslí adekvátní komponentu. Tím je zajištěn efekt přepínání stránek.

Tvorba otázek

Jednou z hlavních funkcionalit webové aplikace je tvorba otázek. Na místo pouze jednoho vyskakovacího okna typu modal je vytvoření otázky rozděleno podle jejího typu do dvou samostatných vyskakovacích oken. Jako základ je v modal oknu k vyplnění pouze znění otázky a počet správných odpovědí, který se vybere ze SelectBoxu. Na změnu hodnoty SelectBoxu je navěšena metoda, která vykreslí adekvátní počet komponentů odpovědí. Která konkrétní komponenta odpovědi se v otázce vysvětlí, závisí na typu otázky. Pro otázku s právě jednou správnou odpovědí se vykreslí komponenta odpovědi obsahující tlačítko typu Radio. Pro otázku s více správnými odpovědmi je vykreslena komponenta odpovědi s tlačítkem typu checkBox.

Následně jsme vytvořili funkci, která dokáže měnit typ otázky z jednoho typu na druhý a obráceně. To při zachování již vyplněných dat, které se v průběhu změny typu otázky mohou dále modifikovat. Realizaci metody jsme provedli přenesením dat ze zdrojového modal okna do cílového modal okna.

Autentizace a autorizace webové aplikace

Modifikujeme kořenovou komponentu aplikace tak, aby nám informace o stavu přihlášení uživatele a o jeho totožnosti probublávala do jednotlivých komponent aplikace. V aplikaci jsme vytvořili v koncových komponentách nový stav, do kterého je probublaná informace o přihlášení uložena. Na základě tohoto stavu zobrazíme buďto přihlašovací obrazovku, nebo celou aplikaci. Formulář umístěný na přihlašovací obrazovce napojíme na aplikační rozhraní a na serverové straně ověřujeme, zda uživatel s takovým uživatelským jménem a heslem existuje. V opačném případě zobrazíme uživateli informační hlášení o neplatnosti přihlašovacích údajů. Následně jsme na serveru privatizovali všechny koncové body aplikačního rozhraní, které využívá webová aplikace. Využití koncových bodů je podmíněno úspěšným přihlášením uživatele. Informaci o tom, že je uživatel přihlášen, uchováváme nejen na klientské straně, ale i na straně serveru. Využíváme k tomu komponentu Express Session [29]. Koncové body mobilní aplikace necháváme veřejné, jelikož k využívání mobilní aplikace nepotřebuje být uživatel přihlášen.

Personalizace uživatele webové aplikace

Na základě schopnosti rozpoznat přihlášeného uživatele v jednotlivých komponentách jsme již mohli aplikaci personalizovat. V renderovacích metodách jednotlivých komponent jsou vytvořeny podmínky, které při cyklu vykreslování jednotlivých dat porovnávají, zda nemá aktuálně přihlášený uživatel s právě vykreslujícími se daty nějakou spojitost. Pokud ano, přidáme HTML elementům obalujícím tyto data speciální třídu, která patřičně modifikuje vzhled elementu. Při tvorbě prezentační vrstvy jsme tuto třídu definovali v souborech obsahujících styly aplikace.

Implementace mailového systému

Založili jsme si nový soubor obsahující komponentu pro odesílání emailů. Vytvořili jsme funkci, která odesílá email po vytvoření registrace a funkci, která odesílá email pro obnovu hesla. K implementaci obou funkcí jsme využili modulu Nodemailer [30]. Do atributu pro HTML podobu emailu jsme vložili šablony emailů vytvořené při implementaci prezentační vrstvy. Předtím jsme však šablony pomocí zápisu JSX modifikovali. Vložili jsme do HTML šablony patřičné javascriptové proměnné. Pomocí těchto proměnných vykreslujeme údaje konkrétních uživatelů, kterým je email zasílán.

4.6.7 Logická vrstva mobilní aplikace

Implementace specifická pro mobilní aplikaci zahrnovala tvorbu cvičného a ostrého testu, jejich vyhodnocení a ukládání uživatelských dat do lokálního úložiště zařízení.

Správa témat

Při implementaci prezentační vrstvy mobilní aplikace bylo vytvořeno textové pole spolu s odesílacím tlačítkem, na které navěsíme metodu provádějící stažení předmětu. Tato metoda se pomocí funkce Fetch dotáže serveru na existenci tématu se zadaným kódem. Pokud takový kód server v databázi témat nenajde, odpoví chybou, kterou aplikace zpracuje a uživateli zobrazí

adekvátní chybové hlášení. Pokud je však téma nalezeno, aplikační rozhraní serveru zažádá model o všechny otázky patřící pod stejný kód předmětu, který modelu zašle v parametru metody. Model aplikačnímu rozhraní vrátí pole objektů otázek. Aplikační rozhraní toto pole vloží do objektu tématu a zašle modifikovaný objekt tématu zpět do aplikace. Přijaté téma je následně uloženo do stavu komponenty aplikace a zobrazeno uživateli v seznamu témat, odkud lze po stisknutí ikony popelnice téma ze stavu komponenty aplikace odebrat. Tím je dosaženo automatického překreslení aplikace, respektive odstranění elementu tématu ze zobrazené struktury aplikace.

Ostrý test

Na záložce s ostrým testem si lze zvolit téma, ze kterého chce být uživatel testován. Po kliknutí na výběr konkrétního tématu je objekt tématu načten ze stavu komponenty aplikace. Dále jsou inicializovány pomocné proměnné uchovávající stav testu. Pole otázek obsažené v objektu tématu následně iterujeme a pro každou iteraci zobrazíme znění otázky spolu s adekvátními odpovědmi. Uživatel se odpovědí na otázku přesune do další iterace pole otázek. Podoba odpovědí závisí na typu aktuální otázky. Pokud na otázku existuje právě jedna odpověď, pak jsou jednotlivé odpovědi zobrazeny tmavě fialovou barvou a kliknutím na jakýkoliv z nich je otázka považována za zodpovězenou. Pokud na otázku existuje 0 až n správných odpovědí, pak jsou elementy podbarveny světle fialovou a pod posledním elementem se zobrazuje tlačítko pro potvrzení svých odpovědí. Kliknutím na tento typ odpovědí se otázka nevyhodnotí jako zodpovězená, ale zavolá se funkce, která odpovědi přidá či odebere ikonu zaškrtnutí. Následně lze své odpovědi na otázku potvrdit prostřednictvím tlačítka umístěného pod poslední odpovědí. Pokud probíhá poslední iterace otázky, pak je po jejím zodpovězení test považován za uzavřený, zresetují se pomocné proměnné testu, a jsou zobrazeny výsledky testu.

Cviční test

Cvičný test je ve skutečnosti implementován jako rozšíření testu ostrého. Z tohoto důvodu v aplikaci nelze provádět oba typy testů najednou. Kdybychom uživateli tuto možnost nezakázali v aplikaci, pak by si testy navzájem přepisovali pomocné proměnné. Rozšíření spočívá v implementaci mechanizmu, který uživateli graficky zvýrazní správnost odpovědi v momentě, kdy by za normálních okolností mělo dojít k iteraci na další otázku.

U otázky typu 0 až n jsme ihned po potvrzení odpovědi na otázku všechny odpovědi otázky graficky zvýraznili. Podmínky v komponentě, která odpovědi vykresluje, byly následující. Nezaškrtnutým odpovědím, které by měly být zaškrtnuty a zaškrtnutým odpovědím, které neměly být zaškrtnuty přiřadíme třídu incorrect. Ta zařídí, že pozadí odpovědi bude zvýrazněno červeně. Obdobný postup byl použit pro vykreslení správných odpovědí.

U otázky s právě jednou správnou odpovědí jsme ihned po výběru některé z odpovědí graficky zvýraznili její správnost či nesprávnost. Uživatel má tedy možnost zkoušet další odpovědi, dokud nenarazí na odpověď správnou. Až po výběru správné odpovědi je uživatel přesměrován na další otázku, a to po časovém limitu 600 milisekund. Bez vytvoření časového limitu by uživatel zvýraznění správné odpovědi neměl čas postřehnout. Aby nedošlo ke zkreslení výsledků testu je po prvním pokusu o zodpovězení otázky vyhodnocena správnost odpovědi na danou otázku a celé hodnocení odpovědí je uzamčeno. Tím pádem další pokusy uživatele už na vyhodnocení testu nemají žádný vliv. Pokud se tedy stane, že uživatel vybere hned po správné odpovědi nedopatřením odpověď chybnou, pak tato odpověď již není zohledněna.

Výsledky testů

Po zodpovězení poslední otázky je z pomocných proměnných testu inicializovaný objekt výsledku testu, který je naplněn potřebnými daty. Do objektu výsledku testu je uložen název testovaného tématu, počet správně zodpovězených otázek, celkový počet otázek tématu a procentuální úspěšnost testu.

Objekt testu je uložen do stavu komponenty aplikace a vykreslen do šablony, která uživateli výsledek testu prezentuje. Pokud se jednalo o dokončení ostrého testu, pak je tento objekt odeslán na aplikační rozhraní serveru. Ze serveru je výsledek ostrého testu načítán do prezentační vrstvy webové aplikace.

Perzistentní lokální úložiště

Aby uživatelé při zavření aplikace neztratili přidaná témata a dosažené výsledky, implementujeme pro tyto účely perzistentní lokální úložiště AsyncStorage [31]. Pomocí jednoduchých get a set metod do lokální databáze ukládáme objekty tématu a objekty výsledku. Jelikož tato databáze umí ukládat pouze textové řetězce, ale ne javascriptové objekty, tak si před uložením do databáze převedeme jednotlivé objekty do podoby řetězců. To nám umožní metoda

JSON.stringify. Při získávání dat z lokální databáze aplikujeme obrácený postup. Jakmile uložíme témata, respektive výsledky do stavů komponent naší aplikace, provedeme i vložení těchto objektů do lokální databáze. Při spuštění aplikace se nejprve dotážeme lokální databáze na existenci témat a výsledků, a případné získané data uložíme opět do stavu komponenty aplikace.

Kapitola 5

Testování

Průběžné testování syntaxe zdrojového kódu bylo dosaženo za pomoci nástrojů, které byly součástí našeho vývojového prostředí mobilní a webové aplikace. Kompletní kontrolu syntaxe kódu webové aplikace zajišťoval Yarn v kombinaci se sadou balíčků Webpack. Byly nám poskytnuty přehledné výpisy jak do konzole bash, tak do samotného okna webové aplikace. Správnou syntaxi kódu mobilní aplikace nám zajišťovali tzv. JavaScript Syntax Transformers, které využívají kompilátor Babel a jsou součástí defaultního vývojového prostředí pro vývoj v React Native [32].

Při vývoji webové aplikace byla její korektní funkčnost testována hned několika způsoby. V první řadě bylo testování funkcionality prováděno za pomocí průběžného výpisu stavu proměnných a výsledků operací do javascriptové konzole prohlížeče Google Chrome. V konzoli prohlížeče jsme mohli vidět i případné chybové hlášení a varování nespadající pod kontrolu syntaxe, kterou prováděl Yarn spolu se sadou balíčků Webpack. Správnou funkcionalitu aplikace jsme na našem lokálním serveru testovali přímo ve webovém prohlížeči. K rychlému testování přidané funkcionality nám pomáhala funkce Hot Reloading. Průběžně jsme také nahrávali build aplikace na server Heroku. Tak jsme měli možnost aplikaci otestovat v produkčním prostředí spolu s možností kontroly automaticky vytvářených logů serveru.

Funkcionalitu mobilní aplikace jsme testovali z pohledu uživatele v simulátoru Genymotion, který nám dovoluje zapnout vzdálené ladění aplikace v nástrojích pro vývojáře prohlížeče Google Chrome. Díky tomuto faktu jsme mohli aplikovat stejné postupy, jako při testování webové aplikace. Průběžně jsme vytvářeli build aplikace, který jsme nahráli a otestovali na reálných

zařízeních s operačním systémem Android.

Následně jsme monitorovali provoz sítě v záložce Network, umístěné také v nástrojích pro vývojáře. Díky možnosti monitorování sítě, jsme tak mohli vidět stav jednotlivých asynchronních požadavků, které odesíláme na server. Mohli jsme i například vidět, jaké konkrétní data či chybové hlášky nám posílá server zpět do aplikace.

Při testování jsme zjistili, že zasílání požadavků na server z přihlašovací stránky webové aplikace se provádí až příliš rychle. Tohoto faktu by někdo mohl zneužít a zkoušet různé kombinace jmen a hesel. Vytvořili jsme tedy alespoň základní ochranu, která po deseti neúspěšných pokusech o přihlášení do aplikace další možnost odeslání formuláře zablokuje.

V závěru testování jsme pomocí záložky Network nasimulovali pomalé internetové připojení. Na základě tohoto testování jsme do webové i mobilní aplikace dodatečně implementovali grafickou zpětnou vazbu indikující načítání dat z externí databáze.

Kapitola 6

Závěr a možné pokračování na práci

Hlavním cílem této bakalářské práce bylo vytvoření webové aplikace, která by umožňovala rychlé vytváření a správu otázek, předmětů a učitelů. Dále bylo cílem bakalářské práce vytvořit mobilní aplikaci pro testování uživatelů. Ta by pracovala s vytvořenými otázkami aplikace webové a umožňovala by uživatelům provádět cvičné a ostré testy.

Nejprve byla stanovena vize projektu a provedena analýza konkurenčních aplikací. Na základě vyhodnocení analýzy konkurence jsme stanovili funkční a kvalitativní požadavky webové a mobilní aplikace. Určili jsme jednotlivé uživatelské role a jejich možnosti byly modelovány do diagramu případu užití. Na základě stanovených požadavků jsme analyzovali platformy a vývojové nástroje, které budou pro vývoj aplikací zapotřebí.

Navrhli jsme strukturu a vytvořili prototypy obou aplikací, které byly použity pro účely testování s uživateli. Testování poskytlo první cennou zpětnou vazbu o tom, jakým směrem se má návrh aplikace dále ubírat. S ohledem na vybrané technologie byla navržena architektura aplikace a analytický diagram tříd, který následně sloužil jako předloha pro navržení datové vrstvy našich aplikací.

Podle našeho návrhu aplikace jsme postupně implementovali modelované požadavky obou aplikací. Po úspěšné implementaci většiny požadavků jsme aplikaci podrobovali testům a dodatečně implementovali některé funkce, které opravily objevené nedostatky aplikace.

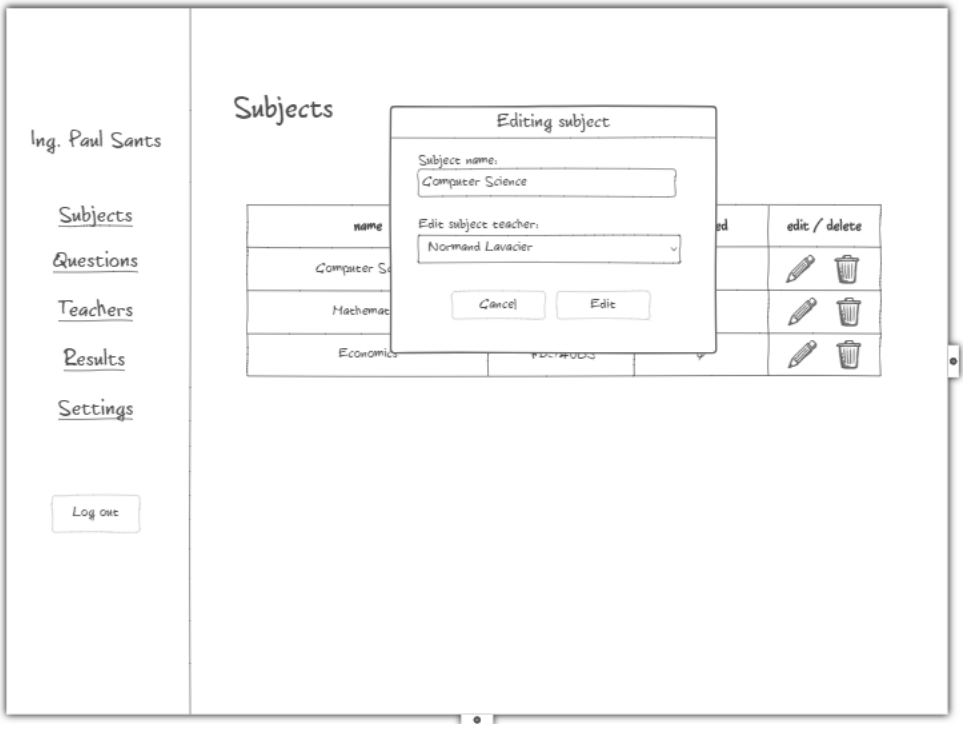
Kvůli náročným požadavkům, které vyústily v obtížnou a zdlouhavou implementaci obou aplikací se v mobilní aplikaci nestihlo implementovat zpracování chyb při pokusu o stažení předmětu bez internetového připojení. Následně se nestihla implementovat jedna minoritní funkcionalita - evidence počtu stažení mobilní aplikace, kde by se na aplikační rozhraní zaslal údaj o stažení dané aplikace a příslušná metoda modelu by zařídila onu inkrementaci atributu stažení aplikace.

Při pokračování na práci by byly tyto drobné nedostatky jednoduše eliminovány. Následně by se mohly obě aplikace rozšířit o možnost dalšího typu otázky, která by využívala při testování uživatelů i obrázková data. Další praktické rozšíření, by mohla být implementace modelu školy a příslušných metod, které by dovolovaly použití webové aplikace několika školám najednou při zachování oddělení jejich dat. Aplikace by mohla být následně nabízena některým školám za účelem získaní zpětné vazby z produkčního prostředí, které by vedly od vylepšení aplikace k její distribuci na trh.

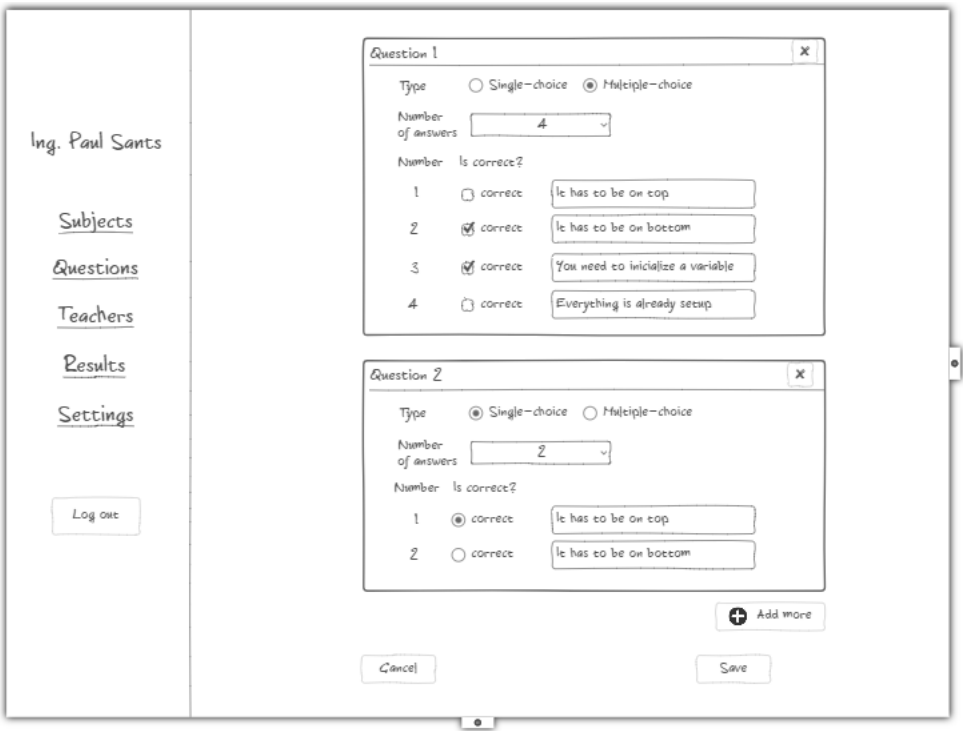
Kapitola 7

Přiložené obrázky

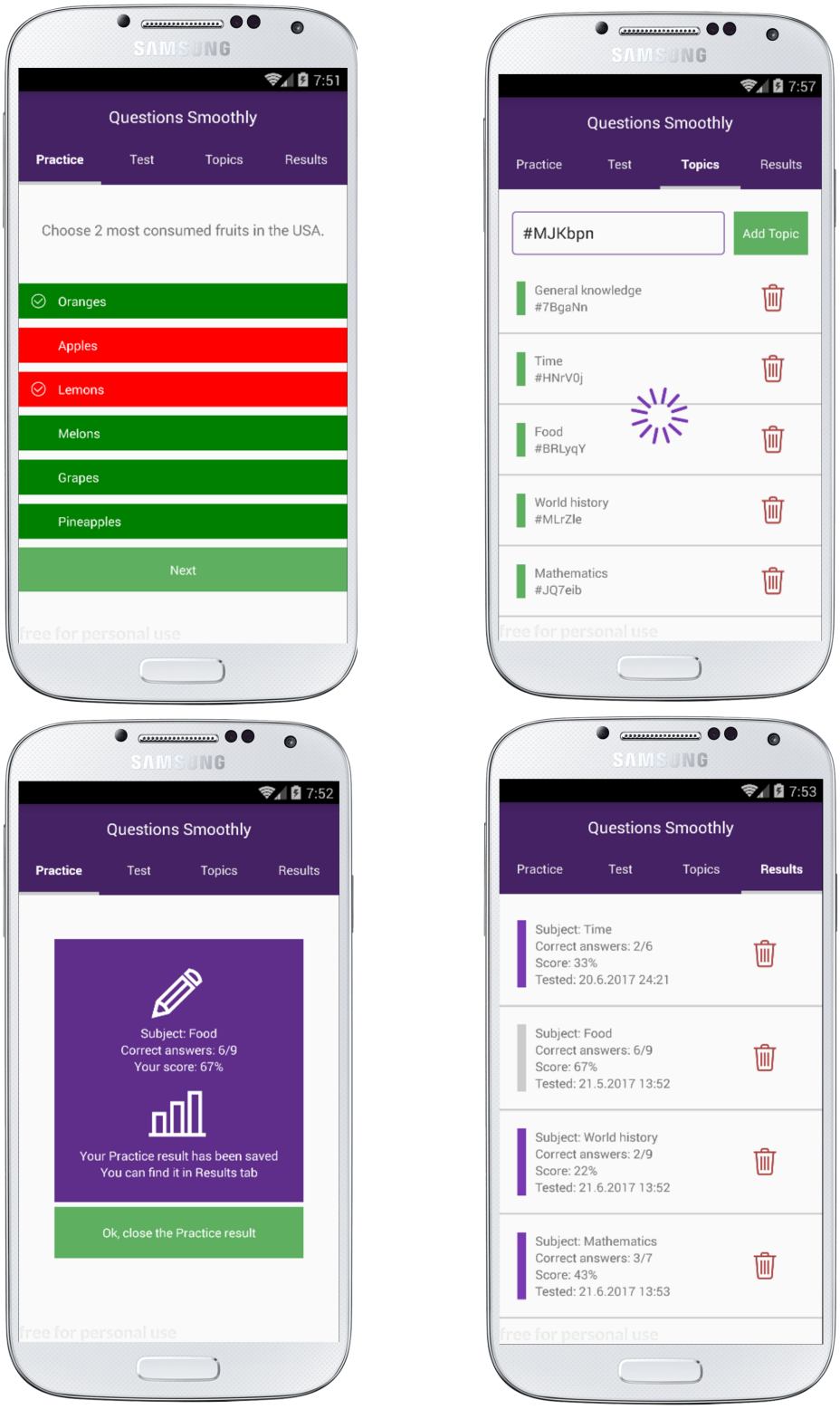
Obrázek 7.1: Prototyp mobilní aplikace



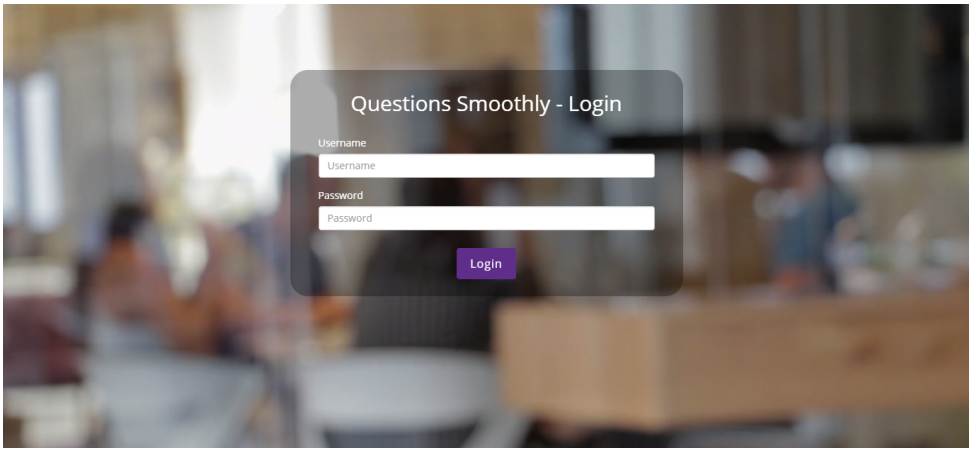
Obrázek 7.2: Prototyp webové aplikace - tvorba předmětů



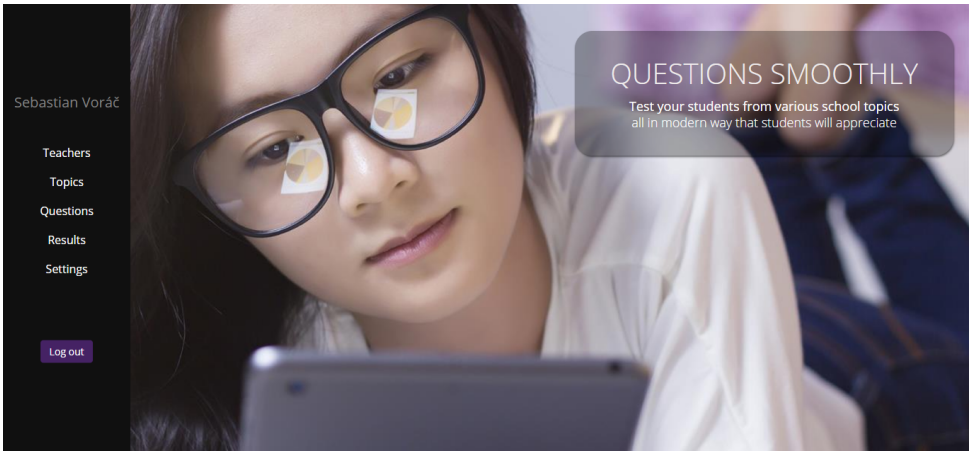
Obrázek 7.3: Prototyp webové aplikace - tvorba otázek.



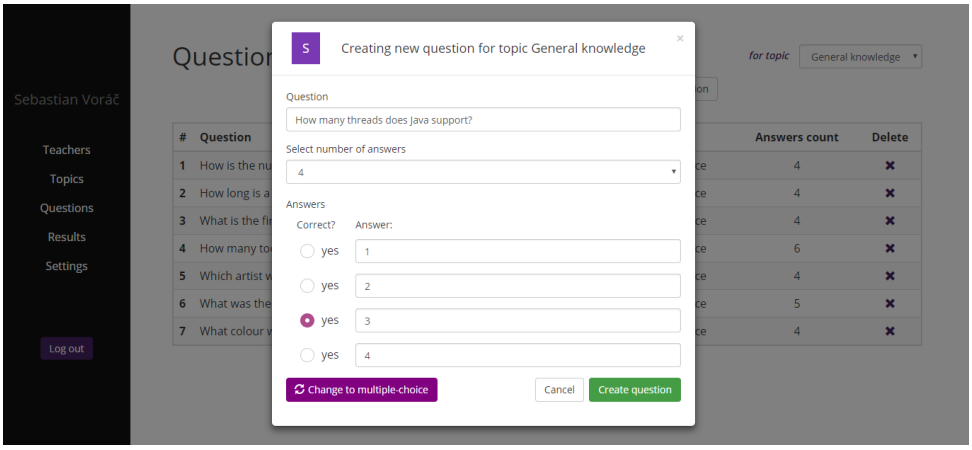
Obrázek 7.4: Prezentační vrstva mobilní aplikace.



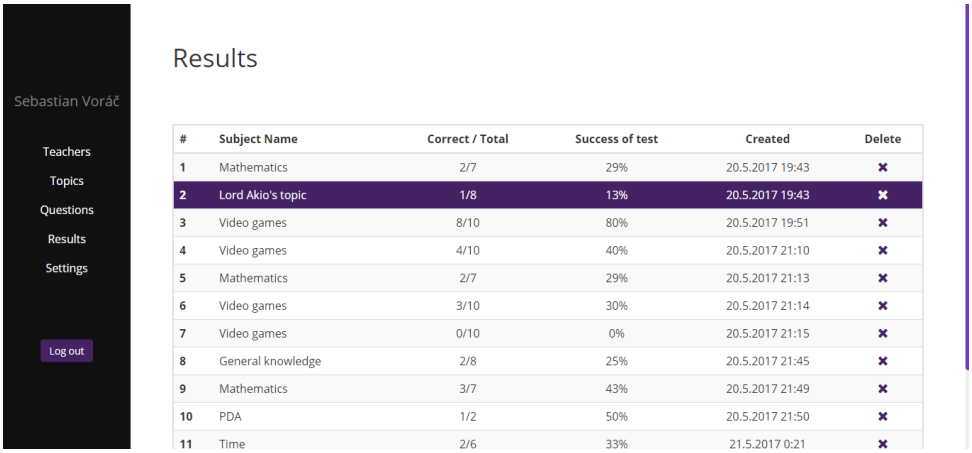
Obrázek 7.5: Přihlašovací obrazovka webové aplikace.



Obrázek 7.6: Domovská stránka webové aplikace.



Obrázek 7.7: Stránka pro přidávání otázek.



Obrázek 7.8: Stránka zobrazující výsledky testování z mobilní aplikace.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Voráč** Jméno: **Sebastian** Osobní číslo: **420366**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Softwarové technologie a management**

Studijní obor: **Web a multimedia**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mobilní a webová aplikace pro testování znalostí uživatelů

Název bakalářské práce anglicky:

Mobile and web applications for knowledge examination

Pokyny pro vypracování:

Analyzujte trh s mobilními aplikacemi, které se zabývají testováním znalostí uživatelů pomocí otázek formou cvičného, respektive ostrého testu. Po konzultaci s vedoucím práce vyberte jednu až dvě cílové skupiny a zaměřte se na její požadavky. Na základě analýzy navrhněte a implementujte mobilní klientskou aplikaci pro vyplňování testů a webovou administrační aplikaci pro správu a vyhodnocování testů. Mobilní aplikaci implementujte s použitím knihovny React Native a pro webovou aplikaci v HTML a Javascriptu. Pro serverovou část použijte vhodné technologie. Aplikaci průběžně testujte jak pomocí uživatelských testů tak technikami testování software.

Seznam doporučené literatury:

B. Eisenman, Learning React Native, O'Reilly Media, 2015
B. Fling, Mobile Design and Development, O'Reilly Media, 2009
T. Lowdermilk, User-Centered Design, O'Reilly Media, 2013

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ivo Malý Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **03.03.2017** Termín odevzdání bakalářské práce: **26.05.2017**

Platnost zadání bakalářské práce: _____

Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

Literatura

[1] Getting Started – *electronic information.*
URL: $< https://facebook.github.io/react-native/docs/getting-started.html >$
[cit. 2017-05-02].

[2] Design | Android Developers – *electronic information.*
URL: $< https://developer.android.com/design/index.html >$ [cit. 2017-05-03].

[3] Paletton - The Color Scheme Designer – *electronic information.*
URL: $< http://paletton.com >$ [cit. 2017-05-03].

[4] W3C HTML – *electronic information.*
URL: $< https://www.w3.org/html >$ [cit. 2017-05-06].

[5] CSS Introduction – *electronic information.*
URL: $< https://www.w3schools.com/css/css_intro.asp >$ [cit. 2017-05-06].

[6] Ondřej Žára, JavaScript: Programátorské techniky a webové technologie, Kapitola 1, 2015
ISBN: 978–80–251–4573–9, 184 stran

[7] What is the Document Object Model? – *electronic information.*
URL: $< https://www.w3.org/TR/WD-DOM/introduction.html >$
[cit. 2017-05-07].

[8] Node.js – *electronic information.*
URL: $< https://nodejs.org/en >$ [cit. 2017-05-07].

[9] Why you should not use AngularJs – Medium – *electronic information.* URL: < *https* : *//medium.com/@mnemon1ck/why − you − should − not− use − angularjs − 1df5ddf6fc99#.77mbyy9bn* > [cit. 2017-05-07].

[10] 1. Modern web applications: an overview – *electronic information.* URL: < *http* : *//singlepageappbook.com/goal.html* > [cit. 2017-05-08].

[11] Sass: Syntactically Awesome Style Sheets – *electronic information.* URL: < *http* : *//sass − lang.com* > [cit. 2017-05-11].

[12] Ondřej Žára, JavaScript: Programátorské techniky a webové technologie, Kapitola 1, 2015
ISBN: 978–80–251–4573–9, 184 stran

[13] draw.io – *electronic information.* URL: < *https* : *//www.draw.io* > [cit. 2017-05-11].

[14] NinjaMock – *electronic information.* URL: < *https* : *//ninjamock.com* > [cit. 2017-05-13].

[15] What is Wizard of Oz prototyping? - Definition from WhatIs.com – *electronic information.* URL: < *http* : *//searchcio.techtarget.com/definition/Wizard − of − Oz − prototyping* > [cit. 2017-05-13].

[16] Do you know what a REST API is? — SitePoint – *electronic information.* URL: < *https* : *//www.sitepoint.com/developers − rest − api* > [cit. 2017-05-13].

[17] JSON Introduction – *electronic information.* URL: < *https* : *//www.w3schools.com/js/js*$_j$*son*$_i$*ntro.asp* > [cit. 2017-05-14].

[18] Getting Started – *electronic information.* URL: < *https* : *//facebook.github.io/react − native/docs/getting − started.html* > [cit. 2017-05-14].

[19] Genymotion Android Emulator – Fast • Easy • Anywhere – *electronic information.* URL: < *https* : *//www.genymotion.com* > [cit. 2017-05-14].

[20] Introducing Hot Reloading – *electronic information.* URL: < *https* : *//facebook.github.io/react − native/blog/2016/03/24/introducing − hot − reloading.html* > [cit. 2017-05-15].

[21] Express - Node.js web application framework – *electronic information.* URL: < *https* : *//expressjs.com* > [cit. 2017-05-17].

[22] Modal · Bootstrap – *electronic information.* URL: < *https* : *//v4 − alpha.getbootstrap.com/components/modal/#content* > [cit. 2017-05-18].

[23] CSS · Bootstrap – *electronic information.*
URL: < *http* : *//getbootstrap.com/css/#grid* > [cit. 2017-05-18].

[24] GitHub - skv-headless/react-native-scrollable-tab-view: Tabbed navigation that you can swipe between, each tab can have its own ScrollView and maintain its own scroll position between swipes. Pleasantly animated. Customizable tab bar – *electronic information.*
URL: < *https* : *//github.com/skv − headless/react − native − scrollable − tab − view* >
[cit. 2017-05-19].

[25] Mongoose ODM v4.10.2 – *electronic information.*
URL: < *http* : *//mongoosejs.com* > [cit. 2017-05-20].

[26] Bootstrap Alerts – *electronic information.*
URL: < *https* : *//www.w3schools.com/bootstrap/bootstrap*$_a$*lerts.asp* >
[cit. 2017-05-20].

[27] State and Lifecycle - *React* – *electronic information.*
URL: < *https* : *//facebook.github.io/react/docs/state − and − lifecycle.html* >
[cit. 2017-05-21].

[28] React Router: *Declarative Routing for React.js* – *electronic information.*
URL: < *https* : *//reacttraining.com/react − router* > [cit. 2017-05-21].

[29] express-session – *electronic information.*
URL: < *https* : *//www.npmjs.com/package/express − session* > [cit. 2017-05-21].

[30] Nodemailer – *electronic information.*
URL: < *https* : *//nodemailer.com/about* > [cit. 2017-05-22].

[31] AsyncStorage – *electronic information.*
URL: < *https* : *//facebook.github.io/react − native/docs/asyncstorage.html* >
[cit. 2017-05-23].

[32] JavaScript Environment – *electronic information.*
URL: < *https* : *//facebook.github.io/react − native/docs/javascript − environment.html* >
[cit. 2017-05-23].