Webový informační systém pro sběr a vyhodnocování dat z ABA terapií

Bc. Peter Mada

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Peter Mada**
Osobní číslo:	**A20939**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Webový informační systém pro sběr a vyhodnocování dat z ABA terapií**
Téma práce anglicky:	**Web-based Information System for Data Collection and Evaluation from ABA Therapies**

Zásady pro vypracování

1. Seznamte se s problematikou aplikované behaviorální analýzy (ABA) a s povahou měřených dat, které poskytuje.
2. Prostudujte vhodné webové technologie pro tvorbu informačního systému pro ukládání a zpracování výše uvedených dat.
3. V součinnosti s terapeuty poskytujícími ABA terapie sesbírejte funkční a nefunkční požadavky na systém.
4. Na základě požadavků vypracujte návrh webového informačního systému.
5. Pomocí vybraných technologií informační systém implementujte a otestujte na reálných datech z ABA terapií.
6. Věnujte také pozornost zabezpečení dat a správě osobních údajů.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ROANE, Henry, Joel RINGDAHL a Terry FALCOMATA, ed. Clinical and Organizational Applications of Applied Behavior Analysis. Cambridge (Massachuttes): Academic Press, 2015. ISBN 0124202497.
2. COOPER, John, Timothy HERON a William HEWARD. Applied Behavior Analysis. 3rd edition. London: Pearson, 2019. ISBN 0134752554.
3. BANKS, Alex a Eve PORCELLO. Learning React: functional web development with React and Redux. Sebastopol, CA: O'Reilly Media, 2017. ISBN 9781491954621.
4. GREBE, Sebastian. Hands-on Full-Stack Web Development with GraphQL and React. Birmingham: Packt Publishing, 2019. ISBN 9781789134520.
5. KEREKI, Federico. Mastering JavaScript Functional Programming: Write clean, robust, and maintainable web and server code using functional JavaScript. 2nd edition. Birmingham: Packt Publishing, 2020. ISBN 9781839213069.
6. MARTIN, Robert. Clean Architecture. New Jersey, USA: Prentice Hall, 2017. ISBN 0134494164.

Vedoucí diplomové práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2021**
Termín odevzdání diplomové práce: **23. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 19. 5. 2022

Peter Mada, v. r.
podpis studenta

ABSTRAKT

Cieľom tejto práce je vytvoriť systém, uľahčujúci prácu v priebehu terapeutických sedení využívajúcich princípu aplikovanej behaviorálnej analýzy. Na základe zozbieraných poznatkov bol vytvorený návrh webovej aplikácie. Za pomoci vhodných technológií bol implementovaný prototyp webovej aplikácie, ktorého funkčnosť bola následne otestovaná v praxi a vyhodnotená používateľmi.

Klíčová slova: webová aplikácia, jednostránková aplikácia, aplikovaná behaviorálna analýza, PERN, JavaScript, React

ABSTRACT

The aim of this thesis is to develop a system that facilitates work during therapy sessions using the principle of applied behavioural analysis. Based on the knowledge gathered, a web application design was created. A prototype of the web application was implemented with the help of appropriate technologies and its functionality was then tested in practice and evaluated by users.

Keywords: web application, single-page application, applied behavior analysis, PERN, JavaScript, React

Ďakujem vedúcemu diplomovej práce, Ing. Radku Valovi Ph.D. za odborné vedenie, cenné rady a pomoc pri vypracovaní diplomovej práce.

Prehlasujem, že odovzdaná verzia diplomovej práce a verzia elektronicky nahraná do IS/STAG sú totožné.

OBSAH

ÚVOD

Aplikovaná behaviorálna analýza je v Českej republike v podstate novinka. Preto i záznam a vyhodnocovanie dát prebieha štandardne metódou tužka-papier, zatiaľ čo v zahraničí už fungujú rôzne informačné systémy v podobe on-line alebo off-line aplikácií. Cudzojazyčná aplikácia však neodpovedá potrebám českých rodín, ktoré sa často do terapeutických sedení zapájajú. Z tohto dôvodu som sa rozhodol vytvoriť systém, ktorý by zefektívnil a uľahčil prácu terapeutom.

V teoretickej časti práce sa čitateľ oboznámi s stručným základom problematiky aplikovanej behaviorálnej analýzy, druhom dát ktoré sú počas terapií zaznamenávané a spôsobmi zaznamenávania týchto dát. Nasleduje stručný rozbor architektúry, popis základných komponent bežnej a oboznámenie s jednotlivými vrstvami webovej aplikácie. Následne sú v rámci teoretickej časti diplomovej práce popísané technológie, pomocou ktorých je možné vytvoriť komplexnú webovú aplikáciu. V poslednej časti je popísaný funkcionálny spôsob programovania a programovací jazyk JavaScript.

Praktická časť diplomovej práce vychádza z nadobudnutých teoretických znalostí aplikovanej behaviorálnej analýzy a naštudovaných vhodných technológií. V práci sú predstavené nevýhody zahraničných nástrojov pre zber dát a dôvody, prečo sa tieto nástroje nepoužívajú v Českej republike. Práca ďalej popisuje postup metodiky vývoja, analýzu požiadaviek, tvorbu návrhu a implementáciu danej aplikácie.

Výstupom práce by mala byť analýza, návrh a implementácia webového informačného systému, ktorý uľahčí prácu pri zbere a spracovaní dát z terapií. Výsledný systém by mal byť ľahko rozšíriteľný o nové dimenzie zbieraných dát.

I. TEORETICKÁ ČÁST

1 APLIKOVANÁ BEHAVIORÁLNA ANALÝZA

“Aplikovaná behaviorálna analýza je vedecká metóda zameraná na pochopenie zákonitostí ľudského chovania a ich vyžutie k zlepšeniu ľudského správania” [1]. Úlohou aplikovanej behaviorálnej analýzy, ako prírodnej vedy, je nájsť vzťah medzi environmentálnymi premennými a sociálne významným chovaním. Pomocou systematických intervencií sa behaviorálny analytik snaží využiť tento vzťah k úprave daného sociálne významného chovania. Výsledkom ABA však nie je len zmysluplné zlepšenie sociálne významného chovania, ale aj analýza všetkých faktorov ktoré zlepšenie tohto chovania spôsobili. Keďže ABA pracuje s vedeckými poznatkami je nutné aby vzťah medzi environmentálnymi premennými a sociálne významným chovaním mal všetky náležitosti vedeckého výskumu. Javy teda musia byť pozorovateľné, merateľné a výsledky opakovateľné. Musí sa teda jednať o objektívny popis tohto vzťahu, musí byť v zhode pozorovateľov. Čo znamená že musí byť možné tento vzťah kvantifikovať, a taktiež musí byť splnená podmienka overiteľnosti výsledkov. Je teda možné vykonať kontrolné experimenty, ktoré tento vzťah overia. Najčastejšie sa využíva intrasubjektívny design výskumu, pri ktorom sa porovnáva základná úroveň, takzvaný baseline, s úrovňou po absolvovaní intervencie. Zároveň je nutné aby sociálne významné chovanie bolo objektívne definované. Sociálne významným chovaním rozumieme také chovanie, ktoré zvyšuje habilitáciu. Keďže ABA je silne vedecky založená disciplína dochádza v ABA k neustálemu testovaniu, vyhodnocovaniu a zlepšovaniu používaných metód. Na tieto zmeny musia reagovať taktiež konkrétne intervencie, ktoré dané poznatky vo svojej praxi využívajú. Behaviorálny analytik musí dodržiavať pravidla dobrej praxe, vrátane využívania najnovších poznatkov. Časová efektivita je jedným z najdôležitejší kritérií pri voľbe metód a postupov. [1]

ABA pracuje so skutočne merateľnými hodnotami správania a nie len s diagnózou. Všetky intervencie, ktoré sa v ABA dejú, sú založené na skutočne reálne meraných dátach. Dá sa jednoznačne určiť efekt akú jednotlivé intervencie skutočne na dané chovanie majú. Je možné pomerne jednoducho určiť pokrok a taktiež či boli skutočne dosiahnuté stanovené ciele. Dôležité je taktiež porovnávať efektivitu s ohľadom na jednotlivé prípady. Podobne účinné metódy môžu byť viac či menej vhodné pre konkrétnych študentov. [1]

1.1 Základné princípy aplikovanej behaviorálnej analýzy

ABA je často používaná v rôznych odvetviach, ako napríklad v personalistike, školstve, reklame, športe a ďalších. Jej najčastejšie využitie je však v oblasti práci s osobami s poruchou autistického spektra (PAS). [1]

Základným princípom ABA je takzvaný ABC princíp, ktorý je založený na antecedente, chovaní a následku. Tento princíp nájdeme prakticky v každom druhu chovania a popisuje vzťahy medzi chovaním a prostredím, rovnako ako stimuly, ktoré chovanie ovplyvňujú. Chovanie stojí medzi antecedentami (environmentálnymi stimulmi, ktoré predchádzajú chovaniu) a konsekventami (environmentálnymi stimulmi či udalosťami, ktoré nasledujú po chovaní), ktoré na nich pôsobia a utvárajú repertoár jedinca v priebehu učenia. Nie len dlhodobo, ale i momentálne v aktuálnej situácii, v ktorej sa jedinec nachádza. Samotné chovanie je pritom výsledok pôsobenia antecedentov a konsekventov v priebehu učenia. Existuje tu pri tom určitá závislosť, kontingencia. [1]

V praxi ABA kladie dôraz na priame zapájanie učiteľov do učenia dieťaťa. Učiteľ aktívne usmerňuje a promptuje, dopomáha, dieťati v procese učenia. ABA pracuje na princípe pozitívneho a negatívneho posilnenia. Posilnením v tomto prípade rozumieme zvýšenie správania. Pozitívne posilnenie nastáva v prípade, že je dieťa za nejakú chovanie alebo reakciu odmenené, a tento proces tak vedie k väčšej pravdepodobnosti výskytu takéhoto chovania v budúcnosti. Toto chovanie teda uvidíme častejšie. Princíp sa využíva vo chvíli, keď chceme nejaké chovanie budovať alebo udržiavať v repertoáre. Spravidla tu dochádza k posilneniu spoločensky pozitívneho chovania a k ignorovaniu negatívneho chovania. Teda toho chovania, ktoré chceme v repertoáru odstrániť z dôvodu jeho spoločenskej nevhodnosti alebo škodlivosti. [2]

1.2 Merateľné dimenzie chovania

Jedným z hlavných znakov aplikovanej behaviorálnej analýzy je systematický zber, spracovanie a vyhodnocovanie dát o chovaní. Chovanie je komplikovaný proces, ktorý môže ovplyvňovať veľké množstvo faktorov. To znamená, že samotný zber dát o chovaní je ešte náročnejší proces, ktorý je navyše závislý na schopnostiach pozorovateľa. Je teda nutné riadne si určiť druh a štruktúru zbieraných dát. Na to, aby bolo možné z dát predikovať vývoj v chovaní, je nutné, aby zbierané dáta splňovali tri základné požiadavky. A to presnosť, platnosť a spoľahlivosť. Zbierané dáta sa pokladajú za presné, ak sledované behaviorálne

dimenzie sú korektne zachytené týmito dátami. Chyba merania je teda čo možno najnižšia. Validita dát určuje rozsah systému, v ktorom sa meranie vykonáva. Spoľahlivosť znamená, že pri použití nazbieraných dát na rovnaké správanie sa dopracujeme k rovnakému výsledku. [3]

Najčastejšie sa u ABA používa takzvané priame pozorovanie. Analytik u tohto typu pozorovanie priamo sleduje chovanie, ktoré aktuálne prebieha a bezprostredne ho zaznamenáva. Toto pozorovanie môžeme rozdeliť na nepretržité a prerušované meranie. U nepretržitého pozorovania sa zaznamenáva chovanie počas uceleného neprerušeného časového úseku. Tento typ merania je vhodný použiť, pokiaľ je nutné získať čo možno najpresnejší záznam o chovaní. Obrovskou nevýhodou tohto typu merania je jeho časová náročnosť. Do nepretržitého merania spadá meranie frekvencie, dĺžky, latencie a intenzity. Prerušované meranie správania poskytuje v podstate len vzorec daného správania. Výsledkom je teda nekompletný záznam. Priame pozorovanie nie je vhodne v prípade, že sa jedná o pozorovanie viacerých študentov alebo je meraných viac cieľov. V takýchto prípadoch je lepšie použiť nepriame hodnotenie, ktoré v takomto prípade zvýši efektivitu zbierania dát. U nepriameho hodnotenia analytik vychádza z už nazbieraných dát alebo permanentných produktov, ako sú napríklad výtvory študentov, testovacie listy a podobne. Je taktiež možné vychádzať z rozhovorov s rodičmi, učiteľmi alebo blízkymi osobami študenta. Nepriame hodnotenie nie je tak presné ako priame pozorovanie, avšak jeho časová náročnosť a zložitosť je omnoho menšia. [3]

1.2.1 Meranie frekvencie

Meranie frekvencie je najčastejšie používaný druh merania chovania používaný v aplikovanej behaviorálnej analýze. Frekvenciou v tomto prípade rozumieme počet opakovaní za nejakú štandardnú časovú jednotku. Táto časová jednotka musí byť vždy presne definovaná u každého merania aby nedochádzalo k nejasnostiam. Zaznamenáva sa teda počet opakovaní nejakého chovania, či už pozitívneho alebo negatívneho, za presne stanovené časové obdobie. Samozrejme má tento druh merania aj svoje nevýhody. Nie je napríklad vhodné merať frekvenciu v prípade, že merané chovanie môže nasať len za určitých obmedzených podmienok. Napríklad v prípade, že chovanie závisí na nejakom predchádzajúcom stimule. Napríklad keď učiteľ ukazuje farebné kartičky a študent z nich vyberá správnu farbu. V tomto prípade meranie frekvencie nie je vhodné z dôvodu závislosti frekvencie na samotnej prezentácii kartičiek. Je preto vhodné použiť iný spôsob merania. A to počet úspešných a

neúspešných pokusov z celkového počtu pokusov. Taktiež sa meranie frekvencie nehodí na zaznamenávanie správania, ktoré prebieha počas dlhšieho časového úseku. [4]

1.2.2 Meranie trvania

Druhý najčastejší druh merania sleduje dĺžku chovania vyjadrenú časom. Jedná sa napríklad o meranie času počas ktorého študent pracuje na zadanej úlohe, alebo sa prejavuje nejakým nevhodným chovaním. Meranie trvania sa používa najčastejšie u chovania, ktoré sa vyskytuje po dlhšiu dobu, viac než pár sekúnd, a trvá nepretržite bez prerušenia. Terapeut musí byť teda veľmi dobre inštruovaný, kedy presne chovanie začína a končí. Meranie je totižto veľmi citlivé na nepresnosti. Meranie sa uskutočňuje pomocou stopiek. [4]

1.2.3 Meranie intenzity

Meranie intenzity je komplikovanejší druh merania. Toto meranie v sebe, okrem iného, zahrnuje merania sily, intenzity, prípadne amplitúdy. Patrí medzi komplikovanejšie druhy merania z toho dôvodu, že sa pri tomto meraní nepožívajú žiadne pomôcky ale je plne závislé na pozorovaní a subjektívnom hodnotení a skúsenostiach každého pozorovateľ zvlášť. Čo sa jednému môže zdať ako silná reakcia môže iný pozorovateľ vyhodnotiť ako slabá reakcia. Objektívne definovať intenzitu je možné, avšak je to časovo veľmi náročné. Často sa meria napríklad sila úderu s akou silou dieťa hodilo predmet, sila vokálneho prejavu a iné. Meranie intenzity sa napríklad častou používa v prípade, že po študentovi chceme zníži jeho hladinu hluku. [3]

1.2.4 Meranie latencie

Latencia predstavuje dobu medzi antecedentom a začatím vykonávania požadovaného chovania. Tento druh merania je teda vhodné používať v prípade, že je chovanie podmienené inštrukciou, direktívou alebo stimulom z prostredia. Meranie latencie sa používa v prípade že nás zaujíma vzťah medzi antecedentom a určitým chovaním. Latencia je vhodná na merania v prípade, že chceme čas medzi chovaním skrátiť alebo naopak predlžiť. Toto meranie sa teda často vykonáva pomocou stopiek. Tento typ merania je obľúbený z dôvodu ľahkého overenia. Neexistuje tu žiadny subjektívny faktor, všetky zúčastnené strany si ľahko môžu overiť presnosť svojich meraní. Samotný čas príkazu a čas začatia určitého chovania je ľahko a jednoznačne určiteľný. Nevýhodou tohto druhu merania je jeho náročnosť v prípade viacerých pozorovaných jednotlivcov. [3]

1.2.5 Meranie intervalu

V tomto prípade je terapia rozdelená do úsekov rovnakej časovej dĺžky a meranie nastáva v každom úseku. Každý interval sa následne hodnotí buď ako pozitívny, to znamená že chovanie sa v danom intervale vyskytlo, alebo negatívny, chovanie sa v danom intervale nevyskytlo. A to podľa vopred zadaných kritérií. Tento druh merania má viacero variácií podľa toho, kedy v jednotlivom intervale požadované správanie skutočne nastalo. Najčastejšie sa využíva u chovania, ktoré sa vyskytuje v určitom čase alebo situáciách a nevídame ho dostatočne často na to, aby sme mohli použiť meranie frekvencie alebo trvania. [3]

1.3 Spôsoby zaznamenávania dát

Vhodnú metódu merania vyberáme podľa povahy dát, ktoré chceme zbierať a následne zvolíme spôsob ich zaznamenávania. V súčasnosti sa stále dá považovať za najčastejšiu metódu zaznamenávania dát u ABA metóda tužka-papier. Táto metóda je finančne veľmi výhodná, avšak takto nazbierané dáta je nutné po skončení terapie často prepisovať a vytvárať grafické vizuálne podklady. Čo je práca terapeuta, ktorú by mohol skôr stráviť prácou so študentom. Čoraz populárnejšie, vzhľadom k svojej dostupnosti, sa stáva elektronický zber dát pomocou rôznych aplikácií. Terapeut, supervízor alebo rodič môže okamžite pristupovať k zapísaným dátam. Výhodou takéhoto zberu je rýchlosť zápisu dát. Dáta sa zaznamenávajú do dopredu pripravených formulárov, prípadne grafov. Vzhľadom k tomu, že aplikovaná behaviorálna analýza je v Českej republike mladým odborom, česká jazyková verzia zahraničných aplikácií zatiaľ neexistuje. Chýbajúca česká lokalizácia je pre množstvo rodičov a terapeutov, ktorý s študentami pracujú, obrovská komplikácia. Nastáva totižto horšie porozumenie obsahu, znižuje sa pružnosť použitia a zároveň nie je možno jednoducho prenášať dáta medzi rôznymi odborníkmi, ktorý často pracujú v rámci školstva alebo zdravotníctva, a kde je materský jazyk nutná podmienka. [5]

2 LEGISLATÍVA

„Povolanie behaviorálneho analytika upravuje v Českej republike zákon číslo 201/2017 Zb. zo dňa 8. Júna 2017, ktorý mení zákon č. 96/2004 Zb., o podmienkach získavania a uznávania spôsobilosti k výkonu nelekárskych zdravotníckych povolaní a k výkonu činností súvisiacich s poskytovaním zdravotnej starostlivosti a o zmene niektorých súvisiacich zákonov (zákon o nelekárskych zdravotníckych povolaniach), v znení neskorších predpisov, a zákon č. 95/2004 Zb. o podmienkach získavania odbornej spôsobilosti a špecializovanej spôsobilosti k výkonu zdravotníckeho povolania lekára, zubného lekára a farmaceuta, v znení neskorších predpisov bol vyhlásený 12.7.2017 zbierkou zákonov v časti 72. Platnosť nadobudol dnom 1.9.2017 (Behaviorálny analytik – §21c, Asistent behaviorálneho analytika – §29, Behaviorálny technik – §29a).“ [6]

Pracovníci poskytujúci behaviorálnu analýzu, v rámci českého zákona o nelekárskych zdravotníckych pracovníkoch v zdravotníckom zariadení, majú povinnosti dané zákonom. A vzťahujú sa teda na nich všetky povinnosti vyplývajúce z týchto zákonov. Predovšetkým o zbieraní dát a ich bezpečnom uložení, vrátane zákona č. 110/2019 Zb. [7]

Behaviorálny analýzu je možné vykonávať i ako medzinárodne certifikovaný behaviorálny analytik s titulom BCBA. Pokiaľ sa nejedná o poskytovanie služieb v rámci zdravotníckeho zariadenia, napríklad v prípade keď analytik dochádza priamo do rodín v rámci domácich programov, vzťahujú sa na jeho prácu pravidlá vyplývajúce zo európskej smernice GDPR, teda zákon č. 110/2019 Zb. – Zákon o zapracovaní osobných údajov. [8]

3 ARCHITEKTÚRA WEBOVÝCH APLIKÁCIÍ

V súčasnosti existuje veľké množstvo rôznych druhov architektúr webových aplikácií. Architektúra webovej aplikácie ovplyvňuje logiku aplikácie, ako interaguje s ostatnými komponentami, funkcionalitu a v neposlednom rade vhodnosť použitia. Jednotlivé architektúry je často možné kombinovať a hranice medzi týmito architektúrami tak nie sú vždy presne určené. Na architektúru webových aplikácií sa môžeme pozerať z pohľadu frontendu, kedy sa jedná napríklad o tradičné webové aplikácie (Multi Page Application) prípadne jednostránkové aplikácie (Single Page Application), alebo z pohľadu backendu. Vtedy sa môže jednať napríklad o mikroslužby (Microservice Architecture) alebo architektúra bez serveru (Serverless Architecture). [9]

3.1 Tradičné aplikácie

Tradičný druh webových aplikácií, alebo tiež viacstránkové aplikácie (MPA) predstavujú klasickú architektúru tvorby webových aplikácií. Táto architektúra zahŕňa aplikácie u ktorých server v odpovedi na požiadavku klienta generuje a zasiela celú HTML stránku. V MPA sa prezenčná vrstva nachádza na strane serveru. Server je teda zodpovedný za dynamickú tvorbu HTML. Server je taktiež zodpovedný za samotný prístup k zdrojom. To znamená že server musí mať dostatočnú výpočetnú kapacitu, aby dokázal spracovať všetky požiadavky od klientov. Vytváraním šablón a ich plnením sa kód stáva neprehladenejším, menej flexibilnejším a ťažko znovu použiteľným. Takáto forma prezentácie dát je veľmi komplikovaná na údržbu a prípadné zmeny. Prehliadač používateľa je v tomto type architektúry už len zodpovedný za vykreslenie HTML prijatého od servera. [10]

MPA sa v súčasnosti asi najviac používajú pre veľké komerčné projekty a rôzne elektronické obchody. Sú vhodné na použitie tam kde je nutné zobrazovať veľké množstvo informácií pre používateľov. Výhodou MPA je totižto veľmi dobrá SEO optimalizácia. Na rozdiel od SPA ktoré sú silne závislé na JavaScripte, ktorý však často nie je vyhľadávačmi podporovaný. Ďalšou výhodou je teda nezávislosť MPA na JavaScripte. Webová aplikácia bude väčšinou plnohodnotne použiteľná aj v prípade, že má používateľ vo svojom prehliadači zakázané použitie JavaScriptu. [11]

Nevýhodu tradičnej architektúry je veľké množstvo dát ktoré je prenášané medzi klientom a serverom po sieti. Väčšina interakcií užívateľa s webovou aplikáciou má za následok vygenerovanie nového požiadavku na server a následnú odpoveď servera vo forme celej

HTML stránky. Veľké množstvo dát je teda generovaných a prenášaných po sieti duplicitne. Napríklad hlavička a pätička webovej aplikácie je zbytočne prenášaná po sieti pri každom požiadavku. [10]

Ďalšou nevýhodou je komplikovanosť vývoja. V MPA totižto prebieha mix štruktúry a obsahu na strane serveru, čo znamená že za tvorbu HTML štruktúry je zodpovedný backend vývojár a nie frontend vývojár. Backend vývojár teda musí mať povedomie o frontend technológiách, štandardoch a osvedčených postupoch. [10]

Nevýhodou tejto architektúry je pomalá doba načítania, keďže každá stránka musí byť znova načítaná. Pre užívateľov tak celá webová aplikácia väčšinou pôsobí ťažkopádne a pomaly. Hlavne v prípade pomalšieho internetového pripojenia. Toto výrazne zhoršuje užívateľský zážitok. Ďalšou nevýhodou je doba vývoja. MPA sú vo všeobecnosti náročnejšie na vývoj než napríklad SPA. [11]

3.2 Jednostránkové aplikácie

Jednostránkové aplikácie (SPA) sa po prvý krát objavili na začiatku 21. storočia spolu s nástupom technológie AJAX. Pomocou AJAX požiadaviek, JavaScriptu a kaskádových štýlov (CSS) začalo byť možné manipulovať obsahom webovej stránky bez nutnosti presmerovania na novú stránku. Takéto webové aplikácie obyčajne poskytujú používateľom lepší užívateľský zážitok (UX) než klasické webové aplikácie. [10]

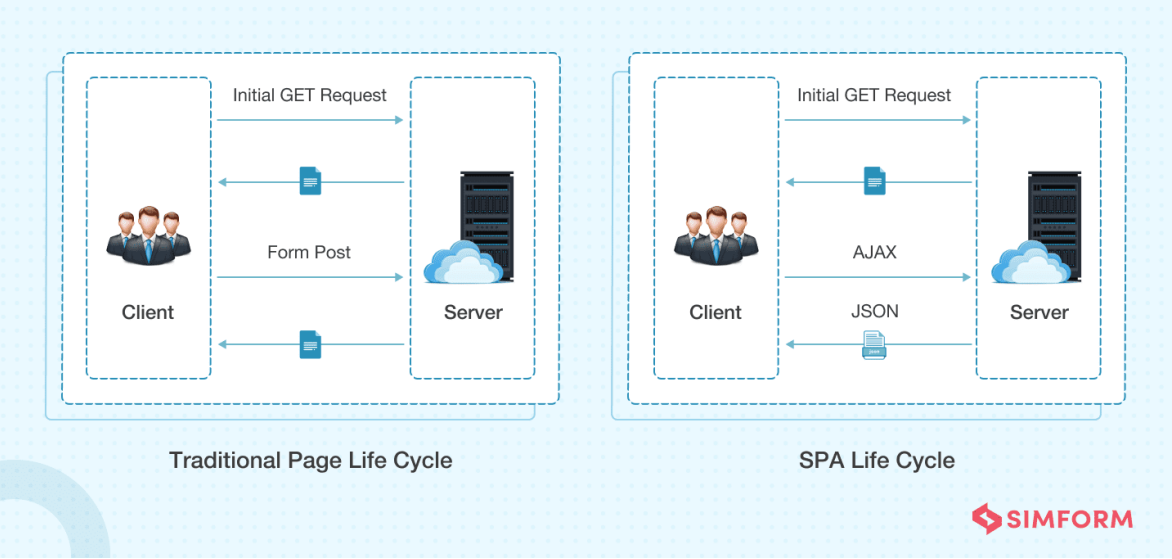
Architektúra SPA sa vyznačuje presunom prezentačnej vrstvy zo strany serveru na stranu klienta. Server už tak nie je zodpovedný za prezentáciu dát. U SPA má zobrazenie dát prijatých zo serveru na starosti klient. Dochádza tu teda k presunu procesu vytvárania zobrazenia do užívateľského prostredia (UI). Server u SPA len poskytuje dáta klientovi, ktorý ich sám následne spracováva a zobrazuje. Dáta sa do HTML teda vkladajú až na strane klienta, teda v rámci frontendu. [12]

SPA webové aplikácie sa vyznačujú absenciou obnovovania stránky webového prehliadača pri interakcii používateľa, nedochádza tu k presmerovaniu na novú stránku. K načítaniu klasického HTML zo servera dochádza len pri prvotnom načítaní. Tento HTML súbor však väčšinou obsahuje len akúsi základnú štruktúru. Obsahuje napríklad tagy *head* a *body* ako v klasickom HTML. *Body* tag však neobsahuje celú stránku, ale väčšinou len jeden *div* tag. Tento *div* následne slúži ako vstupný bod na ktorý sa napájajú ďalšie súčasti aplikácie. Po tom, čo si klient stiahne počiatočný HTML sú stiahnuté všetky potrebné nástroje na

vytváranie a správu ďalších zobrazení. Následne sú na vstupný div napojené všetky potrebné komponenty, a tak je vygenerovaná prvá vstupná stránka. V prípade prechodu na podstránku nedochádza k obnove stránky, ale frontend si vyžiada dáta od serveru. Po obdŕžaní dát, frontend tieto dáta zapracuje do DOM (Document Object Model) napríklad pomocou JavaScriptu. Spracovanie týchto dát má obyčajne na starosti nejaký framework, ako napríklad React, Angular, alebo Vue. Bežný používateľ si počas navigácie v rámci webovej aplikácie ani nevšimne, že sa jedná o SPA. Webová stránka sa chová ako u klasických viacstránkových aplikácií. Napríklad URL adresa v prehliadači sa mení ako u klasickej architektúre. Taktiež aj tlačidlo naspať v prehliadači funguje ako pri klasickej webovej aplikácii. [12]

Výhodou SPA je lepší užívateľský zážitok (UX). Tieto aplikácie totižto pôsobia viac ako natívne aplikácie. SPA obyčajne reagujú rýchlejšie na používateľské požiadavky než MPA, a okrem prvotného načítania sú teda považované za rýchlejšie. U SPA je zároveň striktne oddelený frontend od backend. Ich vývoj je tým pádom jednoduchší. [11]

Nevýhodou je ich zlá SEO optimalizácia. Vyhľadávače majú vo všeobecnosti problém s indexáciou a prehľadávaním obsahu SPA. Tieto aplikácie sú taktiež silne závislé na JavaScripte. Používateľ musí mať povolený JavaScript vo svojom prehliadači aby mohol plnohodnotne zobrazovať a používať SPA. [11]



Obrázok 1. Životný cyklus tradičnej aplikácie a SPA [9]

3.3 Natívne aplikácie

Tieto aplikácie sú vyvíjané pre konkrétnu mobilnú platformu. Jedná sa hlavne o operačné systémy Android a iOS. Takto vyvinuté aplikácie môžu využívať všetky funkcie zariadenia, ako napríklad fotoaparát, zoznam kontaktov alebo čítačku otlačkov prstov. Natívne aplikácie je nutné predtým, ako ich je možné použiť stiahnuť a nainštalovať. Jazykom natívnych mobilných aplikácií je napríklad Java, Kotlin alebo Swift. Tieto aplikácie už nepracujú so štandardnými technológiami webových aplikácií, ako je HTML, CSS a JavaScript. [13]

Natívne aplikácie sú oproti klasickým webovým aplikáciám vo všeobecnosti považované za rýchlejšie a s lepším používateľským prostredím (UI), a tým pádom aj lepším používateľským zážitkom (UX). Tieto aplikácie dokážu pracovať s hardvérovými zdrojmi zariadenia. To znamená, že natívna aplikácia má prístup napríklad k GPS alebo čítačke otlačku prstov. Ďalšou výhodou týchto aplikácií je fakt, že dokážu pracovať na pozadí a majú možnosť zobrazovať notifikácie používateľovi. [13]

Nevýhodou je pomerne náročný vývoj, rozširovanie a udržiavanie týchto aplikácií. Sú totižto vyvíjané na konkrétny operačný systém, a nie sú prenositeľné na inú platformu. Je vždy nutné vytvárať danú aplikáciu nanovo na daný operačný systém. [13]

3.4 Progresívne webové aplikácie

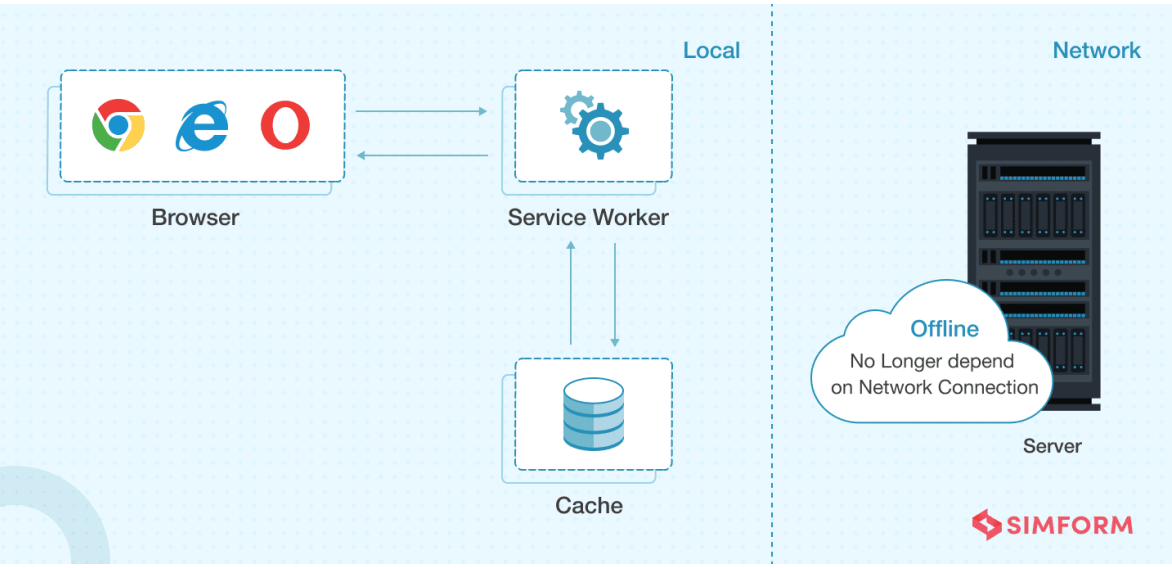
Progresívne webové aplikácie (PWA) predstavujú akýsi medzi krok medzi natívnymi aplikáciami a webovými aplikáciami. PWA teda predstavujú mobilnú aplikáciu, ktorá na rozdiel od natívnych aplikácií je spúšťaná v prehliadači. Nie je teda nutné túto aplikáciu inštalovať, aj keď väčšinou je pre jej plnohodnotné používanie inštalácia nutná. Behové prostredie prehliadača znamená, že tieto aplikácie je možné vyvíjať pomocou bežných technológií webových aplikácií, a to HTML, CSS a JavaScript. [14]

Oproti natívnym aplikáciám je vývoj týchto aplikácií omnoho jednoduchší. Väčšinu už existujúcich webových aplikácií je taktiež možné prekonvertovať na PWA bez väčších problémov. PWA taktiež môžu využívať takmer všetky funkcie, ktoré majú dostupné natívne aplikácie. U PWA je teda možné napríklad spracovávať úlohy na pozadí, zobrazovať notifikácie požívateľom aj keď momentálne PWA nebeží, alebo tiež pridávať ikonu PWA na domácu obrazovku mobilného telefónu tak ako bežnú natívnu aplikáciu. [14]

Výhodou týchto aplikácií je ich relatívne dobrý užívateľský zážitok (UX), vo všeobecnosti tento užívateľský zážitok nie je väčšinou taký dobrý ako pri natívnych aplikáciách ale je

o obyčajne o stupeň lepší než u bežných webových aplikáciách. PWA môžu na rozdiel od bežných webových aplikácií pracovať aj v režime off-line. [14]

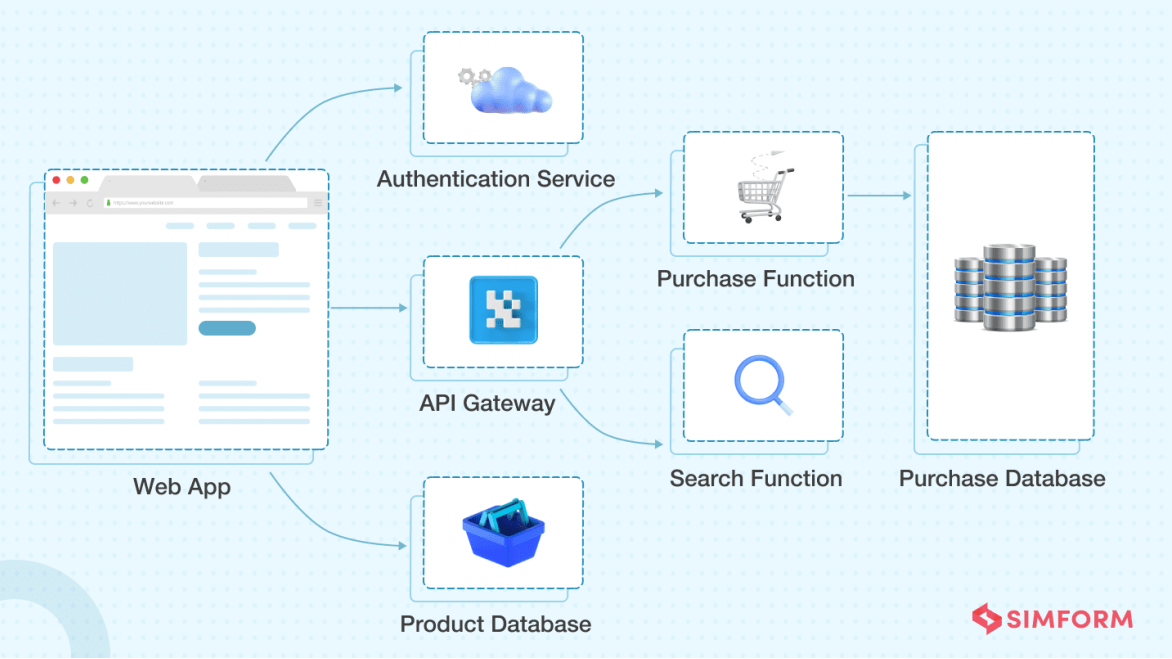
PWA môže pristupovať k veľkému množstvu hardvérových funkcií, tento zoznam funkcií nie je však taký rozsiahly ako u natívnych aplikácií. PWA sú totižto silne viazané na HTML5 štandard. To znamená že pokiaľ HTML5 nepodporuje prístup k čítačke otlačkov prstov tak ani PWA aplikácia s daným zariadením nemôže pracovať. [14]



Obrázok 2. Architektúra PWA [9]

3.5 Architektúra bez serveru

Architektúra bez serveru (Serverless Architecture) je druh architektúry v ktorej sa zodpovednosť za správu a ukladanie dát prenáša na tretiu stranu. To umožňuje vývojárom sa viac sústrediť na samotný vývoj služieb, keďže už nie je nutné riešiť správu uložených dát. V tomto type architektúry, aj napriek názvu, existuje server, len ho má na starostlivosti tretia strana. Výhodou tejto architektúry je zníženie množstva kódu ktorý je nutný na fungovanie aplikácie a tým pádom aj zníženie nákladov. O databázový server a správu dát sa už totižto nie je nutné starať. V tomto type architektúry je zvýšená bezpečnosť dát, tretia strana ktorá sa zaoberá len správou dát, sa môže viac sústrediť na ich bezpečnosť. Nevýhodou týchto aplikácií býva nižšia kontrola nad tým, čo sa s dátami deje a taktiež len obmedzená možnosť upravenia samotnej infraštruktúry. [15]

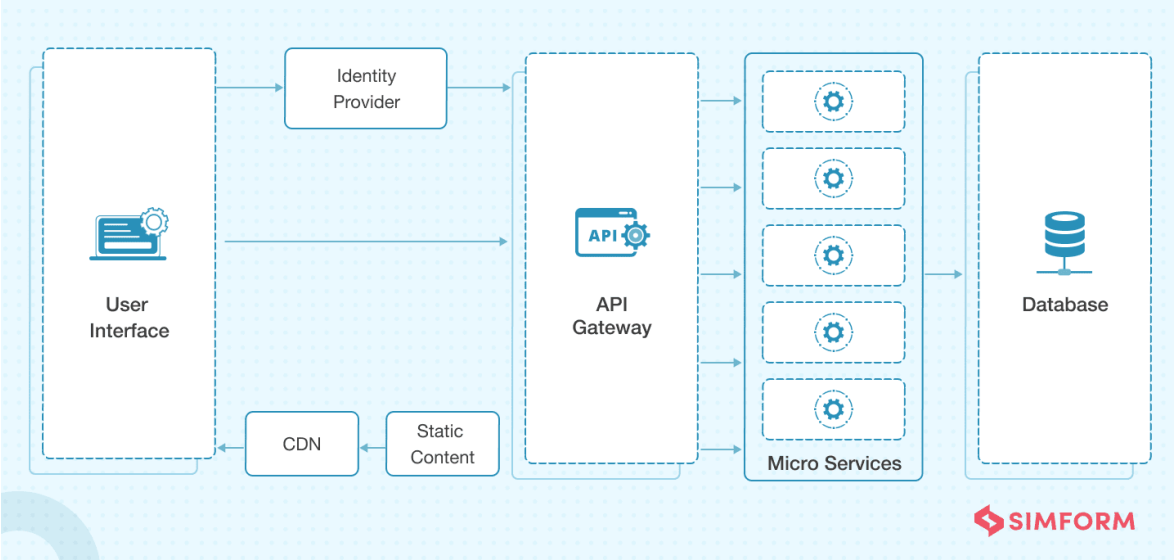


Obrázok 3. Štruktúra aplikácie využívajúci architektúru bez serveru [9]

3.6 Mikroslužby

Mikroslužby (Microservices) už patrí medzi architektúry backendu, konkrétne teda business vrstvy. Mikroslužby sú ideálne na riešenie problému veľkých monolitických aplikácií. U monolitických aplikácií je problém s ich značnou komplexnosťou a tým pádom ťažkopádnym návrhom, vývojom a udržiavaním. U mikroslužieb je každá služba vyvíjaná nezávisle na ostatných. Každá z týchto mikroslužieb má nejaký špecifický účel. V prípade že je potreba, aby viac mikroslužieb spolu spolupracovalo, komunikujú tieto služby skrz API. Skupina takto komunikujúcich mikroslužieb spolupracuje na dosiahnutí určitého cieľa. [16]

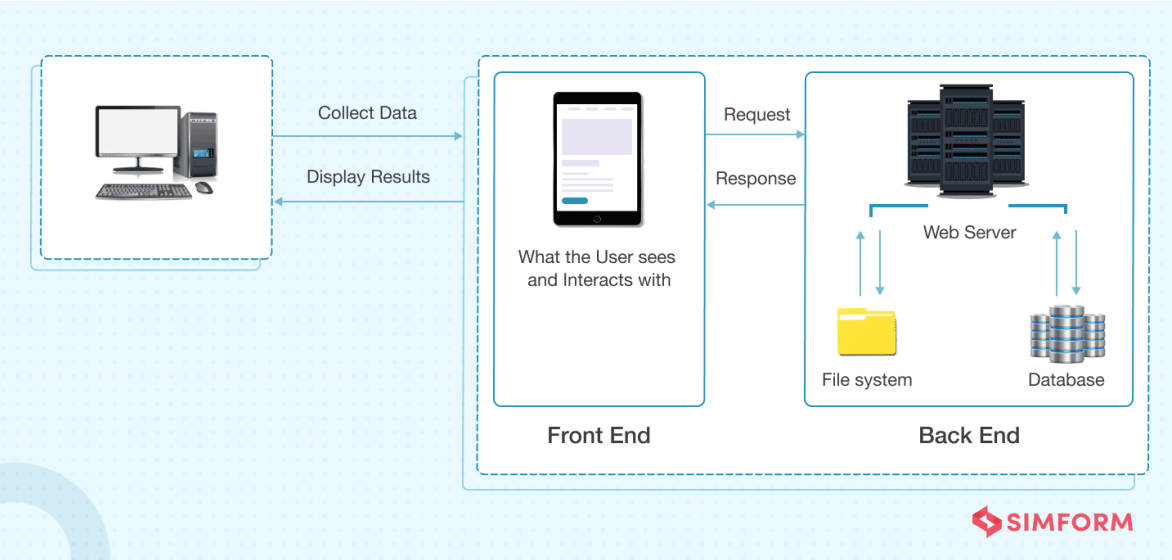
Väčšiu dekompozíciu je možné dosiahnuť pomocou takzvaných mikroslužieb riadených udalosťami. V tomto prípade už spolu služby nekomunikujú pomocou synchrónnych volaní, ale pomocou asynchrónnych volaní, teda pomocou sprostredkovateľa eventov. Je tak omnoho jednoduchšie pridávať nové služby do existujúcej infraštruktúry. Väčšina aplikácií, ktoré sú založené na architektúre mikroslužieb využíva kombináciu synchrónnych a asynchrónnych správ na komunikáciu medzi jednotlivými službami. [16]



Obrázok 4. Architektúra aplikácie využívajúcej mikroslužby [9]

4 KOMPONENTY WEBOVÝCH APLIKÁCIÍ

V súčasnosti sa webové aplikácie skladajú z dvoch základných komponent a to z frontendu, a backendu. Tieto dve komponenty medzi sebou komunikujú prostredníctvom protokolu HTTP. Súčasťou backendu býva webový serer, ktorý komunikuje s databázou a súborovým systémom. [9]



Obrázok 5. Schéma interakcie užívateľa, frontendu a backendu aplikácie [9]

4.1 Frontend webovej aplikácie

Frontend alebo taktiež klientská časť webovej aplikácie predstavuje súbor všetkých elementov ktoré používateľ vidí a s ktorými interaguje pri používaní aplikácie. Frontend predstavuje užívateľské rozhranie aplikácie. V rámci tejto komponenty sa používajú tri základné jazyky a to HTML, CSS a JavaScript. [17]

Vykresľovanie frontendu webovej aplikácie ma na starosti prehliadač. Ten pred samotným vykreslením stránky najprv zapracuje HTML na DOM strom a CSS na CSSOM strom. Tieto dva stromy sú pritom navzájom nezávislé. DOM predstavuje rozhranie, ktoré charakterizuje dokument a umožňuje manipuláciu s týmto dokumentom. S daným stromom môže JavaScript manipulovať pomocou rôznych funkcií. CSSOM predstavuje rozhranie, ktoré reprezentuje štýly dokumentu a umožňuje ich úpravu. Po vytvorení týchto DOM a CSSOM stromov nastáva takzvané renderovanie, pri ktorom sú oba tieto dva stromy skombinované. Vznikne tak DOM strom s aplikovanými štýlmi, takzvaný renderovací strom. Tento strom slúži na výpočet polohy všetkých viditeľných prvkov na obrazovke, na ktorú sú následne vykreslené. Vykresľovanie môže byť prerušené v prípade, že renderovací nástroj

prehliadača narazí na JavaScript kód vložený do HTML. V takom prípade JavaScript engine vykoná daný kód a následne renderovací nástroj dokončí vykresľovanie HTML. [17]

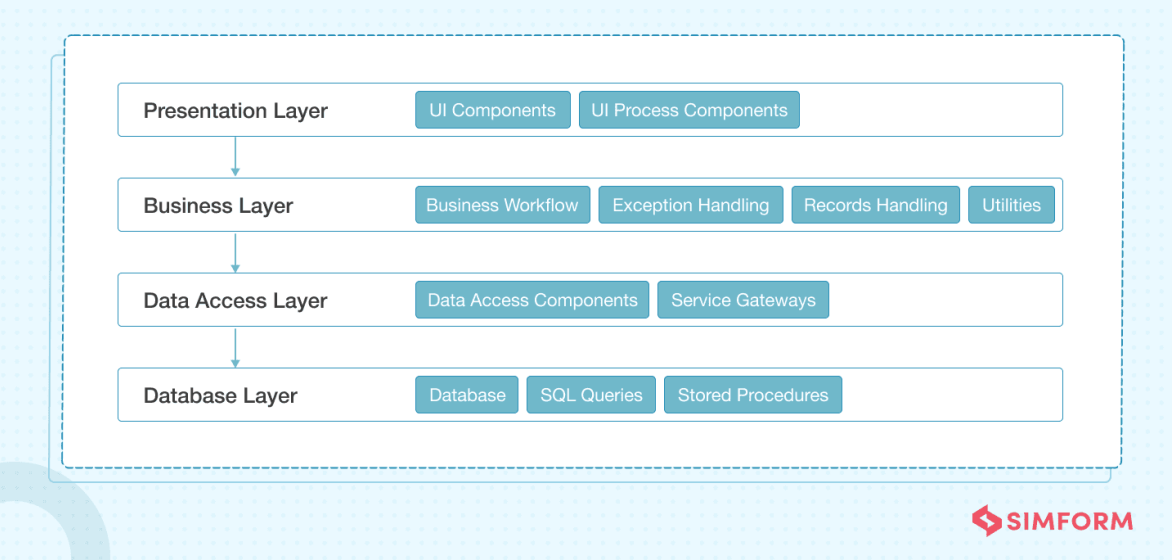
4.2 Backend webovej aplikácie

Backend, alebo taktiež serverová časť, predstavuje aplikačnú logiku webovej aplikácie. Backend určuje, ako sa bude aplikácia chovať. Táto časť v podstate slúži ako sprostredkovateľ medzi frontendom a samotnou databázou. Na základe HTTP požiadavkou, ktoré dostáva z frontendu aplikácie, zasiela dáta z databáze naspať frontendu. [18]

Bežnými programovacími jazykmi backendu sú napríklad PHP, JAVA, .Net. Vznik serverového prostredia Node.js ale umožnil, aby sa na programovanie serverovej časti webových aplikácií používal JavaScript. Ten bol donedávna jazyk používaný čisto len na frontende. Týmto došlo k zjednodušeniu vývoja webových aplikácií. [18]

5 VRSTVY ARCHITEKTÚRY WEBOVÝCH APLIKÁCIÍ

Webové aplikácie sa skladajú z niekoľkých vrstiev. Asi najbežnejšie sa používajú štyri základné vrstvy. A to z prezentačnej, business, vrstvy dátového prístupu a databázovej vrstvy. Každá vrstva pracuje nezávisle na ostatných vrstvách. Je možné vykonávať zmeny v jednej vrstve bez toho, aby bolo nutné zasahovať do ostatných vrstiev. Každá vrstva komunikuje len so susednou vrstvou. [9]



Obrázok 6. Vrstvy webovej aplikácie [9]

5.1 Prezentačná vrstva

Táto vrstva slúži na zobrazovanie užívateľského rozhrania a ako základ pre interakciu užívateľa so samotnou webovou aplikáciou. Okrem iného zobrazuje dáta a spracováva užívateľke požiadavky. Hlavnou úlohou tejto vrstvy je zachytávať užívateľské požiadavky a ich odosielanie na spracovanie a následne zobrazenie výsledku zmieneného spracovania. Z pohľadu štruktúry tak prezentačná vrstva predstavuje frontend aplikácie, zobrazuje sa vo webovom prehliadači. Na tejto vrstve sa pracuje s technológiami ako CSS, HTML, Java a JavaScript. [19]

5.2 Business vrstva

Táto vrstva reprezentuje logiku webovej aplikácie. A je zodpovedná za výmenu dát. Zapúzdruje funkcionalitu a definuje business operácie, pravidlá a požiadavky. Táto vrstva určuje ako bude prebiehať komunikácia medzi frontendom a backendom a predstavuje backend webovej aplikácie. [19]

5.3 Vrstva databázového prístupu

Táto vrstva umožňuje prístup k zdrojom ako je napríklad databáza alebo rôzne iné druhy súborov, ako napríklad binárne súbory, XML súbory a iné. Táto vrstva okrem iného uskutočňuje takzvané CURD operácie. [19]

5.4 Databázová vrstva

Táto vrstva má za úlohu uchovávanie dát. Úlohou tejto vrstvy je teda oddeliť business logiku od prezenčnej vrstvy. Takto sa zvyšuje bezpečnosť ochrana informácií. [9]

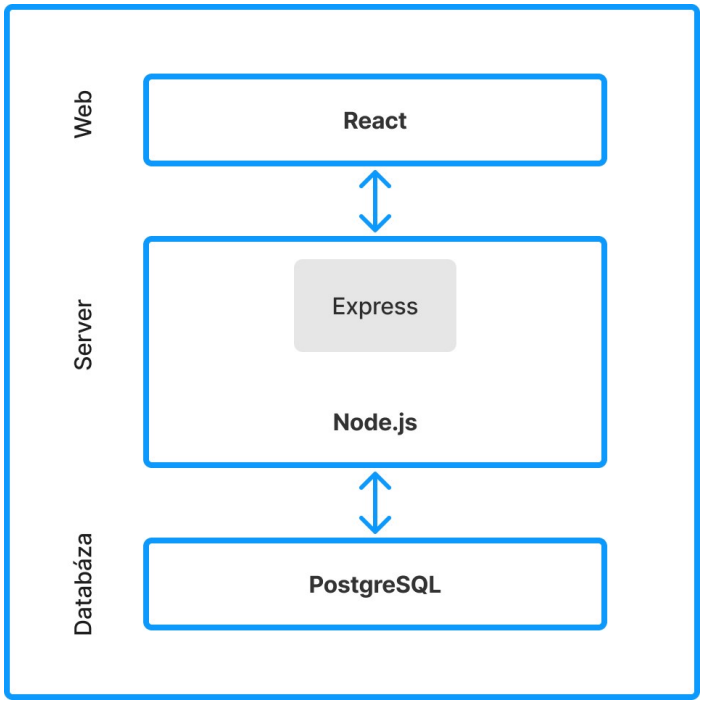
6 WEBOVÝ STACK

Vývoj webových aplikácií v sebe obyčajne zahŕňa veľké množstvo rôznych technológií, frameworkov a programovacích jazykov. Všetky tieto technológie a jazyky sa dajú združiť do takzvaných webových stackov. Webový stack je teda zoznam všetkých nástrojov potrebných na vývoj webovej aplikácie, slúži ako určitý návod aké technológie použiť pri vývoji. Tento stack v sebe obsahuje technológie klientskej, serverovej časti, technológie databáze a webového serveru, pričom technológie používané v rámci webového stacku bývajú väčšinou vo forme takzvaného otvoreného softvéru (Open-source software). [20]

V súčasnosti existuje veľké množstvo webových stackov. Jeden z prvých vytvorených stackov, a historicky asi najznámejším, je LAMP stack. Tento stack sa skladá z technológií Linux, Apache, MySQL, PHP a je určený na vývoj dynamických webových stránok. Prvý stack zameraných špeciálne na vývoj jednostránkových aplikácií je MEAN stack. MEAN stack sa skladá z technológií MongoDB, ExpressJS, Angular a Node.js. Následne začali vznikať rôzne obdoby tohto stacku, ktoré sa líšia v použitých technológiách frontendu a v použitej databázy. Jednou takou odnožou je MERN stack, u ktorého nastalo nahradenie frameworku Angular za knižnicu React. [20]

6.1 PERN stack

PERN stack je webový stack zameraný na vývoj jednostránkových webových aplikácií. Tento stack je obdoba široko obľúbeného MERN stacku, v ktorom je dokumentová databáza MongoDB nahradená objektovo-relačnou SQL databázou PostgreSQL. Skladá sa z technológií PostgreSQL, ExpressJS, React a Node.js. PERN stack je vhodné používať v prípade že namiesto rýchlejšej distribuovanej dokumentovej databázy chceme, aby použitá databáza splňovala ACID (Atomicity, Consistency, Isolation, and Durability) charakteristiky. Zároveň je vhodnejšie použiť PERN stack tam, kde vieme, že ukladané dáta budú mať hierarchickú štruktúru. [21]



Obrázok 7. PERN stack [21]

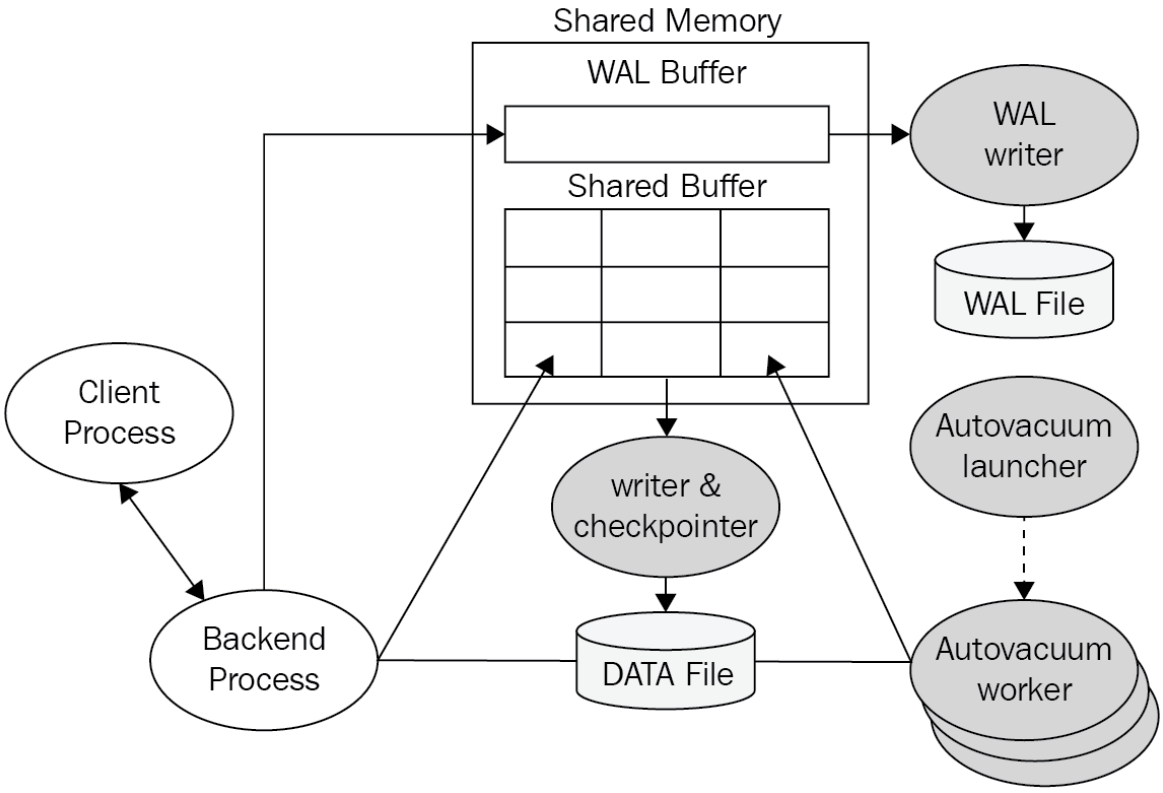
6.1.1 PostgreSQL

PostgreSQL je pokročilý, všestranný, objektovo-relačný databázový systém, ktorý využíva a rozširuje SQL jazyk. PostgreSQL vznikol v priebehu deväťdesiatych rokov na University of California v Berkeley pod hlavičkou Database Research Group. Dôležitou vlastnosťou PostgreSQL je, že sa jedná o takzvaný otvorený softvér. Aj napriek tomu sa na jeho vývoji a údržbe v súčasnosti podieľa obrovské množstvo ľudí z celého sveta. PostgreSQL teda nemá žiadneho vlastníka, jeho vývoj nie je pod hlavičkou žiadnej firmy. Pri použití PostgreSQL nie je po užívateľovi vyžadovaná registrácia použitia. PostgreSQL býva často spojovaný s MySQL, avšak s PostgreSQL sa technicky viac blíži k Oracle databázy než k MySQL. PostgreSQL je možné ľahlo rozšíriť o vlastné dátové typy, operátory alebo funkčný jazyk. Aj napriek tomu, že PostgreSQL funguje na princípe otvoreného softvéru existuje mnoho spoločností, ktoré poskytujú podporu pre firemných užívateľov, nie je teda problém používať PostgreSQL pre podnikové systémy, ktoré si vyžadujú komplexnú formu užívateľskej podpory. [22]

PostgreSQL sa vyznačuje veľmi jednoduchou architektúrou. Pri vývoji PostgreSQL sa dbá na dodržiavanie štandardu databázového jazyka SQL (ISO/IEC 9075), i keď sa nejedná o striktné dodržiavanie tohto štandardu. V prípade, že by dodržanie tohto štandardu bolo v rozpore s filozofiou PostgreSQL a dodržanie tohto štandardu by mohlo viesť k problému

s architektúrou, štandard nie je dodržaný. PostgreSQL 14.1 tak spĺňa 170 zo 177 nutných vlastností SQL:2016 štandardu. V základnej verzii sa PostgreSQL skladá zo zdieľanej pamäte, malého množstva procesov vykonávaných na pozadí a dátových súborov. [22]

PostgreSQL vyniká taktiež tým, že umožňuje používanie testovacích frameworkov. Dáva možnosť použiť techniku vývoja riadeného testami (TDD). Vývojár tak môže používať automatické testovacie nástroje. Tieto testy sa priamo ukladajú do PostgreSQL databáze. Po vykonaní testov sú všetky zmeny opäť navrátené do pôvodného stavu. Nie je teda možné pomocou testov databázu pokaziť. [23]



Obrázok 8. Architektúra PostgreSQL [22]

6.1.1.1 Robustnosť

PostgreSQL je vysoko robustný databázový server. Všetky akcie vykonávané v databázy sa uskutočňujú v rámci takzvaných transakcií. Zoznam všetkých týchto transakcií je logovaný a je možné automatické zotavenie po zlyhaní databáze. Databázový server obsahuje mnoho mechanizmov a nástrojov na diagnostiku a obnovu. Ako napríklad kontrolné súčty na diagnostiku chýb hardvéru alebo PITR. PostgreSQL už v základnej verzii obsahuje replikáciu dát. Pomocou správnej konfigurácie je možné dosiahnuť 99.999% dostupnosť. [22]

6.1.1.2 Bezpečnosť

PostgreSQL obsahuje širokú škálu pravidiel a mechanizmov, ktoré majú čo najviac zvýšiť bezpečnosť. Napríklad prístup k PostgreSQL sa riadi pomocou takzvaných host-based prístupových pravidiel, kde je prístup do databáze udelený hosťovi a ostatný používatelia, ktorý pristupujú z tohto hosťa majú automaticky prístup do danej databáze. Okrem iného PostgreSQL umožňuje plne SSL kódovaný prístup k dátam a taktiež na replikáciu dát už v základnej verzii. Ďalej PostgreSQL umožňuje RLS pre vojenskú úroveň zabezpečenia alebo lekárske záznamy. [22]

6.1.1.3 Rozšíriteľnosť

Do PostgreSQL je navrhnutý ako vysoko rozšíriteľný. PostgreSQL v sebe už v základnej verzii má mechanizmus na inštaláciu a správu rozšírení. Je taktiež možno si nadefinovať vlastné dátové typy, operátory, indexy a funkcie. [22]

6.1.1.4 Výkon

Samotný výkon PostgreSQL je silne závislí od hardvérovej konfigurácie a operačného systému stroja na ktorom databázový server beží. Každá konfigurácia je vhodná na iný prípad použitia. Niektoré nastavenie môže spôsobiť zrýchlenie niektorých dotazov, avšak spomaliť iné dotazy. [23]

6.1.1.5 Vlastnosti ACID

Táto skupina vlastností zaručuje, že vykonané databázové transakcie budú uskutočňované bezpečne. Každá transakcia by mala splňovať požiadavky na nedeliteľnosť, konzistentnosť, izoláciu a trvanlivosť. Nedeliteľnosť znemená, že každá transakcia sa vykoná ako jedna nedeliteľná operácia. V prípade zlyhania transakcie sa databáza vráti do pôvodného stavu a databáza už nebude obsahovať žiadne časti transakcie, ktorá zlyhala. Konzistentnosťou nám zaručuje, že pred a po vykonaní transakcie bude databáza v platnom stave. Izoláciou transakcií rozumieme ochranu transakcií pred vonkajšími vplyvmi. Trvanlivosť zaručuje uchovanie dát v databázy po úspešnom vykonaní transakcie. [24]

6.2 ExpressJS

ExpressJS je open-source framework na tvorbu webových aplikácií pracujúcim na Node.js. Výhodou tohto freameworku je jeho jednoduchosť a veľmi dobrá štruktúra. Ďalšou veľkou výhodou je fakt že programovacím jazykom ExpressJs je JavaScript. ExpressJs je vhodný na tvorbu MPA, SPA alebo hybridných webových aplikácií. Dôležitou súčasťou ExpressJS je takzvaný middleware. [25]

6.2.1 Middleware

Middleware predstavuje funkcie, ktoré sú spracovávané frameworkom ExpressJS uprostred iných funkcií. To znamená, že middleware predstavuje prostredníka medzi príchodom požiadavku a odpoveďou na túto požiadavku. Výstupom požiadavky môže byť priamo výstup alebo vstup do ďalšej funkcie, takzvaného middlewaru. Middleware je takto možné reťaziť, pričom poradie vykonávaných funkcií zostáva zachované. Pomocou middleware je možné upravovať vstupné požiadavky, alebo odpovede na tieto požiadavky. Častým príkladom použitia middleware býva validácia odoslaných formulárov pred samotným zápisom do databáze. [25]

6.3 React

React bol vytvorený Jordanom Walkerom, ktorý v dobe jeho vývoja pracoval ako softwarový vývojár Facebooku. React bol po prvý použitý v takzvaných news feed u Facebooku v roku 2011. Neskôr bol pridaný do Instagramu. Pôvodne bol React vyvíjaný len pre potreby Facebooku. Plnohodnotnou open-source knižnicou sa React stal až v roku 2013. [26]

Zo začiatku React slúžil v podstate ako zobrazovacia vrstva v klasickom MVC návrhovom vzore. React však nie je MVC framework, ale len knižnica, ktorá silne nabáda k vytváraniu znovu použiteľných komponent používateľského rozhrania. Úlohou Reactu teda zo začiatku bolo zobrazovať užívateľské rozhranie pre aplikácie napísané pomocou JavaScriptu. Najväčší rozmach zažil React v rokoch 2015-2016, kedy začalo vznikať obrovské množstvo nástrojov pre React, ako napríklad Mobx, Redux a React Router. Tieto nástroje významne rozšírili funkcionalitu Reactu a prispeli tak k jeho stále sa zväčšujúcej popularite. React už tak prestal predstavovať len spôsob ako efektívne zobrazovať a pracovať s užívateľským rozhraním. Pomocou rozširujúcich knižníc je možné React prispôsobiť aktuálnym potrebám. Napríklad pomocou knižnice React Router je možné ho použiť na smerovanie SPA. React tak prestal predstavovať len zobrazovaciu vrstvu MVC architektúry. Ďalšieho významného

míľniku sa React dočkal v roku 2019, kedy boli predstavene takzvané React Hooks. React Hooks umožňujú zdieľať stavovú logiku skrz jednotlivé komponenty. [26]

6.3.1 React Hooks

Problémom Reactu v minulosti bola chaotická práca s objektom *this*. Objekt *this* slúži ako referenčný odkaz na objekt, ku ktorému patrí. Na aký objekt je *this* referencia silne záleží v akom kontexte je tento objekt volaný. Napríklad u metód je *this* referencia na inštanciu triedy, v prípade, že *this* stojí mimo triedu, alebo funkciu je *this* odkaz na globálny objekt. A v prípade použitia takzvaného strict módu je *this* nedefinovaný. Problém je taktiež možnosť dynamicky meniť referenciu objektu *this*. To znamená, že nie je vždy na prvý pohľad zrejmé na aký objekt *this* ukazuje. Použitie objektu *this* sa tak stáva veľmi chaotické. Naprogramovaný kód nie je ľahké pochopiť a je takmer vždy nutné ho dôkladne študovať. [26]

React Hooks slúžia na zapuzdrenie stavového manažmentu. Používajú k tomu už existujúce funkcie JavaScriptu. React Hooks umožňujú jednoduché znovu použitie stavovej logiky medzi jednotlivými komponentami. Pomocou React Hooks už nemusí vývojár zbytočne zapuzdrovať logiku a vytvárať tak zložité stromy a pod stromy komponent, ktoré sú zložité na pochopenie. [27]

6.4 Node.js

Node.js je viacplatformové, open-source behové prostredie postavene na skriptovacom jazyku JavaScript a využívajúci V8 engine od Googlu. Tento engine je mimo iné používaný aj v prehliadači Google Chrome. Po prvý krát bol Node.js predstavený v roku 2009 Ryanom Dahlom na konferencii European JSConf. Hlavnou úlohou Node.js bolo vytvoriť nové behové prostredie pre JavaScript, ktorý doposiaľ mohol fungovať len v rámci webových prehliadačov, tak aby bolo možné umožniť full-stack vývojárom vytvárať aplikácie v jednom programovacom jazyku. [28]

Node.js je pamäťovo veľmi nenáročný. Jedná sa o takzvaný „ľahký“ webový server. Je určení prevažne na spúšťanie kódu strane serveru. Node.js sa javí ako ideálne riešenie pre situácie, kde je treba spracovať obrovské množstvo požiadavkou. Je vhodný pre aplikácie ktoré, potrebujú veľké toky dát, real-time aplikácie a aplikácie s nadmerným zápisom alebo čítaním. [28]

Node.js sa vyznačuje svojou vysokou rýchlosťou. Väčšina vstupne výstupných požiadaviek, ktoré sú inak blokujúce, sú spracovávane asynchrónne. Node.js obsahuje neblokujúci I/O

model. Node.js obsahuje jednovláknovú slučku udalostí, ktorá spracováva všetky vstupne výstupné operácie asynchrónne. Časovo náročné operácie neblokujú vlákno a Node.js je aj napriek tomu, že má len jedno vlákno, veľmi rýchly. Zároveň Node.js nezaťažuje vývojárov správou vlákien, lebo sa všetko deje na pozadí a vývojár nad tým nemá takmer žiadnu kontrolu. Všetky vstupne výstupné operácie sa vykonávajú v jednom okamžiku. Pre každú požiadavku je na pozadí vytvorené zvláštne vlákno zatiaľ čo hlavné vlákno Node.js obsluhuje ostatné požiadavky. Obsluha týchto požiadaviek sa deje pomocou fronty udalostí, čo znamená že, po ukončení vstupne výstupného požiadavku sa práca presunie automaticky na hlavné vlákno. Prepínanie kontextu sa v Node.js deje minimálne a hlavné vlákno teda takmer vždy niečo vykonáva. Node.js tak dokáže obslúžiť väčšie množstvo klientskych požiadaviek. [28]

Veľkou výhodou Node.js je, že Node.js umožňuje používať JavaScript nie len ako nástroj na strane frontendu, ale aj na strane backendu. Čo znamená, že kód frontendu a backendu je si veľmi podobný. Znižujú sa tak požiadavky na vývojárov. Node.js tak zvyšuje efektivitu s akou sa vyvíjajú webové aplikácie. [28]

7 JAVASCRIPT

Podľa každoročného prieskumu Developer Survey 2021 od StackOverflow je JavaScript najpoužívanejším programovacím jazykom v súčasnosti. Najpoužívanejším programovacím jazykom je JavaScript, podľa zmieneného prieskumu, už deviaty rok v rade. Na obrovskej popularite taktiež získava TypeScript, ktorý je v podstate len nadstavba JavaScriptu. Ten sa v roku 2021 objavil na piatej priečke v najpoužívanejších programovacích jazykoch. [29]

JavaScript je dynamický, vysokoúrovňový, interpretovaný programovací jazyk. Tento programovací jazyk je možno používať v objektovo orientovanom štýle programovania, ale aj vo funkcionálneho štýle programovania. Výhodou JavaScriptu je jeho relatívna jednoduchosť. Pôvodné behové prostredie JavaScriptu bol webový prehliadač. Od roku 2010 je možné používať nové behové prostredie, a to Node.js. Použitie JavaScriptu na Node.js a vo webových prehliadačoch nie je však sto percentne totožné. Node.js obsahuje niektoré funkcie, ktoré nie sú vo webovom prostredí dostupné a naopak. Napríklad vo webových prehliadačoch nie je možné priamo pristupovať do databáze alebo k súborom na disku pomocou JavaScriptu. Toto je však možné v prípade, že JavaScript beží na Node.js. Vo webovom prehliadači je možné pristupovať k objektu *window*, zatiaľ čo u Node.js takýto objekt neexistuje. [30]

8 FUNKCIONÁLNE PROGRAMOVANIE

Medzi najznámejšie paradigma vývoja patrí procedurálne, objektovo orientované a funkcionálne paradigma. Funkcionálne programovanie je takzvané deklaratívne programovacie paradigma. Toto paradigma sa vyznačuje tým, že už nie je dôležité ako sa daná vec deje, ale čo sa má diať. Takže na rozdiel od imperatívneho programovacieho paradigma, ktoré sa sústredí na konkrétny postup krok za krokom ako sa má daný kus kódu vykonávať, sa deklaratívne zaoberá otázkou čo sa má diať. Vývojárov už nezajíma celý konkrétny postup implementácie. V prípade funkcionálneho programovania už vývojár nepracuje s triedami a prototypmi. Funkcionálne programovanie však nie je protikladom objektovo orientovaného programovania. Je možné, a niekedy aj vhodné, oba prístupy mixovať a používať naraz. [30]

Funkcionálne programovanie je programové paradigma zameraná na používanie takzvaných čistých funkcií (pure function), nemenných hodnôt (immutable values) a iných. Cieľom funkcionálneho programovania je sprehľadniť a zjednodušiť pochopenie kódu. U funkcionálneho programovania je dôležité, aby funkcie boli krátke a ľahké na pochopenie. Kladie sa tu taktiež dôraz na správne názvy funkcií. Na pochopenie k čomu daná funkcia slúži by malo stačiť prečítať si iba názov funkcie. [30]

8.1.1 Čisté funkcie

Úlohou čistej funkcie je sprehľadniť kód. Za čistú funkciu môžeme pokladať takú funkciu, ktorá nemá žiadny vedľajší efekt. Čistá funkcia teda nemení žiadne hodnoty ktoré sa nachádzajú mimo túto funkciu. Táto funkcia taktiež nemôže pracovať s externými zdrojmi. Nie je možné napríklad čítať premenné, ku ktorým majú prístup ostatné funkcie, ako napríklad globálne premenné, alebo premenné mimo rozsah aktuálnej funkcie. Neobsahujú žiadny vedľajší efekt. U týchto funkcií ďalej nie je možné, aby parametre funkcie boli menené. Pristupuje sa k nim preto ako k nezmeniteľným hodnotám. Čisté funkcie taktiež neobsahujú pseudo parameter *this*. Čisté funkcie by v sebe taktiež nemali obsahovať volania na funkcie, ktoré nie sú čisté, a to s jednou výnimkou. Čistá funkcia v sebe môže obsahovať volanie na nečistú funkciu v prípade, že táto nečistá funkcia iba mení jej vstupné parametre. V takomto prípade ostáva zachovaná čistota funkcie. [31]

Čistá funkcia je teda taká funkcia, ktorá je deterministická. Pri rovnakých vstupných argumentoch je návratová hodnota funkcie vždy rovnaká. Výhodou použitia čistých funkcií je ich veľmi dobrá čitateľnosť, ich ľahká tvorba, testovanie a ladenie. Čisté funkcie v sebe

neobsahujú žiaden cyklus. Klasické cykly ako *for*, *foreach*, *while* sú nahradzované takzvanými funkciami vyšších rádov alebo rekurziou. [32]

8.1.2 Vedľajší efekt funkcie

Za vedľajší efekt je možné považovať zmenu premennej, ktorá sa uskutoční mimo aktuálny rozsah funkcie alebo nejakú interakciu s vonkajším prostredím, ktorá sa uskutoční behom vykonávania funkcie. Vedľajším efektom môže byť napríklad čítanie alebo zapisovanie do lokálneho úložiska, sieťové volania alebo emitovanie udalostí. Nežiaduce účinky majú za následok nepredvídateľné správanie funkcií, čo má za následok ťažké testovanie a odhaľovanie chýb. [32]

8.1.3 Funkcionálne programovanie v JavaScripte

JavaScript nepatrí medzi klasické funkcionálne programovacie jazyky. Nevyžaduje totižto striktné dodržiavanie princípov funkcionálneho programovania. Umožňuje však tieto princípy, ako napríklad funkcie prvého rádu, anonymné funkcie, a uzávery, využívať. JavaScript v sebe ale taktiež obsahuje obrovské množstvo funkcií, ktoré majú nejaký vedľajší efekt, a tak priamo odporujú princípom funkcionálneho programovania. Je nutné pri programovaní v JavaScripte dávať pozor na prácu s týmito funkciami. [30]

II. PRAKTICKÁ ČÁST

9 MOTIVÁCIA

V súčasnosti na českom trhu neexistuje žiadny aplikačný nástroj, či už platený alebo bezplatný, ktorý by umožňoval jednoducho a efektívne zbierať dáta k ABA terapiám v českom jazyku. Tento stav je spôsobený tým, že je ABA v Českej republike relatívna novinka, na rozdiel od sveta, kde sa odbor ABA rozvíja už desiatky rokov. Počet certifikovaných behaviorálnych analytikov je v Českej republiky zatiaľ veľmi malý. Získanie certifikácie je totižto pomerne náročné, či už finančného alebo časového hľadiska. A tak počet certifikovaných behaviorálnych analytikov narastá len pomaly. V praxi sa využívajú princípy ABA v terapeutickom prostredí, prevažne u detí s neurovývojovými poruchami, a aj keď sa v Českej republike jedná o nelekársky zdravotný odbor, terapie momentálne nie sú hradené zo zdravotného poistenia tak, ako to je bežné napríklad vo všetkých štátoch USA. Kvôli týmto dôvodom zatiaľ zrejme neexistujú žiadne nástroje, ktoré by umožňovali zbierať dáta z terapií a mali by českú adaptáciu. Existuje síce niekoľko zahraničných webových nástrojov ktoré umožňujú zbieranie dát z ABA terapií. Avšak v týchto nástrojoch chýba lokalizácia do českého jazyka a bývajú často cenovo nedostupné.

Momentálne je v Českej Republike asi najpoužívanejší spôsob zápisu dát formou tužka-papier. Tento spôsob zberu dát je veľmi prácny, keďže po skončení terapie je nutné dáta následne prepísať do počítača, skenovať alebo zaniesť ručne do grafov. Časovo efektívnejšie by bolo použitie tohto času na priamu prácu so študentami. Takýto zber dát tak nepriamo zvyšuje cenu jednotlivých terapií. Terapeut a supervízor musí totižto počítať aj s prácou, ktorú strávi prepisovaní týchto dát pri určovaní ceny terapií.

Ako ďalšiu možnú nevýhodu tohto spôsobu zberu dát môžeme pokladať nemožnosť kontrolovať či dáta boli skutočne nazbierané v jednotlivé dni. Napríklad v prípade terapie vykonávanej rodičom sa často stáva, že rodič dáta nezbiera pravidelne, ale namiesto toho ich náhodne zapíše všetky až v deň, keď má predať report behaviorálnemu analytikovi.

Kvôli vyššie uvedeným dôvodom som sa rozhodol vytvoriť systém pre behaviorálnych analytikov a terapeutov, ktorý by umožnil jednoduchý a efektívny zber a vyhodnocovanie dát s automatickým základným vyhodnotením podľa nastavených kritérií a okamžitým prístupom pre spolupracujúci tým.

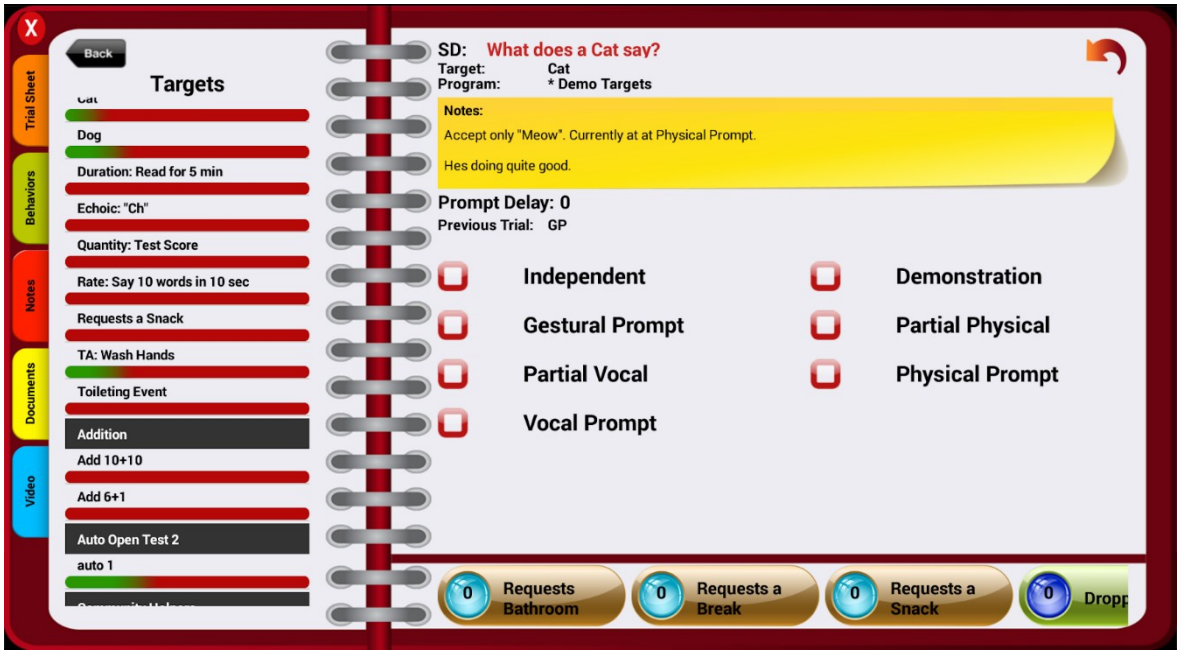
9.1 Súčasné nástroje

V súčasnosti existuje niekoľko zahraničných nástrojov na zber a spracovanie dát z ABA terapií. Hlavnou nevýhodou týchto nástrojov je ich komplikovanosť a prevažne anglická lokalizácia. Toto v podstate znemožňuje ich použitie v Českej republike. Napríklad mnoho štátnych inštitúcií vyžaduje, aby boli záznamy študentov vedené v českom jazyku. Veľkou prekážkou pri používaní týchto nástrojov býva taktiež ich zložitosť. Analytické nástroje sa často nesústredia len na ABA, ale aj na zber rôznych dát z veľkého množstva zdrojov. Obsahujú tak v sebe veľké množstvo funkcií a metód, ktoré sa pri ABA nevyužívajú na dennej bázy. Pre terapeutov býva ťažké sa vyznať vo všetkých funkciách. Je pomerne časté, že je nutných niekoľko hodín školení s daným nástrojom, kým je ho schopný používateľ plnohodnotne používať. Takéto školenie býva často spoplatnené. Napríklad u programu CentralReach hodina online školenia v súčasnosti stojí 150 amerických dolárov.

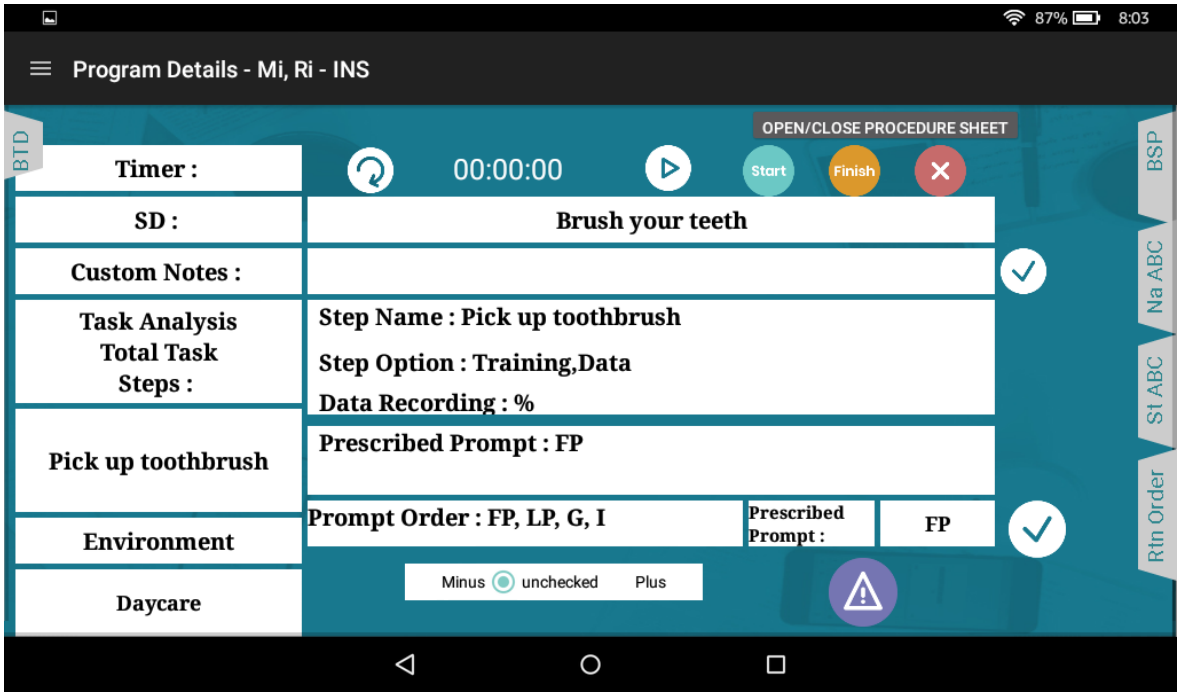
Ďalším spoločným rysom súčasných nástrojov je ich zastaralý dizajn a neintuitívne ovládanie. Napríklad ovládacie prvky bývajú často umiestnené veľmi neprirodzene. Býva preto nutné preštudovať veľké množstvo nápovede alebo dokumentácie, aby bolo možné s daným nástrojom plnohodnotne pracovať. Zmienené nástroje bývajú taktiež veľmi pomalé.

Prekážkou pri používaní týchto nástrojov je taktiež samotná cena. V súčasnosti v podstate neexistuje žiadny nástroj na zber dát z ABA terapií, ktorý by úplne bol bezplatný.

Všetky tieto nevýhody majú za následok to, že väčšina supervízorov, terapeutov a rodičov nepoužíva žiadny elektronický nástroj na zber a spracovanie dát z terapií. A namiesto toho sa spoliehajú len na klasickú metódu tužka-papier, a to pre jej širokú dostupnosť.



Obrázok 9. Užívateľské rozhranie aplikácie Catalyst Client [33]



Obrázok 10. Užívateľské rozhranie aplikácie ABA data collection [34]

10 ANALÝZA A NÁVRH

Pri analýze a návrhu som pôvodne chcel vychádzať mimo iné aj z už existujúcich nástrojov, ktoré mi odporučili sami ABA terapeuti. Tento postup bol však v podstate nemožný, keďže neexistuje takmer žiadny nástroj, ktorý by bol bezplatný alebo by obsahoval aspoň skúšobnú verziu. Keď už existovala skúšobná verzia nejakého nástroja, tak získanie tejto skúšobnej verzie bolo podmienené absolvovaním pohovoru, obyčajne v dĺžke 30 až 60 minút. Účelom týchto pohovorov malo byť overenie, že žiadateľ o skúšobnú verziu je skutočne ABA terapeut.

Samotná analýza a návrh prebiehal v spolupráci s certifikovanými ABA terapeutmi a za čiastočnej účasti rodičov študentov s PAS. Terapeuti ani rodičia nemali presnú predstavu, ako by mala finálna aplikácia vyzerať. Prvotná analýza bola preto neúmerne zdĺhavá.

Výsledkom sérii sedení bola prvotná špecifikácia požiadaviek, prípady použitia, návrh užívateľského rozhrania. V priebehu vývoja sa požiadavky, prípady použitia, a návrh užívateľského rozhrania menil a prispôsoboval aktuálnym požiadavkám terapeutov.

Hlavný nástroj na záznam analýzy a návrhu som použil CASE (Computer Aided Software Engineering) nástroj Enterprise Architect od firmy Sparx Systems. Na tvorbu drôtených modelov som použil prototypový webový nástroj Figma.

10.1 Metodika vývoja

Po rozhovoroch s terapeutmi som sa rozhodol zvoliť agilnú metódu vývoja. Táto metóda vývoja mi umožnila dynamicky reagovať na všetky nové požiadavky od terapeutov. A zároveň som tak mal vždy istotu, že sa projekt uberá správnym smerom. Po rozhovoroch s terapeutmi som totižto nadobudol silný dojem, že každý z nich má iný názor na to, ako by mal finálna aplikácia vyzerať, čo by mala riešiť a ako by to mala riešiť. Nastáva tu teda veľké riziko častých zmien v zadaní a nutnosť dynamicky na tieto zmeny reagovať, a postupne ich zapracovávať do finálneho produktu.

V rámci takzvanej nultej iterácie som s terapeutmi navrhol ako by mala vyzerať základná kostra aplikácie, funkcionálne a nefunkcionálne požiadavky, prípady použitia a grafy aktivít. Ďalšie iterácie nemali pevný časový rámec, ich dĺžka sa dynamicky menila na základe požiadaviek na funkcionalitu. Terapeuti totižto často menili to, ako sa má samotná aplikácia správať a tak bolo prakticky nemožné vytvoriť nejaký presný časový rámec.

10.2 Špecifikácia požiadaviek

Špecifikácia požiadaviek na systém prebiehala na základe pozorovaní terapií a následných rozhovoroch s terapeutmi. Aj keď bola z mojej strany veľká snaha definovať všetky požiadavky v rámci nultej iterácie, nepodarilo sa mi to dodržať. Veľká časť požiadaviek sa do projektu dostala až v priebehu samotného vývoja. Požiadavky boli terapeutmi často menené alebo úplne odstraňované.

10.2.1 Funkcionálne požiadavky

V priebehu analýzy som zaznamenal veľké množstvo funkcionálnych požiadaviek. Všetky tieto požiadavky som zapísal pomocou nástroja Enterprise Architect. Na základe týchto požiadaviek som následne vytvoril prípady požitia. V nasledujúcej tabuľke je stručný zoznam zozbieraných funkcionálnych požiadaviek. Zoznam všetkých funkcionálnych požiadaviek sa nachádza v prílohe.

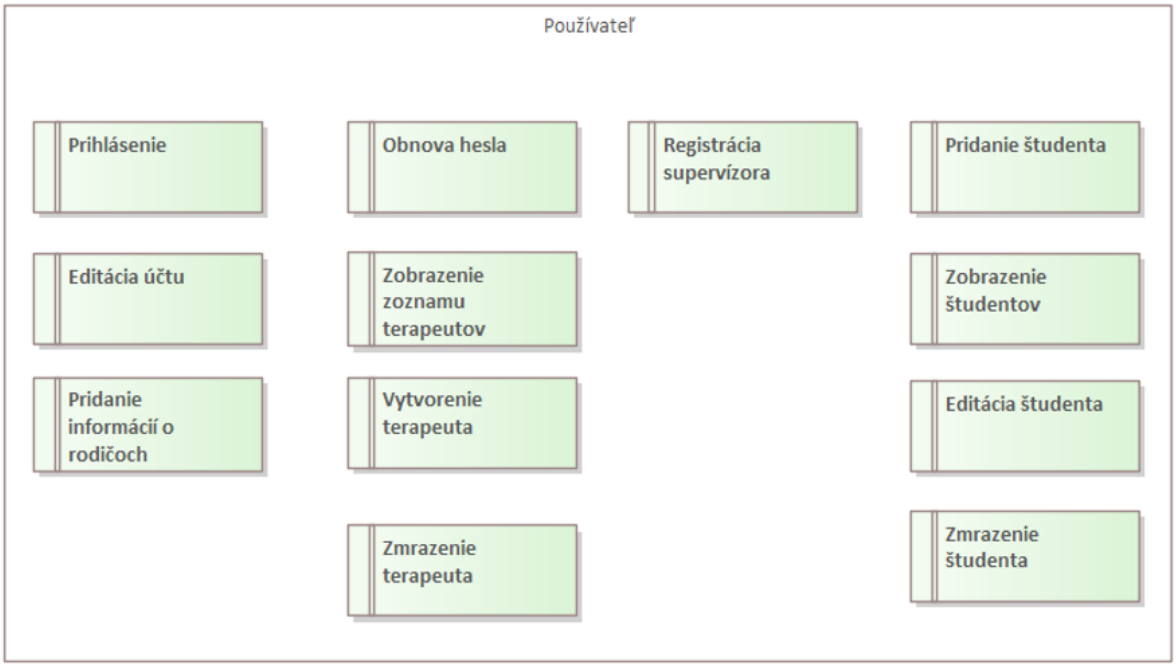
Tabuľka 1. Zoznam vybraných funkcionálnych požiadaviek

ID	Názov	Popis
FP_01	Registrácia supervízora	Systém bude umožňovať registráciu supervízora. Registrácia zaznamenáva mimo iného meno, email a heslo.
FP_02	Vytvorenie terapeuta	Systém bude umožňovať jednotlivým supervízorom pridávanie nových terapeutov. Každý terapeut je priradený k jednému supervízorovi a to k tomu, ktorý ho vytvoril. U terapeuta sa eviduje napríklad jeho meno, email.
FP_03	Pridanie študenta	Systém bude umožňovať pridať nového študenta. Do tohto profilu nie je možné sa prihlásiť. Z dôvodu ochrany osobných údajov je možné namiesto mena študenta používať len kód.

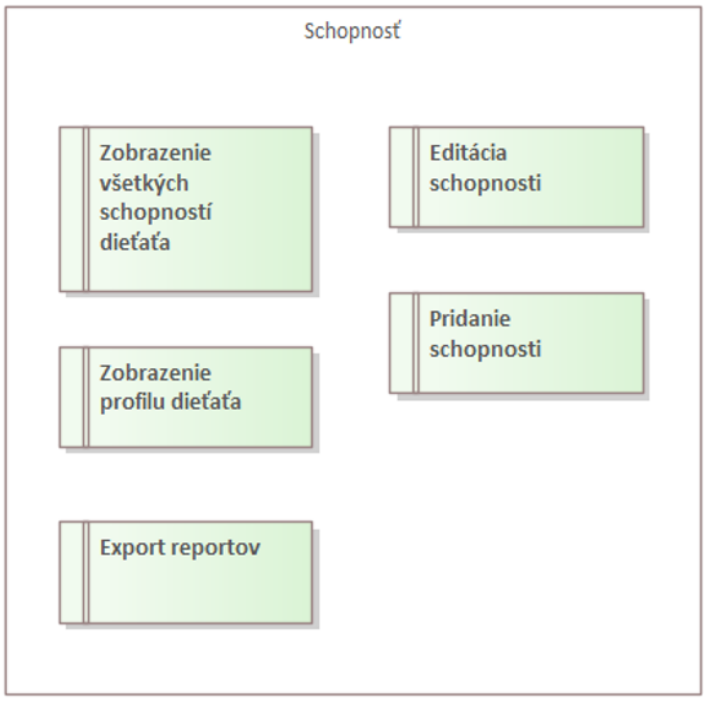
FP_04	Editácia	Systém bude umožňovať zmeniť osobné údaje všetkým aktérom ktorý sa môžu do systému prihlásiť. Je možné editovať všetky údaje okrem emailovej adresy.
FP_05	Pridanie cieľa	Supervízor a terapeut bude môcť pridať nový cieľ pre študenta. U cieľa je možné zadať hodnotu základnej úrovne. Poprípade, či už je základná úroveň programu splnená.
FP_06	Zber dát	Terapeutovi a supervízor budú môcť spustiť takzvané sedenie, v ktorom sa budú zaznamenávať jednotlivé dáta k cieľom. Systém musí zobraziť všetky ciele študenta, ktoré ešte nie sú uzatvorené a ktorých meranie ešte v daný deň neprebehlo. Všetky tieto ciele zobrazí systém na jednej stránke. Na to, aby bolo možné zaznamenávať dáta k jednotlivým cieľom nie je nutné sa preklikávať na podstránky.
FP_07	Kontrola nazbieraných dát	Systém bude umožňovať kontrolu zaznamenaných dát. Po tejto kontrole systém už neumožní terapeutovi zmeniť zaznamenané dáta.
FP_08	Zobrazenie detailu programu	Systém bude umožňovať zobraziť supervízorovi a terapeutovi detail programu. V detaile programu sa nachádzajú všetky ciele tohto programu.
FP_09	Zobrazenie grafov učenia	Systém bude umožňovať zobraziť terapeutom rôzne grafy učenie, ako napríklad kumulatívny

		graf, u každého programu. V rámci týchto grafov bude naznačená základná úroveň programu.
FP_10	Zabudnuté heslo	Systém bude umožňovať používateľom obnovu zabudnutého hesla.

Zoznamy funkcionálnych požiadaviek som sa kvôli sprehľadneniu rozhodol deliť do skupín podľa oblasti ktorej sa týkajú. Takto mi vzniklo niekoľko blokov funkcionálnych požiadaviek. Nasledujúce obrázky názorne ukazujú dve také skupiny.



Obrázok 11. Skupina funkcionálnych požiadaviek



Obrázok 12. Skupina funkcionálnych požiadaviek

10.2.2 Nefunkcionálne požiadavky

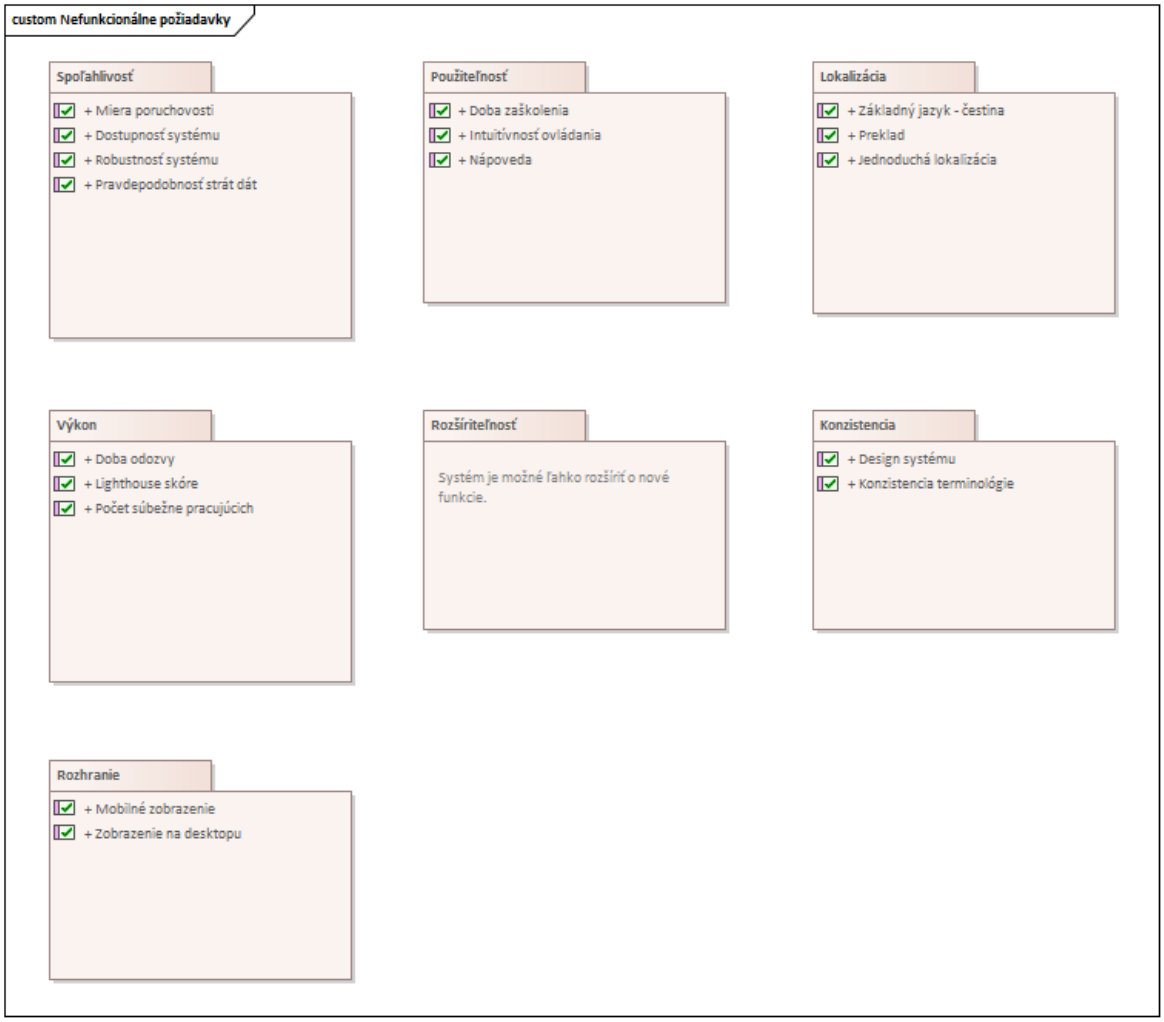
Aj funkcionálne požiadavky som zaznamenával v programe Enterprise Architect. Pri tvorbe nefunkcionálnych požiadaviek mi terapeuti niekoľko krát zdôrazňovali, že chcú aby aplikácia mala jednoduchý design a rýchlo reagovala na interakcie. Nasledujúca tabuľka zobrazuje skrátený prehľad zozbieraných nefunkcionálnych požiadaviek. Kompletný zoznam požiadaviek je súčasťou prílohy.

Tabuľka 2. Zoznam vybraných nefunkcionálnych požiadaviek

ID	Názov	Popis
NPS_01	Miera poruchovosti	Doba obnovy pre kritickú chybu je max 10 hodín. Pre závažnú chybu je to 16 hodín a pre nezávažnú chybu 24 hodín. **Kritická chyba**: webová aplikácia je nedostupná.

		Závažná chyba: webová aplikácia je dostupná, avšak nie je možné so systémom plnohodnotne pracovať. **Nezávažná chyba**: aplikácia sa dá normálne používať, ale používateľský komfort je znížený (napríklad v prípade grafickej chyby).
NPS_02	Dostupnosť systému	Systém bude dostupný nonstop. Ročná dostupnosť systému musí byť minimálne 95 percent.
NPS_03	Pravdepodobnosť strát dát	Systém bude zálohovať dáta, tak aby nedošlo k ich zničeniu. V prípade straty dát je maximálna doba na obnovu 24 hodín.
NPV_02	Prvotné načítanie	Prvé načítanie systému bude rýchle. Doba prvého načítania webovej stránky by nemala presiahnuť 3 sekundy.
NPV_03	Výkon	Systém bude rýchlo reagovať na požiadavky používateľov. Načítanie jednotlivých webových stránok by nemalo zabrať viac ako 1 sekundu.
NPR_01	Podporované prehliadače	Systém bude podporovať dve posledné verzie najpoužívanejších prehliadačov, konkrétne Google Chrome, FireFox, Edge, Safari a Opera.
NPR_02	Zobrazenie na mobilných zariadeniach	Systém bude možno plnohodnotne zobraziť a používať na mobilných zariadeniach.

NPP_01	Doba zaškolenia	Systém nebude vyžadovať len minimálnu dobu na zaškolenie používateľov. Systém by malo byť možné plnohodnotne využívať bez zaškolenia.
NPP_02	Dodržanie legislatívy	Systém bude dodržiavať legislatívu GDPR a všetky platné normy ohľadom zberu, uchovávania a vymazávania dát.
NPL_01	Lokalizácia	Systém bude primárne v českom jazyku.
NPL_02	Preklady	Systém bude môcť byť ľahko preložiteľný do iných jazykov.



Obrázok 13. Nefunkcionálne požiadavky navrhnuté

10.3 Vyhodnotenie požiadaviek

Na základe špecifikácie požiadaviek sa ukázalo, že pri tvorbe webovej aplikácie sa musím sústrediť na tri základné aspekty. A to na užívateľský zážitok, rýchlosť a intuitívnosť ovládania. Pri každom rozhovore mi bolo nakoľko krát pripomenuté, že je nutné, aby celkový zážitok z finálnej webovej aplikácie bol veľmi pozitívny. Počas rozhovorov bol zo strany terapeutov kladený obrovský dôraz na to, aby aplikácia bol rýchla. Terapeuti si neželali dlhé doby načítania jednotlivých stránok, webová aplikácia musí reagovať rýchlo na dotazy.

Špecifikácia požiadaviek jasne ukázala, že je nutné sa sústrediť na tvorbu webovej aplikácie ktorá, bude čo možno užívateľsky najprívetivejšia a zároveň bude dostatočne plynulo a rýchlo reagovať. Preto som sa rozhodol webový systém spracovať ako jednostránkovú aplikáciu. Vhodne naprogramovaná SPA totižto poskytuje vysoký užívateľský zážitok, a zároveň je dostatočne rýchla. Nevýhody SPA v prípade webovej aplikácie pre ABA terapeutov

nebudú také výrazné. Napríklad prvotné načítanie SPA, ktoré trvá u SPA obyčajne dlhšie než u klasických webových aplikácií, nebude veľkou prekážkou. Mojim hlavným cieľom však nie je minimalizovať prvotné načítanie stránky, ale minimalizovať dobu načítania pri interakcií používateľa s aplikáciou. V prípade, že by sa stalo prvotné načítanie prekážkou pri používaní, je možné aplikáciu upraviť tak, aby sa čas na toto načítanie minimalizoval. Napríklad pomocou asynchrónneho sťahovania zdrojov, alebo ukladania statických častí webovej aplikácie do takzvaných CDN (Content delivery network). CDN slúži k zvýšeniu dostupnosti zdrojov pre používateľov.

Ďalšou hlavnou nevýhodu SPA je ich zlá SEO optimalizácia. Táto nevýhoda u systému pre zber a spracovanie dát pre ABA terapie vôbec nevadí. Obsah a logika systému bude totižto dostupná až po registrácii a prihlásení. Vyhľadávače by sa k obsahu webovej aplikácie tak či tak nedostali, ani keby som použil tradičnú architektúru. Vynakladať veľké úsilie o SEO optimalizáciu by bolo zbytočné a nič navyše by neprinieslo. V prípade potreby zobrazovať systém pre vyhľadávače by bolo možné napríklad pripraviť takzvanú vstupnú stránku (landing page) webovej aplikácie ako statickú stránku. Táto vstupná stránka by slúžila ako brána do webovej aplikácie, ktorá by obsahovala odkaz na registráciu a prihlásenie do systému.

10.4 Aktéri

V spolupráci s terapeutmi boli definovaný jednotlivý aktéri a taktiež používateľské scenáre. Tieto scenáre následne môžu slúžiť ako základ napríklad pre akceptačné testy. Vo webovej aplikácii môžeme rozlíšiť dve skupiny aktérov a to: prihlásených a neprihlásených používateľov. Neprihlásený používateľ môže s aplikáciou interagovať len obmedzene. Prihasený používateľ predstavuje skupinu troch základných aktérov, ktorý sa môžu do webovej aplikácie prihlásiť a to: supervízora, terapeuta a rodiča. Daný aktéri sa od seba odlišujú v stupni oprávnenia. Používateľ ktorý sa registruje do systému získa automaticky oprávnenia supervízora. Ten následne môže vytvoriť jednotlivé profily pre ostatných aktérov. Terapeut a rodič sa sami nemôžu do aplikácie registrovať. Systém umožňuje komukoľvek sa do webovej aplikácie registrovať ako supervízor.

Supervízor – Supervízor predstavuje hlavného aktéra. Supervízor môže pridávať, upravovať a odstraňovať zručnosti, programy a ciele. Supervízor môže taktiež pridávať nových terapeutov, rodičov a študentov. Zároveň tiež môže k priradzovať terapeutov a rodičov k jednotlivým študentom. Iba supervízor sa môže sám registrovať do aplikácie. Každý používateľ pri registrácii sa automaticky stáva supervízorom.

Terapeut – Terapeut predstavuje ďalšiu aktívnu roľu. Úlohou terapeuta je hlavne zaznamenávať dáta k jednotlivým cieľom. Okrem toho môže terapeut pridávať jednotlivé ciele. Tieto ciele však už nemôže upravovať a odstraňovať. Na rozdiel od supervízora nemôže pridávať nové programy a zručnosti. A ani ich nemôže nijakým spôsobom upravovať. Terapeut má presne definovaných študentov, ku ktorým zaznamenáva jednotlivé dáta podľa vopred definovaných cieľov.

Rodič – Rodič predstavuje pasívnu rolu. Tento aktér má oprávnenia len na prezeranie detailov študentov ktorých má priradených. Nemôže žiadnym spôsobom nič upravovať ani pridávať. Profil rodiča teda slúži čisto len na pasívne prezeranie postupu študenta v učení. V prípade že by mal rodič figurovať ako terapeut, je mu vytvorený profil terapeuta a nie rodiča.

Používateľ – Používateľ predstavuje súhrnnú skupinu všetkých prihlásených používateľov. Táto skupina sa skladá z rodičov, terapeutov a supervízorov.

Neprihlásený používateľ – Tento aktér predstavuje používateľa, ktorý sa nachádza na hranici systému. Okrem registrácie, prihlásenia alebo obnovy hesla nemôže nijak inak interagovať so systémom. Nemôže prezerať ani upravovať dáta.

10.5 Prípady použitia

Jednotlivé prípady použitia vznikli na základe pozorovania terapií, diskusie s terapeutmi a rodičmi. Z mojej strany bola snaha definovať čo najviac prípadov použitia ešte pred samotným vývojom. Avšak to sa nakoniec ukázalo viacmennej ako nereálne a tak som jednotlivé prípady použitia často pridával, odoberal alebo rôzne upravoval počas samotného vývoja. A to z toho dôvodu že terapeuti presne nevedeli, čo od systému očakávajú. Každý mal inú predstavu, ako by sa mal systém chovať. S finálnou verziou prípadov použitia nakoniec boli všetci viacmennej spokojný. Nižšie uvádzam len pár týchto prípadov použitia ako ukážku, všetky prípady použitia sú súčasťou prílohy.

Tabuľka 3. Prípad použitia prihlásenia do systému

Prípad použitia: Prihlásenie do systému
Účastníci: Neprihlásený používateľ
Vstupné podmienky: Používateľ nie je prihlásený do systému
Tok udalostí: 1. Neprihlásený používateľ zadá prihlasovacie údaje 2. Systém skontroluje prihlasovacie údaje 3. Systém presmeruje prihláseného používateľa na nástenku
Vedľajší scenár: Neplatné Prihlasovacie Údaje
Nasledujúce podmienky: Používateľ je prihlásený do systému

Tabuľka 4. Prípad použitia neplatné prihlasovacie údaje

Prípad použitia: Neplatné prihlasovacie údaje
Účastníci: Neprihlásený používateľ
Vstupné podmienky: Používateľ nie je prihlásený do systému
Vedľajší scenár: 1. Prípad použitia začína v 2. kroku prípadu použitia „Prihlásenie do systému“, keď používateľ zadá nesprávne prihlasovacie údaje

2. Systém zobrazí chybové hlásenie
Nasledujúce podmienky:

Tabuľka 5. Prípad použitia meranie všetkých cieľov

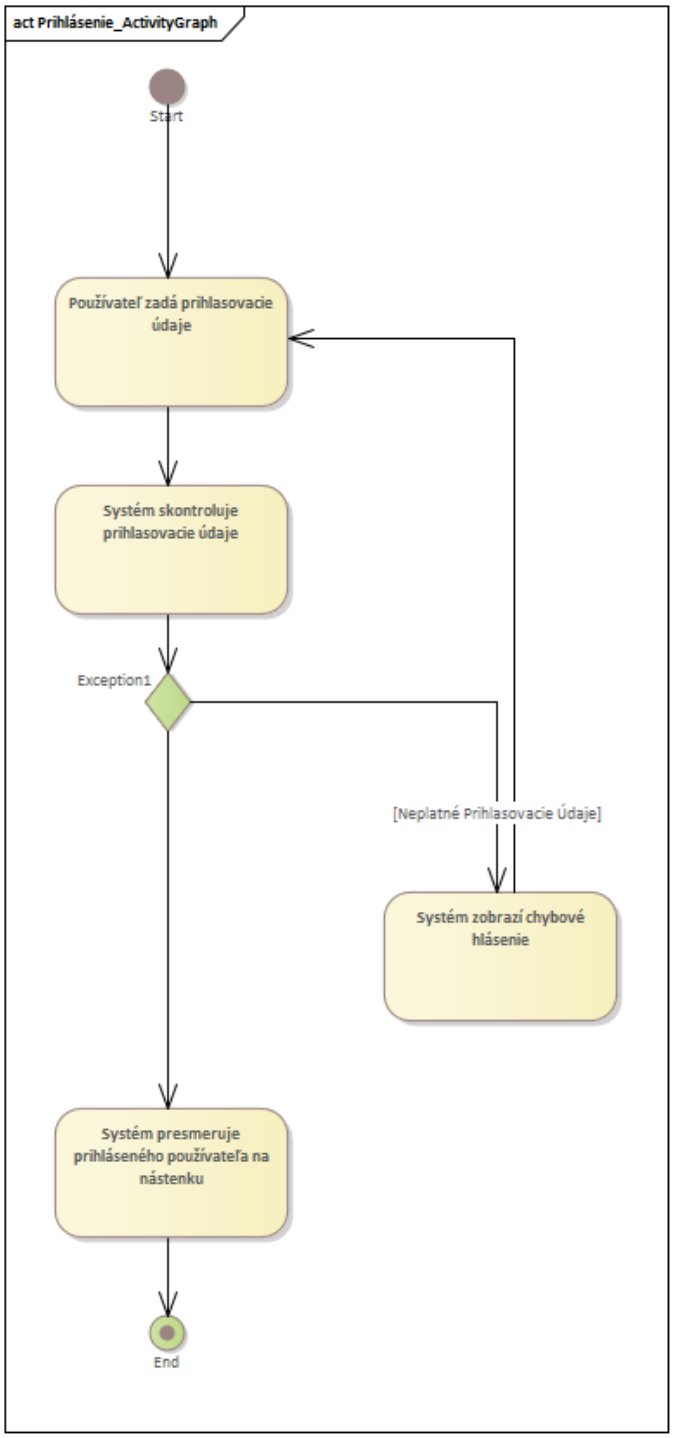
Prípad použitia: Meranie všetkých cieľov
Účastníci: Terapeut, supervízor
Vstupné podmienky: Terapeut alebo supervízor je prihlásený do systému a nachádza sa na stránke detail študenta.
Tok udalostí: 1. Používateľ klikne na tlačidlo zahájiť sedenie. 2. Systém presmeruje používateľa na stránku sedenia. 3. Systém vygeneruje všetky merania pre všetky otvorené ciele študenta, ktoré ešte v daný deň neboli zaznamenané a nie sú označené ako pozastavené. 4. Používateľ zadáva jednotlivé merania. 5. Systém každé meranie po jeho zadaní odstráni z obrazovky. 6. Systém zobrazí správu o úspešnom uložení merania.
Vedľajší scenár: Nedostupnosť meraní
Nasledujúce podmienky: Používateľ môže skontrolovať zadané merania k jednotlivým cieľom

Tabuľka 6. Používateľský scenár nedostupnosť meraní

Prípad požutia: Nedostupnosť meraní
Účastníci: Terapeut, supervízor
Vstupné podmienky: Terapeut alebo supervízor je prihlásený do systému a nachádza sa na stránke detail študenta.
Vedľajší scenár: 1. Prípad použitia začína v 3. kroku prípadu použitia „Meranie všetkých cieľov“, keď nie sú dostupné žiadne merania. 2. Systém zobrazí správu o nedostupnosti nových meraní.
Nasledujúce podmienky:

10.6 Diagramy aktivít

Na základe prípadov použitia som v programe Enterprise Architect vygeneroval jednotlivé diagramy aktivít. Nižšie je ukážka diagramov aktivít k výše uvedeným prípadom použitia. Všetky diagramy aktivít sú súčasťou prílohy.



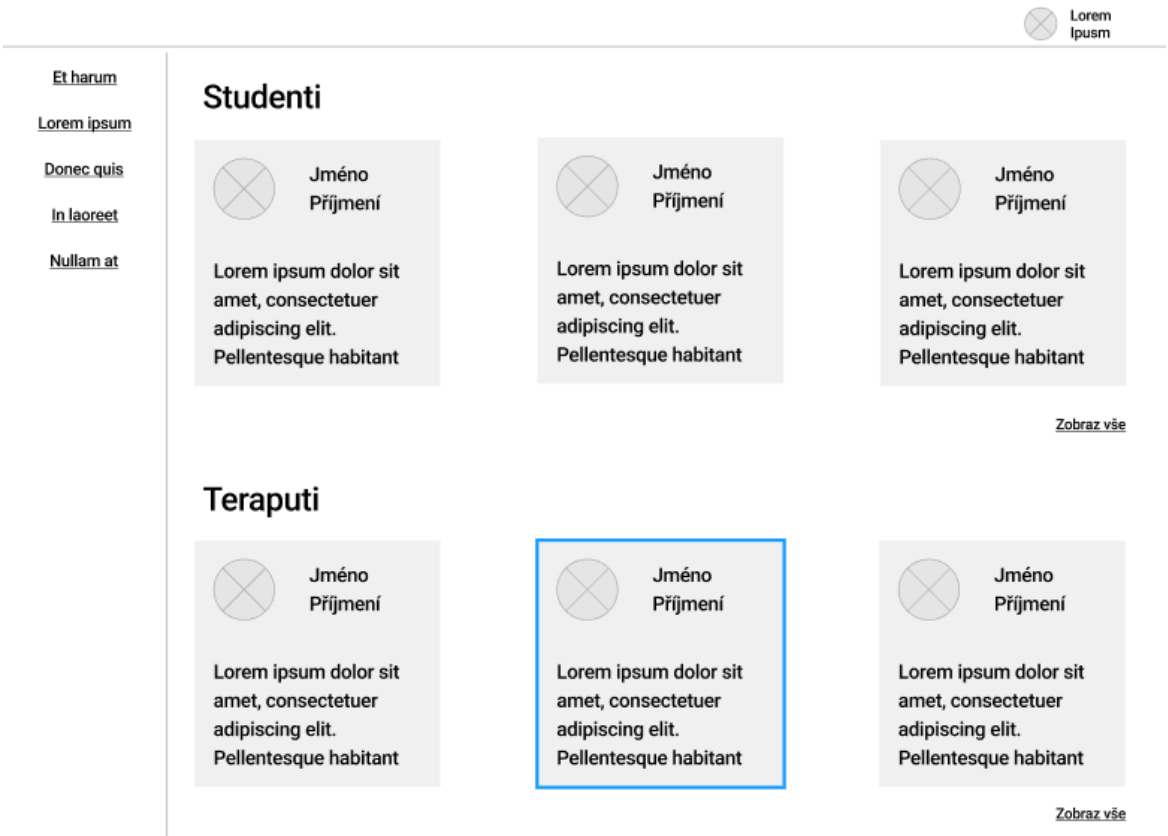
Obrázok 14. Diagram aktivít prihlásenie do systému

10.7 Analýza požiadaviek na užívateľské rozhranie

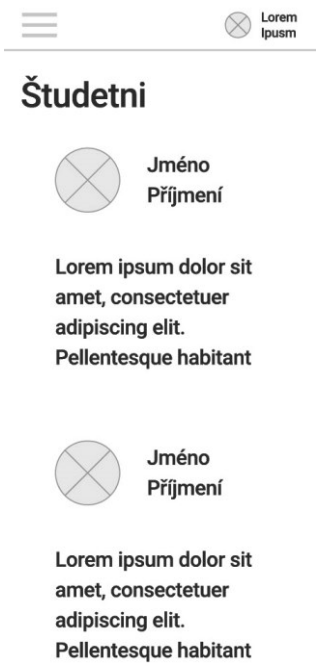
Moderné a vhodné užívateľské rozhranie a užívateľský zážitok je pre terapeutov asi najdôležitejšou vlastnosťou systému. Preto som sa rozhodol pripraviť návrh užívateľského rozhrania v úzkej spolupráci s terapeutmi. Na základe rozhovorov s terapeutmi, pozorovaním jednotlivých terapií a už vytvorených prípadoch použitia bol vypracovaný prototyp,

takzvaný drôtový model (wireframe). Drôtový model slúžil ako základ na tvorbu užívateľského rozhrania a zároveň ukázal budúcim používateľom, ako sa bude s webovou aplikáciou pracovať. Pri vytváraní drôteného modelu vyšli najavo niektoré skutočnosti, ktoré boli chybne alebo nevhodne zachytené v prípadoch použitia. Napríklad v pôvodných prípadoch použitia sa počítalo so zadávaním kritéria ukončenia ku každému cieľu samostatne. Po návrhu drôteného modelu sa zadávanie kritérií ukončenia presunulo o úroveň vyššie a to do celého programu. Takto už teda nebude nutné zadávať u každého cieľa toto kritérium samostatne.

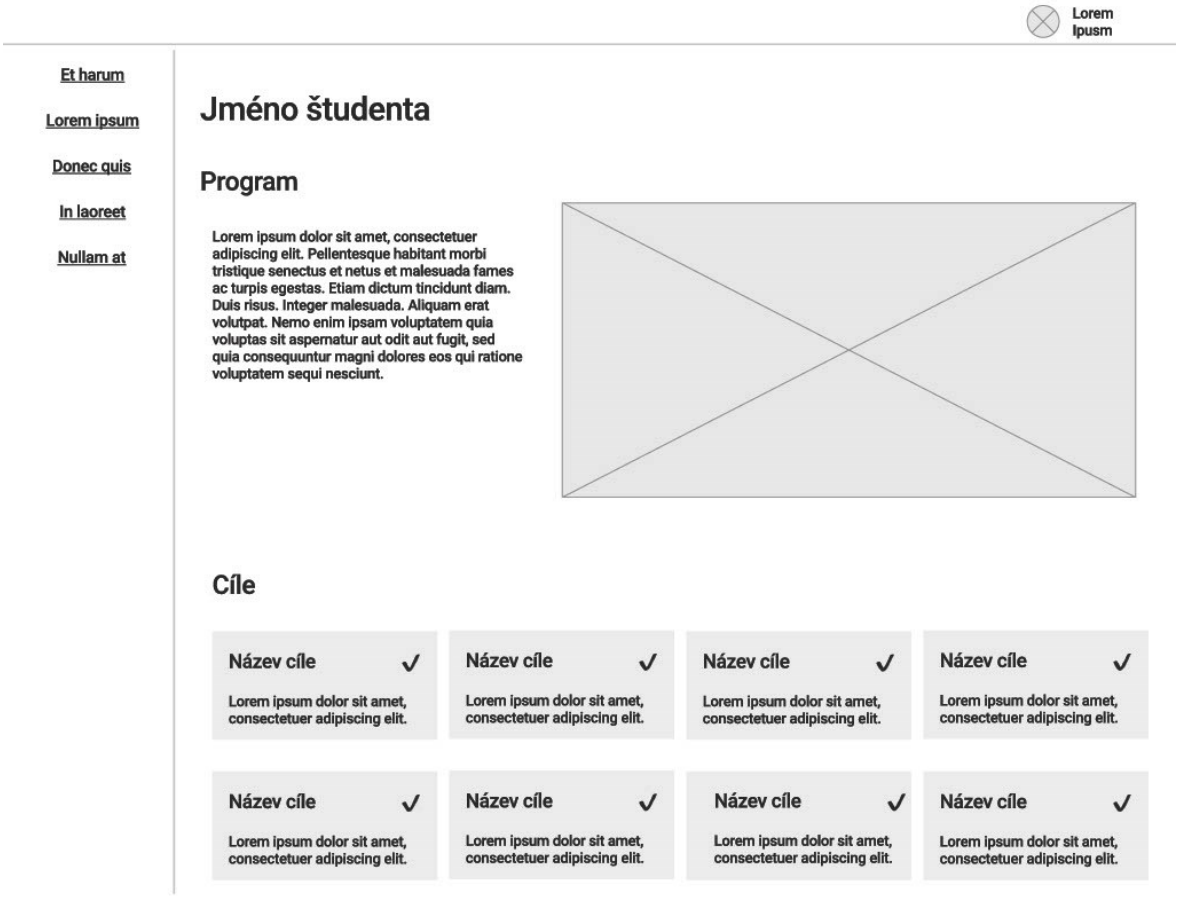
Návrh konkrétnej podoby dizajnu, som kvôli jeho časovej náročnosti nevykonával. Terapeuti nechali konkrétnu podobu systému na mne. Rozhodol som sa tak pri implementácii použiť CSS framework Tailwind. Použitie tohto freameworku umožnilo zrýchliť samotný vývoj. A vo finálnej verzii tak mať ucelený a moderný vzhľad webovej aplikácie.



Obrázok 15. Drôtený model detailu študenta



Obrázok 16. Drôtený model mobilného zobrazenia detailu študenta



Obrázok 17. Drôtený model detailu programu

11 IMPLEMENTÁCIA

Po analyzovaní všetkých požiadaviek som sa rozhodol vypracovať webový informačný systém ako jednostránkovú aplikáciu (SPA). A to hlavne kvôli možnosti lepšieho UX pre používateľov a rýchlejšej dobe načítania stránok. Tieto dve vlastnosti boli pri analýze pre budúcich používateľov webovej aplikácie najdôležitejšie.

Na samotnú implementáciu som sa rozhodol použiť webový stack PERN. Tento stack je na analyzovaný problém vhodnejší než napríklad omnoho populárnejší MERN stack a to kvôli použitiu PostgreSQL. Z návrhu totižto vyplýva, že ukladané dáta budú mať určitú hierarchickú štruktúru. Z toho dôvodu je omnoho vhodnejšie použiť PostgreSQL než Mongo databázu. Je tu taktiež nutné, aby bola zaručená integrita dát. Databáza teda musí spĺňovať takzvané ACID vlastnosti.

Použitím webového stacku PERN vznikne takzvaná jednostránková aplikácia, kde bude frontend a backend aplikácie oddelený. Frontend a backend budú spolu komunikovať pomocou HTTP metód POST, GET, PUT, PATCH a DELETE. Jedná sa v podstate o ekvivalent CURD operácií perzistentných úložísk. Backend zasiela v odpovedi na požiadavku frontendu len JSON objekt. Backend by mal byť čo možno najjednoduchší a mal by slúžiť len ako medzi vrstva na prístup do databáze. O uchovávanie stavu súčasnej relácie sa bude starať frontend a backend bude slúžiť len ako REST API.

Spomínané oddelenie backendu a frontendu umožní v budúcnosti ľahko pridávať novú funkcionalitu alebo nahradiť použité technológie v prípade vylepšovania aplikácie.

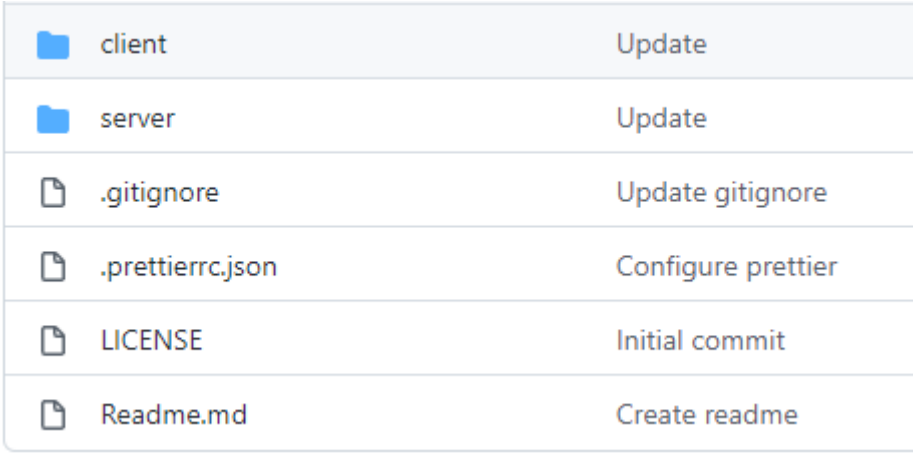
11.1 Vývojové prostredie

Ako vývojové prostredie (IDE) som sa rozhodol používať Visual Studio Code (VS Code) vyvíjané spoločnosťou Microsoft. Toto vývojové prostredie je totiž ideálne na vývoj webových aplikácií. A z môjho subjektívneho pohľadu sa mi spomínané vývojové prostredie zdá byť najlepšie na prácu s programovacím jazykom JavaScript. Pri vývoji bude nutné, aby som aktívne používal terminál, či už k spúšťaniu Node.js projektov alebo k samotnému testovaniu, a preto je potrebné, aby terminál bol súčasťou vývojového prostredia. Je taktiež nutné aby IDE umožňovalo integráciu verzovacích nástrojov. Projekt bude totižto vyvíjaný pomocou verzovacieho nástroja GitHub. VS Code v sebe obsahuje terminál a umožňuje jednoduchú integráciu verzovacích nástrojov, ako napríklad GitHub.

Ďalším dôvodom pre výber VS Code je fakt, že podľa každoročného prieskumu Developer Survey 2021 od StackOverflow je VS Code najpoužívanejšie vývojové prostredie súčasnosti. [29]

11.2 Štruktúra aplikácie

Aplikácia sa skladá z dvoch samostatných zložiek, pričom jedna zložka obsahuje celý frontend a druhá celý backend webovej aplikácie. Obe tieto zložky sú navzájom nezávislé a v prípade potreby môžu byť rozdelené. Každá časť tak napríklad môže byť spracovávaná pomocou rozdielneho verzovacieho nástroja alebo umiestnená na iný hosting. Pri vývoji som sa rozhodol umiestniť obe zložky do jedného verzovacieho nástroja.



Obrázok 18. Štruktúra projektu vo verzovacom nástroji GitHub

11.3 Frontend

Mojím cieľom bolo mať čo možno najviac oddelený frontend a backend aplikácie. Aby v prípade nutnosti bolo možné nahradiť len jednu časť webovej aplikácie a nemusela sa vyvíjať celá aplikácia odznova. V prípade, že frontend a backend spolu komunikujú len skrz REST API, je taktiež jednoduchšie upravovať alebo pridávať novú funkcionalitu. Zároveň je uľahčené napojenie iného systému alebo zariadenia na backend.

Pre samotný frontend aplikácie som sa rozhodoval medzi použitím knižnice React a frameworku Vue. Nakoniec som vybral knižnicu React. A to z toho dôvodu, že knižnica React pracuje len čisto s JSX formátom, na rozdiel od Vue. Vue totižto pracuje so šablónami v formáte HTML a CSS. Mojím cieľom bolo čo najviac zjednodušiť budúci údržbu, aktualizáciu a rozvoj aplikácie. Primiešavanie HTML šablón by kód zbytočne skomplikovalo. Pre

použitie knižnice React som sa taktiež rozhodol z dôvodu väčšieho množstva voľne dostupných knižníc a pluginov. Framework Vue ich má totižto podstatne menej.

11.3.1 React

Inštalácia knižnice React prebieha pomocou takzvaného Node.js package manager (NPM). NPM slúži na správu JavaScriptu balíčkov. Najjednoduchší spôsob ako vytvoriť React aplikáciu je pomocou Node package execute (NPX). NPX umožňuje spúšťanie balíčkov bez nutnosti ich inštalovania. Základná React webová aplikácia sa dá teda vytvoriť napríklad príkazom:

```
$ npx create-react-app my-app
```

Druhou variantnou je postupná inštalácia všetkých potrebných balíčkov ručne, ako napríklad Webpack, Babel, React a ReactDOM. Ja som si zvolil prvú možnosť kvôli jej jednoduchosti a rýchlosti, aj keď to odporuje takzvanej „You Aren't Gonna Need It“ filozofii (YAGNI). YAGNI znamená, že by sme mali implementovať len to, čo skutočne je nutné a aktuálne potrebné.

Pri tvorbe frontendu som sa rozhodol opustiť klasické paradigma objektovo orientovaného programovania. Namiesto toho som použil takzvané funkcionálne programovanie. Finálna React aplikácia sa už neskladá z klasických tried, ale z jednoduchých funkcií. Napríklad každá komponenta je reprezentovaná jednou funkciou. Táto funkcia môže samozrejme prijímať parametre a ako návratovú hodnotu ma React element. Funkcionálne komponenty sprehľadňujú kód. Už nie je nutné používať konštruktor. Funkcionálna komponenta taktiež neobsahuje napríklad funkciu *render*. Pre využitie funkcionálnych komponent som sa rozhodol z dôvodu ich relatívnej jednoduchosti testovania.

Frontend webovej aplikácie obsahuje veľké množstvo rôznych formulárov. Zvolil som si pre to pre samotnú logiku formulárov použiť knižnicu Formik. Formik uľahčuje validáciu formulárov, zobrazovanie chybových správ pri nesprávne vyplnených poliach formulára a zjednodušuje odosielanie formulára. Napríklad nastavenie počiatočných hodnôt pre jednotlivé polia formulára a validácia týchto polí za pomoci knižnice Formik je viacmennej triviálna záležitosť, ako je zrejmé z JavaScript kódu uvedeného nižšie.

```
initialValues={{ email: '', password: '' }}
validate={(values) => {
  const errors = {};
  if (!values.email) {
    errors.email = 'Povinné pole';
  } else if (
    !/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i.test(values.email)
  ) {
    errors.email = 'Špatná emailová adresa';
  }

  if (!values.password) {
    errors.password = 'Povinné pole';
  }
  return errors;
}}
```

Obrázok 19. Kód validácie formulára

11.3.2 Smerovanie

Keďže na úrovni frontendu bolo nutné mať presmerovanie na iné stránky, rozhodol som sa použiť štandardnú smerovaciu knižnicu React Router. Spomínaná knižnica umožňuje jednoduché vytváranie ciest na úrovni frontendu. Inštalácia danej knižnici prebieha klasicky pomocou NPM.

V rámci React Router sa kontroluje či je používateľ overený, má teda správny JWT token. A v prípade, že nie je používateľ overený, je presmerovaný naspať na prihlásenie. V opačnom prípade nastáva presmerovanie na správnu nástenku. Smerovanie pomocou React Router je relatívne jednoduché, ako je vidieť z kódu uvedeného nižšie.

```
<Route
  exact
  path="/"
  element={
    !isAuthenticated ? (
      <Login setAuth={setAuth} />
    ) : (
      <Navigate replace to="/dashboard" />
    )
  }
/>;
```

Obrázek 20. Kód smerovania pomocou React Router

11.3.3 Štýly

Vzhľad webovej aplikácie je pre terapeutov veľmi dôležitý. Preto som sa rozhodol pre použitie CSS frameworku. Okrem toho, že výsledný vzhľad bude mať ucelenú formu, samotný vývoj štýlov bude prebiehať rýchlejšie.

Pri výbere CSS frameworku som sa rozhodoval medzi použitím frameworku Bootstrap a Tailwind. Bootstrap je síce populárnejší, ale Tailwind je viac zameraný na použitie u komponent. Nakoniec som sa vybral framework Tailwind len z toho dôvodu, že sa jedná o novšiu technológiu.

Použitie frameworku Tailwind je veľmi jednoduché. Tento framework je totižto založený na takzvaných utilitách, čo znamená, že jedna trieda predstavuje jednu konkrétnu CSS vlastnosť. Stačí len v k zadaným HTML tagom pridať príslušné triedy.

Vytváraním vlastných tried pre štýly, či už v Tailwind alebo v klasických css súboroch, som sa pokúšal pokiaľ možno vyhnúť. Snažil som sa udržať konzistenciu v štýloch, a tak som používal prevažne triedy z frameworku Tailwind. Nevytváral som teda žiadne nové triedy. Aj keď by to bolo v niektorých prípadoch jednoduchšie a ušetrilo by mi veľké množstvo času. Popísaný prístup som zvolil z toho dôvodu, aby budúce úpravy boli čo možno najjednoduchšie. Toto sprehľadní a uľahčí budúcu údržbu alebo pridávanie nových komponent. Nižšie je uvedený príklad kódu, ako vyzerá požitie tried z frameworku Tailwind u komponenty prihlásenie. Je vidieť, že použitých tried je pomerne dosť.

```
<h1 className="font-medium leading-tight text-5xl mt-0 mb-2 text-blue-600">
  Přihlášení
</h1>
```

Obrázok 21. Použitie tried frameworku Tailwind

11.3.4 Testovanie

Keďže sa celý vývoj realizoval pomocou testami riadeného vývoja, bol pre mňa výber testovacích nástrojov veľmi dôležitý. Na testovanie frontendu som sa rozhodol použiť kombináciu testovacieho frameworku Jest a testovaciu utilitu React Testing Library.

React Testing Library umožňuje testovať skutočný stav obrazovky. Testuje sa tu to, čo skutočne používateľ uvidí. Takýto prístup k testom zvyšuje takzvanú najlepšiu prax (best practice), a tak aj samotnú prístupnosť stránok (Web accessibility).

Tieto testy sú pomerne jednoduché a často sa opakujú. Príklad kódu takého testu je uvedený nižšie.

```
it('render input for email', () => {
  renderLogin();
  const field = screen.getByPlaceholderText('Email');
  expect(field).toBeInTheDocument();
  expect(field.type).toEqual('email');
  expect(field.id).toEqual('email');
});
```

Obrázok 22. Kód testu

Samotné písanie testov som pojal viac ako behaviorálne testy, než ako klasické testy. Tento postup sa mi zdá byť vhodnejší, keďže to umožní spúšťať a vyznať sa v testoch aj netechnicky zdatným osobám. Terapeuti si tak mohli sami overiť, že skutočne systém robí to čo má. Na obrázku 19., je vidieť, že čitateľnosť testu je pomerne dobrá a v tom čo daný test robí, nie je vôbec ťažké sa vyznať.

```
PASS test/components/login/Login.test.js (35.807 s)
  Login
    √ render login form (57 ms)
    √ render heading for form (64 ms)
    √ render input for email (21 ms)
    √ shows label for email input (18 ms)
    √ render input for password (15 ms)
    √ render label for password input (16 ms)
    √ render a login button (40 ms)
    √ has right name in submit button (37 ms)
    √ has link to register page (36 ms)
    √ has link to forgot password page (25 ms)
    √ show error message when email field is empty on submit (67 ms)
    √ show error message when password field is empty on submit (49 ms)
    √ has submit button (17 ms)
    submiting
      √ shows success message on sucessful login (583 ms)
      √ shows error message on unsucessful login (94 ms)

Test Suites: 1 passed, 1 total
Tests:       15 passed, 15 total
```

Obrázok 23. Výstup testu komponenty prihlásenia

Väčší problém nastáva pri testovaní hraníc aplikácie. Tu je nutné simulovať odpovede backendu na HTTP požiadavky. Celý postup spočíva vo vyplnení formulára, odoslaní formulára a simulácie odpovede na požiadavku a následného testovania zobrazenia správnych údajov na obrazovke. Je nutné, aby takýto druh testu bol asynchrónny. Test totižto musí čakať na

odpoveď backendu, aj keď iba simulovanú. V prípade že by sa nejednalo o asynchrónny test, tak by bol test vždy neúspešný. A to z toho dôvodu, že by k vyhodnoteniu testu došlo ešte pred prijatím odpovede. Alebo v horšom prípade by test mohol skončiť falošne pozitívne. Takáto chyba testu sa potom následne ťažko odhaľuje.

11.4 Backend

Backend aplikácie predstavuje REST API. Čo znamená, že som musel nájsť spôsob ako efektívne spracovávať HTTP požiadavky prichádzajúce od frontendu. Nasledujúci spôsob som si vybral preto, aby bolo v budúcnosti možné ľahko frontend nahradiť alebo napojiť novú funkcionalitu k backendu. Na backend webovej aplikácie používam framework Express. Tento minimalistický framework, ktorý využíva behové prostredie Node.js, mi umožnil pomerne rýchlo vytvoriť fungujúci backend. Úlohou backendu v tomto prípade je spracovávanie HTTP požiadaviek, komunikácia s databázou, a odosielanie odpovedí na požiadavky frontendu.

Inštalácia prebieha pomocou NPM a to príkazom:

```
$ npm install express --save
```

Express po inštalácii obsahuje len minimum kódu. Filozofia vývojárov tohto frameworku je totižto zameraná na to, aby vývojár musel čo najviac kódu naprogramovať sám. Vývojár je len minimálne obmedzovaný systémom a má väčšiu voľnosť.

11.4.1 Pripojenie na databázu

Komunikácia frameworku Express a databáze PostgreSQL prebieha pomocou súboru modulov Node-postgres. Túto kolekciu modulov bolo nutné najprv nainštalovať pomocou NPM, a to príkazom:

```
$ npm install pg
```

Po inštalácii som vytvoril modul, ktorý má na starosti napojenie na databázu, tento modul je zobrazený v kóde nižšie. V module som zároveň použil premenné prostredia, ktoré v sebe obsahujú všetky citlivé údaje. Týmto došlo k zabezpečeniu toho že nastavenia databáze a prihlasovacích údajov do databáze budú na jednom mieste. V prípade nutnosti dotazov na databázu potom už iba stačí pridať do JavaScript import s týmto modulom.

```
const Pool = require('pg').Pool;
require('dotenv').config();

const pool = new Pool({
  user: process.env.PG_USER,
  password: process.env.PG_PASSWORD,
  host: process.env.PG_HOST,
  port: process.env.PG_PORT,
  database: process.env.PG_DATABASE,
});

module.exports = pool;
```

Obrázok 24. Kód nastavenie premenných prostredia

Z časových dôvodov som sa rozhodol, že databázu nebudem inštalovať na svoj localhost, ale namiesto toho som použil ľahko netradičný postup. Databázu som priamo nainštaloval na cloudovú platformu Heroku. Táto databáza sa nepoužívala len čisto na vývoj, ale slúžila aj na samotné testovanie pre terapeutov. Ušetril som tak veľké množstvo času, keď v prípade že som chcel nejakú funkcionalitu dať na testovanie terapeutom, stačilo keď som zlúčil vývojovú vetvu vo verzovacom nástroji GitHub s vetvou určenou na akceptačné testovanie. Cloudová platforma Heroku umožňuje takzvané automatické nasadzovanie. Heroku automaticky nasadilo nový kód, ktorý sa následne dal ľahko otestovať, keďže bol prístupný terapeutom skrz internet. Nebolo už nutné pred každým testovaním napríklad premazávať a znova nasadzovať databázu. Terapeuti taktiež neprišli o svoje dáta z predchádzajúceho testovania, ktorým si ušetrili mnoho času a nervov. Terapeuti totižto neboli moc ochotný pred každým testovaním zadávať všetky informácie znova do aplikácie. Niekedy však aj napriek tomu bolo nutné celú databázu premazať a pridať všetky dáta odznova. Napríklad keď sa výrazne menila štruktúra databáze z dôvodu zmeny požiadaviek od terapeutov. Postup so sebou priniesol aj nevýhody, testovanie týmto spôsobom totižto nebolo nezávislé a už existujúce dáta mohli ovplyvňovať výsledok testu. Výhody však vysoko prevyšovali nevýhody, a preto som sa rozhodol pre takýto postup.

Po inštalácii frameworku Express a modulu Node-postgres už bol backend pripravený na samotný vývoj.

11.4.2 Spracovávanie požiadaviek

Pre každý požiadavku na backend bolo následné nutné pripraviť koncové body, takzvané endpoint. V rámci každej iterácie som postupne pridával jednotlivé koncové body. Ako prvé prišli na rad registrácia a prihlasovanie do aplikácie. Následne som vytvoril koncové body pre nástenku, profil študenta a terapeuta, pridávanie zručností, programov, a cieľov.

Každá požiadavka na backend je zachytená frameworkom Express. Táto požiadavka býva, vo väčšine prípadov, najprv spracovaná pomocou takzvaného middlewaru. Middleware kontroluje, či má používateľ oprávnenia pristupovať k danému koncovému bodu. Okrem tohto, napríklad v prípade odosielania formulárov, slúži middleware aj ako kontrola správne vyplnených polí vo formulári. Napríklad v prípade registrácie prebieha kontrola vyplnenia všetky povinných údajov, ale taktiež kontrola správneho tvaru zadaného emailu. V prípade že bude jedna z týchto podmienok splnená tak bude frontendu, ešte v rámci middleware, odoslaný JSON so správou o chybe s chybovým kódom 401. Kód k tomuto prípadu použitia middleware je uvedený nižšie.

```
if (req.path === '/register') {
  if (![email, firstName, lastName, password].every(Boolean)) {
    return res.status(401).json('Chybějící údaje');
  } else if (!validEmail(email)) {
    return res.status(401).json('Neplatný e-mail');
  }
}
```

Obrázok 25. Príklad použitia middleware

11.4.3 Autorizácia

Autorizáciu má na starosti middleware frameworku Express. Na autorizáciu existuje veľké množstvo modulov pre Express. Ja som sa však rozhodol žiadny modul nepoužiť a namiesto toho spraviť celú autorizáciu sám. Použitie modulov síce urýchľuje vývoj, ale zároveň znižuje nezávislosť kódu. A tomu som sa ja pokúšal čo možno najviac vyhnúť. V prípade autorizácie som chcel mať plnú kontrolu nad tým, ako bude webová aplikácia fungovať.

K autorizácii som sa rozhodol použiť technológiu JSON Web Token (JWT). A to hneď z niekoľkých dôvodov. Prvým dôvodom je to, že sa jedná o bezstavový spôsob autentifikácie. To znamená, že v základnej verzii JWT nie je nutné zapisovať nič do databáze. Tým sa

podstatne znižuje náročnosť vývoja. V prípade nutnosti je ale možné funkcionalitu JWT rozšíriť tak aby pracoval s databázou. Použitím JWT sa presúva časť logiky autorizácie zo strany servera na stranu klienta. Čo je plne v súlade s filozofiou single page aplikácií. Tento typ autorizácie je možné v prípade potreby v budúcnosti ľahko rozšíriť a zvýšiť tak bezpečnosť.

Počas prihlásenia do webovej aplikácie Express skontroluje zhodu prihlasovacieho emailu a hashu hesla. V prípade, že sa jedná o zhodu, backend vygeneruje nový JWT na základe UUID používateľa a zadaného tajomstva. Takto vygenerovaný token je následne poslaný naspať ako odpoveď na požiadavku frontendu. Doba životnosti vygenerovaného tokenu je 1 hodina. Príklad kódu generovania a zaslania tokenu v frameworku Express:

```
const token = jwtGenerator(user.rows[0].user_id);
res.json({ token })
```

Obrázok 26. Generovanie JWT

Životnosť tokenu sa predlžuje, respektíve je vygenerovaný nový token v prípade, že terapeut alebo supervízor spustil sedenie. A to z toho dôvodu, aby sa nestala situácia kedy bude používateľ počas sedenia odhlásený z aplikácie.

Na autorizáciu a autentifikáciu sú v backende aplikácie definované tri koncové body. Tieto body sú znázornené v tabuľke nižšie.

Tabuľka 7. Koncové body pre autorizáciu a autentifikáciu

HTTP metóda	Cesta	Popis
POST	/auth/register	Registrácia nového supervízora.
POST	/auth/login	Prihlásenie do aplikácie.
GET	/auth/verify	Autorizácia používateľa

Koncový bod *verify* slúži na kontrolu tokenu, ktorý posiela frontend v hlavičke požiadavku. V prípade chýbajúceho alebo nesprávneho tokenu vráti framework Express odpoveď na požiadavku so statusom 403 a hlásením o chybe. Táto funkcionalita je znázornená v kóde uvedenom nižšie.

```
router.get('/verify', authorization, (req, res) => {
  try {
    res.status(200).json(true);
  } catch (err) {
    res.status(403).send('Chyba serveru');
  }
});
```

Obrázok 27. Kód koncového bodu *verify*

12 TESTOVANIE

K testovaniu webovej aplikácie som sa rozhodol použiť len unit testy a následne až akceptačné testovanie používateľmi. Integračné a systémové testy som sa z časových dôvodov vynechal. K vývoju frontendovej časti aplikácie som použil testami riadený vývoj, čo znamená že na konci každej iterácie už nebolo nutné vytvárať žiadne ďalšie testy. Pri tvorbe backendovej časti som však tento postup musel z časových dôvodov opustiť. Keďže som celý systém vyvíjal agilnou metodikou, na konci každého šprintu bolo možné sledovať pokrok vo vývoji aplikácie na základe úspešných akceptačných testov. Výsledkom každej iterácie bola dokončená komponenta, ktorá bola otestovaná používateľmi. Pre takýto postup som sa rozhodol pre to, aby som minimalizoval množstvo chýb ktoré používatelia nájdu u finálneho beta testovania. Takto bolo tiež možné lepšie reagovať na často sa meniace požiadavky na systém.

13 NASADENIE

Pôvodne mala byť celá webová aplikácia nasadená na hostingu lokalizovanom v Českej republike. Bohužiaľ sa mi nepodarilo nájsť žiadny vhodný hosting, ktorý by splňoval všetky nároky, ktoré som si kládol. Nakoniec mi neostávalo nič iné, než sa pohliadnuť po zahraničných hostingových službách. Z možností som výber zúžil na troch hlavných kandidátov: Heroku, Vercel a Netlify. Na začiatku som mal v pláne, aby frontend a backend fungovali každý na inom hostingu, z dôvodu diverzifikácie zdrojov a zvýšenia bezpečnosti. Preto mi v užšom výbere skončili hneď traja kandidáti na hosting. Po dlhom zvažovaní som sa rozhodol, že najvhodnejšia platforma bude Heroku. Táto cloudová služba umožňuje ľahko pridať, v rámci takzvaných doplnkov, databázu PostgreSQL. Systém akým Heroku vypočítava poplatok za svoje služby sa mi taktiež zdal byť vhodnejší. Nepracuje sa tu totižto s fixným mesačným poplatkom, ale s takzvanými dyno hodinami. Tieto hodiny predstavujú čas, ktorý systém skutočne pracoval. V Heroku sa platí len za čas, v ktorom systém skutočne niečo vykonáva. V prípade, že na systém neprichádzajú žiadne požiadavky, je v celý systém v režime spánku. Každý používateľ Heroku má určité množstvo týchto hodín zdarma a je možné minimalizovať cenu za hosting. Na fungovanie webovej aplikácie na Heroku taktiež nie je nutné registrovať žiadnu doménu, čo znižuje náklady.

Celý systém som sa na základe analyzovaných požiadaviek a dlhej úvahy, rozhodol nasadiť na cloudovú platformu Heroku. Pričom backend a frontend sú oddelené a každá časť má svoju vlastnú URL adresu. Platforma Heroku slúžila taktiež aj ako prostredie pre akceptačné testovanie.

Samotné nasadenie na Heroku je v podstate veľmi jednoduchý proces. Heroku totižto umožňuje prepojenie s verzovacím systémom GitHub, a tým pádom aj automatické nasadzovanie. Celý vývoj prebiehal tak, že všetky zmeny boli v GitHube vykonané vo vývojovej vetvy a v prípade potreby testovania funkcionality používateľmi stačilo túto vetvu spojiť s hlavnou vetvou a Heroku už automaticky nasadilo nové zmeny. Tieto zmeny boli následne prístupné pre používateľov na testovanie. Heroku totižto každej aplikácii generuje vlastnú URL adresu a je možné k vytvorenej webovej aplikácii bez problému pristupovať prostredníctvom internetu.

Heroku taktiež poskytuje možnosť jednoduchého pridania vlastnej domény. Frontend aplikácie tak môže mať v prípade nutnosti českú doménu, ale backend môže ďalej fungovať len

v rámci Heroku subdomény. Z hľadiska bezpečnosti a diverzifikácie je spomínaný postup asi najlepší.

14 ÚDRŽBA

Samotná aplikácia sa skladá z veľkého množstva open-source modulov a je nutné priebežne všetky tieto moduly aktualizovať. Toto je často zdĺhavý a náročný proces. Jednotlivé moduly totižto často závisia na iných moduloch, a tie zase závisia na iných moduloch atď. Nie je pre to vylúčené, že postupom času nastane situácia, kedy bude nutné nejaký ten modul z aplikácie z bezpečnostných dôvodov odstrániť a nahradiť ho novým modulom. Celý systém je na podobné situácie ale vcelku dobre pripravený a nemalo by byť zložité moduly nahradzovať.

GitHub umožňuje automatickú aktualizáciu závislostí pomocou takzvaného Dependabota. Ten priebežne kontroluje a aktualizuje všetky závislosti v projekte. V prípade vzniku nejakej bezpečnostnej hrozby z dôvodu zlej závislosti, posiela bot emaily so vzniknutou hrozbou a možnosťou nápravy v prípade, že nejaká možnosť nápravy existuje. Týmto sa celý proces údržby automatizuje a je nutné zasahovať len u tých incidentov, ktoré Dependabot nedokáže vyriešiť sám. Potrebná údržba implementovanej webovej aplikácie je minimálna.

15 VYHODNOTENIE

Hlavným cieľom práce bolo vytvoriť a implementovať systém, ktorý zjednoduší a zefektívni prácu terapeutom, a zároveň podporí aktuálnosť zaznamenávaných dát. Do porovnania bola zahrnutá vytvorený prototyp webovej aplikácie a najčastejšie využívaná metóda tužka-papier. Oba tieto postupy overovali dva rôzni terapeuti v priebehu niekoľkých sedení u dvoch rôznych klientov. Pričom ich programy obsahujú prevažne základnú úroveň zbieraných dát. Konkrétne sa jedná o dáta zamerané na záznam podmienky splnené alebo nesplnené. Analýza efektivity aplikácie oproti štandardnej metóde tužka-papier je porovnávaná s ohľadom na čas strávený zápisom a zberom dát, a tiež presnosťou zbieraných dát. V rámci vyhodnotenia prebieha diskusia s terapeutmi zameraná na ich užívateľský zážitok s aplikáciou.

V priebehu dvoch týždňov boli zaznamenané dáta metódou tužka-papier. A to normálnym spôsobom, tak ako sú terapeuti zvyknutý pracovať behom sedení. Merania sa uskutočňovali iba na sedeniach, kde bol prítomný behaviorálny analytik. Tento analytik meral čas spojený s zbieraním dát. Od zápisu jednotlivých cieľov a potrebných informácii do týždenných záznamových listov, doplňovanie nových cieľov a zaznačovanie zvládnutia cieľov. A v neposlednej rade vyhodnotenie študentovej aktuálnej úrovne v jednotlivých cieľoch, následný prepis do knihy študenta obsahujúci dlhodobejšie záznamy a grafické znázornenie dát. Pri práci mal terapeut všetky potrebné záznamové listy na klipsovej doske. Túto dosku mal odloženú na určitom mieste. Charakter práce totižto neumožňoval, aby si terapeut nosil celú zložku so sebou. Činnosti sú totižto často pohybovo náročné. Terapeut sa behom sedení vracal k zápisu dát, pričom čas strávený zápisom dát sa počítal do celkového vymedzeného času. Ku konci sedenia terapeut dopísal všetky dáta z jednotlivých záznamových listov do knihy dát. Nastáva tu teda dvojitý zápis dát a to pre účely terapie na individuálnych listoch a pre účely analytického zhodnotenia v záveru sedenia.

V nasledujúcej tabuľke vidíme, že sa celkový čas strávený záznamom dát metódou tužka-papier pohyboval v rozmedzí 11-20 minút na študenta A a v rozmedzí 19 až 22 minút na študenta B. Nedá sa určiť presný čas, keďže sa tento čas odvíja od počtu a zložitosti konkrétneho programu študenta. Aj keď by sa všetky dáta mali zaznamenávať a vizualizovať denne, u niektorých položiek pristupuje terapeut k zjednodušeniu zápisu a dáta spracuje až na konci týždňa. Na konci týždňa teda nastáva nárast potrebného času na zápis dát ku koncu sedenia, ktorý terapeut nemôže venovať študentovi. Rýchlosť záznamu dát sa u terapeutov

líši v rádoch sekúnd a nebol pre účely porovnania nijako významný. Aplikácia bola vytvorená tak, aby nebolo nutné časovo náročné zaškolenie v jej používaní.

Tabuľka 8. Čas potrebný na zápis dát pomocou metódy tužka-papier

Druh záznamu	Študent A	Študent B
Vypísanie cieľov do týždenného, prípadne denného záznamu.	3-5 minút	4-6 minút
Záznam úrovne jednotlivých dát.	5-10 sekúnd krát počet zápisov. Najčastejšie 10-16 položiek.	8-12 sekúnd
Zápis do knihy na konci sedenia.	3-5 minút	5-8 minút
Zápis do knihy na konci týždňa.	5-10 minút	10-15 minút
Celkový čas.	11-20 minút	19-29 minút

Pri použití webovej aplikácie v priebehu ďalšieho týždňa došlo k výraznému obmedzeniu času stráveného záznamom na konci sedenia, a predovšetkým na konci týždňa. V aplikácii totižto dochádza k automatickému prepisu do virtuálnej knihy študenta, a zároveň k automatickému generovaniu grafického spracovania dát. Úvodné vypisovanie cieľov je taktiež výrazne obmedzené, zvládnuté ciele sa totižto automaticky odstraňujú a terapeut môže okamžite zaradiť nový ciel, ktorý sa prepisuje do nasledujúceho týždňa, pokiaľ nie je zvládnutý behom učenia. Úsporu času znázorňuje tabuľka nižšie.

Tabuľka 9. Čas potrebný na zápis dát pomocou implementovanej aplikácie

Druh záznamu	Študent A	Študent B
Vypísanie cieľov do týždenného, prípadne denného záznamu.	0-2 minút	0-3 minúty
Záznam úrovne jednotlivých dát.	5-15 sekúnd krát počet zápisov. Najčastejšie 10-16 položiek.	8-15 sekúnd
Zápis do knihy na konci sedenia.	-	-
Zápis do knihy na konci týždňa.	-	-
Celkový čas.	0-3 minúty	0-4 minúty

Čo sa týka efektivity, aplikácia má lepšie výsledky s ohľadom na ušetrený čas. Zároveň z výpovedí terapeutov sa dozvedáme ďalšie výhody aplikácie oproti metóde tužka-papier:

„V aplikaci můžu snadno zobrazit aktuální cíle, není potřeba nic vypisovat, stačí si vygenerovat novou relaci pro sezení. Velmi se mi líbí, že jakmile jsou potřebná data zanesena do aplikace, cíl se už nezobrazuje."

„Aplikace si sama tvoří grafy, což velmi usnadňuje práci a také se nestane, že by byla data zapsaná špatně – samo si to spočítá, zda už je cíl zvládnut nebo ne. Některá data stále musíme zapisovat na papír, jelikož v aplikaci zatím nejsou všechny možnosti, které by se nám v práci hodily."

„Vyhovuje mi, že nemusím běhat pro desky s papíry, nestane se, že některý záznamový list chybí (není vytištěn), ale můžu velmi rychle vše zaznamenat do telefonu nebo tabletu."

„S webem se snížil počet dní, ve kterých nejsou připraveny veškeré materiály pro práci předem. Analytik může ihned vidět, zda je vše na sezení nachystáno, a pokud není, urguje k nápravě."

„Během analýzy oceňuji, že mám okamžitý přístup k datům, které mi terapeuti zaznamenávají. I přes krátkou dobu používá je patrné, že dochází k menšímu počtu chyb ze strany terapeutů při záznamu."

16 FINÁLNY STAV WEBOVEJ APLIKÁCIE

Aplikácia sa v súčasnosti nachádza v stave prvotného prototypu. V aplikácii je možné vytvárať nových supervízorov a terapeutov, ale zatiaľ chýba možnosť vytvárania profilov pre rodičov. Aplikácia sa tak dá plnohodnotne používať na zber základných dát. Bolo by však vhodné aplikáciu rozšíriť o možnosť pridávania ďalších druhov dát, ktoré boli popísané v teoretickej časti práce. Poprípade pridať do aplikácie možnosť nadefinovania vlastných druhov dát. Aplikácia taktiež zatiaľ generuje len základný typ grafu, a to kumulatívny graf. Tento druh grafu plne vyhovuje pri zbieraní základných dát, avšak z používateľského hľadiska by bolo dobré mať možnosť generovať viac druhov grafov.

Každú chýbajúcu funkcionalitu je možné pomerne jednoducho do aplikácie pridať. Pridanie novej funkcionality totižto spočíva len vo vytvorení nového koncového bodu a malej úprave už existujúceho kódu. Aj keď je prototyp aplikácie stále vo fázy vývoja, používatelia oceňovali značné zjednodušenie práce už v základnej verzii. Samozrejme zber dát pomocou aplikácie mohol prebiehať iba u programov, ktoré nevyžadujú odlišné metódy merania. Pre tie je nutné stále používať papierovú formu.

17 ĎALŠÍ VÝVOJ

Výsledná webová aplikácia je len prototypom, ktorý je ale možné v budúcnu jednoducho rozširovať o novú funkcionalitu. Ďalší vývoj sa môže zamerať napríklad na pridanie systém logovania jednotlivých meraní. Aby supervízor mal prehľad nad tým, kto a v akom čase zapisoval alebo menil dáta študentov. Je možné taktiež vylepšiť samotnú správu používateľských účtov, napríklad pridaním možnosti odstránenia používateľského účtu.

Možnosťou ako vylepšiť samotný frontend a trochu zvýšiť používateľský komfort môže byť napríklad pridanie prepínania medzi motívom aplikácie. V súčasnosti aplikácia podporuje iba svetlý motív, čo nemusí každému vyhovovať. Bolo by vhodné v budúcnosti zapracovať ešte tmavý motív. Táto úprava by nemala zabrať veľa času, keďže aplikácia je na pridanie tmavého motívu pripravená. Stačí už len pridať správne Tailwind triedy. Bolo by taktiež vhodné pridať možnosť ľahkého prekladu celej aplikácie. V súčasnosti je aplikácia totižto len v českom jazyku.

Čo sa týka technickej stránky, samotné testy v časti frontendu by mohli využívať namiesto metód *fireEvent* knižnicu *@testing-library/user-event*. Táto nová knižnica totižto ešte lepšie reprezentuje, ako používateľ skutočne pracuje s aplikáciou.

V rámci zvýšenia bezpečnosti webovej aplikácie je nutné do budúcna vyriešiť napríklad vhodné dvojfázové prihlásenie, či už za použitia rôznych bezpečnostných tokenov alebo za použitia špeciálnych mobilných aplikácií.

ZÁVĚR

Cieľom predloženej diplomovej práce bolo analyzovať, navrhnúť a implementovať webový informačný systém, ktorý by terapeutom aplikovanej behaviorálnej analýzy uľahčil zber a spracovanie dát.

V rámci teoretickej časti bol najprv popísal druhy dát zbieraných počas terapií a legislatíva, ktorá sa vzťahuje na behaviorálnych analytikov. Následne sú v práci konkretizované vhodné webové technológie, ktoré boli použité pri vývoji webovej aplikácii. Tento popis môže v budúcnosti slúžiť vývojárom pri výbere vhodných webových technológií.

Praktickej časti diplomovej práci sa venuje popisu metodiky vývoja. Veľkú pozornosť smeruje k analýze požiadaviek a návrhu samotnej webovej aplikácie. Táto analýza a návrh môžu v budúcnosti slúžiť pri návrhu a vývoji ďalších podporných systémov aplikovanej behaviorálnej analýzy. Práca poskytuje v českom prostredí jedinečný systém pre praktickú aplikáciu behaviorálnych terapií s využitím najnovších technológií.

Implementácia webovej aplikácie je v súčasnosti vo fázy základného prototypu, tento prototyp sa dá používať, avšak druh zbieraných dát je stále silne obmedzený na oblasť tých najnutnejších. Väčšina funkcionálnych a nefunkcionálnych požiadaviek terapeutov na aplikáciu však bola splnená. A v rámci spätnej väzby bol prototyp aplikácie veľmi kladne hodnotený.

Táto diplomová práca môže slúžiť ako základ pre budúci vývoj produktov pre aplikovanú behaviorálnu analýzu. Výsledný prototyp webovej aplikácie je možné jednoducho rozširovať o novú funkcionalitu alebo zariadenia, napríklad o inteligentné hodinky.

SEZNAM POUŽITÉ LITERATURY

[1] COOPER, John, Timothy HERON a William HEWARD. Applied Behavior Analysis. 2nd edition. London: Pearson Education, 2007. ISBN 978-0-131-42113-4.

[2] CASEY, Laura B. a Stacy L. CARTER. Applied Behavior Analysis in Early Childhood Education. New York: Routledge, 2016. ISBN 978-1-138-02511-0.

[3] FISHER, Wayne, Cathleen PIAZZA a Henry ROANE, ed. Handbook of Applied Behavior Analysis. 2nd ed. New York: The Guilford Press, 2021. ISBN 978-1-462-54375-5.

[4] MATSON, Johnny, ed. Applied Behavior Analysis for Children with Autism Spectrum Disorders. New York (USA): Springer, 2009. ISBN 978-1-441-90087-6.

[5] VOZÁKOVÁ, Lucie. Behaviorálna analytička [ústní sdělení]. Brno, 6.4.2022.

[6] Legislativa: Proces legitimizace aplikované behaviorální analýzy – ABA v České republice. CSABA [online]. [cit. 2022-04-12]. Dostupné z: https://csaba.cz/legislativa/

[7] Vyhláška č. 98/2012 Sb.: Vyhláška o zdravotnické dokumentaci. Zákony pro lidi: Sbírka zákonů ČR v aktuálním konsolidovaném znění [online]. [cit. 2022-04-12]. Dostupné z: https://www.zakonyprolidi.cz/cs/2012-98

[8] VOZÁKOVÁ, Lucie. Behaviorálna analytička [ústní sdělení]. Brno, 12.4.2022.

[9] DHADUK, Hiren. An Ultimate Guide to Web Application Architecture. Simform [online]. [cit. 2022-04-18]. Dostupné z: https://www.simform.com/blog/web-application-architecture/

[10] SATERNOS, Casimir. Client-Server Web Apps with JavaScript and Java: Rich, Scalable, and RESTful. Sebastopol (California): O'Reilly Media, 2014. ISBN 978-1-449-36933-0.

[11] Single Page Application (SPA) vs Multi Page Application (MPA: Two Development Approaches. ASPER BROTHERS [online]. [cit. 2022-03-15]. Dostupné z: https://asperbrothers.com/blog/spa-vs-mpa/

[12] SCOTT, Emmit. SPA design and architecture: understanding single-page web applications. Shelter Island: Manning, 2016. ISBN 978-1-617-29243-9.

[13] GILLIS, Alexander. What is native app. TechTarget [online]. 2020 [cit. 2022-03-28]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/native-application-native-app

[14] WARGO, John. Learning Progressive Web Apps. Boston: Addison-Wesley Professional, 2020. ISBN 978-0136484226.

[15] SBARSKI, Peter, Yan CUI a Ajay NAIR. Serverless Architectures on AWS. 2nd Edition. Shelter Island: Manning, 2020. ISBN 978-1617295423.

[16] ROCHA, Hugo. Practical Event-Driven Microservices Architecture: Building Sustainable and Highly Scalable Event-Driven. New York: Apress, 2021. ISBN 978-1-4842-7468-2.

[17] MYERS, Dominic. Front-End Developer: BCS Guides to IT Roles. Swindon: BCS, 2020. ISBN 978-1-78017-4778.

[18] FERGUSON, Nicole. What's the Difference Between Frontend and Backend Web Development. CareerFoundry [online]. December 27, 2021 [cit. 2022-04-17]. Dostupné z: https://careerfoundry.com/en/blog/web-development/whats-the-difference-between-frontend-and-backend/

[19] BESTAIEVA, Diana. Web Applications Architectures: Components, Layers, and Types. Cleveroad [online]. 2021 [cit. 2022-04-13]. Dostupné z: https://www.cleveroad.com/blog/web-application-architecture

[20] SUBRAMANIAN, Vasan. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. Second Edition. Bangalore (India): Apress, 2019. ISBN 978-1-4842-4391-6.

[21] Difference between PERN and MERN stack. GeeksforGeeks: A computer science portal for geeks [online]. 2021 [cit. 2022-04-23]. Dostupné z: https://www.geeksforgeeks.org/difference-between-pern-and-mern-stack/

[22] RIGGS, Simon, Gianni CIOLLI a Sudheer MEESALA. PostgreSQL 14 Administration Cookbook. Sixth Edition. Birmingham: Packt Publishing, 2022. ISBN 9781803248974.

[23] HA LE, Quan a Marcelo DIAZ. Developing Modern Database Applications with PostgreSQL. Birmingham: Packt Publishing, 2021. ISBN 978-1-838-64814-5.

[24] VAIDYA, Madhavi. RDBMS In-Depth: Mastering SQL and PL/SQL Concepts, Database Design, ACID Transactions, and Practice Real Implementation of RDBM. India: BPB, 2021. ISBN 978-8194837701.

[25] LIM, Greg. Beginning Node.js, Express & MongoDB Development. Independently published, 2019. ISBN 978-1078379557.

[26] BANKS, Alex a Eve PORCELLO. Learning React: Modern Patterns for Developing React Apps. Second Edition. Sebastopol, CA: O'Reilly, 2020. ISBN 978-1-492-05172-5.

[27] BUGL, Daniel. Learn React Hooks: Build and refactor modern React.js applications using Hooks. Birmingham: Packt Publishing, 2019. ISBN 978-1838641443.

[28] ZAMMETTI, Frank. Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker. New York: Apress, 2020. ISBN 978-1-4842-5737-1.

[29] Stack Overflow Developer Survey 2021. Stack Overflow [online]. [cit. 2022-03-18]. Dostupné z: https://insights.stackoverflow.com/survey/2021

[30] KEREKI, Federico. Mastering JavaScript Functional Programming: Write clean, robust, and maintainable web and server code using functional JavaScript. 2nd Edition. Birmingham: Packt Publishing, 2020. ISBN 978-1839213069.

[31] SALCESCU, Cristian. Functional Architecture with React and Redux. London: Amazon, 2020. ISBN 979-8607681111.

[32] SALCESCU, Christian. Discover Functional JavaScript: An overview of Functional and Object Oriented Programming in JavaScript. London: Amazon, 2019. ISBN 978-1095338780.

[33] Catalyst Client. In: Google Play [online]. [cit. 2022-04-10]. Dostupné z: https://play.google.com/store/apps/details?id=com.datafinch.catalyst

[34] ABA Data Collection. In: Google Play [online]. [cit. 2022-04-10]. Dostupné z: https://play.google.com/store/apps/details?id=com.accupointmed.bigdatamobile_droid

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ABA	Applied behavior analysis – aplikovaná behaviorálna analýza.
ABC	Antecedent, chovanie a následok.
PAS	Porucha autistického spektra.
MPA	Multi Page Application – viacstránkové aplikácie
SPA	Single Page Application – jednostránkové aplikácie
HTTP	Hypertext Transfer Protocol – internetový protokol určený ku komunikácii s webovými servermi
SEO	Search engine optimization – optimalizácia pre vyhľadávače
AJAX	Asynchronous JavaScript and XML – označenie technológií pre vývoj interaktívnych webových aplikácií
CSS	Cascading Style Sheets – kaskádové štýly
DOM	Document Object Model – objektový model dokumentu
CSSOM	CSS Object Model – objektový model kaskádových štýlov
UX	User Experience – užívateľský zážitok
UI	User Interface – užívateľské rozhranie
URL	Uniform Resource Locator – jednotný vyhľadávač zdrojov
PWA	Progressive Web Apps – progresívne webové aplikácie
API	Application Programming Interface – rozhranie pre programovanie aplikácií
HTML	Hypertext Markup Language – hypertextový značkovací jazyk
XML	Extensible Markup Language – rozšíriteľný značkovací jazyk
CURD	Create, read, update, and delete – štyri základné operácie používané pri práci s dátami
LAMP	Linux, Apache, MySQL, PHP
MEAN	MongoDB, ExpressJS, Angular, Node.js – webový stack
MERN	MongoDB, ExpressJS, React, Node.js – webový stack

PERN	PostgreSQL, ExpressJS, React, Node.js – webový stack
ACID	Atomicity, Consistency, Isolation, and Durability – štyri žiadúce vlastnosti databáze
SQL	Structured Query Language – štandardizovaný štruktúrovaný dopytovací jazyk
TDD	Test-driven development – testami riadený vývoj
PITR	Point-in-time recovery – metóda zálohovania a obnovy databáze
SSL	Secure Sockets Layer – vrstva bezpečných soketov
RLS	Row level security – bezpečnostná funkcia PostgreSQL
MVC	Model-view-controller – softvérová architektúra
I/O	Input/Output – vstup/výstup
CASE	Computer Aided Software Engineering – počítačom podporované softwarové inžinierstvo
CDN	Content delivery network – sieť slúžiaca na doručovanie obsahu
POST	Metóda HTTP protokolu slúžiaca k načítaniu dát zo serveru
GET	Metóda HTTP protokolu slúžiaci k načítaniu dát zo serveru
PUT	Metóda HTTP protokolu slúžiaca na vytvorenie nového zdroja, alebo aktualizáciu už existujúceho zdroja na serveru
PATCH	Metóda HTTP protokolu slúžiaca k čiastočnej aktualizácii zdroja na serveru
DELETE	Metóda HTTP protokolu slúžiaca na odstránenie zdroja zo serveru
REST API	Representational State Transfer API – rozhranie pre programovanie aplikácií slúžiace k prístupu k zdrojom pomocou HTTP protokolu
IDE	Integrated Development Environment – vývojové prostredie
VS Code	Visual Studio Code – vývojové prostredie vyvíjané spoločnosťou Microsoft
JSX	JavaScript Syntax Extension – rozšírenie syntaxe jazyka JavaScript
NPM	Node.js package manager – správca balíčkov pre JavaScript
NPX	Node package execute – nástroj pre beh balíčkov

YAGNI	You Aren't Gonna Need It – filozofia používaná pri vývoji softvéru
JWT	JSON Web Token – spôsob pre bezpečnú výmenu dát

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Obrázky použité v práci.

Projekt v programe Enterprise Architect obsahujúci analýzu.

Drôtený model.

Zdrojový kód webovej aplikácie obsahujúci postup inštalácie.