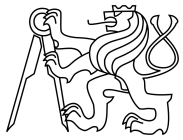
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Webová aplikace pro tvorbu a vyplňování testů
Student:	Bc. Karolína Běhalová
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat webovou aplikaci, která bude součástí vzdělávacího projektu Selftester. Aplikace bude primárně sloužit k vytváření nových testů obsahujících minimálně jednoduchý textový/číselný vstup, dále single-choice a multiple-choice otázky. Jádrem projektu bude dostupné API rozhraní, se kterým bude webová aplikace komunikovat. Práce si dále klade za cíl porovnat různé přístupy a technologie, používané v současnosti ke tvorbě webových služeb a také realizaci kvalitního uživatelského rozhraní.

Postupujte dle následujících kroků:
Proveďte analýzu jiných webových aplikací, které umožňují tvorbu a vyplňování testů.
Na základě této analýzy navrhněte pomocí metodik softwarového inženýrství a implementujte funkční aplikaci a uživatelské rozhraní.
Aplikaci podrobte vhodným testům uživatelského rozhraní, dále aplikaci při vývoji řádně testujte.
Na základě provedených testů aplikaci vhodně upravte.
Proveďte hodnocení aplikace a navrhněte další rozvoj.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 16. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Webová aplikace pro tvorbu a vyplňování testů

Bc. Karolína Běhalová

Vedoucí práce: Ing. Jiří Huňka

4. května 2017

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. § 46 odst. 6 tohoto zákona 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 4. května 2017 .....

Odkaz na tuto práci

Běhalová, Karolína. *Webová aplikace pro tvorbu a vyplňování testů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cílem práce je vytvořit webovou aplikaci, která bude součástí rozsáhlejšího vzdělávacího projektu Selftester. Práce popisuje celý proces od analýzy až po testování a nasazení do zkušebního provozu. Je kladen důraz na vývoj kvalitního uživatelského rozhraní, které je průběžně testováno. Součástí práce je dále zcela oddělené API, které obsluhuje jak webovou, tak mobilní aplikaci Selftester, jenž v rámci své diplomové práce vytváří Bc. Jakub Kalina.

Práce také pojednává o současných možnostech a přístupech na poli vývoje webových aplikací.

Klíčová slova vzdělávací aplikace, uživatelské rozhraní, backend, frontend, Node.js

Abstract

The main objective of the thesis is building a web application that is a part of educational project called Selftester. The thesis documents the whole process, from an analysis, throught design and implementation stages, to usability testing and deployment. The thesis emphasizes the importance of a proper user interface, its design and development as well as its continuous testing. The essential part of the application is a standalone API. The API provides

data for the web application and a mobile application. The mobile application is also a part of the Selftester project and it is being developed by Bc. Jakub Kalina as the main objective of his diploma thesis.

The thesis also describes and compares current options and basic principles of web application development in general.

Keywords educational application, user interface, backend, frontend, Node.js

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Úvod

Cílem této práce je navrhnout a vytvořit výukovou aplikaci, jenž bude sloužit k upevňování znalostí. Proces učení a jeho efektivita byly již předmětem mnoha výzkumů a bylo publikováno množství rad a doporučených postupů. S rozvojem především mobilních aplikací možností dále přibylo.

Projekt Selftester má fungovat jako webová a mobilní platforma k opakování učiva pomocí vyplňování testů, vygenerovaných z předem daných otázek. Veškerý obsah tedy záleží na uživateli, ten se může nacházet na straně učitele (v případě, že je tím, kdo test vytváří) a i studenta. Aplikace si klade za cíl poskytovat pouze tuto jednoduchou službu - umožnit komukoliv, aby si mohl vytvořit bodovaný, automaticky vyhodnocovaný test (všechny funkční požadavky jsou rozepsané v kapitole Analýza 1), ten posléze opakovaně vyplňovat, zkoumat výsledky. Tímto uživateli poskytneme nástroj poučit se z vlastních chyb a tak se zlepšit a doplnit si znalosti. Uživatel nemá být omezen tématem nebo obsahem testu, platí pouze to, že každou otázku musí být možné automaticky vyhodnotit. Předmětem otázek tak může být prakticky cokoliv, například příprava na zkoušky z autoškoly nebo populárně naučný test o Harrym Potterovi. Důraz je kladen zejména na flexibilitu testů a kvalitně navržené uživatelské rozhraní.

V rámci této diplomové práce vznikne API a webová aplikace projektu Selftester. Webová aplikace bude sloužit primárně k vytváření, editaci a správě testů. API bude obsluhovat tuto aplikaci a mobilního klienta pro platformu Android. Mobilní aplikaci v rámci své diplomové práce vytváří Bc. Jakub Kalina.

Kapitola 1

Analýza

Webové aplikace, jež uživateli nabízí tvorbu vlastních formulářů, nejsou v dnešní době nic nového ani neobvyklého. Formuláře samy o sobě jsou velmi důležitým stavebním kamenem všech dynamických webových stránek a mají širokou škálu využití, primárně jsou ale určeny ke sběru dat zadaných uživatelem. Mezi formuláři, které mají strukturu otázka - odpověď, můžeme rozlišit tyto základní typy (uvažujeme jen automaticky vyhodnocované formuláře):

1. Dotazníky: zaměřují se především na sběr odpovědí uživatelů pro zadávající stranu ([6]). Může se jednat například o hodnocení nějaké služby, názory na nějakou problematiku (průzkumy obyvatelstva) a podobně. Tyto formuláře neposkytují uživateli žádnou zpětnou vazbu kromě faktu, že je byl schopen správně vyplnit a byly správně odeslány.

2. Kvízy: Jedná se o testy, které si uživatel vyplňuje zpravidla pro zábavu (například test osobnosti) nebo proto, aby se něco dozvěděl (například, zda má příznaky nějaké nemoci). Kvíz obvykle dělí respondenty do několika předem definovaných typů podle toho, jak odpovídali. Uživateli je poté zobrazen výsledný typ.

3. (Znalostní) testy, zkoušky: Tyto formuláře slouží ke zkoušení znalostí uživatele. Bývají poměrně široce využívány (například testy z autoškoly, test po školení o bezpečnosti na pracovišti). Dobře reflektují známou situaci ze školních lavic, často bývají časově omezené. Občas je test sám schopen generovat náhodně vytvořené otázky nebo aspoň vybírat náhodné sady otázek. Uživatel je po skončení testu informován o úspěchu či neúspěchu, dozví se, které odpovědi nebyly správné a občas také jaké bylo správné řešení.

4. Anketa: podobá se dotazníku, ale zpravidla bývá kratší. Uživatel po hlasování vidí průběžné výsledky.

   Tvorba, propagace a management dotazníků jsou služby atraktivní pro většinu společností, které nabízí nějakou službu a zajímají se o to, jak je

vidí jejich zákazníci či zaměstnanci. Aplikace z první kategorie se tedy zaměřují na tyto klienty. Volnočasové kvízy lze zase využít jako marketingový nástroj, zvýšit povědomí o nějaké problematice či produktu [7]. Znalostní testy mohou mít využití pro učitele, studenty nebo například pro zaměstnavatele, který chce nebo musí udržovat povědomí o dovednostech svých zaměstnanců.

1.1 Vymezení záměru projektu Selftester

Jak již bylo nastíněno v úvodu, náš projekt si klade za cíl vytvořit platformu pro učení formou tréninku znalostních testů, tedy takových testů, které výsledky uživatele nějakým způsobem ohodnotí a oznámí mu, jak si vedl, popřípadě, zda prošel či neprošel. Aplikace tak má sloužit primárně jako pomůcka pro učení a opakování znalostí jakéhokoliv druhu. Testovaný uživatel je pro nás stejně důležitý jako tvůrce testu, možná i důležitější. Výsledky a zpětná vazba aplikace mají především pomoci studentovi ke zlepšení - bude si tedy moci prohlížet svou historii a uvidí vývoj svých výsledků v daném testu. Statistický sběr výsledků pro autory testu je až druhořadý. Tímto se Selftester nejvíce liší od nalezených konkurenčních řešení. Dalším významným rozdílem je důraz na edukační potenciál testů či kvízů. Většina konkurenčních aplikací nabízí spíše tvorbu volnočasových kvízů (typu do jaké kategorie podle svých odpovědí spadáte) anebo dotazníků, jejichž výsledky mají pochopitelně cenu především pro jejich autory.

1.1.1 Upřesnění pojmů

V rámci této diplomové práce vznikají souběžně dvě aplikace. Je vhodné si ujasnit, o které přesně dále v textu referujeme, zejména proto, že termín „webová aplikace“ může být vnímán různě. Termínem „aplikační logika“ je myšlen soubor procesů a logických postupů, které mají být implementovány aby konečná aplikace fungovala jak má. K implementaci takového procesu je zpravidla potřeba kooperace obou níže definovaných aplikací, některé ale mohou být realizovány jen na jedné z nich.

1. API: Slouží hlavně ke správě a poskytování dat, ale implementuje i část aplikační logiky (například vygenerování testu) API tedy představuje backend pro webovou a mobilní aplikaci.

2. Webová aplikace (nebo webový klient): Běží přímo u klienta - tedy v prohlížeči uživatele. Často se používá také termín frontendová aplikace, čímž se zdůrazňuje nepřítomnost backendové části - ta je realizována zvlášť. V této práci se používají obě pojmenování.

Je zřejmé, že uživatel interaguje přímo pouze s webovou aplikací.

Pod pojmem **Backend** se v oblasti webových služeb rozumí samotný server, databáze a aplikační logika. Hlavní je práce s daty, která musí být rychlá, bezpečná a efektivní. **Frontend** je pak samotná webová stránka, se kterou uživatel interaguje, vývojář se proto zaměřuje na to, poskytnout co nejlepší zážitek. **Backend** i **Frontend** se může řešit v rámci jedné webové aplikace, což je klasický přístup, pokud je aplikace hodně interaktivní anebo rozsáhlá, můžou se vyvíjet zvlášť ([8]). Jako klasický případ takto rozdělené aplikace se uvádí Gmail [9].

1.2 Konkurenční řešení

Primárně se zaměřujeme se na webové aplikace, které umožňují online vytvoření, správu a vyplňování testů, kvízů či dotazníků. Dále zkoumáme, zda k produktu patří i nějaká mobilní aplikace. Aplikace je v češtině a proto se budeme soustředit i na českou nabídku.

1.2.1 Metoda vyhodnocení konkurenčních řešení

Každá analyzovaná aplikace je ohodnocena podle několika kritérií ve dvou oblastech - nabízených funkcích a uživatelském rozhraní. V první oblasti zkoumáme, jak moc se nabízená funkcionalita přibližuje tomu, co bude implementováno v Selftesteru. V druhé oblasti analyzujeme uživatelské rozhraní aplikace a hledáme rozpory se zásadami Nielsenovy heuristické analýzy alias Nielsenova desatera 4.1.1.3. Body Nielsenovy heuristiky byly pro potřeby naší metody sloučeny do čtyřech výsledných kategorií a to na základě podobnosti zkoumané oblasti (například body heuristiky, které nějakým způsobem kontrolují stav systému, relevanci a konzistentnost zobrazených informací jsou pohromadě). Dalším důvodem je, že pokud by byly body zkoumány každý zvlášť, mnoho by zůstalo prázdných (bez zjištěné chyby). Každé kritérium bude bodově ohodnoceno dle poskytnuté stupnice. Bodování funguje podobně jako známky ve škole - nejkvalitnější aplikace bude ta s nejmenším počtem nasbíraných bodů (nejlepší možný výsledek je 8 bodů).

Kvalita uživatelského rozhraní je testována zejména v části aplikace, která se zabývá tvorbou samotných testů, popřípadě průchodem výsledků. Tyto dvě oblasti UI byly vyhodnoceny jako ty nejobtížnější (1.6)

1.2.1.1 Bodové ohodnocení:

- **1 bod**: aplikace splňuje kritérium.
- **3 body**: aplikace splňuje kritérium s výhradami.
- **5 bodů**: aplikace nesplňuje kritérium.

1.2.1.2 Kritéria hodnocení:

1. Funkcionalita:
   a) Možnost vytvoření vlastního testu (zdarma).
   b) Možnost rozšířit nebo sdílet test mezi potenciálními uživateli.
   c) Možnost získat historii vlastních vyplněných testů.
   d) Propojení s mobilní aplikací.
2. Uživatelské rozhraní: Kritéria jsou vlastně sloučené zásady Nielsenovy heuristické analýzy. Jsou uvedeny originální anglické názvy těchto zásad, jelikož překlad by mohl bez dalšího kontextu být zavádějící. Důkladný popis včetně překladů zásad do češtiny se nachází v kapitole Analýza (4.1.1.3)
   a) Zásady: Recognition rather than recall, visibility of system status, Consistency and standards, Aesthetic and minimalistic design. Tato položka kontroluje, zda má systém přiměřené nároky na paměť uživatele, zda ho nezahlcuje informacemi, které nejsou potřebné, zda je uživatel vždy obeznámen se stavem systému a zda jsou zobrazované zprávy srozumitelné a konzistentní.
   b) Zásady: Error prevention, User control and freedom. Položka se zabývá zejména kvalitou formulářů (obecně zadávání vstupů a provádění změn) v aplikaci. Zkoumá, zda je možné například zvrátit některé akce nebo jak aplikace zareaguje na chybně vyplněné formuláře apod.
   c) Zásady: Flexibility and efficiency of use, Match between system and the real world. Zkoumá se, zda aplikace nabízí základní a pokročilé funkce vhodně oddělené, ale zároveň dostupné (jinými slovy aplikace uspokojí náročné uživatele, ale nový uživatel se v ní neztratí). Dále kontrolujeme vhodně použité pojmy a přirovnání. Vzhledem k tomu, že školní písemku či firemní dotazník poznal na vlastní kůži prakticky každý, mohli by být uživatelé na toto citliví.
   d) Zásady: Help users recognize, diagnose, and recover from errors, Help and documentation. Jde zejména o srozumitelné chybové hlášky a o správné nasměrování uživatele k nápravě. Pokud tyto informace nestačí, aplikace nabízí možnost ucelenější nápovědy.

Další zájmové oblasti:

- cílová skupina aplikace
- dostupnost, flexibilita (např. je možné vyplnit test bez registrace, jsou testy zaměřené pouze na nějaké téma?)

Tyto oblasti nehrají roli pro celkové hodnocení aplikace v rámci navržené metodiky, ale jsou pro analýzu konkurence i přesto důležité.

1.2.2 Aplikace v anglickém jazyce

1.2.2.1 QuizMaker [10]

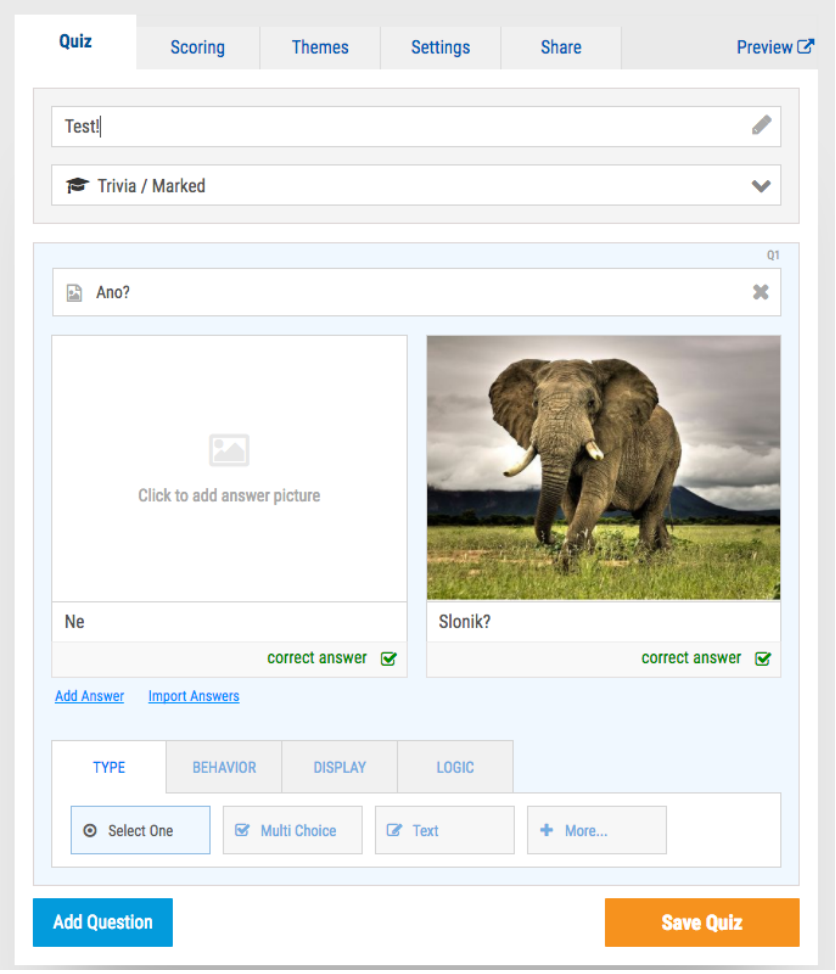
QuizMaker umožňuje uživatelům vytvářet jak znalostní (ohodnocené) testy, tak volnočasové kvízy, kde se výsledek dělí do několika kategorií (například typy osobností). Mezi ostatními analyzovanými produkty zcela vyniká přístupností své hlavní funkce - tedy tvorby testu. Formulář na vytvoření testu je totiž to první, co uživatel uvidí, když vstoupí na stránku, nepotřebuje registraci a může test okamžitě vytvořit a sdílet na sociálních sítích. Aplikace má propracovaný proces tvorby testů z pohledu UI. Dále aplikace nabízí množství možností pro vyplňování testu, například po kolika otázkách má dojít k přechodu na novou stránku nebo zda se má po vyplnění otázky automaticky přejít na novou stránku.

Klady:

- editor otázek a odpovědí umí vkládat obrázky, komplikovanější nastavení šikovně schovává tak, aby uživatele nerušilo a „neděsilo“, zároveň je ale snadno dostupné - pokud je uživatel hledá
- test je možné vytvořit bez registrace - snadné seznámení s použitím aplikace
- neplacená trial verze (je možno využívat základní licenci po dobu třiceti dní zdarma)
- propojení se sociálními sítěmi (snadné sdílení vytvořených testů)

Zápory:

- UI má v některých momentech nepředvídatelné chování - například při změně typu otázky bez varování vymaže již vyplněné odpovědi
- orientuje se pouze na autory testů, ne na vyplňující, nelze tedy například vidět historii svých vyplněných testů
- malá velikost písma ve formuláři



Obrázek 1.1: Vytváření testu v aplikaci QuizMaker

1.2.2.2 Online Quiz Creator [11]

Tato aplikace se vyniká především jednoduchostí a snadnou dostupností. Test je možné vytvořit bez registrace. Nabízí také soutěžní prvky, a jednoduchou správu výsledků. Web je zaměřen především na učitele, testy jsou primárně určeny jako doplněk výuky. Aplikace má z pohledu funkcionality několik variant, ta nejzákladnější je dostupná zdarma. Jako jediná ze zkoumaných aplikací působí, že je do nějaké míry zaměřená na dětské uživatele a je určena do školního prostředí.

Klady:

- Přímočarost - snadný přechod k tvorbě testu.
- Testy mají několik šablon (herní kvíz, zkouška).

	Body	Odůvodnění
1. Funkcionalita		
a)	1	Jedná se o první věc, kterou uživatel uvidí na stránce.
b)	1	Široká škála možností sdílení - odkazem, přes sociální sítě atp.
c)	5	Kvízy nejsou na stránce dostupné, natož jejich historie.
d)	5	Nenalezeno.
2. Uživatelské rozhraní		
a)	3	Přehled vytvořených testů a i editor jsou dobré, při větším počtu otázek mi ale UI nijak nepomůže, je to prostě dlouhá nepřehledná řada formulářů.
b)	1	Nesprávně vyplněná pole jsou dobře vyznačena a uživatel je dobře poučen o chybě.
c)	1	Silná stránka editoru, nabízí řadu pokročilých nastavení, ale zároveň vše funguje dobře bez nich, pokud je uživatel nehledá.
d)	1	viz. b), uživatel je dobře poučen, jak své chyby napravit. Editor také obsahuje různé vysvětlivky, co která položka znamená i s příklady.
Celkem		18

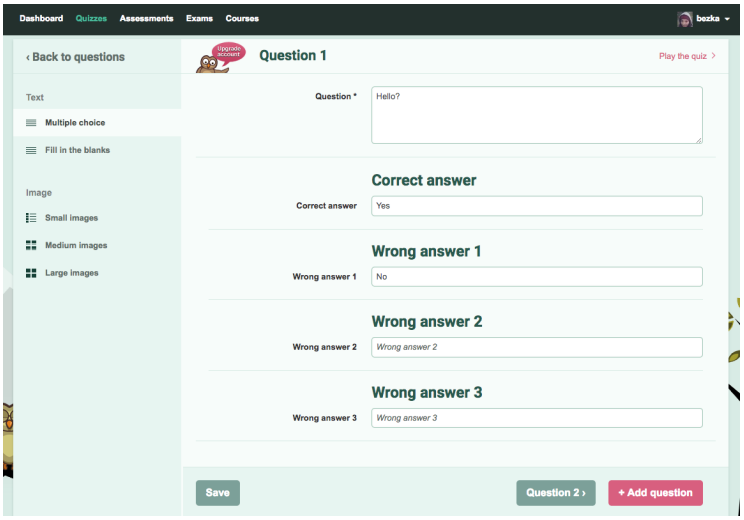
Tabulka 1.1: Hodnocení aplikace QuizMaker

- Produkt je jasně zamířen na cílovou skupinu a uzpůsoben cílové skupině - učitelům.

Zápory:

- Otázky jsou pouze testového typu (jedna nebo více správně) a lze do nich zadat pouze text.

- Závažné chyby z pohledu UI - formulář s novou otázkou nereaguje na tlačítko odeslat, pokud je prázdný (a tedy nevalidní), chybí přehled již vytvořených otázek, což znesnadňuje uživateli orientaci.



Obrázek 1.2: Vytváření testu v aplikaci Online Quiz Creator

	Body	Odůvodnění
1. Funkcionalita		
a)	1	Je nutná registrace.
b)	1	Lze sdílet přes odkaz či na sociálních sítích, lze také kvíz vložit do vlastní stránky.
c)	5	Z pohledu „vyplňujícího“ kvízy nelze vyhledat, natož mít nějakou historii.
d)	5	Nenalezeno.
2. Uživatelské rozhraní		
a)	1	Editor testů je robustní a přehledný (užitečná drobečková navigace), uživatel nemá pocit, že se mu během editace něco ztratí.
b)	3	Nesprávně vyplněná pole jsou po odeslání formuláře opatřena chybovou hláškou, ale zvýraznění chybových polí by mohlo být výraznější.
c)	5	Veškerá nastavení jsou viditelná pro jakéhokoliv uživatele - dokonce i ta přístupná jen pro platící uživatele, což značně snižuje přehlednost stránky.
d)	3	viz b).
Celkem		24

Tabulka 1.2: Hodnocení aplikace Online Quiz Creator

1.2.2.3 QZZR [3]

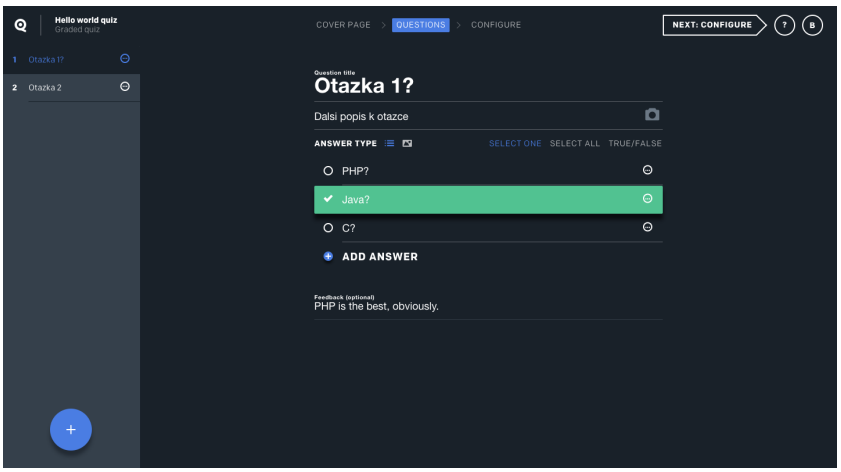
Tato aplikace působí z porovnávaných zdaleka nejvíce profesionálně. Je pojata velmi interaktivně a z pohledu grafického vzhledu a návrhu UI se nejvíce podobá našim představám. Na druhou stranu se nezaměřuje na výuku nebo pomoc při učení, ale na generování nových uživatelů spíše pomocí volnočasových kvízů, které mají vyplňující především bavit. Slouží tedy primárně jako obchodní nástroj. Nemá také žádnou mobilní aplikaci.

Klady:

- propracované uživatelské rozhraní
- možnost testy vkládat do vlastních stránek
- dobře pracuje s vkládáním médií

Zápory:

- neposkytuje dostatečnou zpětnou vazbu uživateli, který by nástroj používal k opakování učiva, uživatel si nemůže procházet své staré výsledky
- nenabízí možnost generovat test ze sady otázek



Obrázek 1.3: Vytváření testu v aplikaci QZZR

	Body	Odůvodnění
1. Funkcionalita		
a)	1	Ano, po registraci.
b)	1	Lze sdílet přes odkaz či na sociálních sítích, lze také kvíz vložit do vlastní stránky.
c)	5	Na stránce si lze vyplnit pár ukázkových testů, výsledek ale není pro uživatele nikam zaznamenán.
d)	5	Nenalezeno.
2. Uživatelské rozhraní		
a)	3	Chybí nějaká forma drobečkové navigace. Po přihlášení uživatel vidí jen přehled svých testů a nemůže se dostat na původní domovskou stránku. V rámci editoru testů se uživatel orientuje už lépe, ale pořád může mít pocit, že je tam jaksi „uzamčen“.
b)	1	Nezjištěn problém.
c)	1	Nezjištěn problém.
d)	1	Chybové hlášky jsou pochopitelné, aplikace navíc zjistí, pokud ji uživatel používá poprvé a velmi účinně jej provádí celým procesem editace, zobrazuje tipy, co kam vyplnit, aby byl test například populárnější a tak dále.
Celkem		18

Tabulka 1.3: Hodnocení aplikace QZZR

1.2.2.4 Google Forms [12]

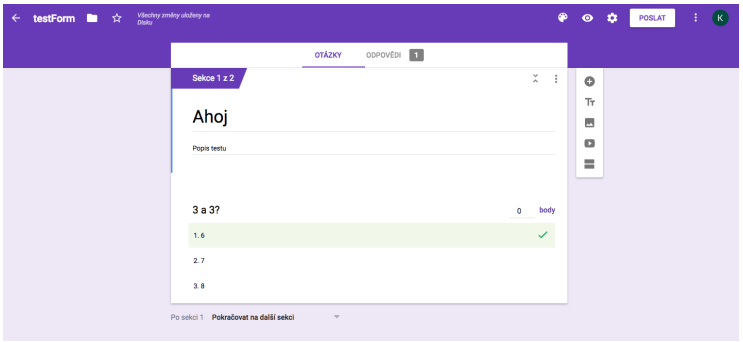
Google Forms je součástí souboru online kancelářských aplikací Google Docs [13] a slouží k vytváření formulářů obecně. Aplikace je častěji využívána ke shromažďování dat skrze dotazníky (a na domovské stránce můžeme najít šablony k různým dotazníkům), lze ale formulář změnit na kvíz a poté bude výsledek ohodnocen. Pro používání aplikace je nutné mít založen účet Google.

Klady:

- Propracované rozhraní editace formulářů
- Možnost editace ve více lidech
- Průběžné ukládání

Zápory:

- Celý editovaný formulář se nachází na jedné (potenciálně velmi dlouhé stránce). Tím může utrpět orientace uživatele na stránce.
- Chybí pokročilejší nastavení, například časový limit na kvíz.
- Pokud je typ odpovědi „textová odpověď“, není možno navolit klíč k vyhodnocení správnosti.



Obrázek 1.4: Vytváření testu v aplikaci Google Forms

	Body	Odůvodnění
	1. Funkcionalita	
a)	3	Formulář je nutno přepnout do varianty „kvíz“. Tato možnost je poněkud ukrytá v obecném nastavení.
b)	1	Lze sdílet přes odkaz na sociálních sítích, pozvat respondenty přes email nebo formulář vložit do vlastní stránky.
c)	5	Historie vyplněných testů pro respondenta opět chybí.
d)	5	Nenalezeno.
	2. Uživatelské rozhraní	
a)	3	Chybí nějaká forma přehledu vytvořených otázek. Otázky se navíc vkládají pořád na jednu stránku, ta tedy může být časem velmi dlouhá a uživatel ztrácí přehled.
b)	1	Nezjištěn problém.
c)	1	Nezjištěn problém.
d)	3	Vkládání a práce s otázkami je v pořádku, kvízové rozšíření formuláře postrádá vysvětlivky.
Celkem		22

Tabulka 1.4: Hodnocení aplikace Google Forms

1.2.3 Aplikace v českém jazyce

1.2.3.1 Survio [14]

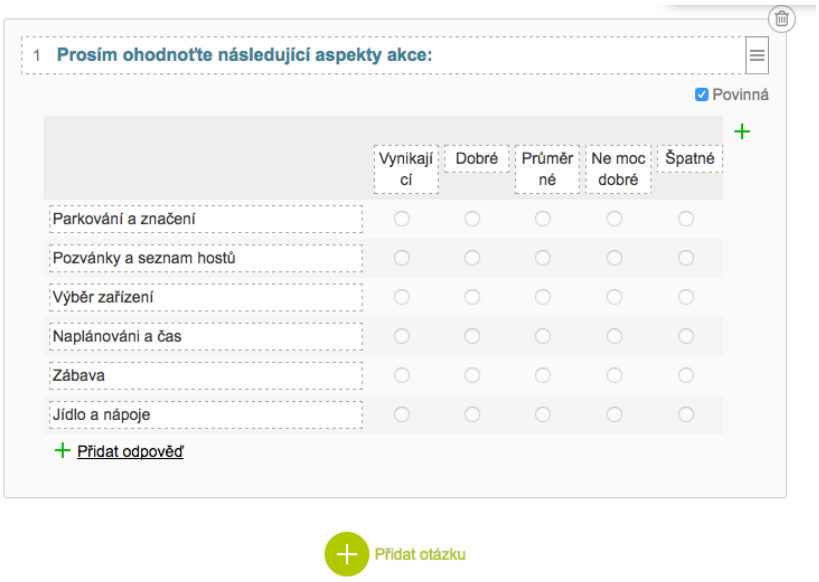
Aplikace Survio se primárně zaměřuje na snadné vytváření dotazníků, sběr a vyhodnocení výsledků. Poskytuje množství funkcí pro správu otázek (například prohazování pořadí otázek) a umožňuje uživateli nastavit si vzhled stránky s testem.

Klady:

- Velké množství typů otázek (například seřazení odpovědí dle důležitosti)
- Webová stránka je přehledná a dobře strukturovaná - je jasné, co se nabízí, k čemu to slouží a za jakých podmínek.

Zápory:

- Testy fungují pouze jako dotazníky - nehodnotí uživatele a neposkytují mu zpětnou vazbu.
- Otázky dotazníku se zobrazí jednoduše všechny na jednu stránku, což vede ke snížené použitelnosti a přehlednosti pro velký počet otázek.



Obrázek 1.5: Vytváření testu v aplikaci Survio

	Body	Odůvodnění
1. Funkcionalita		
a)	3	Ano, po reqistraci. Nejedná se ale o test v pravém slova smyslu, uživateli není poskytnut žádný výsledek.
b)	1	Lze sdílet přes odkaz, na sociálních sítích nebo emailem.
c)	5	Survio nabízí dotazníky, výsledky jsou tedy z principu pro vyplňujícího nerelevantní.
d)	5	Nenalezeno.
2. Uživatelské rozhraní		
a)	3	Chybí nějaká forma drobečkové navigace, dále chybí nějaký přehled otázek v editoru dotazníku, uživatel tak ztrácí přehled o tom, jaké otázky už tam má či nemá.
b)	1	Nezjištěn problém. Aplikace účinně používá různé dynamické typy formulářových prvků, aby tomuto zabránila.
c)	1	Nezjištěn problém. Rozšířené možnosti editace otázek jsou skryté ve zvláštním menu.
d)	1	Chybové hlášky i napovídání uživateli jak v editoru postupovat jsou opět na vysoké úrovni.
Celkem		20

Tabulka 1.5: Hodnocení aplikace Survio

1.2.3.2 VyplňTo [15]

Tato aplikace funguje podobně jako Survio, primárně nabízí tvorbu, vyplňování a vyhodnocování dotazníků. Nad to také nabízí databázi respondentů a často se využívá pro průzkumy veřejného mínění. Jako jediná ze sledovaných služeb poskytuje také mobilní aplikaci s možností vyplňování dotazníků offline. Na svých webových stránkách avizují brzké spuštění nové funkcionality - online testů s automatickým vyhodnocováním. Pokud službu spustí, bude se jednat o přímou konkurenci naší aplikace.

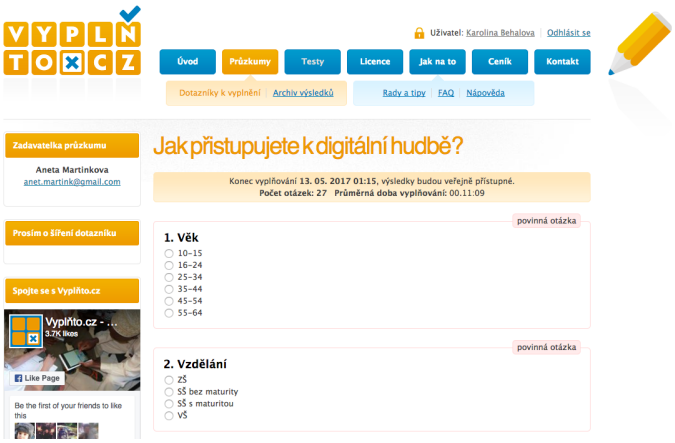
Klady:

- Zavedená služba v oblasti průzkumů.
- Přehledný web a kvalitní sekce s návody a nápovědou pro uživatele.

Zápory:

- Otázky dotazníku se zobrazí jednoduše všechny na jednu stránku, což vede ke snížené použitelnosti a přehlednosti pro velký počet otázek.

- Aktuálně testy fungují jen jako dotazníky.



Obrázek 1.6: Průchod testem v aplikaci VyplňTo

	Body	Odůvodnění
1. Funkcionalita		
a)	3	Ano, po registraci a vyplnění několika cizích testů. Nejedná se o test, ale o dotazník, uživateli není poskytnut žádný výsledek.
b)	1	Lze sdílet přes odkaz, na sociálních sítích nebo emailem.
c)	5	VyplňTo nabízí (zatím) dotazníky, výsledky jsou tedy z principu pro vyplňujícího nerelevantní.
d)	1	Služba nabízí mobilní aplikaci na vyplňování dotazníků.
2. Uživatelské rozhraní		
a)	3	Chybí nějaká forma přehledu vyplňovaného testu, při vyplňování otázek nevidíme žádné obecné údaje o testu.
b)	3	Aplikace některé chyby prostě ignoruje, například nechá uložit otázku, ke které nejsou definované žádné odpovědi.
c)	1	Editor dotazníku je možné spustit v několika režimech podle pokročilosti uživatele.
d)	1	Chybové hlášky i napovídání uživateli jak v editoru postupovat jsou opět na vysoké úrovni.
Celkem		18

Tabulka 1.6: Hodnocení aplikace VyplňTo

1.2.4 Vyhodnocení rešerše konkurence

1.2.4.1 Společné rysy konkurenčních aplikací

Bylo zjištěno, že většina stávajících řešení se soustředí především na první dvě kategorie testových formulářů - tedy kvízy a dotazníky. Nutno dodat, že nástroj na tvorbu vlastních zkouškových testů bývá často součástí větší edukační nebo podnikové aplikace - například Moodle [16]. I v těchto případech se ale aplikace zaměřují spíše na učitele a sběr finálních výsledků, než na studenty. Zkoumané aplikace mají několik často se opakujících rysů. Testy či dotazníky lze vytvářet podle jednoduchých šablon, často lze i používat šablony odpovědí. Důležitým prvkem také je, že hotové testy jsou snadno dostupné pod vlastní URL (tudíž se snadno sdílí) nebo se dají přímo vložit do stránky. Webová aplikace, která by se zaměřovala na potřeby testovaného a zároveň umožňovala vytvářet testy libovolného obsahu, tedy nebyla na českém trhu nalezena.

1.2.4.2 Bodové vyhodnocení konkurenčních aplikací

Aplikace	Skóre	Pořadí
QuizMaker	18	1. - 3.
QZZR	18	1. - 3.
VyplňTo	18	1. - 3.
Survio	20	4.
Google Forms	22	5.
Online Quiz Creator	24	6.

Tabulka 1.7: Celkové bodové vyhodnocení konkurenčních aplikací

Bylo zjištěno, že žádná ze zkoumaných aplikací nesplňuje požadavek **1 c)** 1.2.1.2 - nenabízí historii vyplněných testů pro testovaného uživatele. Jen jedna aplikace nabízí svým uživatelům také mobilní verzi 1.6. Nejčastější problém s uživatelským rozhraním byl ten, že aplikace nezohlednily situaci, kdy by uživatel chtěl vložit do testu či dotazníku opravdu velký počet otázek, například 100. Pokud by se všech 100 otázek v editoru nacházelo na jedné stránce bez jakéhokoliv souhrnného přehledu nebo rozdělení do sekcí - tedy všechny otázky by se nacházely jednoduše pod sebou a tvůrce testu by byl odkázán, mohlo by dojít ke ztížení orientace uživatele na stránce a tak dále. Tato problematika je dále rozvinuta níže (1.6).

Nejlepšího hodnocení dosáhly rovnou tři konkurenční aplikace. Jelikož Selftester bude realizován v českém jazyce, jako hlavní oponent byla zvolena aplikace VyplňTo (1.2.3.2), jenž je také v češtině.

1.3 Funkční požadavky

Uživatel bude mít k dispozici následující funkce:

1. Správa uživatelského účtu
   a) registrace
   b) přihlášení, odhlášení
   c) editace uživatelského profilu
2. Prohlížení a vyplňování testů
   a) vyhledat test nebo tag nebo uživatele podle username
   b) filtrovat seznam testů podle populárních tagů
   c) zobrazit detail testu
   d) přidat komentář k testu
   e) odebrat komentář k testu (pokud na to má oprávnění)
   f) ohodnotit test
   g) seřadit testy podle abecedy, hodnocení a popularity
   h) vyplnit test
   i) zobrazit výsledky
   j) přidat test do svého seznamu oblíbených
   k) nahlásit chybu v otázce
   l) sdílet výsledek na sociální síti Facebook
   m) sdílet celý test na sociální síti Facebook
   n) získat permanentní odkaz na daný detail testu
3. Správa testů
   a) vytvořit test s obecným nastavením
      i. nastavit jméno, krátký popis
      ii. nastavit počet otázek pro test - otázek může být víc, ale bude se generovat právě tento počet
      iii. nastavit časový limit pro test
   b) přidat nebo odebrat otázku k testu
   c) uložit rozpracovaný test
   d) publikovat test (uživatelé mohou vyplnit pouze publikovaný test)
   e) smazat test
4. Správa testových otázek

   a) vytvořit otázku s nastavením
      i. typ otázky (právě jedna odpověď správně atp)
      ii. bodové ohodnocení otázky
      iii. znění otázky
      iv. přidat odpověď nebo odpovědi k otázce
      v. nastavit správné odpovědi
      vi. přidat vysvětlivky k otázce
   b) vymazat otázku
   c) editovat otázku
5. Vkládání multimediálního obsahu
   a) nahrát obrázek z URL nebo z počítače k testu
   b) nahrát obrázek z URL nebo z počítače k otázce
6. Přístup k historii testů
   a) zobrazit si výsledek každého dokončeného testu

1.4 Případy použití (use cases)

Případy použití byly vytvořeny na základě funkčních požadavků.

1.4.1 Účastníci

1. Anonymní uživatel: účastník, který není přihlášen do systému
2. Přihlášený uživatel: účastník, který má vlastní účet a je k němu přihlášený
3. Editor testu: účastník, který vytvořil nebo byl přidán jako editor k nějakému testu

1.4.2 Seznam případů užití

UC1 Vytvořit uživatelský účet

UC2 Editovat údaje v uživatelském účtu

UC3 Vyhledávat testy

UC4 Vyhledávat uživatele a tagy

UC5 Filtrovat seznam testů

UC6 Hodnotit testy

UC7 Přidávat komentáře k testům

UC8 Přidávat testy do oblíbených

UC9 Vyplňovat testy

UC10 Nahlásit chybu v testu

UC11 Procházet historii vyplněných testů

UC12 Sdílet výsledek testu na sociální síti

UC13 Vytvořit veřejný test

UC14 Přidat otázku do testu

UC15 Odebrat otázku z testu

UC16 Nahrát obrázek k testu/otázce

UC17 Spravovat seznam editorů testu

UC18 Měnit stav testu na veřejný/neveřejný

1.4.3 Tabulka pokrytí (1.8) a diagram případů použití

Seznam funkčních požadavků bez detailů

F1 Správa uživatelského účtu

F2 Prohlížení a vyplňování testů

F3 Správa testů

F4 Správa testových otázek

F5 Vkládání multimediálního obsahu

F6 Přístup k historii testů

1.4.3.1 Tabulka - mapování FP na UC

1.4.3.2 Diagram případů použití

1.5 Nefunkční požadavky

Požadavky kladené na systém z hlediska implementace, zabezpečení, popřípadě udržitelnosti:

	F1	F2	F3	F4	F5	F6
UC1	✓	✗	✗	✗	✗	✗
UC2	✓	✗	✗	✗	✗	✗
UC3	✗	✓	✗	✗	✗	✗
UC4	✓	✓	✗	✗	✗	✗
UC5	✗	✓	✗	✗	✗	✗
UC6	✗	✓	✗	✗	✗	✗
UC7	✓	✓	✗	✗	✗	✗
UC8	✓	✓	✗	✗	✗	✗
UC9	✗	✓	✗	✗	✗	✗
UC10	✗	✓	✗	✗	✗	✗
UC11	✓	✗	✗	✗	✗	✓
UC12	✓	✓	✗	✗	✗	✗
UC13	✗	✗	✓	✗	✗	✗
UC14	✗	✗	✓	✓	✗	✗
UC15	✗	✗	✓	✓	✗	✗
UC16	✗	✗	✓	✓	✓	✗
UC17	✓	✗	✓	✓	✗	✗
UC18	✗	✗	✓	✗	✗	✗

Tabulka 1.8: Tabulka pokrytí funkčních požadavků případy použití

1.5.1 Nefunkční požadavky na API

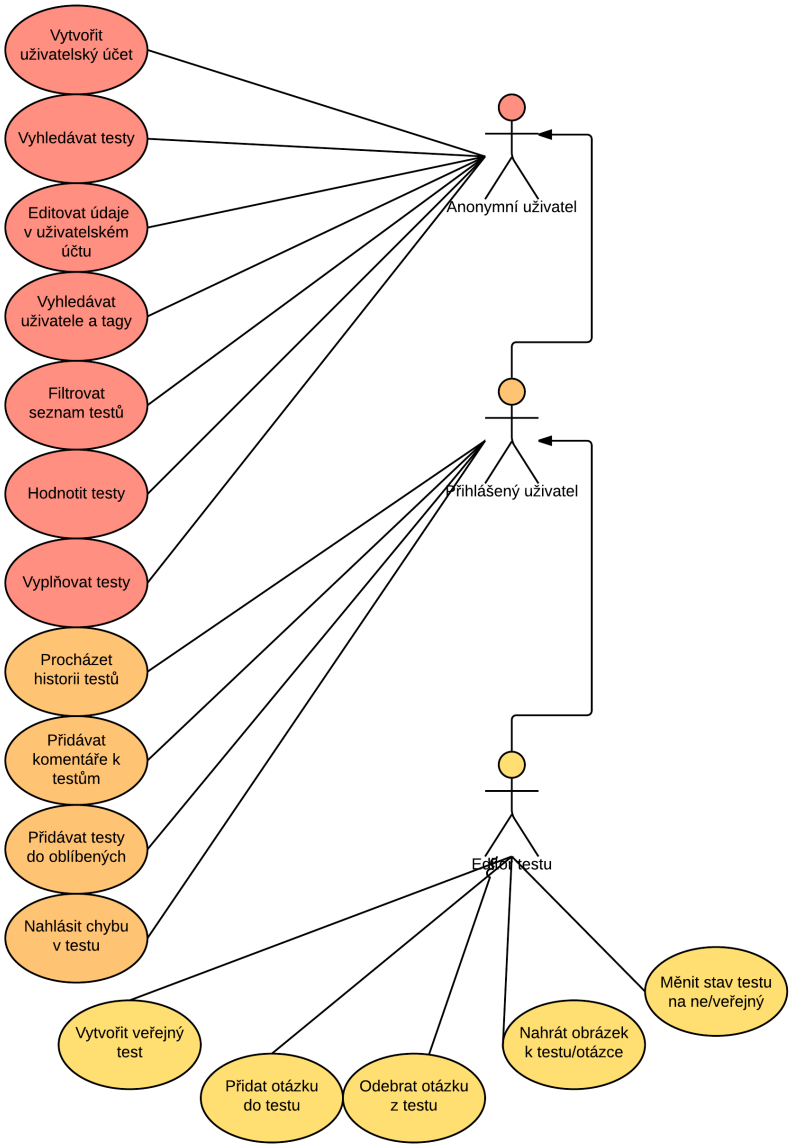
1. Autentizace a zajištění integrity zpráv pomocí JWT tokenů [17].

2. Detail testu včetně hodnocení a komentáře je možné stáhnout zvlášť (vlastní endpointy).

3. API je dostatečně pokryto testy.

1.5.2 Nefunkční požadavky na webového klienta

1. Klient udržuje sezení uživatele pomocí JWT tokenů.



Obrázek 1.7: Diagram případů použití podle účastníků

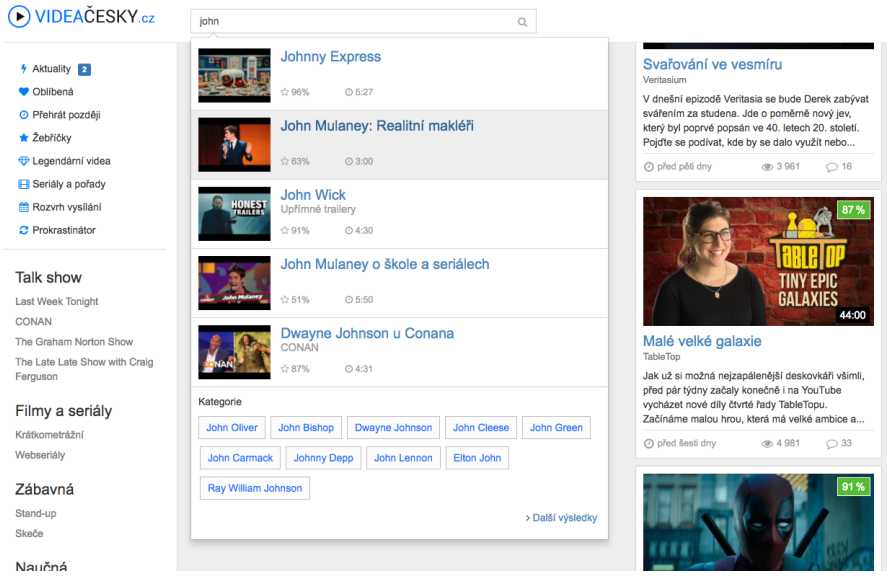
1.6 Klíčové prvky pro uživatelského rozhraní Selftesteru

Následující UI komponenty byly vyhodnoceny jako důležité pro pohodlné používání naší aplikace v celém jejím plánovaném rozsahu. Než se tedy pustíme do jejich návrhu, je vhodné analyzovat, jak k nim přistoupit, na jaké klíčové body UI se zaměřit a ukázat si podobné řešení na příkladu.

1.6.1 Chytré vyhledávání testů

Tato funkce bude dostupná pro registrované i anonymní uživatele a bude hrát důležitou roli při objevování vhodných testů pro návštěvníka. Obecně vzato jde o důležitou část každé složitější stránky, která by měla být snadno použitelná a také snadno k nalezení [18].

Vyhledávání bude fungovat pomocí tzv. našeptávání, uživateli budou nejlepší výsledky nabídnuty před odesláním vyhledávacího formuláře a bude na ně moci rovnou přejít. Zároveň ale bude k dispozici stránka s kompletními výsledky, na kterou uživatel bude moci přejít. Touto cestou bude také předem upozorněn, že zadaná klíčová slova žádné výsledky nevrací. Kromě konkrétních testů bude vyhledávání také nabízet tagy, které jsou testům přiřazeny, a uživatele. Důležitá je jak dobrá viditelnost vyhledávacího panelu, tak vhodně zvolený obsah panelu našeptávání [19].



Obrázek 1.8: Ukázka chytrého vyhledávání na webu VideaČesky [2]

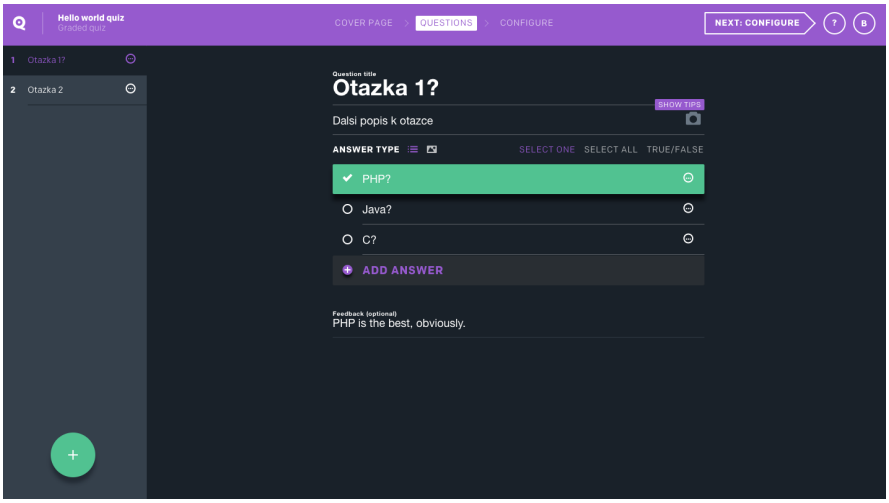
1.6.2 Editor testů

Editor testů je klíčová část celé webové aplikace. Obecně je nutno navrhnout (podle vhodných směrnic [20]) sadu několika různých formulářů, které se budou dynamicky generovat (například budeme mít proměnlivý počet odpovědí k otázce).

Hlavní cíle UI editoru testů:

- Poskytnout uživateli kontext a pomoci mu v orientaci - na jakém testu vlastně pracuje, kolik otázek už má vytvořených a tak podobně. Ve velmi dlouhém formuláři uživatel snadno ztratí přehled [20].

- Test bude moci obsahovat desítky otázek a bod 1 bude přesto zachován.
- Omezit počet nutných klikání, ale zároveň dát uživateli najevo, že žádný obsah se mu neztratí a bude se moct kdykoliv vrátit.



Obrázek 1.9: Ukázka editoru testů v hodnocené aplikaci QZZR [3]

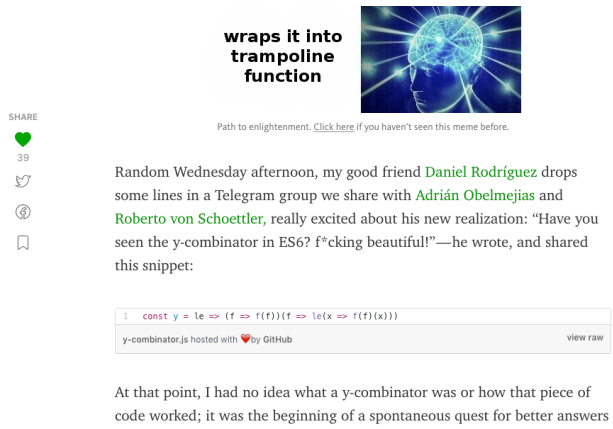
1.6.3 Snadné přidání testu do oblíbených

Tento požadavek vychází z předpokladu, že testovaný uživatel bude možná chtít nalezený test opakovat nebo si ho odložit na později. Jinými slovy bude chtít vytvořit nějaký vlastní seznam testů, které ho zaujaly. Je proto důležité, aby tato funkce byla snadno dostupná na různých místech aplikace (při listování testy nebo po odeslání vypracovaného testu) a aby byla snadno rozpoznatelná jak pro přidání do seznamu tak pro zobrazení samotného seznamu. Toho dosáhneme podle [21] pomocí podobných vizuálních prvků (nejčastěji se používají ikony na motiv srdce nebo hvězdy). Zároveň je nutné zdůraznit, že se nejedná o ohodnocení testu.

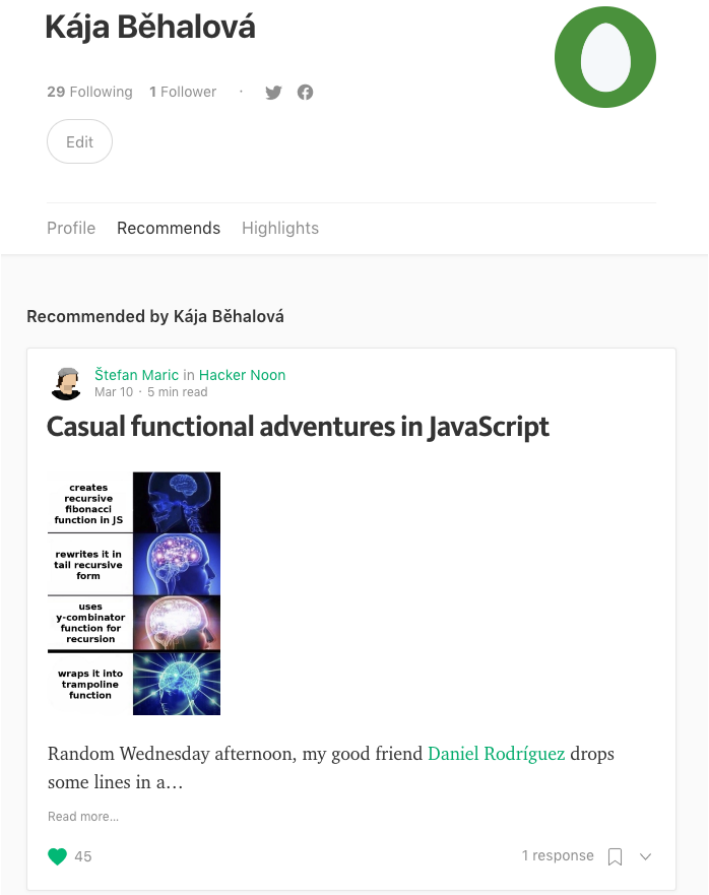
1.7 Shrnutí

V této kapitole byla provedena analýza jak českých tak zahraničních konkurenčních aplikací (1.2.4) a byl identifikován největší konkurent - aplikace VyplňTo (1.2.3.2). Dále byly shromážděny funkční i nefunkční požadavky (1.3) a případy použití. Dále byly identifikovány a rozebrány klíčové části uživatelského rozhraní (1.6). V rámci klientské aplikace budou implementovány pouze některé funkční požadavky tak, aby to odpovídalo zadání práce (3, 4 a 1). To samé platí pro API, jeho funkcionalita ale bude většího rozsahu, jelikož

musí obsluhovat požadavky jako webového tak mobilního klienta (funkční požadavky 1, 6, 3, 4, 2).



(a) Na levé straně se nachází možnost přidat článek do oblíbených pomocí ikony srdce



(b) Následný vygenerovaný seznam oblíbených

Obrázek 1.10: Ukázka použití „Přidat do oblíbených“ ve webové aplikaci Medium [4]

Kapitola 2

Metodiky vývoje software

Vývoj software je náročný a často dlouhotrvající proces, existuje však řada možností, jak tento proces zefektivnit. Metodika vývoje software je soubor postupů a pravidel pro návrh, plánování a řízení tohoto procesu. Metodika odpovídá na otázky jako „Jaké fáze budeme v průběhu vývoje rozlišovat?“ anebo „Co bude vstupem a výstupem těchto fází?“ [22]. Obecně se jedná o způsob, jak dělit práci v týmu vývojářů, dále je důležité, jak do procesu bude zapojen zadavatel práce a ideálně i jak bude každý provedený krok zpětně vyhodnocen [23]. Metodika je většinou nastavena v rámci týmu, jenž na zadané aplikaci nebo systému pracuje (do tohoto týmu patří nejen programátoři, ale i analytici, kodéři, administrátoři a tak dále), ale může být zavedena i v rámci celé organizace [23].

2.1 Volba správné metodiky

Při výběru je nutno uvážit množství faktorů, detailní popsání celého procesu je ovšem nad rámec této práce, proto budou kritéria jen krátce nastíněna:

- Velikost projektu
- Velikost týmu
- Preference a zvyklosti klienta
- Stabilita požadavků na produkt

Je nutno mít na paměti, že ačkoliv metodika přináší množství výhod, jedná se vždy o práci navíc [22]. Pokud navíc zvolíme komplexní typ metodiky pro vývoj velmi jednoduchého projektu, může se stát, že tým stráví dokumentací jednotlivých kroků či obecně dodržováním zásad této metodiky příliš mnoho času na úkor dalších projektů. Metodiku je proto nutno volit citlivě, obecně platí, že pro menší projekty jsou vhodnější spíše agilní druhy a pro velké a

dlouhotrvající projekty s detailní specifikací od klienta zase ty klasické (viz níže).

2.2 Základní druhy metodik

Rozlišujeme metodiky klasické a agilní [22]. Obecně agilní metodiky povolují a vítají změny požadavků a zpětnou vazbu od klienta v průběhu vývoje, dodávají produkt v několika iteracích a jsou vhodnější pro menší projekty. Důležitá je komunikace v týmu a aktivní zadavatel - díky tomu se produkt postupně vylepšuje. Klasické metodiky bývají komplexnější, obecně produkují více dokumentace a hodí se pro rozsáhlé projekty.

1. Klasické metodiky (podle modelu životního cyklu)

   a) Vodopád: Jedná se o čistě sekvenční přístup. Mezi fázemi (specifikace, návrh, implementace, integrace, dodání a údržba) nelze přeskakovat ani se vracet. Požadavky na systém se tedy nesmí změnit. Tento přístup se již moc nevyužívá, jelikož se v průběhu nemůže přizpůsobit žádné změně v zadání.

   b) Spirálový model [24]: Jedná se o velmi komplexní model, který vyžaduje zkušený tým a vedoucí projektu. Vývoj iteruje ve čtyřech fazích (definování cílů, analýza rizik, vývoj a testování, plánování další fáze). Po dokončení jedné iterace se tým rozhodne, jestli je třeba pokračovat nebo skončit. Není problém v průběhu procesu zpracovávat nové funkční požadavky, neustále jsou kontrolována rizika projektu. Tato metodika je vhodná na velké a riskantní projekty, řadí se mezi nejnáročnější na čas a schopnosti týmu.

2. Agilní metodiky

   a) Extrémní programování [22]: Tato metoda si zakládá na jednoduchém přístupu. Programuje se jen to, co je v dané chvíli potřeba, nikdy nic předem, návrh je tedy co nejjednodušší. Je nutná velmi častá komunikace se zadavatelem i v rámci týmu. Pokud aktuální zadání neřeší problém, celé se zahodí a vytvoří se jiné - programátoři se nesmí bát do projektu výrazně zasáhnout. Vše se neustále testuje, takže tým i zákazník znají stav produktu v každé jeho fázi. Vzhledem k tomu, že nové verze produktu vznikají velmi často, měla by součástí procesu i průběžná integrace. Velmi obtížně se tvoří jakékoliv odhady a tento přístup je velmi náročný na čas zadavatele. Obecně se tedy dá říci, že tento přístup vývoje software je velmi náročný na čas a důvěru zadavatele v tým.

   b) Test-driven development [22]: Jedná se vlastně o jednu z programovacích praktik, jenž spadají pod Extrémní programování (dalšími

jsou již zmíněná průběžná integrace nebo důraz na jednoduchý návrh). V zásadě jde o to, že než se implementuje nějaká nová funkcionalita, nejprve jsou vytvořeny příslušné testy. Pokud se najde chyba, vytvoří se na ni nové testy. Jednotkové testy se berou jako nutnost, nejsou ale dostačující (dále by se měly pravidelně provádět například akceptační testy).

c) SCRUM: Patrně nejznámnější z agilních metodik. Vývoj se skládá z předem plánovaných iterací, zvaných sprinty. Každý sprint má naplánovaný seznam funkčních požadavků, jenž se mají realizovat. Metodika zavádí tři fáze [22]:

   i. Předehra: plánování, sběr funkčních požadavků, sestavení tzv. „product backlog“ - veškerá funkcionalita, která se od aplikace očekává, seřazená podle priorit zadavatele a podle logických závislostí.

   ii. Hra: Iterativně probíhají jednotlivé sprinty (jeden sprint trvá typicky jeden měsíc). Během jednoho sprintu se projde všemi standardními fázemi vývoje software (analýza, návrh, implementace, testování, předání). Na začátku sprintu je během plánovací schůzky sestaven Sprint backlog - požadavky z Product backlog. Během sprintu se konají každodenní schůzky, během kterých členové týmu referují o stavu přidělených úkolů. Na konci sprintu proběhne hodnotící schůzka.

   iii. Dohra: Integrace systému, předání kompletního produktu, akceptační testy a dokumentace.

   SCRUM je oblíbená metodika vhodná pro středně velké a menší projekty. Je kompromisem mezi všemi uvedenými. I když se na počátku projektu sepíše seznam požadavků, metodika umožňuje tento seznam později pozměňovat, zpřesňovat či vylepšovat. Tým je považován za samostatnou jednotku, jenž sama zvládne všechny fáze vývoje.

2.3 Volba metodiky pro projekt Selftester

Jelikož je projekt vyvíjen týmem o velikosti dvou osob, které jsou navíc zároveň i zadavateli, a požadavky na aplikaci nejsou specifikovány dostatečně detailně, budeme volit nějakou formu agilní metodiky. Nutno také podotknout, že popsané varianty jsou spíše modelové a nemusí se naprosto přesně dodržovat, naopak konkrétní způsob vývoje musí být vždy přizpůsoben projektu a aktuálním možnostem týmu. Náš postup se bude nejvíce podobat zásadám extrémního programování zejména kvůli schopnosti této metodiky pružně reagovat na změny ve funkčních požadavcích.

Kapitola 3

Architektura frontend aplikací

Na úvod je vhodné vymezit názvosloví, používané v této kapitole. Rozdíl mezi frontendem a backendem je popsán výše (viz 1.1.1). Frontend aplikace jsou dále označovány jako klientské aplikace anebo SPA (single-page aplikace, vysvětleno níže). Jedná se o to samé, ale zvolený název lépe sedí kontextu nebo lépe popisuje typické vlastnosti aplikace.

Frontend aplikace již lze zcela separovat od backendu. Tento přístup přináší výhody a podle některých autorů se jedná o přirozený vývoj v oblasti webových aplikací ([25]), znamená to ale množství nových problémů, zejména:

- **Uchovávání stavu aplikace**: U klasických webových aplikací (tím je myšleno u takových aplikací, které nerozlišují mezi backend a frontend částmi), je stav a veškerá data uchováván v databázi. Co je uložené v databázi je „správné“ - tím je myšleno, že pokud by došlo k nějaké kolizi v datech, to, co je v databázi, je považováno za správné. Relační i nerelační databáze jsou zajisté jedním z nejvíce komplexních nástrojů, se kterými webový vývojář běžně pracuje. Samostatná frontend aplikace potřebuje nějaký efektivní způsob, jak uchovávat a chránit globální stav dat.

- **Simulace historie prohlížeče a dalších funkcí**: Je nutno brát v potaz, že pokud se celá stránka vykresluje jen pomocí Javascriptu, běžné funkce prohlížeče (jako například tlačítko Zpět) nebudou fungovat samy o sobě správně, protože při kliknutí na odkaz ve skutečnosti nedojde k přechodu na jinou stránku a tak dále. Tyto vlastnosti je třeba zachovat, pokud chceme, aby aplikace nebyla pro běžného uživatele matoucí.

- **Optimalizace pro vyhledávače**: Pokud je obsah stránek vykreslen pouze pomocí Javascriptu, nebude správně zpracován vyhledávacími roboty a stránka nedosáhne na takové pozice ve vyhledávání, jako mají klasicky generované stránky. I když vyhledávač Google do jisté míry umí spustit skripty na straně klienta ([26]), výsledky stále nejsou opti-

mální. Navíc může aplikace cílit na jiné vyhledávače, které Javascript nevykonávají vůbec.

- **Klient musí mít povolený Javascript**

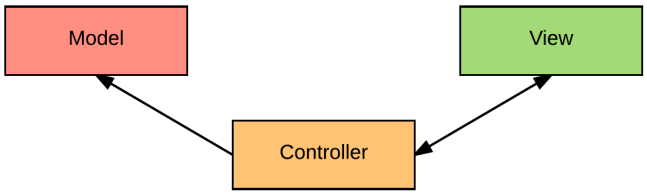
Výhody single-page aplikací [25]:

- Omezení počtu dotazů na API (a tím i na databázi)
- Zrychlení aplikace: Stahují se jen nová či aktualizovaná data a ne celý HTML dokument.
- Větší volnost pro vývojáře: a tím i lepší možnosti reakce na uživatele, zlepšení jeho zážitku a tak dále (například validace formuláře ještě před odesláním).

3.1 MVC a frontend

MVC je tradiční a velmi populární návrhový vzor využíváný mimo jiné při tvorbě webových aplikací. Je jednoduchá a zároveň efektivní ve svých principech. MVC bylo popsáno již roku 1979 a jeho primární myšlenka se týkala především rozdělení aplikační logiky a uživatelského rozhraní (tehdy grafického i textového) [27]. Je tedy pochopitelné, že od doby svého vzniku se odlišily různé modifikace MVC jako návrhového vzoru. Navíc se na základě MVC postupně vyvinuly další návrhové vzory jako MVP nebo MVVM.

Většina webových vývojářů se setkala s lehce odlišnou verzí MVC, jenž reflektuje specifika této platformy - **View** se nachází na klientovi a není možné jej notifikovat o změně **Modelu** kvůli povaze klient-server komunikace (server nemůže kontaktovat klienta, HTTP požadavek je vždy vytvořen klientem). [27] mluví o této verzi MVC jako o Ruby on Rails MVC (viz 3.1. Model je zpravidla reprezentován databází. Rozbor tradičního MVC (a jemu podobných), používaného na backendu (často se užívá i termín server-side, tedy na straně serveru), není předmětem této kapitoly.



Obrázek 3.1: Ruby on Rails (webové) MVC

Poté, co začaly vznikat samostatné klientské webové aplikace, velmi rychle následovaly i první frameworky, určené čistě pro frontend (a tedy v Javascriptu).

Tyto frameworky jako Angular nebo Backbone stavěly na MVC jako ověřené architektuře na poli webových aplikací [28].

Mezi server-side a client-side MVC je jeden hlavní rozdíl. V server-side MVC View vrstva nemůže dostávat notifikace o změnách přímo, zatímco u frontend MVC je to jeden z nejdůležitějších funkčních požadavků, díky kterému je vlastně možné vytvářet vysoce interaktivní aplikace, jenž se rozšířily v posledních letech.

Myšlenka nebyla špatná, ale reálná implementace MVC například v oblíbeném frameworku Angular (verze 1) [29] nebyla ideální. Rozdělení pravomocí mezi Model, View a Controller porušovalo Single responsibility principle (Princip jedné zodpovědnosti) - Controller často byl příliš závislý na View (nebo byl přímo jeho součástí), data v Modelech se dala modifikovat jak z Controlleru, tak z View a mohlo proto být obtížné určit, jak došlo ke změně [28]. Nutno dodat, že se nejednalo pouze o Angular, podobně fungovaly i jiné frameworky.

Největší problémy tohoto přístupu můžeme obecně shrnout: [30]:

1. **View vrstva má přímý přístup k Model vrstvě**: View může měnit údaje v Modelu bez volání Controlleru. Po uložení změn v Modelu se View překreslí. Stejného efektu (zobrazení dat) bychom ale mohli dosáhnout i tak, že zavoláme nějakou URI - tímto způsobem se do hry dostane routování a poté Controller a zobrazí ten samý výstup. Máme tak dvě různé možnosti, jak definovat kroky (či změny UI) v aplikaci, což zvyšuje složitost. V klasickém MVC se dalo View změnit pouze jedním způsobem, po routování na serveru Controller zpracoval požadavek a vrátil nová data.

2. **Roztříštěná Model vrstva**: jedna změna stavu aplikace může vyústit v update na několika různých Modelech. Navíc tato vrstva neukládá stav UI aplikace, pouze reflektuje entity v databázi. Stav UI je ale potřeba také efektivně skladovat tak, aby si každá komponenta nemusela udržovat vlastní verzi, ale mohla se spolehnout na Model vrstvu. Chybí tedy nějaké jednotné úložiště, které by fungovalo podobně jako databáze v klasickém MVC.

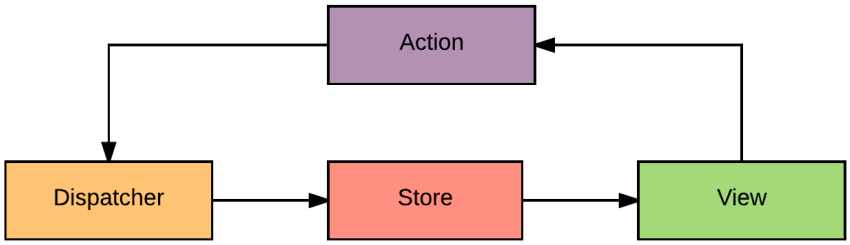
3. **Stav UI bývá uložen ve View vrstvě (například zaškrtnuté checkboxy v aktuálně vykresleném formuláři)**: tohle je jasným porušením konceptu MVC - stav má být pouze v Modelu.

Tato tematika je předmětem diskusí, uveďme si ale ilustrační příklad - jedním ze známých problémů frameworku Angular bylo, že změna dat v modelu často vyústila v kaskádu dalších změn ve stavu aplikace. Díky tomu potenciálně mohlo docházet k nekonečné rekurzi, což autoři frameworku vyřešili zavedením magické konstanty na počet zanoření [31]. To je jistě funkční řešení problému, vede to ale k otázce, zda je takový framework vhodně navržen.

3.2 Jednosměrný tok dat a Flux

Jako reakce na tyto problémy první generace frontend frameworků začaly vznikat frameworky či knihovny, jenž zavádí striktní vyžadování jednosměrného toku dat (unidirectional data flow).

Prvním známějším řešením byl architektonický vzor **Flux** [32], který vyvinul Facebook. Nebudeme zabíhat do detailů tohoto konkrétního řešení, ale na zjednodušeném nákresu Flux architektury si ukážeme základní myšlenky jednosměrného toku dat (diagram viz 3.2).



Obrázek 3.2: Architektonický vzor Flux

Jakákoliv změna vyvolaná uživatelem vytvoří Action (akci) (pomocí tzv. action creator) - standartizovaný objekt s explicitně definovaným obsahem. Je nutno vyjmenovat, jaká data se mají v rámci akce poslat. Všechny akce zachytává Dispatcher (jedná se vlastně o sběrnici - event bus) a přeposílá je dál do Store. Store je úložiště uchovávající kompletní stav aplikace, ať už se jedná o entity, jež reflektují modely z databáze nebo stav UI. Store tedy obdrží data, jež jsou součástí Action a na jejich základě změní svůj stav. Jinak než touto cestou se Store nemůže modifikovat. View komponenty mohou (ale nemusí) poslouchat změny ve Store. Pokud dojde ke změně, která je pro danou komponentu relevantní, View se překreslí. Tím se uzavírá celý cyklus.

I když se tento vzor nenazývá MVC, podle mnohých autorů ([28], [30]) se mu podobá mnohem více, než nástroje popsané v 3.1, zejména co se týče lepšího rozdělení zodpovědností. Store je analogií MVC Modelu a Dispatcher je analogií Controlleru. Veškeré změny stavu musí projít přes něj. Tento přístup do jisté míry řeší problémy popsané výše 3.1. Ačkoliv - nebo právě proto, že byl Flux prvním známým architektonickým vzorem svého typu, obsahuje několik nedostatků [30]:

- Umožňuje mít více Stores: narazíme na problém, jakmile jeden Store potřebuje dostat data z jiného.
- Funkcionalita Dispatcheru/Controlleru: chybí koncept aplikační logiky v Dispatcheru, který v Controlleru máme.

V praxi se častěji používá architektonický vzor Redux, jenž je Fluxu velmi blízký, ale vylepšuje některé jeho nedostatky a zjednodušuje použití [33]. Oba vzory se dobře používají v kombinaci s knihovnou pro vývoj UI React. Oboje je důkladněji popsáno v další kapitole.

3.3 Komponentový přístup (component-based approach) [1]

V zásadě se jedná o rozdělení prvků UI do sémanticky sdružených částí - komponent. Například je možné vytvořit komponentu, která bude představovat speciální položku ve formuláři (datepicker) a tu pak používat nejen v různých formulářích, ale i v různých aplikacích. Komponenta má zajišťovat jak prezentační stránku, tak chování v rámci UI a poskytuje API, pomocí kterého je možné ji nastavit. Může se jednat o CSS třídy nebo funkce, které se mají provolat pokud dojde k události vyvolané prohlížečem (například kliknutí na tlačítko). Komponenta může obsahovat nebo být součástí jiné komponenty, nebo obojí.

Tato myšlenka není nová a nevznikla současně s knihovnou React nebo Flux. Obecně vzato se jedná o jedno z programovacích paradigmat - zapouzdření. Podobné komponenty nebo pluginy byly dostupné například v rámci knihovny jQuery.

Knihovna React [34], sloužící pro tvorbu interaktivního UI, rozvedla komponentový přístup ještě více a také ho zpopularizovala. Za použití Reactu je možné generovat HTML čistě pomocí Javascriptu a zavádět tak nové, abstraktní tagy. Takto vytvořené komponenty mohou mít vnitřní stav, mohou se do sebe vkládat a vůbec splňují všechny vlastnosti popsané výše. React přináší mnoho dalších výhod, například při překreslení DOMu je schopný udělat nejmenší možný počet změn. Ekosystém pluginů a komponent do Reactu je velmi rozsáhlý, ačkoliv knihovna je poměrně nová [34].

CSS je nepostradatelnou součástí každé webové klientské aplikace. Obyčejné CSS neobsahuje žádné metody zapouzdření či vytváření modulů, což sebou přináší všechny možné nevýhody - především globální scope (všechny CSS třídy jsou na stejné úrovni a lze je tak přepisovat) anebo nutnost definovat složité selektory, jejichž vyhodnocování pak stránku zpomaluje. Aby CSS bylo platnou součástí vytvářených komponent, musí být rozloženo do modulů. Používají se dva přístupy:

- **BEM (Block-element-modifier)**[35]: Modularita je zajištěna pomocí pravidel pojmenovávání selektorů. Modularita tedy musí být udržována a zachovávána vývojáři.

- **CSS Modules**[36]: CSS Modules umožňují importovat CSS soubor přímo do Javascriptu. Pokud je pak nějaká z CSS tříd použita, je jí ve výsled-

ném CSS vygenerováno náhodné jméno a tím je dosaženo potřebného zapouzdření.

3.4 Závěr

Komponentový přístup a architektura s jednosměrným tokem dat se často používají dohromady. **View** ve Fluxu či Reduxu je většinou vytvořeno z komponent, i když ani jedno to druhé přímo nevyžaduje. Komponenta, která se stará o zobrazení například uživatelského profilu, může naslouchat jen těm změnám ve **Store**, které se týkají uživatele a naopak může být jediná oprávněná k tomu tyto změny vytvářet (tedy odesílat **Action**). Oba koncepty se dobře doplňují a je praktické je používat dohromady.

Kapitola 4

Návrh

Tato kapitola se nejprve zabývá problematikou návrhu uživatelského rozhraní pro webové aplikace a dále pak návrhem frontend a backend aplikace.

4.1 Uživatelské rozhraní

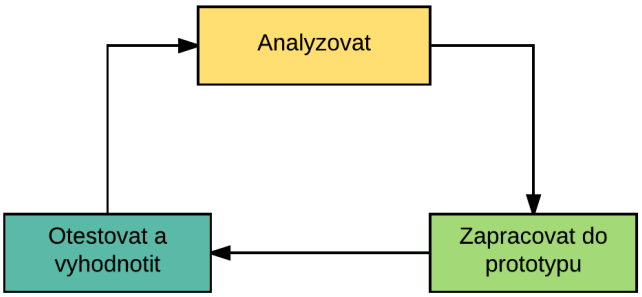
Návrh uživatelských rozhraní a i celý obor Human-computer interaction (interakce člověk - počítač) se dostávaly do pozornosti vývojářů postupně. Mělo to několik příčin, mezi nejvýznamnější patří zvyšování výpočetních kapacit zařízení (v 80tých letech šlo o stolní počítače, podobný boom UI nastal po roce 2000 s rozmachem chytrých telefonů), dále masové rozšíření zařízení mezi veřejnost a s tím související větší konkurence mezi aplikacemi [37]. Podle [38] v dnešních aplikacích UI typicky spotřebuje až 50 % procesorového času a kódu.

Nejdůležitějším konceptem je tzv. User-centered design - návrh uživatelského rozhraní, které musí na první místo klást samotného uživatele a vytvářet design na míru jeho potřebám a očekáváním ([38]). Testování poté probíhá s realnými uživateli.

4.1.1 Obecný postup

Proces návrhu uživatelského rozhraní se obvykle skládá z následujících kroků. Hlavním cílem je vytvořit dva prototypy UI, první, takzvaně **lo-fi prototyp**, vzniká rychleji, jeho cílem je určit základní rozvržení funkcionality na obrazovkách. Tento prototyp se nazývá mock-up nebo wireframe. Poté, co je tento návrh dokončen a otestován, vytváří se **hi-fi prototyp**, který je interaktivní a funguje už na cílové platformě. Jeho vytvoření trvá řádově déle.

Tyto prototypy se obvykle vyvíjí za použití agilních metodik (viz 2), velmi důležitá je zejména spolupráce s klientem tak, aby designér co nejlépe pochopil produkt či aplikaci. Při návrhu prototypů dochází k iteraci mezi fázemi viz 4.1:



Obrázek 4.1: Jedna iterace procesu návrhu UI

4.1.1.1 Klíčové kroky v procesu návrhu UI [39]

Kroky by měly probíhat v uvedeném pořadí - není například vhodné vytvářet hi-fi prototyp bez kompletní analýzy požadavků a tak podobně. Naopak je možné některé kroky přidat (testování použitelnosti na lo-fi prototypu) nebo opakovat.

1. **Lo-fi fáze**

   a) Definování produktu/aplikace: Co je cílový produkt, co dělá a co není.

   b) Use cases (případy užití): Všechny způsoby a možnosti, jakými bude uživatel produkt používat. Tým se snaží různými technikami (například brainstorming) zjistit, co by uživatel mohl chtít a potřebovat.

   c) Seznam požadavků, graf požadavků: vzniká na základě use cases. Pomáhá vidět produkt z pohledu uživatele. Musí být kompletní a obsahovat i zdánlivě jasné (návrat na domovskou stránku) a zbytečné požadavky. Požadavky dále analyzujeme, nastavujeme priority a tak podobně.

   d) Prototyp: vzniká na základě analýzy požadavků. Vytváří se rychle a prochází častými změnami. Na jeho výrobu existují různé nástroje (například Balsamiq [40]), ale postačí i tužka a papír. Prototyp může, ale nemusí být interaktivní.

   e) Otestování použitelnosti lo-fi prototypu: Vybranou metodou testování bez skutečných uživatelů (test tedy provádí tzv. expert - viz 4.1.1.2).

   f) poznámka: Poslední dva body se mohou opakovat, dokud návrh není dostatečně kvalitní.

2. **Hi-fi prototyp**: Vizuální podoba i interakce by měly být stejné jako ve výsledné kompletní aplikaci. Jedná se o model UI, který běží na cílové platformě. Musí obsahovat všechny důležité prvky UI, ale nemusí být zcela kompletní [39]. Pokud produkt bude distribuovaný na více platforem, musí se udělat prototyp na každou z nich.

3. **Testování použitelnosti**: reální uživatelé pracují s vytvořeným hi-fi prototypem. Výsledky testů jsou zapracovány do prototypu. Opět může být provedeno ve více iteracích (viz 4.1.1.4).

4. Implementace UI do konečné verze produktu.

4.1.1.2 Testování bez uživatelů [39]

Jedná se o způsoby testování prototypů, jenž jsou prováděny odborníkem na UI. Nejčastěji se provádí tak, že prototyp je ohodnocen pomocí nějaké heuristiky, pokud se s pravidly popsanými v heuristice rozchází, je to považováno za chybu. Tato metoda je rychlejší a levnější než testování s uživateli, ale nehodí se například pro atypická uživatelská rozhraní - může se stát, že pro ně nebudeme mít vhodnout heuristiku. Odborník, co test provádí, by neměl být tentýž, co aplikaci navrhoval. 5-6 odborníků odhalí okolo 75 % problémů [39].

UI naší aplikace bude testováno pomocí Nielsenovy heuristiké analýzy.

4.1.1.3 Nielsenova heuristika [41]

1. Visibility of system status: Uživatel musí být neustále informován o tom, že/na čem systém pracuje. Pokud nějaká akce trvá déle než uživatel očekává (Podle [42] je horní hranice, dokdy má uživatel s počítačem slitování, až 10 vteřin), musí o tom aplikace informovat například pomocí progress baru.

2. Match between system and the real world: Aplikace používá „jazyk“ uživatele, neobsahuje přehnaně technické termíny (pokud nejsou součástí slovníku cílového uživatele), zachovává konvence a metafory z reálného světa.

3. User control and freedom: Aplikace nesmí uživatele stresovat, uživatel se nesmí z jakéhokoliv důvodu bát vykonat nějakou akci. Akce by měly jít zvrátit, popřípadě by UI mělo zobrazit varovné hlášky. Mělo by být možné provést krok zpět i krok vpřed.

4. Consistency and standards: Termíny použité v aplikaci by měly být konzistentní a srozumitelné. Aplikace by měla být postavena z obvyklých komponent (pro danou platformu). U webových aplikací je rozptyl docela široký, toto pravidlo je více striktní například pro mobilní aplikace.

5. Error prevention: jedná se zpravidla o vyplňování formulářů, uživateli je zabráněno nebo je varován ještě před odesláním, že zadal špatnou hodnotu.

6. Recognition rather than recall: nízké nároky na uživatelovu paměť, systém je přehledný, uživatel ví nebo má na očích, kde se nachází a jak má dál postupovat.

7. Flexibility and efficiency of use: nabízí základní a pokročilé funkce vhodně oddělené, ale zároveň dostupné (jinými slovy aplikace uspokojí náročné uživatele, ale nový uživatel se v ní neztratí).

8. Aesthetic and minimalist design: Pokud informace není relevantní, neměla by být vůbec zobrazena. Grafické prvky by neměly ztěžovat práci.

9. Help users recognize, diagnose, and recover from errors: jde zejména o srozumitelné chybové hlášky a o správné nasměrování uživatele k nápravě.

10. Help and documentation: nápověda, vysvětlivky k systému.

4.1.1.4 Použitelnost a testování použitelnosti

Použitelnost, anglicky **Usability**, je kvalitativní vlastnost uživatelského rozhraní, která určuje, jak snadné je toto rozhraní užívat, zároveň se tak nazývá i soubor metod, jenž mohou použitelnost jako vlastnost změřit a vylepšit [43]. Podle [43] se míra použitelnosti skládá z těchto základních součástí:

1. Naučitelnost (v originále learnability): Jak snadné je pro uživatele vykonat základní úkoly při prvním setkání s aplikací?

2. Efektivita (v originále efficiency): Jak rychle uživatelé dokáží pracovat s aplikací poté, co se s ní už naučili?

3. Zapamatovatelnost (v originále memorability): Jak se uživatelé opětovně zorientují v práci s aplikací poté, co ji nějakou dobu nepoužívali?

4. Chybovost (v originále errors): Kolik chyb uživatelé dělají v práci s aplikací, jak závažné jsou, dokážou chyby napravit?

5. Spokojenost (v originále learnability): Jak příjemné uživatelům je pracovat s aplikací?

Pokud chceme použitelnost zlepšit, nejprve musíme **otestovat** stávající řešení. Tím odhalíme závady našeho UI. Obecně platí ([38], že problémy zjištěné při testování s reálnými uživateli bývají jiné, než při testování provedeném experty (viz 4.1.1.2). Dle postupu navrženého výše, bude první testování na

lo-fi prototypu provedeno bez uživatelů a druhé testování hi-fi prototypu právě s uživateli.

Testování použitelnosti s uživateli probíhá následovně [44]:

- Podmínky: test by měl být proveden v prostředí, jenž je testovanému pohodlné nebo v laboratoři k tomu určené. Důležité je, aby testovaný znal a uměl zacházet se zařízením, na němž provádí úkoly (například na vlastním PC či telefonu).

- Úkoly: testovaný dostane sadu úkolů - mělo by se jednat o typické činnosti vykonávané pomocí testované aplikace (například založit uživatelský účet). Formulace úkolů by neměla uživatele navádět k tomu, **jak** co vykonat, měla by být spíše obecná - cílem testů je zjistit, jak by si uživatel poradil v reálné situaci.

- Pozorování: Testovaný by ideálně měl mít k práci klid, neměl by se cítit pod tlakem a neměl by se bát vyslovit svůj názor - aby například neurazil tvůrce UI.

- Záznam: činnost testovaného by měla být zaznamenána pro další vyhodnocení. Pro naše účely postačí nahrávání obrazovky.

Podobně jako s testováním experty stačí otestovat cca pět osob, aby byly zjištěny ty největší chyby v UI [43].

Dále je důležité vybrat vhodné subjekty k testování - ideálně by to měli být lidé z cílové skupiny produktu či aplikace. V této souvislosti se často vytváří tzv. **persony** - reprezentanti osob z cílových skupin produktu. Persony jsou vytvořené předem na základě demografické analýzy. Pro uživatelské testy se poté vybírají lidé, kteří se personám co nejvíce podobají. Tak je otestováno méně lidí, ale přesnost by měla zůstat zachována [44].

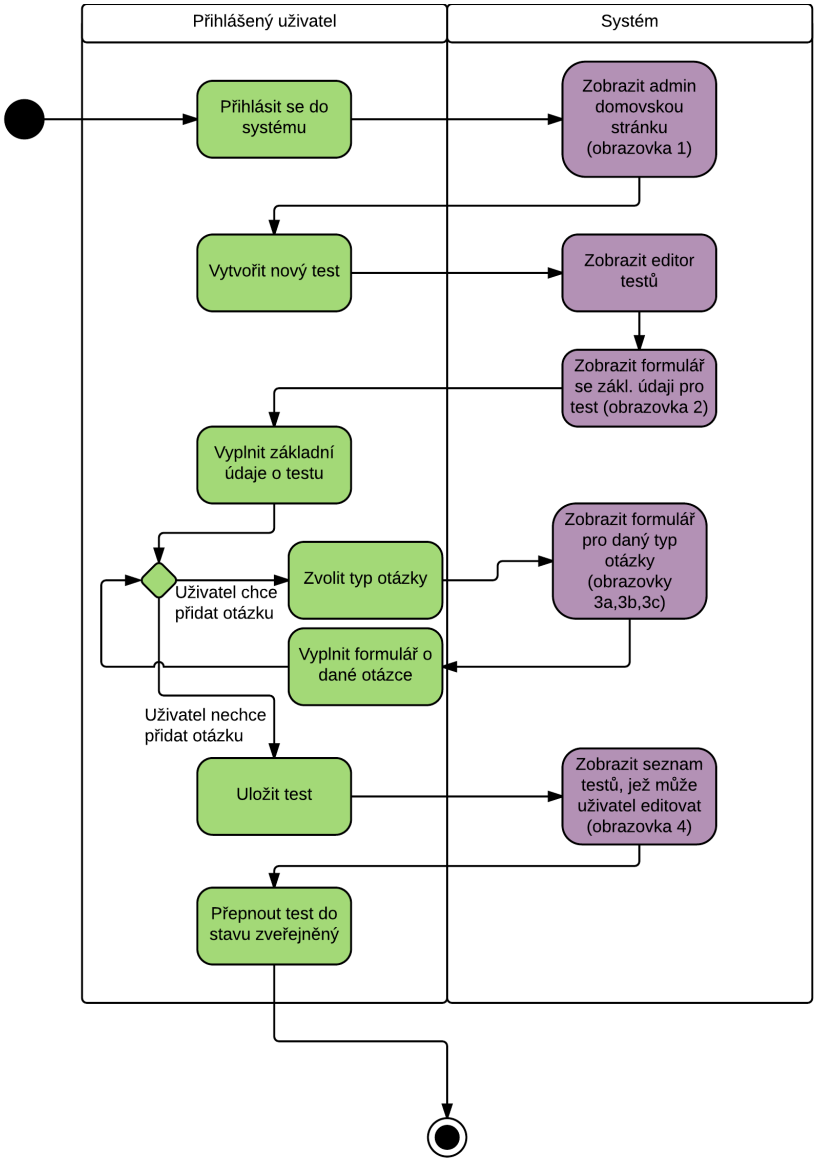
Tuto techniku nebudeme při uživatelských testech využívat. Cílová skupina pro účely této práce je pouze jedna - **studenti** (středních a vysokých škol).

4.1.2 Případy použití

Případy použití jsou převzaty z kapitoly Analýza (1.4). Dva procesně komplikovanější případy použití jsou rozkresleny pomocí diagramu aktivit.

4.1.2.1 Případ použití - Vytvořit veřejný test UC13

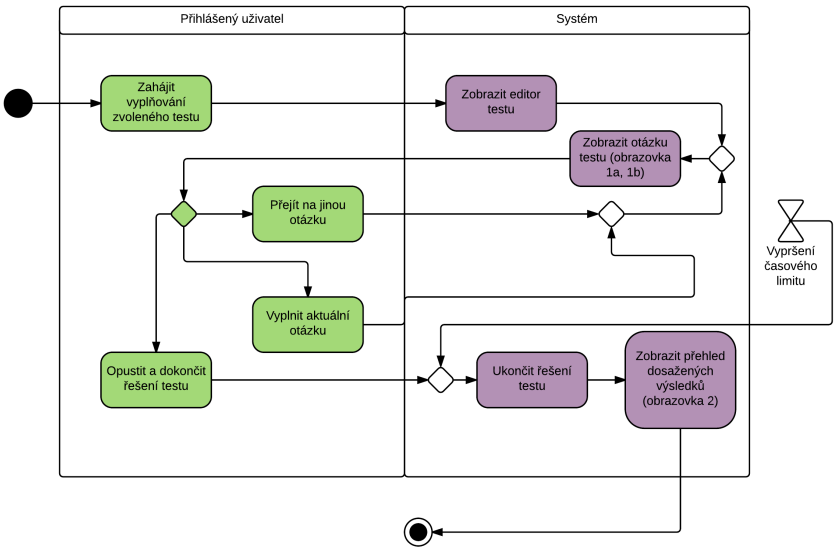
Přiložený diagram znázorňuje pouze hlavní scénář tohoto případu použití, kdy uživatel nechce nic mazat. Alternativní scénáře by zahrnovaly zrušení vytváření testu nebo konkrétní otázky, jelikož k těmto akcím může dojít ve většině aktivit, diagram by se stal velmi nepřehledným. Příslušné obrazovky jsou dostupné v příloze C.



Obrázek 4.2: Diagram UC Vytvořit veřejný test

4.1.2.2 Případ použití - Vyplnit test UC9

Příslušné obrazovky jsou dostupné v příloze B.



Obrázek 4.3: Diagram UC vyplnit test

4.1.3 Seznam požadavků UI pro Selftester a rozvržení do obrazovek

Seznam požadavků doplňuje a rozšiřuje případy použití a je seřazen podle obrazovek či sekcí, do kterých bude daný požadavek spadat. Jedná se vlastně o souhrn veškerých možných interakcí z pohledu uživatele. I když by se požadavky daly rozepsat do ještě detailnějších (například vyplň specifickou položku formuláře při přihlašování), z pohledu uživatele by už neměly smysl a proto nejsou takto uvedené.

4.1.3.1 Akce dostupné pro neregistrované uživatele

1. **Obecné funkce (dostupné v hlavičce):**
   a) Registruj se
   b) Přihlaš se (login)
   c) Odhlaš se (logout)
2. **Homepage - vyhledávání a procházení testů k vyplnění:**
   a) Seřaď testy podle abecedy/hodnocení/času poslední úpravy
   b) Filtruj testy podle nabízených populárních tagů
   c) Filtruj testy podle uživatelského jména autora
   d) Vyhledej testy podle klíčových slov

e) Ohodnoť test

f) Zobraz detail testu

g) Přejdi na vyplňování testu

3. **Detail testu**

a) Ohodnoť test

b) Přidej komentář

c) Odstraň komentář

d) Přejdi na vyplňování testu

4. **Vyplňování testu a průchod výsledky**

a) Přejdi na další/předchozí otázku v pořadí

b) Přejdi na libovolnou otázku podle čísla v pořadí

c) Zadej odpověď

d) Opakuj test

e) Nahlaš nesprávnou otázku

5. **Vyplňování testu a průchod výsledky**

a) Přejdi na další/předchozí otázku v pořadí

b) Přejdi na libovolnou otázku podle čísla v pořadí

c) Zadej odpověď

d) Opakuj test

e) Nahlaš nesprávnou otázku

4.1.3.2 Akce pouze pro registrované uživatele - administrace

Mnoho požadavků je stejných nebo velmi podobných těm uvedeným výše. Zde jsou uvedeny pouze ty nad rámec předchozího seznamu.

1. **Administrace - Homepage**

a) Zobraz přehled posledních vyplněných testů

b) Zobraz přehled oblíbených testů

c) Přejdi na vytvoření testu

2. **Administrace - Historie uživatele**

a) Zobraz seznam vyplněných testů

b) Zobraz detail vyplněného testu (přehled výsledků)

3. **Administrace - Správa uživatelského účtu**

    a) Edituj zadané osobní údaje

4. **Administrace - Správa mých testů**

    a) Změň stav testu z veřejného na neveřejný a zpět

    b) Přidej dalšího editora testu

    c) Odeber editora testu

    d) Přidej tag testu

    e) Odeber tag testu

5. **Administrace - Editace testu**

    a) Edituj základní údaje o testu

    b) Přidej otázku do testu

    c) Odeber otázku z testu

    d) Vytvoř k otázce sadu odpovědí

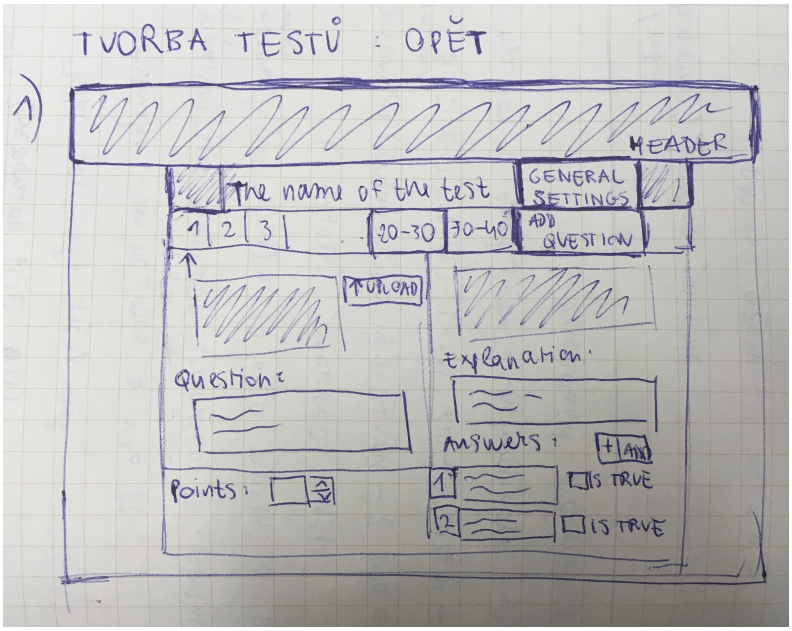
    e) Nahraj k testu ilustrační obrázek

4.1.4 Proces prototypování UI Selftesteru

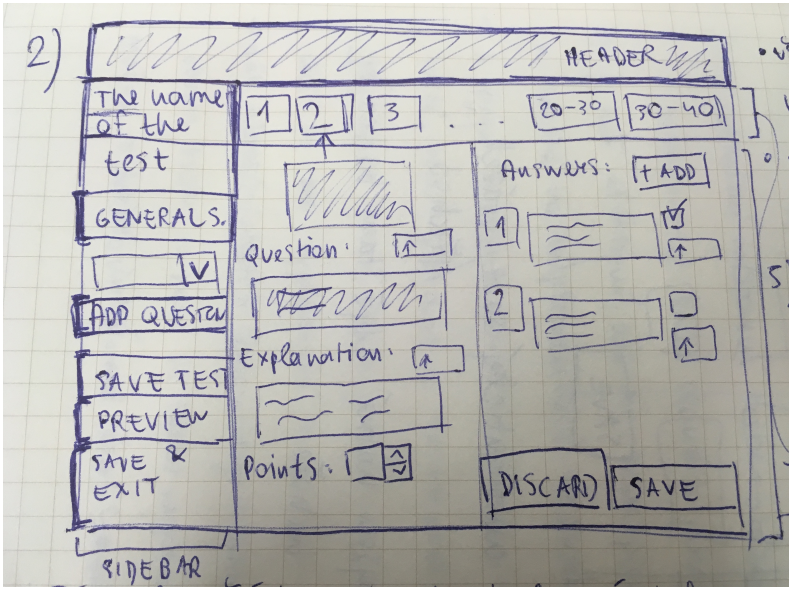
Lo-fi prototypy vznikaly v několika iteracích, nejprve na papíře a poté v proklikávací podobě v již zmíněném nástroji Balsamiq [40]. Kompletní seznam vytvořených prototypů je k dispozici na přiloženém mediu E. Zde bude uvedeno jen několik vybraných pro demonstrování vývoje prototypu.

Detailní **Hi-fi prototyp** v tento moment teprve vzniká. Vzhledem ke zvolené platformě (viz níže) bude prototyp až na malé úpravy totožný s kompletní frontend aplikací - jen se do něj připojí API s reálnými daty. To je jedna z výhod tohoto přístupu - viz Důvody pro rozdělení API a webové aplikace 4.2.1.

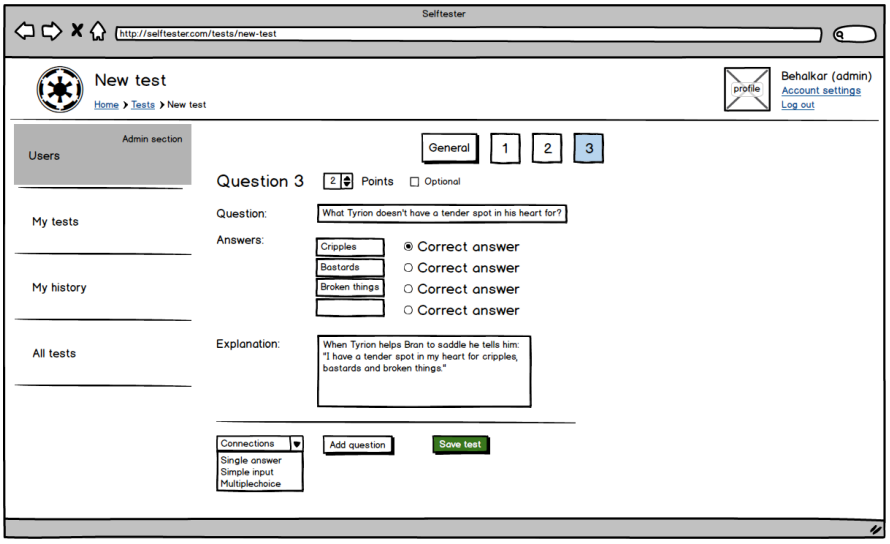
4.1.4.1 Návrhy pro obrazovku s editorem testů



Obrázek 4.4: Kreslené návrhy - První varianta



Obrázek 4.5: Kreslené návrhy - Druhá varianta - odděleny základní funkce a panel s otázkami



Obrázek 4.6: Detailnější obrazovka vytvořena pomocí nástroje Balsamiq

4.1.5 Testování Lo-fi prototypu

Testování bylo vykonáno na proklikávacím prototypu a bylo provedeno pomocí Nielsenovy heuristické analýzy (4.1.1.3). Každá nalezená chyba byla ohodnocena stupněm podle závažnosti.

Body	Popis
1	Drobná chyba, neovlivní počínání uživatele
2	Chybu uživatel vnímá, zpomalí ho, ale je schopen pokračovat.
3	Chyba uživatele výrazně zpomalí. (na více jak 10 vteřin)
4	Chyba uživateli znemožňuje dokončit činnost.

Tabulka 4.1: Stupnice závažnosti chyb podle heuristické analýzy

4.1.5.1 Výsledky testování

1. Visibility of system status:
   - **Problémy**
   - **3 body**: Generování testu k vyplnění může zabrat delší čas, v návrhu neexistuje obrazovka, která by obsahovala příslušný progress bar, hlášku s informací o tom, že systém pracuje atd.
   - **Vylepšení**
   - Návrh obsahuje drobečkovou navigaci, což uživateli pomáhá orientovat se na stránce.

- Pokud je výpis testů filtrován, je tato informace vhodně zobrazena, je také zřejmé, jak filtraci zrušit.

2. Match between system and the real world:

   - **Problémy**
   - **1 bod**: Termín „znalostní test“ nemusí každému uživateli připadat v kontextu přesný.

3. User control and freedom:

   - **Problémy**
   - **3 body**: V editaci testu chybí možnost vymazat z testové sady pouze jednu otázku. Kdyby uživatel chtěl tuto akci provést, musel by smazat celý test a znovu vytvořit.
   - **Vylepšení**
   - „Nebezpečné“ uživatelské akce jsou opatřeny varovnou hláškou a navíc je i možné je zvrátit.
   - Z každého místa aplikace je možné vrátit se na start díky drobečkové navigaci a odkazu v hlavičce.
   - Při vyplňování testu je možno editovat odpověd k jakékoliv otázce v každé fázi vyplňování. Rychlá navigace v záhlaví usnadňuje přechody mezi otázkami.

4. Consistency and standards:

   - **Vylepšení**
   - Aplikace používá konzistentní pojmy.
   - Aplikace zavádí obecně užívané a známé funkce (například ohodnotit test hvězdičkami - čím více, tím lepší hodnocení).

5. Error prevention:

   - **Vylepšení**
   - Validace formulářů se zadanými výsledky při vyplňování testu proběhne ještě před odesláním - pokud nepozorný uživatel například omylem vynechal otázku, je na to upozorněn.

6. Recognition rather than recall:

   - **Problémy**
   - **1 bod**: uživatel by mohl být výrazněji upozorněn, když mu například dochází čas na vyplnění testu.
   - **Vylepšení**

- Při vyplňování testu je uživatel informován o tom, v jaké části se nachází (vidí, kolik otázek mu chybí do konce). Zároveň obrazovka neobsahuje žádné další informace, které by ho mohly rušit nebo zapříčinit odchod.

7. Flexibility and efficiency of use:
   - **Problémy**
   - **2 bod**: Komplexní vyhledávání, které nabízí tři typy výsledků, může být pro nové uživatele zpočátku nesrozumitelné.
   - **2 bod**: Není zřejmé, jak zrušit filtr.
8. Help users recognize, diagnose, and recover from errors: Všechny zprávy v prototypu jsou srozumitelné, ale jsou bohužel v angličtině - v této fázi vývoje jsme počítali s anglickou aplikací.
9. Help and documentation: Není obsaženo v prototypu.

4.1.5.2 Závěr

V první verzi hi-fi prototypu budou odstraněny chyby se závažností 3 a 4 body dle stupnice.

4.1.6 Grafický design

Pro úplnost uvedeme ještě základní návrh grafického designu aplikace, jenž vznikal v nástroji Sketch. Cílem tohoto designu je pouze navrhnout základní vizuální podobu stránky, barevnost, použitá písma a rozvržení některých prvků. Grafický návrh respektuje a vznikal na základě lo-fi prototypu.

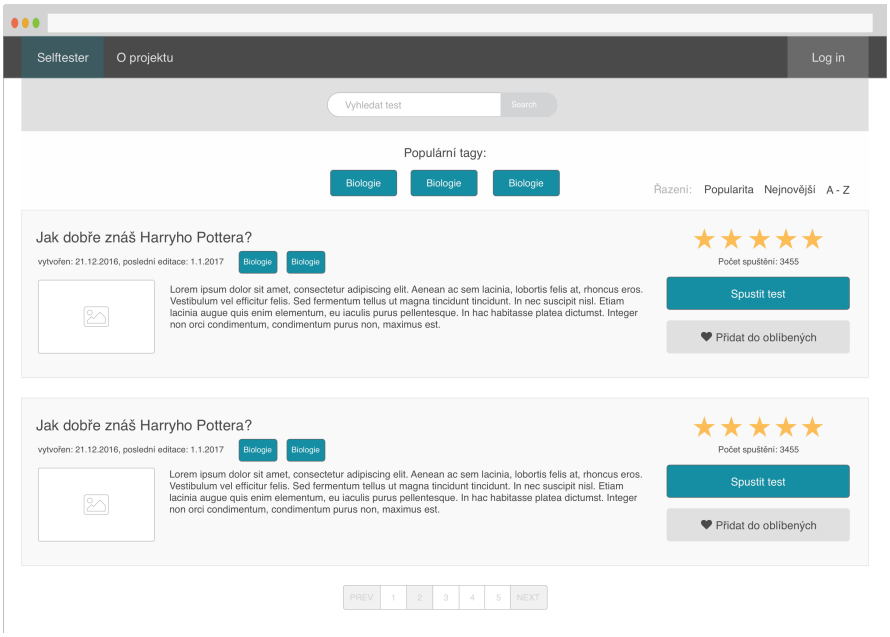
Další návrhy se nachází v příloze D.

4.1.7 Závěr

V této podkapitole byl nejprve navržen a popsán jeden z možných postupů návrhu uživatelského rozhraní. Poté byl zdokumentován samotný proces návrhu UI frontend aplikace Selftester. V rámci této kapitoly ale nebyly splněny všechny kroky zvoleného postupu - vzhledem k tomu, že v tomto případě je hi-fi prototyp prakticky totéž co výsledná aplikace, jeho vývoj a testování se bude rozebírat detailně v následujících kapitolách.

4.2 Webová (frontend) aplikace

Jak již bylo nastíněno v úvodu a v kapitole Analýza, v rámci projektu Selftester budujeme odděleně webovou aplikace a backend, tedy API, které obsluhuje i mobilní aplikaci.



Obrázek 4.7: Úvodní stránka

4.2.1 Důvody rozdělení API a webové aplikace

Nejvýznamnějším důvodem pro rozdělení je požadavek obsloužit jak web, tak mobilní aplikaci. Dříve bylo nutné zasílat mobilním klientům data v jednom formátu (JSON) a pro webové generovat HTML pomocí nějakého šablonovacího systému. Zasílání čistého HTML se lze vyhnout díky technologii AJAX a tím oba klienty - jak mobilní, tak webový - dát na stejnou úroveň a nechat je používat tytéž metody. To je pochopitelně mnohem výhodnější než řešit obě varianty odděleně.

Dalšími výhodami jsou oddělení zodpovědnosti (separation of concerns) - na API a frontend aplikaci teď mohou pracovat dva týmy vývojářů s tím, že každý se může lépe soustředit na svou část povinností, a snadnější testování, budování a nasazení obou aplikací - výhodou je to, že oba procesy mohou probíhat odděleně a nezávisle na sobě [25].

Nutno podotknout, že popsaný přístup se hodí pro komplexnější aplikace co do interakce s uživatelem. Je velmi často využíván v kontextu takzvaných SPA (single page applications, tedy aplikace, které jsou plně obsluhovány přes Javascript a nedochází u nich ke klasickému překreslování celé stránky).

4.2.2 Zvolené technologie a knihovny

Klientská webová aplikace bude vytvořena v programovacím jazyce Javascript. Aplikační kód včetně HTML šablony a CSS stylů je konvertován a minifikován do výsledné podoby pomocí nástroje Webpack [45]. Webpack umožňuje

připravit pro použití v aplikaci všechny potřebné soubory (tzv. assets) - umí vkládat fonty, pracovat s obrázky a tak dále. Jedná se opravdu o velmi mocný nástroj.

K rychlejšímu a pohodlnějšímu vývoji aplikace slouží další dva nástroje - Babel a statický linter (Eslint). Babel je transpiler - umí transformovat kód podle zadaných pravidel. Tím umožňuje vývojáři používat novější konstrukty jazyka Javascript (ES6, ES7) a výrazně mu tak usnadňuje práci. Eslint je velmi rozšířený zejména v ekosystému Javascriptu a to proto, že Javascript je jazyk dynamicky typovaný a svému uživateli povoluje prakticky cokoliv. Jedná se vlastně o okamžitou statickou analýzu kódu podle nastavených pravidel (existují veřejně dostupné soubory pravidel například Airbnb eslint [46]). Pravidlo například může kontrolovat, zda jsou operátory porovnání použity korektně nebo zda se správně pracuje s callback funkcemi. Pravidlo může varovat vývojáře nebo rovnou zakázat nekorektní použití a poté nedojde k transpilaci a kód s chybou se nedá použít.

Komponenty uživatelského rozhraní jsou vybudovány pomocí knihovny React (viz 3.3). Stav klientské aplikace je ošetřován za použití knihovny Redux.

Celá aplikace je zálohovaná pomocí verzovacího systému Git.

4.2.2.1 Redux

Redux je Javascriptová knihovna, která implementuje „předvídatelný“ kontejner, sloužící k uchovávání stavu aplikace. Aktuálně se jedná o jednu z nejpoužívanějších knihoven svého druhu. Do velké míry vychází z architektonického vzoru Flux (viz 3.2), v několika podstatných věcech se ale liší. Zavádí některé principy funkcionálního programování a celkově se snaží dosáhnout lepší předvídatelnosti co se změn dat týče a usnadnit debugování a testování aplikace. Stejně jako u Fluxu, Store je možné měnit pouze na základě nějaké Action.

Hlavní principy Reduxu: [47]

1. **Existuje pouze jediný zdroj pravdy**: Tím je myšleno, že existuje pouze jediný **Store**. Flux připouští existenci více Stores.

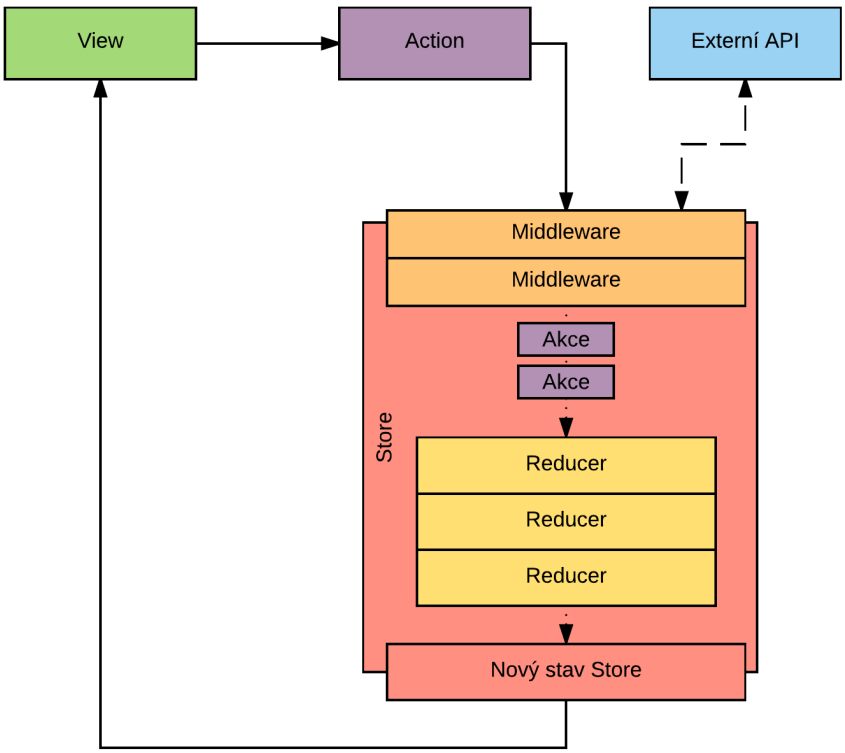
2. **Store nelze editovat**: Store je immutable - tedy neměnný. Pokud na tomto objektu chceme provést nějaké změny, musíme ho zkopírovat. Jedná se o jeden z principů, jenž se často využívá ve funkcionálním programovaní. Hlavní výhodou je to, že při porovnání dvou objektů stačí porovnat jejich reference.

3. **Změny ve Store provádí pure (čisté) funkce**: Pure funkce jsou takové funkce, které při svém běhu nepracují s žádnými proměnnými, které existují mimo tuto funkci (pouze tedy se vstupními parametry). Tím pádem můžeme mít jistotu, že taková funkce není ovlivněna a ani

nemění kontext, ve kterém běží a pro stejný vstup vrátí vždy stejný výstup. Použití takových funkcí zlepšuje předvídatelnost chování aplikace a významně usnadňuje testování.

Poznámka: díky těmto vlastnostem Reduxu dosahují nástroje na debugování na poli frontendu zcela nevídaných rozměrů - změny stavu aplikace je možné krokovat dozadu a tak dále. Existují i další důvody, proč se Redux prosadil přes prvotní implementace Fluxu. Jeho autoři detailně navrhli a popsali workflow, což spoustě vývojářů usnadnilo pochopení nového typu jednosměrných UI architektur [48] (viz 3.2). Navíc autoři jasně definovali, kde a jak má dojít k zapojení pluginů třetích stran (například API) - takzvaných vedlejších efektů [48]. Podobný koncept ve Fluxu chyběl. Tím pádem došlo k prudkému rozvoji knihoven a modulů, určených čistě pro Redux, které vývojářům dále usnadňují práci. Klientská aplikace Selftester například používá knihovnu redux-sagas, jenž pomáhá s koordinací požadavků na externí API a s množstvím dalších úkonů. V rámci této knihovny je vlastně implementovaná veškerá aplikační logika.

Popis toku dat v Reduxu 4.8



Obrázek 4.8: Diagram Redux architektury

1. View na základě různých událostí generuje explicitní Akce.

2. Akce mohou být detekovány ještě před zpracováním ve Store různými Middleware funkcemi. Middleware funkce může na základě vstupních dat volat různé vedlejší efekty (například uložení dat do API), může poté odeslat vlastní Akce (například o tom, že požadavek na API vrátil chybu) nemůže ale zastavit propagaci Akce.

3. Reducery jsou již zmíněné pure funkce, které mění data, uložená ve Store. Nic jiného než Reducer nemůže data ve Store změnit.

4. Změnám ve Store naslouchají komponenty z View vrstvy a podle potřeby se překreslují.

4.2.3 Chytré a hloupé komponenty [49]

React a Redux je nutno vzájemně explicitně propojit. To probíhá tak, že se k některým vybraným React komponentám přiřadí část Store, o který se mají zajímat a seznam Akcí, které mohou odeslat. Takto je možné vymezit dva druhy React komponent:

- Chytré komponenty (kontejnery): Tyto komponenty přímo pracují s Reduxem. Samy neobsahují žádné prvky UI, jen se starají o to, aby v nich vnořené komponenty dostávaly potřebná a aktuální data a koordinují odesílání akcí na základě událostí (od uživatele či programatických).

- Hloupé (prezentační) komponenty: Tyto komponenty se starají o zobrazení UI prvků, textu a tak dále. Od svých rodičovských komponent dostávají data a také funkce, které se mají provolat jako reakce na různé UI události (handlery - například handler události kliknutí na tlačítko).

4.3 Datový model

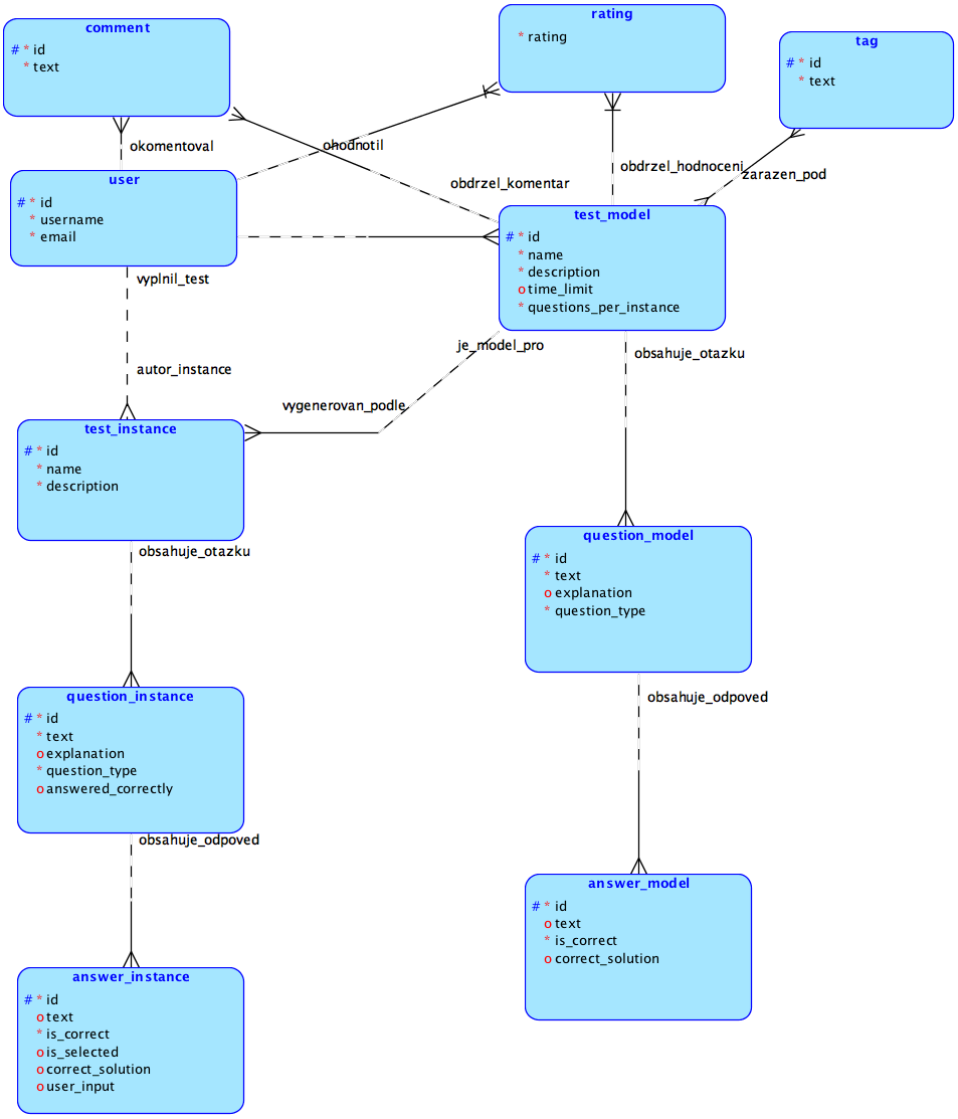
V této kapitole je popsán konceptuální datový model. Tento model neobsahuje datové typy sloupců nebo cizí klíče, jeho cílem je popsat entity a vztahy mezi nimi. Dále je vysvětlen účel jednotlivých entit a některých relací, zejména těch, co vytváří smyčky.

4.3.1 Popis entit

V konceptuálním modelu 4.9 jsou zavedeny tyto entity:

1. `user`: reprezentuje registrovaného uživatele

2. `test_model`: Předloha pro všechny testové instance, lze s ní pracovat v editoru, reprezentuje obecný test.



Obrázek 4.9: Konceptuální model databáze

3. `question_model`: Obdobně jako test_model - editovatelná předloha pro testovou otázku. Test_model může obsahovat více question_model, otázka je přiřazena vždy právě k jednomu test_modelu.

4. `answer_model`: Obdobně jako question_model a test_model - jedna otázka může obsahovat více odpovědí.

5. `test_instance`: Konkrétní test, který je uživateli vygenerován z modelových otázek. Nedá se editovat, je možné pouze vyplnit odpovědi.

6. `question_instance`: Kopíruje obsah question_model, dále obsahuje sloupec pro výsledek answered_correctly.
7. `answer_instance`: Kopíruje obsah answer_model, dále obsahuje sloupce pro zaznamenání toho, co přesně uživatel vyplnil - user_input a is_selected.
8. `comment`: Komentář k danému testu od daného uživatele. Uživatel může komentovat různé testy a to i opakovaně.
9. `tag`(štítek): Test může mít více tagů, tag může náležet k více testům (minimálně ale k jednomu).
10. `rating`(hodnocení): Registrovaný uživatel může test ohodnotit - pouze jednou. Proto je použita identifikační závislost jak od testu, tak od uživatele.

4.3.2 Popis smyček

1. `user` - `comment` - `test_model`: Test může komentovat i uživatel, který není autorem.
2. `user` - `rating` - `test_model`: Test může hodnotit i uživatel, který není autorem.
3. `user` - `test_instance` - `test_model`: Testovací instanci si může vytvořit i uživatel, který není autorem. Tato vazba je nepovinná, protože testy můžou vyplňovat i neregistrovaní návštěvníci aplikace.

4.3.3 Rozdělení testu na model a instanci

Datový model musí umožňovat splnění obou následujících požadavků:

- Test se dá kdykoliv editovat, dají se do něj přidávat či odebírat otázky.
- Uživatel může procházet historii vlastních již vyplněných testů.

Je tedy jasné, že test, ke kterému uživatel už má nějaké výsledky, se mu nesmí měnit pod rukama. Výsledky a otázky by nemusely dávat smysl, když by autor testu například editoval odpovědi a tak dále. Navíc zodpovězené otázky nemusí v aktuální podobě testu už vůbec existovat. Je nutné zavést nějakou (potenciální) redundanci a pamatovat si test přesně v tom stavu, v jakém ho uživatel vygeneroval a vyplnil. Bylo zvoleno jednoduché řešení, pokud chce uživatel vyplnit test, aktuální údaje z modelu se zkopírují do instance a tím se vyřeší možné nepřesnosti v datech. Nebude se kopírovat úplně celý model, ale jen ty otázky, které se do instance náhodně vybraly (výběr zajišťuje aplikační vrstva). Jedná se o čisté a jednoduché řešení, které ale může vést k velké paměťové zátěži. Pro účely této práce je však zcela dostačující.

4.3.4 Různé typy otázek

Jak je vidět na diagramu, model i instance otázky obsahují pole `question_type`. Otázka tedy může být různých typů, jak je uvedeno ve funkčních požadavcích. V databázi je tento požadavek vyřešen tak, že entita `answer_model/instance` zahrnuje všechny možné varianty. Jediný povinný sloupec je boolean `is_correct`, který vyjadřuje, zda je daná odpověď správně nebo ne. Poté je v případě textových odpovědí doplněna o `correct_solution`, které se porovná se vstupem od uživatele nebo v případě otázek typu vyber jednu/více správně o pole `text` - znění odpovědi, které se uživateli zobrazí při vyplňování. Entita `answer_instance` podobně reflektuje i rozdílné typy vstupů od uživatele. Pokud otázka vyžadovala textový vstup, ten se uloží do `user_input`, pokud se odpovědi jen zatrhávaly, je tato informace uložena v `is_selected`.

Z toho vyplývá, že mnoho sloupců nemá omezení `NOT NULL`. To může při implementaci vést k různým chybám, nicméně je to nutné i z důvodu, že je třeba uživateli umožnit, aby celý test (nebo jeho část) nechal prázdný. Pokud by „odpověď nevím“ byla zakázaná, lze předpokládat, že by tím utrpěla použitelnost aplikace.

4.3.5 Vylepšení - pohledy

V rámci dalšího vývoje aplikace bude vhodné vytvořit několik pohledů s předpočítanými daty, které budou agregovat průměrné výsledky uživatele, průměrné výsledky, kterých dosahují uživatelé u daného testu a dále také průměrné hodnocení testu. Tyto agregační dotazy bývají typicky poměrně výpočetně náročné a budou velmi často potřeba - například při zobrazení seznamu všech testů.

4.4 Webové API - backend

4.4.1 Zvolené technologie

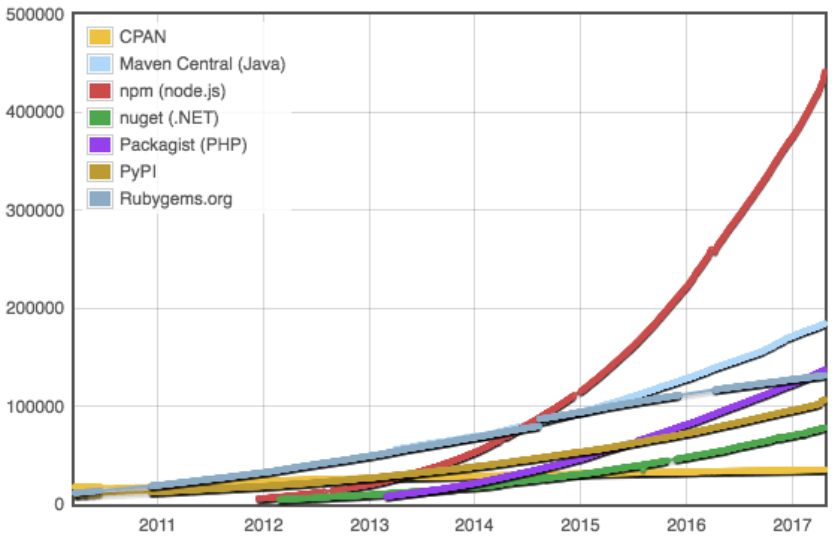
4.4.1.1 Programovací jazyky

Pro tvorbu API i webové aplikace byl zvolen programovací jazyk Javascript. Fakt, že vývojář používá ten samý jazyk pro tvorbu klienta i API, velmi usnadňuje práci a mentální zátěž. API bude implementováno v Node.js [50].

4.4.1.2 Node.js

Node.js primárně funguje jako runtime (prostředí pro běh) Javascriptu mimo webový prohlížeč. Node.js je založeno na javascriptovém enginu V8 [51], který vyvíjí společnost Google. Mimo to ale poskytuje i rozsáhlé API, obsahující například metody pro síťovou komunikaci, streamy - tedy proudové zpracování

vstupu a výstupu nebo vlastní systém modulů. Node.js zavádí asynchronní přístup k I/O operacím, založený na událostech a reakcích na ně. Díky tomu se Node.js obzvlášť hodí k implementaci jednoduchých HTTP serverů - všechna volání jsou neblokující (tedy například dotaz na databázi aplikaci nenutí čekat na výsledek) a všichni klienti jsou obslouženi pouze v jednom vlákně. Díky tomu se servery v Node.js vyznačují velkou propustností a zároveň náchylností k výpočetně náročným operacím. Mezi další výhodu Node.js patří velmi rozsáhlý ekosystém modulů a knihoven `npm` [52]. Co do počtu registrovaných modulů je vůbec největší, a přitom Node.js je velmi mladou technologií (vznikl v roce 2011).



Obrázek 4.10: Vývoj počtu modulů či knihoven v různých jazycích (zdroj: modulecounts.com [5])

4.4.1.3 Webový framework

API používá webový framework Koa [53], jenž se velmi podobá populárnímu Node.js frameworku Express [54], liší se zejména tím, že ve své aktuální verzi podporuje použití Javascriptových klíčových slov `async/await` viz 4.4.2. Koncepce obou těchto frameworků je mírně odlišná od klasických MVC/MVP webových frameworků (například Ruby on Rails či Nette), zakládají si na modularitě a jednoduchosti. Koa a Express jsou v podstatě routery, které umožňují k cestám registrovat další funkce (v terminologii Koa middleware funkce), které se vykonají podle zvoleného pořadí. Middleware je možné přiřadit pro celou aplikaci - tedy pro všechny cesty API (například middleware, který bude parsovat tělo HTTP requestu), pro podmnožinu cest (například

část API, která vyžaduje autorizaci) nebo pro jedinou cestu (například validace vstupu pro konkrétní endpoint). Mimo to ale oba frameworky mají podporu i pro šablonovací systém a práci se statickými soubory, lze pomocí nich tedy vybudovat plnohodnotnou webovou aplikaci, ne pouze API.

Koa v sobě neobsahuje další obvyklé moduly, například nemá vestavěné validátory dat nebo ORM. Vývojáři je tak ponechána možnost výběru, jaký externí modul bude na co použit. API tím pádem využívá velké množství externích modulů a knihoven. Jeden z těch nejvýznamnějších pro projekt je ORM pro Postgres a MySQL jménem Sequelize [55].

4.4.1.4 Databáze a ORM

V aplikaci je použita relační databáze PostgreSQL. Vzhledem k tomu, že datová struktura je poměrně složitá, je nutné použít i nějaký nástroj pro ORM, tedy Objektově relační mapování. Nutno uvést, že v této oblasti Node.js moduly lehce zaostávají a zatím neexistuje řešení tak kvalitní a propracované jako například Doctrine (ORM knihovna pro PHP) [56]. Nakonec bylo zvolen populární Node.js modul Sequelize [55], který poskytuje jak základní ORM funkce (mapování tabulek na objekty a zpět), tak poměrně robustní query builder, nástroj pro pouštění migrací a podporuje i transakce.

4.4.2 Asynchronní volání

Každý, kdo někdy programoval v jazyce Javascript, nejspíš narazil na princip tzv. callbacků, tedy funkcí, které jsou předány jako parametr jiným funkcím a jsou volány v reakci na nějakou událost. To je samo o sobě v pořádku, pokud se ale podobná volání začnou více řetězit, vznikne těžko přehledný kód. Takový kód je náročné číst a editovat, protože nebude vykonán tak, jak by se dalo očekávat, tedy synchronně od prvního řádku po poslední. Místo toho při běhu bude skákat různými funkcemi a programátor tak musí udržovat v hlavě jakýsi „zpětný“ model - pamatovat si, který callback je přiřazen jakému volání. Postupně bylo v Javascriptu zavedeno několik nástrojů, jak tomuto problému zamezit a asynchronní funkce zapisovat tak, aby vypadaly jako ty synchronní. Nejdůležitější je koncept tzv. `Promise`, což je speciální objekt, který reprezentuje data, která budou získána nějakým asynchronním voláním (například čtením souboru, dotazem na server) někdy v budoucnu. Promise objekt se může nacházet ve třech stavech - čekání, úspěch a chyba. Objekty se na sebe dají řetězit. Díky tomu je možné se callbackům vyhnout a nahradit je řetězem Promise objektů, což se už čte poněkud lépe. Ani tento zápis se úplně nepodobá klasickému synchronnímu kódu, dalším krokem proto je použití generátorů nebo klíčových slov `async/await`. Oba způsoby v podstatě využívají princip `Promises`.

Rozdíl mezi klasickým callback přístupem a nejnovější notací je znázorněn v ukázkách 1 a 2.

```
request('http://google.com', (err, response) => {
    if (err) {
    /* pokud HTTP request skončil chybou,
     * bude předána sem */
    }
    const body = response.body
    /* pokračuj v~práci s~výsledkem requestu */
})
```

Listing 1: Ukázka použití callbacku

```
async function foo() {
    try {
        const response = await request('http://google.com')
        console.log(response.body)
        /* pokračuj v~práci s~výsledkem requestu */
    } catch (err) {
    /* pokud HTTP request skončil chybou,
     * bude předána sem */
    }
}
```

Listing 2: Ukázka použití async/await notace

4.4.3 Nasazení a hosting aplikace

Z důvodů již popsaných výše API i klientská webová aplikace běží na oddělených serverech, hostovaných službou Heroku [57]. Heroku funguje jako PaaS (platforma jako služba). Nabízí hosting aplikací v různých jazycích (Node.js, PHP, Go, Java atd.), používá vlastní systém kontejnerů, ale umí pracovat i s Docker kontejnery. Dále poskytuje množství tzv. „addons“. K vlastní aplikaci lze připojit různé další služby, například databázi nebo monitorovací systém.

4.4.4 RESTful API

Zásady REST architektury formuloval v roce 2000 Roy Fielding [58]. API či jiné rozhraní nazýváme RESTful, pokud je navrženo podle základních REST principů. Klíčovým elementem REST architektury je **zdroj** (resource). Jedná se vlastně o data, kterým lze přiřadit nějaké jméno a tím se dají popsat (auto, zaměstnanec, obrázek). Zdroje zpravidla rozšiřují entity z datového modelu.

Zdroj má tyto vlastnosti [59]:

- **Identifikátor**(URI): jednoznačný a unikátní klíč.

- **Reprezentace**: Stav daného zdroje v daném čase, vyjádřený v nějakém formátu (nebo formátech). Používá se například JSON nebo XML.

4.4.4.1 Zásady REST architektury:[60]

1. Klient/server: oddělení UI a úložiště dat vede k lepší přenositelnosti UI a škálovatelnosti serveru.

2. Bezestavovost (stateless): Každý klientský požadavek musí obsahovat veškeré informace, nutné k vykonání. Stav sezení (session) či kontext musí být záležitost pouze klienta.

3. Cache: Požadavky by měly být explicitně rozdělené na ty, jenž se mohou a nemohou kešovat.

4. Jednotné rozhraní (uniform interface): Pravidla, kterých by se RESTful rozhraní měla držet, aby navržené rozhraní bylo pochopitelné pro jakékoliv jeho uživatele. Jedná se například o vhodné pojmenování zdrojů (pomocí URI) či dostatečně popisnou reprezentaci zdrojů. Klade důraz na oddělení zdroje a jeho reprezentace.

5. Vrstvy systému: Klient nemusí a nemá vědět, zda komunikuje přímo se serverem nebo s nějakým prostředníkem (například proxy).

4.4.4.2 Operace se zdroji

REST architektura definuje konečný počet operací, které je možno se zdroji provádět. Operace jsou nezávislé na doméně - tedy stejné pro jakékoliv RESTful API. To jsou veškerá omezení na operace podle originální specifikace ([58]). Nejčastěji se používají tzv. CRUD operace - tedy create (vytvořit), read (číst), update (upravit) a delete (odstranit). Tyto operace bývají vyjádřeny HTTP metodami, `GET` pro čtení, `POST` pro vytvoření zdroje, `DELETE` pro odstranění a `PUT` nebo `PATCH` pro úpravu.

Selftester API bude používat pro úpravu zdroje metodu `PATCH`, protože bude vykonávat částečné úpravy již vytvořeného zdroje.

4.4.4.3 Jmenné konvence a další zásady [61]

Je vhodné rozlišit kolekce a jednotlivé elementy zdrojů.

- Požadavek čtení pro kolekci:

  ```
  HTTP GET http://api.com/users
  ```

- Požadavek čtení pro element:

  ```
  HTTP GET http://api.com/users/{id}
  ```

Další důležitá zásada je konzistence při vytváření identifikátorů. Používá se množné číslo a URI obsahují jen podstatná jména - akce se vyjádří pouze pomocí HTTP metody. Dále je vhodné využít sémantiky, která se váže k HTTP status kódům - jak u úspěšných požadavků tak u těch, co vrátí chybu klienta.

4.5 Závěr

V této kapitole byly detailně popsány všechny aspekty návrhu webového klienta a API pro aplikaci Selftester. Případy užití byly zapracovány do návrhu UI, byl vytvořen a analyzován lo-fi prototyp. Byly detailně rozepsány konkrétní technologie a postupy pro implementaci obou aplikací a byl vytvořen a popsán příslušný datový model.

Kapitola 5

Implementace

5.1 Webová aplikace - klient

V rámci implementace webové aplikace byla kompletně dokončena pouze sekce s editorem, přihlašování a registrace. Klient je kompletně napojen na API a nepřímo tedy komunikuje i s mobilní aplikací (testy vytvořené na webu je možné vyplňovat v mobilu). Některé další sekce webové aplikace jsou již rozpracovány.

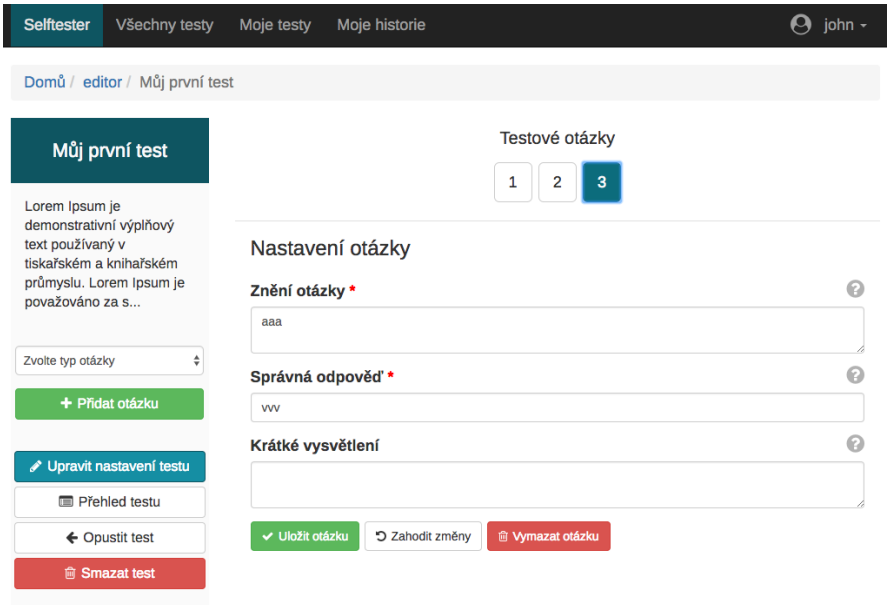
5.1.1 Rozvržení editoru do komponent

Editor testu má fungovat jako single-page komponenta v rámci celého webového klienta. Pro vytvoření kompletního testu uživatel nemusí opustit tuto stránku. Editor se skládá ze tří hlavních chytrých komponent:

- `Sidebar`: boční panel na levé straně, obsahuje většinu akcí, které může uživatel s testem provést.
- `QuestionsBar`: horní panel s odkazy na konkrétní otázky. Z důvodu uspoření místa je otázka reprezentována pouze číslem. Otázky jsou řazeny od nejnovější po nejstarší. Pořadí otázek nehraje roli při vyplňování testů - otázky v `testInstance` jsou vybírány náhodně.
- `ContentWrapper`: obsah této komponenty se dynamicky mění podle stavu editoru. Zobrazuje se v ní buď formulář k editaci testu či dané otázky anebo přehled testu.

Editor pochopitelně obsahuje množství prezentačních komponent, které renderují HTML a nejsou připojené ke `Store`.

Počet otázek na jeden test je limitovaný na 100. Pokud je otázek u testu více než deset, do panelu se přidá jednoduché stránkování (viz 5.2).



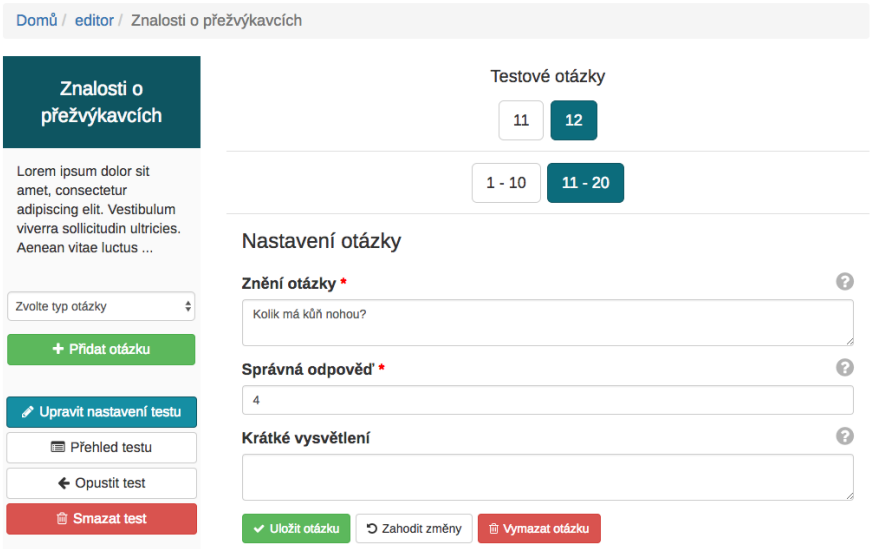
Obrázek 5.1: Konečná podoba UI - editor testu

5.1.2 JWT tokeny a udržování sezení klienta

Udržování sezení uživatele je základním nefunkčním požadavkem u každé webové i mobilní aplikace. Jak je vysvětlené níže, uživatel prokazuje svoji identitu pomocí podepsaného tokenu, který API vydává po přihlášení. Tento token je nutno v klientské aplikaci bezpečně a trvale uložit. Existuje několik variant:

1. Přímo ve Store: Token nepřežije obnovení stránky.
2. LocalStorage: Token není automaticky smazán po expiraci.
3. SessionStorage: Token nepřežije, pokud uživatel zavře okno prohlížeče.
4. Cookies: Token je uložen dokud nevyprší jeho expirace (je snadné datum expirace přečíst z těla tokenu a pak ho nastavit při ukládání cookie). Je nutné podotknout, že cookies jsou použité pouze jako mechanismus k uchování dat, ne k samotné autentizaci.

Ani jedna z variant nemusí být špatně, záleží na potřebách konkrétní aplikace, nicméně v tomto případě byla zvolena poslední varianta - token je uložen mezi cookies. Pokud dojde k novému načtení stránky, aplikace se nejprve pokusí použít stávající token. Dokud token není zpracován, aplikace nezobrazí žádná data. Případné přesměrování z chráněné části aplikace realizuje modul `react-router`. Je důležité, aby uživatel s validním tokenem zůstal na chráněné URI i po kompletním obnovení stránky.



Obrázek 5.2: Konečná podoba UI - editor testu se stránkováním

5.1.3 Lokální stav aplikace

V předchozích kapitolách již byly popsány principy a způsoby uchování stavu webové klientské aplikace. Reduxový `Store` obsahuje množství dat z API a zároveň i další pomocné údaje (například stav aktuálně zobrazeného formuláře). I když to není nutné, je více než vhodné data normalizovat, tedy vyhnout se jakýmkoliv duplicitám.

Toto je implementované pomocí Javascriptové knihovny Normalizr ([62]). Veškerá data z API jsou uložena v části `Store` jménem `entities` a v dalších částech (například `comments`, `tests`) jsou uloženy pouze ID a pomocné pole (například údaj o tom, že se data právě stahují, na což mohou komponenty zareagovat tak, že vykreslí progress bar a tak dále).

Pro účely komponent data jsou denormalizována, aby každá komponenta dostala kompletní údaje, potřebné ke svému fungování (například detail testu včetně komentářů). Denormalizace je realizována pomocí tzv. selektorů.

Analogie s relačními databázemi je poměrně přímočará, pomocí knihovny `Normalizr` odstraňujeme redundance a ukládáme data do pomyslných tabulek, selektory se potom podobají SQL `select` dotazům. Celá struktura je ale mnohem více dynamická a ukládají se jen ty data a vazby, které jsou potřeba.

Dalšími důležitými částmi `Store` jsou:

- `form`: Informace o stavu vykreslených formulářů.
- `auth`: Informace o přihlášeném uživateli.
- `appState`: Obecné informace o aplikaci jako jazyk, notifikace.

- `editor`: Soubor pomocných údajů o aktuálně upravovaném testu.

5.2 API

Jak již bylo popsáno v kapitole Návrh, API se - zjednodušeně řečeno - skládá z middleware funkcí, controllerů a robustní vrstvy, jenž komunikuje s databází. Datová vrstva je oddělena od controllerů na základě MVC principu. API poskytuje data pouze ve formátu `JSON`.

5.2.1 Autentizace s JWT

API na své straně neukládá žádné informace, spojené s aktuálním stavem klientské aplikace (například údaje o přihlášeném uživateli). Vše, co je nutné si pamatovat, je uložené na straně klientské aplikace. Každý HTTP request na API, jenž vyžaduje nějaké doplňující informace, musí být obohacen o autorizační token. Token zároveň obsahuje všechny informace, které API potřebuje k vyhodnocení požadavku.

Především se pomocí tokenu ověřuje identita uživatele. JWT token [17] se skládá ze třech částí - hlavičky, těla a podpisu. Podpis je vlastně hash, vygenerovaná serverem a podepsaná tajným, dostatečně dlouhým náhodným řetězcem, jenž zná pouze server. Informace obsažené v tokenu tedy není možné podvrhnout (například vyměnit své id za id jiného uživatele). Token ale pochopitelně může být odposlechnut a ukraden, proto je nutné jej používat v kombinaci s HTTPS protokolem (stejně jako jakákoliv jiná data, posílaná přes HTTP). Dále je nutné nastavit expiraci tokenu. API generuje tokeny s defaultní dobou platnosti dvě hodiny.

5.2.2 Middlewares

Jak již bylo řečeno, middleware je funkce, kterou lze přiřadit k nějakému API endpointu. Middleware lze nastavit na úrovni aplikace (bude zavoláno při jakémkoliv HTTP requestu), pro skupinu endpointů nebo jen pro jeden konkrétní. API využívá celou řadu middlewares, jak vlastních, tak stažených jako NPM moduly. Middlewares se vykonávají v tom pořadí, v jakém jsou zaregistrované (tedy například ty na úrovni aplikace jsou z pohledu zavádění kódu jako první, takže se vykonají vždy nejdříve).

5.2.2.1 Middlewares na aplikační úrovni

Jedná se zejména o externí moduly, sloužící k parsování těla a hlavičky HTTP requestu nebo ke kompresi HTTP odpovědi. Na aplikační úrovni se zavádí jeden velmi důležitý modul pro stabilitu aplikace - globální error handler. Middleware zajišťuje, aby instance API nebyla neočekávaně ukončena, aby se vyjímky správně logovaly a chybové hlášky správně odesílaly zpět klientovi.

Pokud API běží v `development` režimu, odešle se kompletní chybová hláška, pokud běží v produkčním režimu, tato informace se neodešle.

5.2.2.2 Sdílené middlewares

Další v pořadí jsou middlewares, které se registrují při definování samotných endpointů (ukázka 3, používá dvě middleware
funkce - `middleware.auth.fetchUser()` dohledá uživatele, pokud má k dispozici token a `middleware.auth.isLogged()` dohledá uživatele a pokud se to nepodaří, vyvolá `AuthorizationError`, který vrátí odpověď s HTTP status kódem 401 - Unauthorized). Tyto funkce řeší autentizaci (tedy ověření identity uživatele) a autorizaci (tedy kontrolu práv uživatele na danou akci). Identita uživatele je ověřena přečtením obsahu přiloženého JWT tokenu. Middleware `middleware.auth.userIsOwner` kontroluje, zda je nalezený uživatel vlastníkem entity, na kterou se HTTP request dotazuje a pokud není, vyvolá příslušnou chybu s HTTP status kódem 403 - Forbidden.

```
/* Test Instances */
const testInstances = new Router({
    prefix: 'tests',
})
/* Vygenerovat novou instanci */
testInstances.get('/models/:test_model_id',
    middleware.auth.fetchUser(),
    controllers.testInstance.generate)
/* Ziskat detail testu do historie */
testInstances.get('/:test_instance_id',
    middleware.auth.isLogged(),
    controllers.testInstance.get)
router.use(testInstances.routes())
```

Listing 3: Ukázka definice API endpointů s middleware funkcemi

5.2.2.3 Middleware pro specifický endpoint

Jako poslední v pořadí se provedou middlewares, které jsou přiřazeny jednotlivým endpointům. Jedná se o nepostradatelnou část každého webového API - tedy validaci vstupů. Je možno validovat tělo nebo hlavičku (query string) HTTP požadavku podle explicitně zadaných pravidel. K definování validačních pravidel se používá Node.js modul jménem Joi.

5.2.2.4 Pořadí middleware funkcí

V poslední řadě je nutné zdůraznit důležitost pořadí vykonávání jednotlivých middleware funkcí. Ty, které se zabývají bezpečností, by vždy měly být zavedeny co nejdříve, aby API neprozrazovalo žádné informace navíc. Ještě před tím ale ovšem musí běžet pomocné middleware funkce, které ostatním v pořadí umožní pracovat se součástmi HTTP požadavku jako s objekty a ne stringy.

5.2.3 Rozvržení endpointů

Endpointy jsou rozděleny do několika skupin podle účelu a nastavení přístupu:

1. `/tests`: označuje akce s `testInstance` modely, aktuálně je využívaný pouze mobilním klientem. Některé akce nevyžadují autentizaci, některé ano (například získávání instancí ke zobrazení historie). Je zde také endpoint, který vygeneruje z `testModel` konkrétní `testInstance` a tím zahájí vyplňování testu. Endpoint funguje tak, že náhodně vybere nějaký počet otázek z `testModel` a ty zkopíruje do instance. Tento počet je buď nastavený v rámci editace testu, nebo se dá nastavit pomocí parametru v hlavičce daného HTTP požadavku a pokud ani jedno není nastaveno, budou součástí instance všechny otázky z modelu.

2. `/testModels`: označuje akce s `testModel` modely, které nesouvisí s editorem. Jedná se například o přidávání hodnocení nebo komentářů, stahování různě detailních informací o testModelu.

3. `/editor`: Kompletní CRUD akce, které slouží k editaci `testModel` nebo `questionModel` modelů. API nevystavuje žádné endpointy na editaci odpovědí, ty se dají editovat výlučně skrze otázku, ke které jsou přiřazeny. Pokud objekt s otázkou obsahuje pole odpovědí, metoda v controlleru synchronizuje tabulku v databázi s tímto polem, pokud tedy API obdrží prázdné pole odpovědí, stávající odpovědi budou vymazány.

API dále obsahuje klasické endpointy na registraci, přihlášení uživatele, akce s komentáři a tak dále.

Všechny URI se skládají z podstatných jmen v množném čísle anebo ID daných entit. API neobsahuje žádné endpointy, vytvořené předem - všechny endpointy se používají buď mobilním nebo webovým klientem a vznikaly podle jejich požadavků.

5.2.4 Chybové hlášky

Součástí API jsou vlastní třídy Javascriptových errorů, které jsou pak v aplikaci využívány. Třídy slouží především k tomu, aby poskytovaly klientovi konzistentní odpovědi především co se týče HTTP status kódů a také pro zvýšení přehlednosti a testovatelnosti kódu aplikace.

- `ValidationError` (status 400): chyba při validaci vstupu. Zpráva je generována validační knihovnou.
- `UnauthorizedError` (status 401): nevalidní, prošlý či chybějící autorizační token. Celá škála případů, kdy se 401 vrací je k dohledání v integračních testech.
- `ForbiddenError` (status 403): Autentizace byla úspěšná, ale daný uživatel nemá pravomoce na vykonání daného požadavku.
- `NotFoundError` (status 404): Požadovaný zdroj nebyl nalezen.
- `ConflictError` (status 409): Záznam již existuje (nejčastěji kvůli zachování `UNIQUE` omezení v databázi).
- `ApiError` (status 500): Jakákoliv jiná neznámá chyba.

5.2.5 Testování

Zdrojový kód k API obsahuje klasické jednotkové testy, ale také počet integračních testů, které vždy testují jeden konkrétní endpoint. Integrační testy běží za použití vlastní databáze a pro zachování integrity a nezávislosti každého testu se obsah databáze před každým integračním testem celý smaže a znovu vytvoří.

Současně je monitorováno pokrytí zdrojového kódu testy. Aktuální pokrytí celého API je okolo 88 %.

5.3 Závěr

V této kapitole byly popsány některé implementační problémy nebo jinak významné součásti obou aplikací z pohledu implementace. Obě aplikace v této podobě běží v rámci testovacího provozu na těchto adresách:

- https://selftester-web.herokuapp.com - webová aplikace
- https://selftester-api.herokuapp.com - API

Kapitola 6

Průběh vylepšování UI a Uživatelské testování

V kapitole Návrh byl popsán lo-fi prototyp a otestován pomocí Nielsenovy heuristiky. V rámci implementace vznikl hi-fi prototyp, který se následně (po připojení ostrého API) stal konečnou funkční aplikací. Nad touto aplikací bude provedeno uživatelské testování, ještě před tímto krokem došlo k opravě několika problémů, jenž odhalila heuristická analýza nad lo-fi prototypem:

6.1 Opravy na základě testování lo-fi prototypu

Seřazeno dle závažnosti, přiřazené v rámci vyhodnocení heuristiky (viz 4.1.5.1). Některé problémy nebyly opraveny, protože ve výsledné aplikaci nebyly implementovány.

- Je možné odstranit z editovaného testu pouze jednu otázku.
- Aplikace je v češtině (prototyp byl v angličtině).
- Formuláře v editoru jsou opatřeny vysvětlivkami, což uživateli objasní, co přesně která položka znamená.
- Význam jednotlivých akcí/tlačítek je podpořen barvami (například červená pro mazání) a ikonami.
- Editor nenechá uživatele opustit editovaný a neuložený formulář. Tím zamezí váhání uživatele, který se může bát, aby neztratil neuložená data a jasně mu řekne, jak má postupovat.
- Otázky lze ukládat i rozpracované. Změny je vždy možné uložit nebo zahodit a tím se formulář vrátí do posledního uloženého stavu.

6.2 Uživatelské testování

Ačkoliv aplikace je dostupná online, testování bude probíhat na notebooku autorky, aby bylo možné zaznamenávat obrazovku. Zvolení testeři jsou studenti anebo lidé, kteří i po ukončení studia pokračují v sebevzdělávání. Před začátkem je tester obeznámen s účelem aplikace, popis ale není příliš detailní, aby uživatel nedostal přílišnou nápovědu. Po ukončení samotného testovaní uživatel a zadávající diskutují nad použitelností aplikace, zjišťuje se, jaké části uživateli nevyhovovaly, co mu přišlo matoucí nebo co mu chybělo. Diskuse má za úkol posbírat další podněty na vylepšení či vytvoření nových funkcí v dalších fázích vývoje, zatímco testy by měly odhalit chyby, jež budou opraveny hned v další iteraci vývoje webového klienta.

6.2.1 Zadání uživatelských testů

Jednotlivé úkoly mají reflektovat chování uživatelů, kteří na aplikaci narazí poprvé a postupně se tak učí ji používat. Úkoly tedy mají být vykonávány v uvedeném pořadí, tester nemá přeskakovat (a neměl by zadání dalších úkolů ani vidět, dokud se k nim nedostane).

1. Zaregistrujte se do aplikace a vytvořte test, který je určen pro Vaše kamarády a má vyhodnotit, jak dobře vás znají (otázky typu „jaké je moje oblíbené jídlo“). Test bude obsahovat tři otázky a každá otázka bude jiného typu. Nepovinné položky vynecháte. Po dokončení editace se vraťte na seznam všech testů.

2. Vytvořte další test (téma není důležité) a v něm otázku, která má mít několik možných správných odpovědí. Vytvořte k této otázce čtyři různé odpovědi. Dvě z odpovědí budou označeny jako správné, dvě jako špatné. Až to bude hotové, vraťte se na seznam všech testů.

3. Změňte název tohoto testu na „Můj druhý test“.

4. K otázce z prvního testu o Vás, která je typu Textová odpověď, přidáte vysvětlení, proč je daná odpověď zrovna taková (může jít například o nějakou historku, která se k otázce váže :))

5. Smažte test jménem „Můj druhý test“.

6. Odhlašte se z uživatelského účtu.

7. Přihlašte se pod uživatelským jménem john a heslem password (vše malými písmeny).

8. V již vytvořeném testu v tomto uživatelském účtu přejděte na otázku s číslem 13 a nastavte poslední odpověď jako tu správnou. Poté se odhlaste.

6.2.2 Uživatelské scénáře - zamýšlené řešení uživatelských testů

1. Uživatel přejde na webovou stránku aplikace a přejde pomocí odkazu v horním menu na registrační formulář. Po odeslání je přesměrován na přehled editoru, kde klikne na tlačítko Vytvořit test. Dostane se do editoru. Nejprve vyplní obecné údaje o testu a poté přidá různé otázky pomocí dropdown výběru po levé straně. Při tvorbě otázek vynechá políčko Vysvětlení. Bez snahy test nějak kompletně uložit uživatel přejde na úvodní stránku.

2. Uživatel vytvoří test a vyplní obecné údaje. Poté z dropdownu na přidání otázky zvolí otázku typu Více odpovědí správně. Přidá další možnou odpověď. U dvou odpovědí odškrtne checkbox s popisem „správná odpověď“ a u dvou jej zaškrtne. Poté se opět na jedno kliknutí vrátí na úvodní stránku.

3. Uživatel přejde na seznam svých testů a klikne na Editovat test. Z panelu na levé straně vybere Editovat test, změní jméno a výsledek uloží.

4. Uživatel přejde na seznam svých testů a klikne na Editovat test. Z panelu s otázkami nahoře vyhledá otázku správného typu, vyplní pole Vysvětlení a výsledek uloží.

5. Uživatel klikne na Smazat test v přehledu všech testů nebo editaci konkrétního testu. Správně odklikne následné modální okno s potvrzením akce.

6. Uživatel rozklikne lištu s odkazy u své přezdívky v pravé horní části okna a zvolí Log out.

7. Uživatel přejde pomocí odkazu Login v horním menu na formulář a ten vyplní dle zadání a odešle.

8. Uživatel přejde na editaci předpřipraveného testu. Aby se dostal na otázku číslo 13, nejprve klikne na odkaz v dolní části panelu s otázkami, který slouží k paginaci „11-20“ a poté vybere otázku číslo 13. Ze seznamu odpovědí vybere právě tu poslední jako správnou, tedy zaškrtne checkbox anebo radio box „správná odpověď“. Výsledek uloží a odhlásí se (viz úkol č. 6).

6.2.3 Výsledky první iterace testování

V podkapitole je popsáno chování a problémy jednotlivých testerů a dále některé podněty, které vyšly najevo ze závěrečné diskuse. Zjištěné problémy jsou ohodnoceny dle závažnosti. Chyby jsou ohodnoceny podle tabulky z testování lo-fi prototypu 4.1.

6.2.3.1 Tester 1 (student FIT ČVUT)

Výsledky testu:

1. Při vytvoření nového testu měl problém přejít na formulář se základními údaji o testu (který je nutné vyplnit jako první), protože ignoroval delší úvodní text v hlavní části editoru, který mu vysvětloval další kroky (2 body). Nebylo jasné, k čemu slouží pole Počet otázek ve vygenerovaném testu a toto pole je povinné (3 body).
2. Tlačítko Přidat odpověď se nachází nahoře u seznamu otázek a když uživatel vyplní poslední odpověď a pak chce přidat další, tak ho nevidí (2 body).
3. Bez problémů.
4. Příslušné pole bylo nalezeno, ale nebylo jasné, k čemu přesně slouží (1 bod). Nebylo jasné, že horní panel s čísly označuje otázky (2 body).
5. Bez problému.
6. Bez problému.
7. Bez problému.
8. Bez problému.

Další poznámky:

- Po uložení otázky by měl formulář zmizet a měl by se zobrazit nějaký přehled i s otázkami.
- Není jasné, že horní panel označuje otázky.
- Drobečková navigace by měla být více srozumitelná.

6.2.3.2 Tester 2 (student FIT ČVUT)

Výsledky testu:

1. Tester chtěl editovat přímo název a popis v panelu na levé straně. Trvalo dlouho, než pochopil, že musí pokračovat na základní formulář a také jak na něj přejít (3 body). Tester si nevšiml, které údaje jsou označeny jako nepovinné (1 bod).
2. Bez problému.
3. Bez problému.
4. Tester hledal seznam s otázkami (2 body).

5. Bez problému.
6. Bez problému.
7. Bez problému.
8. Bez problému.

Další poznámky:

- Tlačítko Přidat otázku by nemělo „zdržovat“ s výběrem typu otázky - mělo by fungovat víc přímočaře a na typ otázky se doptat až potom.
- Není jasné, že horní panel označuje otázky.
- Některé názvy jsou matoucí - například „editor“ vlastně označuje testy daného uživatele.

6.2.3.3 Tester 3 (student FSv ČVUT)

Výsledky testu:

1. Tester chtěl editovat přímo název a popis v panelu na levé straně (2 body).
2. Bez problému.
3. Bez problému.
4. Tester chtěl použít Přehled testu a tak zjistit, která otázka je jakého typu, ale tato informace se tam nezobrazuje (1 bod).
5. Bez problému.
6. Bez problému.
7. Bez problému.
8. Bez problému.

Další poznámky:

- Pole Časový limit by se mohlo nastavovat pro otázku a ne pro celý test. Pokud by to bylo pro celý test, bylo by nutné limit měnit při změnách počtů otázek.
- Není jasné, že horní panel označuje otázky.

6.2.3.4 Tester 4 (student UK)

Výsledky testu:

1. Tester nevěděl co vyplnit do pole Počet otázek ve vygenerovaném testu (3 body).
2. Tester si nevšiml, že pole Vysvětlení je nepovinné (1 bod).
3. Bez problému.
4. Bez problému.
5. Bez problému.
6. Bez problému.
7. Bez problému.
8. Bez problému.

Další poznámky:

- Při hledání otázky je nutné všechna tlačítka proklikat, chtělo by to nějaký přehled.

6.2.4 Analýza zjištěných problémů z první iterace

V této kapitole budou zjištěné problémy agregovány a budou identifikovány ty nejčastější a nejzávažnější (viz tabulka 6.1).

Žádný z problémů nebyl natolik závažný, aby nějakému testerovi zabránil v dokončení úkolu. Všechny zjištěné problémy jsme se pokusili vyřešit v rámci úprav pro druhou verzi UI, úspěšnost oprav je ale nutné opět ověřit testy.

6.2.5 Úpravy

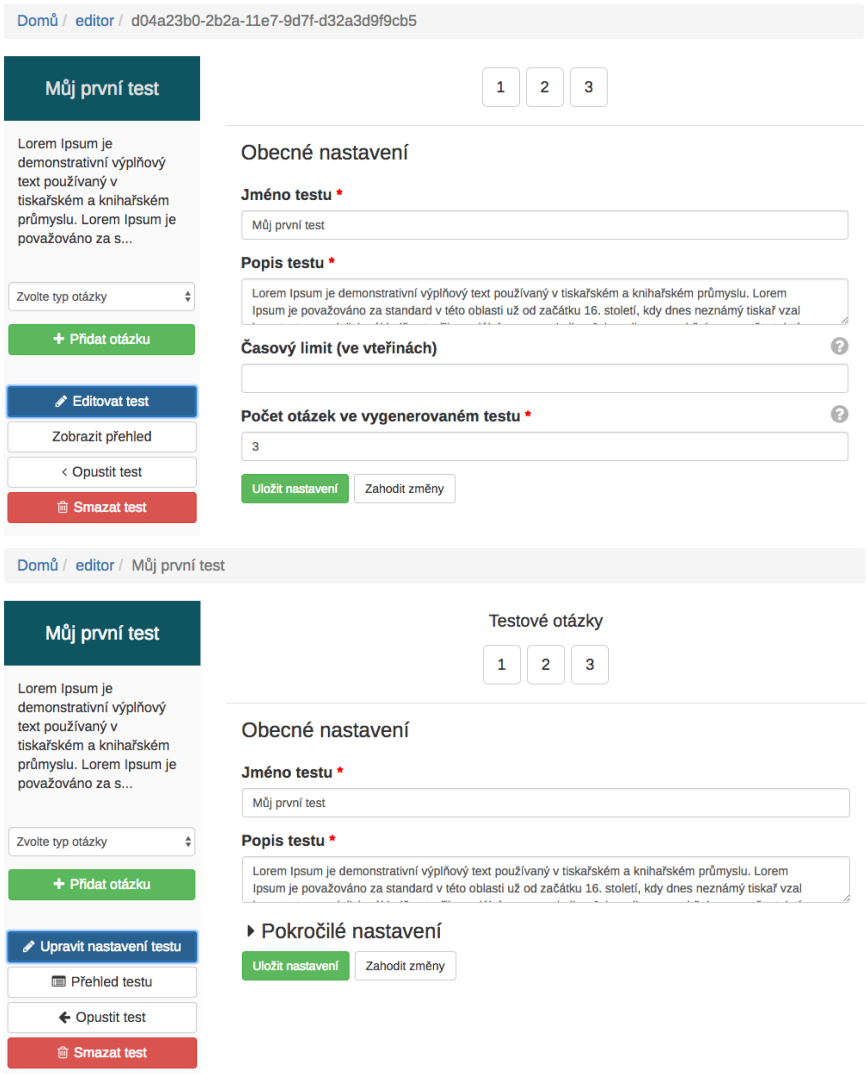
Na základě problémů, zjištěných při testování, a podnětů z diskuse s testery byly navrženy a implementovány tyto změny UI:

- Formulář s obecnými nastaveními testu byl rozdělen na základní a pokročilé položky. Pokročilé možnosti nastavení (časový limit a počet otázek ve vygenerovaném testu) jsou defaultně skryté. Tuto změnu bude nutno otestovat. Tento princip přímo vychází ze zásady Flexibility and efficiency of use z Nielsenovy heuristiky (7). Změna je zdokumentovaná zde 6.1.
- Po vytvoření nového testu se okamžitě objeví tento formulář a uživatel jej musí vyplnit, než se posune dál. Změna je zdokumentovaná zde 6.2.

Popis	Nejvyšší zjištěná závažnost	Počet výskytů
Nesrozumitelné a povinné pole Počet otázek ve vygenerovaném testu	3 body	3
Vyplnit a najít úvodní formulář s obecným nastavením testu.	3 body	3
Panelu s odkazy na otázky chybí popisek (není poznat, že se jedná o otázky) Přehled testu nezobrazuje informace o otázkách.	2 body	3
Tlačítko s možností přidat další odpověď není po ruce, pokud uživatel vyplňuje nastavení otázky postupně.	2 body	1
Nepovinné položky ve formulářích nejsou dostatečně výrazně označené	1 bod	2

Tabulka 6.1: Přehled problémů v UI z první iterace testů

- Panel s otázkami je opatřen o nadpis.
- Přehled testu zobrazuje i seznam otázek se základními údaji (typ otázky a zadání - zobrazí se max 60 znaků).
- Když dojde k uložení formuláře nebo resetu změn, uživatel sice zůstane na dané stránce, ale stránka vyjede nahoru, aby zpětná vazba byla výraznější (k uložení opravdu došlo). Podle některých testerů oznamovací hláška nebyla dostatečně výrazná, nebo si jí vůbec nevšimli.
- Došlo ke změnám v pojmenování tlačítek v hlavní navigaci a v bočním panelu.
- Drobečková navigace zobrazuje jméno testu nebo jiné entity dle definice, ne pouze údaje, které jsou dostupné z URI (doteď zobrazovala pouze ID testu, protože ostatní údaje pro ni nebyly dostupné).



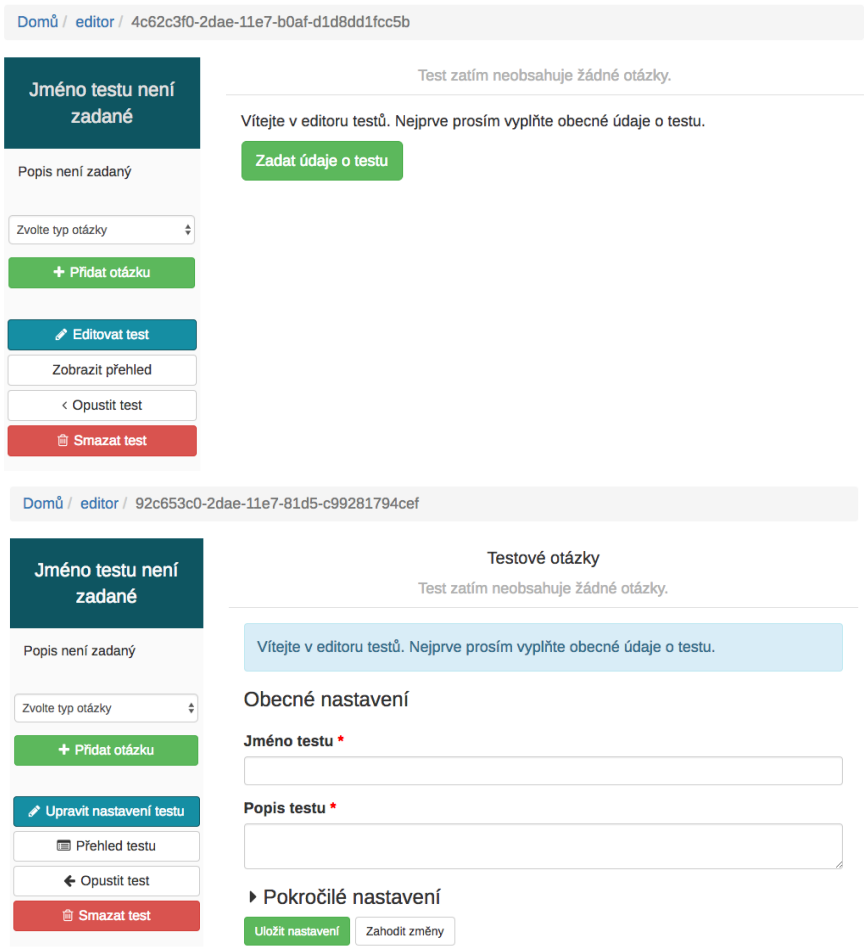
Obrázek 6.1: Úprava UI - skytí některých nastavení mezi pokročilé

6.2.6 Zadání uživatelských testů druhé iterace

Zadání bude totožné s první iterací (6.2.1), přibyde pouze jedna nová devátá otázka. Otázka navazuje na předchozí, takže v kontextu bude testerům srozumitelná.

9. Nastavte tomuto testu časový limit 5 minut, ve kterém uživatelé musí tento test stihnout celý vyplnit. Poté se odhlaste.

`Scénář`:
Uživatel přejde na editaci předpřipraveného testu. Pomocí tlačítka Upravit nastavení testu přejde na příslušný formulář a rozklikne lištu s pokročilým



Obrázek 6.2: Úprava UI - úvodní formulář pro nový test

nastavením. Do pole Časový limit vepíše hodnotu 300. Uloží změny a poté se odhlásí (viz úkol 6).

6.2.7 Výsledky druhé iterace testování

6.2.7.1 Tester 1 (student UTB)

Výsledky testu:

1. Bez problému.
2. Bez problému.
3. Bez problému.
4. Bez problému.

5. Bez problému.
6. Bez problému.
7. Bez problému.
8. Bez problému.
9. Chvíli trvalo, než tester našel rozklikávací panel s pokročilým nastavením, ale věděl, ve které části editace má hledat (1 bod).

Další poznámky:

- Tester se zezačátku bál odejít z editace testu, protože si nebyl jistý, že všechno uložil. Ocenil by někde napsané ujištění, že vše bylo úspěšně uloženo.

6.2.7.2 Tester 2 (student UTB)

Výsledky testu:

1. Testera zmátlo, že po uložení formuláře zůstal na té samé stránce (2 body).
2. Testerovi chvíli trvalo, než našel horní panel s otázkami (1 bod).
3. Testerovi chvíli trvalo, než našel tlačítko s odkazem na úpravy nastavení testu (1 bod).
4. Bez problému.
5. Bez problému.
6. Bez problému.
7. Bez problému.
8. Kvůli pomalejšímu internetu uložení otázky trvalo delší dobu a nebylo jasné, jestli došlo k odeslání formuláře (2 body).
9. Chvíli trvalo, než tester objevil panel s pokročilým nastavením (1 bod).

Další poznámky:

- Tester by očekával, že po uložení otázky aplikace opustí stránku s formulářem.
- Pokročilé nastavení u testu dává smysl, ale odkaz není dostatečně graficky zvýrazněn.

6.2.7.3 Tester 3 (student VŠE)

Výsledky testu:

1. Bez problému.
2. Bez problému.
3. Bez problému.
4. Tester chtěl přejít na editaci nastavení testu z přehledu testových údajů, tam ale chybí odkaz (1 bod).
5. Bez problému.
6. Bez problému.
7. Bez problému.
8. Bez problému.
9. Tester také nejprve přehlídl rozklikávací panel s pokročilým nastavením (1 bod).

6.2.8 Analýza výsledků druhé iterace a další navržené úpravy

Dobrou zprávou je, že nejzávažnější problémy z první iterace testování se ve druhé iteraci již neopakovaly (problémy se závažností 3, viz 6.1). Vyskytly se problémy s maximální závažností druhého stupně. Problémy spočívaly hlavně v tom, že se uživatel přehlídl ale z pohledu samotného procesu tvorby testu a přidávání otázek probíhalo vše v pořádku. Z toho vyplývá, že problémy, objevené v této iteraci, by šly řešit jen vizuálními úpravami editoru, což je samozřejmě mnohem snažší, než nějak měnit logiku fungování editoru.

Navržené úpravy:

- Lépe graficky zvýraznit panel s možnostmi pokročilého nastavení testu.
- Doplnit přehled testu o přímé odkazy k editaci testu i otázek.
- Explicitně uživatele informovat o tom, že všechny změny v testu jsou uloženy. Toto oznámení by mělo využívat stejných barev a ikon jako tlačítka v bočním panelu (stejná zelená nebo červená barva).
- Uživatel by měl být informován o tom, že formulář je odesílán (tlačítka uložit a resetovat by měla být v době odesílání zablokována a měla by se objevit informace o tom, že se požadavek zpracovává).

Tyto úpravy už spadají do dalšího potenciálního vývoje aplikace.

6.2.9 Závěr

Díky uživatelskému testování se podařilo odbourat množství procesních i vizuálních chyb v uživatelském rozhraní. Jelikož bylo zajištěno 7 testerů, testování bylo rozděleno do dvou iterací, přičemž druhá skupina pracovala s vylepšenou verzí aplikace a tím se hned ověřilo, že některé chyby se pravděpodobně povedlo vyřešit.

Ideální počet testerů pro každou iteraci je 5 až 6, ale potřebný počet se bohužel nepodařilo domluvit. I tak ale testování bylo velmi přínosné pro další rozvoj aplikace.

Kromě problémů UI se také pozorovalo, jak jsou uživatelé schopni naučit se s aplikací pracovat. První úkol sloužil vlastně k tomu si osahat všechny funkce a ostatní úkoly pak byly více specifické. Ukázalo se, že učící proces funguje poměrně dobře, po dokončení prvního úkolu tester pracoval s aplikací mnohem rychleji a málokdy hledal, jak další úkoly vyřešit.

V budoucnu bude nejspíše nutné učinit poměrně velké rozhodnutí ohledně UI - a to zda po odeslání formuláře mají uživatelé na stránce zůstat, nebo přejít někam jinam. Testeři byli na toto téma dotazováni. Čtyřem z nich vyhovovalo, že na stránce zůstali a mohli pokračovat v práci, nebo si ověřit zadané údaje, zbylí ale byli tímto chováním spíše zaskočeni.

Bude tedy nutné podrobit toto chování aplikace důkladnějšímu testování například formou dotazníku s větším počtem lidí a na základě toho případně provést změny.

Závěr

Hlavní náplní této práce bylo navrhnout a vypracovat webovou aplikaci, jenž bude součástí projektu Selftester. Projekt bude fungovat jako otevřená platforma pro tvorbu a vyplňování znalostních testů, a tím poslouží jako pomůcka k učení pro studenty jakéhokoliv zaměření. Je známo, že opakování dělá mistra a právě tímto způsobem by měl Selftestr pracovat.

Tato práce je úzce provázána s diplomovou prací Bc. Jakuba Kaliny. Jeho cílem bylo vytvořit mobilní aplikaci, která bude rovněž součástí projektu Selftester.

Z důvodů, jež jsou detailně popsány v kapitole Analýza (1.7), byla webová aplikace rozdělena na dvě zcela samostatné části - API a webového klienta. Obě aplikace jsou v rámci zkušebního provozu dostupné z internetu a mobilní aplikace je na ně napojena.

Jak je zřejmé z analýzy funkčních požadavků (1.3), tato webová aplikace je značně komplexním projektem. Proto byly pro návrh a implementaci webového klienta vybrány jen ty funkční požadavky, které odpovídaly potřebám projektu a v neposlední řadě také zadání práce.

Ideálním výsledkem obou prací (z pohledu autorů) bylo zkompletovat celý proces od tvorby testu až po jeho vyplnění a práci s výsledky. V rámci webové aplikace jsou tedy implementovány funkční požadavky týkající se tvorby, editace testů a správy uživatelských účtů (přesně jak odpovídá zadání), v rámci mobilní aplikace poté vyplňování a vyhodnocování výsledků testů. API pochopitelně implementuje veškeré tyto požadavky, jinak by nemohlo být využito jak webovým, tak mobilním klientem.

Větší část textu této práce se tedy zabývá popisem tvorby obou aplikací. Jsou zdokumentovány všechny fáze vývoje software, analýza (1), volba metodiky pro další postup (2), návrh uživatelského rozhraní, webového klienta i API (4), implementace (5) a proces testování uživatelského rozhraní (6). Uživatelské rozhraní bylo testováno ve třech iteracích, první test byl proveden bez uživatelů (4.1.5.1), další dvě iterace byly testovány za pomoci skutečných uživatelů z cílové skupiny - tedy studentů (6). Po prvních dvou iteracích byly

navrženy a implementovány změny, problémy nalezené v poslední iteraci budou předmětem dalšího vývoje aplikace.

Vývoj aplikací je tedy popsán opravdu komplexně, po všech stránkách, od uživatelského rozhraní, přes návrh databáze až po tvorbu API, které funguje na pozadí a obsluhuje obě klientské aplikace.

Dále práce popisuje a porovnává základní přístupy a principy tvorby webových aplikací (kapitola 3). Možností existuje celá řada, ale text se soustředí zejména na srovnání klasického přístupu za použití komplexního MVC frameworku a novější přístup, který dělí funkcionalitu mezi API a plnohodnotnou klientskou aplikaci, jenž běží v prostředí webového prohlížeče. Oba způsoby mají své výhody a nevýhody a případy použití.

Možnosti dalšího rozvoje obou aplikací jsou značné. Jedná se vlastně o zbývající funkční požadavky (viz. 1.3), jenž nebyly doposud kompletně implementovány. Mezi nejvýznamnější patří určitě editace testu více uživateli zároveň a zavedení multimediálního obsahu do testů. Dále by se dalo pracovat na optimalizaci obsahu jednotlivých vygenerovaných testových instancí - mohlo by se generovat primárně z otázek, na které daný uživatel v předchozích pokusech neznal odpověď a tak dále. Aplikaci plánujeme dále rozvíjet.

I když výukových aplikací existuje již celá řada, věříme, že tento projekt by mohl mít naději na úspěch a mohl by pomoci celé řadě žáků a studentů.

Literatura

[1] Bayley, D.: Building A Component-Based Web UI With Modern JavaScript Frameworks. August 2015, [cit. 2017-03-20]. Dostupné z: https://derickbailey.com/2015/08/26/building-a-component-based-web-ui-with-modern-javascript-frameworks/

[2] Videacesky: Videacesky.cz. [cit. 2017-02-14]. Dostupné z: http://videacesky.cz/

[3] QZZR: QZZR - Free online quiz tool. [cit. 2017-02-14]. Dostupné z: https://www.qzzr.com/

[4] Medium: Medium. [cit. 2017-02-14]. Dostupné z: https://medium.com

[5] DeBill, E.: Modulecounts. [cit. 2017-02-14]. Dostupné z: http://www.modulecounts.com/

[6] Prof. PhDr. Rudolf Kohoutek, C.: Dotazník. May 2005, [cit. 2017-03-20]. Dostupné z: http://www.ssvp.wz.cz/Texty/dotaznik.html

[7] Whittier, C.: How this Awesome Online Quiz Generated $1 Million in Cash [online]. 2015, [cit. 2017-02-14]. Dostupné z: http://blog.boombox.com/million-dollar-quiz-revenue/

[8] Long, J.: I Dont́ Speak Your Language: Frontend vs. Backend. September 2012, [cit. 2017-03-20]. Dostupné z: http://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend

[9] Google: GMail. [cit. 2017-03-14]. Dostupné z: https://gmail.com/

[10] QuizMaker: Quiz Maker. [cit. 2017-02-14]. Dostupné z: https://www.quiz-maker.com/

[11] QuizWorks: Online Quiz Creator. [cit. 2017-02-14]. Dostupné z: https://www.onlinequizcreator.com/

[12] Google: Google Forms. [cit. 2017-02-14]. Dostupné z: https://docs.google.com/forms/

[13] Google: Google Docs. [cit. 2017-02-14]. Dostupné z: https://docs.google.com/

[14] survio: Survio.com. [cit. 2017-02-14]. Dostupné z: http://www.survio.com/cs/

[15] vyplňTo: VyplňTo - řešení pro online průzkumy. [cit. 2017-02-14]. Dostupné z: https://www.vyplnto.cz/

[16] Moodle.org: Moodle - open source learning platform. [cit. 2017-02-14]. Dostupné z: https://moodle.org/

[17] Jones, M.: *JSON Web Token*. IETF, May 2015, [cit. 2017-02-14]. Dostupné z: https://tools.ietf.org/html/rfc7519

[18] Nielsen, J.: Search: Visible and Simple. May 2001, [cit. 2017-03-8]. Dostupné z: https://www.nngroup.com/articles/search-visible-and-simple/

[19] Fekete, G.: Designing The Holy Search Box: Examples And Best Practices. December 2008, [cit. 2017-03-8]. Dostupné z: https://www.smashingmagazine.com/2008/12/designing-the-holy-search-box-examples-and-best-practices/

[20] Whitenton, K.: Website Forms Usability: Top 10 Recommendations. May 2016, [cit. 2017-03-8]. Dostupné z: https://www.nngroup.com/articles/web-form-design/

[21] Network, Y. D.: Favorites UI Design pattern. [cit. 2017-02-14]. Dostupné z: https://developer.yahoo.com/ypatterns/social/objects/collecting/favorites.html

[22] Mlejnek, I. J.: Metodiky vývoje. 2016, [cit. 2017-03-8]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/11/11.prednaska.pdf

[23] Richta, K.: Metodiky vývoje software, MDA. April 2011, [cit. 2017-03-8]. Dostupné z: https://edux.fit.cvut.cz/oppa/BI-SI1/prednasky/BI-SI1-P10m.pdf

[24] TatvaSoft: Top 12 Software Development Methodologies. May 2015, [cit. 2017-03-8]. Dostupné z: http://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-disadvantages/

[25] Hus, M.: The case for separating front- and back-end. October 2014, [cit. 2017-03-20]. Dostupné z: http://dontpanic.42.nl/2014/10/the-case-for-separating-front-and-back.html

[26] Boyer, S.: Does Google execute JavaScript? August 2016, [cit. 2017-03-20]. Dostupné z: https://www.stephanboyer.com/post/122/does-google-execute-javascript

[27] Salihefendic, A.: Model View Controller: History, theory and usage. March 2011, [cit. 2017-03-20]. Dostupné z: https://web.archive.org/web/20150111052848/http://amix.dk/blog/post/19615

[28] Moldovan, A.: Is Model-View-Controller dead on the front end? October 2016, [cit. 2017-03-20]. Dostupné z: https://medium.freecodecamp.com/is-mvc-dead-for-the-frontend-35b4d1fe39ec

[29] Google: AngularJS. [cit. 2017-02-14]. Dostupné z: https://angularjs.org/

[30] Alfoni, C.: Why we are doing MVC and FLUX wrong. August 2015, [cit. 2017-03-20]. Dostupné z: http://www.christianalfoni.com/articles/2015_08_02_Why-we-are-doing-MVC-and-FLUX-wrong

[31] Žára, O.: JavaScriptové MVC frameworky. May 2013, [cit. 2017-03-20]. Dostupné z: https://www.zdrojak.cz/clanky/javascriptove-mvc-frameworky/

[32] Facebook: Flux. [cit. 2017-03-20]. Dostupné z: https://facebook.github.io/flux/

[33] Staltz, A.: Unidirectional user interface architectures. August 2015, [cit. 2017-03-20]. Dostupné z: https://staltz.com/unidirectional-user-interface-architectures.html

[34] Facebook: React. [cit. 2017-03-20]. Dostupné z: https://facebook.github.io/react/

[35] Strukchinsky, V.: BEM. [cit. 2017-02-14]. Dostupné z: http://getbem.com/

[36] css modules: CSS Modules. [cit. 2017-02-14]. Dostupné z: https://github.com/css-modules/css-modules

[37] Rounds, D.: A Short History of Computer User Interface Design. April 2016, [cit. 2017-03-20]. Dostupné z: http://blog.usabilla.com/short-history-computer-user-interface-design/

[38] Žikovský PhD., I. P.: Návrh uživatelských rozhraní - úvod. 2016, [cit. 2017-03-20]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x01-uvod.pdf

[39] Žikovský PhD., I. P.: Návrh uživatelských rozhraní - návrh a prototyping. 2016, [cit. 2017-03-20]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x02-navh_a_prototyping.pdf

[40] Studios, B.: Balsamiq - rapid, effective and fun wireframing software. [cit. 2017-02-14]. Dostupné z: https://balsamiq.com/

[41] Nielsen, J.: 10 Usability Heuristics for User Interface Design. January 1995, [cit. 2017-03-8]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[42] Nielsen, J.: Website Response Times. June 2010, [cit. 2017-03-8]. Dostupné z: https://www.nngroup.com/articles/website-response-times/

[43] Nielsen, J.: Usability 101: Introduction to Usability. January 2012, [cit. 2017-03-8]. Dostupné z: https://www.nngroup.com/articles/usability-101-introduction-to-usability/

[44] Žikovský PhD., I. P.: Návrh uživatelských rozhraní - testování s uživateli. 2016, [cit. 2017-03-20]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x04-usability_special_testing_personas.pdf

[45] webpack.io: Webpack. [cit. 2017-02-14]. Dostupné z: http://webpack.github.io/

[46] airbnb: eslint-config-airbnb. [cit. 2017-02-14]. Dostupné z: https://github.com/airbnb/javascript/tree/master/packages/eslint-config-airbnb

[47] Abramov, D.: Redux docs - Three basic principles. [cit. 2017-03-20]. Dostupné z: http://redux.js.org/docs/introduction/ThreePrinciples.html

[48] Clark, L.: A cartoon intro to Redux. October 2015, [cit. 2017-03-20]. Dostupné z: https://code-cartoons.com/a-cartoon-intro-to-redux-3afb775501a6

[49] Abramon, D.: Presentational and Container Components. May 2015, [cit. 2017-03-20]. Dostupné z: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

[50] foundation, N.: Node.js. [cit. 2017-03-14]. Dostupné z: https://nodejs.org/en/

[51] Google: V8 Engine. [cit. 2017-02-14]. Dostupné z: https://developers.google.com/v8/

[52] Npm: Npmjs. [cit. 2017-02-14]. Dostupné z: https://npmjs.com

[53] foundation, N.: Koa.js. [cit. 2017-03-14]. Dostupné z: http://koajs.com/

[54] foundation, N.: Express.js. [cit. 2017-03-14]. Dostupné z: https://expressjs.com/

[55] Sequelize: Sequelize. [cit. 2017-03-14]. Dostupné z: https://github.com/sequelize/sequelize

[56] Doctrine: Doctrine project. [cit. 2017-02-14]. Dostupné z: http://www.doctrine-project.org/

[57] Heroku: Heroku. [cit. 2017-02-14]. Dostupné z: https://www.heroku.com/platform

[58] Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures. 2000, [cit. 2017-03-20]. Dostupné z: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[59] doc. Ing. Tomáš Vitvar, P.: SOAP and REST. January 2015, [cit. 2017-03-20]. Dostupné z: http://mdw.vitvar.com/pdf/lecture8-1p.pdf

[60] RESTfulAPI.net: Rest API Tutorial. 2011, [cit. 2017-03-20]. Dostupné z: http://restfulapi.net/rest-architectural-constraints/

[61] Hauer, P.: RESTful API Design. Best Practices in a Nutshell. March 2015, [cit. 2017-03-20]. Dostupné z: https://blog.philipphauer.de/restful-api-design-best-practices/

[62] Armstrong, P.: *Normalizr*. May 2016, [cit. 2017-02-14]. Dostupné z: https://github.com/paularmstrong/normalizr

Příloha **A**

Seznam použitých zkratek

JSON Javascript Object Notation

REST Representational State Transfer

URI Uniform resource identifier

ORM Object-Relational Mapper

AJAX Asynchronous JavaScript and XML

JWT Json Web Tokens

MVC Model-View-Controller

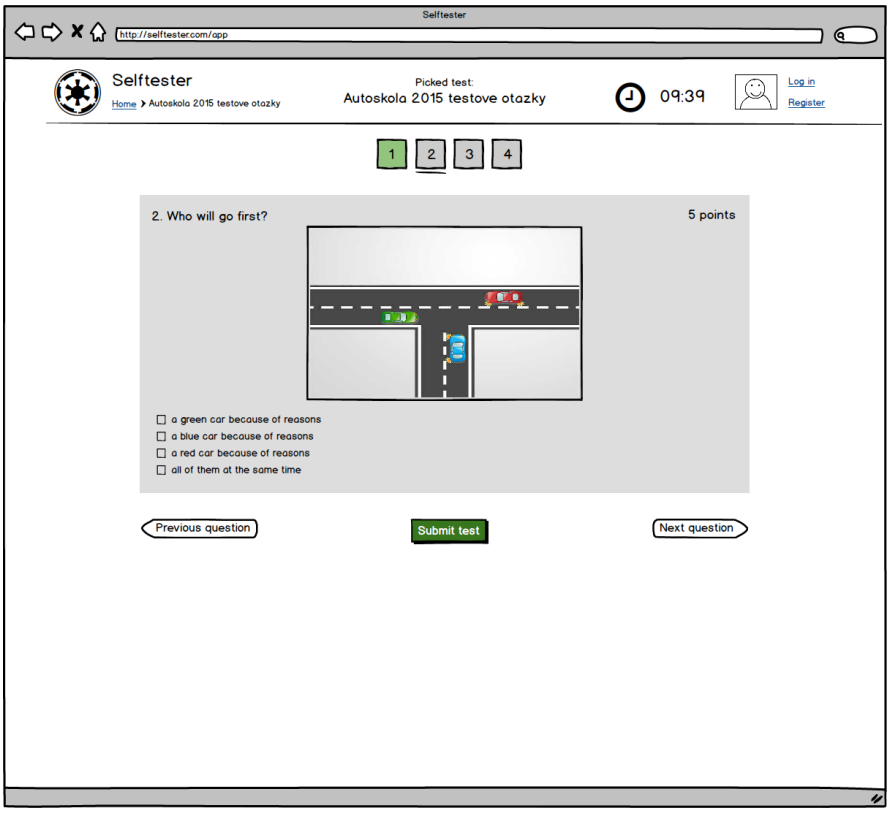
MVP Model-View-Presenter

MVVM Model-View-View-Model

Příloha B

Obrazovky k případu použití Vyplnit test

Obrázek B.1: První otázka v pořadí

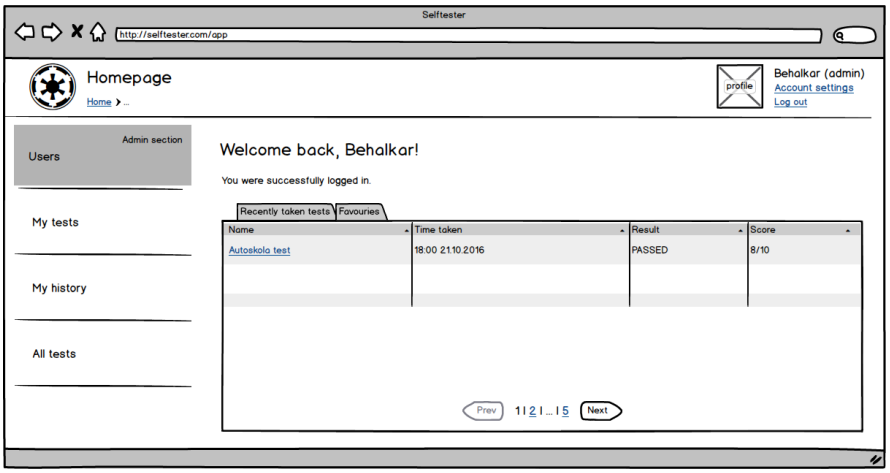


Obrázek B.2: Druhá otázka v pořadí

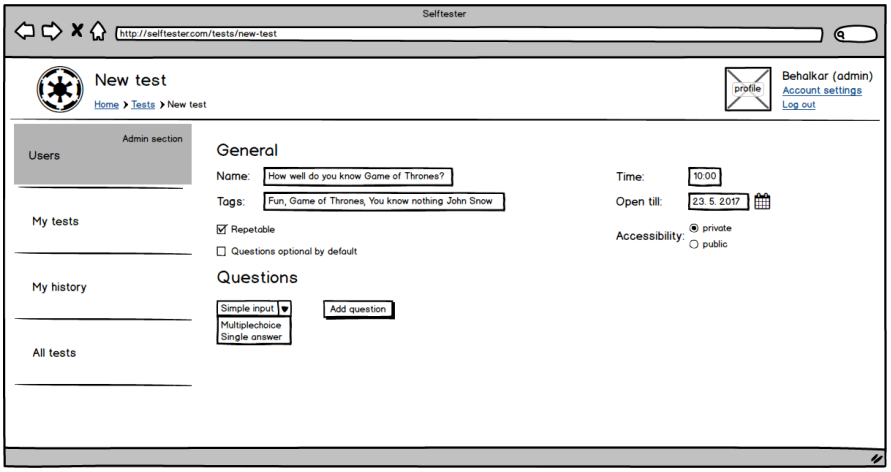
Obrázek B.3: Přehled výsledků

Příloha C

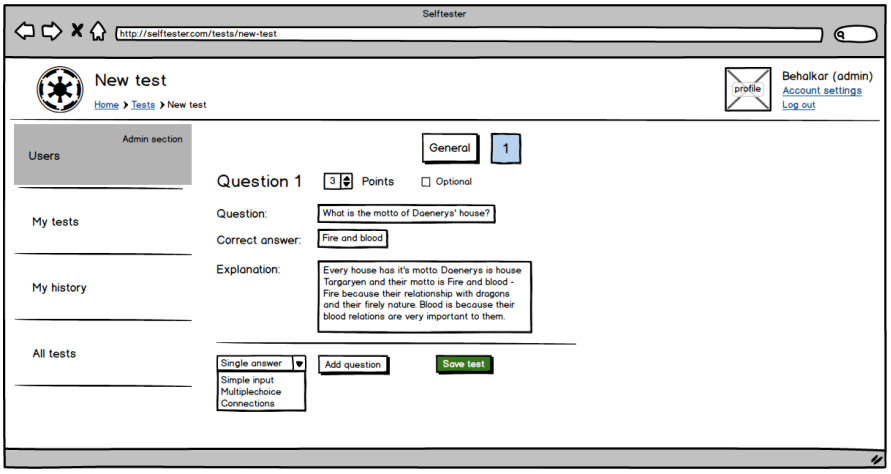
Obrazovky k případu použití Vytvořit veřejný test



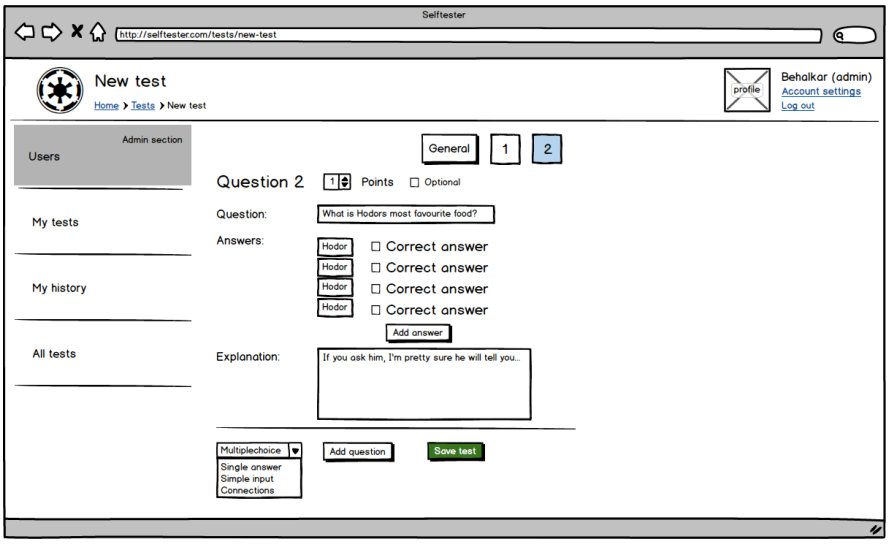
Obrázek C.1: Úvodní stránka administrace



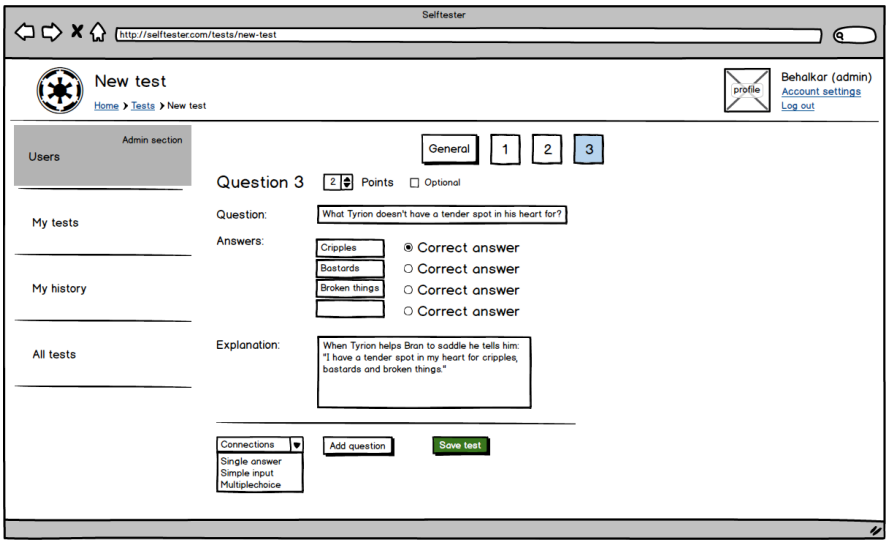
Obrázek C.2: Základní údaje testu



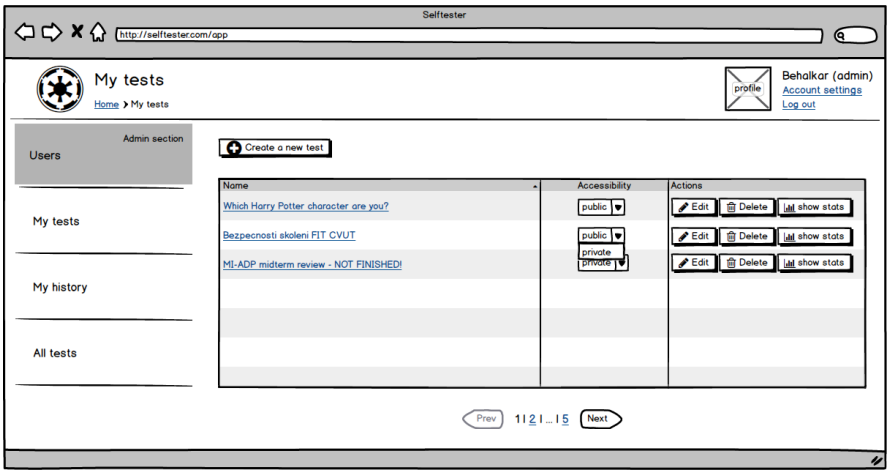
Obrázek C.3: Otázka typu textový vstup



Obrázek C.4: Otázka typu jedna odpověď správně



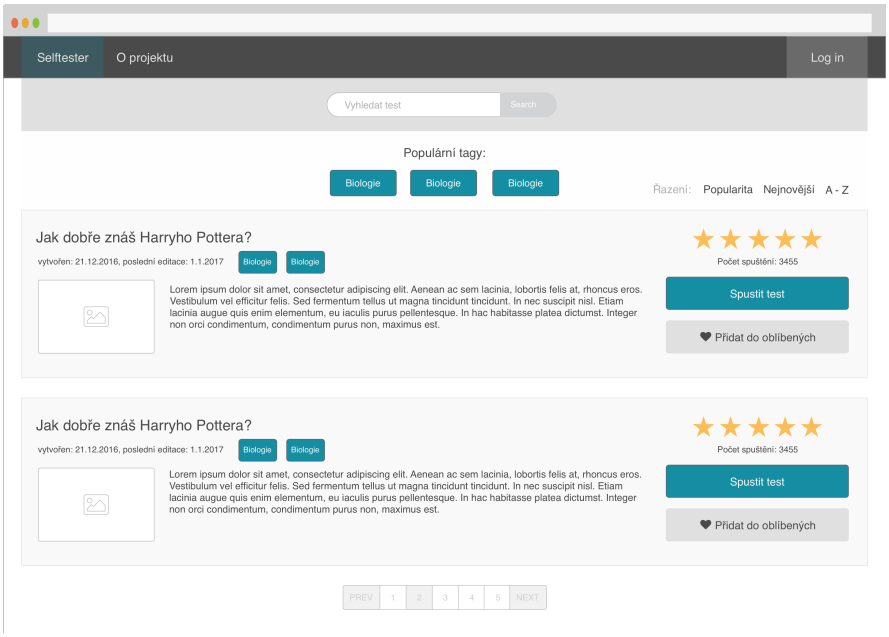
Obrázek C.5: Otázka typu více odpovědí správně



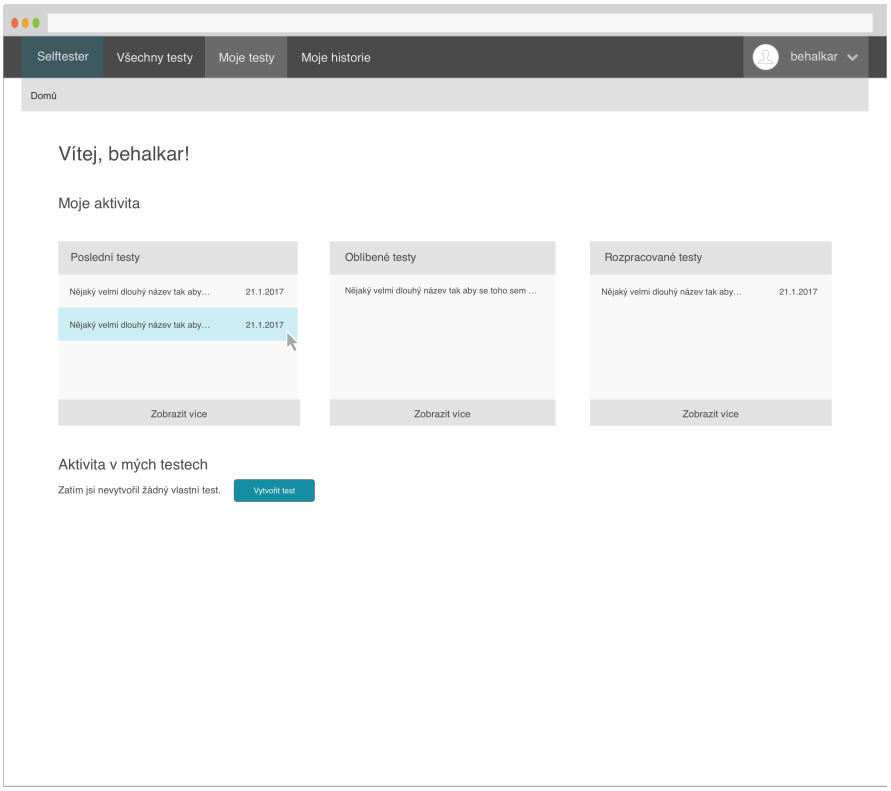
Obrázek C.6: Seznam editovatelných testů

Příloha D

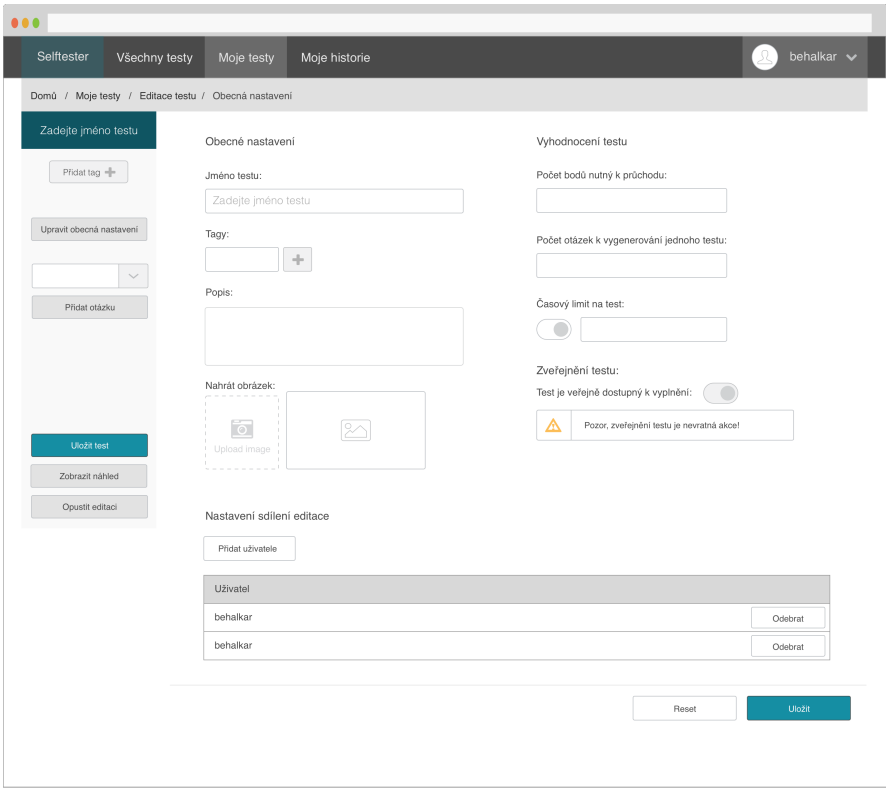
Grafické návrhy



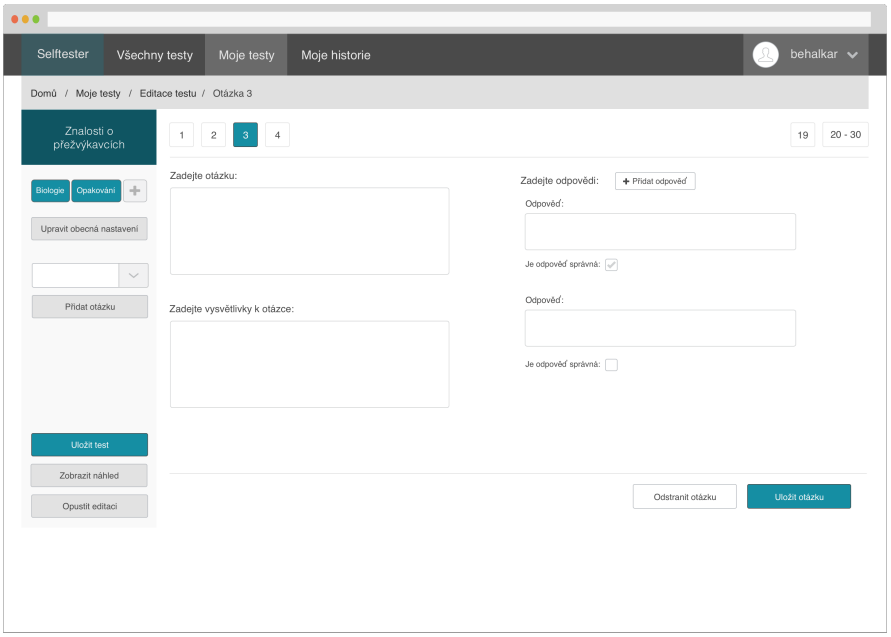
Obrázek D.1: Úvodní stránka



Obrázek D.2: Úvodní stránka administrace - po přihlášení



Obrázek D.3: Editor testu - obecné údaje



Obrázek D.4: Editor testu - editace otázky

Příloha **E**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `uzivatelske-testy.zip` ... Záznamy obrazovky z uživatelského testování
- `lo-fi-prototyp.pdf` Kompletní lo-fi prototyp (wireframes) uživatelského rozhraní