FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Myšák - Webový klient pro vzdělávací aplikaci
Student:	Radek Ježek
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Myšák je týmový projekt zaměřený na vývoj výukové a diagnostické aplikace pro děti předškolního věku v prostředí WWW a OS Android. Cílem praktické části je realizovat prototyp webového rozhraní sloužícího ke správě a přiřazování scénářů ve vzdělávací aplikaci Myšák.
1. Analyzujte požadavky zadavatele a zpracujte je ve formě funkčních a nefunkčních požadavků.
2. Proveďte analýzu protokolů a technologií používaných k API a zvolte nejvhodnější (REST, RPC, GraphQL).
3. Navrhněte a otestujte API pro komunikaci mezi frontendem a backendem ve zvolené technologii (backendová část není součástí této práce).
4. Pomocí metod softwarového inženýrství navrhněte frontendovou část aplikace a uživatelsky přívětivé rozhraní.
5. Dle návrhu implementujte prototyp webového rozhraní aplikace Myšák ve zvolených technologiích.
6. Aplikaci podrobte vhodným testům (uživatelské, API).

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 10. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Myšák – Webový klient pro vzdělávací aplikaci

Radek Ježek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Chludil

6. května 2019

Poděkování

Rád bych poděkoval vedoucímu své práce Ing. Jiřímu Chludilovi, za pomoc a motivaci při tvorbě této práce a Bc. Ondřeji Brémovi, za konzultace ohledně uživatelského testování. Další poděkování učitelkám v mateřské škole Lvíčata, za jejich pomoc a ochotu při testování. V neposlední řadě bych chtěl poděkovat Ing. Elišce Šestákové za cenné rady a svým rodičům za podporu a trpělivost při tvorbě této práce. Závěrem bych chtěl poděkovat autorovi Freepik ze služby flaticon.com za poskytnutí ikon pro vývoj a testování aplikace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl 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.

V Praze dne 6. května 2019 ……

Odkaz na tuto práci

Ježek, Radek. *Myšák – Webový klient pro vzdělávací aplikaci.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce řeší návrh, implementaci a testování webové administrace pro správu scénářů ve výukové platformě pro děti s názvem *Myšák*. Aplikace umožňuje vytvářet a přiřazovat dětem scénáře složené z implementovaných pracovních listů navržených psychologem. Listy jsou zaměřené na různé oblasti vývoje dítěte, např. zrak, sluch, motorika apod. Součástí práce je analýza a výběr vhodných technologií a protokolů API, které slouží pro komunikaci mezi webovým klientem a serverem. K implementaci webového rozhraní byl použit framework Angular pro JavaScript společně s komponenty Angular Material. Výsledky práce umožňují učitelům v mateřských školách efektivně a cíleně využít mobilní dotyková zařízení ke zlepšování schopností a dovedností dětí.

Klíčová slova webový klient, návrh uživatelského rozhraní, analýza API, vzdělávání dětí, mateřské školy, REST, Angular

Abstract

This thesis focuses on design, implementation, and testing of web administration for management of scenarios in the educational platform for children called *Mouse*. The application allows to create and assign scenarios to children. Scenarios consist of implemented worksheets designed by a psychologist. Worksheets are focused on different areas of children's skill development – i.e., vision, hearing, motor skills, etc. The thesis contains analysis and selection of appropriate API technologies and protocols for communication between web client and server. To implement web client, framework Angular for JavaScript was selected, together with Angular Material components. Results of this thesis allow teachers in kindergartens to effectively use mobile touch devices to improve children's skills and abilities.

Keywords web client, user interface design, API analysis, children education, kindergartens, REST, Angular

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Do mateřských škol se dnes stále častěji dostávají nejrůznější chytrá zařízení, kterými jsou velmi často dotykové tablety. Motivací k nákupu těchto tabletů bývá nejčastěji jejich využití k výuce. Na Fakultě informačních technologií Českého vysokého učení technického v Praze již v minulosti existovalo několik projektů, mobilních aplikací, které měly společný cíl – rozvoj a vzdělávání dětí předškolního věku.

V rámci softwarového týmového projektu, na kterém jsem pracoval, jsme pro zadavatele, firmu TechSophia, s. r. o., zabývající se vývojem vzdělávacích aplikací, vyvinuli mobilní aplikaci obsahující úkoly (minihry) vytvořené na základě pracovních lisů navržených psychologem. Přímo v aplikaci se pak v rozhraní pro dozor z těchto úkolů dá poskládat jakýsi scénář, tedy posloupnost, ve kterém se budou dítěti úkoly spouštět. Jednotlivé úkoly jsou zaměřené na různé oblasti vývoje dítěte, např. zrak, sluch nebo myšlení a řeč. Lze tedy např. vybrat několik souvisejících úkolů, poskládat z nich scénář a ten přiřadit dítěti.

Nevýhodou této aplikace však je, že se scénáře musí vytvářet a přiřazovat individuálně, každému dítěti zvlášť a navíc přímo na zařízení, na kterém dítě hraje. Navíc si děti nemohou tablet vyměnit, aniž by přišly o rozehraný scénář. To je v mateřské škole, kde je více dětí a tabletů, značná nevýhoda. Pouze nastavení a spuštění scénářů na každém tabletu zvlášť je obtížné a časově náročné. Proto jsme se s týmem, zadavatelem a vedoucím práce rozhodli tuto aplikaci rozšířit o informační systém umožňující vytváření, spravování a upravování scénářů vzdáleně. Mým úkolem je pro tento systém vytvořit webové uživatelské rozhraní, které usnadní učitelům v mateřských školách práci se scénáři.

Práce je rozdělena do čtyř kapitol: analýza, návrh, realizace a testování. První kapitola se nejprve zabývá analýzou požadavků zadavatele a přístupů k rozhraní mezi klientem a serverem (zkráceně API, z anglického Application Programming Interface), dále jsou popsány technologie požadované zadavate-

lem a nakonec zanalyzovány webové prohlížeče. Na analýzu navazuje návrh, který se věnuje uživatelskému rozhraní, API, zvoleným technologiím pro vývoj a architektuře webové aplikace. V kapitole věnované implementaci lze najít souborovou strukturu aplikace, podrobnější popis architektury a zajímavé části kódu. V poslední kapitole se práce zabývá testováním výsledné aplikace a to hned čtyřmi různými způsoby: testování serverové implementace API, jednotkové (unit) testy, end-to-end testy a uživatelské testy.

Cíl práce

Hlavním cílem práce je navrhnout, implementovat a otestovat prototyp webového rozhraní sloužícího ke správě a přiřazování scénářů ve výukové aplikaci *Myšák*. Jeho dosažení napomáhá analýza funkčních a nefunkčních požadavků zadavatele, porovnání různých metod komunikace mezi serverem a klientem a návrh a testování aplikačního rozhraní.

Kapitola 1

Analýza

Tato kapitola se po uvedení do problematiky zaměřuje na analýzu a specifikaci funkčních a nefunkčních požadavků zadavatele, způsoby užití a analýzu přístupů a metod pro tvorbu aplikačního rozhraní. V závěru jsou analyzovány technologie a podporované prohlížeče vyplývající z požadavků.

1.1 Projekt Myšák

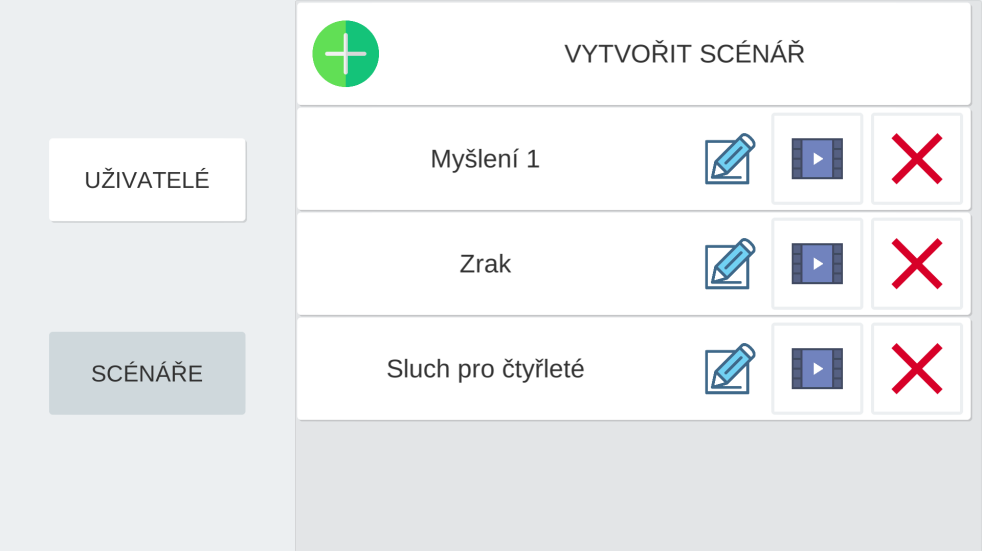
Jedná se o softwarový týmový projekt, krerý si klade za cíl umožnit učitelům v mateřské škole dětem hraní organizovat. Tedy zvolit, které oblasti dovedností mají děti procvičovat. K tomu má sloužit možnost vytvářet scénáře (posloupnosti miniher) a ty přiřazovat jednotlivým dětem.

V současné době byl již vyvinut prototyp mobilní aplikace pro platformu Android, obsahující jednotlivé minihry implementované na základě psychologem navržených pracovních listů. Je tedy předem známo, které oblasti vývoje (např. zrak nebo jemná motorika) tyto minihry procvičují. Učitel tak bude mít možnost vytvořit scénáře zaměřené na jednu nebo více konkrétních oblastí vývoje dítěte. To je v České republice unikátní mechanika s potenciálním pozitivním dopadem na rozvoj dětí.

1.1.1 Prototyp mobilní aplikace

Zmiňovaný prototyp mobilní aplikace *Myšák* se již ve svém brzkém stádiu vývoje pokouší individuální přiřazování scénářů vyřešit. Umožňuje přímo v aplikaci přejít do tzv. „rozhraní pro dozor“ (viz obrázek 1.1), kde lze vytvářet scénáře a ty přiřazovat dětem. Aplikace má však v současné době několik zásadních nevýhod:

- je určená pouze pro lokální práci na tabletech se zařízením Android,
- umožňuje pouze individuální nastavení scénářů jednotlivým dětem, učitel tak musí nastavovat scénáře na každém zařízení zvlášť,



Obrázek 1.1: Prototyp mobilní aplikace – rozhraní pro dozor

- scénáře nelze přenášet mezi zařízeními,
- uživatelské účty dětí nelze přenášet mezi zařízeními,
- děti nejsou organizovány do skupin.

Prototyp nicméně, hlavně díky testování dětmi, dobře posloužil jako jednoduché ověření, zda vytváření, přiřazování a především hraní scénářů bude fungovat v praxi.

1.2 Požadavky

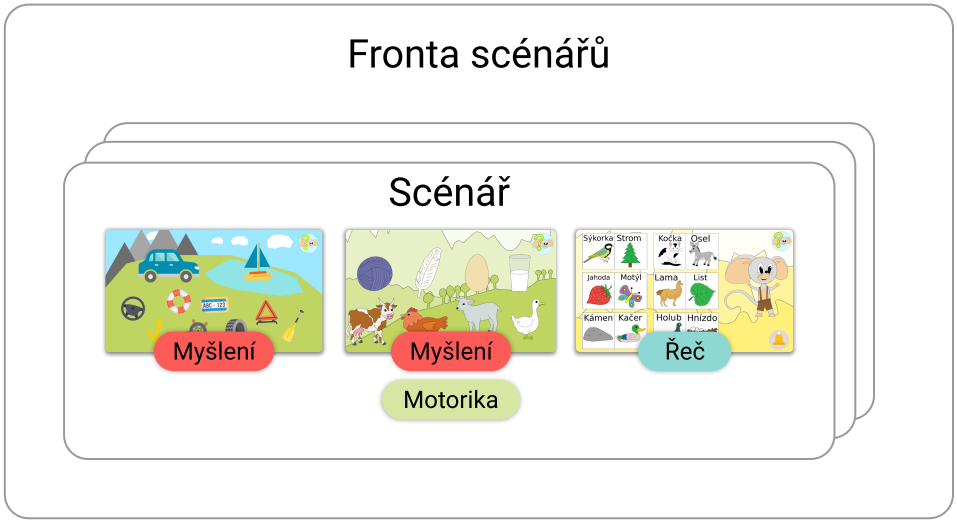
Aplikace je vyvíjena v rámci týmového projektu pro firmu TechSophia, s. r. o., která se zabývá vývojem výukových her a aplikací pro děti zejména předškolního věku. Tato sekce obsahuje požadavky na celý systém, dále se pak práce věnuje pouze těm, které se týkají webového rozhraní pro učitele.

1.2.1 Vysvětlení pojmů

Klíčové pojmy pro aplikaci (názorně zobrazeny na obrázku 1.2):

kategorie – oblast vývoje dítěte (např. myšlení, řeč, motorika, zrak);

úkol – typ úlohy, digitální pracovní list navržený psychologem, zařazený do jedné nebo více kategorií (například třídění nebo spojování);



Obrázek 1.2: Základní koncepty výukové aplikace

konfigurace úkolu – konkrétní realizace úkolu s pevně danou obtížností (například třídění objektů do auta nebo na loď), na rozdíl od úkolu je konfigurace již konkrétní minihra, kterou může dítě hrát. Úkol je pouze obecný a obsahuje více různých konfigurací. Na obrázku 1.2 jsou znázorněné pouze konkrétní konfigurace;

scénář – posloupnost úkolů nebo konfigurací v pořadí, v jakém se budou dítěti úkoly spouštět;

fronta scénářů – posloupnost scénářů v pořadí v jakém se budou dítěti scénáře spouštět.

Zde je důležité podrobněji se zaměřit na význam scénáře v různých kontextech. V navrhovaném webovém uživatelském rozhraní totiž scénář může obsahovat jak obecné úkoly, tak jejich konkrétní realizace. Scénář tedy může obsahovat například *třídění (úkol), spojování (úkol),* auto a loď *(konfigurace úkolu třídění)*. To znamená, že učitel si nebyl jistý nebo nechtěl vybírat konkrétní konfiguraci a namísto toho ponechal výběr na systému (zvolil například pouze *třídění* namísto konkrétního *třídění kuliček*).

V mobilní výukové aplikaci však musí dojít ke spuštění konkrétní konfigurace. Samotný výběr konfigurace je předmětem backendové části systému a zatím nebyl specifikován způsob, kterým se vybírá. Pro prototyp budou konfigurace vybírány zcela náhodně, výsledný systém by mohl přihlížet k předchozím výsledkům dítěte. Tato otázka je ponechána na budoucí vývoj projektu.

1.2.2 Metody pro sběr požadavků

Sběr požadavků probíhal při dohodnutých schůzkách s Mgr. Tomášem Sýkorou [1], který zastupoval firmu TechSophia, s. r. o. Některé implementační části byly diskutovány také s Bc. Tomášem Doubravským, který se zabývá technickou realizací aplikací.

1.2.2.1 FURPS

Pro zachycení představy zadavatele o výsledném produktu byla zvolena analýza požadavků metodou FURPS (Functionality, Usability, Reliability, Performance, Supportability). Tato metoda rozděluje požadavky celkem na pět kategorií, v první z nich jsou funkční požadavky a následují čtyři kategorie nefunkčních požadavků:

functionality – funkční požadavky

usability – použitelnost

reliability – spolehlivost

performance – výkon

supportability – rozšiřitelnost, udržovatelnost

1.2.2.2 Postup sběru požadavků

Popisu průběhu a doporučením pro jednotlivé fáze se věnuje článek Karla Wiegerse [2], kterým byl proces inspirován. Sběr požadavků probíhal v následujících čtyřech krocích:

zjišťování – získávání požadavků od zadavatele

analýza – rozebrání požadavků stadardizovanými způsoby

specifikace – zachycení požadavků do textu a digramů

verifikace – ověření požadavků společně se zadavatelem

Tento postup byl několikrát opakován, dokud nedošlo ke vzájemnému schválení. Požadavky jsem konzultoval nejen se zadavatelem, ale také s ostatními členy týmu, z toho důvodu byly některé části postupně upraveny. V této práci je obsažena pouze poslední verze požadavků.

1.2.3 Specifikace

Aplikace Myšák slouží pedagogům k diagnóze vývoje dovedností dětí předškolního věku. Projekt se skládá z mobilní dotykové aplikace pro operační systém (zkráceně OS, z anglického Operating System) Android, která obsahuje implementované pracovní listy navržené psychology, ze kterých se sbírají statistiky pro každé dítě. Aplikace umožňuje z těchto pracovních listů vytvořit scénáře (posloupnosti). Navrhovaný systém by je měl umožnit pedagogům vytvářet a přiřazovat vzdáleně pomocí webového rozhraní. Přehled požadavků zadavatele podle metody FURPS je zobrazen v tabulce 1.1.

Functional	F1: Přiřazení scénářů dětem F2: Správa a vytváření scénářů F3: Hraní scénáře v mobilní aplikaci
Usability	N1: Podpora hlavních prohlížečů a OS Android v rozumné verzi N2: Podpora stávající Android aplikace Myšák
Reliability	N3: Uptime: $\geq 95\,\%$, Downtime: max. 6 hodin
Performance	N4: V jednu chvíli může pracovat nejvýše 1 000 lidí
Supportability	N5: Umožnění podpory a údržby třetí stranou N6: Umožnění správy scénářů i v jiných výukových aplikacích N7: Možnost budoucí integrace do existujícího diagnostického nástroje N8: Integrace s existujícím systémem správy účtů

Tabulka 1.1: Tabulka funkčních a nefunkčních požadavků podle FURPS

F1 – systém umožňuje vzdáleně a hromadně přiřadit herní scénáře dětem. Systém bude dělit scénáře podle výukových aplikací (viz N6), nelze kombinovat scénáře a úkoly napříč aplikacemi. Děti mohou být rozděleny do skupin, v takovém případě musí být umožněno přiřazovat scénáře celé skupině najednou;

F2 – aplikace umožňuje vytvářet a upravovat scénáře skládající se z úkolů nebo konfigurací. Pokud se ve scénáři nachází úkol, při spuštění scénáře vybere systém jeho příslušnou konfiguraci automaticky;

F3 – po otevření mobilní aplikace dítětem se mu spustí první scénář, který má ve své frontě;

N1 – aplikace bude podporovat nejpoužívanější prohlížeče Google Chrome [3] a Firefox [4] ve verzích vyplývajících z analýzy prohlížečů 1.6;

N2 – systém rozvíjí a navazuje na již existující výukovou Android aplikaci Myšák vyvinutou v předmětu BI-SP1;

N5 – požadavky na použité technologie: frontend – Angular (JavaScript framework) [5], backend – Django (python framework) [6];

N6 – systém umožňuje oddělenou správu scénářů ve více různých výukových aplikacích pro OS Android;

N8 – integrace se stávajícím systémem pro správu účtů na serverech TechSophia, s. r. o.

Součástí požadavků není úprava, mazání a přidávání dětí, ani autentizace a přihlašování učitelů. Děti a skupiny totiž pocházejí z jiných systémů zadavatele a jsou spravovány centrálně v jiné aplikaci. Autentizace učitelů je na přání zadavatele (ze stejného důvodu) vynechána. Způsob přihlášení zatím nebyl specifikován a pro prototyp není podstatný. Přihlášení totiž může probíhat v jiné aplikaci zadavatele a poté dojde pouze k předání autentizačních údajů.

1.2.4 Způsoby užití

Tato část se věnuje uživatelům systému a jejich práci s aplikací. UML (z anglického Unified Modeling Language) diagram na obrázku 1.3 zobrazuje případy užití vázané k jednotlivým rolím uživatelů. Uživatelé se dělí do dvou rolí:

- učitel, který vytváří a přiřazuje dětem scénáře,
- dítě, které následně přiřazené scénáře hraje.

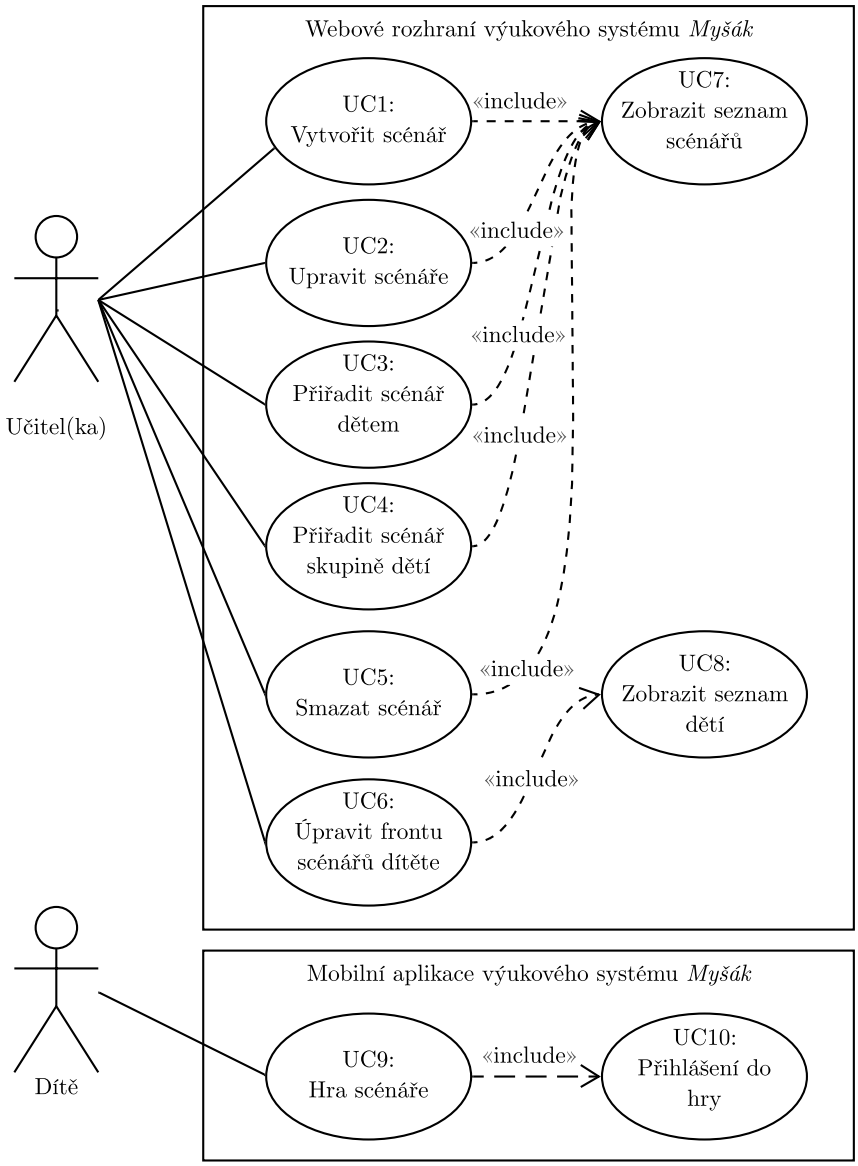
Případy užití jsou popsány několika body (byly vybrány nejdůležitější body z knihy [7, str. 4–5]):

vstupní podmínky – předpoklady, které musí být splněny, aby případ mohl nastat

konečný stav – výsledný stav systému po provedení případu užití

scénář – průběh interakce uživatele se systémem za účelem dosažení konečného stavu

Tabulka 1.2 zobrazuje jakým požadavkům jednotlivé případy užití odpovídají (byly vynechány vložené případy UC7 a UC8, které jsou zahrnuté v ostatních).



Obrázek 1.3: Model případů užití

	Funkční požadavky		
Případy užití	F1	F2	F3
UC1		↑	
UC2		↑	
UC3	↑		
UC4	↑		
UC5	↑		
UC6	↑		
UC9			↑

Tabulka 1.2: Matice požadavků a případů užití

Všechny případy užití se nachází v příloze této práce B. Zde je pro ilustraci pouze první z nich – vytvoření nového scénáře:

UC1 – Vytvořit scénář

vstupní podmínky – uživatel je přihlášen

konečný stav – v systému je nový scénář

způsob vyvolání – uživatel se rozhodne vytvořit nový scénář

scénář

1. *include* (UC7: Zobrazit seznam scénářů)
2. uživatel klikne na *Vytvořit scénář*
3. systém zobrazí rozhraní pro tvorbu scénáře
4. uživatel uloží scénář

1.3 API

Pro vývoj softwarového projektu je typické dělení celkové funkcionality do několika nezávislých částí. Vychází to z přirozené povahy lidského myšlení a potřeby dekompozice problémů. Tento přístup má mnoho výhod:

- vývoj jednotlivých částí může probíhat odděleně,
- lze zaměnit jednu část bez většího dopadu na zbytek systému,
- jednotlivé části lze testovat samostatně.

Princip dělení celku na části je pro software natolik fundamentální, že se zpravidla uplatňuje hierarchicky, prakticky všude, kde je to možné.

I výuková platforma *Myšák* bude rozdělena do více částí (podrobněji rozebráno později v sekci 1.4). Tyto části spolu komunikují prostřednictvím aplikačního rozhraní API (Application Programming Interface). Cílem této sekce je analyzovat různé způsoby a přístupy komunikace mezi serverem (backendem) a webovým klientem (frontendem):

backend – realizuje ukládání, zpracování dat a další výpočty, tzv. byznys logiku, dále také označovaný jako *server*;

frontend – realizuje komunikaci s uživatelem, zobrazuje a manipuluje s daty prostřednictvím backendu. Dále také označovaný jako *klient*, v případě aplikace *Myšák* se jedná o *webového klienta*.

1.3.1 Komunikace mezi podsystémy

Části systému jako backend a frontend jsou typicky vyvíjeny odděleně, každá z nich přitom může být nasazena na jiném fyzickém serveru, případně běžet přímo v prohlížeči uživatele. Proto veškerá komunikace mezi nimi téměř vždy probíhá po síti. Ačkoliv metody pro síťová API většinou umožňují použít různé protokoly (SMTP – Simple Mail Transfer Protocol, HTTP – Hypertext Transfer Protocol, TCP – Transmission Control Protocol), pro webová API je zdaleka nejrozšířenější a nejvýhodnější použití protokolu HTTP. (Jedním z důvodů je i fakt, že porty, na kterých funguje jsou ve výchozím stavu povoleny v každém firewallu). Proto se pro účely této analýzy omezím právě na HTTP.

1.3.2 Srovnání architektur API

Pro realizaci API po síti se používají nejrůznější způsoby. Většinu z nich lze rozdělit do tří základních přístupů: RPC (vzdálené volání procedur, z anglického Remote Procedure Call), REST (z anglického Representational State Transfer) a nově GraphQL (QL v názvu pochází z anglického Query Language neboli dotazovací jazyk). Na to, který z nich je nejlepší, se stále vedou vášnivé debaty, některé zdroje mezi nimi dokonce vidí jakýsi lineární vývoj, např. [8] tvrdí, že „GraphQL je lepší REST“. Pravda je však složitější, stejně jako v mnoha jiných případech při návrhu softwaru neexistuje žádné nejlepší řešení a pro každý projekt je potřeba zvolit přístup, který nejvíce vyhovuje jeho povaze. „There is no silver bullet.“[1] (dle [9]). Proto se tato sekce věnuje právě výběru vhodného stylu API pro propojení backendu a frontendu vyvíjené aplikace.

[1] Volně přeloženo jako „Neexistuje žádné jednoduché řešení“

1.3.2.1 Srovnávací metodika

Srovnávací metodiku jsem vytvořil tak, aby reflektovala požadavky na aplikaci a zároveň, vzhledem k omezeným časovým možnostem, umožňovala rychlý a efektivní postup v projektu. Jednotlivé kategorie mají přiřazenou míru důležitosti – váhu a bodové hodnocení na stupnici od 1 do 5 ve smyslu 1 – nejhorší, 5 – nejlepší. Výsledné skóre pro každou technologii se pak počítá jako vážený průměr:

$$\overline{x} = (\sum_{i=1}^{n} v_i b_i) / \sum_{i=1}^{n} b_i$$

kde v_i je váha kategorie a b_i je bodové ohodnocení dané technologie v kategorii i. Na počátku je vždy počet bodů 5, v jednotlivých kategoriích jsou pak vyjmenované bodové srážky (v závorce za názvem kategorie je váha):

pracnost (4) – odhadovaná složitost návrhu a implementace API vzhledem ke zkušenostem mým i ostatních členů týmu:

- složitější implementace (−1b)
- složitější návrh (−2b)
- vstupní bariéra (málo zkušeností v týmu) (−1b)

caching (2) – využívání lokálního úložiště při opakovaných dotazech oproti neustálému posílání dotazů na server, zde může nastat právě jedna z následujících možností:

- vůbec nepodporuje cache (−4b)
- nepodporuje HTTP cache (−3b)
- nedostatečně podporuje HTTP cache (−2b)

provázanost (3) – jak úzce je svázaný frontend a backend:

- silná provázanost klienta a serveru při vývoji (−1b)
- nedostatečný kontrakt a datové typy (−1b)
- API není prozkoumatelné (nepodporuje introspekci – zjištění všech zdrojů a akcí, které API nabízí) (−1b)
- klient se musí sám rozhodovat o dalším postupu – nedozví se o možných akcích (−1b)

efektivita (2) – jak moc přenos dat zatěžuje síťové spojení klienta:

- v určitých situacích se posílá příliš málo (underfetching) nebo naopak moc dat (overfetching) (−2b)
- neefektivní datový formát (−2b)

použití (3) – kde se toto API nejčastěji používá a subjektivní shoda této oblasti s navrhovaným systémem *Myšák* na škále 1–5. Údaje a doporučení o použití jsou uvedeny např. v [10] nebo [9].

1.3.2.2 Vzdálené volání procedur (RPC)

je nejstarší a nejjednodušší přístup. Z názvu vyplývá, že je tento styl orientován procedurálně. Jedná se o prosté volání funkcí, které server vystavuje a odeslání návratové hodnoty zpět klientovi.

RPC lze implementovat pouze nad jedním endpointem (koncovým bodem), kdy volanou funkci a její argumenty specifikuje klient v těle požadavku typu POST. Pro RPC spojené s HTTP je však typické uvádět v URL (z anglického Uniform Resource Locator) název metody. Argumenty se pak předávají opět v těle požadavku typu POST nebo pomocí HTTP parametrů v URL. Výpis kódu 1.1 ukazuje příklad použití přístupu RPC.

Základní jednotkou je *funkce*.

pracnost – myšlenka, která stojí za RPC je poměrně jednoduchá. V praxi existuje několik různých frameworků a technologií, které ji implementují (za zmínku stojí např. gRPC nebo Apache Thrift – viz [9]). Implementační náročnost tedy zahrnuje seznámení se s příslušným frameworkem, není potřeba žádné výrazné snahy na pochopení konceptu, neboť princip funkcí je každému programátorovi dobře známý. Návrh je subjektivně jednodušší, průměrně složité může být navrhnout API škálovatelné a zařídit správnou funkčnost cache. (4b)

caching – využití HTTP cache je prakticky nemožné, pokud je API implementováno pouze pomocí POST požadavků nad jedním endpointem. Situace je lepší, pokud je API navrženo tak, že se využívá požadavků typu GET a parametry se předávají přímo v URL. (3b)

Požadavek:

```
POST /getDepartment HTTP/1.1
HOST: api.example.com
Content-Type: application/json

{ "id": "10" }
```

Odpověď:

```
{
    "departmentId": 10,
    "departmentName": "Administration",
    "locationId": 1700,
    "managerId": 200,
}
```

Výpis kódu 1.1: Ukázka komunikace pomocí RPC [11, upravil autor]

provázanost – je potenciální nevýhodou RPC. Klient musí přesně znát jaké funkce jsou k dispozici a musí se vždy sám rozhodnout jak pokračovat a které akce jsou v daný okamžik legitimní (je zmíněno např. v [12]). To může přinést celou řadu problémů. Klienti mohou obsahovat více logiky a být nekonzistentní. RPC API také typicky není „prozkoumatelné“. Na druhou stranu vzniká mezi klientem a serverem pevný kontrakt a definice datových typů, což snižuje možné dohadování a neporozumění mezi různými klienty a serverem. (2b)

efektivita – může nastat tzv. overfetching (přehnané množství dat), kdy klient dostane více dat než potřebuje, nebo underfetching (nedostatečné množství dat), kdy klient musí naopak vyvolat několik dotazů, aby se dozvěděl vše, co potřebuje. (zmíněno v [8], jako jedna z výhod GraphQL).

Některé starší protokoly z této kategorie (např. SOAP – Simple Object Access Protocol) využívají neefektivní datové formáty (XML, z anglického Extensible Markup Language) a posílají tak mnoho nadbytečných dat. Tím mohou způsobit zahlcení klientů se špatným připojením, zejména v mobilních sítích. Jiné však umožňují využití moderních, efektivnějších datových formátů, např. gRPC využívá moderní protokol Buffers. (3b)

použití – O využití SOAP píše článek [10], ale ve většina případů se dá zobecnit na využití RPC. Používá se často v podnikových apikacích, starších systémech, systémech orientovaných procedurálně (tj. na akce spíše než data) nebo aplikacích a velkých bankovních systémech, které využijí lepší možnosti zabezpečení nabízené např. protokolem SOAP. Také je vhodný pro architekturu microservices.

Používá se tedy v situacích, kdy je nutné navázat mezi klientem a serverem striktní kontrakt a kdy je klient a server vyvíjen pod jednou střechou. Navrhovaný systém *Myšák* je orientovaný spíše datově než procedurálně a nevyžaduje ani míru zabezpečení podobnou bankovním systémům. (2b)

1.3.2.3 REST

Representational State Transfer (REST) se dnes stal téměř standardem při vyvíjení moderních webových aplikací. Revoluční myšlenka, kterou přinesl, je definice zdrojů a operací, které nad nimi lze vykonávat, tedy určitá abstrakce systému zaměřená na data. Ve spojení s HTTP jsou pak jednotlivé zdroje reprezentovány jako URI (Uniform Resource Identifier – unikátní identifikátor zdroje), operace jsou definovány pomocí HTTP sloves (GET, POST, PUT, DELETE apod.) [13].

REST je ale pouze styl architektury, sada doporučení, které vydal Roy Fielding, jeden ze spoluautorů protokolu HTTP, ve své disertační práci v roce

Požadavek:

```
GET /management/departments/10 HTTP/1.1
HOST: api.example.com
Content-Type: application/json
```

Odpověď:

```
{
    "departmentId": 10,
    "departmentName": "Administration",
    "locationId": 1700,
    "managerId": 200,
    "links": [
        {
            "href": "10/employees",
            "rel": "employees",
            "type" : "GET"
        }
    ]
}
```

Výpis kódu 1.2: Ukázka použití REST API [11]

2000 [14]. Pokud se API drží principů, které zde popisuje, může se nazývat tzv. RESTful. Zde nastalo velké nedorozumění a nepochopení vývojáři, protože dnes bývá jako REST označováno téměř jakékoliv API, které využívá zdroje nad HTTP, přestože se vůbec nedrží klíčových principů a omezení. *„Existuje mnoho variant* takzvaných *REST API, to jsou API, které se neřídí všemi požadovanými principy RESTu, ale přesto si dovolují se označovat jako REST“* [15, přeložil autor]. Výpis kódu 1.2 ukazuje příklad použití přístupu REST.

Základní jednotkou je *zdroj*.

pracnost – z implementačního hlediska se jedná o poměrně jednoduchý úkol. REST dnes zná téměř každý programátor, který přišel do styku s webem, tedy i ostatní členové týmu vyvíjející backend ovládají technologie a frameworky sloužící pro tvorbu rozhraní v tomto paradigmatu. Problém však může nastat při návrhu, protože správně dodržet principy a vytvořit opravdové REST API je přinejmenším náročné. V již zmiňovaném článku autor uvádí: *„Vyvinout pravé REST API je neuvěřitelně těžké. Jsou tak vzácné, že kromě mých (Good API) klientů bych je mohl spočítat na dvou rukách.“* [15, přeložil autor]. (3b)

caching – je jednoduché a velmi výhodné využít cache v HTTP, jejíž podpora je dnes vestavěná do většiny prohlížečů. To vyžaduje využití více různých URI, což je přesně přístup, který je pro REST typický. V tomto bodu REST vede oproti technologiím, které používají pouze jeden endpoint a caching musí řešit nějakým jiným, externím způsobem. Caching je zmíněn např. v [15] nebo [12]. (5b)

provázanost – jednou z klíčových vlastností RESTu, jak je popsáno v [13], je hypermedia controls (často se používá zkratka HATEOAS neboli Hypermedia as the Engine of Application State). Zkrátka místo toho, aby byl klient fixován na konkrétní URI adresy, server mu sám nabídne, jakými akcemi může pokračovat a kde se nacházejí. Tedy když se například klient pro online knihovnu dotáže na detaily o nějaké knize, server mu v odpovědi vrátí i informace, jak může pokračovat, třeba *zarezervovat knihu* a příslušný zdroj (URI), pomocí kterého lze akci provést.

Tento princip je klíčový pro oddělení klienta a serveru. Nejen, že lze měnit adresy zdrojů, ale klient se ihned dozví, co lze v daný okamžik vykonat za akce. Nemusí tedy zjišťovat, zda je kniha už rezervovaná, ale pokud je, prostě se v odpovědi možnost *zarezervovat knihu* neobjeví. Na druhou stranu není ve výchozím stavu mezi serverem a klientem vynucen silný kontrakt a datové typy a API není prozkoumatelné. (3b)

efektivita – stejně jako u RPC, může nastat problém under- nebo overfetchingu, zde je však nutné podotknout, že záleží na způsobu, jakým je API vytvořeno. Může být obecné pro veřejnost, nebo optimalizované pro konkrétní použití, kdy se lze těmto problémům vyhnout. To však nic nemění na tom, že např. GraphQL je adresuje mnohem lépe [8]. (3b)

použití – REST se používá napříč všemi obory, zařízeními, nehledě na to, zda jsou tato použití správná. Prakticky by se dalo říci, že je na něm postaven dnešní web. Například ProgrammableWeb uvádí, že zastoupení RESTu mezi API v jejich adresáři je až 84 % [16]. Díky nezávislosti serveru a klienta se REST hodí obzvláště, pokud má systém mnoho klientů, kteří se třeba ani se serverem neznají, pro architekturu microservices nebo pro tvorbu veřejných API.

U obecných API bez konkrétního konzumenta však narážíme na problém již diskutovaného over- a underfetchingu. Díky obrovské paletě služeb, které používají REST se najde mnoho případů, které se podobají nebo téměř shodují s vyvíjeným systémem *Myšák*. (5b)

1.3.2.4 GraphQL

QL v názvu tohoto přístupu je zkratkou *query language* neboli dotazovací jazyk. Je to zcela nový, moderní způsob vyvinutý společností Facebook. Jeho revoluční myšlenka je eliminace over- a underfetchingu tím, že si klient v každém dotazu specifikuje pouze ty atributy objektu, které zrovna potřebuje.

Ve spojení s HTTP je implementovaný pouze nad jedním endpointem a komunikace probíhá prostřednictvím požadavku typu POST, detailně popsáno v [8]. Součástí payloadu tohoto požadavku jsou pak jednotlivé úkony, které se dělí na dva základní typy: query (dotaz) a mutation (úprava dat). Svým přístupem má nejblíže RPC. *„GraphQL je vpodstatě RPC s výchozí procedurou posytující dotazovací jazyk, trochu jako SQL.“* [12, přeložil autor]. Výpis kódu 1.3 ukazuje příklad použití GraphQL.

Základní jednotkou je *dotaz*.

pracnost – každá nová technologie si s sebou nese určité nároky na prvotní seznámení. Zde je to navíc umocněno tím, že se v podstatě jedná o nový jazyk, který má svá pravidla a omezení. Z implementačního hlediska může být i vývoj samotný poněkud pracnější, neboť pro každý atribut každého objektu existuje v systému funkce, která ho „dodává“ [17].

Navrhnout kvalitní API pro různá konkrétní použití je o něco méně obtížné, neboť nemusíme na počátku příliš přihlížet k tomu, jaké bude každý klient potřebovat atributy. Porozumění uživatelským potřebám se však nelze vyhnout. *„S GraphQL můžete oddálit moment zkoumání, jak uživatelé používají vaše API do té doby než začnete profilovat dotazy, vyhodnocovat jejich komplexitu a identifikovat pomalé dotazy.“* [15, přeložil autor]. (3b)

caching – vzhledem k implementaci nad jedním endpointem opět nelze využít HTTP cache a je nutné přistoupit k méně pohodlnému, externímu řešení pomocí frameworků (např. Apollo nebo Relay). (2b)

provázanost – v jazyce GraphQL nic jako princip HATEOAS není, klient se serverem je proto úzce svázaný, na základě pevného kontraktu. Také v odpovědích nejsou dostupné možné akce, tudíž klient musí sám vědět, jak může nebo nemůže pokračovat. Na druhou stranu je API prozkoumatelné a vynucuje dodržování datových typů.

V GraphQL je také poněkud netradičně pojato verzování. Oficiální doporučení a best practice [17] totiž tradiční verzování nedoporučují a namísto toho doporučují přidávat nové atributy k již existujícím dotazům. Článek na `symfony.fi` [18] uvádí jako příklad logo, které na počátku existuje pouze ve formátu png, následně je představen vektorový formát svg a poté, o pár let později, je opět změněno, tentokrát na gif. Zatímco v RESTu by to bylo vyřešeno například třemi různými verzemi, značenými v URI (/v1/, /v2/, /v3/), v GraphQL by tato skutečnost byla naznačena přidáním nových parametrů k objektu, tedy například logo (původní), logo_svg a logo_gif. To může mít různé důsledky, například problém, jak upozorněnit klienty na nové verze, zrušení starých atributů apod. Oproti tomu např. dle článku [19] REST umožňuje odeslat klientům speciální HTTP hlavičku (299 – deprecated API). (3b)

Požadavek:

```
POST /graphql HTTP/1.1
HOST: api.example.com
Content-Type: application/json

query{
  department(id: 10){
    name
    manager{
      firstname
      surname
    }
    employees{
      surname
    }
  }
}
```

Odpověď:

```
{
   "data": {
      "department": {
        "name": "PR",
        "manager": {
          "fistname": "Jan",
          "surname": "Novák"
        }
        "employees": [
          {
            "surname": "Vopršálek"
          }
        ]
      }
   }
}
```

Výpis kódu 1.3: Ukázka použití GraphQL [11, upravil autor]

efektivita – je jedním z klíčových argumentů pro volbu GraphQL. V každé situaci se totiž po síti posílá přesně tolik dat, kolik klient potřebuje, neboť si sám vybírá atributy. To eliminuje overfetching. Fakt, že se lze zanořovat neomezeně hluboko ve všech vazbách mezi entitami (např. pro

danou knihu lze v jednom dotazu zobrazit seznam rezervací a pro každou rezervaci ještě jména lidí, které je mají půjčené) eliminuje underfetching. To vše bez nutnosti specializovat API pro konkrétní použití. (5b)

použití – GraphQL se používá v oblastech, kdy je kladen důraz na efektivitu dat, například pro pomalá mobilní připojení nebo kdy existuje velké množství různých klientů, kteří chtějí pohodlně a efektivně konzumovat jedno společné API. Ne náhodou vznikl tento koncept ve Facebooku, kde je v době psaní této práce podíl mobilních zařízení přes 75 % [20]. Stejně jako RPC se hodí v situacích, kdy je vyžadován formální kontrakt a typový systém.

Technologie je stále mladá, přesto ji již používají velké společnosti, jako např. GitHub, Pinterest nebo již zmiňovaný Facebook. Vyvíjený systém *Myšák* bude dle požadavků konzumovat pouze malé množství klientů ne-li jeden a problém s neefektivitou dat lze tedy vyřešit i např. specializací REST API. Navíc povaha aplikace a jejího použití nevyžaduje přílišnou míru úspory dat. (3b)

1.3.2.5 Shrnutí

Z tabulky 1.3 je vidět, že nejvíce se pro projekt hodí REST. Jedná se o sázku na jistotu prověřenou roky v průmyslu. REST má nativní podporu HTTP

Kritérium	Váha	RPC	REST	GraphQL
Pracnost	4	4	3	3
složitější implementace (−1b)				*
složitější návrh (−2b)			*	
vstupní bariéra (−1b)		*		*
Caching	3	3	5	2
nepodporuje cache (−4b)				
nepodporuje HTTP cache (−3b)				*
obtížné využití HTTP cache (−2b)		*		
Provázanost	3	2	3	3
silná provázanost při vývoji (−1b)		*		*
nedostatečný kontrakt (−1b)			*	
API není prozkoumatelné (−1b)		*	*	
klient se sám rozhoduje (−1b)		*		*
Efektivita	2	3	3	5
over- a underfetching (−2b)		*		*
neefektivní datový formát (−2b)				
Použití	3	2	5	3
Výsledný vážený průměr (zaokr.)		2,9	3,8	3,1

Tabulka 1.3: Výsledek srovnání (bodové srážky označené hvězdičkou)

cache, nejlépe odděluje klienta a server, existuje mnoho doporučení, jak ho verzovat a díky existujícím zkušenostem v týmu a jednoduchosti implementace se nestane brzdou projektu. Problém může nastat ve snaze navrhnout API kvalitní, škálovatelné a zároveň uzpůsobené případům užití, aby nedocházelo k overfetchingu nebo underfetchingu. To je ale mým úkolem v rámci této práce.

1.3.3 REST podle Fieldinga

Roy Thomas Fielding ve své disertační práci [14] zavádí pravidla, která by mělo rozhraní splňovat, aby bylo tzv. RESTful:

client–server – omezení na základní architekturu aplikace;

bezstavovost – každý požadavek musí obsahovat veškeré informace potřebné k jeho vyřešení, server nesmí využívat žádný uložený kontext;

cache – data v odpovědi by měla být explicitně nebo implicitně označena, zda je lze ukládat do mezipaměti nebo ne;

jednotnost – rozhraní by mělo být uniformní, nezávislé na implementaci. Toho je docíleno dodržením několika principů, např. HATEOAS;

systém vrstev – hierarchická kompozice API z více vrstev tak, že každá komponenta vidí pouze vrstvu, se kterou bezprostředně komunikuje. To umožňuje jednoduše přidat např. vyvažování zátěže (load balancing) nebo cache jako jednu z vrstev;

kód na vyžádání – umožňuje rozšířit funkčnost klienta stáhnutím a spuštěním kódu ze serveru.

V praxi se některá z těchto pravidel ukázala jako příliš přísná nebo obtížná na implementaci, více viz sekce návrhu 2.2.1.

1.3.4 Strojově zpracovatelné formáty

REST je nechvalně známý princip návrhu API, nicméně je to pouze sada zmíněných doporučení. Jeho nevýhodou tedy je, že není nijak standardizovaný. Právě proto existují formáty jako JSON API (JSON, z anglického JavaScript Object Notation, je nejčastěji používaný datový formát ve spojení s přístupem REST) a HAL (z anglického Hypertext Application Language). Ty mají za cíl zjednodušit návrh vynucením striktních pravidel a standardizací. Výhodou těchto formátů je, že jsou čitelné pro stroj a potenciálně umožňují jistou úroveň automatizace a snadnější konzumaci API. Jsou však hůře čitelné pro lidi.

Z požadavků vyplývá, že technologiemi pro vývoj jsou JavaScriptový framework *Angular* pro webového klienta a Python framework *Django* pro vývoj serverové části. Proto jsem prozkoumal možnosti využití formátů HAL a JSON API, ve spojení s těmito technologiemi.

Diplomová práce [21, str. 98–100], ve které se autor zabývá implementací REST rozhraní v různých frameworcích pro Python zmiňuje knihovnu, která implementuje HAL – *drf-hal-json*. Z GitHubu [22] tohoto modulu je však očividné, že se nejedná o populární nebo výrazně aktivní projekt a sám autor zmiňované práce odhalil chybu, kterou knihovna obsahovala. Na frontendu je situace podobná, pro Angular existují frameworky *angular-hal* a *angular4-hal*, první z nich přitom ani není kompatibilní s nejnovější verzí Angularu. Situace s JSON API není o moc lepší, knihovny pro Angular i Django sice existují, ale opět nejsou příliš populární ani dostatečně aktivní.

1.4 Architektura systému

Celková architektura systému byla navržena společným úsilím v rámci softwarového týmového projektu. Systém se dá rozdělit na tři základní části:

výuková aplikace – mobilní aplikace pro OS Android obsahující implementované minihry (konfigurace úkolů) pro děti;

webový klient (frontend) – administrativní rozhraní pro učitele umožňující vytváření scénářů a jejich přiřazení dětem. Webový klient je hlavním předmětem této práce;

server(backend) – mozek systému, obsahuje byznys logiku, databázi a integraci ostatních systémů zadavatele.

Na obrázku 1.4 jsou detailněji vidět vazby mezi komponentami, ze kterých se systém skládá.

1 REST API mezi výukovou aplikací a serverem – rozhraní, pomocí kterého zjistí výuková aplikace aktuální konfiguraci, kterou má dítě hrát. Slouží také k odesílání jednoduchých statistik zpět na server;

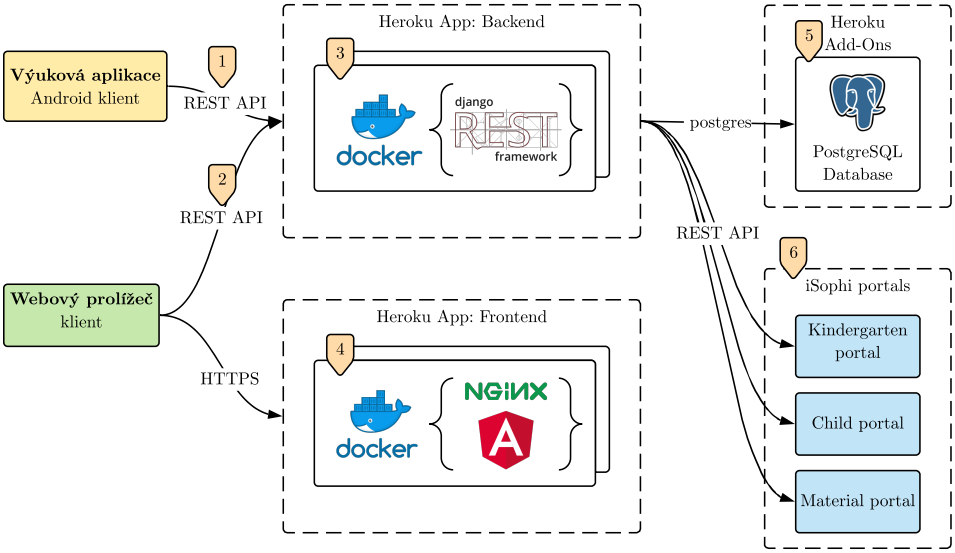
2 REST API mezi webovým klientem a serverem – rozhraní umožňující webovému klientovi získat seznam dětí, scénářů a upravovat fronty scénářů dětí. Návrh tohoto rozhraní je mimo jiné předmětem této práce;

3 Backendový server – výsledná serverová Django aplikace, která vystavuje obě předchozí rozhraní a sama komunikuje s databází a dalšími portály zadavatele;

4 Webový server pro frontend – obsahuje výslednou webovou Angular aplikaci využívající webový server nginx;

5 PostgreSQL databáze – obsahuje data systému;

6 Portály zadavatele – další servery, se kterými systém komunikuje.



Obrázek 1.4: Diagram architektury systému vytvořený backendovým vývojářem Jakubem Topičem [23, zjednodušil autor]

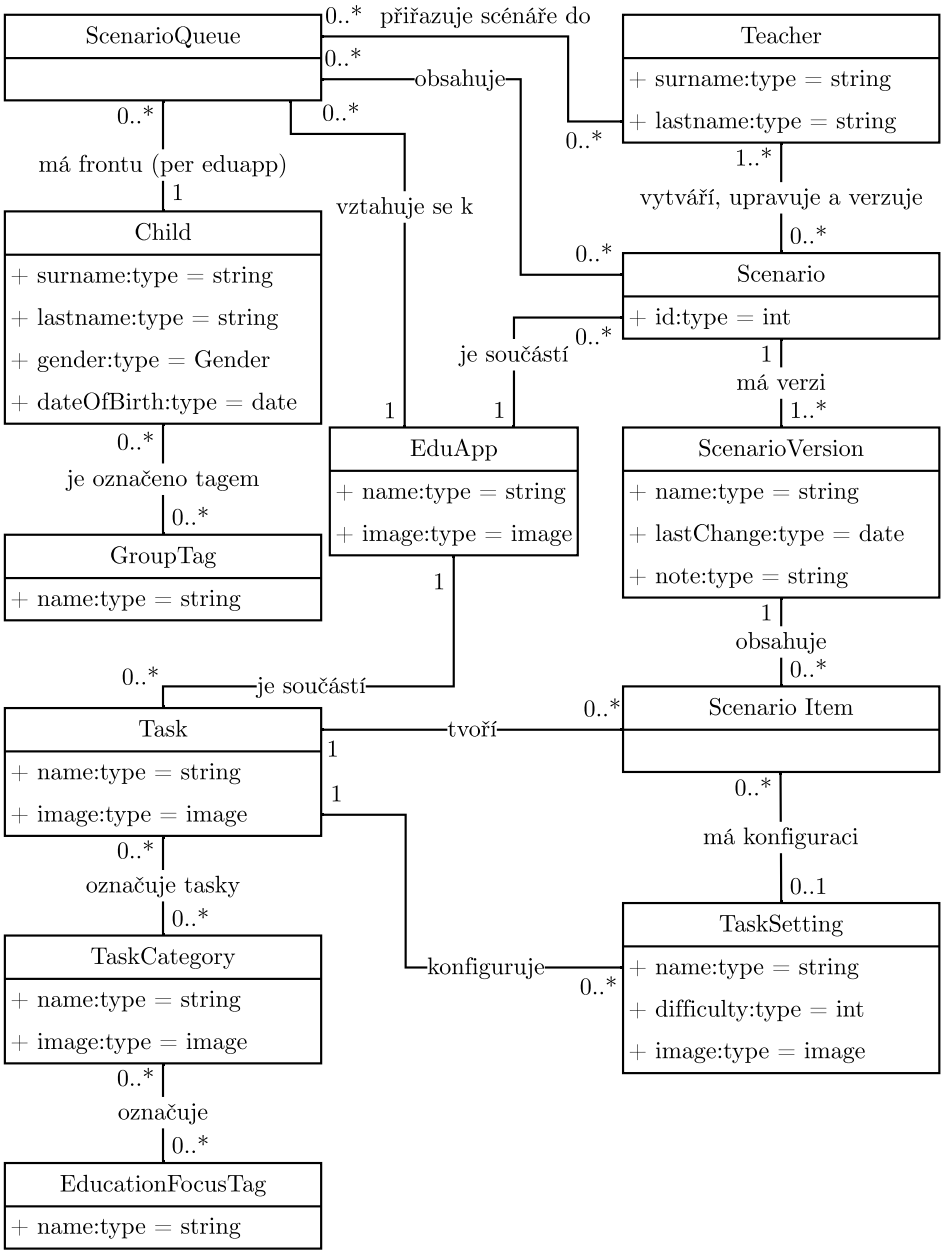
V práci se dále vyskytují pojmy frontend (případně klient, webový klient) a backend (případně server). Nebude-li řečeno jinak, prvním ze zmiňovaných pojmů bude myšlena webová Angular aplikace běžící v prohlížeči (označená jako *4* na obrázku 1.4). Backend bude označovat serverovou Django aplikaci, databázi a další portály zadavatele (čísla 3, 5, 6 na obrázku 1.4).

1.4.1 Doménový model

Pro popis domény a tvorbu databáze vytvořil backendový vývojář Ondřej Šlejtr doménový model (obr. 1.5), který zobrazuje jednotlivé entity a vztahy mezi nimi. Tento model byl později užitečný nejen pro návrh databáze, ale spolu s mnou navrženými modely obrazovek pomohl také při návrhu API.

1.5 Analýza technologií

Volba základních technologií pro tvorbu systému vyplývá z požadavků zadavatele. Tato práce je zaměřená na frontendovou část aplikace, proto se bude nadále zabývat pouze technologiemi určenými pro vývoj webového klienta. Dle požadavku N5 ve specifikaci (sekce 1.2.3) má být webový klient realizován ve frameworku Angular pro JavaScript.



Obrázek 1.5: Doménový model vytvořený backendovým vývojářem [24]

1.5.1 Angular

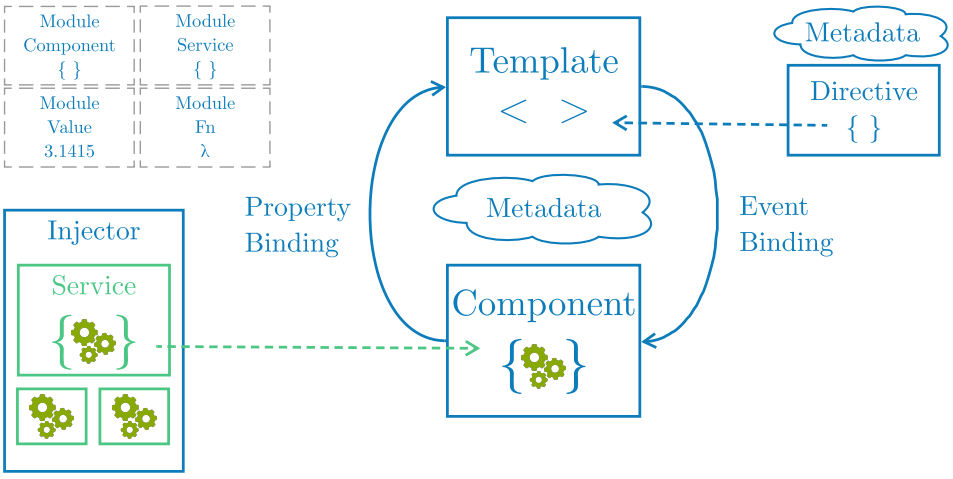
Jedná se o moderní, populární framework, který umožňuje vyvíjet webové aplikace snadno a efektivně. Výhodou frameworku je přehledné logické členění aplikace, mnoho předpřipravených řešení (např. pro routování, cachování apod.) nebo snadná instalace a uvedení do chodu.

Architektura frameworku je znázorněna na obrázku 1.6, který pochází z oficiální dokumentace. Celá Angular aplikace je logicky strukturovaná do komponent (*component*), které jsou do sebe vnořované principem vkládání. Příkladem komponenty může být tlačítko, tabulka, navigační lišta nebo celá stránka. Komponenty se skládají z HTML šablony (*template*), CSS stylů a TypeScript třídy (logika komponenty). Jednotlivé komponenty využívají k získání dat ze serveru služby (*service*), které jsou do nich doplňovány pomocí vkládání závislostí (*dependency injection*). Komponenty jsou sdružované do *modulů*, které představují funkční celek. Podrobnější popis architektury lze nalézt také v [25].

Jednotlivým stavebním kamenům Angularu se později věnuje sekce 3.3 kapitoly realizace, kde lze nalézt i praktické ukázky kódu.

1.5.2 TypeScript

TypeScript je programovací jazyk vycházející z JavaScriptu rozšířený o typovou kontrolu, obsahuje statické datové typy včetně generických tříd a metod. Tento jazyk je kompilován do JavaScriptu, vývoj je tak díky časnému odhalení některých chyb již při kompilaci pohodlnější a efektivnější. Angular ve výchozím stavu používá právě tento programovací jazyk.



Obrázek 1.6: Architektura Angularu [25, překreslil autor]

1.6 Analýza prohlížečů

Vzhledem k zaměření aplikace pouze na české mateřské školy a učitele, se budu nadále zabývat pouze webovými prohlížeči používanými v České republice. Data použitá pro účely této analýzy byla v lednu 2019 zveřejněna na webu `gs.statcounter.com` [26].

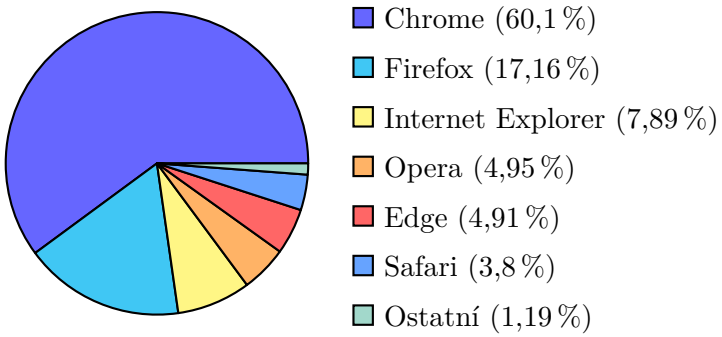
1.6.1 Desktopové prohlížeče

V kategorii desktopových prohléžečů má jednoznačnou převahu Chrome s podílem uživatelů 60,1 %, na druhé příčce je s velkým odstupem Firefox se 17 % uživatelů. Celá statistika je vidět na obrázku 1.7. Přes stálou přítomnost dnes již zastaralého prohlížeče Internet Explorer, bylo po konzultaci požadavků rozhodnuto tento prohlížeč pro jeho množství problémů a nekompatibility nepodporovat.

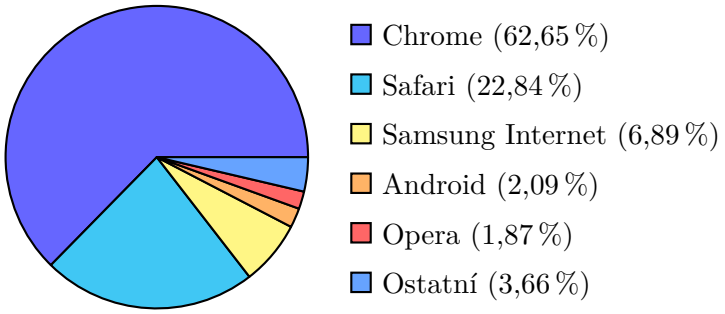
Z desktopových prohlížečů bude aplikace podporovat Google Chrome ve verzích 60+ (v lednu 2019 – nejrozšířenější a nejnovější verze 71) a Firefox ve verzích 52+ (v lednu 2019 – nejrozšířenější a nejnovější verze 64). Tyto verze vyhovují nefunkčním požadavkům zadavatele a zároveň umožňují efektivně a rychle vyvíjet moderní web. Celkem bude tedy podporováno přibližně 68 % desktopových uživatelů. Informace o verzích pocházejí opět z [26].

1.6.2 Mobilní a tabletové prohlížeče

U mobilních prohlížečů má největší podíl opět Chrome s 62,65 %, nicméně na druhé příčce se nyní nachází prohlížeč Safari s 22,84 %. Tento prohlížeč je však dostupný pouze na zařízeních americké společnosti Apple, vzhledem k absenci těchto zařízení pro testování a primárnímu zaměření aplikace *Myšák* na desktopovou a Android platformu, bylo po diskuzi se zadavatelem upuštěno od podpory tohoto prohlížeče. Celá statistika je vidět na obrázku 1.8.



Obrázek 1.7: Podíl desktopových prohlížečů v ČR (graf vytvořil autor dle [27])



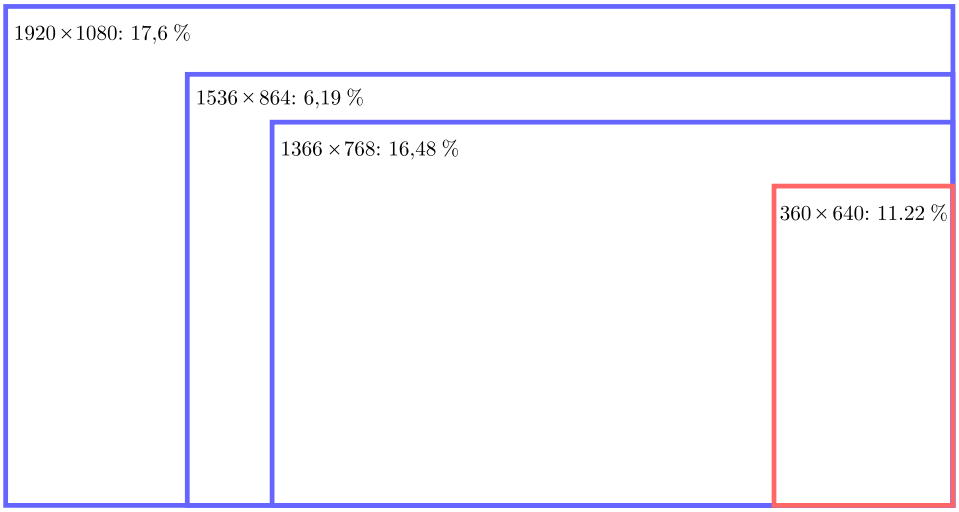
Obrázek 1.8: Podíl mobilních a tabletových prohlížečů v ČR (graf vytvořil autor dle [28])

Z mobilních prohlížečů tak bude podporován pouze Google Chrome. Bohužel u mobilní verze prohlížeče Chrome nejsou snadno dostupné informace o využívání jednotlivých verzí. Optimistický odhad je tedy podpora okolo 60 % uživatelů. Primární cílová skupina aplikace *Myšák* jsou uživatelé desktopu, avšak Google Chrome pro desktop i mobilní telefony a tablety používá v jádru stejný renderovací engine. Z toho důvodu by neměly vzniknout problémy s nekompatibilitou (pouze s různým rozlišením displeje, více později v části 2.1.3 věnující se responzivnímu designu).

1.6.3 Rozlišení obrazovky

Dvě nejčastější desktopová rozlišení jsou 1920×1080 a 1366×768 obrazových bodů, tedy standartní poměr stran 16:9. Vzhledem k primárnímu zaměření na desktopovou platformu bude aplikace optimalizovaná právě pro tato rozlišení.

Nicméně každá webová aplikace by v dnešní době měla být schopná přizpůsobit se různým poměrům stran a rozlišení (uživatelé mění velikost okna, přistupují na stránky stále častěji z mobilních zařízení), z toho důvodu jsem se rozhodl přistoupit k tzv. responzivnímu designu. Ten má za úkol přerovnat prvky na obrazovce tak, aby se stránka pohodlně ovládala při různých rozlišeních na různých zařízeních. Více se tímto tématem zabývám později v sekci 2.1.3. Na obrázku 1.9 jsou znázorněná různá rozlišení obrazovek spolu s jejich zastoupením mezi uživateli v lednu 2019.



Obrázek 1.9: Nejčastější rozlišení obrazovek a jejich zastoupení v ČR, desktopová a tabletová – modře, mobilní – červeně (vytvořil autor dle [29])

Kapitola 2

Návrh

V této kapitole se práce zaměří nejprve na návrh uživatelského rozhraní a modely obrazovek. Druhá část je věnovaná návrhu API a použitým technologiím a na ni navazuje návrh architektury a struktura webové aplikace.

2.1 Návrh uživatelského rozhraní

Při návrhu uživatelského rozhraní jsem se řídil především principy zmíněnými v článku Jakoba Nielsena [30], který se zabývá heuristikami dobrého návrhu uživatelského rozhraní. Patří mezi ně:

viditelnost stavu systému – uživatel by měl být v každém okamžiku informovaný o tom, co systém dělá, např. pomocí ukazatele průběhu, znázornění načítání. Tyto grafické prvky nejsou obsažené v modelech obrazovek, které se nacházejí v další podsekci, nicméně ve výsledném prototypu je načítání a ukládání dat znázorněno;

shoda systému a reálného světa – systém obsahuje termíny a metafory známé z běžného světa, informace jsou logicky seřazené a strukturované;

uživatelská kontrola a svoboda – uživatelé často zvolí některé funkce systému omylem a potřebují jednoduchý způsob, jak akci zrušit, bez nutnosti procházet složitými dialogy;

konzistence a standardy – systém by měl dodržovat konvence dané platformy. K maximalizaci konzistence pro webovou platformu byl klíčový výběr vhodných technologií, především knihovny Angluar Material [31], podrobněji se mu věnuje sekce 2.3.2;

rozpoznání nad pamatováním – minimalizace nutnosti uživatele pamatovat si vlastnosti systému zviditelněním akcí, objektů a vlastností. To je v navrhovaném systému demonstrováno např.s hlavní lištou s navigací;

prevence chyb – ještě lepší než dobré chybové hlášky je pečlivý návrh, který vzniku problému předchází. Tato podmínka se ve výsledném návrhu projevuje například zobrazením hvězdičky u povinných polí (např. při zadávání názvu scénáře) nebo neumožněním zadat název překračující maximální délku;

flexibilita a efektivita používání – zkratky a akcelerátory, často neviditelné pro nového uživatele, mohou urychlit práci zkušeným uživatelům. Aplikace *Myšák* reflektuje tuto heuristiku například možností přidat scénář více dětem najednou;

estetika a minimalistický design – dialogy by neměly obsahovat nepotřebné nebo irelevantní informace. Estetický vzhled a minimalistický design byl docílen především použitím frameworku Angular Material, více v sekci 2.3.2;

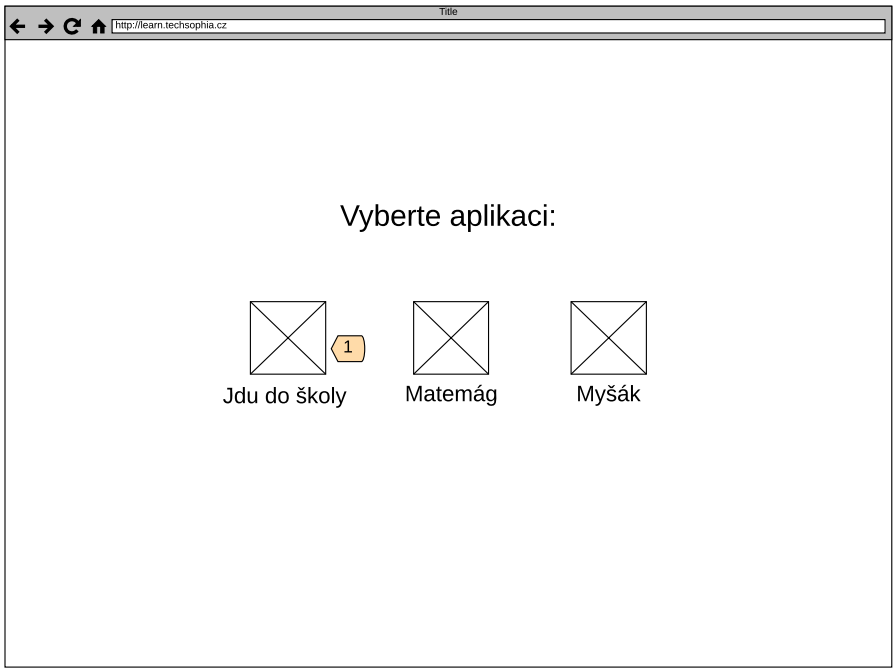
možnost uživatelů diagnostikovat a zotavit se z chyb – chyby by měly být vyjádřeny jednoduchým jazykem (žádné kódy), přesně indikovat daný problém a konstruktivně navrhnout řešení. V systému se toto projevuje například při úpravě scénáře dvěma lidmi najednou. Systém v takové situaci zobrazí druhému uživateli varování, že scénář byl mezitím upraven a navrhne mu uložit scénář pod novým názvem;

nápověda a dokumentace – přestože je lepší, když je systém použitelný bez dokumentace, je vhodné poskytnout uživatelskou nápovědu a dokumentaci. Ta by měla být snadno prohledatelná, zaměřená na daný úkon uživatele, popisovat přesné kroky, které mají být vykonány a být stručná. Uživatelská příručka se nachází v příloze C této práce.

2.1.1 Modely obrazovek

Pro zlepšení komunikace se zadavatelem a ostatními členy týmu jsem vytvořil modely obrazovek (wireframes), které zobrazují hlavní obrazovky aplikace a přechody mezi nimi. K jejich tvorbě byl použit nástroj Lucidchart [32], který obsahuje předdefinované tvary a UI prvky, např. tlačítka, posuvníky, vyhledávací pole a mnoho dalších. Modely obrazovek se typicky dělí na dva základní typy:

low-fidelity (nízkoúrovňové) – zobrazují pouze základní strukturu, data a ovládací prvky. Samotný vzhled modelu je maximálně zjednodušený, bez vizuálních efektů a jiných rušivých elementů. Ve výsledném produktu mohou být prvky přeskládány, měla by však být zachována struktura dat. Používají se především v rané fázi projektu pro komunikaci se zadavatelem a vyjasnění požadavků;



Obrázek 2.1: Výběr aplikace

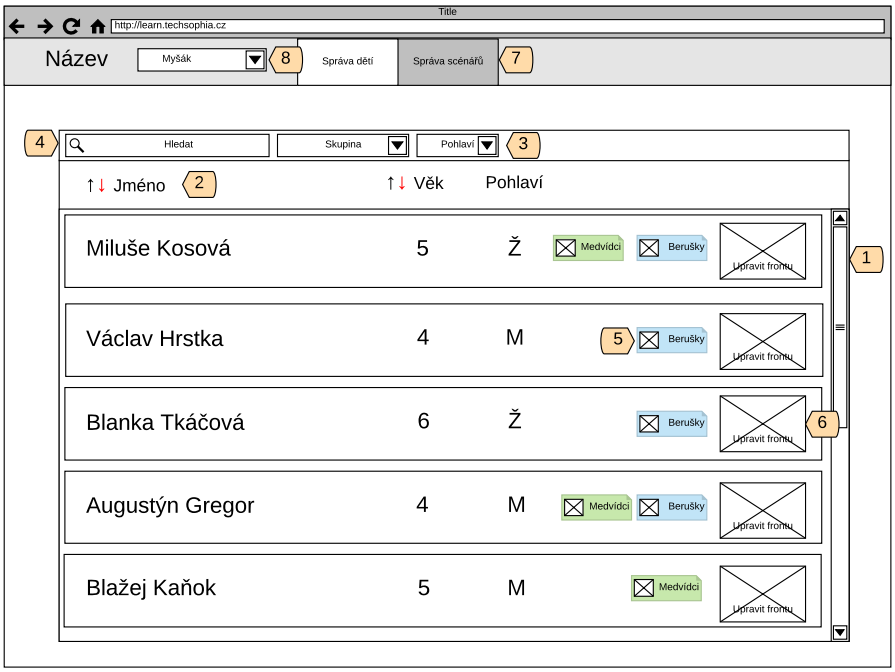
high-fidelity (vysokoúrovňové) – zobrazují vše co low-fidelity a navíc se zaměřují na výsledný design stránky, jednotlivé prvky, tlačítka, data a tabulky jsou tedy již umístěné tak, jak bude vypadat výsledný produkt. Obsahují také styly, stíny a barvy. Vzhled se snaží být co možná nejvíce identický s výsledným produktem. Tento druh wireframů se používá především pro designéry a frontend vývojáře, kteří dotvářejí výsledný vzhled.

Pro jednoduchost a urychlení komunikace se zadavatelem byl vytvořen pouze první druh wireframů (obrázky 2.1 – 2.6). Výsledný design je ovlivněný především výběrem frameworků pro tvorbu uživatelského rozhraní.

2.1.1.1 Výběr aplikace

Na obrázku 2.1 je vidět model obrazovky obsahující seznam podporovaných výsukových aplikací:

1 Seznam aplikací – abecedně seřazený seznam aplikací s jejich ikonami, po kliknutí se otevře webové rozhraní pro správu scénářů ve vybrané aplikaci, konkrétně správa dětí, viz sekce 2.1.1.2.



Obrázek 2.2: Správa dětí

2.1.1.2 Správa dětí

Obrázek 2.2 znázorňuje model obrazovky se seznamem dětí:

1 Seznam dětí – ve výchozím stavu abecedně seřazený seznam dětí;

2 Řazení – prvky sloužící k řazení, děti lze řadit podle jména nebo věku, sestupně nebo vzestupně;

3 Filtrování – filtrovat lze podle pohlaví a skupiny, do které je dítě přiřazeno;

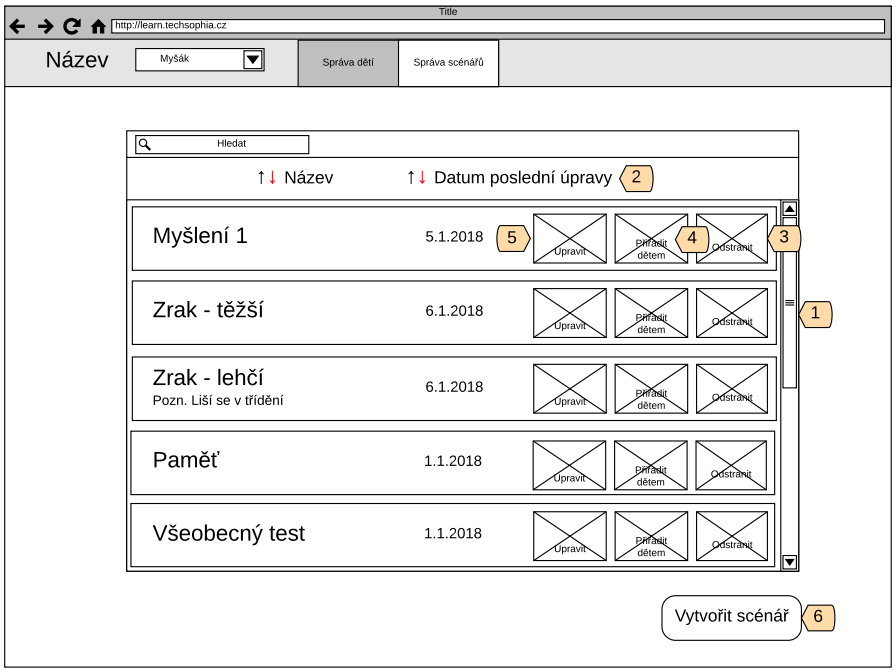
4 Hledání – vyhledávací pole, hledá se ve jméně dítěte;

5 Skupina dítěte – štítek s názvem a piktogramem označující skupinu, do které je dítě zařazeno. Dítě může patřit do více skupin;

6 Úprava fronty – tlačítko odkazující na rozhraní pro úpravu fronty scénářů dítěte, viz sekce 2.1.1.6;

7 Hlavní menu – přepínání mezi seznamem dětí a seznamem scénářů;

8 Výběr aplikace – změna zvolené aplikace, podobně jako v sekci 2.1.1.1.



Obrázek 2.3: Správa scénářů

2.1.1.3 Správa scénářů

Podobně jako seznam dětí je na obrázku 2.3 model obrazovky se seznamem scénářů v systému:

1 Seznam scénářů – ve výchozím stavu abecedně seřazený seznam scénářů;

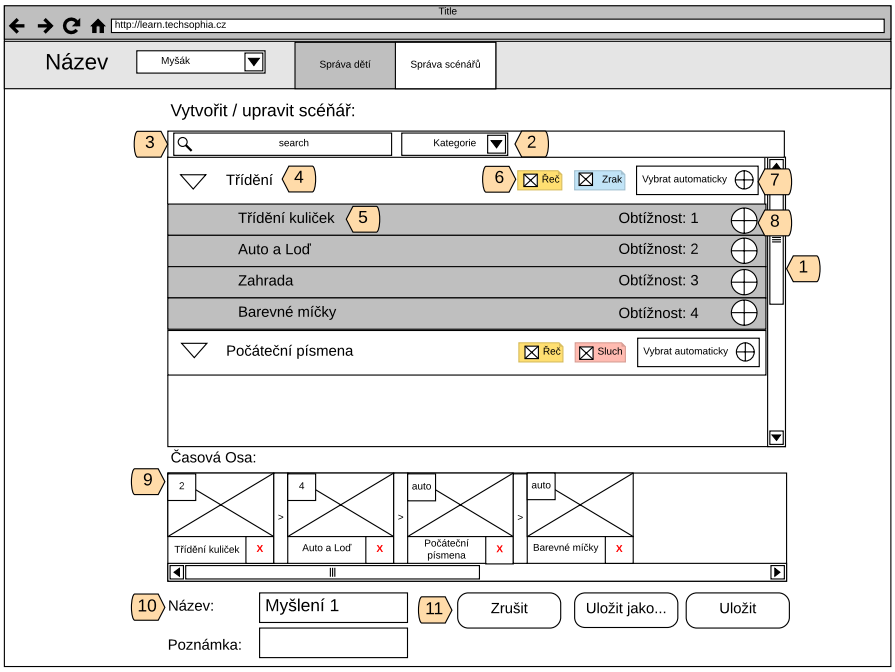
2 Řazení – prvky sloužící k řazení, scénáře lze řadit podle názvu nebo data poslední úpravy, sestupně nebo vzestupně;

3 Odstranit – tlačítko pro odstranění vybraného scénáře. Po jeho stisknutí se zobrazí potvrzovací dialog;

4 Přiřadit dětem – po stisknutí přepne uživatele do rozhraní pro výběr dětí, kterým scénář přiřazuje, viz sekce 2.1.1.5;

5 Upravit – tlačítko odkazující na rozhraní pro úpravu vybraného scénáře, viz sekce 2.1.1.4;

6 Vytvořit scénář – zobrazí rozhraní pro tvorbu scénáře (stejné jako rozhraní pro úpravu scénáře – jen s prázdnou časovou osou, viz sekce 2.1.1.4).



Obrázek 2.4: Úprava scénáře

2.1.1.4 Úprava scénáře

Model obrazovky pro úpravu a tvorbu scénáře popisuje obrázek 2.4:

1 Seznam úkolů – ve výchozím stavu abecedně seřazený seznam úkolů a jejich konfigurací;

2 Filtrování – úkoly lze filtrovat dle kategorie;

3 Vyhledávání – v úkolech a jejich konfiguracích se dá vyhledávat, pokud se vyhledávací řetězec shoduje s názvem úkolu, zobrazí se ve výsledcích úkol a všechny jeho konfigurace. Pokud se řetězec shoduje s názvem konfigurace, zobrazí se ve výsledcích odpovídající úkol a v druhé úrovni seznamu bude pouze ona shodující se konfigurace. Pokud by se řetězec shodoval s názvem úkolu i konfigurace, aplikuje se sjednocení množin výsledků. Např. řetězec „třídění“ odpovídá úkolu *Třídění* i konfiguraci *Třídění kuliček*, aplikuje se sjednocení a protože třídění kuliček je obsaženo v třídění, výsledek je stejný, jako by se řetězec shodoval pouze s úkolem třídění;

4 Úkol – obecný úkol, který může mít více různých konfigurací s různými obtížnostmi a herními předměty;

5 Konfigurace úkolu – konkrétní konfigurace s definovanou obtížností a herními předměty;

6 Kategorie – úkol může spadat do několika kategorií, které jsou vyznačeny štítkem a piktogramem;

7 Přiřazení úkolu – po stisknutí tohoto tlačítka se do časové osy (9) vloží vybraný úkol s obecnou konfigurací, tedy v obtížnosti se zobrazí *auto* a systém vybere vhodnou konfiguraci až při spuštění daného scénáře dítětem;

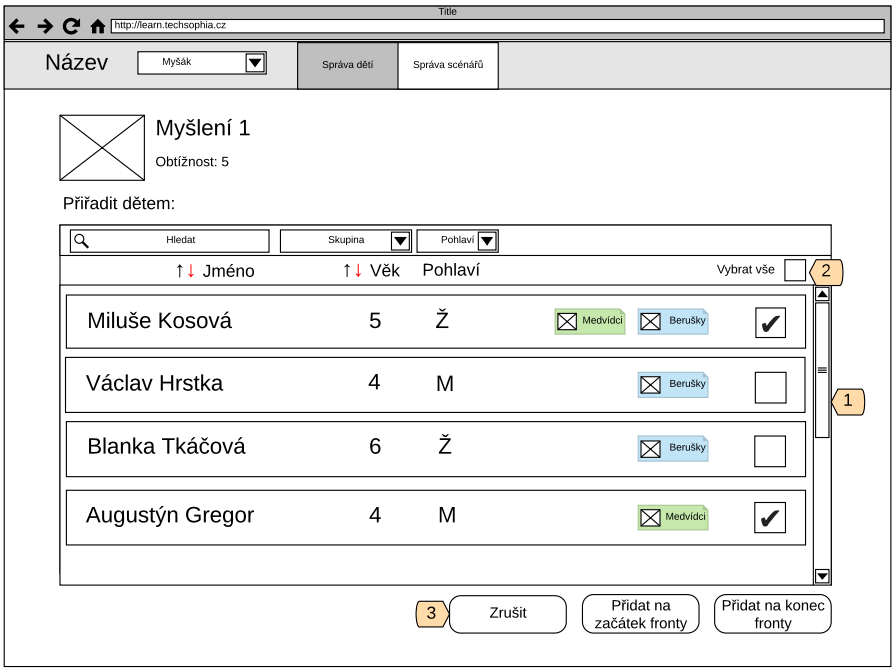
8 Přiřazení konfigurace – toto tlačítko naopak na konec časové osy vloží konkrétní vybranou konfiguraci s danou obtížností;

9 Časová osa – posloupnost úkolů a konfigurací řazených zleva doprava tak, jak se budou dítěti spouštět. Lze měnit jejich pořadí. V levém horním rohu je zobrazena obtížnost, která odlišuje úkoly od konfigurací. Úkol má obtížnost *auto*, zatímco konfigurace konkrétní číslo, např. „3“, křížkem lze úkol nebo konfiguraci z časové osy odstranit;

10 Název a popis – při vytváření nového scénáře musí uživatel zadat povinně název. Při úpravě existujícího scénáře bude toto pole předvyplněno existujícím názvem a uživatel ho může ponechat nebo změnit. Toto pole neslouží pro uložení scénáře pod novým názvem, to lze pomocí tlačítka *Uložit jako* (11);

11 Uložení – tato sekce obsahuje tři tlačítka: *Zrušit* – zahodí provedené změny, *Uložit* – uloží změny a nahradí tím původní upravovaný scénář (pokud se nejedná o vytváření nového scénáře, v takovém případě systém vytvoří nový scénář) a *Uložit jako* – uloží scénář pod novým názvem.

Pokud by se stalo, že dva uživatelé budou upravovat stejný scénář současně, systém vyřeší situaci následovně: Před zahájením úprav scénáře si systém zapamatuje datum poslední úpravy scénáře a při kliknutí na *Uložit* toto datum opět zkontroluje. Pokud se nezměnilo, nikdo scénář mezitím neupravil, tudíž je bezpečné scénář uložit. Pokud se však změnilo, uživateli se zobrazí upozornění, že scénář byl mezitím upraven někým jiným a dialogové okno pro uložení scénáře pod novým názvem. Po dokončení akce je uživatel vrácen zpět na obrazovku, odkud úpravu scénáře vyvolal.



Obrázek 2.5: Přiřazení scénáře

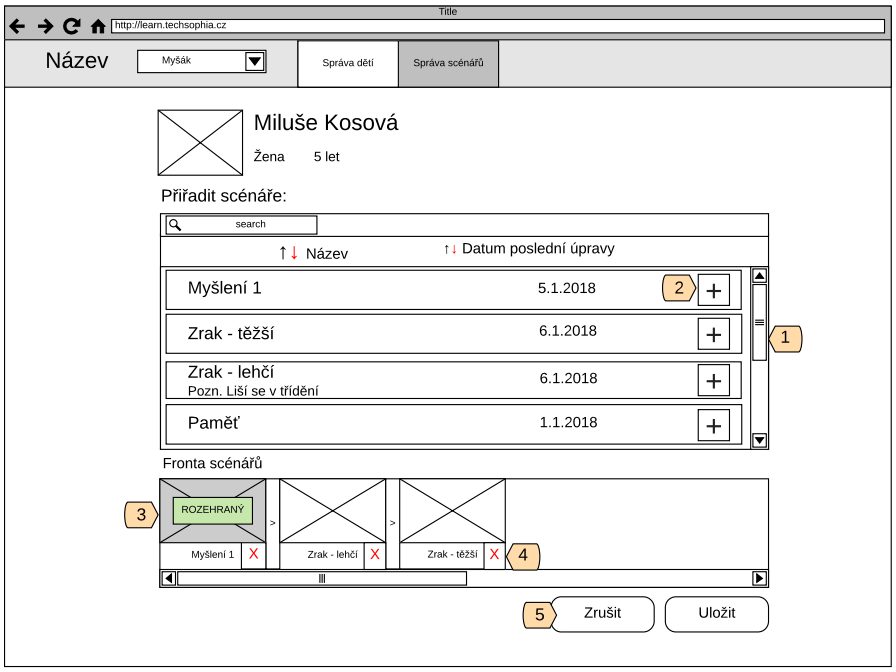
2.1.1.5 Přiřazení scénáře dětem

Model obrazovky sloužící k přiřazování scénářů dětem popisuje obrázek 2.5:

1 Seznam dětí – ve výchozím stavu abecedně seřazený seznam dětí se stejnými možnostmi řazení a filtrování jako v seznamu dětí v sekci 2.1.1.2;

2 Vybrat vše – zvolí všechny děti, které odpovídají filtru (tedy i ty, které nejsou aktuálně vidět a bylo by potřeba pohnout posuvníkem, aby se zobrazily);

3 Potvrzení – tlačítko *Zrušit* zahodí provedené změny. Tlačítko *Přidat na začátek fronty* přiřadí scénář na první místo ve frontě dítěte (pokud má dítě ve frontě nějaký scénář rozehraný, vloží se ihned za něj). Bude se tedy hrát okamžitě nebo ihned, co dítě dohraje rozehraný scénář. Tlačítko *Přidat na konec fronty* dle očekávání zařadí scénář na konec fronty scénářů dítěte. Dítě ho tedy bude hrát jako poslední. Pokud je zvoleno více než jedno dítě, provedou se tyto akce pro každé dítě zvlášť, tedy scénář se přiřadí na začátek nebo konec fronty všem zvoleným dětem.



Obrázek 2.6: Úprava fronty

2.1.1.6 Úprava fronty dítěte

Poslední model obrazovky na obrázku 2.6 ukazuje rozhraní pro úpravu fronty scénářů dítěte:

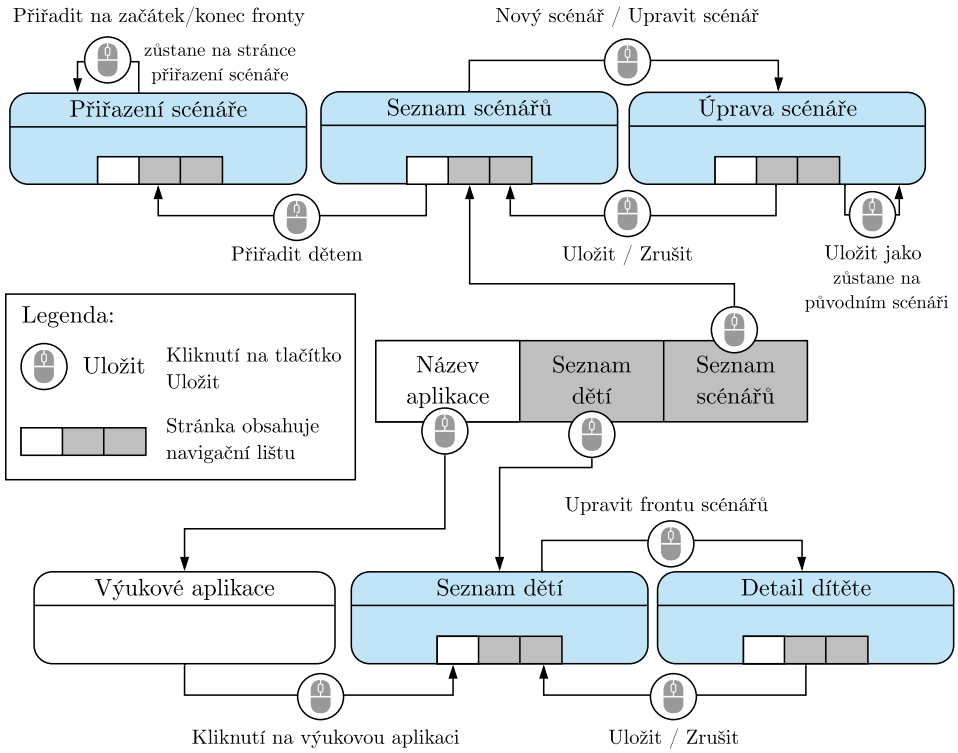
1 Seznam scénářů – ve výchozím stavu abecedně seřazený seznam scénářů se stejnými možnostmi řazení a filtrování jako v sekci 2.1.1.3;

2 Přidání scénáře – tlačítko + přidá scénář na konec fronty, objeví se tedy dole ve frontě jako poslední;

3 Fronta scénářů – zobrazuje časovou osu, posloupnost scénářů tak, jak se budou dítěti spouštět (zleva doprava). Scénářům lze měnit pořadí na ose. Pokud má dítě rozehraný scénář, zobrazí se jako první a bude graficky odlišen od ostatních scénářů. S takovým scénářem nelze hýbat a nelze před něj předřadit jiný scénář;

4 Odstranění scénáře – při stisknutí tohoto tlačítka na běžném scénáři je tento scénář odebrán z fronty. Pokud se jedná o rozehraný scénář, danému dítěti se scénář zruší, následně je také odebrán z fronty;

5 Potvrzení – zrušení nebo uložení provedených změn, uživatel je následně vrácen zpět na obrazovku, ze které úpravu fronty vyvolal.



Obrázek 2.7: Diagram přechodů mezi obrazovkami

2.1.2 Přechody mezi obrazovkami

Diagram na obrázku 2.7 znázorňuje přechody mezi jednotlivými obrazovkami. Hlavní navigační lišta, která se nachází na všech stránkách kromě výběru výukových aplikací, je pro zjednodušení zobrazena uprostřed. Také jsou vynechány přechody *Zpět*, které může uživatel vyvolat v každém okamžiku.

2.1.3 Responzivita

Z analýzy vyplývá, že cílové rozlišení obrazovky je primárně 1920×1080 neboli Full HD, nicméně je nutné podporovat i nižší rozlišení obrazovky, především mobilní telefony a tablety s OS Android. Kromě rozlišení se také může lišit poměr stran nebo orientace obrazovky. Při návrhu byl proto kladen důraz na tzv. responzivní zobrazení – přeskládání a přizpůsobení ovládacích prvků a komponent na obrazovce, aby byl uživatelský komfort co možná nejvyšší na každém rozlišení displeje.

Pro dosažení responzivity byl použit framework Angular Flex-Layout [33], který umožňuje přemisťovat, zobrazovat a schovávat prvky nebo styly v zá-

vislosti na rozlišení obrazovky. Tomuto frameworku se práce detailněji věnuje později v sekci 2.3.3.

Přesto, že je při návrhu přihlíženo k responzivnímu designu, vysokoúrovňové modely obrazovek zachycující rozložení prvků pro mobilní, tabletovou a desktopovou verzi webu jsou z důvodu časového omezení mimo rozsah projektu.

2.2 Návrh API

Podle analýzy je nejvhodnější REST jako přístup k API pro účely této aplikace. V následujících částech jsou podrobněji rozebrána učiněná návrhová rozhodnutí.

Vzhledem k tomu, že aplikace obsahuje dvě API, jedno mezi webovým rozhraním a serverem a druhé pro komunikaci mezi serverem a mobilní výukovou aplikací, rozhodli jsme se s backendovým vývojářem Jakubem Topičem na jejich návrhu spolupracovat. Druhé zmiňované API je totiž také z důvodu jednoduchosti na používání a implementaci formou REST. Umožnilo nám to rychle a efektivně odhalit možné nedostatky a hlavně se domluvit na některých konvencích a společných rysech, které usnadní vývoj backendu, testování a kromě produktivity zvýší i čitelnost obou rozhraní.

To však neznamená, že tato API jsou stejná. Každé z nich bylo pečlivě navrženo pro konkrétní způsoby užití, shodují se pouze některé endpointy, datové struktury, které jsou stejné v obou konzumentech a konvence, např. způsob, jakým je vyřešeno stránkování nebo stavové kódy. Samotná data a API webového rozhraní jsem vyvíjel a přizpůsobil hlavně pomocí způsobů užití (sekce 1.2.4) a návrhu obrazovek (sekce 2.1.1). Vzhledem k zaměření této práce se budu nadále zabývat pouze frontendovým API.

2.2.1 Styl architektury API

Z analýzy (sekce 1.3.3) vyplývá, že pro vytvoření správného REST API nestačí použít formát JSON a protokol HTTP, ale je potřeba držet se určitých doporučení.

Dodržet všechny tyto principy se však v praxi ukázalo jako obtížné a zbytečně navyšující komplexitu backendu i frontendu. Cílem přitom není navrhnout obecně nejlepší API, nýbrž takové, jaké nejvíce vyhovuje potřebám systému *Myšák*. Zde se tedy návrh mírně odchyluje od analýzy směrem k praktické použitelnosti namísto dodržování principů RESTu za každou cenu.

Žádoucí je především jednoduchost, jednotnost, čitelnost pro člověka, škálovatelnost a nízká obtížnost implementace. Například implementace principu HATEOAS je netriviální a většinou vyžaduje použití externích knihoven. Ukázalo se, že projektu nepřináší mnoho výhod a při jeho plném využití by byla výsledná konzumace API klientem naopak komplikovanější. HATEOAS by bylo výhodné použít až v momentě, kdy API konzumuje více různých klientů.

2.2.2 Volba formátu

Jednoznačnou volbou při implementaci REST rozhraní je datový formát JSON. V sekci 1.3.4 se analýza zabývá jeho standardizovanými podobami. Vzhledem k množství problémů a nejistoty, kterou však volba takového formátu přináší bylo od těchto technologií upuštěno. Formáty jsou totiž méně čitelné pro člověka, tím pádem je pro programátora obtížnější je pochopit a jejich podpora v knihovnách je nedostatečná. Místo toho byl upřednostněn formát uzpůsobený potřebám projektu u kterého byla snaha o maximální jednoduchost a čitelnost.

2.2.3 Volba technologií

Pro návrh a dokumentaci rozhraní jsem se, stejně jako backendový vývojář, rozhodl použít online nástroj Apiary [34]. Ten mimo jiné umožňuje jednoduchý popis rozhraní v jednom ze dvou nejpoužívanějších jazyků pro popis API: API Blueprintu nebo Swaggeru. S Jakubem jsme se shodli na použití Apiary ve spojení s API Blueprint.

2.2.3.1 Apiary

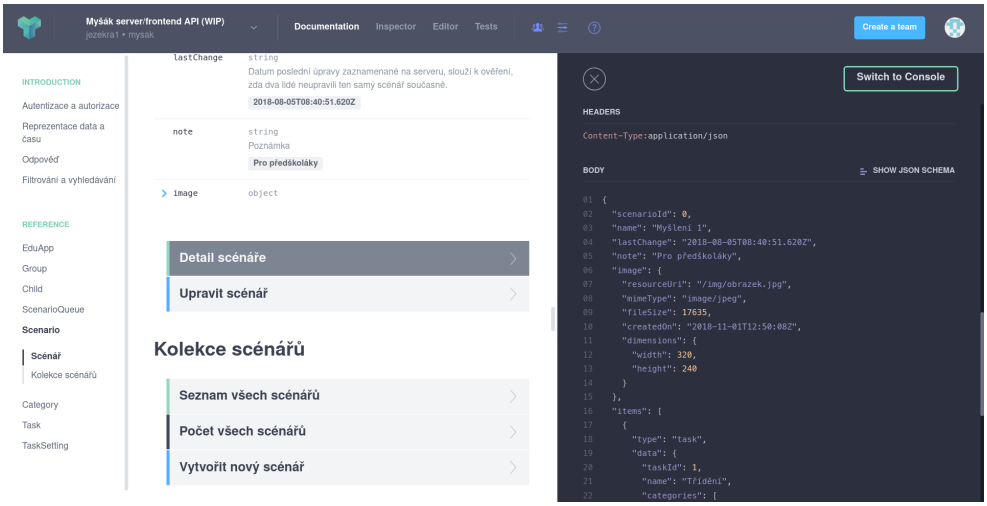
Jedná se o online službu umožňující snadný návrh, popis a testování REST API. Ve spojení s API Blueprint obsahuje mnoho užitečných funkcí:

- automatické vytvoření mockového serveru
- automatizované testování a přehledné zobrazení výsledků
- správa a verzování API na GitHubu

Mockový server slouží pro základní náhled na výsledné API (mock znamená „falešný“, tedy neúplný, neobsahující implementaci). Chová se podobně jako reálné API, ale data se neukládají a obsahuje pouze velmi omezenou množinu ukázkových dat z dokumentace. Je však výhodné ho použít pro tvorbu API testů nebo v rané fázi vývoje frontendu.

Automatické testy poskytované Apiary jsou na tom podobně. Ověří totiž pouze, zda serverová implementace odpovídá dokumentaci, obsahuje správné stavové kódy a vrací správný formát odpovědí. Jedná se tedy o syntaktické testy. Pro kontrolu logické správnosti rozhraní (např. že přidání scénáře ho skutečně přidá do seznamu scénářů) je již nutné vytvořit testy ručně.

Hlavním přínosem Apiary je však generování přehledné interaktivní dokumentace ve formě webové stránky s přehledem endpointů a jejich popisem. Na ty lze přímo v dokumentaci posílat požadavky. Ukázka výsledné dokumentace je na obrázku 2.8.



Obrázek 2.8: Ukázka vygenerované dokumentace API, navržené ve spolupráci s backendovým vývojářem, zdroj Apiary [34, snímek pořídil autor]

2.2.3.2 API Blueprint a MSON

API Blueprint [35] je moderní jednoduchý jazyk zpříjemňující návrh API. Obsahuje syntaxi pro definici a popis endpointů, HTTP metod, hlaviček, parametrů, požadavků a odpovědí. Samotná těla požadavků a odpovědí mohou být popsána ve formátu JSON nebo MSON, který je snadný na pochopení a ve většině případů vhodnější.

MSON je náhrada za tradiční JSON. Má jednodušší a přehlednější syntaxi, mezi výhody patří:

- syntaxe vycházející ze známeho jazyka Markdown
- definice datových struktur, které redukují množství opakujícího se kódu
- dědičnost mezi datovými strukturami

Výpis kódu 2.1 ukazuje definici zdroje a metod v API Blueprint a výpis kódu 2.2 ukazuje syntaxi definice dat pomocí MSON.

2.2.4 Navržené API

V této podsekci se věnuji celkové struktuře a architektuře navrženého API, proto tato část obsahuje pouze stručné shrnutí nejdůležitějších endpointů a datových struktur, kompletní dokumentaci lze nalézt na adrese `https://mysak.docs.apiary.io/`.

```
# Uživatelé [/users]
Kolekce uživatelů.

## Seznam všech uživatelů [GET]

+ Response 200 (application/json)

        + Attributes(User)
```

Výpis kódu 2.1: Definice zdroje a metod v API Blueprint

```
#User
+ id: 1 (number, required) - identifikátor uživatele
+ firstName: Jan (string) - jméno uživatele
+ lastName: Novák (string) - příjmení uživatele
+ address (object)
    + street: Borovanská 1234 (string) - ulice a číslo popisné
    + zipcode: 37008 (number) - poštovní směrovací číslo
    + city: České Budějovice (string) - město
```

Výpis kódu 2.2: Příklad syntaxe MSON

2.2.4.1 HTTP Metody

Z důvodu použití protokolu HTTP byla při návrhu snaha dodržovat sémantický význam HTTP metod použitých při manipulaci se zdroji:

`GET` – získá data ze zdroje, nemění stav dat

`POST` – přidání nového záznamu

`PUT` – úprava existujícího zdroje

`PATCH` – částečná úprava zdroje

`DELETE` – smazání záznamu ze zdroje

Mezi metodou `PUT`, `POST` a `PATCH` je jemný rozdíl, který se ale v navrhovaném API rozlišuje. Metoda `POST` slouží výhradně k vytvoření nového záznamu. Tedy například `POST /eduapps/1/scenarios` slouží pro vložení nového scénáře do systému.

Metody `PUT` a `PATCH` slouží k úpravě zdroje. Metoda `PUT` se vyznačuje tím, že pokud je poslán identický požadavek několikrát po sobě se stejnými daty, výsledný zdroj bude ve stejném stavu jako kdyby byl poslán pouze jednou. Metoda `PATCH` slouží naopak pro případy, kdy dochází k částečné aktualizaci

zdroje, byla použita pro vložení scénáře na začátek nebo na konec fronty. Opakované odesílání požadavku `PATCH` by pak vložilo několik nových scénářů do fronty dítěte.

2.2.4.2 Struktura endpointů

Aplikace obsahuje několik základních entit, některé z nich již byly zmíněné v části vysvětlení pojmů 1.2.1. Vzhledem k datové orientaci principu REST má každá entita vlastní příslušný zdroj (endpoint).

Výukové aplikace

`GET /eduapps` – seznam výukových aplikací

`GET /eduapps/{id}` – detail výukové aplikace s identifikátorem `{id}`

Většina zdrojů se váže ke konkrétní výukové aplikaci (eduapp). Z toho důvodu mají tyto endpointy ID výukové aplikace součástí URI. Například seznam dětí lze získat požadavkem `GET /eduapps/{eid}/children`. Parametr `{eid}` pak představuje unikátní identifikátor výukové aplikace. Pro přehlednost budou URI následujících zdrojů zkráceny, `/eduapps/{eid}/` bude z URI vynecháno, tedy například `/eduapps/{eid}/children` bude zkráceno na `/children`.

Skupiny

Děti jsou v mateřské škole rozděleny do skupin (například medvídci nebo berušky), z funkčních požadavků vyplývá, že systém musí umět snadno přiřadit scénář celé skupině dětí. Seznam všech skupin nebo detail jedné konkrétní lze získat následujícími požadavky:

`GET /groups` – seznam skupin

`GET /groups/{id}` – detail skupiny s identifikátorem `{id}`

Děti

Tento zdroj neumožňuje úpravy, děti jsou na backendovém serveru získávané z jiného portálu, v aplikaci *Myšák* jsou tato data jen pro čtení, proto lze odeslat pouze požadavky typu `GET`.

`GET /children` – seznam dětí

`GET /children/{id}` – detail dítěte s identifikátorem `{id}`

Fronta scénářů dítěte

Každé dítě má frontu scénářů, se kterou se dá manipulovat následujícími požadavky:

`GET /children/{id}/scenario-queue` – fronta scénářů dítěte s indentifikátorem `{id}`

`PUT /children/{id}/scenario-queue` – úprava fronty scénářů dítěte s identifikátorem `{id}`

`PATCH /children/{id}/scenario-queue/front` – vložení scénáře na začátek fronty dítěte s identifikátorem `{id}`

`PATCH /children/{id}/scenario-queue/back` – vložení scénáře na začátek fronty dítěte s identifikátorem `{id}`

Scénáře

Tvorba, mazání a úprava scénářů je jedním z funkčních požadavků aplikace. Proto se zde projevují různé HTTP metody. Pro přidání nebo úpravu nového scénáře je nutné odeslat serveru data (payload) obsahující nový nebo upravený scénář (název, poznámku a úkoly/konfigurace ve scénáři).

`GET /scenarios` – seznam scénářů,

`POST /scenarios` – přidání nového scénáře

`GET /scenarios/{id}` – detail scénáře s identifikátorem `{id}`

`PUT /scenarios/{id}` – úprava scénáře s identifikátorem `{id}`

`DELETE /scenarios/{id}` – smazání scénáře s identifikátorem `{id}`

Kategorie

Úkoly (a tím pádem i konfigurace) jsou rozděleny do kategorií, které reprezentují oblasti vývoje dítěte (myšlení, zrak, …). Informace o kategoriích lze získat následovně:

`GET /categories` – seznam kategorií

`GET /categories/{id}` – detail kategorie s identifikátorem `{id}`

Úkoly a konfigurace

Jak již bylo zmíněno v sekci 1.2.1 věnující se vysvětlení pojmů, úkol (task) má více konfigurací (task setting), které již mají konkrétní podobu a obtížnost. Každá konfigurace se váže k některému z úkolů. Úkoly a konfigurace jsou dané konkrétní výukovou aplikací a ve webovém rozhraní *Myšák* jsou proto pouze pro čtení.

`GET /tasks` – seznam úkolů,

`GET /tasks/{id}` – detail úkolu s identifikátorem `{id}`

`GET /tasks/{id}/tasksettings` – seznam konfigurací úkolu s identifikátorem `{id}`

`GET /tasks/{id}/tasksettings/{tid}` – detail konfigurace s identifikátorem `{tid}` úkolu s identifikátorem `{id}`

2.2.5 N+1 query problém

Tento nechvalně známý problém způsobuje zbytečně velké množství odeslaných požadavků na server kvůli špatnému návrhu API. Nechť pro ilustraci například entita *úkol* obsahuje *kategorie* pouze jako ID kategorií, do kterých spadá, atribut úkolu: `"categories": [ { "categoryId": 1 } ]` bude určovat, že daný úkol spadá do kategorie s ID 1. Pro získání detailních informací o dané kategorii je pak nutné odeslat nový požadavek, konkrétně `GET /eduapps/{eid}/categories/1`.

Pokud by tedy cílem bylo zobrazit seznam všech N úkolů a ke každému jeho kategorii, bylo by nutné se Nkrát dotázat serveru na kategorie každého z úkolů (případně vícekrát, úkoly mohou být ve více kategoriích). Tento přístup zbytečně zatěžuje klienta i server. Nabízí se dvě řešení:

- klient získá předem seznam všech kategorií a pak pouze místo ID kategorie dosadí skutečnou kategorii,
- kategorie rovnou celé zabudovat do těla JSON odpovědi.

Druhá ze zmiňovaných možností by však způsobila příliš dlouhé odpovědi. Pokud by v odpovědi bylo například 30 úkolů ve stejné kategorii, detaily kategorie by se zopakovali 30krát. Z toho důvodu bylo zvoleno první možné řešení, tedy přenechat zodpovědnost klientovi.

Z případů užití dále také vyplývá, že vždy při zobrazení detailu dítěte je potřeba zobrazit také jeho frontu scénářů, podobně pokaždé při zobrazení detailu scénáře se zobrazuje i seznam úkolů ve scénáři (toho si lze všimnout například v modelech obrazovek v sekci 2.1.1). Proto při dotazu na detail konkrétního dítěte je součástí odpovědi i jeho fronta scénářů. Při dotazu na celý seznam dětí, jejich fronty scénářů chybí (opět by byla odpověď příliš dlouhá). Tento přístup se opakuje u následujících zdrojů:

- dítě → fronta scénářů
- scénář → úkoly a konfigurace
- úkol → konfigurace úkolu
- skupina → děti patřící do skupiny
- kategorie → úkoly patřící do kategorie

2.2.6 Ukázka JSON požadavku a odpovědi

Jako formát těla odpovědi a požadavků byl zvolen JSON, výhodou je obrovská rozšířenost a jednoduché použití díky zabudované podpoře v mnoha knihovnách. Angular i Django podporují formát JSON ve výchozím stavu.

Výpis kódu 2.3 ukazuje reálnou komunikaci s mockovým serverem Apiary. Ukázka imituje vkládání scénáře (s ID 1) dítěti (s ID 1). Některé hlavičky (a obrázek scénáře) byly pro přehlednost vynechány.

Požadavek (vložení scénáře s ID 1):

```
PATCH /eduapps/1/children/1/scenario-queue/front HTTP/1.1
Host: private-1df806-mysak.apiary-mock.com

{
  "scenarioId": 1
}
```

Odpověď (vložený scénář):

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Wed, 20 Feb 2019 18:36:19 GMT

{
    "scenarioId": 1,
    "name": "Myšlení 1",
    "lastChange": "2018-08-05T08:40:51.620Z",
    "note": "Pro předškoláky",
    "image": "..."
}
```

Výpis kódu 2.3: Ukázka vložení scénáře na začátek fronty dítěte pomocí API

2.2.7 Cache

Ukládání dat do mezipaměti (cache) je výhodné především pro zlepšení doby odezvy výsledné aplikace nebo v situacích, kdy má uživatel špatné internetové připojení. Přestože analýza zmiňuje možnost implementace cache pomocí protokolu HTTP jako výhodu REST API oproti ostatním, Angular umožňuje snadnou realizaci cache pomocí tzv. interceptorů (viz později v sekci 3.3.3). Proto byla upřednostněna implementace cache na úrovni JavaScriptu namísto HTTP hlaviček.

V různých situacích je výhodné použít různé principy pro ukládání dat do mezipaměti:

žádná cache – každý požadavek o data se posílá přímo na server;

použít cache – použijí se data uložená v cache. Pokud mezipaměť data neobsahuje, pošle se dotaz na server a data se do ní uloží;

použít cache a poté obnovit data – stejně jako v předchozím případě se použijí data z cache, ale zároveň odešle dotaz na server. Data jsou aktualizována hned jak přijde odpověď.

Aplikace *Myšák* využívá první a poslední zmiňovaný přístup. Požadavky typu `GET` na získání seznamu výukových aplikací, dětí, scénářů, úkolů, konfigurací nebo skupin používají cache s obnovou dat. Předpokládá, že obsah v těchto zdrojích se nemění příliš často. Naopak fronta dítěte nebo obsah scénáře se mohou měnit často a je proto potřeba mít vždy aktuální data. Stejně tak požadavky typu `POST`, `PUT`, `PATCH` nebo `DELETE`, určené pro manipulaci s daty, nepoužívají cache, protože odpověď serveru indikuje úspěch či neúspěch operace.

2.2.8 Mockování API

Vývoj backendu a frontendu může probíhat jako dvě nezávislé činnosti, pokud je zajištěn zdroj testovacích dat umožňující vývoj frontendu. Nabízí se použít již dříve zmiňovaný Apiary mockový server, který je vygenerovaný na základě dokumentace. Jeho nedostatky, zejména malé množství dat a nemožnost ukládat změny, však neumožňují dostatečně otestovat veškeré vlastnosti systému. Tento server je proto vhodný pouze pro rané fáze vývoje a bylo nutné najít jiné řešení.

Tento problém je na tolik častý, že tvůrci Angularu zavedli podporu pro mockování dat přímo do frameworku. Je tak možné přidat vlastní implementaci serverového API v paměti, která přeruší veškeré skutečné požadavky na server a nahradí je vlastními odpověďmi. Toto řešení je velmi výhodné, protože umožňuje vyvinout celý frontend a po následném vypnutí tohoto mockového

API (což může být i tak jednoduché jako změna jedné proměnné) bude fungovat i se skutečným serverem (samozřejmě pouze pokud jeho implementace API dodrží dokumentaci).

Pro vývoj byl tedy vytvořen mockový API server v paměti obsahující generovaná testovací data i s možností je modifikovat. Navíc lze velmi snadno simulovat různé chyby serveru nebo dlouhou odezvu a pomocí toho ověřovat implementaci chybových dialogů a reakce frontendu na nepříznivé situace. Výsledek je natolik přesvědčivý, že uživatel téměř nepozná, že nepracuje s reálným serverem, což se později ukázalo jako výhodné zejména pro uživatelské testování.

2.3 Zvolené webové technologie

Technologie a frameworky byly pečlivě vybrány tak, aby odpovídaly požadavkům zadavatele, vycházejí také z analytické části práce viz sekce 1.5, kde je již popsán framework Angular a jazyk TypeScript.

2.3.1 RxJS

RxJS je zkratkou pro Reactive Extensions for JavaScript (neboli reaktivní rozšíření pro JavaScript), hlavním cílem této především funkcionální knihovny je usnadnit zpracování dat, které jsou ze své podstaty asynchronní, proudové (stream). Klíčový je typ `Observable`, který představuje jakýsi zdroj asynchronních dat. Umožňuje registrovat callback funkci, která se zavolá vždy, když `Observable` „vyzáří“ další hodnotu.

Některé vestavěné třídy Angularu používají `Observable` jako návratovou hodnotu, například `HttpClient` Angularu. Výhodou knihovny je především možnost odebírat data z jednoho zdroje na více místech zároveň, jejich modifikace po cestě a inteligentní zpracování chyb.

2.3.2 Angular Material

Tato knihovna [31] je implementací vizuálního jazyku Material Design. Obsahuje předpřipravené komponenty (tlačítka, vyhledávací pole, tabulky, seznamy, ikony apod.), které již obsahují CSS styly respektující zmiňovaný jazyk. Material Design je již také plně integrován do operačního systému Android a některých webových služeb (například Gmail, Fotky Google, vyhledávání Google atd.), z důvodu maximalizace konzistence uživatelského rozhraní vzhledem k použité platformě, jednoduchosti a povědomí uživatelů je výhodné řídit se tímto standardizovaným designem. Hlavní výhodou jsou hotové komponenty, jejich jednotnost, čistý minimalistický vzhled a hotové CSS styly.

2.3.3 Angular Flex-Layout

Flex-Layout [33] byl do projektu přidán až v průběhu vývoje. Umožňuje snadnou definici rozvržení rozhraní přímo v HTML šabloně komponenty využívající moderní CSS atributy Flex (pružné rozhraní) a Grid (mřížka). Kromě toho umožňuje snadnou definici responzivního chování, přesun nebo schovávání prvků při zmenšení okna prohlížeče, například na desktopu mohou být položky navigační lišty vedle sebe a na telefonu pod sebou.

Hlavní nevýhodou je nefunkčnost ve starších verzích prohlížečů, které CSS Grid a Flex nepodporují. Dle webu caniuse.com jsou však tyto atributy podporované v prohlížeči Google Chrome od vere 57 a ve Firefoxu od verze 52 (CSS Flex dokonce ještě dříve). Z analýzy vyplývá, že tyto verze vyhovují požadavkům zadavatele a jsou v souladu s dohodnutými podporovanými verzemi dle analýzy prohlížečů v sekci 1.6.

2.4 Návrh architektury webové aplikace

V části analýzy 1.5 věnující se požadovaným technologiím již byla popsána základní architektura Angularu. Tato sekce se podrobně věnuje jednotlivým stavebním kamenům a jejich návrhu pro webového klienta aplikace *Myšák*.

2.4.1 Služby

K získání a zpracování dat z API jsou v Angularu určené služby (*services*). Služba je třída, která je anotovaná dekorátorem `@Injectable`. Lze ji pak díky systému dependency injection použít v jiných službách a komponentách. Vkládání závislostí zajišťuje Angular automaticky, stačí je deklarovat jako parametr v konstruktoru třídy. Detailní popis služeb a vkládání závislostí je nad rámec této práce.

2.4.1.1 Řešení chyb

Pro snadné odchytávání chyb (například chyba sítě nebo serveru), obsahuje Angular tzv. interceptory (více později v sekci 3.3.3). Jejich problémem však je, že pracují na globální úrovni, pro každý požadavek. Je tedy velmi snadné implementovat obecnou chybovou zprávu jako „něco se pokazilo“.

Jsou však situace, kdy je vhodné uživatelům sdělit, co se pokazilo, či případně reagovat na chyby jiným způsobem (zopakovat pokus, vyzvat uživatele k opravě údajů apod.). Jedním příkladem je stav, kdy je scénář současně upraven několika učiteli. V takovém případě má být uživatel požádán o zadání nového názvu, pod kterým systém uloží scénář. Řešení pomocí interceptorů tak není dostatečně flexibilní, a proto byla tato funkcionalita přenechána na služby. Realizací této myšlenky je rozdělení služeb na *datové* a speciální službu pouze pro odchytávání chyb `ApiService`.

2.4.1.2 Datové služby

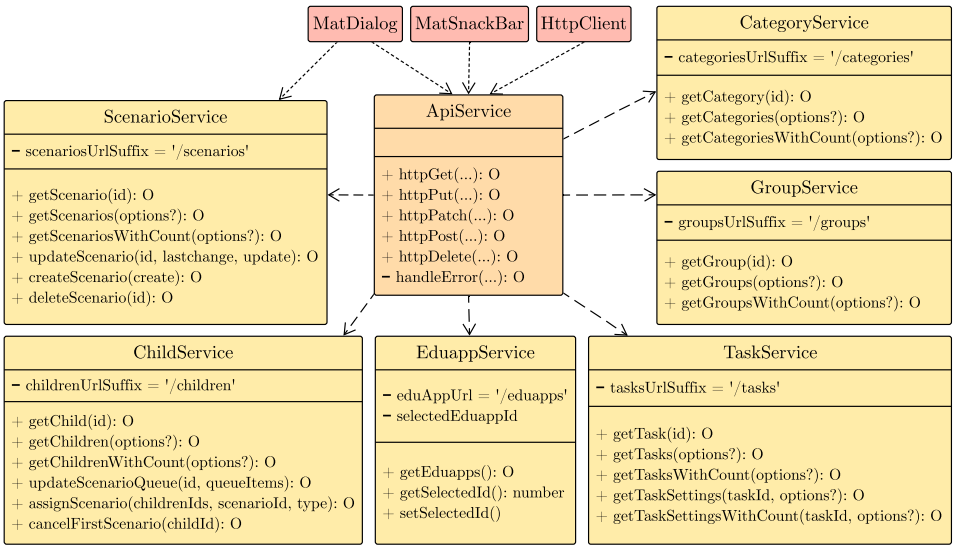
Obsahují metody určené pro získání, vkládání, úpravu a mazání prvků a používají metod `ApiService` pro interakci s API. Instance `ApiService` je dodána automaticky systémem vkládání závislostí. Datové služby jsou v diagramu na obrázku 2.9 znázorněny žlutě.

2.4.1.3 ApiService

Tato služba je mezivrstvou mezi datovými službami a HTTP klientem Angularu. Datové služby ji používají namísto přímého volání HTTP klienta. Zjednodušeně řečeno tedy `ApiService` pouze přepošle požadavek HTTP klientovi a po cestě odchytí možné chyby. Datové služby si přitom mohou zvolit text a typ chybové zprávy (dialog s upozorněním či pouhá vyskakovací notifikace) nebo úplně vynechat odchytnutí chyby a vyřešit jí později.

2.4.1.4 Diagram služeb

Diagram tříd na obrázku 2.9 znázorňuje služby a závislosti mezi nimi vkládané systémem dependency injection (znázorněné přerušovanými šipkami). Červeně jsou znázorněné externí závislosti: `MatDialog` a `MatSnackBar` pocházejí z frameworku Angular Material, `HttpClient` je služba Angularu určená pro komunikaci s API. Návratový typ `O` označuje `Observable`.



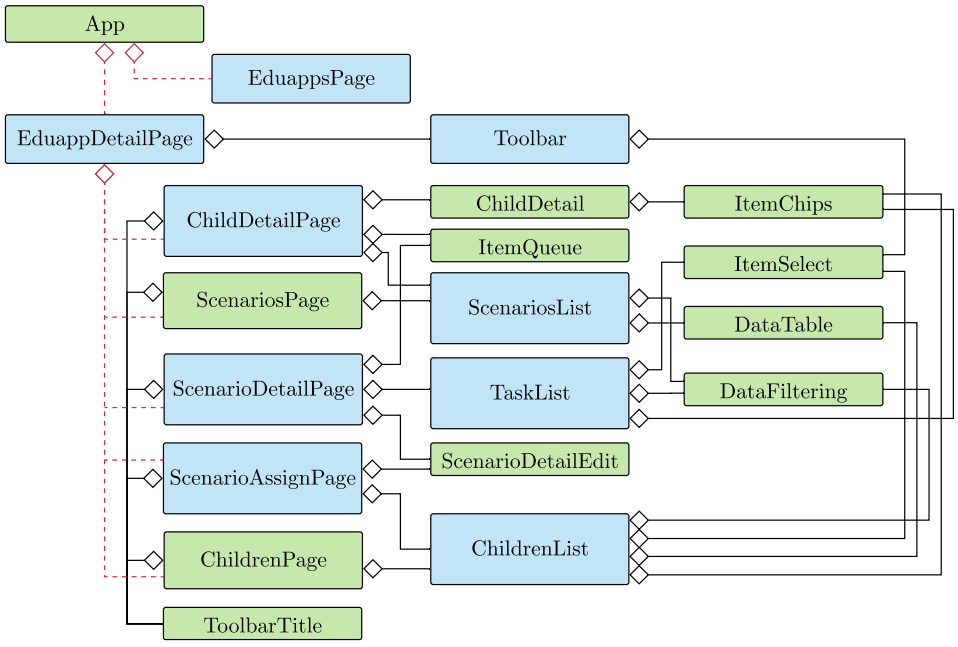
Obrázek 2.9: UML diagram tříd reprezentující služby (zjednodušený)

2.4.2 Komponenty

Jedním z hlavních stavebních kamenů aplikace jsou komponenty. Vycházejí přitom z konceptu již dávno známého webovým programátorům – HTML tagů. Ukázalo se, že je velmi výhodné tento koncept rozšířit a umožnit programátorům vytvořit „vlastní HTML tagy“, právě to je jedním z důvodů, proč jsou webové frameworky jako Angular nebo React tak populární. Vedle klasických tagů, jako `<div>` nebo `<h1>`, lze vytvořit komponenty s vlastním tagem, např. `<app-scenario-list>`.

Konkrétně aplikace *Myšák* obsahuje například komponenty: navigační lišta, datová tabulka, seznamy (dětí, scénářů, úkolů), fronta nebo i celé stránky. V Angularu má každá komponenta svou HTML šablonu, která může obsahovat klasické HTML tagy i vlastní komponenty. Druhé zmiňované Angular vytvoří automaticky a vloží je na příslušné místo v šabloně. Princip skládání se tak aplikuje postupně od kořene (hlavní komponenta `App`), do které jsou vložené další komponenty, až je postupným zanořováním a vkládáním vytvořená celá aplikace.

Z implementačního pohledu jsou komponenty třídy vyznačující se dekorátorem `@Component`, ve kterém jsou jako metadata předány odkazy na HTML šablonu a styly.



Obrázek 2.10: Diagram skládání komponent, routování je znázorněno červenou přerušovanou čarou

Komponenta může být velmi jednoduchá, například pouhé tlačítko s předdefinovanými styly, barvou a vloženým textem nebo velmi komplexní, zobrazující data z API a obsahující vnitřní logiku. Podle toho se dají dělit na *aplikační komponenty*, které získávají data ze služeb a *prezentační komponenty*, které pouze přejímají data od ostatních komponent. Na obrázku 2.10 jsou prezentační znázorněny zeleně a aplikační modře. Důvodem dekompozice na prezentační komponenty je především jejich srozumitelnost a znovupoužitelnost. Např. stejná komponenta `DataTable` (datová tabulka, znázorněná na obrázku 2.10 zeleně), je použita nejen v seznamu scénářů, ale také v seznamu dětí. Stejně je tomu u `DataFiltering`.

2.4.2.1 Skládání komponent

Jak již bylo zmíněno, hlavním principem uplatňujícím se při tvorbě aplikace je vnořování komponent do sebe. Diagram na obr. 2.10 ukazuje, jak bylo skládání navrženo. Například komponenta `ChildrenPage` (stránka s dětmi) obsahuje vnořené komponenty `ToolbarTitle` (titulek stránky) a `ChildrenList` (seznam dětí), druhá zmíněná přitom sama obsahuje další vnořené komponenty.

Červená přerušovaná čára značí routování, tedy vložení komponenty podle aktuální URL v prohlížeči. Například komponenta `App` obsahuje `EduappsPage`, pokud URL končí na `/eduapps` nebo `EduappDetailPage`, pokud již uživatel vybral výukovou aplikaci a URL obsahuje `/eduapps/:id-aplikace`.

2.4.2.2 Vkládání závislostí do komponent

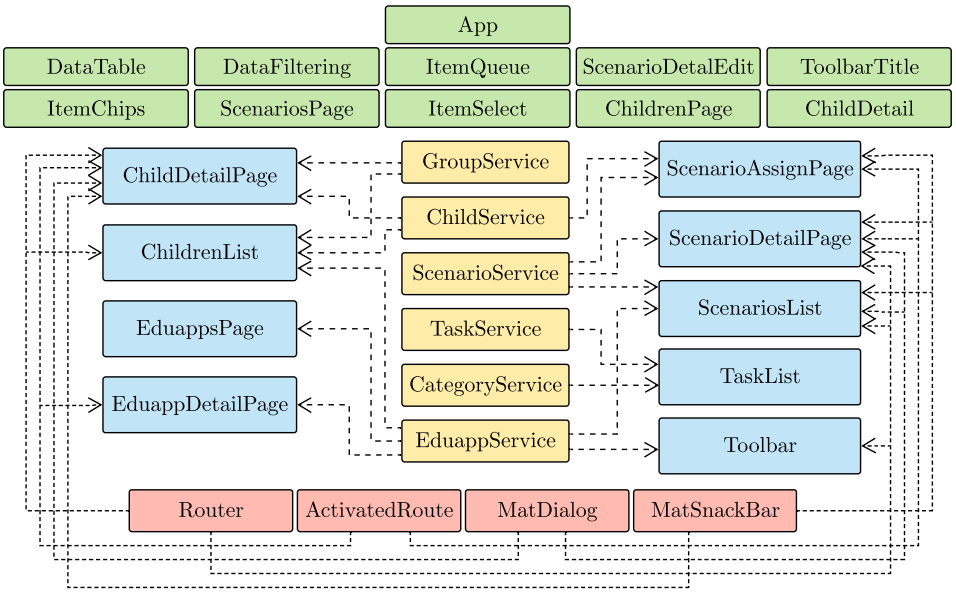
Aplikační komponenty získávají data z API pomocí služeb. Diagram na obrázku 2.11 znázorňuje služby vkládané systémem dependency injection. Zeleně jsou zobrazené komponenty prezentační, tedy nevyužívající žádné datové služby. Červená představuje externí závislosti (stejně jako na obrázku 2.9).

2.4.3 Navigace a Routing

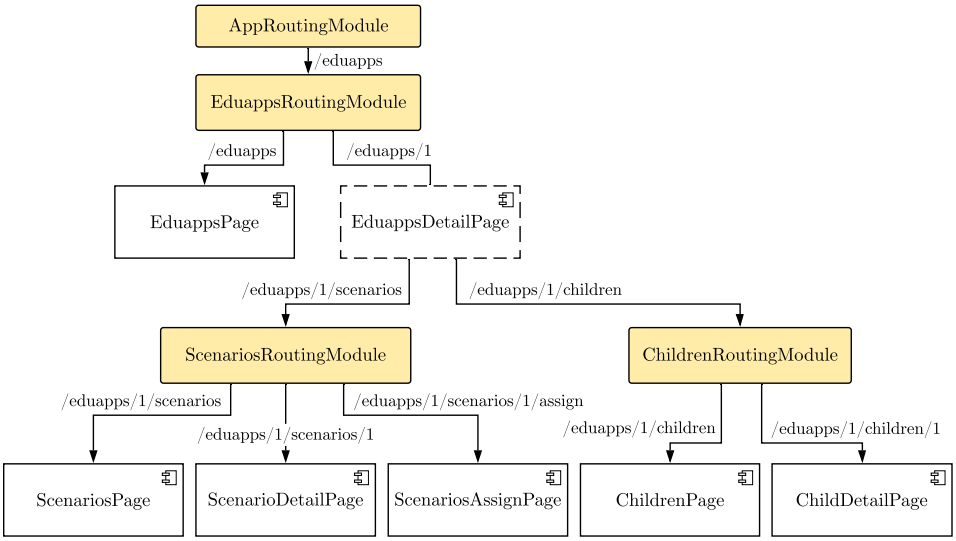
K navigaci mezi jednotlivými stránkami aplikace slouží Angular router. Těch je typicky několik a jsou hierarchicky uspořádáné, viz obrázek 2.12. Při shodě URL s některou z možných router načte a zobrazí odpovídající komponentu, zde např. `/eduapps/1/children` zobrazí komponentu `ChildrenPage`, zodpovědnou za zobrazení seznamu dětí.

Kromě navigace také umožňuje pomocí tzv. *guards* (strážců) zabránit načtení nebo naopak odchodu z konkrétní URL cesty. Toho lze využít například pro zamezení ztráty neuložených změn na stránkách editace scénáře a fronty dítěte. Uživatel je tak upozorněn, že má neuložené změny a je dotázán, zda chce ze stránky opravdu odejít.

Komponenta `EduappsDetailPage` (na obrázku 2.12 znázorněná přerušovaně) obsahuje hlavní navigační lištu. Je přítomná v celém podstromu pod `/eduapps/1`.



Obrázek 2.11: Diagram využívání služeb komponentami



Obrázek 2.12: Využití routerů (žlutě) pro načtení různých komponent(bíle)

Kapitola 3

Realizace

Tato kapitola je zaměřená na implementaci a technické řešení navrženého systému. Obsahuje zejména ukázky zajímavých částí kódu a souborovou strukturu aplikace.

3.1 Příprava

Před zahájením samotného vývoje bylo provedeno několik přípravných kroků od založení repozitáře až po vytvoření samotné aplikace a spuštění vzorové Angular aplikace.

3.1.1 Repozitář

Pro verzování kódu byl vybrán verzovací systém GIT. Repozitář byl hostovaný na serveru `gitlab.com` [36] kvůli jeho snadné konfiguraci a možnostem vestavěné continuous integration. Důležitým faktorem při volbě těchto technologií byla také dobrá znalost verzovacího systému GIT a jeho vlastností. Vývoj probíhal v samostatných větvích věnujících se převážně jedné funkcionalitě (feature branch), které byly začleňovány do hlavní větve master. Vždy když docházelo k začlenění práce, byl spuštěn proces continuous integration.

3.1.2 Continuous integration

Continuous integration (CI) nastavil v rámci celého projektu (tedy i frontendové části) backendový vývojář Jakub Topič. Použil k tomu nástroje Gitlab CI, které umožňují snadnou konfiguraci CI za použití vlastností GIT repozitáře. Proces byl nastaven tak, aby se aplikace při začlenění práce do hlavní větve vždy sestavila a následně byla nasazena na testovací webový server v cloudové platformě Heroku. Automatické nasazení aplikace také usnadnilo uživatelské testování a sdílení výsledků práce se zadavatelem a týmem, protože bylo možné na poslední verzi aplikace ve větvi master přistupovat z internetu.

3.1.3 Vytvoření počáteční aplikace

Samotná instalace a spuštění ukázkové aplikace byla snadná, pomocí několika příkazů dle oficiální dokumentace. Angular také inicializoval prostředí nástroje *npm* pro správu především JavaScriptových knihoven a balíčků. Jeho pomocí byly doinstalovány závislosti `@angular/material` a `@angular/flex-layout` zmiňované v sekci 2.3 věnující se zvoleným webovým technologiím. Knihovna RxJs byla již nainstalovaná jako součást Angular aplikace.

3.1.4 Vývojové prostředí

Angular v základní instalaci obsahuje vešeré nástroje potřebné k lokálnímu vývoji, testování a sestavení aplikace. K vývoji jsem použil WebStorm [37], vývojové prostředí od IntelliJ, které je dobře integrováno s Angularem, včetně automatické kontroly syntaxe a datových typů pomocí nástroje tslint [38].

3.2 Souborová struktura

Soubory a složky zdrojových kódů jsou logicky uspořádané především podle entit, ke kterým se vztahují. Sdílené globální služby, komponenty a další třídy jsou umístěny ve složce `shared`. Struktura aplikace je naznačena na obr. 3.1.

Ve složce s názvem entity (např. *children* reprezentující děti), jsou seskupené komponenty reprezentující stránky, model entity, znovupoužitelné komponenty a definice navigace mezi stránkami, viz obrázek 3.2.

```
src
└── app ..... zdrojové kódy aplikace
    ├── children ..... stránky a komponenty pro zobrazení dětí
    ├── categories ..... kategorie
    ├── eduapps ..... výukové aplikace
    ├── groups ..... skupiny
    ├── scenarios ..... scénáře
    ├── tasks ..... úkoly a konfigurace
    └── shared ..... sdílené globální třídy a moduly
        ├── guards ..... chrání cesty URL před načtením nebo opuštěním
        ├── interceptors ..... upravují HTTP požadavky
        ├── mocking ..... pomocná falešná data
        ├── testing ..... pomocná testovací data
        ├── utility ..... pomocné třídy a funkce
        ├── widgets ..... prezentační UI komponenty
        └── services ..... globální služby
└── e2e ..... end-to-end (konec-konec) testy
```

Obrázek 3.1: Souborová struktura aplikace

```
children ..... stránky a komponenty pro zobrazení dětí
    ├── children-page ..... stránka obsahující seznam dětí
    ├── child-detail-page ..... stránka obsahující detail dítěte
    ├── children-routing.module.ts .... lokální navigace modulu dětí
    ├── children-module.ts .... modul deklarující komponenty stránek
    ├── child.model.ts ..... model dítěte
    └── shared ..... sdílené znovupoužitelné moduly a komponenty
        ├── children-list ..... tabulka obsahující seznam dětí
        └── child-detail ..... modul zobrazující detail dítěte
```

Obrázek 3.2: Souborová struktura jedné entity

3.3 Implementace navržené architektury

Základní architektura Angular aplikace byla již popsána v sekci 1.5. Tato sekce se věnuje implementaci jednotlivých částí webové aplikace včetně ukázek zajímavých kusů kódu.

3.3.1 Služby

V sekci 2.4.1 již bylo popsáno, jakým způsobem byly služby navržené a rozdělené. Tato část se podrobněji věnuje jejich praktické implementaci.

3.3.1.1 Datové služby

Zkrácená ukázka datové služby `GroupService` (viz výpis kódu 3.1) obsahuje metody `getGroups` a `getGroup`, které slouží k získání skupin dětí z API. Lze si všimnout, že namísto HTTP klienta používá k získání dat instanci `apiService` služby `ApiService` a její metodu `httpGet`, jak bylo zmíněno v sekci 2.4.1 věnující se návrhu služeb. Tato služba je deklarovaná v konstruktoru (`private apiService: ApiService`) a Angular systémem vkládání závislostí zajistí její instanci.

3.3.1.2 ApiService

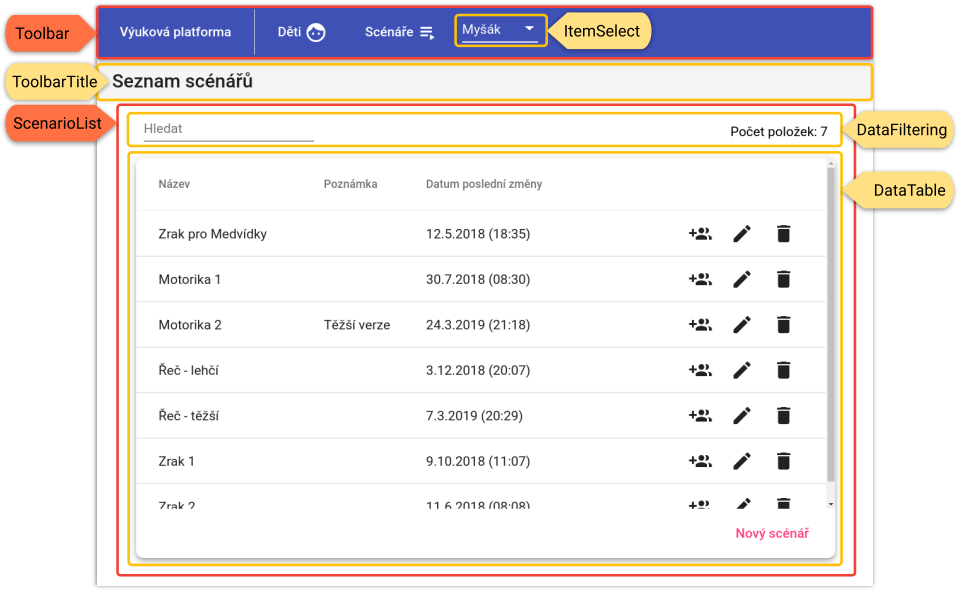
Ve zkrácené ukázkce služby `ApiService` (viz výpis kódu 3.2) si lze všimnout vloženého HTTP klienta (`private http: HttpClient`). Ten zprostředkovává odeslání požadavků, jejich zpracování a serializaci odpovědí serveru do objektů. Následně je odpověď HTTP klienta předána do metody `handleError`, která odchytí případné chyby dle konfigurace (způsob konfigurace chybových hlášek je nad rámec tohoto textu, lze jej najít ve zdrojových kódech aplikace na přiloženém paměťovém médiu).

```
@Injectable({providedIn: 'root'})
export class GroupService {
  private groupsUrlSuffix = '/groups';

  constructor(private apiService: ApiService) {}

  getGroup(id: number): Observable<Group> {
    return this.apiService.httpGet<Group>(
      this.groupsUrlSuffix + '/' + id);
  }

  getGroups(options?: GroupsOptions): Observable<Group[]> {
    return this.apiService.httpGet<Group[]>(
      this.groupsUrlSuffix,
      {params: toParams(options)});
  }
  ...
}
```

Výpis kódu 3.1: Ukázka datové služby

```
@Injectable({providedIn: 'root'})
export class ApiService {

  constructor(private http: HttpClient, ... ) {}

  private handleError<T>(request ... ) { ... }

  httpGet<T>(url: string, options ... ) {
      return this.handleError<T>(
               this.http.get<T>(url, options), ... );
  }

  httpPost<T>(...) { ... }
  httpPut<T>(...) { ... }
  httpPatch<T>(...) { ... }
  httpDelete<T>(...) { ... }
}
```

Výpis kódu 3.2: Zkrácená ukázka ApiService



Obrázek 3.3: Komponenty, červeně - aplikační, žlutě - prezentační

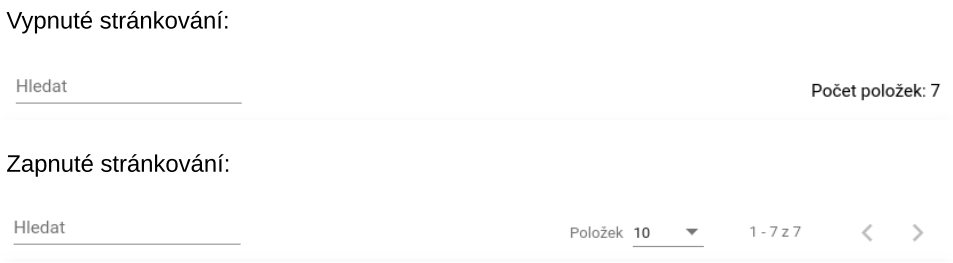
3.3.2 Komponenty

Na obrázku 3.3 je zobrazena konkrétní realizace komponent a jejich vkládání, dle návrhu komponent v sekci 2.4.2. Prezentační komponenty jsou znázorněny žlutě a aplikační červeně. Obrázek 3.4 znázorňuje souborovou strukturu obsahující komponentu, její test, HTML šablonu, a styly.

Pro popsání vlastností a principů implementace byla jako příklad zvolena prezentační komponenta `DataFilteringComponent`. Slouží k zobrazení vyhledávacího pole, stránkování (pokud je zapnuté), případně dalších specializovaných komponent pro filtraci datových tabulek. Na obrázku 3.3 je zobrazena nad datovou tabulkou žlutě. Obrázek 3.5 ještě ukazuje, jak se zapnuté/vypnuté stránkování odrazí na vzhledu komponenty.

```
example.....................................složka komponenty Example
├── example.component.ts....................třída a logika komponenty
├── example.component.spec.ts...............unit test komponenty
├── example.component.html..................HTML šablona komponenty
└── example.component.scss..................styly komponenty
```

Obrázek 3.4: Souborová struktura jedné komponenty



Obrázek 3.5: Komponenta `DataFilteringComponent` a stránkování

```
<div ...>
  <div ...>
    <mat-form-field fxFlex="initial">
      <input matInput
        (keyup)="onSearch($event.target.value)"
        placeholder="Hledat">
    </mat-form-field>
    <ng-content></ng-content>
  </div>
  <div fxFlex="auto"></div>
  <span fxShow.lt-sm="false"
        *ngIf="!paginate && count"
        fxFlexAlign="center"
        style="padding-right:8px">
    Počet položek: {{count}}
  </span>
  <mat-paginator *ngIf="paginate"
                 [pageSizeOptions]="[5, 10, 20, 50]"
                 [pageSize]="pageSize"
                 [length]="count"
                 (page)="onPageChanged($event)">
  </mat-paginator>
</div>
```

Výpis kódu 3.3: Šablona komponenty DataFiltering

3.3.2.1 HTML Šablona komponenty

Kód šablony (viz výpis kódu 3.3) této komponenty `<app-data-filtering>` ukazuje hned několik principů, které byly při vývoji uplatněny:

vnořování komponent – lze si všimnout hned několika vnořených komponent: `<mat-paginator>`, `<mat-form-field>` nebo speciálního kontejneru `<ng-content>`, který na svém místě v šabloně zobrazí komponenty přejaté od rodiče (mezi otevíracím a zavíracím tagem). Např. `<app-data-filtering><p>text</p></app-data-filtering>` by místo kontejneru `<ng-content>` dosadilo odstavec `<p>text</p>`;

data a proměnné – HTML šablona (výpis kódu 3.3) může používat instanční proměnné třídy obsahující logiku komponenty (výpis kódu 3.4). V šabloně je například vidět použití proměnné `count` pro zobrazení počtu položek `Počet položek: {{count}}`;

předávání dat a data binding – stejně jako v HTML lze jednotlivým komponentám předávat parametry. Syntaxe je také velmi podobná HTML, avšak je zde několik rozdílů. Angular totiž zajišťuje tzv. data binding, neboli automatickou propagaci a synchronizaci dat mezi komponentami.

V šabloně si lze všimnout, že komponentě `<mat-paginator>` jsou předávány parametry `[length]="count"`, `[pageSize]="pageSize"` aj., které ovlivňují stránkování. Argumenty `count` a `pageSize` jsou přitom instanční proměnné třídy `DataFilteringComponent`. Angular poté zajistí, že kdykoliv se změní jedna z těchto proměnných, novou hodnotu automaticky obdrží i `<mat-paginator>`. To je smyslem hranatých závorek při předávání proměnné. Kulaté závorky naopak umožňují přejímat data od vnořené komponenty. `(page)="onPageChanged($event)` např. zajišťuje, že pokud uživatel klikne na další stránku, zavolá se metoda `onPageChanged` s parametrem popisujícím změnu;

využití komponent Angular material – zmiňované vnořené komponenty s předponou *mat-* jako `<mat-paginator>`, jsou součástí knihovny předpřipravených komponent s designem material zmiňované v sekci 2.3.2;

podmíněné zobrazení komponent – direktiva `*ngIf` použitá v šabloně hned dvakrát, způsobí zobrazení nebo zmizení komponenty na základě podmínky předané parametrem. Zde se toho využívá u stránkování, které se zobrazuje pouze pokud je zapnuté, tedy pokud je instanční proměná `paginate` třídy `DataFilteringComponent` nastavená na `true`. V opačném případě direktiva `*ngIf="paginate"` zajistí schování této komponenty a zobrazí se pouze tag `<span>` obsahující počet položek v tabulce, jako je tomu na obrázku 3.3;

responzivní zobrazení – direktivy s předponou *fx-*, jako `fxFlex` a `fxShow` ukazují použití knihovny Angular flex-layout pro responzivní zobrazení zmiňované v sekci 2.3.3. Například direktiva `fxShow.lt-sm="false"` způsobí zmizení elementu `<span>` zobrazující počet položek na obrazovkách s nižším rozlišením.

3.3.2.2 TypeScript třída komponenty

Výpis kódu 3.4 třídy komponenty `DataFilteringComponent` ilustruje zmiňované předávání dat do a ven z komponenty použitím dekorovaných atributů. Proměnné označené dekorátorem `@Input()` slouží pro předávání dat komponentě a k němu opačný `@Output()` slouží k vysílání událostí z komponenty. V ukázce tak komponenta přijímá vstup, zda má být seznam stránkován `paginate`, velikost `count` a počet stran `pageSize` a upozorňuje ostatní komponenty o změně prostřednictvím `updateOptions` (například pokud uživatel klikne na další stránku nebo zadá text do vyhledávacího pole).

Data získaná od své rodičovské komponenty pak dále poskytuje komponentně `<mat-paginator>`, jak je vidět v HTML šabloně komponenty (viz výpis kódu 3.3). Přijímání veškerých dat dekorátory `@Input()` je hlavní odlišující znak prezentačních komponent od aplikačních, které data získávají ze služeb.

```
@Component({
  selector: 'app-data-filtering',
  templateUrl: './data-filtering.component.html',
  styleUrls: ['./data-filtering.component.scss']
})
export class DataFilteringComponent {
  @Input() paginate: boolean;
  @Input() pageSize: number;
  @Input() count: number;
  @Output() updateOptions = new EventEmitter<CollectionOptions>();

  onSearch(searchString: string) {
    this.updateOptions.emit({search: searchString});
  }

  onPageChanged(page: PageEvent) {
    this.pageSize = page.pageSize ? page.pageSize : this.pageSize;
    this.updateOptions.emit({
      limit: this.pageSize,
      offset: page.pageIndex ? page.pageIndex * page.pageSize : 0
    });
  }
}
```

Výpis kódu 3.4: Ukázka prezentační komponenty DataFiltering

3.3.3 Interceptory

Angular má vestavěné mechaniky pro úpravu HTTP požadavků, nazývané interceptory, které zjednodušují kód a umožňují definovat globální chování při každém požadavku namísto duplikování kódu při každém volání např. metody `GET` HTTP klienta. Jednou ze situací, ve které je výhodné použít interceptory je například caching nebo doplnění hlaviček a autentizačních tokenů.

3.3.3.1 Caching

Třída `CachingInterceptor implements HttpInterceptor` implementuje metodu `intercept`, ve které dle konfigurace uloží odpověď serveru na daný požadavek do cache (kterou zprostředokovává služba `CachingService`). Endpointy, které lze uložit do cache jsou definovány v globální konfiguraci prostředí. Strategie ukládání odpovědí do cache jsou popsány v sekci 2.2.7 věnující se návrhu cache.

3.3.3.2 Doplnění URL o ID výukové aplikace

Pro požasdavky zaslané na server je potřeba v URL uvést ID výukové aplikace, které se požadavek týká. Vzhledem k nutnosti provedení tohoto kroku u každého požadavku je výhodné použít Interceptor, který parametr doplní. To samé platí pro část URL před lomítkem, adresu backendového serveru, která je určená proměnou prostředí (například `http://example.com`). Požadavek na `/children` je tedy doplněn na `http://example.com/eduapps/1/children`, za předpokladu, že je zvolena výuková aplikace s ID 1.

Kapitola 4

Testování

Pro ověření, že je aplikace navržena a funguje správně, je vhodné podrobit ji různým druhům testů. Právě tomu je věnována tato kapitola, která podrobně rozebírá testy, které byly provedeny a jejich účel.

4.1 Plán testování

Práce se zaměřuje na webové uživatelské rozhraní (frontend) a API pro komunikaci se serverem. Při testování se tedy věnuji primárně těmto částem:

1. komunikace backendu a frontendu
2. funkčnost webového rozhraní
3. uživatelská přívětivost webového rozhraní

4.1.1 Testovací metody a druhy testů

V dnešní době existuje nepřeberné množství druhů testů, každý zaměřený na jiné aspekty systému. Testy se dají dělit podle přístupu ke kódu na dva základní typy:

black-box – není známa implementace, testy kontrolují pouze, jak se systémová komponenta chová na jejím rozhraní. Výhodou je, že nejsou ovlivněné změnou implementace;

white-box – je znám kód, testují se vlastnosti implementace, jsou silně ovlivněné změnou implementace (při změně kódu je nutné přepisovat i testy).

Testy se také dělí podle rozsahu a předmětu testování (obdobně také v [39, str. 499–500]):

jednotkové (unit) testování – jak již název napovídá, zabývá se pouze jednou jednotkou, kterou testuje samostatně, tj. izolovanou od ostatních. Jednotkou může být například metoda, třída nebo jiný malý kus programu;

testování komponent – testování většího kusu kódu, na kterém spolupracuje více programátorů nebo týmů. V kontextu celého projektu může být za komponentu považován např. celý backend. Podobně jako u jednotkových testů je komponenta testována samostatně, izolovaná od ostatních komponent a podsystémů;

integrační testování – testuje především, jak dvě nebo více částí systému (třídy, komponenty, podsystémy) komunikují mezi sebou;

regresní testování – opakování stejné sady testovacích případů za účelem zjištění, zda se nerozbila některá z částí kódu, která dříve fungovala;

systémové testování – testuje všechny části systému jako celek, včetně integrace s jinými systémy, na kterých aplikace závisí. Předmětem testování je bezpečnost, výkon, odezva a další aspekty, které se obtížně testují ostatními druhy testů.

4.1.2 Vybrané druhy testů

Protože hlavním cílem této práce a softwarového projektu je dodat zadavateli funkční prototyp, byl výběr testů přizpůsoben časovým možnostem projektu. Otestovat všechny aspekty systému může být velmi obtížné a proto je potřeba zohlednit přínos testů vzhledem k jejich náročnosti. Vybrány byly následující čtyři druhy testů:

testování REST API rozhraní backendu – tato sada testů by se dala charakterizovat jako black-box testování komponent, backend je zde testován izolovaně s testovací databází obsahující testovací data. Jejich snahou je zjistit, zda backend korektně implementuje navržené rozhraní;

unit testování frontendu – cílem je ověřit funkčnost jednotlivých aspektů architektury a implementace frontendu, vzhledem k silné závislosti na implementaci se jedná o white-box jednotkové testy;

end-to-end (konec-konec) testování frontendu – testování systému jako celku tak, jak ho používá uživatel. V praxi jde o automatizované procházení webového rozhraní většinou podle jednotlivých případů užití. Svým charakterem spadají mezi integrační nebo systémové testy, neboť zkoumají celou aplikaci včetně integrace a komunikace všech podsystémů;

uživatelské testování s učitelkami – simulace používání aplikace v reálných podmínkách za účelem ověření, zda se uživatel chová dle očekávání a zda je uživatelské rozhraní navrženo správně a přívětivě.

Vybrané druhy testů budou podrobněji samostatně popsány později. Regresní, výkonové a další testy byly vynechány kvůli jejich náročnosti, nízkému přínosu pro vývoj prvotní verze aplikace a časovému omezení projektu.

4.2 Testování REST API rozhraní backendu

Důvodem psaní těchto testů byla potřeba ověřit, že backend dodává správná data ve správném formátu podle dokumentace. Testy se skládají z několika sad HTTP požadavků a ověření, že se API chová dle očekávání.

4.2.1 Postman

Jako vhodný nástroj pro psaní tohoto druhu testů byl vybrán Postman [40]. Jedná se o testovací nástroj, který umožňuje snadné odesílání HTTP požadavků a testování odpovědí pomocí JavaScriptu.

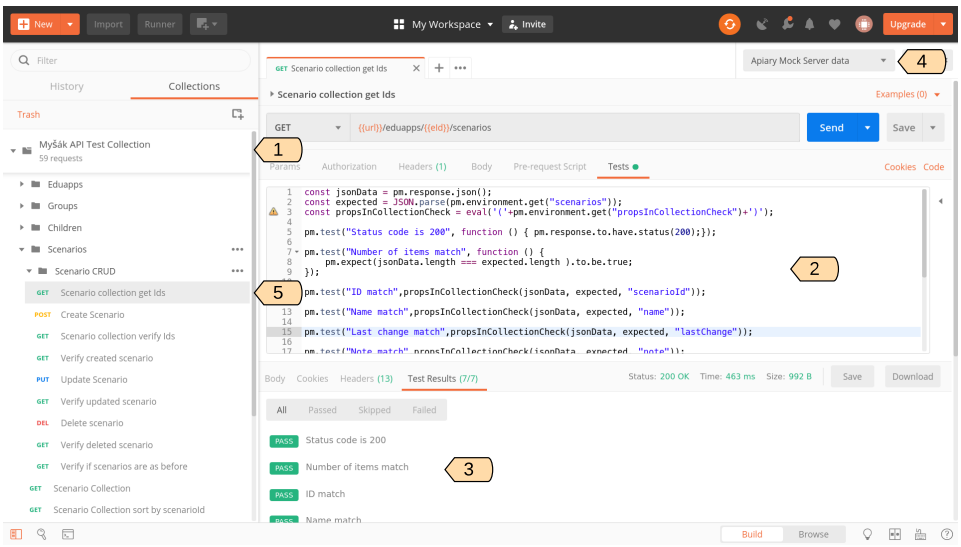
Požadavky jsou definované v tzv. kolekci, která obsahuje celou sadu testů, které je možné ještě dále dělit do složek. Před každým testem, složkou nebo kolekcí je pak možné definovat tzv. pre-request script, kus kódu, který slouží jako příprava před odesláním samotného požadavku, složky nebo celé kolekce požadavků.

Dále nástroj obsahuje dva prostory pro ukládání proměnných: globální proměnné a proměnné prostředí. Prostředí může být několik a lze mezi nimi přepínat, což umožňuje definovat několik různých situací pro testování.

Nevýhodou, která se projevila až v průběhu testování je nemožnost snadné definice funkce sdílené mezi různými testy. To vede na zbytečné duplikace kódu a proto jsem se rozhodl situaci vyřešit podobně jako v odpovědi na fóru Stack Overflow [41], tedy serializací funkcí do textového formátu, uložení do globálních proměnných a následné evaluace při jejím používání. To je velmi kostrbatý a nepohodlný způsob, bohužel je to však jediná možnost, jak zamezit opakujícímu se kódu při větším množství podobných testů.

4.2.2 Realizace

Při psaní těchto testů byl velmi užitečný mockovací server poskytovaný automaticky od Apiary. Snadno jsem na něm totiž mohl vyzkoušet, zda testy fungují správně. Z toho důvodu jsem se rozhodl vytvořit dvě prostředí, jedno pro Apiary a druhé pro backendový server. Ta obsahují kromě rozdílné URL API také rozdílné sady testovacích dat. Pro Apiary musela být nutně použita stejná data jako v dokumentaci, avšak pro reálné testování backendu by tato



Obrázek 4.1: Ukázkový test v nástroji Postman [40, snímek pořídil autor]

sada byla nedostatečná. Například každá kolekce totiž obsahuje pouze jednu entitu. Proto jsem se rozhodl vytvořit vlastní sadu testovacích dat.

Kolekce testů a prostředí lze z postmana snadno vyexportovat a uložit ve formátu json. Pro testy spolu s testovacími daty byl vytvořen nový GIT repozitář, který byl následně sdílený s backendovými vývojáři. Na obrázku 4.1 je vidět jeden z testů a struktura aplikace Postman.

1 Kolekce testů – všechny testy pro API mezi backendem a frontendem aplikace *Myšák*, dále členěné do složek podle jednotlivých entit;

2 Test – ukázka kódu JavaScriptového testu pro seznam scénářů;

3 Výsledek testu – ukazatel, zda implementace splňuje všechny části testu;

4 Prostředí – obsahuje specifické proměnné a testovací data pro jednu testovací situaci, kliknutím na toto tlačítko lze mezi prostředími přepínat a tím změnit testovací data, server apod.;

5 Požadavek – jeden z požadavků, který je testován. Požadavky jsou spouštěny sekvenčně shora dolů tak, jak jsou v kolekci vytvořeny, je tedy možné, aby na sebe navazovaly a testovaly komplexnější vlastnosti API. Lze tedy například vytvořit nový scénář, následně zkontrolovat, zda se nachází v seznamu, smazat ho a opět ověřit, zda byl smazán.

4.2.3 Automatizace

Vyexportované testy a kolekce lze snadno automatizovat, tedy není nutné mít nainstalovaného grafického klienta a spouštět testy manuálně. Pomocí CLI (z anglického Command Line Interface) nástroje `newman` je možné načíst kolekci, příslušné prostředí a spustit testy na příkazové řádce. To lze pak snadno začlenit do procesu automatického sestavení a nasazení aplikace pomocí continuous integration. Začlenění provedl backendový vývojář Jakub Topič.

4.3 Unit testování frontendu

Jednotkové testování je důležité především z důvodu zajištění funkčnosti jednotlivých částí, ze kterých se aplikace skládá. Díky izolovaným testům malých kusů kódu (jednotlivých metod, služeb, komponent, apod.) je poté snadné odhalit původ chyb. Oproti tomu celkové testování systému (typicky formou black-box), chybu sice může odhalit, nicméně je obtížnější zjistit odkud pochází.

4.3.1 Jasmine a Karma runner

Angular již ve výchozím stavu po instalaci nabádá vývojáře k tvorbě jednotkových testů. Obsahuje totiž předinstalovaný testovací framework Jasmine a vždy při tvorbě nové komponenty nebo služby automaticky také vytvoří šablonu pro test (viz souborová struktura komponenty v sekci 3.3.2).

Všechny jednotkové testy se jednoduše spustí příkazem `ng test`, který deleguje svou práci nástroji Karma runner. Následně se otevře okno prohlížeče, které zobrazuje přehled výsledků a příkaz `ng test` v pozadí pozoruje změny v testových souborech. Pokud se jeden nebo více souborů změní, testy jsou přehodnoceny a v okně prohlížeče se objeví nové výsledky.

4.3.2 Realizace

Při testování jsem postupoval především podle oficiální dokumentace Angularu, kde je popsána celá řada situací a problémů, které mohou nastat.

Jednodušší bylo testování služeb, které bylo velmi podobné klasickému jednotkovému testování známemu z objektově orientovaného způsobu programování. Jedinou „nepříjemností“ bylo doplňování závislostí na jiné služby a jejich záměna za falešné mocky.

Problémy s testováním komponent

Testování komponent se ukázalo mnohem komplikovanější, než se na první pohled mohlo zdát. Objevuje se hned několik problémů, nejdůležitější z nich jsou:

asynchronní data – nejen, že komponenty mohou záviset na službách, ale tyto služby typicky vrací asynchronní data, nejčastěji typ `Observable` (krátce zmíněný také v sekci 2.3.1). Tento typ se vyznačuje tím, že může „vyzářit/vysílat“ data hned několikrát a komponenta se podle toho může zachovat různě. Této vlastnosti se využívá například při strategii ukládání dat do mezipaměti, která napřed využívá cache a následně pošle dotaz na server a aktualizuje data, popsané v sekci 2.2.7. Pro usnadnění lze použít RxJS Marbles – pomocné funkce, které umožňují snadno vytvořit `Observable` objekty se specifickým chováním a simulovat tak různé asynchronní situace a toky dat, nicméně stále tento problém zvyšuje časovou náročnost a komplexitu testování a opět je nutné dodat komponentě falešné mockové služby;

vnořené komponenty – typické pro Angular je vnořování komponent principem vkládání a předávání dat mezi rodičem a potomkem pomocí tzv. data bindingu. Komponenta může obsahovat mnoho hluboko zanořených potomků a většinou je proto nutné bezprostřední potomky nahradit falešnými;

předávání dat mezi komponenty – dobrý test by měl ověřit, zda komponenta svým potomkům předává správná data a naopak zda testovaná komponenta správně přebírá data od své rodičovské komponenty. To opět vede na mockování komponent, a to rodičovských i potomků.

Kvůli těmto problémům, které značně zvyšovaly náročnost testování, bylo upuštěno od kompletního pokrytí a byla otestována pouze část aplikace, a to veškeré služby a pouze vybrané komponenty. Test metody `getChild()` služby `ChildService` (viz výpis kódu 4.1) naznačuje způsob použití frameworku Jasmine (zejména metody `describe()`, `it()`, `expect()` a `beforeEach()`).

4.3.3 Pokrytí kódu testy

Z výše zmíněných důvodů bylo pokrytí jednotkovými testy omezeno pouze na služby a některé prezentační komponenty. Statistiky testů udávají pokrytí přibližně 53 % řádek kódu, to je však mírně zkreslené, protože některé řádky jsou pokryté automaticky, např. definice modelů.

4.4 End-to-end testování frontendu

Pro ověření, že se systém jako celek chová správně jsou nejvhodnější tzv. end-to-end (česky konec-konec) testy. Jak již název napovídá, testují celý systém napříč od uživatele až po databázi. Pro webové aplikace se často využívají automatické testy, které procházejí webovou stránku, klikají na tlačítka a vyplňují dotazníky tak, jak by to dělal uživatel. Jedná se o black-box testy, tedy nezávislé na implementaci.

```
describe('#getChild', () => {
  beforeEach(() =>
    apiServiceSpy.httpGet.and.returnValue(child$));

  it('should call httpGet with id of child', () =>
    testHttpGet(() => childService.getChild(42),
      apiServiceSpy.httpGet,
      '/children/42'));

  it('should return observable of child', () =>
    childService.getChild(1)
      .subscribe(res =>
        expect(res).toEqual(child), fail));

  it('should return error on http error', () =>
    testError(() => childService.getChild(42),
      apiServiceSpy.httpGet));
});
```

Výpis kódu 4.1: Ukázka části unit testu ChildService

4.4.1 Protractor a Jamine

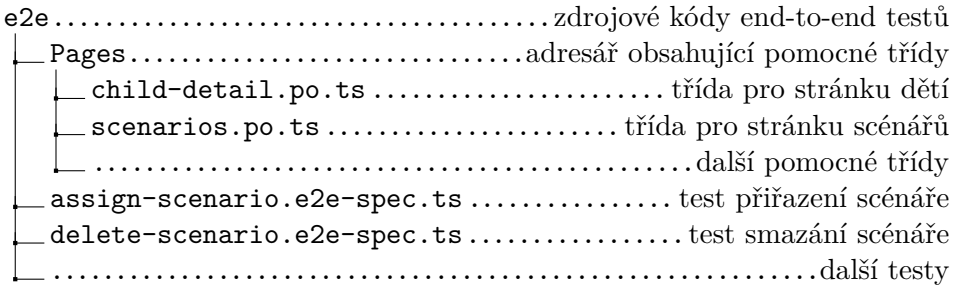
Jako technologie pro tento druh testu byl zvolen již předinstalovaný framework Protractor [42] ve spojení s Jasmine [43]. Výhodou je snadné použití s minimální nutností konfigurace (vše je již připravené od tvůrců Angularu) nebo jednoduchá syntaxe testů, stejná jako u jednotkových.

4.4.2 Realizace

Protože backend v této fázi ještě nebyl hotový, byly testy založeny opět na mockové databázi v paměti (viz sekce 2.2.8). Z toho důvodu byla rozšířena o některé dosud nepodporované operace tak, aby byl mockový backend z uživatelského pohledu nerozeznatelný od skutečného. Na funkčnosti testu to nic nemění, stačí, aby pak skutečný backend dodržel stejnou sadu testovacích dat.

Jednotlivé testy vycházejí z případů užití. Každý případ je podle svého scénáře nasimulován automatizovaným průchodem aplikací ve webovém prohlížeči. Následně je zkontrolováno, zda byla operace úspěšně vykonána (např. zda přiřazení scénáře dítěti mu skutečně přiřadilo scénář).

Protože jsou testy nezávislé na implementaci, jsou umístěné do samostatné složky mimo zdrojové kódy aplikace. Souborová struktura testů je vidět na obrázku 4.2.



Obrázek 4.2: Souborová struktura end-to-end testů

4.5 Uživatelské testování s učitelkami

Pro ověření správnosti návrhu uživatelského rozhraní, přívětivosti a odhalení případných chyb byl web otestován testery věkově a profesně shodnými s budoucími uživateli.

Cílovou skupinou uživatelů pro webového klienta pro platformu *Myšák* jsou učitelé a učitelky mateřských škol. Proto bylo uživatelské testování provedeno za spolupráce s univerzitní mateřskou školou Lvíčata. Průběh testování se skládal z více kroků:

1. příprava testovacího scénáře a očekávaného průběhu
2. předběžné testování se studenty
3. oprava hlavních nedostatků na základě zpětné vazby studentů
4. uživatelské testování s učitelkami z MŠ Lvíčata
5. analýza a zjištění problémů
6. oprava problémů a zlepšení uživatelského rozhraní

První tři kroky jsou přípravné, následuje samotné testování a poté zpracování výsledků. Jednotlivé části budou nyní podrobně rozepsány.

4.5.1 Příprava

Základem pro správný průběh testování a zajištění objektivních výsledků je nutnost dodržet stejné podmínky pro každého ze zúčastněných testerů. Z toho důvodu byl sepsán testovací scénář, který krok po kroku popisuje průběh testu.

4.5.1.1 Testovací scénář

Scénář by měl obsahovat veškeré aktivity týkající se celého procesu testu, tedy i přivítání účastníků, uvedení do problematiky apod. Tím je zamezeno chybám způsobených tím, že každý tester obdrží jiné informace nebo instrukce.

Nejprve byla vytvořena prvotní verze scénáře, která byla postupně vylepšena na základě zpětné vazby od studentů. Testování se studenty bude podrobně popsáno později. Scénář vychází ze způsobů užití, které jsou popsány v sekci 1.2.4. Protože otestovat všechny případy užití by pro testery bylo příliš časově náročné, byly zvoleny následující případy:

- UC1: Vytvořit scénář
- UC10: Přiřadit scénář skupině dětí
- UC5: Upravit frontu scénářů dítěte

A samozřejmě také případy, které jsou součástí vybraných: Zobrazit seznam scénářů (UC6) a Zobrazit seznam dětí (UC7).

Testování se studenty ukázalo, že je časově možné testovací scénář ještě rozšířit. Proto byly do finální verze zahrnuty ještě tyto dva případy:

- UC4: Smazat scénář
- UC5: Upravit frontu scénářů dítěte

Finální verzi scénáře a očekávaného průchodu lze nalézt v příloze této práce.

4.5.1.2 Testování se studenty

V rámci přípravy bylo uživatelské rozhraní předběžně otestováno za pomoci studentů Fakulty informačních technologií Českého vysokého učení technického v Praze na cvičení z předmětu *Tvorba uživatelského rozhraní*. Cílem cvičení bylo ukázat, jak probíhá uživatelské testování. Toho bylo využito ve prospěch této práce.

Testování probíhalo průběžně, na každém paralelním cvičení s jedním studentem, celkem se tedy zúčastnili čtyři. Každý z nich dostal stejný úvod do problematiky, avšak mírně se lišila sada úkolů, kterou měl zúčastněný ve webovém rozhraní vykonat. Důvodem bylo experimentování se scénářem testování, upravení časové náročnosti a srozumitelnosti testu.

Tento postup měl hned několik přínosů pro výsledný test. Nejen, že bylo možné test předem ověřit, ale i vylepšit jeho průběh, cíle a také odstranit nejzásadnější chyby uživatelského rozhraní aplikace. Dalším důležitým cílem bylo zjistit, jak přistupovat k testování webového klienta bez funkčního backendu, který je stále ve fázi vývoje, a proto se testování odehrává na falešných datech uložených pouze v operační paměti (viz sekce 2.2.8 věnující se simulaci API serveru). To vše na základě zpětné vazby od pozorujících studentů a experta v oboru tvorby uživatelského rozhraní doc. Ing. Jana Schmidta, Ph.D.

4.5.1.3 Zpracování výsledků předběžného testování

Předchozí přípravný test vedl ke změně hned několika věcí:

1. úprava testovacích dat tak, aby byla smysluplnější, např. umístění dětí do více skupin současně způsobovalo zmatení uživatelů, tento aspekt aplikace však nebyl předmětem testování,
2. úprava testovacího scénáře, rozšíření o další dva případy užití,
3. vytvoření kartiček obsahujících jednotlivé úkoly testera a požadovaná vstupní data (název scénáře, který má tester vytvořit, jméno dítěte, jehož frontu má upravit apod.),
4. úprava uživatelského rozhraní a dat:
   a) odstranění stránkování ze seznamů, které způsobovalo zbytečné zmatení uživatelů (především při označování dětí z celé skupiny na stránce přiřazení scénáře). Vzhledem k běžnému počtu dětí v jedné mateřské škole byl potenciální dopad na výkon serveru označen za zanedbatelný. Pokud by však v budoucnu byl server příliš zatěžován, bylo umožněno snadné opětovné zapnutí stránkování,
   b) zlepšení chování mockovaného backendu, ukládání některých změn do paměti (umožněno například smazání scénáře),
   c) změna ikon některých tlačítek, které neodpovídaly očekávání uživatelů.

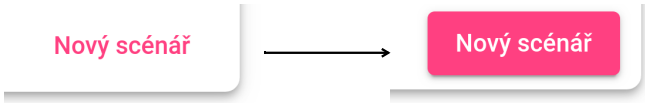
4.5.2 Testování s učitelkami

Po důkladné přípravě konečně mohlo proběhnout uživatelské testování. Uskutečnilo se v mateřské škole Lvíčata, se čtyřmi učitelkami (testovacími subjekty). Učitelky byly mladší věkové kategorie a přesně vystihly cílovou skupinu uživatelů aplikace.

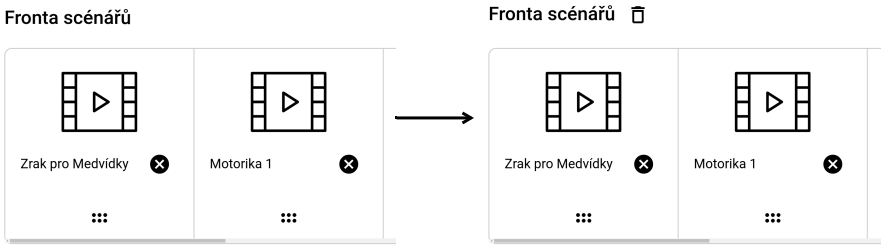
4.5.2.1 Průběh

Pro účely pozdější analýzy byl pořízen videozáznam z externí kamery a webkamery a také bylo na testovacím počítači nastaveno snímání obrazovky. Časová vytíženost učitelek a nutnost věnovat se dětem způsobila mírné odchýlení od testovacího scénáře, a to sice namísto společné prezentace a úvodu do problematiky, probíhala tato část s každým subjektem samostatně. Také nebylo možné zajistit testovací místnost a proto se testování odehrávalo nejprve v jedné třídě se dvěma subjekty a následně se zbylýma dvěma v jiné třídě.

Aplikace byla spuštěna na notebooku s rozlišením 1920×1080 (Full HD) ve webovém prohlížeči Google Chrome. Po průchodu aplikací a splnění všech pěti úkolů dle scénáře byly učitelky vyzvány pro vyplnění krátkého dotazníku zaměřeného na hodnocení obtížnosti jednotlivých úkolů.



Obrázek 4.3: Zvýraznění tlačítek



Obrázek 4.4: Přidání tlačítka na vyčištění fronty

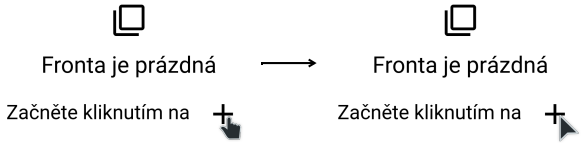
4.5.2.2 Zpracování výsledků a oprava nedostatků

Některým zúčastněným chvíli trvalo se v aplikaci zorientovat, posléze však splnili úkoly bez větších potíží. V některých situacích byla potřeba asistence moderátora, ale šlo převážně o objasnění zadání úkolu nebo domény. Po průchodu scénářem následovaly otázky moderátora, jejichž cílem bylo zjistit nejasnosti a nedokonalosti systému na základě chování subjektu. Objevily se některé drobné i závažnější nedostatky. Ty drobné se podařilo snadno vyřešit:

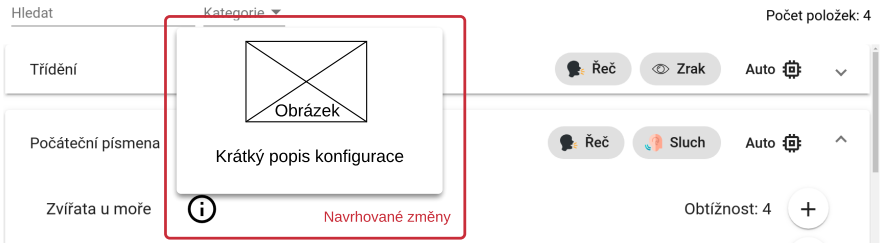
1. tlačítko *Nový scénář* na stránce scénářů bylo příliš nenápadné a všichni zúčastnění by ocenili jeho zvýraznění. V rámci jednotného vzhledu byla zvýrazněna všechna podobná tlačítka v systému – *Uložit*, *Přiřadit na konec fronty* apod. (viz obrázek 4.3),

2. položky ve frontě dítěte nešlo smazat jedním kliknutím a všichni zúčastnění by ocenili možnost vyčistit frontu jedním kliknutím. Proto bylo přidáno tlačítko umožňující toto provést (viz obrázek 4.4),

3. při najetí myší na ikonu + v nápovědě prázdné fronty se změnil kurzor na ručičku indikující, že na ikonu lze kliknout, to však nelze. Tento problém byl triviálně vyřešen pomocí CSS `cursor: default` (viz obrázek 4.5).



Obrázek 4.5: Oprava matoucího kurzoru myši



Obrázek 4.6: Navrhované změny v seznamu úkolů a konfigurací

Někteří zúčastnění pomohli odhalit i možné rozsáhlejší nedostatky, které více zasahují do architektury aplikace a jejichž oprava by byla náročnější:

1. Na obrazovce úpravy scénáře se seznamem všech úkolů a konfigurací by mohly jednotlivé konfigurace obsahovat krátký popis nebo obrázek. To by pomohlo uživatelům snadno poznat, jak vypadá která konfigurace bez nutnosti pamatovat si ji podle názvu (viz obrázek 4.6),

2. Přiřazování scénáře celé skupině by mohlo být řešeno jinak. Někteří zúčastnění se nejprve pokoušeli přiřadit scénář celé skupině v seznamu dětí, nikoliv kliknutím na tlačítko *Přiřadit dětem* u konkrétního scénáře v seznamu scénářů.

Pro ověření, že je uživatelé skutečně konzistentně shledávají problémové, by však bylo potřeba extenzivnější testování s více subjekty. Jejich oprava by pak vyžadovala zásahy do návrhu, databáze nebo API. Z časových důvodů je proto ověření a oprava takových chyb mimo rozsah projektu a jejich prošetření je ponecháno na budoucí rozvoj aplikace.

Obrázek 4.7: Jedno z testovacích stanovišť v mateřské škole [fotografie autor]

Závěr

Cílem této práce bylo navrhnout a implementovat webového klienta pro výukovou platformu *Myšák*. Výsledek práce splňuje cíl a zapadá do softwarového projektu dále se skládajícího z backendového serveru a mobilní aplikace.

Nejprve byly sbírány požadavky zadavatele, jejichž analýza a zpracování vedla k návrhu výsledné aplikace. Pro komunikaci mezi frontendovou a backendovou částí bylo nutné navrhnout rozhraní API. Proto se analýza podrobně zabývala různými technologiemi a přístupy k API vhodnými pro projekt. Vítězem analýzy se stal nejrozšířenější přístup REST, který byl následně použit při návrhu. Během tvorby rozhraní API byl kladen důraz zejména na jednoduchost a konzistenci. Výsledné API je snadno čitelné pro člověka a jeho rozsáhlá dokumentace s praktickými ukázkami umožňuje snadnou komunikaci a vyšší úroveň porozumění mezi vývojáři.

Způsoby použití aplikace a její jednotlivé obrazovky byly důkladně navrženy a následně implementovány. Během toho byl kladen důraz na dodržení principů dobrého uživatelského rozhraní. Také byla snaha o responzivitu rozhraní, jejíž naplnění způsobilo, že aplikace je plnohodnotně použitelná na počítači i dotykových zařízeních s různým rozlišením displeje.

Aplikace nebyla ponechána ani bez důkladného testování hned několika způsoby. Korektnost backendové implementace je testována robustními testy API. Webový klient byl podroben uživatelským testům s učitelkami z mateřské školy a částečně i jednotkovým testům kódu. Pro kontrolu funkčnosti celého systému byly napsány end-to-end testy, které automatizovaně procházejí webovou aplikací podle scénářů případů užití.

Budoucí rozvoj

Přes veškerou snahu backendových vývojářů se nepodařilo včas dokončit serverovou část. Celý vývoj frontendu tedy probíhal izolovaně za pomoci Apiary mockového serveru a Angularu, jež obsahuje možnosti připravené právě pro tyto situace. Pro další vývoj aplikace je tedy klíčové dokončení implementace

backendového serveru a jeho propojení s mobilní aplikací. Frontendová část byla navržena tak, aby přepnutí z testovacích dat na ostrý backend bylo triviální, přepsáním dvou řádků kódu. End-to-end testy budou také okamžitě fungovat, pokud server dodrží sadu očekávaných testovacích dat.

Samotná frontendová část by mohla být obohacena o další jednotkové testy, případně otestována více uživateli za účelem zlepšení uživatelského komfortu. Také by měla být obohacena o autentizaci uživatelů, která záměrně nebyla součástí požadavků z důvodu integrace do ostatních systémů zadavatele.

Další zajímavé rozšíření by bylo sbírání podrobnějších statistik z odehraných úkolů a jejich přehledné zobrazení učitelům. Statistiky by mohly být zaměřeny na jednotlivé oblasti vývoje dítěte a vyhodnocovat schopnosti dětí v těchto oblastech. To by umožnilo učitelům snadnější zjištění slabších a silnějších stránek dítěte, na jejichž základě by mohli přímo v aplikaci vytvářet scénáře.

Vytvořená aplikace pomůže v rozvoji digitální výuky v mateřských školách a umožní individuální a efektivní přístup k vývoji schopností dětí.

Osobní přínos

Během tvorby aplikace jsem okusil mnoho nových, zajímavých věcí, konkrétně jsem poznal nové technologie, zejména Angular, TypeScript a naučil se s nimi efektivně pracovat. Zjistil jsem, že svět API je rozmanitý a mnoho lidí má odlišné názory a způsoby, jak navrhnout rozhraní správně. Vyzkoušel jsem si práci v týmu na větším projektu, kde je naprosto klíčová komunikace a spolupráce. To mělo i své negativní stránky, protože má práce závisela na ostatních částech systému, které se nepodařilo dokončit. To se mi však podařilo překonat a přesto vyvinout ucelenou komponentu, která snadno zapadne do finálního systému.

Při testování frontendu jsem zjistil, že ne vždy se vyplatí vkládat velké úsilí a mnoho času do činností, které nemají dostatečný přínos, ale je lepší soustředit se na jiné, důležitější. To samé se přihodilo i při ladění vzhledu stránky, kde mé přílišné puntičkářství a upravovaní detailů způsobilo zbytečné časové zdržení.

Obrovským přínosem je však pro mě dokončení větší práce, vstup do pro mě zcela nové, neznámé oblasti informačních technologií, kterou jsem svými silami zvládl pochopit. Také bylo radostí pracovat na projektu, který má reálné smysluplné využití a pomůže rozvoji těch nejdůležitějších, kteří mají ve svých rukou naši budoucnost, dětí.

Seznam použité literatury

1. SÝKORA, Tomáš [online hovor hangouts]. TechSophia, s. r. o., 2018.
2. WIEGERS, Karl E. When telepathy won't do: Requirements engineering key practices. *Cutter IT Journal* [online]. 2000, roč. 13, č. 5 [cit. 2018-10-29]. Dostupné z: `http://www.mathcs.emory.edu/~cengiz/cs540-485-soft-eng-fa14/resources/telepathy.pdf`.
3. GOOGLE. *Google Chrome. Version 74* [software]. 2019 [cit. 2019-04-27]. Dostupné z: `https://www.google.com/chrome/`.
4. MOZILLA CORPORATION. *Firefox. Version 66* [software]. 2019 [cit. 2019-04-27]. Dostupné z: `https://www.mozilla.org/cs/firefox/`.
5. GOOGLE; OPEN-SOURCE KOMUNITA. *Angular. Version 7.2.12* [software]. 2019 [cit. 2019-04-27]. Dostupné z: `https://angular.io/`.
6. DJANGO SOFTWARE FOUNDATION AND INDIVIDUAL CONTRIBUTORS. *Django. Version 2.2* [software]. 2019 [cit. 2019-04-27]. Dostupné z: `https://www.djangoproject.com/`.
7. COCKBURN, Alistair. *Writing effective use cases*. Addison-Wesley Professional, 2000. ISBN 9780201702255.
8. GraphQL is the better REST. *V: How to GraphQL* [online] [cit. 2018-10-22]. Dostupné z: `https://www.howtographql.com/basics/1-graphql-is-the-better-rest/`.
9. SMITH, Tom. APIs: RPC versus REST versus GraphQL. *V: DZone – vývojářský magazín* [online]. 2018 [cit. 2018-10-22]. Dostupné z: `https://dzone.com/articles/rpc-versus-rest-versus-graphql`.
10. Common Cases When Using SOAP Makes Sense. *V: Nordic APIs - magazín zaměřený na vývoj API* [online]. 2015 [cit. 2018-10-22]. Dostupné z: `https://nordicapis.com/common-cases-when-using-soap-makes-sense/`.

11. *HATEOAS Driven REST APIs* [online] [cit. 2018-10-29]. Dostupné z: https://restfulapi.net/hateoas/.

12. STURGEON, Phil. Understanding RPC, REST and GraphQL. *V: APIs You Won't Hate – blog zaměřený na vývoj API* [online]. 2018 [cit. 2018-10-22]. Dostupné z: https://blog.apisyouwonthate.com/understanding-rpc-rest-and-graphql-2f959aadebe7.

13. HANÁK, Drahomír. Stopařův průvodce REST API. *V: ITnetwork – IT magazín* [online]. 2013 [cit. 2018-10-22]. Dostupné z: https://www.itnetwork.cz/nezarazene/stoparuv-pruvodce-rest-api.

14. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures* [online]. 2000 [cit. 2018-10-29]. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Disertační práce. University of California, Irvine.

15. NĚMEC, Zdeněk. REST vs. GraphQL: A Critical Review. *V: Good API – blog zaměřený na vývoj API* [online]. 2018 [cit. 2018-10-22]. Dostupné z: https://medium.com/good-api/rest-vs-graphql-a-critical-review-5f77392658e7.

16. SANTOS, Wendell. Which API Types and Architectural Styles are Most Used? *V: ProgrammableWeb* [online]. 2017 [cit. 2018-10-22]. Dostupné z: https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26.

17. GraphQL Best Practices. *V: GraphQL.org – oficiální GraphQL dokumentace* [online dokumentace] [cit. 2018-10-29]. Dostupné z: https://graphql.org/learn/best-practices/.

18. TARVAINEN, Jani. Versioning an API in GraphQL vs. REST. *V: Symfony Finland* [online]. 2016 [cit. 2018-10-24]. Dostupné z: https://symfony.fi/entry/versioning-an-api-in-graphql-vs-rest.

19. CURRY, Joshua. Introduction to API Versioning Best Practices. *V: Nordic APIs – magazín zaměřený na vývoj API* [online]. 2017 [cit. 2018-10-25]. Dostupné z: https://nordicapis.com/introduction-to-api-versioning-best-practices/.

20. Share of Facebook users worldwide who accessed Facebook via mobile from 2013 to 2018 [online graf]. 2019 [cit. 2019-04-01]. Dostupné z: https://www.statista.com/statistics/380550/share-of-global-mobile-facebook-users/.

21. HRONČOK, Miroslav. *RESTful API v jazyce Python.* České vysoké učení technické v Praze. Vypočetní a informační centrum, 2016. Magisterská práce.

22. BREDEHÖFT, Sebastian. *drf-hal-json. Verze 0.9.1* [software]. GitHub [cit. 2019-03-15]. Dostupné z: `https://github.com/seebass/drf-hal-json`.

23. KASALICKÁ, Kateřina; JEŽEK, Radek; TOPIČ, Jakub; TISLICKÝ, Jan; ŠLEJTR, Ondřej; VÁCLAVIKOVÁ, Zuzana. Návrh serverového modulu – 2. iterace SP2. 2018. České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství. Vedoucí projektu Jiří Chludil.

24. KASALICKÁ, Kateřina; JEŽEK, Radek; TOPIČ, Jakub; TISLICKÝ, Jan; ŠLEJTR, Ondřej; VÁCLAVIKOVÁ, Zuzana. Analýza serverového modulu – 1. iterace SP2. 2018. České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství. Vedoucí projektu Ing. Jiří Chludil.

25. Architecture overview. *V: Angular.io – oficiální Angular dokumentace* [online dokumentace] [cit. 2019-04-01]. Dostupné z: `https://angular.io/guide/architecture`.

26. *GlobalStats* [online]. StatCounter, 2019 [cit. 2019-04-01]. Dostupné z: `http://gs.statcounter.com/`.

27. Desktop Browser Market Share Czech Republic. *StatCounter GlobalStats – statistický portál* [online statistika]. 2019 [cit. 2019-04-01]. Dostupné z: `http://gs.statcounter.com/browser-market-share/desktop/czech-republic/#monthly-201801-201901`.

28. Mobile & Tablet Browser Market Share Czech Republic. *StatCounter GlobalStats – statistický portál* [online statistika]. 2019 [cit. 2019-04-01]. Dostupné z: `http://gs.statcounter.com/browser-market-share/mobile-tablet/czech-republic/#monthly-201801-201901`.

29. Screen Resolution Stats Czech Republic. *StatCounter GlobalStats – statistický portál* [online statistika]. 2019 [cit. 2019-04-01]. Dostupné z: `http://gs.statcounter.com/screen-resolution-stats/all/czech-republic/#monthly-201801-201901`.

30. NIELSEN, Jakob. 10 usability heuristics for user interface design. *Nielsen Norman Group*. 1995, roč. 1, č. 1.

31. GOOGLE; OPEN-SOURCE KOMUNITA. *Angular Material. Version 7.3.7* [software]. 2019 [cit. 2019-04-29]. Dostupné z: `https://material.angular.io/`.

32. LUCID SOFTWARE INC. *Lucidchart* [software]. 2019 [cit. 2019-04-28]. Dostupné z: `https://www.lucidchart.com/`.

33. *Angular Flex-Layout. Version 7.0.0-beta.24* [software]. 2019 [cit. 2019-04-29]. Dostupné z: `https://github.com/angular/flex-layout`.

34. ORACLE AND/OR ITS AFFLIATES. *Apiary. Version 5543* [software]. 2019 [cit. 2019-04-08]. Dostupné z: `https://apiary.io`.

35. *API Blueprint. Version Format 1A9* [software]. 2019 [cit. 2019-04-29]. Dostupné z: `https://apiblueprint.org`.

36. GITLAB, INC.; OPEN-SOURCE KOMUNITA. *GitLab. Version 11.10* [software]. 2019 [cit. 2019-04-29]. Dostupné z: `https://gitlab.com`.

37. JETBRAINS, S. R. O. *WebStorm. Version 2019.1.1* [software]. 2019 [cit. 2019-04-29]. Dostupné z: `https://www.jetbrains.com/webstorm/`.

38. PALANTIR TECHNOLOGIES; OPEN-SOURCE KOMUNITA. *TSLint. Version 5.11.0* [software]. 2019 [cit. 2019-04-29]. Dostupné z: `https://palantir.github.io/tslint/`.

39. MCCONNELL, Steve. *Code complete*. Pearson Education, 2004. ISBN 9780735619678.

40. POSTMAN, INC. *Postman. Version 6.7.2* [software]. 2019 [cit. 2019-04-08]. Dostupné z: `https://www.getpostman.com/`.

41. LOPATKIN, Sergej. *Answer to: How to Write Global Functions in Postman* [online] [cit. 2019-03-04]. Dostupné z: `https://stackoverflow.com/a/45678770`.

42. GOOGLE; OPEN-SOURCE KOMUNITA. *Protractor. Version 5.4.2* [software]. 2019 [cit. 2019-04-29]. Dostupné z: `https://www.protractortest.org`.

43. PIVOTAL SOFTWARE, INC.; OPEN-SOURCE KOMUNITA. *Jasmine. Version 3.4.0* [software]. 2019 [cit. 2019-04-29]. Dostupné z: `https://jasmine.github.io`.

Dodatek A

Seznam použitých zkratek

API Application Programming Interface

CI Continuous Integration

CLI Command Line Interface

CSS Cascading Style Sheets

FURPS Functionalily, Usability, Reliability, Performance, Supportability – druhy požadavků

HAL Hypertext Application Language

HATEOAS Hypermedia as the Engine of Application State

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

MSON Markdown Syntax for Object Notation

REST Representational State Transfer

RPC Remote Procedure Call

SMTP Simple Mail Transfer Protocol

SOAP Simple Object Access Protocol

SQL Structured Query Language

TCP Transmission Control Protocol

UC Use Case

UML Unified Modeling Language

URI Uniform Resource Identifier – adresa jednoho zdroje (např. `/student`)

URL Uniform Resource Locator – celá adresa včetně přístupové metody (např. `https://fit.cvut.cz/student/`), URL jsou podmožinou URI

RxJS Reactive Extensions for JavaScript

Dodatek B

Případy užití

V této příloze lze nalézt kompletní seznam případů užití, diagram případů užití a jejich další popis se nachází v sekci 1.2.4.

UC1 – Vytvořit scénář

vstupní podmínky – uživatel je přihlášen

konečný stav – v systému je nový scénář

způsob vyvolání – uživatel se rozhodne vytvořit nový scénář

scénář

1. *include* (UC7: Zobrazit seznam scénářů)
2. uživatel klikne na *Vytvořit scénář*
3. systém zobrazí rozhraní pro tvorbu scénáře
4. uživatel uloží scénář

UC2 – Upravit scénář

vstupní podmínky – uživatel je přihlášen

konečný stav – jeden scénář je upraven oproti předchozímu stavu

způsob vyvolání – uživatel se rozhodne upravit scénář

scénář

1. *include* (UC7: Zobrazit seznam scénářů)
2. uživatel vybere scénář a klikne na *Upravit*
3. systém zobrazí rozhraní pro úpravu scénáře
4. uživatel provede změny a výsledek uloží
5. systém uloží upravený scénář

UC3 – Přiřadit scénář dětem

vstupní podmínky – uživatel je přihlášen

konečný stav – dítěti/dětem je přiřazen scénář (scénáře) do fronty

způsob vyvolání – uživatel se rozhodne přiřadit scénář dětem

scénář

1. *include* (UC7: Zobrazit seznam scénářů)
2. uživatel vybere scénář, který chce přiřadit
3. systém zobrazí seznam dětí
4. uživatel vybere děti, kterým chce přiřadit scénář
5. uživatel potvrdí výběr kliknutím na *Přidat na začátek fronty*, nebo *Přidat na konec fronty*
6. systém přiřadí dětem vybraný scénář buď na konec nebo na začátek fronty dítěte, dle uživatelovy volby

UC4 – Přiřadit scénář skupině dětí

vstupní podmínky – uživatel je přihlášen

konečný stav – skupině dětí je přiřazen scénář (scénáře) do fronty

způsob vyvolání – uživatel se rozhodne přiřadit scénář definované skupině dětí

scénář

1. *include* (UC7: Zobrazit seznam scénářů)
2. uživatel vybere scénář, který chce přiřadit
3. systém zobrazí seznam dětí
4. uživatel vybere skupinu, které chce přiřadit scénář a (např. tlačítkem *Vybrat vše*) zvolí všechny děti, které patří dané skupině
5. uživatel potvrdí výběr kliknutím na *Přidat na začátek fronty* nebo *Přidat na konec fronty*
6. systém přiřadí dětem vybraný scénář buď na konec, nebo na začátek fronty dítěte dle uživatelovy volby

UC5 – Smazat scénář

vstupní podmínky – uživatel je přihlášen

konečný stav – systém smaže scénář

způsob vyvolání – uživatel se rozhodne smazat scénář

scénář

1. *include* (UC7: Zobrazit seznam scénářů)
2. uživatel vybere scénář a klikne *Smazat*
3. systém zobrazí varování a po souhlasu uživatele smaže scénář

UC6 – Upravit frontu scénářů dítěte

vstupní podmínky – uživatel je přihlášen

konečný stav – dítěti je upravena fronta scénářů

způsob vyvolání – uživatel se rozhodne změnit uživateli frontu scénářů

scénář

1. *include* (UC8: Zobrazit seznam dětí)
2. uživatel vybere dítě a zvolí *Upravit frontu*
3. systém zobrazí rozhraní pro úpravu fronty
4. uživatel provede změny a uloží frontu
5. systém vybranému dítěti upraví frontu scénářů

UC7 – Zobrazit seznam scénářů

vstupní podmínky – uživatel je přihlášen

konečný stav – systém zobrazí seznam scénářů

způsob vyvolání – uživatel klikne na hlavní stránce na *Správa scénářů*

UC8 – Zobrazit seznam dětí

vstupní podmínky – uživatel je přihlášen

konečný stav – systém zobrazí seznam dětí

způsob vyvolání – uživatel klikne na hlavní stránce na *Správa dětí*

UC9 – Hra scénáře

vstupní podmínky – uživatel je přihlášen

konečný stav – uživatel dohraje scénář a systém zaznamená statistiku

způsob vyvolání – uživatel spustí aplikaci

scénář

1. Scénář začíná, když uživatel spustí hru
2. *include* (UC10 - přihlášení do hry)
3. systém spustí uživateli nejnovější scénář v jeho frontě scénářů
4. *Výjimka (fronta je prázdná)*: Aplikace přejde do režimu volné hry
5. uživatel dohraje scénář
6. systém zaznamená statistiku a spustí další scénář ve frontě
7. krok 3–5 se opakuje, dokud fronta scénářů není prázdná nebo uživatel neukončí hru

UC10 – Přihlášení dětského uživatele

konečný stav – uživatel je přihlášen do aplikace

způsob vyvolání – uživatel spustí aplikaci

poznámka – pro přihlášení musí být použit vhodný způsob, který zvládne uživatel sám, bude schopný ho jednoznačně identifikovat a zamezí hraní uživatele za někoho jiného

scénář

1. systém zobrazí uživateli přihlašovací obrazovku
2. uživatel se přihlásí

Dodatek C

Uživatelská příručka

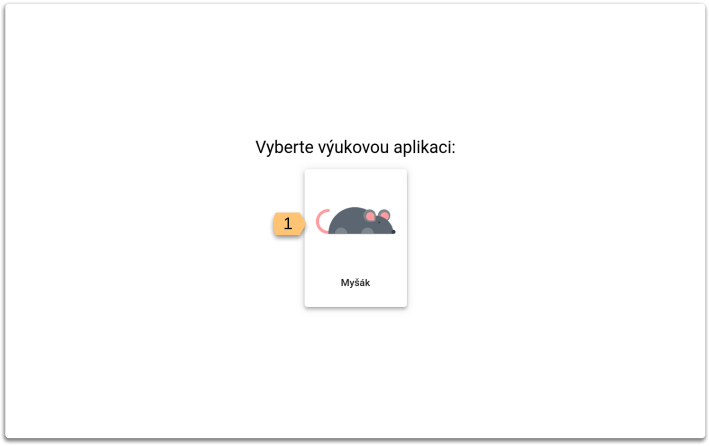
Aplikace *Myšák* umožňuje vytvářet nové scénáře v podporovaných výukových aplikacích, přiřazovat je dětem a spravovat fronty scénářů dětí.

C.1 Minimální požadavky

Pro vstup do aplikace je nutné použít webový prohlížeč Chrome (verze 60+) nebo Firefox (verze 52+).

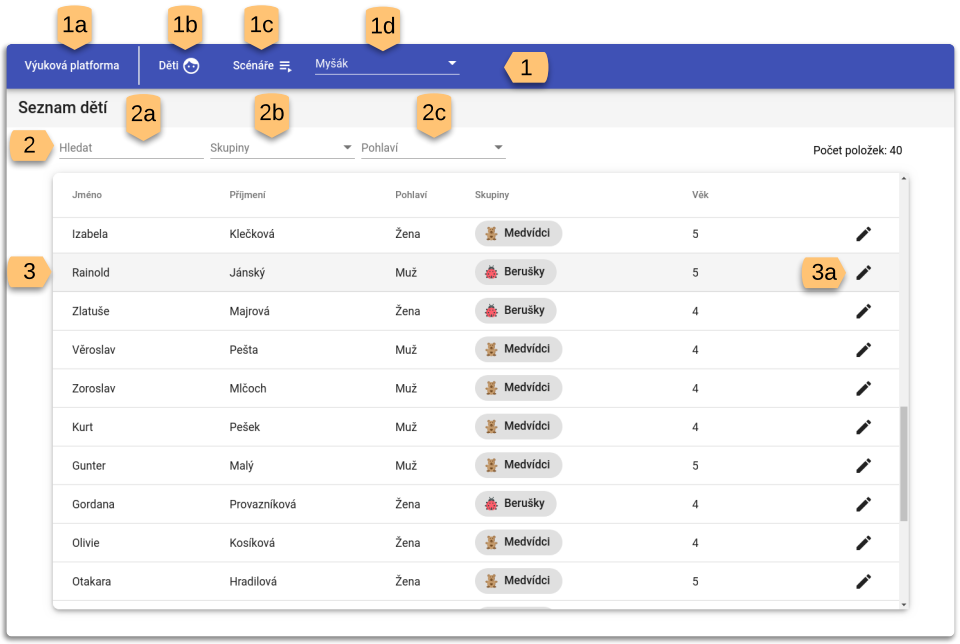
C.2 Výběr výukové aplikace

Při vstupu do aplikace je nejprve nutné zvolit výukovou aplikaci ze seznamu podporovaných aplikací (viz obrázek C.1).



1 – Seznam výukových aplikací

Obrázek C.1: Seznam výukových aplikací



1 – Navigační lišta
1a – návrat na seznam aplikací
1b – přejít na seznam dětí
1c – přejít na seznam scénářů
1d – změnit výukovou aplikaci

2 – Filtrování dětí
2a – vyhledat dítě
2b – filtrovat děti podle skupiny
2c – filtrovat děti podle pohlaví

3 – Seznam dětí
3a – upravit frontu dítěte

Obrázek C.2: Seznam dětí

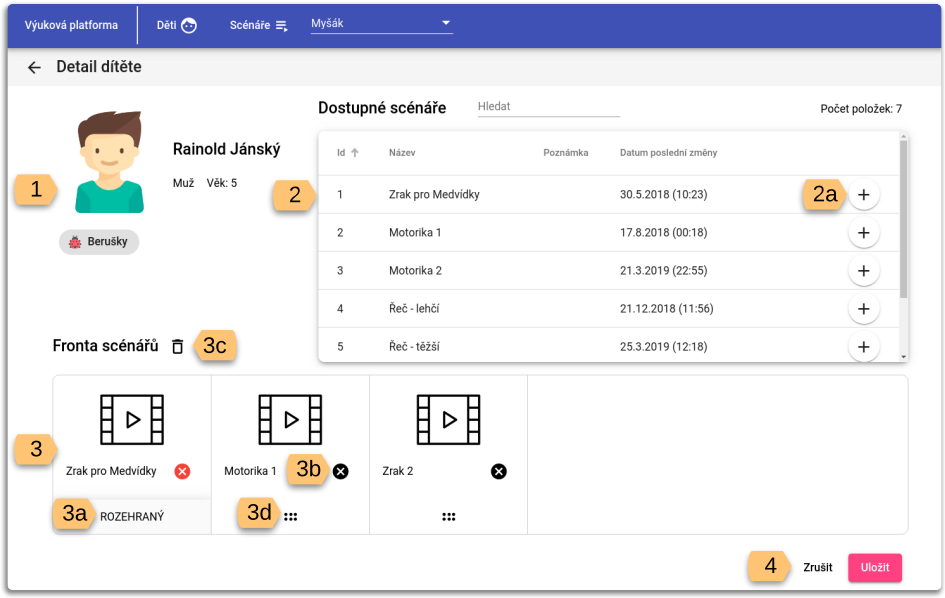
C.3 Seznam dětí

Po zvolení výukové aplikace se zobrazí seznam dětí v mateřské škole (viz obrázek C.2). Mezi dětmi lze vyhledávat nebo je filtrovat podle skupiny a pohlaví. Seznam se dá také seřadit podle jména, příjmení nebo věku. Kliknutím na položku dítěte nebo na tlačítko *Upravit frontu scénářů* (*3a*) lze přejít do režimu editace fronty scénářů dítěte.

V horní části obrazovky se nachází navigační lišta, která umožňuje přepínat mezi seznamem dětí, scénářů, nebo zvolit jinou výukovou aplikaci.

C.4 Detail dítěte

Po kliknutí na úpravu fronty dítěte se zobrazí stránka na obrázku C.3. Zde se nachází přehled údajů dítěte, jeho fronta scénářů a seznam dostupných scé-



1 – Detail dítěte

2 – Dostupné scénáře
2a – přidat scénář do fronty

3 – Fronta scénářů
3a – rozehraný scénář
3b – odebrat scénář z fronty
3c – vyčistit celou frontu (kromě rozehraného scénáře)
3d – změnit pořadí scénářů

4 – Uložit/zrušit změny

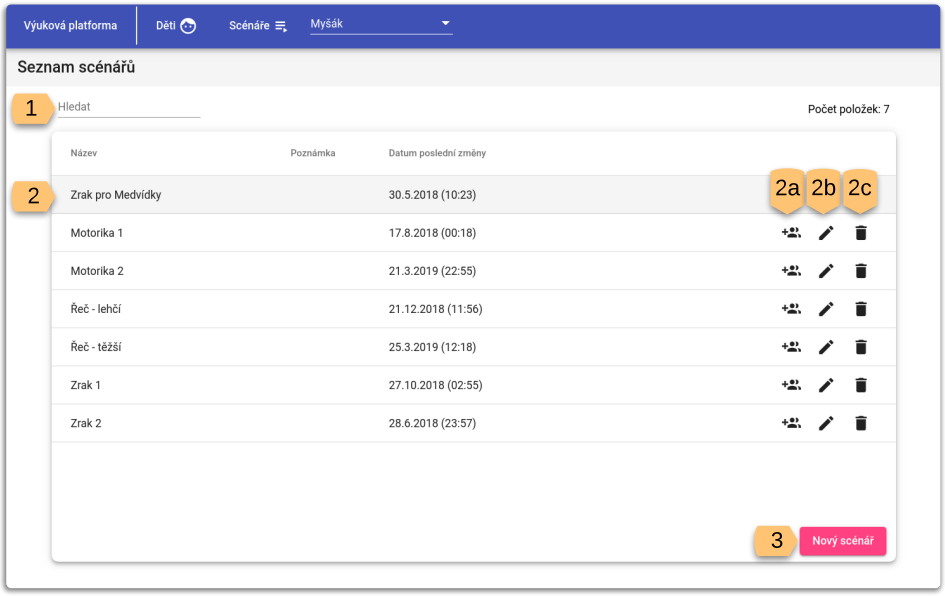
Obrázek C.3: Detail dítěte

nářů. Položky lze z fronty odebírat (*3b*), měnit jejich pořadí (*3d*) nebo přidávat ze seznamu dostupných sčénářů (*2b*). Pokud má dítě aktuálně rozehraný nějaký scénář, systém ho zvýrazní (*3a*). Rozehraný scénář lze zrušit, narozdíl od ostatních změn ve scénáři, které je nutné uložit (*4*), je tato operace nevratná.

C.5 Seznam scénářů

Na této stránce (viz obrázek C.4) jsou vidět všechny scénáře k dispozici pro vybranou výukovou aplikaci. Podobně jako v seznamu dětí lze mezi nimi vyhledávat, lze je seřadit podle jména nebo data poslední změny. Kliknutím na tlačítko *Nový scénář (3)* lze do systému přidat scénář nový.

U každého scénáře se nachází tři tlačítka (*2a*, *2b* a *2c*), které slouží k přiřazení scénáře dětem (jednomu i více), úpravě scénáře nebo jeho smazání.



1 – Vyhledávání scénářů **3 – Vytvořit nový scénář**

2 – Seznam scénářů
2a – přiřadit scénář dětem
2b – upravit scénář
2c – odstranit scénář

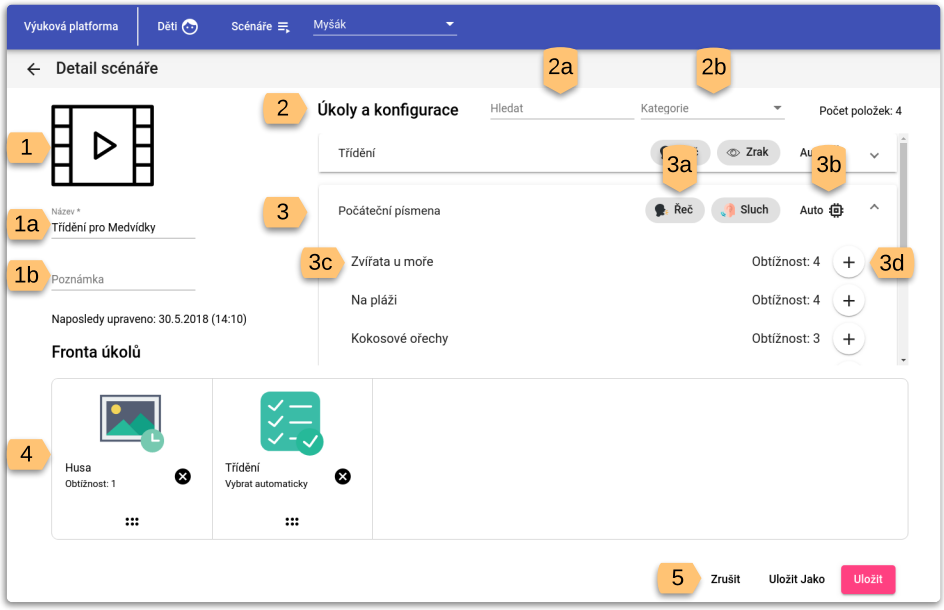
Obrázek C.4: Seznam scénářů

C.6 Úprava a tvorba scénáře

Stránka pro úpravu i tvorbu nového scénáře je identická s tím rozdílem, že nový scénář je vždy prázdný (viz obrázek C.5). Při vytváření scénáře je potřeba vyplnit název scénáře (*1a*) a volitelně poznámku (*1b*).

Na pravé straně je seznam úkolů a jejich konfigurací. Seznam lze prohledávat a úkoly lze filtrovat podle kategorií. Kliknutím na úkol (např. *3 – Počáteční písmena*) lze zobrazit seznam jeho konfigurací. Tlačítko pro přidání konfigurace (*3d*) vloží do fronty úkolů konkrétní konfiguraci. Volbu konfigurace lze také ponechat na systému a kliknutí na tlačítko pro automatický výběr (*3b*) přidá do scénáře pouze úkol. Systém pak při hraní vybere konkrétní konfiguraci automaticky. Fronta úkolů (scénář) v dolní části obrazovky funguje stejně jako na stránce detailu dítěte viz sekce C.4.

Na konci je potřeba scénář uložit. Na výběr jsou dvě možnosti, tlačítko *Uložit* přepíše aktuální scénář, při kliknutí na *Uložit jako* je uživatel vyzván k zadání nového názvu, pod kterým je uložena kopie aktuálně upravovaného scénáře. Tlačítko *Uložit jako* se nezobrazuje při vytváření nového scénáře.



1 – Detail scénáře
1a – název scénáře
1b – poznámka

2 – Filtrování úkolů
2a – vyhledat úkol/konfiguraci
2b – fitrovat podle kategorie
2c – odstranit scénář

3 – Seznam úkolů
3a – kategorie úkolu
3b – automatický výběr konfigurace
3c – konfigurace úkolu
3d – přidat konfiguraci do frotny

4 – Fronta úkolů

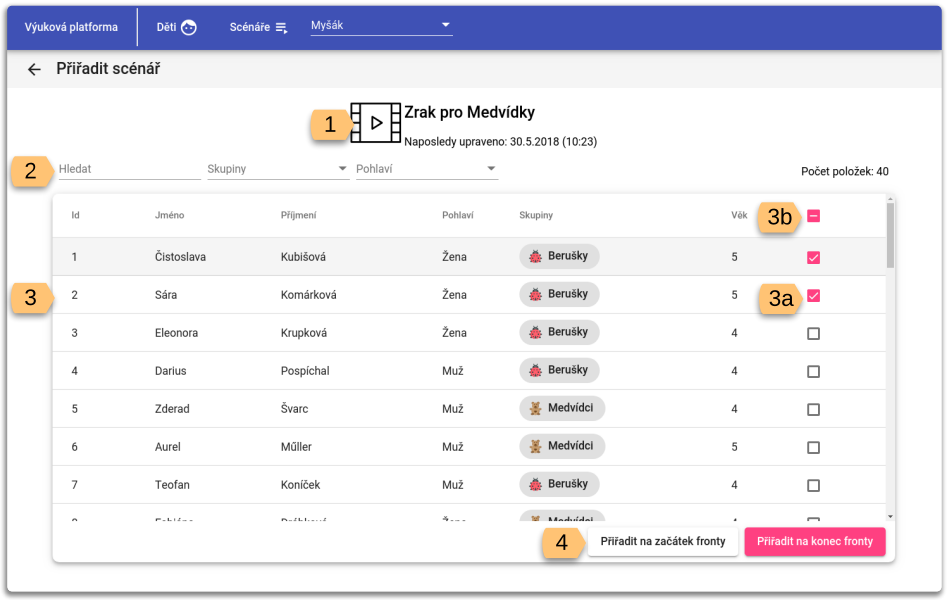
5 – Uložit/zrušit změny

Obrázek C.5: Úprava a tvorba scénáře

C.7 Přiřazení scénáře dětem

Hromadnému přiřazení scénáře dětem slouží právě tato stránka (viz obrázek C.6). Lze se na ni dostat kliknutím na tlačítko *Přiřadit dětem* u konkrétního scénáře na stránce se seznamem scénářů C.5.

Seznam dětí lze filtrovat podle skupiny nebo pohlaví. Po zvolení dětí zaškrtávacími políčky vpravo (*3a*) lze scénář přiřadit na začátek nebo konec jejich fronty scénářů (*4*). Pokud má dítě nějaký scénář rozehraný a uživatel klikne na *Přiřadit na začátek fronty*, přiřazovaný scénář se vloží hned za ten rozehraný. Seznam umožňuje také vybrat vše (*3b*). Lze tak například filtrovat děti podle skupiny a pak přiřadit scénář celé skupině.



1 – Přiřazovaný scénář

2 – Vyhledávání a filtrování

3 – Seznam dětí
3a – vybrat dítě
3b – vybrat vše

4 – Vložit na začátek/konec fronty

Obrázek C.6: Přiřazení scénáře dětem

Dodatek D

Vývojářská příručka

D.1 Stažení a instalace závislostí

Poté co vstoupíte do adresáře s projektem, je nutné stáhnout a nainstalovat veškeré potřebné závislosti pomocí správce balíčků `npm`. Pokud nemáte `npm` nainstalovaný, učiňte tak dle způsobu instalace softwaru na vašem operačním systému. Například pro Ubuntu lze npm jednoduše nainstalovat příkazem:

```
$ sudo apt install npm
```

Poté stačí nainstalovat veškeré závislosti příkazem:

```
$ npm install
```

D.2 Vývoj

Nyní je již vše připraveno pro vývoj. Níže je ještě uveden seznam užitečných příkazů:

`$ npm run ng serve`	spuštění vývojového serveru a aplikace
`$ npm run ng build --prod`	sestavení aplikace pro produkci
`$ npm run ng test`	spuštění jednotkových (unit) testů
`$ npm run ng e2e`	spuštění end-to-end testů
`$ npm run e2e-headless`	spuštění end-to-end testů jen v terminálu

Pro spuštění end-to-end testů je vyžadován nainstalovaný webový prohlížeč `Chromium`. Jeho instalace se opět liší podle operačního systému, v Ubuntu lze nainstalovat příkazem:

```
$ sudo apt install chromium-browser
```

Pokud nechcete před každý Angular příkaz (`ng`) psát `npm run`, můžete globálně nainstalovat nástroj `@angular/cli`. Nainstalujete ho příkazem:

```
$ npm install -g @angular/cli
```

Přepínač `-g` označuje globální instalaci. Dle umístění globálních `npm` balíčků na vašem systému je případně nutné vykonat tento příkaz s rootovským oprávněním (`sudo`). Příkaz `ng`, který tento balíček poskytuje bude nyní k dispozici napříč celým systémem.

Další příkazy a postupy práce s Angularem se nachází v oficiální dokumentaci Angularu (`https://angular.io`).

Dodatek E

Scénář uživatelského testování

E.1 Vymezení pojmů

Tyto pojmy se budou dále vyskytovat v testovacím scénáři, proto následuje jejich krátké vysvětlení.

Pro účely uživatelského testování byly zvoleny dvě role:

tester – uživatel pečlivě vybraný z cílové skupiny, který je primárním zdrojem informací o použitelnosti systému,

moderátor – řídí komunikaci s testerem, od jeho přivítání, uvedení do problematiky po zadávání instrukcí a úkolů, které má v systému vykonat.

E.2 Scénář moderátora

Moderátor zastupuje i roli asistenta, tedy zajišťuje celý průběh testování včetně přípravy místností, testovacích zařízení, kamer a doprovodného programu.

E.3 Uvedení do problematiky

Před samotným zahájením testování je nutné testerům vysvětlit smysl a účel aplikace. Proto moderátor před testováním všem zúčastněným promítne krátkou prezentaci a objasní veškeré pojmy, souvislost s mobilní aplikací a odpoví na všechny jejich případné dotazy. V prezentaci se však nesmí nacházet žádné ukázky testovaného webového rozhraní, a to ani návrhy obrazovek.

E.4 Doprovodné aktivity

Poté, co jsou všichni zúčastnění obeznámeni s problematikou, přichází na řadu samotné testování. Moderátor postupně po jednom odvede testery zvlášť do

testovací místnosti. Ostatním zúčastněným dá k dispozici tablet s mobilní aplikací *Myšák*, aby si mohli vyzkoušet dětskou část aplikace, zatím co čekají na test. I první tester dostane čas na seznámení s dětskou aplikací.

E.5 Hlavní část

Před testem moderátor naváže krátkou neřízenou konverzaci s testerem pro navození přátelské atmosféry a ujistí ho, že testovaným subjektem není on, ale webová aplikace. Také vyzve testera, aby přemýšlel nahlas a ptal se v případě nejasností. Poté nastaví na počítači webový prohlížeč na úvodní stránku aplikace.

E.5.1 První úkol – vytvoření scénáře

Moderátor vysvětlí testerovi jeho situaci a první úkol:

Děti se brzy vrátí z procházky a nastane čas na vzdělávací hraní. Každé dítě obdrží svůj tablet, na kterém je spuštěna výuková aplikace Myšák. Děti jsou rozděleny do tvou tříd: Medvídci a Berušky. Každá třída procvičuje jinou dovednost, Berušky procvičují řeč a Medvídci myšlení. Pro řeč již vaše kolegyně vytvořila scénář pojmenovaný Řeč – lehčí*, pro myšlení se zatím v systému nenachází žádný vhodný scénář, vaším prvním úkolem je tedy takový scénář vytvořit. Měl by obsahovat 3–5 úkolů zaměřených na myšlení.* V tuto chvíli předloží moderátor zúčastněnému kartičku popisující jeho první úkol. Kartička obsahuje požadovaný název a zaměření nového scénáře a počet úkolů.

E.5.2 Druhý úkol – přiřazení scénářů

Poté, co tester vytvoří v systému nový scénář dle instrukcí moderátor přejde k dalšímu úkolu:

Právě jste vytvořil/a nový scénář pro Medvídky, vaším úkolem je nyní přiřadit vytvořený scénář všem Medvídkům, následně přiřaďte kolegyní vytvořený scénář Řeč – lehčí *všem Beruškám. Vaším cílem je, aby děti mohli ihned začít hrát tyto scénáře.* Moderátor opět předloží kartičku se stručným zadáním úkolu.

E.5.3 Třetí úkol – vyčištění fronty dítěte

Další úkol je zaměřený na detail dítěte a jeho frontu scénářů:

Aničku (jedno z dětí, které je součástí testovacích dat) *přišel vyzvednout rodič. Proto je potřeba vyčistit její frontu scénářů, aby se mohla další den snadno zapojit do navazujícího hraní.* Kartička se zadáním úkolu obsahuje mimo jiné jméno dítěte, kterému je potřeba vyčistit frontu.

E.5.4 Čtvrtý úkol – úprava fronty scénáře

Tento úkol je opět zaměřený na frontu scénářů, tentokrát na úpravu pořadí:

Zjistil/a jste, že jedno z dětí má scénáře přeházené. Přesuňte jeho poslední scénář na první pozici ve frontě. Kartička opět obsahuje úkol a jméno dítěte.

E.5.5 Pátý úkol – smazání scénáře

Scénář na myšlení již nebudete potřebovat, je potřeba ho tedy smazat. Tester obdrží poslední kartičku s úkolem smazat stejný scénář, který předtím vytvořil.

E.5.6 Zakončení a dotazník

Moderátor položí dodatečné otázky ohledně uživatelského rozhraní, postupu testera a jeho pocitů. Na závěr mu předloží dotazník zaměřený na uživatelský komfort při provádění předchozích úkolů. U každého úkolu je škála, na které tester označí, jak moc snadné pro něj bylo daný úkol splnit.

E.6 Očekávaný průchod

Vzorové řešení úkolů, podle kterého bylo testování vyhodnocováno.

E.6.1 První úkol

Uživatel přejde na stránku scénářů, zvolí *Nový scénář* a vloží do fronty scénáře 3–5 úkolů z kategorie *Myšlení*. Volitelně může filtrovat úkoly podle kategorie. Nezáleží na tom, jestli vloží úkol (ponechá volbu konfigurace na serveru) nebo konfiguraci, důležitý je pouze jejich počet a kategorie. Následně klikne na uložit a potvrdí akci.

E.6.2 Druhý úkol

Uživatel přejde zpět na stránku scénářů a klikne na ikonu *Přiřadit dětem* u scénáře, který právě vytvořil. Následně pomocí filtrů vybere všechny děti ze třídy *Medvídci*, zaškrtne políčko pro *Označit vše* a klikne *Přiřadit na začátek fronty* (protože cílem je, aby děti mohli hrát scénář ihned).

E.6.3 Třetí úkol

Uživatel přejde na stránku se seznamem dětí, vyhledá Aničku (může použít filtrování), klikne na *Upravit frontu scénářů* nebo na celou položku tabulky a postupně smaže všechny položky fronty. Pokud má Anička nějaký rozehraný scénář, potvrdí jeho zrušení. Následně klikne na uložit a potvrdí akci.

E.6.4 Čtvrtý úkol

Uživatel postupuje stejně jako při třetím úkolu, jen místo smazání položek scénáře přesune poslední položku ve frontě na první místo.

E.6.5 Pátý úkol

Uživatel přejde na stránku scénářů, klikne na ikonu *Odstranit* u scénáře, který předtím vytvořil a potvrdí akci.

Dodatek F

Obsah přiloženého paměťového média

- `readme.txt` ..... stručný popis obsahu paměťového média
- `uzivatelska-prirucka.pdf` ..... Uživatelská příručka ve formátu PDF
- `vyvojarska-prirucka.pdf` ..... Vývojářská příručka ve formátu PDF
- `src`
  - `mysak-fe` ..... zdrojové kódy a konfigurační soubory webové aplikace
    - `src` ..... zdrojové kódy implementace
    - `e2e` ..... zdrojové kódy end-to-end testů
  - `mysak-api-test` ..... testy backendového REST API
    - `mock-data` ..... sada testovacích dat pro backend
    - `postman` ..... testovací kolekce a prostředí nástroje Postman
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS