VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2019 Daniel Rusnák

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE KLINICKÉHO VYŠETŘENÍ PACIENTŮ PRO SYSTÉM ANDROID
ANDROID APPLICATION FOR CLINICAL ASSESSMENT OF PATIENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Daniel Rusnák
AUTHOR

VEDOUCÍ PRÁCE Ing. Jiří Mekyska, Ph.D.
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**
Ústav telekomunikací

Student: Daniel Rusnák ***ID:*** 173736
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Aplikace klinického vyšetření pacientů pro systém Android

POKYNY PRO VYPRACOVÁNÍ:

V rámci bakalářské práce bude navržena a implementována aplikace, která bude klinikami využívána k diagnóze a hodnocení různých onemocnění. Aplikace umožní vkládat jednotlivé klinické testy, popř. diagnostické postupy formou modulů. V testech bude probíhat výpočet skórů dle vzorců, které autor testu předem definuje. Do aplikace bude navržen a implementován zabezpečený přístup, šifrování a export dat (do Excelu nebo PDF).

DOPORUČENÁ LITERATURA:

[1] SMYTH, Neil. Android Studio Development Essentials - Android 7 Edition: Learn to Develop Android 7 Apps with Android Studio 2.2. 1. CreateSpace Independent Publishing Platform, 2016. ISBN 978-1535425339.

[2] DARWIN, Ian. Java Cookbook: Solutions and Examples for Java Developers. 3. O'Reilly Media, 2014. ISBN 978-1449337049.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: Ing. Jiří Mekyska, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Táto bakalárska práca sa venuje návrhu a implementácii systému na digitalizovanie procesu klinického testovania pacientov s neuroligickými ochoreniami formou testovacích dotazníkov. Systém sa skladá z troch aplikácií, ktoré spolu tvoria funkčný systém a pomôžu doktorom z efektívniť a úrychliť ich prácu.

KĽÚČOVÉ SLOVÁ

Java, Android, Angular, NGINX, webová bezpečnosť, architektúra systému,

ABSTRACT

This bachelor thesis deals with the design and implementation of a system for digitizing the process of clinical testing of patients with neurological diseases in the form of test questionnaires. The system consists of three applications that together form a functional system to help doctors work efficiently and speed up their work.

KEYWORDS

Java, Angular, Android, NGINX, web security, system architecture

RUSNÁK, Daniel. *Aplikace klinického vyšetření pacientů pro systém Android*. Brno, 2019, 62 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Jiří Mekyska, Ph.D.

Vysázeno pomocí balíčku `thesis` verze 2.63; `http://latex.feec.vutbr.cz`

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Aplikace klinického vyšetření pacientů pro systém Android“ vypracoval(a) samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor(ka) uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil(a) autorské práva tretích osôb, najmä som nezasiahol(-la) nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý(-á) následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

POĎAKOVANIE

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Jiřímu Mekyskovi , Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v tejto bakalárskej práci bol realizovaný v laboratóriách podporených projektom SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výzkum a vývoj pro inovace.

Brno ....

podpis autora(-ky)

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM VÝPISOV

ÚVOD

Mnohé vedecké disciplíny sa stávajú multidisciplinárnymi a táto interkonektivita je zjavná predovšetkým v medicíne. Dnes, kedy potreba digitalizácie narastá každým dňom, by sa už ani tento obor nezaobišiel bez napredovania v informačnom odvetví. Tento trend si začali uvedomovať už aj kliniky a zdravotné zariadenia pôsobiace v Brne. Tie ešte aj v týchto dňoch používajú na svoju prácu málo efektívne zastaralé prostriedky, ktoré ich prácu spomaľujú. Deje sa to aj napriek tomu, že v súčastnosti sú už k dispozícii softwarové aplikácie, ktoré by im pomohli pri urýchlení procesu vyšetrenia a určovania diagnóz.

Z toho dôvodu sa na tento problém zameriava aj táto bakalárska práca, ktorá má pomôcť pri rýchlejšom určovaní diagnóz a rôznych ochorení. S týmto problémom sa hlavne potýka aj tím neurológov okolo prof. Rektorovej pôsobiacej na 1. neurologickej klinike LF MU u sv. Anny v Brne. Jej tím sa dennodenne zaoberá vyšetrovaním pacientov s neurologickými ochoreniami a pomocou klinických testov sa snaží určovať ich diagnózi.

Cieľom tejto bakalárskej práce je preto navrhnúť riešenie, ktoré by pomohlo pri jednoduchšom vytváraní takýchto testov a ich používaní pri vyhodnocovaní diagnóz.

V prvej časti sa preto zaoberáme navrhovaním a vytvorením systému, ktorému sa venuje prvá kapitola.

V druhej časti budeme ďalej anylozavať tento systém z hladiska bezpečnosti, ktorému bude venovaná ďalšia kapitola.

A nakoniec v tretej časti bude popísaný proces implementácie a testovania tohto systému, ktorému sa venuje tretia kapitola.

1 ÚVOD DO KLINICKÉHO TESTOVANIA PACIENTOV S NEUROLOGICKÝM OCHORENÍM

1.1 Klinické testy

Klinické alebo tzv. screeningové testy [5] sa často používajú na diagnostiku neurologických ochorení a hodnotia celkový stav pacienta. Jedná sa napr. o fyzický, pohybový, kognitívny (poznávací) výkon a iné. Týmito výkonmi je myslené to ako sa pacient správa v prebiehu dňa. Toto správanie je testované formou dotazníkov, ktoré sa zameriavajú na základné denné aktivity akými sú napr. nakupovanie, manipulácia s peniazmi, osobná hygiena, obliekanie sa apod. Taktiež sú nimi sledované poruchy chovania alebo psychické problémy, ako sú napr. halucinácie, bludy, depresia, agresivita, poruchy sexuálneho správania...

Dotazníky sa v súčasnej dobe najčastejšie vypĺňajú na papier a ich administrácia zaberie v priemere 20–40 minút, v závislosti na type testu. Ako už bolo spomenuté, testov existuje viacero typov, z ktorých každý sa zameriava na iný typ ochorenia. Testy sú väčšinou zložené z niekoľkých častí, kde každá časť obsahuje niekoľko otázok a každá má za cieľ sledovať rôzne každodenné aktivity, fyzické a psychické prejavy, pocity a nálady pacienta a iné. Káždá otázka je ohodnotená určitým počtom bodov, ktorý určuje pacientov stav v danej oblasti. Väčšinou sú hodnotené v nejakom rozmedzí napr. 0–5, ale zvyčajne sa to v každom teste líši. Výsledné skóre testu je súčet bodov z jednotlivých otázok, kde najlepšie skóre sa môže líšiť od typu testu. Napr. v jednom teste môže byť najlepšie skóre 0 bodov, v inom zase 100 bodov, ale môžu byť aj iné. Otázky v dotazníkoch majú väčšinou formu zaškrtávacích políčok (checkboxov) s jedným (radio button) alebo viacerými výbermi, alebo textovými políčkami na doplnenie slov alebo numerických hodnôt.

1.1.1 MDS-UPDRS test

Príkladom klinického testu je MDS-UPDRS (Movement Disorders Society - Unified Parkinson's Disease Rating Scale) zameraný na Parkinsonove ochorenie. Tento test má štyri časti:

1. Non–motorické aspekty každodenných aktivít
2. Motorickké aspekty každodenných aktivít
3. Vyšetrenie motoriky
4. Motorické komplikácie

Test sa skladá celkovo zo 70 otázok, kde otázky majú väčšinou formu piatich alebo dvoch checkboxov s jedným výberom. Najlepšie hodnotenie pre otázku je 0 a najhoršie 4. Niektoré otázky vypĺňa vyšetrujúci, iné zas sám pacient, prípadne s pomocou opatrovateľa. Odpoveď na každú otázku by mala odrážať pacientov stav z obdobia predchádzajúceho týždňa, vrátane dňa, kedy je pacient vyšetrovaný.

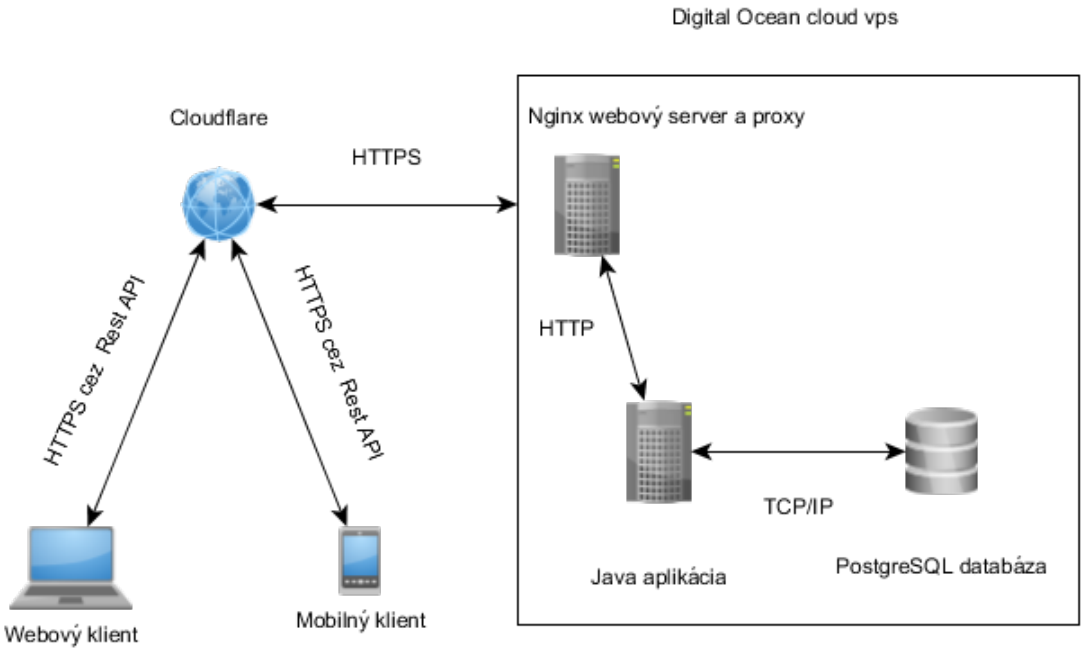
1.2 Využitie a dopyt

V súčasnosti pacienti a lekári stále vypĺňajú screeningové testy na papier, čo je zdĺhavý proces. Lekári musia výsledné skóre otázok prejsť, vyhodnotiť, a až následne vedia z toho určiť diagnózu. Výsledné skóre sa ešte aj zvykne prepisovať do Excelu a pri tom môže dochádzať ku chybám, napr. že nesprávne odpíšu číslo. Okrem toho musia viesť kartotéku pre každého jedného pacienta a administrácia s tým spojená tiež zaberá čas. Z toho dôvodu vzišla požiadavka od neurológov z 1. neurologickej kliniky LF MU u sv. Anny v Brne na zdigitalizovanie klinických testov a vytvorenie systému na tvorbu dotazníkov. Systém bude využívaný aj výzkumnou skupinou Aplikovanej neurovedy v CEITEC MU Brno. Systém bude zložený z dvoch klientských a jednej serverovej aplikácie. Jeho účelom bude zkrátiť čas potrebný na spracovanie výsledku diagnózy pacienta, komfortnejšie vyhľadanie minulých výsledkov a tiež možnosť vytvárať nové klinické testy.

2 NÁVRH SYSTÉMU A IMPLEMENTÁCIA

Prvotne, cieľom bakalárskej práce bolo len vytvoriť Android aplikáciu. Ale po premyslení toho, že v nej bude treba vytvárať aj klinické testy, čo by asi nebol pohodlný spôsob ich vytvárať na tablete, bolo nutné pre tento účel vytvoriť aj webovú aplikáciu. Následne z toho vyplýva, že vytvorené testy bude treba niekde ukladať, ale aj dostať do mobilnej aplikácie. Za účelom skvalitnenia práce, bude treba ešte vytvoriť aj serverovú aplikáciu, kde sa budú tieto vytvorené testy ukladať a zasielať na Android aplikáciu. V budúcnosti by webová aplikácia mohla mať aj ďalšie využitia ako len tvorbu testov. Príkladom by mohla byť evidencia pacientov a ich celkový prehľad zdravotného stavu, plánovanie návštev lekára, rôzne štatistiky, napr. či sa pacientovi zlepšuje alebo zhoršuje stav za dané obdobie, alebo porovnania pacientov s rovnakou chorobou, či okrem nej majú ešte aj nejaké iné ochorenia apod.

2.1 Schéma systému



Obr. 2.1: Topológia systému.

Ako je možné vidieť na obr. 2.1, systém sa bude skladať z dvoch klientských, jednej serverovej aplíkacie aplikácie v jave a ďalšími uzlami medzi nimi. Z tejto topológii je vidieť, že sa jedná o klient – server model, kde webová a mobilná aplikácia sa dotazujú serverovej. Komunikácia medzi nimi, bude prebiehať cez zabezpečený

HTTP (Hypertext Transfer Protocol) protokol a REST[1] (Representational State Transfer) API[2] (Application Programming Interface) rozhranie na serverovej aplikácii. Obe klientské aplikácie budú komunikovať s Cloudflare službou2.5, ktorá je prostredník a funguje ako reverzná proxy a DNS proxy. Okrem toho je to aj prvá vrstva ochrany celej architektúry, pretože rieši problémy s DDoS (Distributed Denial of Service) útokmi a vie fungovať aj ako webový firewall. Ďalším uzlom v komunikácii je nginx2.7 webový server, ktorý je umiestnený spolu s javovskou aplikáciou a relačnou PostgreSQL databazou na VPS[3] u cloudového poskytovateľa Digital Ocean2.6.

2.2 Webová aplikácia

Ako už bolo spomenuté v úvode kapitoly 2, účelom webovej aplikácie je umožniť lekárom ľahšie vytvárať klinické testy vo forme dotazníkov, ktoré by potom boli pacientmi vyplnené v mobilnej aplikácii. Hlavný problém, ktorý sa tu teda musel riešiť, bol vytvoriť editor na tvorbu dotazníkov. Predtým sa ale ešte bolo treba zamyslieť, ako sa k celému návrhu webovej aplikácie postaviť.

2.2.1 Porovnanie návrhových vzorov webových aplikácií

Možnosti, ako navrhnúť webovú aplikáciu je dnes mnoho, ale zvyčajne dve najviac používané webové návrhové vzory sú:

1. SPA – Single Page Application
2. MPA/MVC[4] – Multi Page Apllication/Model–View–Controller

Obe z týchto návrhových vzorov majú svoje výhody a nevýhody, ktoré bolo treba zvážiť pre náš prípad (viď. [7] a [4]).

Single Page Application

SPA je typ aplikácie, ktorej celá funkcionalita je obsiahnutá v jednej stránke a server používa len ako úložisko a zdroj dát. Všetky potrebné súbory sa stiahnu pri prvom príchode na stránku a Javascriptom sa vyrenderujú v prehliadači. Následne akákoľvek operácia spojená zo zmenou obsahu je vykonaná dynamicky a jediné čo sa zo serveru stiahne, sú nové dáta pre nový obsah. Toto správanie môžeme vidieť na obr. 2.2, kde v hornej časti bolo na stránku pristúpené prvý raz a v dolnej sa po zmene už len vymieňajú dáta. Ako už bolo spomenuté, takýmito technológiami, ktoré

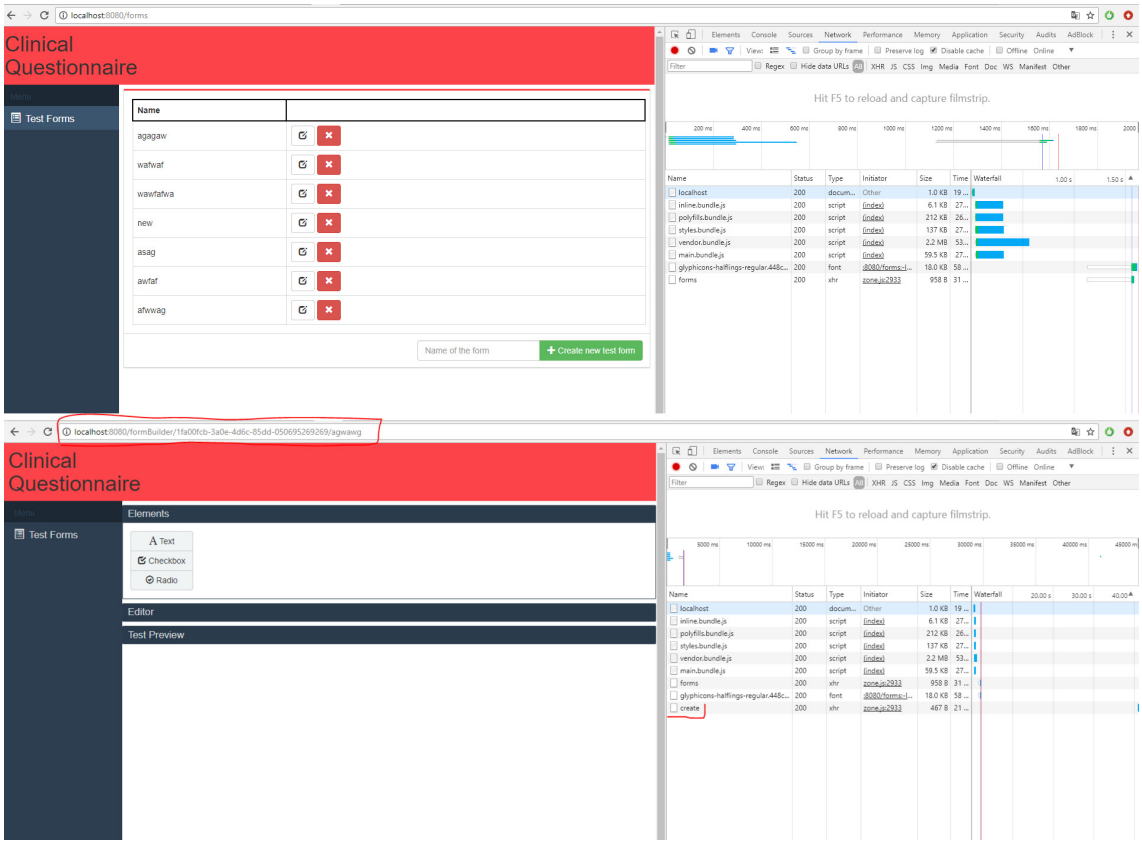
[1] Bližší popis ako funguje REST API je v časti 2.3.2.

[2] Toto je myslené v kontexte webu. API môže mať aj iný význam v terminológii softvérového inžinierstva.

[3] Virtuálny privátny server.

[4] Pre lepšiu predstavu bol uvedený najčastejšie používaný návrhový typ pre MPA.

fungujú na princípe SPA sú javascriptové frameworky ako AngularJS, Meteor.js a iné...



Obr. 2.2: Ukážka sťahovania súborov zo servera v SPA aplikácii.

Prečo používať SPA prístup:

- Rýchlosť. Väčšina zdrojov je stiahnutá len raz, a iba dáta sú prenášané tam a späť.
- Prehliadač nemusí vykreslovať znova celú stránku, ale len jej zmenené časti.
- Jednoduchší mobilný vývoj. Rovnaká serverová aplikácia môže byť použitá aj pre webovú, aj mobilnú aplikáciu.
- Flexibilita v UI (User Interface) návrhu. Oproti MPA je možné ľahšie prepísať klientskú časť, bez toho, aby to malo nejaký dopad na serverovú časť. Toto je možné, ak sa pevne definuje API na výmenu dát.

Nevýhody sú:

- Pri prvom načítaní stránky sa musia stiahnuť aj všetky balíčky frameworkov.
- Nefunkčnosť bez JavaScriptu. Pre určitý typ webov, hlavne na darknete je to zásadný problém.

- SEO (search engine optimization)[5] problémy. Roboty vyhľadávačov budú mať problém spracovávať obsah, keďže jeho aktuálizácia je vykonaná bez refreshu stránky.
- V porovnaní s inými webovými návrhovými vzormi, je SPA menej bezpečný, pretože zraniteľnosti v JavaScripte môžu byť závažným bezpečnostným rizikom.

Multi Page Application

Na druhej strane, MPA aplikácie pracujú tradičným spôsobom. Každá zmena na stránke spôsobí, že sa znova pošle žiadosť na server, pre získanie súborov a dát potrebných na zobrazenie konkrétnej stránky v prehliadači. Viď obr. 2.3, kde v obidvoch častiach je vidieť, že po zmene stránky sa súbory znova sťahujú. Tento spôsob fungovania je náročnejší na čas a znižuje UX (User Experience)[6]. Túto nevýhodu čiastočne pomáha zlepšiť AJAX (Asynchrónny JavaScript a XML), ktorý umožňuje vyžiadať si dáta, len pre časť stránky a tú zobraziť, bez toho aby sa musela znova načítať celá stránka. Pomohlo to zlepšiť UX, ale návrh stránky sa stal zložitejší. Medzi používané technológie pre MPA aplikácie patria ASP.NET MVC, Spring pre Javu, Django pre Python a asi najviac používané je PHP z jeho množstvom frameworkov, ako Nette, Laravel, Zend a iné...

Takou najväčšou výhodou MPA v porovnaní s SPA aplikáciami je v ich veľmi ľahkom manažovaní SEO. MPA weby majú vyššiu šancu byť lepšie ohodnotené v rámci výsledkov vyhľadávania. Toto je spôsobené hlavne tým, že jednotlivé stránky môžu byť optimalizované pre odlišné kľúčové slová.

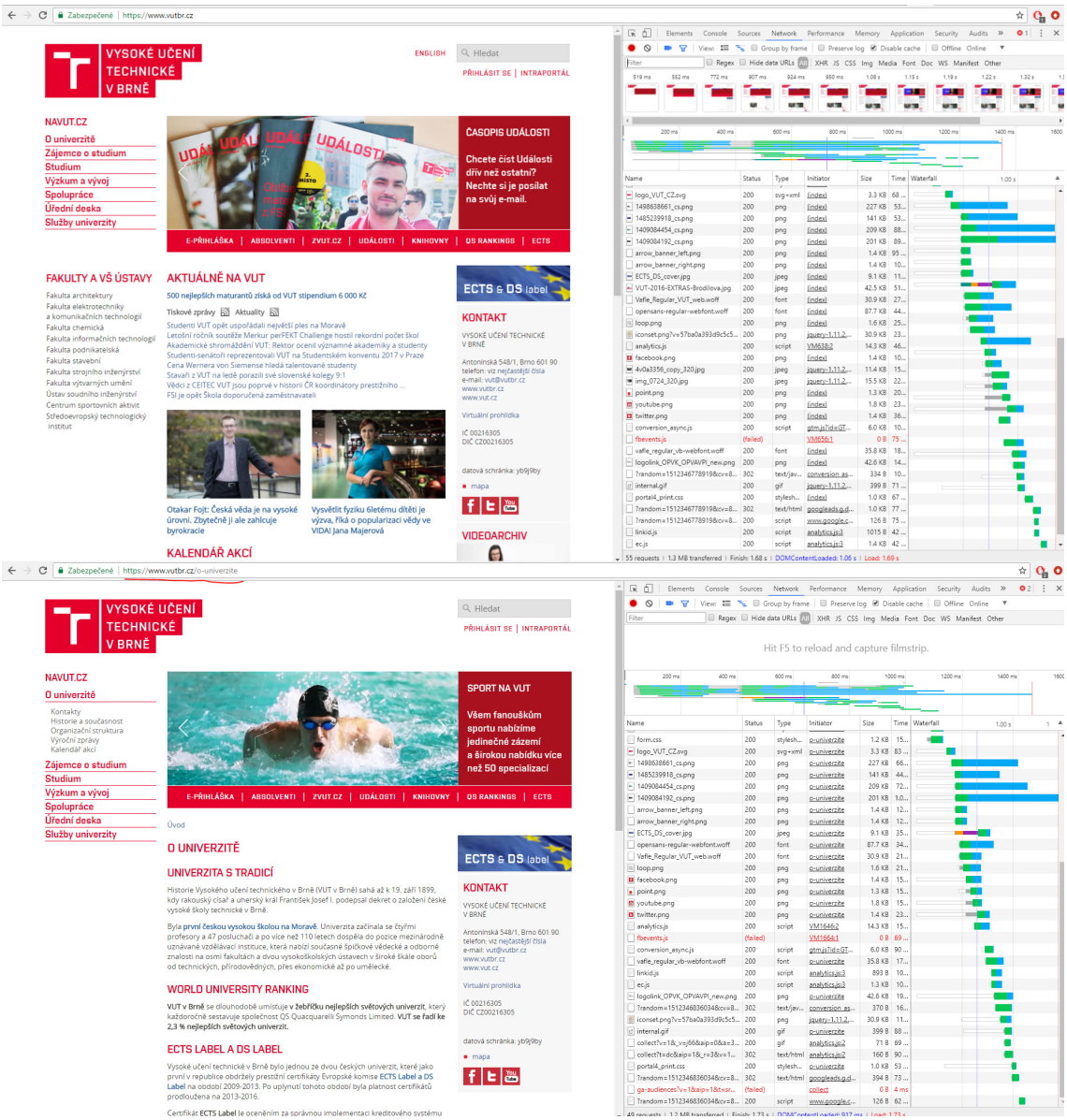
Okrem opätovného načítavania všetkých súborov pre novú stránku, nevýhodou ešte je, že vývoj je pomerne zložitejší, lebo frontend a backend sú úzko spojené.

2.2.2 Výber technológie pre webovú aplikáciu

Pri výbere technológie bol kladený dôraz hlavne na rýchlosť a jednoduchosť vytvárania dotazníkov, pretože tie sa budú vytvárať dynamicky. Keďže tvorba dotazníkov bude generovať veľké množstvo dát, ďalším dôležitým kritériom bol preto výber formátu, akým sa budú tieto dáta posielať na server. Okrem toho tieto dáta bude treba parsovať na strane mobilnej aplikácie, aby sa z nich potom mohli naspäť renderovať výsledné formuláre.

[5] V preklade je to optimalizácia pre vyhľadávače, čo je vlastne proces ovplyvňovania toho ako je webová stránka viditeľná na internete. Asi najdôležitejšou úlohou SEO je analyzovať kľúčové slová vo webe.

[6] V preklade je to užívateľská skúsenosť.



Obr. 2.3: Ukážka MPA aplikácie.

Na výber sme teda mali z dvoch najviac používaných formátov, ktoré sa dnes používajú na prenos pri komunikácii cez HTTP protokol. Takýmito formátmi sú JSON (JavaScript Object Notation) alebo XML (eXtensible Markup Language). V porovnaní týchto dvoch formátov pre naše potreby je lepší JSON, a to z týchto dôvodov:

- **Menej výrečný**. XML používa viac slov ako je potrebné. Jeho ukončovacie tagy, zvyšujú veľkosť kódu.
- **Rýchlejší**. Parsovanie je rýchlejšie a ľahšie, zatiaľ čo XML je pomalé a ťažkopádne. Mnoho knižníc na manipuláciu XML DOM[7] (Document Object Model)

[7]Je to štandard akým sa pracuje s XML elementmi.

štruktúry, môžu viesť k tomu, že v aplikácii je potreba alokovať veľké množstvo pamäte. Toto je nežiadúce hlavne pre prenosné zariadenia, ktoré majú nižšie pamäťové kapacity. Pri nich je dôležité pracovať, s čo najmenšími pamäťovými nárokmi, pretože ľahko sa môže stať, že aplikácia spadne, z dôvodu nedostatku pamäte.

- **Natívny formát v JavaScripte**. Zahŕňa rovnaké dátové typy ako sú reťazce, čísla, polia, boolean a objekty.

Jedinou výhodou XML oproti JSON formátu je, že je ľahšie čitateľný človekom a viacej vývojárov mu rozumie, ale to je v našom prípade zanedbateľné (viď. [9] a [10]).

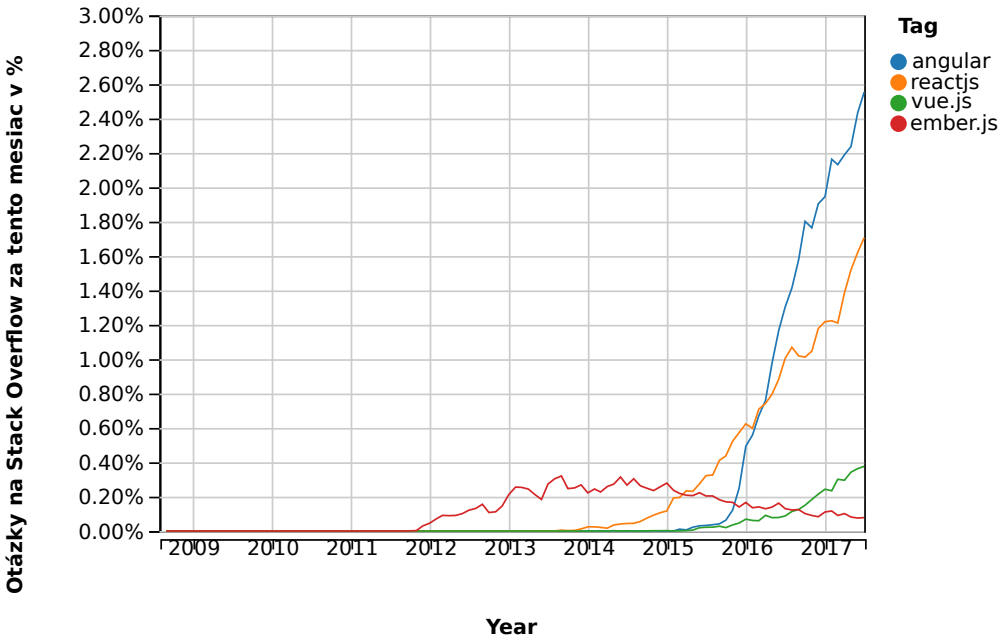
Keďže tvorba dotazníkov je dynamický proces a obsahuje nejakú logiku, ktorá bude potrebná pri ich pohodlnom vytváraní a okamžitom vizualizovaní, z toho dôvodu to bude lahšie vykonateľné práve nejakým moderným JavaScriptovým frameworkom. V nich existujú nástroje, ktoré tento problém efektívne riešia bez toho, aby sme potrebovali veľa kódu na ich naprogramovanie. Toto by v prípade nejakej MPA technológie nebolo tak ľahko zrealizovateľné, pretože frontendový a backendový kód by boli viac zložitejšie. Okrem toho, by sa pri každom vytvorenom prvku v dotazníku musela táto informácia odoslať na server cez AJAX, aby s touto zmenou vedel backend pracovať. Tým pádom tieto operácie zvyšujú aj režijné náklady. Zatiaľ čo pri SPA, by sa všetky tieto zmeny držali v klientskej aplikácii a až na konci po vykonaní všetkých operácií, ktoré užívateľ potrebuje, by sa tieto dáta odoslali na server. Posledná vec, čo v našom prípade nehrá rolu je zameranie aplikácie na SEO, keďže celá klientská aplikácia, bude len pristupná lekárom. V konečnom dôsledku, pri porovnaní návrhových vzorov v podkapitole 2.2.1 a následných uváženiach v tejto sekcii 2.2.2, je pre nás vhodnejším riešením SPA prístup.

JavaScriptové frameworky

V súčasnosti popularita SPA aplikácii má zvyšujúci sa trend a je to hlavne spôsobené aj JavaScriptovými frameworkami, ktoré vývojárom úľahčujú a urýchlujú ich prácu tým, že sa už nemusia starať o to ako štrukturovať ich aplikáciu. Framework im slúži ako kostra systému a vedie ich k zaužívaným konvenciám, ktoré im umožňujú sa hlavne zamerať na funkcionalitu výsledných riešení. Ďalšou veľkou výhodou je ich členská základňa a zázemie veľkých firiem ako sú napríklad Facebook alebo Google, ktoré sa podieľajú na ich vývoji.

Obrázok ukazuje 2.4, na ktoré frameworky sa vývojári dotazujú v snahe nájsť riešenie ich problému na známom webe Stack Overflow[8]. Na vodorovnej osi je možné

[8]Stránka zameraná na výmenu znalostí a riešenia problémov medzi vývojármi. Viď. https://stackoverflow.com/



Obr. 2.4: Porovnanie popularity JavaScriptových frameworkov na známom vývojárskom webe Stack Overflow. Zdroj: https://insights.stackoverflow.com/trends

vidieť, v ktorom roku jednotlivé frameworky vznikly a ako rástol ich záujem v podobe otázok kladených na ne v priebehu mesiacov, čo sa prejavilo na zvislej osi. Z porovnania je zrejmé, že v čase písania tejto bakalárskej práce sú najviac populárne dva frameworky – Angular od Googlu a React od Facebooku. Následný prehľad nám viac priblíži, ktorý z týchto dvoch spomenutých frameworkov bude vhodnejší pre našu klientskú aplikáciu.

Angular

Ako už bolo spomenuté, patrí medzi najpopulárnejšie frameworky, vyvíjané spoločnosťou Google a používa ho aj plno iných spoločností pre svoje webové, ale aj mobilné aplikácie. Tento framework má už niekoľko vydaných verzií a v čase písania tejto bakalárskej práce, bola jeho najvyššia verzia nesúca názov Angular 4. U Angularu sa však v minulosti udiali radikálne zmeny v architektúre a to hlavne od verzie 2, ktorá sa berie ako odlišný framework oproti Angularjs. Tieto dve verzie sú odlišné hlavne z hľadiska architektúry, kde obidve stavajú na iných návrhových vzoroch, a tým pádom už nie sú tak kompatibilné. Okrem toho nové verzie už používajú ako programovací jazyk TypeScript. Avšak väčšina firiem ešte stále používa k vývoju Angularjs oproti novším verziám, aj napriek tomu, že v znikol v roku 2010 (viď. [2]).

Angular na rozdiel od Reactu je plnohodnotný framework a ponúka všetky veci

pohromade v jednom balíku. To vedie vývojárov k rýchlejšiemu a ľahšiemu návrhu aplikácie, bez potreby prvotného analyzovania a väčšej zodpovednosti pri výbere knižníc ako sa to robí pri Reacte (viď. [2]).

Ako už bolo spomenuté vyššie, Angular používa jazyk TypeScript vyvinutý firmou Microsoft, za účelom zníženia zložitosťi pri písaní komplexného kódu v jazyku JavaScript (viď. [8]). TypeScript je nadstavbou JavaScriptu a je do neho aj kompilovaný, takže z toho dôvodu nie je problém použiť, už existujúce populárne Javascriptové knižnice a volať ich kód práve v TypeScipte. Jazyk zahŕňa koncepty ako sú voliteľné statické typy, nové dátové štruktúry a viac objektovo–orientovaných vlastností ako sú triedy a rozhrania. Práve zavedenie typovosti zo sebou prinieslo aj jednu veľkú výhodu a to, že umožňilo IDE[9] (Integrated Development Environment) nástrojom vykonávať funkcie ako statickú typovú kontrolu[10], refaktorizáciu kódu[11] alebo prekliknutie do definície nejakého dátového typu.

Výpis 2.1: Ukážka jednoduchého kódu v jazyku TypeScript.

```
1 class Greeter {
2     greeting: string;
3     constructor (message: string) {
4         this.greeting = message;
5     }
6     greet(): string {
7         return "Hello, " + this.greeting;
8     }
9 }
```

Výpis 2.2: Ukážka toho istého kódu v jazyku JavaScript.

```
1 var Greeter = (function () {
2     function Greeter(message) {
3         this.greeting = message;
4     }
5     Greeter.prototype.greet = function () {
6         return "Hello, " + this.greeting;
7     };
8     return Greeter;
9 })();
```

[9]V preklade vývojové prostredie je software uľahčujúci prácu vývojárom.

[10]Proces overovania dátových typov v zdrojovom kóde.

[11]Proces reštruktualizácie existujúceho kódu, z dôvodu jeho lepšej čitateľnosti a zníženiu zložitosti, bez zmeny jeho správania.

Z kódu 2.1 je možné vidieť, že TypeScript deklaruje typy členských premenných ako aj parametrov v konštruktore, a tiež deklaruje návratové typy metód. Oproti tomu kód 2.2 ukazuje rozdiel, že v JavaScripte nie sú typy premenných a parametrov deklarované, ale sú zistené za behu programu.

Čo sa týka štruktúry kódu, Angular rozdeluje logiku do tzv. komponentov a užívateľské rozhranie (ďalej len UI) do šablón. Proces ako to funguje je taký, že užívateľ interaguje s UI, ktoré posiela dáta do komponenty alebo ich z nej prijíma a vykresluje. V komponente prebieha ich spracovanie a následné rozhodovanie čo s nimi, či ich poslať na server alebo ich z neho získať a podobné operácie. Jednoduchý príklad ako taká interakcia medzi šablónov a komponentom funguje, môžeme vidieť v kóde 2.3. Tu vidíme, že šablóna (začínajúca na riadku 7) a logika sú oddelené a komunikujú spolu cez funkcie, ktoré prijímajú dáta od užívateľa a následne s nimi ďalej pracujú. Pri väčšom množstve HTML kódu v šáblóne, sa zvyčajne píšu vo vlastnom súbore, podobne ako je to so štýlmi. Tie sú tiež uložené v ich vlastnom súbore, čo môžeme vidieť na riadku 13, kde sa definujú štýly pre túto šablónu.

Výpis 2.3: Príklad interakcie medzi komponentov a šablonou v Angulare.

```
import {Component} from '@angular/core';
import {FormBuilderService} from '../form-builder.service';
import {FormField} from '../../form-field';

@Component({
  selector: 'app-text-edit',
  template: `<div class="box-form">
  <input class="name-input" placeholder="Write␣the␣name␣of␣
     part..." #name (keyup)="addName(name.value)">
  <textarea class="text-input" placeholder="Write␣your␣text␣
     here..." #letter (keyup)="addText(letter.value)">

  </textarea>
</div>`,
  styleUrls: ['./text/edit/text-edit.component.css']
})
export class TextEditComponent {
  formField: FormField;

  text = '';
  partName = '';

  constructor(private builderService: FormBuilderService) {}

  addName(letter: string) {
    this.partName = letter;
    this.builderService.changeNameEdit(this.formField.id,
       this.partName);
  }

  addText(letter: string) {
    this.text += letter;
  }
}
```

React

React je popisovaný ako zoskúpenie knižníc na tvorbu uživateľských rozhraní, ktoré sú udržiavané a vyvíjané spoločnosťou Facebook. Spoločnosť ho aj sama používa na niektorých svojich častiach, ale nie však ako SPA aplikáciu. Oproti Angularu je

verzovanie Reactu plynulé a v jeho vývoji sa neudiali také drastické zmeny. Súčasná verzia má číslo 16 a nesie pomenovanie React Fiber[12].

V porovnaní s Angular je React viac flexibilný a ponúka vývojárom väčšiu nezávislosť v tom, že nemusia sťahovať celý balík, ale len jednotlivé balíky, ktoré vo svojom projekte potrebujú. Tento prístup má však aj nevýhody v tom, že veľa knižníc nevyvíja len Facebook, ale aj nezávislí vývojári. Následkom toho, prípadné aktualizácie alebo ešte horšie, ich neudržiavanie si musia vývojári, ktorí ich používajú sledovať sami. Okrem toho, každý projekt sa stáva odlišný z hľadiska architektúry zvolených knižníc a vývojari sa musia s nastavením každého projektu zoznamovať zvlášť (viď. [2]).

Čo sa týka programovacieho jazyka, React používa svoje vlastné rozšírenie pre syntax jazyka Javascript, a to JSX (JavaScript XML) (viď. [3]). Tento jazyk používa najnovší štandard ECMAScript 6 vydaný v roku 2015, ktorý definuje normy pre skriptovacie jazyky. JSX je voliteľným preprocesorom[13], ktorý umožňuje písať XML kód a následne ho kompilovať do JavaScriptu .

Výpis 2.4: Ukážka jednoduchégo kódu v JSX jazyku.

```
1  <div id="myReactApp"></div>
2
3  <script type="text/babel">
4    class Greeter extends React.Component {
5      render() {
6        return <h1>{this.props.greeting}</h1>
7      }
8    }
9
10   ReactDOM.render(<Greeter greeting="Hello␣World!" />,
       document.getElementById('myReactApp'));
11 </script>
```

Podobne ako Angular, tak aj React pracuje s princípom komponentov, ale trošku iným spôsobom. Ako je vidieť v kóde 3.1, tak ako aj logika, aj šablóna sú spolu v jednom súbore. Pri väčšom množstve kódu, sa to môže stať neprehladnejšie, ale má to aj veľkú výhodu a to, že kompilácia a dopĺňanie kódu prebieha rýchlejšie.

Súhrn

Pri výbere frameworku sme sa hlavne rozhodovali podľa týchto kritérií:

[12]Zdroj: https://github.com/acdlite/react-fiber-architecture

[13]Počítačový program, ktorý spracováva vstupné dáta tak, aby výstupné mohli byť ďalej spracované iným programom, zvyčajne práve kompilátorom.

- **Počiatočná časová náročnosť**. Tou je myslené, koľko času zaberie zoznámenie sa s frameworkom, prípadne výber vhodných knižníc.
- **Programovacia paradigma**. Porovnanie a uváženie, ktorý jazyk je vhodnejšie sa učiť a pochopiť.
- **Budúci rozvoj aplikácie**. V prípade, že by sa projekt rozvinul do väčších rozmerov je potrebné, aby eventuálne noví programátori sa jednoduchšie zapojili do vývoja projektu.

Z odpovedí na tieto požiadavky nám vyplýva, že vhodnejší framework pre nás bude Angular. Vychádzali sme hlavne z toho, že pri Reacte, by počiatočné analyzovanie a rozhodovanie, ktoré knižnice vybrať spotrebovali väčšie množstvo času. Ďalej čo sa týka programovacieho jazyka, tak je jednoduchšie uvažovať v dvoch jazykoch z objektovo–orientovanou paradigmou, ako skákať medzi paradigmami pri vývoji. Posledné kritérium berie do úvahy to, že v Angulare sú logika a UI oddelené v jednotlivých súboroch, a to vedie k čistejšiemu kódu, ktorý nový programátora rýchlejšie pochopia.

2.2.3 Implementácia a popis webového klienta

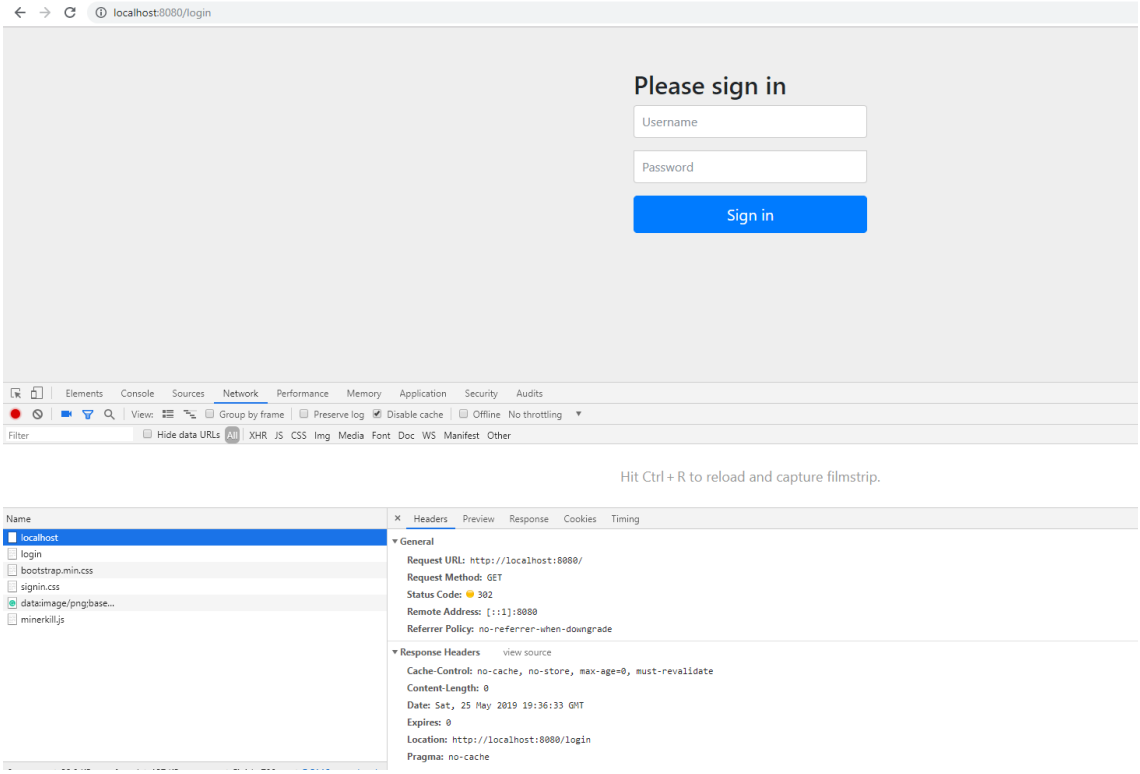
Webová aplikácia sa skladá z troch stránok:

- **Prihlosovacia stránka**.
- **Stránka zobrazujúca zoznam vytvorených testov**.
- **Stránka na vytvorenie testového dotazníka**.

V ďalších podčastiach budú popísané jednotlivé stránky a ich funkcia.

Prihlasovacia stránka

Po zadaní webovej adresy do prehliadača sa ako prvá načíta stránka s prihlasením. Dole na obrázku2.5 je vidieť, že pri zadaní adresy **http://localhost:8080/** sme boli presmerovaný na adresu **http://localhost:8080/login**. Toto chovanie za nás rieši knižnica Spring securiy, ktorá aj generuje túto stránku na strane servera, a jej výhody sú hlavne v rýchlej implementácii základnej autentizácie užívateľov (viď. [11]). Toto zabezpečenie je hlavne z toho dôvodu, že s webovým klientom môžu pracovať iba autentifikovaný užívatelia.



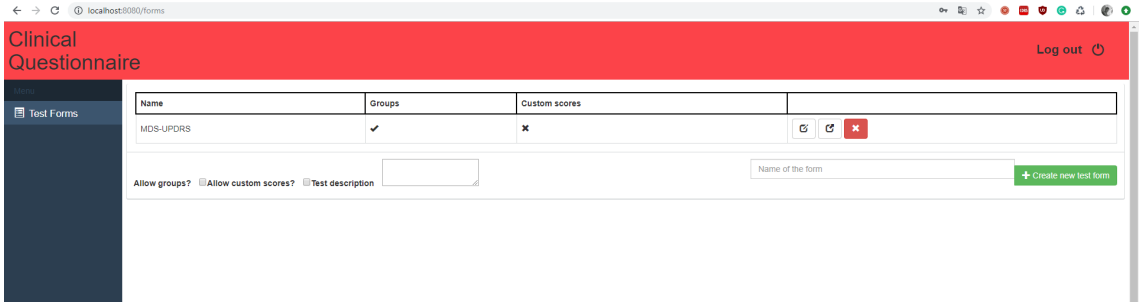
Obr. 2.5: Stránka prihlásenia.

Stránka zobrazujúca zoznam vytvorených testov

Po úspešnom prihlásení, sa stiahnu všetky javascriptové súbory, spolu s obrázkami, a dáta sú získané z **http://localhost:8080/forms** restovej adresy. Jednotlivú navigáciu medzi stránkami si už rieši odteraz Angular a iba dáta pre ne sú ziskavané zo servera HTTP dotazmi (viď. [12]). Následne po dokončení sťahovania, sa vyrenderuje stránka zo zoznamom už vytvorených testov. Tento zoznam reprezentuje tabuľka2.6, kde riadky sú jednotlivé testy a stĺpce definujú len základné informácie testov, ako sú názov, informácia či test obsahuje skupiny otázok alebo či hodnotené otázky môžú mať body definované užívateľom. V poslednom sĺpci sú tri tlačidlá,

kde prvá dve editujú test a tretím ho mažú. Prvým tlačidlom editujeme položky, ktoré na tejto stránke nevidno, ale budú vysvetlené v ďalšej podkapitole. Druhým tlačidlom sa editujú položky zobrazené v tabuľke.

Čo sa týka vytvárania testov, tak tie je možné vytvárať na tejto stránke a to vyplnením formulárových políčok pod tabuľkou a zaškrtnutím checkboxov, ktoré definujú základné údaje testu rovnako ako v stĺpcoch v tabuľke. Test je následne vytvorený kliknutím na zelené tlačidlo "Create new test form", po ktorom sa následne pošle HTTP POST požiadavka na server s vyplnenými údajmi pre test. Po tom čo sa nový test uloží do databázy, sa vráti odpoveď zo servera, ktorá nesie informáciu o unikátnom identifikátore pre test, tzv. **UUID**, čo je vlastne 128 bitov dlhý reťazec náhodne vygenerovaných znakov (viď. [13]), sa z tejto informácie a názvu poskladá url stránka pre daný test.



Obr. 2.6: Stránka zoznamu vytvorených testov.

2.2.4 Stránka na vytvorenie testového dotazníka

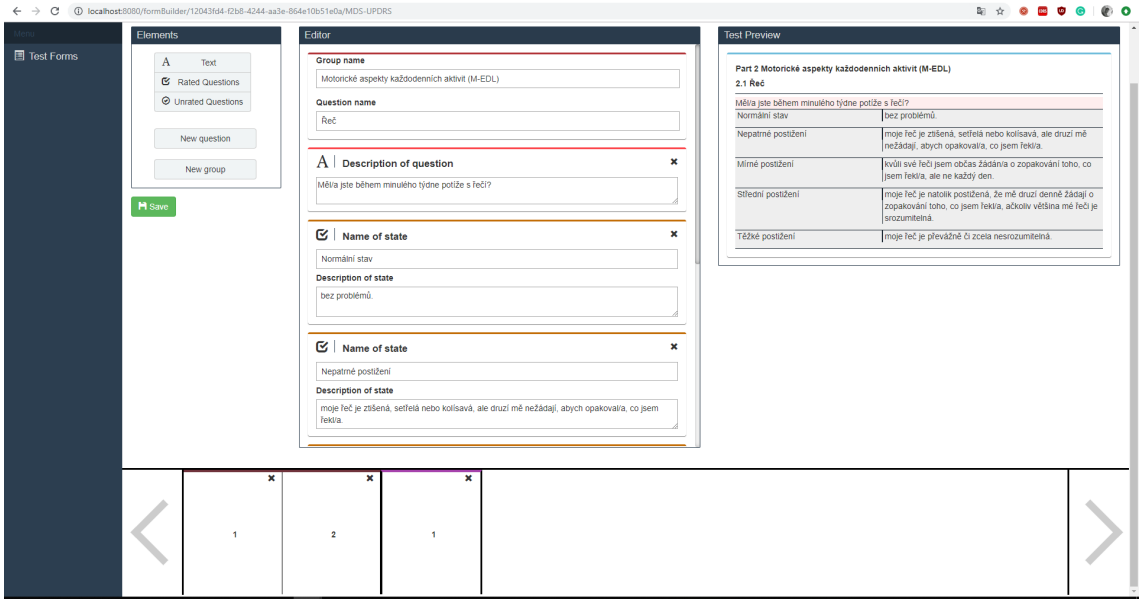
Ako je možné vidieť na obr. 2.7, tak stránka je rozdelená na štyri časti: tri vertikálne stĺpce a jednu horizontálnu — tzv. slider. Všetky tieto časti reprezentujú prácu pri vytváraní definície testu.

Začneme sliderom, v ktorom sú zobrazené jednotlivé skupiny testov. Skupiny sú odlíšené farebnými pruhmi pre ich lepšiu identifikáciu. Jednotlivé otázky v skupinách sú indexované od čísla 1, a preto môžete vidieť v slideri dve číslice 1. Ak je v slideri veľa otázok a prekročili by šírku obrazovky, tak na kvôli tomu sú tam tie šípky, ktorý ním dokážu hýbať.

Vo vertikálnych častiach začneme ľavým stĺpcom, ktorý má názov **Elements** a v ňom sú pvky ktorými vieme definovať test. Prvé tri položky zhora, definujú z akých prvkov sa môže skladať otázka a zvyšné dve vytvárajú nové otázky a skupiny testu a pridajú do slideru novú otázku. Pri klikaní na položky s ikonami sa v strednom a pravom stĺpci zobrazia tieto elementy, ktoré definujú to, ako sa konkrétna otázka

zobrazí potom v mobilnej aplikácii a aký bude mať obsah. Konkrétne, stredný stĺpec slúži ako editor týchto prvkov a pravý stĺpec zobrazuje náhľad na konkrétnu otázku.

Aby sa test mohol zobraziť potom v mobilnej aplikácii, je potrebné ho uložiť, kliknutím na tlačidlo “Save“.



Obr. 2.7: Stránka na vytvorenie testového dotazníka.

2.3 Serverová aplikácia

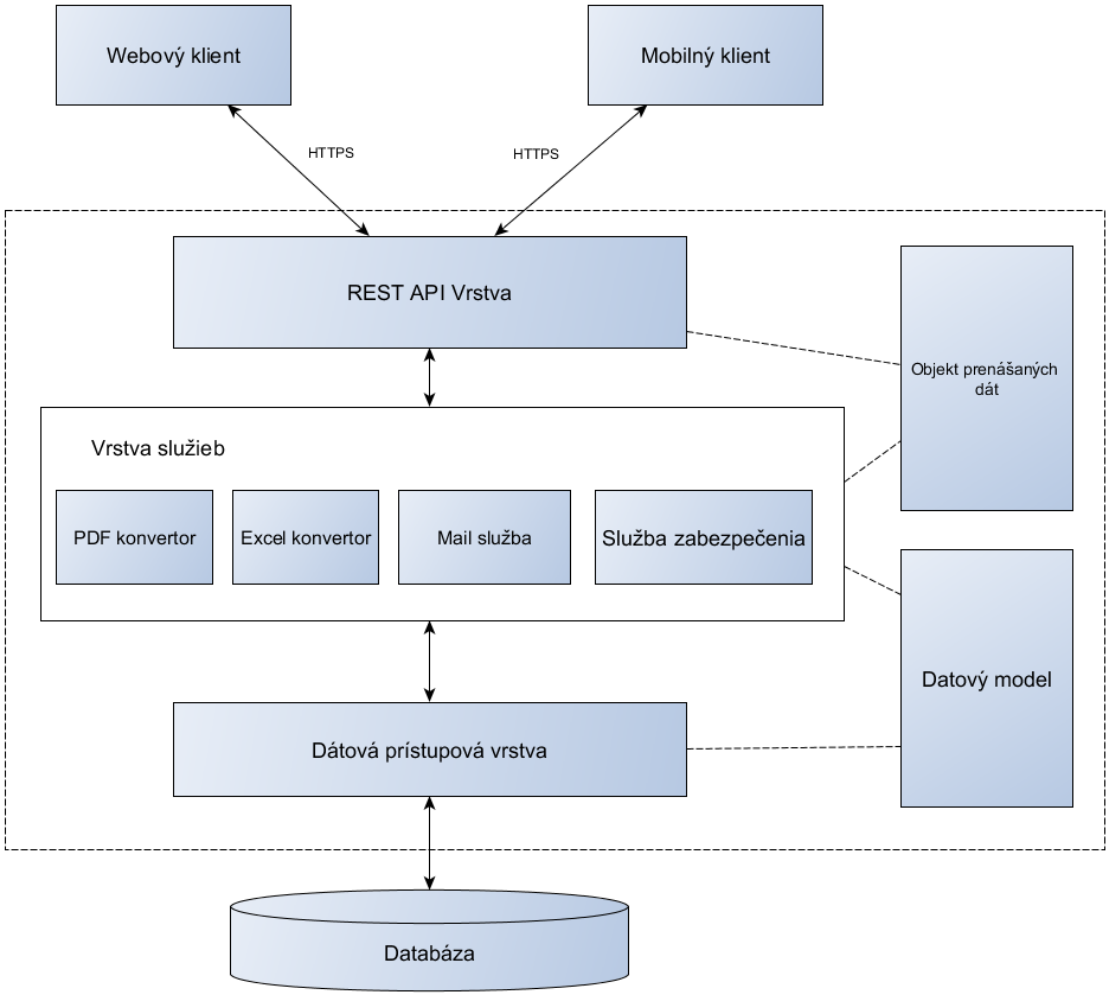
Ďalším dôležitým článkom tohto systému je serverová aplikácia. Okrem toho, že hostuje zdrojový kód webovej aplikácie, plní aj iné dôležité funkcie, ako je odpovedanie na HTTP žiadosti a vracanie HTTP odpovedí na klienstké aplikácie alebo uchovánie vytvorených testov v databáze. Mimo iné obsahuje aj funkcionality, ktoré riešia export výsledkov testov do PDF alebo XLSX formátu a následne ich odosielanie na e-mail.

2.3.1 Výber technológie pre serverovú aplikáciu

Pri písaní serverových aplikácií máme na výber z viacej programovacích jazykov, medzi ktoré patria PHP, Python, Java, C# a iné. . . Každý má iné vlastnosti a líši sa v niečom inom. Napríklad v tom ako sú navrhnuté, či sa kompilujú alebo interpretujú, či dobre fungujú na konkrétnom operačnom systéme apod. Pre nás je ale najviac dôležitá posledne spomenutá vec, a to z dôvodu, že sme ešte dopredu nevedeli na akom operačnom systéme aplikácia bude bežať. Z toho nám vychádza, že vhodný

programovací jazyk by mohol byť Java pretože je to multiplatformový jazyk[14]. Tento výber má aj ďalší dôvod, prečo sa nám oplatí písať serverovú časť v Jave, a to ten, že Android je poväčšine napísaný[15] v Jave a tým pádom sa môžeme lepšie zamerať na vývoj v rovnakom jazyku.

2.3.2 Popis procesov v serverovej aplikácii



Obr. 2.8: Architektúra serverovej aplikácie.

Na obr. 2.8 môžeme vidieť, že serverová aplikácia má trojvrstvovú architektúru, kde každá vrstva má za úlohu plniť určitú funkciu. Tento typ architektúry je bežný pre javovské aplikácie (viď. [1]) a jeho hlavnou myšlienkou je, aby aplikácia bola jednoduchá a prehľadná, aj pri jej prípadnom budúcom raste.

[14]Funguje na rôznych operačných systémoch.

[15]Niektoré časti sú napísané aj v jazyku C++, ktoré vyžadajú pracovať s natívnym kódom z linuxového jadra.

REST API vrstva

Hlavnou úlohou API vrstvy je komunikovať s inými aplikáciami a je to jedným zo vstupných bodov do aplikácie. Cez API je možné predávať dáta do aplikácie alebo ich z nej získavať. V tejto vrstve by nemala byť implementovaná žiadna logika, a preto všetky požiadavky smeruje k vrstve služieb, ktorá ďalej rieši čo urobiť s požadovaným dotazom. Čo sa týka komunikácie s otatnými aplikáciami, tak na to sa využíva najčastejšie HTTP protokol alebo jeho zabezpečená verzia HTTPS.

HTTP protokol obsahuje telo a hlavičku. Hlavička obsahuje metadata, čo sú informácie ako napr. použitý prehliadač, typ prenášaných dát, cookies[16], autentizácia a iné... V tele sa zas posielajú dáta, ktoré majú byť spracované aplikáciou, v našom prípade sú to vytvorené testové dotazníky.

Aby serverová aplikácia spoznala, čo od nej žiadajú klientské aplikácie, k tomu účelu služia HTTP metódy (viď. [6]):

- **GET** metóda slúži na získavanie dát od servera.
- **POST** metóda je používaná na uloženie nových alebo dát.
- **PUT** metóda je podobná metóde POST, ale narozdiel od nej edituje už existujúce dáta.
- **DELETE** metóda je určená na mazania dát.

HTTP metód existuje ešte viacej, ale tieto sú najviac používané. Ďalšou dôležitou vecou čo sa týka komunikácie, je spôsob akým HTTP metódy pristupujú k datám. K tomuto slúžia princípy, založené na REST architektúre, ktoré pomáhajú definovať webové rozhranie. Tieto princípy sa používajú na jednotný prístup k zdrojom na serveri. Zdroje môžu byť dáta alebo aj stavy. Stavom myslíme nejaký proces, ktorý sa má vykonať. Napríklad v našom prípade, by to mohol byť export do PDF formátu. Zdroje sú definované pomocou URI[17] (Uniform Resource Identifier) identifikátorov a špecifikujú konkrétny zdroj. Napríklad v našom prípade by mohol byť zdroj definovaný takto http://localhost:8080/forms. Toto by mohol byť príklad pre požiadavok HTTP metódy GET z webovej aplikácie, ktorá by v odpovedi, dostala všetky už existujúce testy.

Vrstva služieb

Ako už bolo spomenuté, vrstva služieb je miesto, kde sa nachádza logika celej aplikácie. Táto vrstva spolupracuje so spomínanou API vrstvou, ktorá jej predáva požiadavky na spracovanie a následne sa jej vracia výsledok, ktorý má vrátiť na klientskú aplikáciu. Požiadavok môže byť cielený na získanie dát alebo spracovanie

[16] Slúžia ako pomocné dáta pre server, ktoré sa uložia na počítač užívateľa.
[17] Je to reťazec znakov, ktorý identifikuje zdroj.

stavov. V prípade dát je žiadosť ďalej preposlaná na datovú vrstvu. U stavov je to iné, lebo tie majú za úlohu vykonávať určitú prácu v aplikácii. V našej aplikácii máme štyri služby, ktoré majú za úlohu plniť konkrétne funkcie a fungovať nezávisle na aplikácii. K týmto službám sa pristupuje cez ich API metódy, ktoré príjmu žiadosť, spracujú ju a vrátia výsledok.

- **PDF konvertor** má na starosti z výsledkov testov prijatých z mobilnej aplikácie, ktoré mu prídu v JSON formáte, vygenerovať výsledný pdf dokument. Tieto dáta sa však predtým budú musieť previesť do vhodného formátu a to z dôvodu toho, že konvertovanie bude prehiehať volaním aplikácie **WKHTMLTOPDF**[18], ktorá berie na vstupe html. Služba vráti cestu na vygenerovaný súbor.
- **Excel konvertor** má podobnú funkciu ako PDF konvertor s tým rozdielom, že výsledný dokument bude nejakého excel formátu ako je xlst, xls alebo csv.
- **Mail služba** má za úlohu odosielať v prílohe vygenerovaný dokument s výsledkami testov na zadanú e-mailovú adresu. Táto e-mailová adresa príde s výsledkami testov z mobilnej aplikácie. Služba vracia výsledok, či operácia prebehla úspešne alebo vráti chybu.
- **Služba zabezpečenia** má za hlavný cieľ vykonávať kontrolu prístupov do klientských aplikácií, tzv. autentizáciu.

Dátová vrstva a databáza

Dátová vrstva alebo niekedy nazývaná aj perzistentná vrstva, je najnižšou vrstvou v serverovej aplikácii a slúži ako rozhranie pre prístup k databáze. Podobne ako služby v predchádzajúcej vrstve, aj táto ponúka API vo forme metód, ktorými je možné vykonávať dotazy do databázy a takto operovať nad jednotlivými dátovými objektmi.

Ako môžte vidieť z obr. 2.8 databáza sa nachádza mimo našej aplikácie, a to z dôvodu toho, že je to samostatná počítačová aplikácia, s ktorou aby sme mohli komunikovať, musíme vytvoriť spojenie. Toto spojenie v javovských aplikáciach sa volá JDBC (Java Database Connection) a je to API, ktoré definuje ako pristupovať k databáze.

[18]Zdroj: https://wkhtmltopdf.org/

Dátový model a Objekt prenášaných dát

So všetkými vrstvami úzko súvisia Dátový model a objekt prenášaných dát (v skratke DTO ako Data Transfer Object).

- **Dátový model** definuje dátovú štruktúru v našej aplikácii, ktorá slúži hlavne na uchovávanie informácií. Jednotlivé prvky v dátovej štruktúre sa nazývajú entity alebo dátové objekty. Entity popisuje logickú jednotku v aplikácii, ktorá definuje jej vlastnosti cez atribúty a správanie pomocou metód.

Výpis 2.5: Ukážka definovania dátovej jednotky v aplikácii.

```
1  @Entity
2  @Table(name = "test_form_definition")
3  public class TestFormDefinitionEntity {
4
5      @Id
6      @GeneratedValue
7      @GenericGenerator(name = "uuid", strategy = "uuid2")
8      @Column(unique = true)
9      private UUID uuid;
10
11     @Column
12     private String name;
13
14     @Column
15     private String definition;
16
17     public TestFormDefinitionEntity() {}
18
19     public TestFormDefinitionEntity(String name, String
          definition) {
20         this.name = name;
21         this.definition = definition;
22     }
23
24     public String getName() {
25         return name;
26     }
27
28     public UUID getUuid() {
29         return uuid;
30     }
31
32     public String getDefinition() {
33         return definition;
34     }
35
36     public void setName(String name) {
37         this.name = name;
38     }
39
40     public void setDefinition(String definition) {
41         this.definition = definition;
42     }
43 }
```

Príklad takejto entity v našej aplikácii, môžete vidieť v kóde 2.5, ktorá reprezentuje jeden testový dotazník. V databáze sa tieto entity mapujú na tabuľky, ktorých stĺpce reprezentujú jednotlivé atribúty v entite.

- **Objekt prenášaných dát** predstavuje dátový kontajner pre entitu, ktorý sa používa na prenos jej dát medzi jednotlivými vrstvami v aplikácii. Je to z dôvodu toho, že entita môže mať väzby aj na iné entity a v prípade zmien na tejto entite, by to mohlo ovplyvniť aj ostatné. Okrem toho DTO môže rozšíriť chovanie entity pre špecifické účely, napríklad, ak chceme pridať nejakú informáciu pri odoslaní na klientské aplikácie.

2.4 Android aplikácia

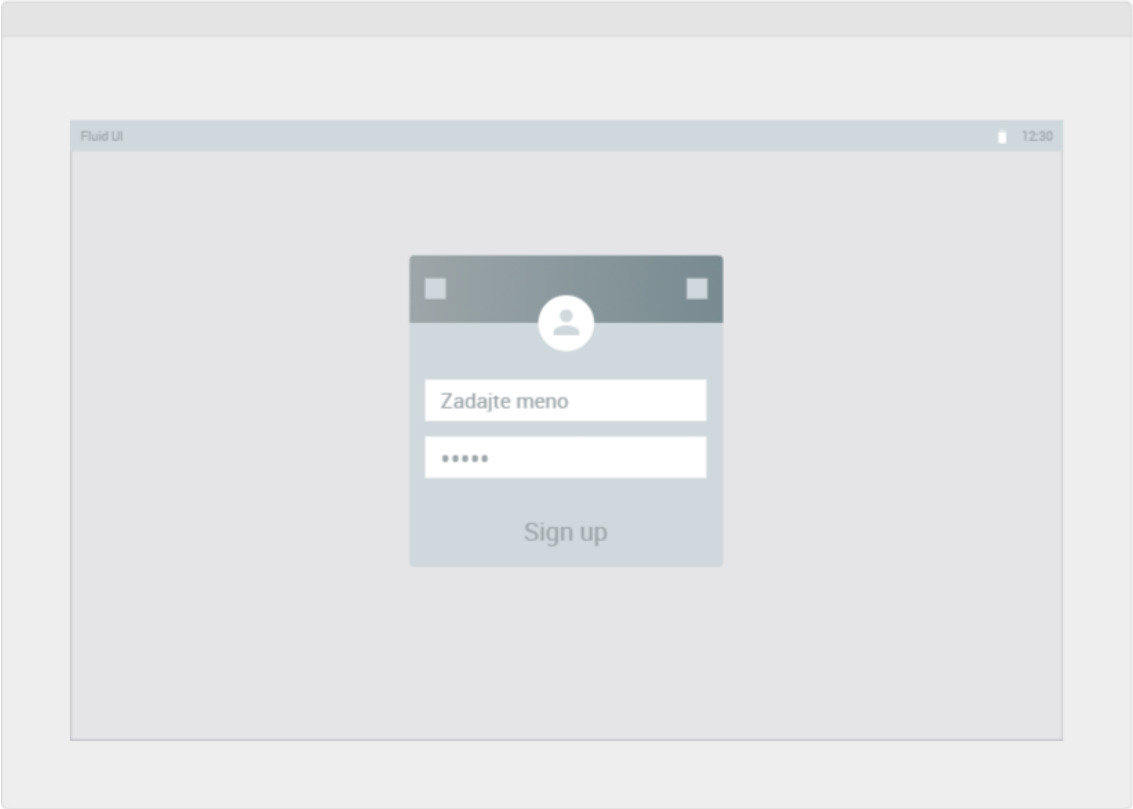
Poslednou časťou systému je mobilná Android aplikácia, ktorá bude pomáhať lekárom pri diagnostikovaní pacientov. Pacienti s ňou budú pracovať pri ich vyšetrení a budú na nej vypĺňať dotazníky, ktoré boli predtým vytvorené vo webovej aplikácii. Po vyplnení celého testu, aplikácia automaticky vyhodnotí ich výsledky a zobrazí výsledok vo forme získaných bodov. Výsledné body reprezentujú stav pacienta a lekár podľa nich určí diagnózu.

2.4.1 Popis procesov v mobilnej aplikácii

Kvôli väčšiemu pohodliu sa bude s aplikáciou pracovať na tabletoch, z toho dôvodu bude aplikácia primárne vyvíjaná na tieto typy zariadení z verziou Androidu 5.1, ktorý nesie pomenovanie **Lollipop**. V tejto podkapitole budú podrobne popísané jednotlivé procesy fungovania aplikácie.

Prihlásenie

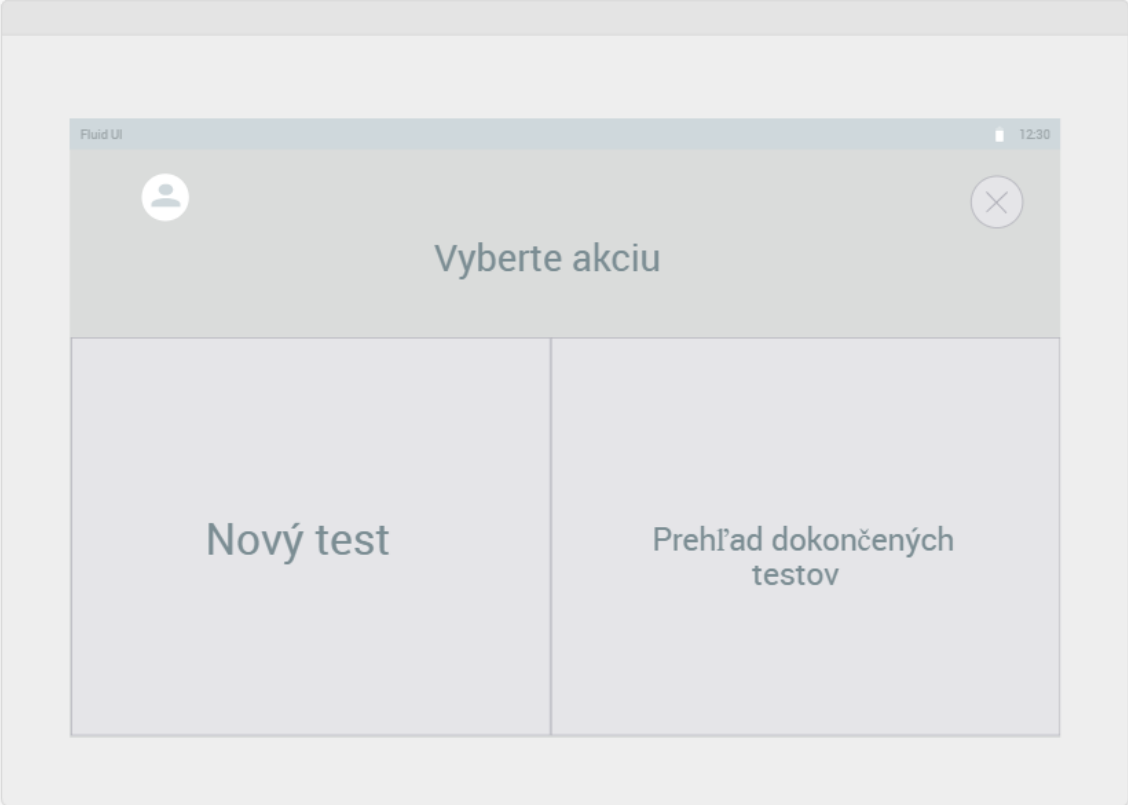
Po spustení aplikácie sa zobrazí obrazovka s prihlásením (obr. 2.9), ktoré je potrebné z dôvodu zamedzenia prístupu nepovoleným osobám k výsledkom už testovaných pacientov. Prístup do aplikácie by mali mať len lekári, ktorým boli vytvorené účty vo webovej aplikácii. Proces prihlasovania prebieha tak, že do políčok vyplnia ich prihlasovacie údaje, kde z hesla sa vypočíta jeho hash a ten sa s menom odosiela na server k overeniu. Ak bolo overenie úspešné, užívateľ môže pokračovať ďalej v aplikácii.



Obr. 2.9: Obrazovka prihlásenia.

Základná obrazovka

Na základnej obrazovke, ktorú môžeme vidieť na obr. 2.10 bude len jednoduchý výber dvoch akcií a dve ikony. Jedna na odhlásenie a ďalšia na prechod do nastavenia užívateľského účtu lekára.

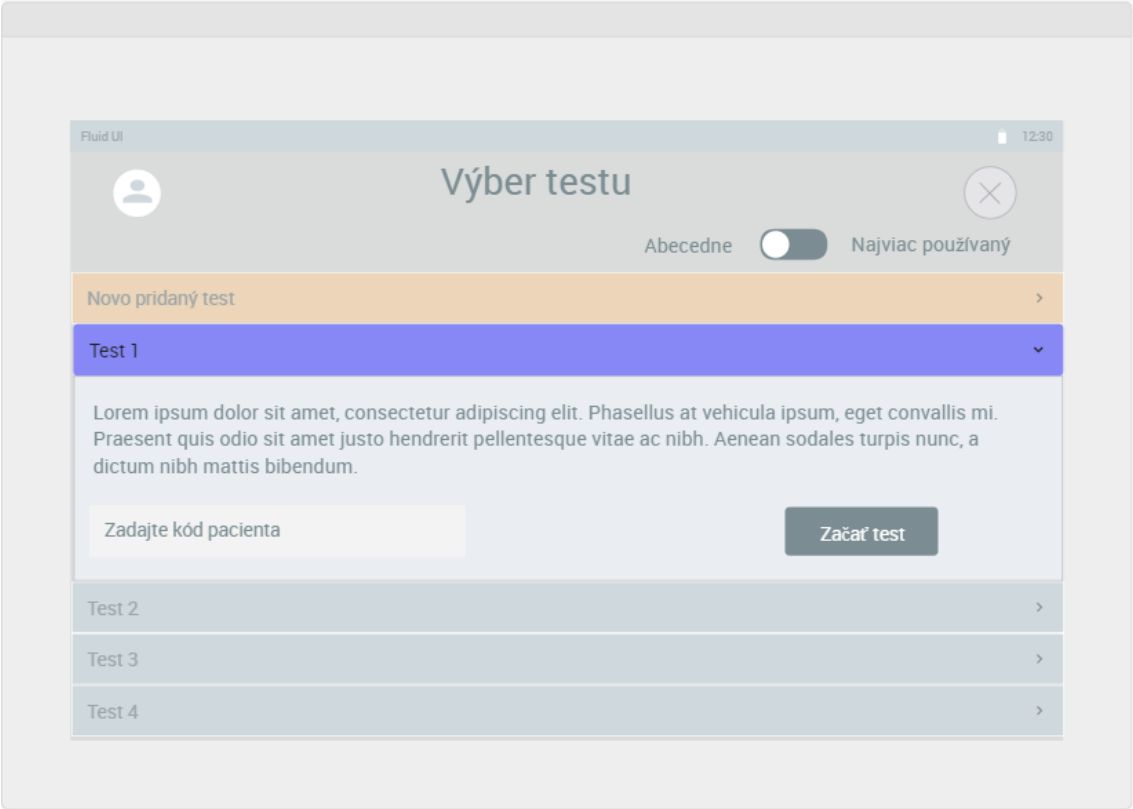


Obr. 2.10: Úvodná obrazovka s výberom akcie.

Obrazovka výberu nového testu

Po výbere akcie nového testu sa nám otvorí nová aktivita[19], kde sa nachádza zoznam existujúcich testov (viď. obr. 2.11). Testy môžu byť radené abecedne alebo podľa ich používanosti. V prípade, že vo webovej aplikácii sa vytvorí nový test, tak ten sa objaví na vrchu zoznamu a zvýraznení inou farbou. Pri nasledujúcom príchode na túto aktivitu, už bude radený podľa výberu radenia. Pri výbere niektorého testu, sa vyroluje popis testu, povinné pole na zadanie kódu pacienta a tlačidlo na začatie testu. K začatiu testu dochádza, len ak je vyplnené pole kódu pacienta.

[19] Aktivita je základný prvok v Androide pre vykreslovanie komponent.



Obr. 2.11: Obrazovka pri výbere nového testu.

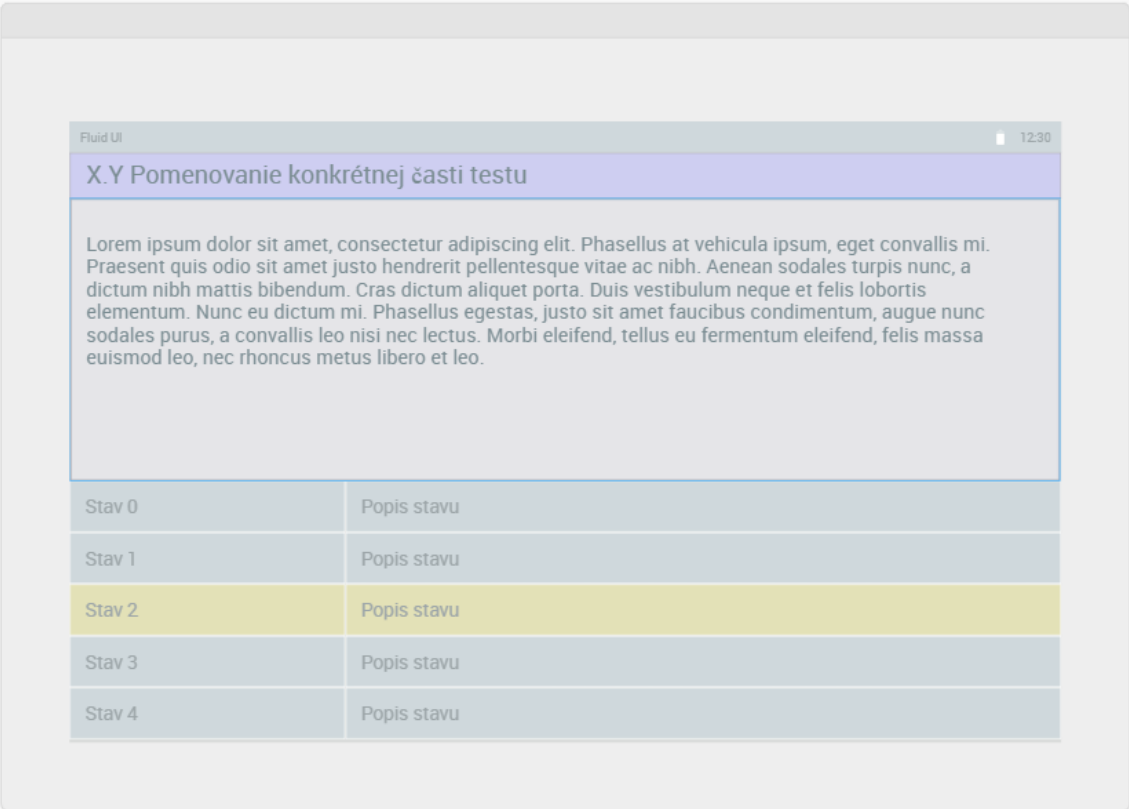
Priebeh testu

Po tom ako sa spustí test, sa postupne budú renderovať jeho jednotlivé časti, tak ako boli vytvárané vo webovej aplikácii. Na obr. 2.12 môžeme vidieť príklad, ako by mohla vyzerať jedna otázka z testu. V reálnej prevádzke však testy môžu vyzerať odlišne a to z dôvodu toho, že ten čo ich tvorí, môže vyberať z rôznych typov prvkov. Čo však bude pevnej štruktúry, tak to budú jednotlivé časti, čo musí obsahovať otázka. Týmito časťami sú:

- **Názov otázky**. Ten sa nachádza v hornej časti a okrem neho tam môžeme vidieť, aj veľké písmená X a Y. Tie v našom príklade znamenajú:
  1. **Písmeno X** označuje skupinu testovacích okruhov, ktoré sa týkajú jednej testovanej oblasti. V reálnom prípade by za písmeno X bolo dosadené číslo skupiny.
  2. **Písmeno Y** bude následne označovať konkrétnu otázku v skupine. Rovnako ako pre X aj za Y bude dosadené číslo.
- **Popis otázky**. Ten sa nachádza v strednej časti a popisuje čoho sa otázka týka.

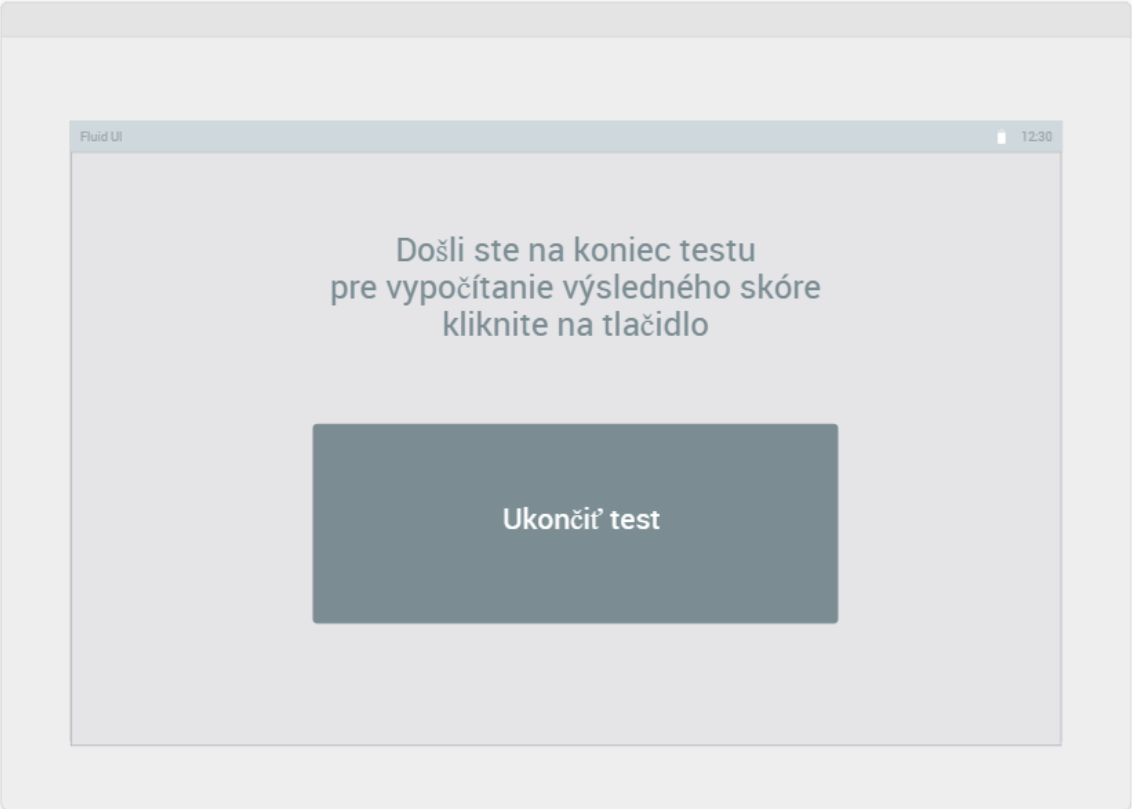
- **Aktívne prvky**. S týmito prvkami interaguje pacient a uchovávajú v sebe hodnotu, ktorá bola dopredu definovaná vo webovej aplikácii. Po výbere nejakej sa táto hodnota uloží pre túto otázku, ktorá bude potom započítaná do celkového hodnotenia testu.

Po dokončení konkrétnej otázky sa na ďalšiu prejde potiahnutím prstom po obrazovke do ľava. Opačným potiahnutím, do prava by ste sa vrátili o otázku naspäť. Tlačítkom späť na tablete by sa dalo odísť z testu aj v jeho priebehu a aplikácia by si pamätala stav, kde sme prestali.



Obr. 2.12: Príklad obrazovky ako by mohla vyzerať otázka v teste.

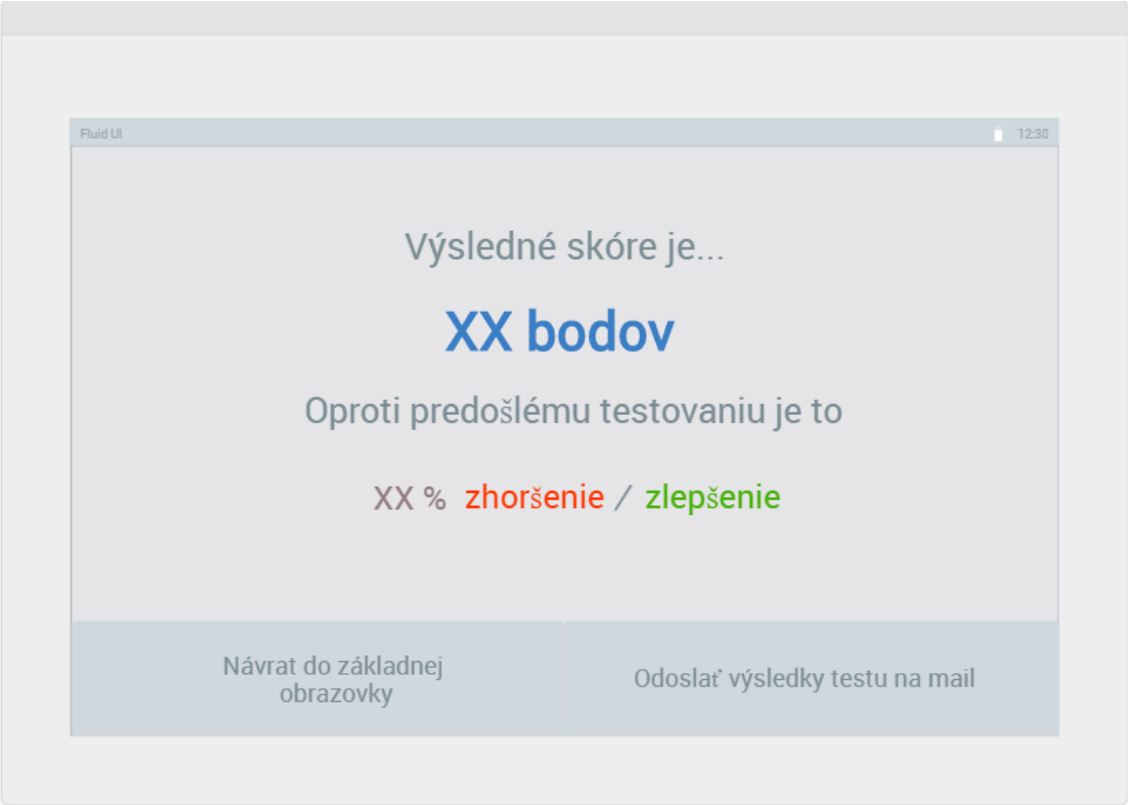
2.4.2 Koniec testu a výsledné skóre



Obr. 2.13: Obrazovka ukončenia testu.

Po dokončení všetkých otázok sa užívateľ dostane na obrazovku s oznámením o konci testu (viď. obr. 2.13). V tejto chvíli sa ešte môže vracať k otázkam a meniť svoje odpovede. Akonáhle klikne na tlačidlo **Ukončiť test**, test sa uzavrie a zobrazí sa mu obrazovka s vypočítaným skóre. Akonáhle sa skóre dopočíta, tak výsledok testu spolu s kódom pacienta, je uložený do databázy.

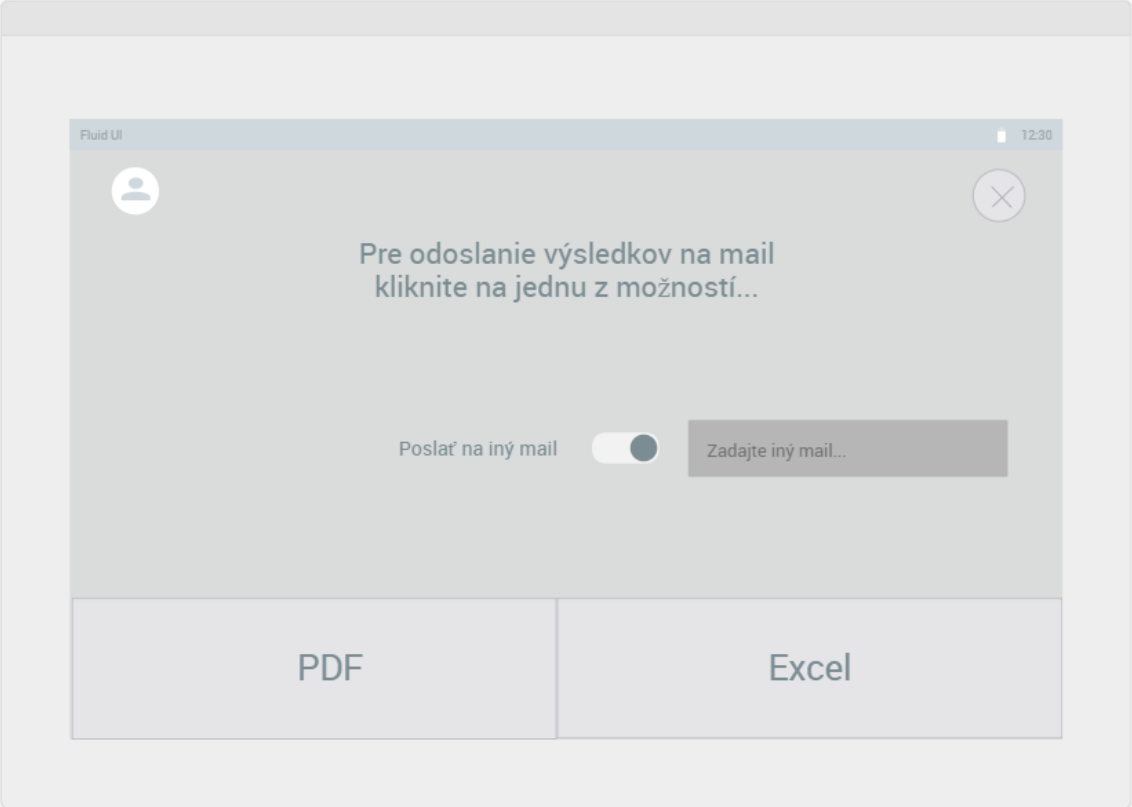
V prípade, že pacient už bol niekedy testovaný, tak sa porovná jeho posledný výsledok s aktuálnym, čo môžete vidieť na obr. 2.14. Z tejto obrazovky sa dá následne dostať, buď na základnú obrazovku alebo prípadne na obrazovku pre zaslanie výsledkov na e-mail.



Obr. 2.14: Obrazovka výsledného skóre.

2.4.3 Obrazovka pre odoslanie výsledkov na e-mail

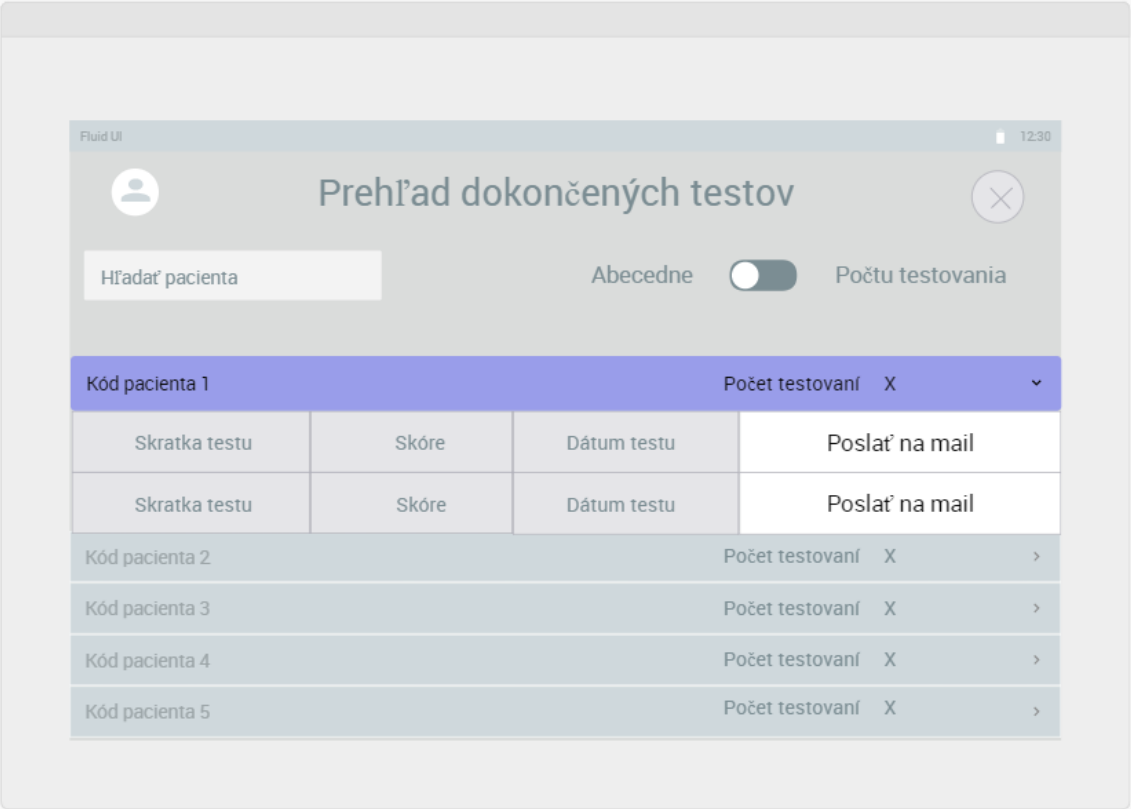
V prípade, že sa užívateľ rozhodol poslať testové výsledky pacienta aj na e-mail, má na výber v akom formáte ich chce poslať (viď. obr. 2.15). Prípadne má ešte možnosť zmeniť adresáta. Ak sa tak nerozhodne, výsledky sa zašlú na e-mail, ktorý má nastavený v jeho užívateľskom účte. Do nastavena jeho úžívateľského účtu sa dá dostať cez ikonu v ľavo hore. Po tom, čo užívateľ klikne na jednu možnosť pre odoslanie, výsledky sa pošlú na serverovú aplikáciu, ktorá jeho žiadosť spracuje a vráti mu výsledok operácie.



Obr. 2.15: Obrazovka odoslania výsledkov e-mailom.

2.4.4 Obrazovka s prehľadom dokončených testov

Poslednou dôležitou obrazovkou v Android aplikácii je prehľad dokončených testov jednotlivých pacientov. Po kliknutí na nejakého pacienta v zozname sa vyrolujú jeho dokončené testy, ktoré sú zobrazené vo forme mriežky. V jednotlivých stĺpcoch budú zobrazené hodnoty podľa obr. 2.16. V poslednom stĺpci je možnosť znova poslať výsledky testu na e-mail. Pacienti sú radený v zozname abecedne alebo podľa počtu testovaní. Okrem toho je tam možnosť si vyhľadať pacienta aj zadaním jeho kódu do políčka s vyhľadávaním.



Obr. 2.16: Obrazovka prehľadu dokončených testov.

2.4.5 Ukladanie výsledkov testov pacientov

Ako už bolo viackrát spomenuté, pacienti budú v aplikácii označovaný unikátnymi kódmi. Tieto kódy poznajú len lekári a je to hlavne kvôli ich anonymite a väčšej ochrane citlivých údajov. Ďalším bezpečným prvkom pri ochrane údajov bude, že výsledné dáta nebudú ukladané na serverovej aplikácii, pretože tá bude bežať na hardware mimo kontroly kliniky LF MU u sv. Anny, ale je to aj z dôvodu byrokratických obmedzení.

2.5 Cloudflare služba

Cloudflare služba je jednou z najväčších sieťových poskytovateľov obsahu[20] na svete, ktorá rieši problémy s latenciou ich tzv. **edge**[14] servermi, ktoré sú úmiestnené čo najbližšie ku klientskej stanici. To znižuje dobu nahrania stránky a obsahu. Týchto edge serverov má Cloudflare, niekoľko desiatok po celom svete a sú aj vstupným bodom do ich siete. Okrem toho edge servery znižujú riziko, že webová stránka spadne počas prenosovej špičky alebo sa stane nefunkčnou z dôvodu DDoS útokov.

Hlavným účelom použitia služby Cloudflare bolo rýchle nastavanie DNS (Domain Name System) záznamov a ich rýchle rozšírenie medzi DNS servery po svete[21]. Ďalšou vecou, kvôli ktorej bol Cloudflare použitý, je bezpečnosť. Cez ich webovú aplikáciu je veľmi jednoduché manažovať rôzne bezpečnostné nastavenia, ako napríklad manažment SSL certifikátov alebo prenosový firewall. Okrem toho ponúkajú ďalšie zaujímavé funkcie akými sú prezeranie štatistík o webovom prenose alebo návštevnosti.

2.5.1 Cloudflare a DNS manažment

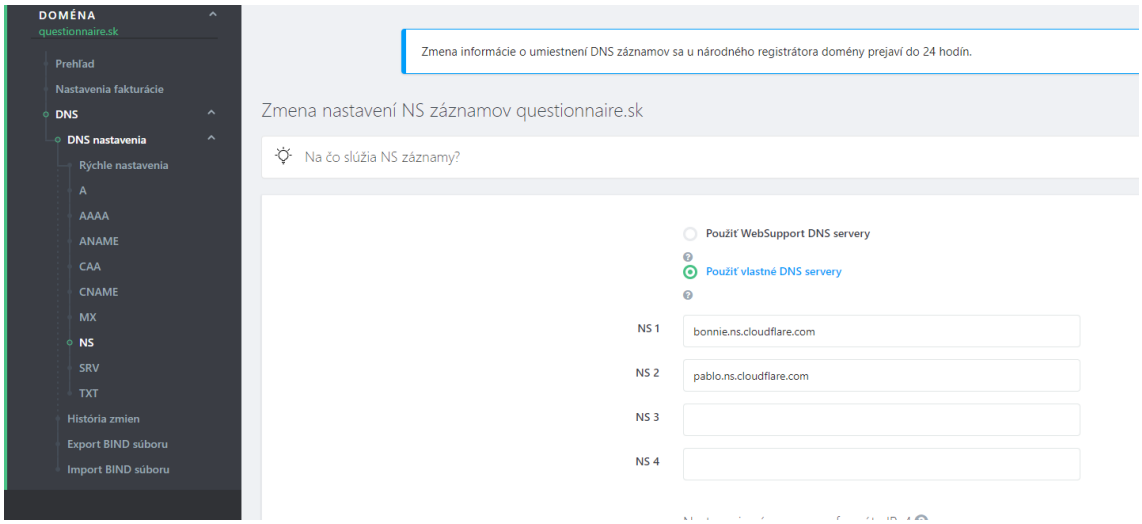
DNS znamená systém doménových mien a slúži na prekladanie IP adries na doménové mená. Má to výhodu hlavne v tom, že ľudia si nemusia pamätať IP adresy, ktoré sú ťažšie zapamätateľné ako doménové názvy alebo ak sa zmení IP adresa zariadenia na ktorom aplikácia beží, tak doména ostáva stále stejná. Tento proces prekladania nastáva medzi zadaním názvu domény úživateľom do prehliadača a získáním IP adresy od koreňového doménového servera (viď. [15]).

V našom prípade aby sme mohli používať našu webovú aplikáciu, tak sme najprv museli získať doménu. Doména sa dá získať od registrátorov domén. V našom prípade sme si zvolili ako registrátora domény službu websupport.sk, kvôli peknému užívateľskému rozhraniu a rýchlej registrácii .sk top level domény. Akonáhle bola doména zaregistrovaná, registrátor to musí oznámiť národnému registrátorovi pre .sk top level doménu, a tým je SK-NIC. Aby národný registrátor vedel, kde sa nachádza autoritatívny menový server pre naše DNS záznamy, kvôli smerovaniu prichádzajúcich požiadaviek pre danú adresu, bolo potrebné špecifikovať autoritatívny menový server v službe websupport (viď. 2.17).

Tieto autoritatívne menové servery sú v našom prípade servery od služby cloudflare s adresami **bonnie.ns.cloudflare.com** a **pablo.ns.cloudflare.com**. Uvádzajú sa dve kvôli redundancii, ak by jeden server nebol funkčný. Ďalším krokom je nastaviť v službe Cloudflare naše DNS záznamy. To môžete vidieť na obr. 2.18, kde je

[20] Content delivery network.

[21] Bližší popis bude v kodkapitole o DNS. 2.5.1

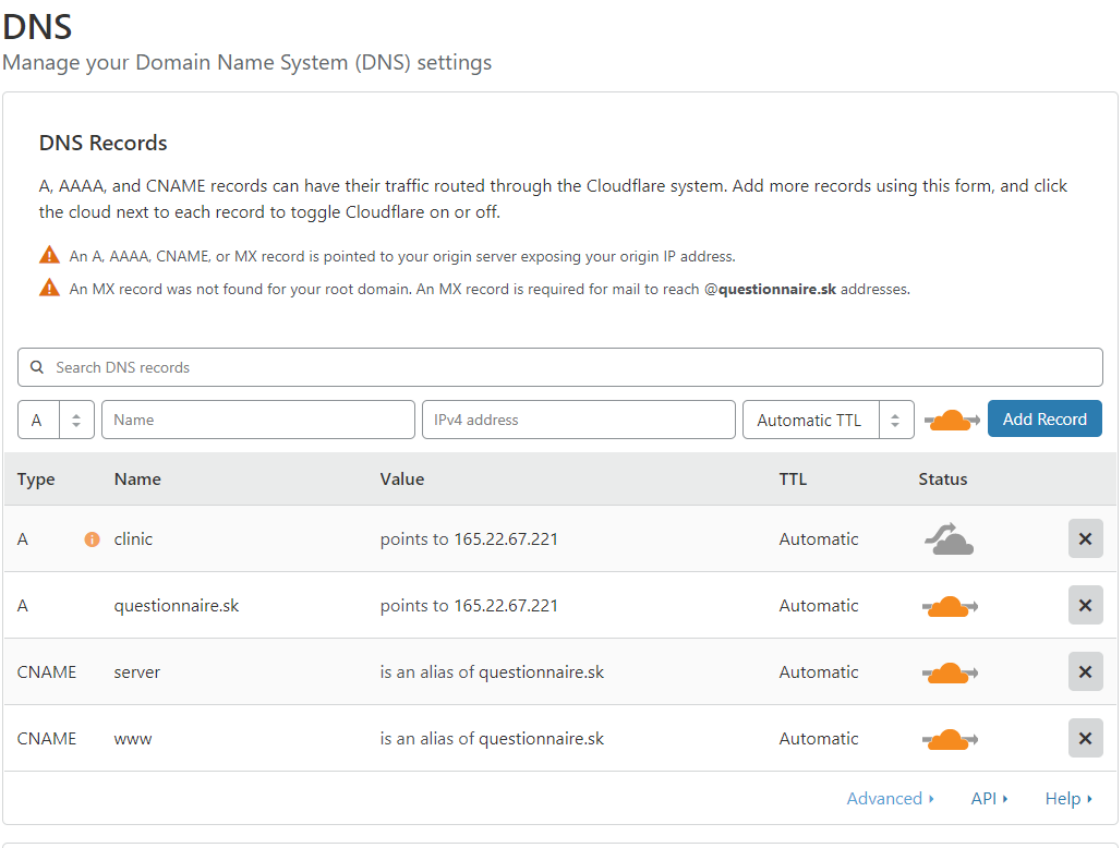


Obr. 2.17: Nastavenie autoritatívneho menového servera.

tabuľka so štyrmi riadkami. Prvé dve riadky sú záznamy typu **A**, ktoré sú najdôležitejším typom DNS záznamu. Slúžia na nastavenie konkrétnej IP adresy pre danú doménu. V našom prípade máme dve A záznamy a obe smerujú na IP adresu našeho cloudového poskytovateľa VPS. Dôležitý je však hlavne ten druhý s názvom questionnaire.sk, cez ktorý sa vedia užívatelia dostať na našu webovú aplikáciu. Prvý A záznam slúži iba pre administrátorské účely.

Zvyšné dve záznamy sú typu **CNAME**, ktoré slúžia k nasmerovaniu subdomény na cieľovú doménu typu A alebo inej subdomény. V našom prípade je to subdoména www.questionnaire.sk, ktorá smeruje na doménu questionnaire.sk. Je to hlavne z dôvodu toho, keď užívateľ zadá do prehliadača www.questionnaire.sk, aby DNS server vedel, kde má ďalej hľadať v záznamoch a mohol vrátiť IP adresu na ktorú smeruje A záznam. Poslednou vecou ktorá nás zaujíma pri nastavovaní DNS v službe Cloudflare je či HTTP požiadavky majú byť smerované, cez Cloudflare servery. To môžeme vidieť v stĺpci **Status**, kde šípka prechádzajúce cez oranžový oblak znamená, že všetky požiadavky smerujú cez cloudflare, ktorá je vlastne v tomto prípade proxy pre našu skutočnú adresu webového servera. Táto funkcia má aj bezpečnostný charakter, pretože Cloudflare zastavuje škodlivý prenos tým, že analyzuje potencionálne hrozby v návštevníckych http požiadavkach podľa rôznych parametrov[16]:

- IP adresa návštevníka.
- Dotazované zdroje.
- Obsah http požiadavky a jej frekvencia.
- A pravidlá firewallu definované zákazníkom.



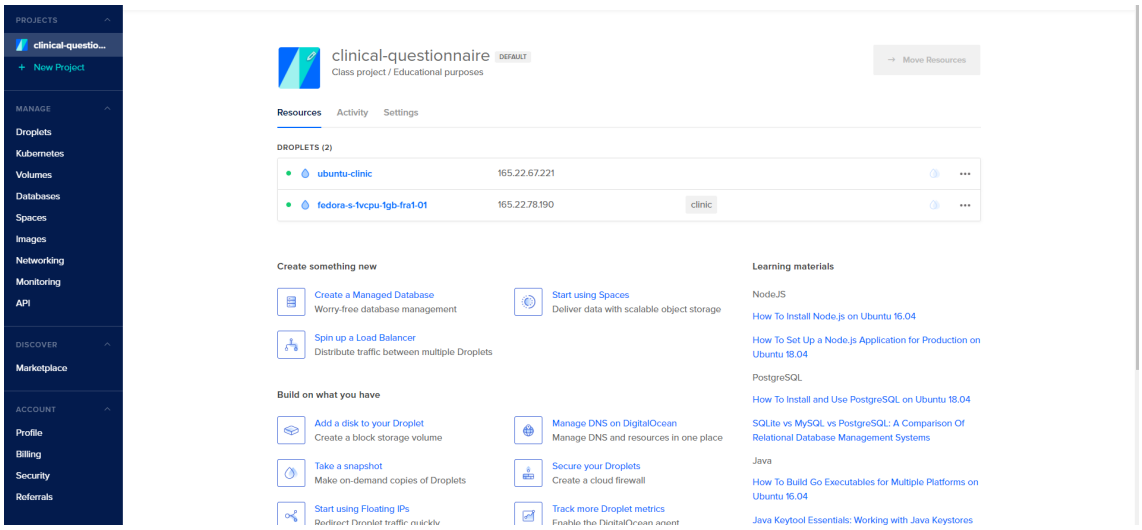
Obr. 2.18: Nastavenie DNS záznamov v Cloudflare.

2.6 Digital Ocean VPS

Aby sme mohli niekde našu serverovú aplikáciu hostovať, poskytovať zdroje pre webovú a mobilnú aplikáciu, a taktiež ukladať niekde dáta týchto aplikácií, tak nato potrebujeme mať vybodované prostredie. Zvyčajne je toto prostredie formou virtuálnych privátnych serverov, ktoré bežia na virtualizovanom hardware[22] prevádzkované hostingovým poskytovateľom. Výhodou VPS, je že používateľ má vyhradený server s výpočtovým výkonom len pre neho. Jedným z takýchto VPS hostingových poskytovateľov aj cloudová služba Digital Ocean, ktorá ponúka cloudové výpočetné služby cez ich niekoľko desiatok cloudových centier po svete. Hlavným dôvodom prečo bola táto služba vybratá, je ich jednoduché a zrozumiteľné užívateľské rozhranie, ale taktiež ich sada návodov, ktoré zjednodušujú nastavenie prostredia pre spustenie serverových aplikácií.

Na obr. 2.19 môžeme vidieť toto užívateľské rozhranie, kde v strede obrazovky vidieť už vytvorené privátne virtuálne stroje. Tieto virtuálne stroje nazvali tzv. droplety, ktoré sú virtuálne zariadenia na základe linuxového operačného systému. Pre

[22]Zariadenie, ktoré dovolí spúšťať viacero operačných systémov zároveň.



Obr. 2.19: Užívateľské rozhranie služby Digital Ocean.

našu serverovú aplikáciu bol vybraný operačný systém Ubuntu verzie 18.04 x64. Po nainštalovaní systému je potom už len na užívateľovi čo všetko si na systém nainštaluje a čo tam pobeží. Na náš systém sa pripájame pomocou protokolu SSH (secure shell), ktorý nám zabezpečuje komunikáciu po sieti, medzi klientským zariadením a serverom. V prvom rade, je vždy potreba aktualizovať a upgradovať už súčasné nainštalované balíčky hlavne z dôvodu bezpečnosti. V ďalšom kroku sme potrebovali doinšťalovať 3 veci:

- **Java Runtime Environment**. Je program, ktorý slúži na spustenie javovských aplikácií.
- **PostgreSql**. Je relačná databáza, v ktorej budú ukladané vytvorené definície testov z webovej aplikácie.
- **Webový server NGINX**. Je to program, ktorý vie fungovať ako webový server, reverzná proxy, cache alebo load balancing[23].

2.7 NGINX webový server a proxy

NGINX je program s otvoreným zdrojovým kódom, ktorý bol vytvorený aby vyriešil tzv. **C10K** problém (viď. [17]), ktorý mali v tej dobe už existujúce webové servery, ktoré museli zvládnuť nápor desaťtisíc súbežných sieťových spojení v jeden čas. Okrem toho, že je to jeden z najrýchlejších webových serverov súčastnosti, má aj iné dôležité funkcie, kvôli ktorým sme ho použili. Tými sú hlavne reverzná proxy2.7.1 alebo ľahšie nastavenie https protokolu a výmeny certifikátov medzi Cloudflare službou a NGINX serverom. Dosť často sa používa aj na optimalizáciu v doručovaní

[23]Vyrovnávač záťaže.

statického obsahu kvôli jeho lepšej efektivite.

2.7.1 Reverzná proxy

Reverzná proxy je server, ktorý je umiestnený medzi internou serverovou aplikáciou a klientskou aplikáciou, ktorá smeruje HTTP požiadavky klienta na vhodnú aplikáciu. Aj keď mnoho interných aplikácií, tak ako aj naša vie fungovať ako webový server, keďže vnútry je embedovaný Tomcat server[18]. Tieto aplikačné servery však predvolene fungujú na iných portoch ako sú tie štandardné pre webové služby, ktorými sú port 80 pre HTTP protokol a 443 zabezpečený HTTPS protokol. Toto je hlavne z bezpečnostných dôvodov, keďže porty 80 a 443 patria do privilegovaných portov pod číslo 1024 a tieto porty nie sú povolené, aby boli použité bežnými používateľmi v systéme. Tomuto pojmu v počítačovej bezpečnosti sa hovorí **separácia privilégií**[19] a je hlavne používaná na zmiernenie možného zneužitia zraniteľností v aplikácii. Normálne aplikácie by totiž nemali byť spustené pod administrátorskými právami, kvôli zníženiu šance útočníka na eskalovanie na administrátorské práva. Ďalšiu vec, ktorú rieši nginx ako proxy, je už spomenutý problém, že Tomcat používa predvolene port 8080, ktorý nie je štandardom pre webovú službu, a tým pádom by sa k nemu muselo pristúpiť cez adresu http://questionnaire.sk:8080 a toto nie je správne z dôvodu bezpečnosti, ale taktiež bežný užívatelia nemusia nájsť našu webovú aplikáciu, keďže by si museli pamätovať presnú url s portom.

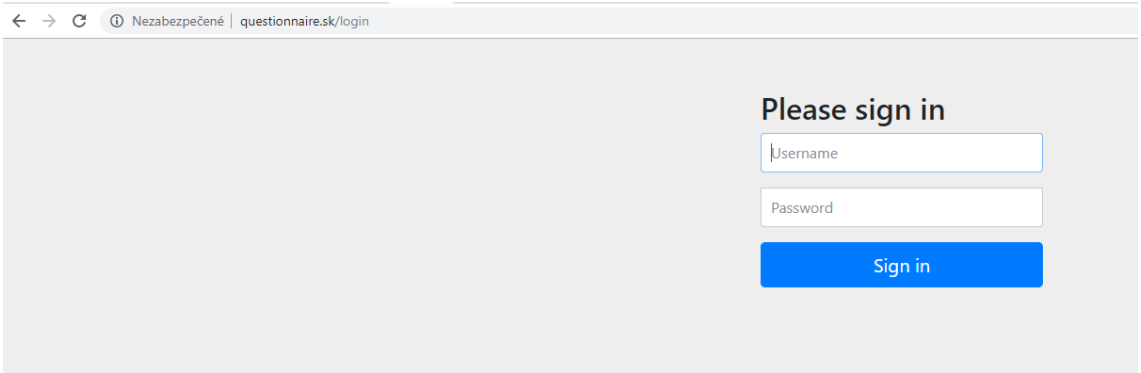
2.7.2 Základná konfigurácia NGINXu ako proxy

Základná konfigurácia pre nastavenie NGINxu ako proxy pre našu serverovú aplikáciu vyzerá takto.

Výpis 2.6: Ukážka nastavenia http proxy .

```
1  server {
2      listen 80;
3      listen [::]:80;
4      server_name questionnaire.sk;
5
6      location / {
7          proxy_set_header X-Forwarded-For
               $proxy_add_x_forwarded_for;
8          proxy_set_header X-Forwarded-Host $host;
9          proxy_set_header X-Forwarded-Server $host;
10         proxy_pass http://127.0.0.1:8080;
11     }
12 }
```

V tejto konfigurácii sme povedali NGINXu aby počúval na porte 80 a príjimal požiadavky pre obe IPv4 a IPv6 adresy a názov hosťa, ktorý musí byť questionnaire.sk. V ďalšej časti, *location* blok špecifikuje prefix pre URI požiadavku. Táto lokácia poskytuje najkratší prefix. Direktíva *proxy_pass* nastaví protokol a adresu zastupovaného servera na ktorý má potom presmerovávať požiadavky. Ďalšia direktíva *proxy_set_header* špecifikuje aby informácie o hostovi pripojené do hlavičky požiadavky pre zastupovaný server. Takže do našej javovskej serverovej aplikácii prí http požiadavky aj s týmito informáciami.



Obr. 2.20: Spustená aplikácia v nezabezpečenom móde.

Momentálne ako môžeme vidieť na obr. 2.20, naša webová aplikácia je teraz prístupná cez prehliadač v nechránenom HTTP protokole, a preto v ďalšej kapitole si povieme čo treba spraviť aby mohla fungovať pod zabezpečeným HTTPS protokolom.

3 ANALÝZA BEZPEČNOSTNÝCH RIZÍK A ICH ELIMINÁCIA

Doteraz naše aplikácie fungovali nezabezpečene a celá komunikácia medzi klientskými aplikáciami zo serverom bola nešifrovaná a ktokoľvek ju mohol ľahko zachytiť. Aby sme mohli tomuto zabrániť musíme túto komunikáciu chrániť šifrovaním. K tomu slúži HTTPS protokol, ktorý zabraňuje odpočúvaniu prenosu alebo inak presne nazvaným Man-in-the-middle útokom. HTTPS protokol je vlastne kombináciou štandardného HTTP protokolu a bezpečnostného protokolu SSL/TLS (Secure Socket Layer/Transport Layer Security)[20].

3.0.3 Čo znamená SSL/TLS protokol?

Transport Layer Security (TLS) je protokol, ktorý šifruje dáta prenášané cez internet. Tento protokol vzišiel z jeho predchodcu Secure Socket Layer (SSL), ktorý sa ako prvý adaptoval a bol široko používaný pre šifrovanie webového prenosu, a aby opravil jeho najväčšie bezpečnostné chyby.

Protokol SSL je založený na asymetrickej kryptografii, kde je používaný pár kľúčov: verejný a súkromný. Tieto kľúče sú reprezentované dlhým reťazcom náhodných znakov.

Aby boli webové stránky a dáta užívateľov v nich bezpečné je potrebné overiť vlastníctvo stránok a zábraniť tak útočníkom vytvárať falošné stránky a získať dáta týchto užívateľov. Ktomu sú potrebné SSL certifikáty.

SSL certifikáty

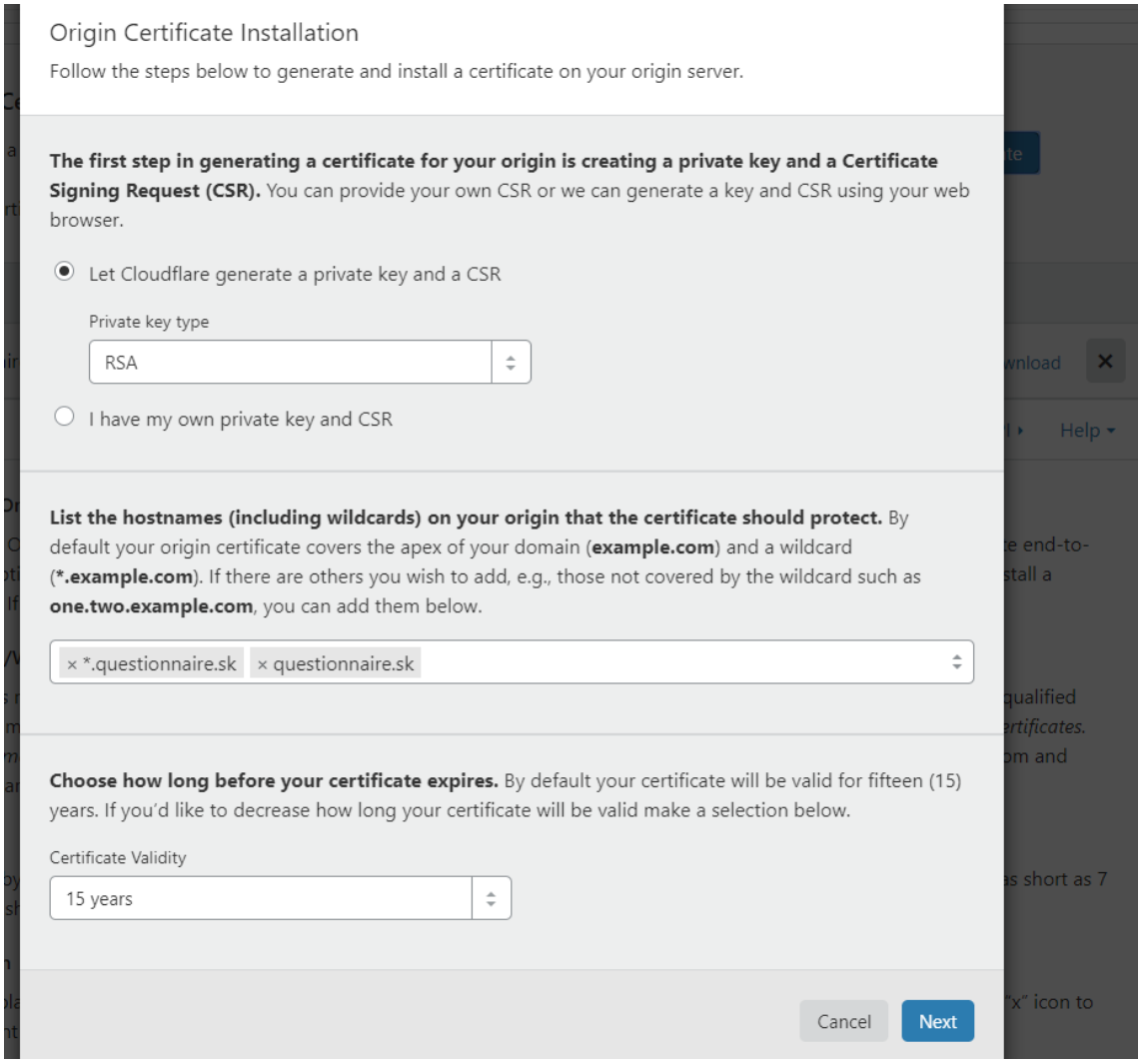
Certifikáty obsahujú súbory s informáciami o vlastníkovi webovej stránky a verejný kľúč z asymetrického páru kľúčov použitého kryptografického algoritmu. Ktomu aby bol tento certifikát vlastníka stránky dôveryhodný je potrebné aby bol digitálne podpísaný **certifikačnou autoritou (CA)**, ktorá overí, že informácie v certifikáte vlastníka sú správne. Toto je kvôli tomu, že operačné systémy a webové prehliadače obsahujú zoznam certifikačných autorít, ktorým implicitne veria. V prípade, že webová stránka obsahuje certifikát, ktorý je podpísaný nedôveryhodnou certifikačnou autoritou, tak prehliadač varuje návševníka, že niečo nie je v poriadku.

Cloudflare ako certifikačná autorita

V našom prípade je certifikačnou autoritou služba Cloudflare, ktorá je voči nám dôveryhodná tretia strana, ktorá vygenerovala a vydala SSL certifikáty. Certifikačná

autorita potom tieto certifikáty podpíše z ich súkromným kľúčom. Akonáhle sú certifikáty vydané je potrebné ich nahrať server užívateľa, ktorý hostuje webovú stránku.

Tento proces vydávania certifikátov Cloudflarom je veľmi jednoduchý a je vykonávaný cez ich užívateľské rozhranie, podobne ako to bolo pri nastavovaní DNS2.5.1.



Obr. 3.1: Vytvorenie certifikátov.

Na obr. 3.1 môžeme vidieť, že potrebujeme vybrať možnosti akým kryptografickým algoritmom budú certifikáty generované, v našom prípade to bude RSA. Ďalej je potrebné zadať pre ktoré domény majú byť certifikáty vytvorené, v našom prípade je to pre doménu questionnaire.sk a spolu s ňou všetky subdomény. A v poslednom rade, je potrebné určiť, dokedy majú byť certifikáty platné.

Po kliknutí na tlačidlo next, sa vygenerujú privátny a verejný kľúče, ktoré je potrebné nahrať na našu VPS. Tieto kľúče musíme nahrať do zložky /etc/nginx/conf.d/ aby si ich vedel NGINX server načítať.

Výpis 3.1: Ukážka nastavenia https proxy .

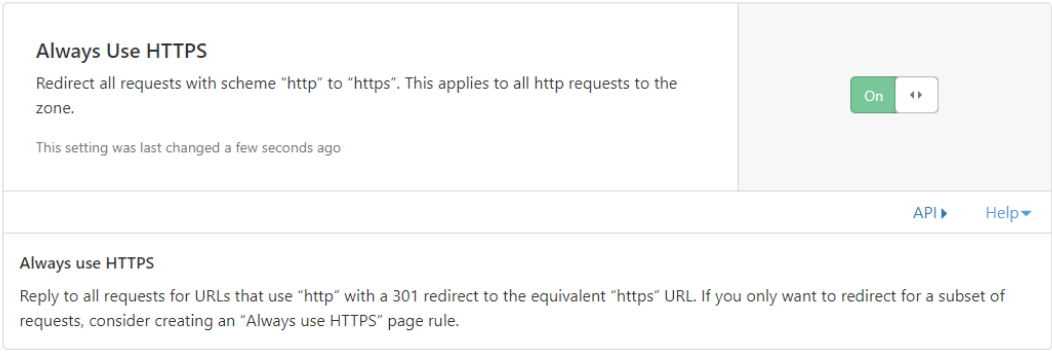
```
server {
    listen 80;
    listen [::]:80;
    server_name questionnaire.sk;
    return 302 https://$server_name$request_uri;
}

server {

    listen 443 ssl http2;
    listen [::]:443 ssl http2;

    ssl        on;
    ssl_certificate         /etc/nginx/conf.d/cert.pem;
    ssl_certificate_key     /etc/nginx/conf.d/key.pem;

    server_name questionnaire.sk;

    location / {
        proxy_pass http://127.0.0.1:8090;
        proxy_set_header X-Forwarded-For
           $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Port $server_port;
    }
}
```

V prvom bloku definujeme, že prípadné http požiadavky má NGINX presmerovať na adresu https://questionnaire.sk, kde *$request_uri* je suffix pre ľubovoľný zvyšok url. V ďalšom bloku definujeme, aby NGINX počúval na porte 443 a prikážame aby použil http2 protokol pre kominikáciu s prehliadačom. Direktívy *ssl_certificate* a *ssl_certificate_key*, definujú kde má nájsť tieto certifikát a preň. Zvyšné časti sú rovnaké ako pre predchádzajúci blok pre HTTP2.7.2.

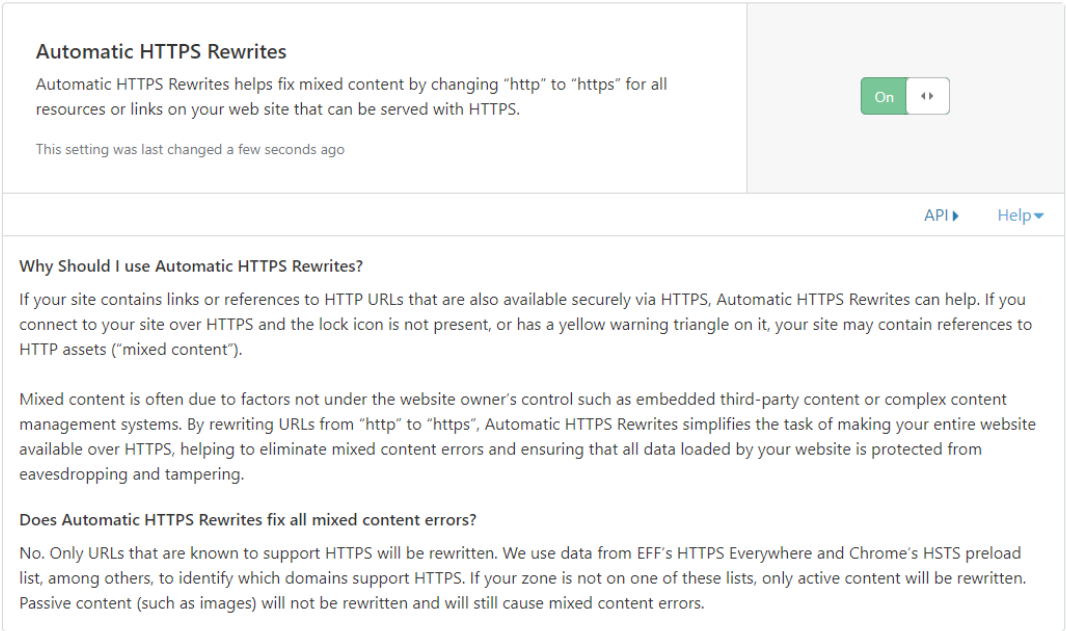
Posledné dve zaujímavé funkcie, ktoré nám ponúka Cloudflare a malo by zmysel ich nastaviť sú:

- **Always Use HTTPS**. Toto nastavenie prikáže, aby boli všetky HTTP požiadavky presmerované na HTTPS3.2.
- **Automatic HTTPS Rewrites**. Táto funkcia zas automaticky opraví zmiešaný obsah, tzn. že použije HTTPS komunikáciu aj pre zdroje, ktoré nie sú

obslúžené HTTPS protokolom3.3.



Obr. 3.2: Nastavenie "Always Use HTTPS"



Obr. 3.3: Nastavenie "Automatic HTTPS Rewrites"

3.0.4 Nastavenie cloudového firewallu v Digital Ocean

Posledným bezpečnostným nastavením ktoré si ukážeme, je na strane cloudového poskytovateľa a tým je sieťový firewall. Firewally slúžia ktomu aby boli vstupnou bariérou medzi našim serverom a vonkajším svetom, kde ich úlohou je chrániť pred vonkajšími útokmi. Tieto firewally môžu byť umiestenené na hostovi, vtedy je napr. použítá služba **IPTables**[21], ktoré funguje pre server, alebo je definovaný na základe siete, v tom prípade blokuje prenos už predtým ako dosiahne na konkrétne zariadenia.

Firewall rules control what inbound and outbound traffic is allowed to enter or leave a Droplet.

Inbound Rules

Set the Firewall rules for incoming traffic. Only the specified ports will accept inbound connections. All other traffic will be blocked.

Type	Protocol	Port Range	Sources
SSH	TCP	22	All IPv4 All IPv6
HTTP	TCP	80	All IPv4 All IPv6
HTTPS	TCP	443	All IPv4 All IPv6

New rule

Outbound Rules

Set the Firewall rules for outbound traffic. Outbound traffic will only be allowed to the specified ports. All other traffic will be blocked.

Type	Protocol	Port Range	Destinations	
ICMP	ICMP		All IPv4 All IPv6	More
All TCP	TCP	All ports	All IPv4 All IPv6	More
All UDP	UDP	All ports	All IPv4 All IPv6	More

New rule

Obr. 3.4: Nastavenie sieťového firewallu pre VPS.

Na obr. 3.4 je možné vidieť nastavenie tákehoto sieťového firewallu, ktorý je na úrovni Digital ocean. Cez toto rozhranie vieme nastaviť pravidlá pre prichádzajúci a odchádzajúci sieťový prenos. Naše aplikície komunikujú zo serverom len cez webové porty, a preto sme povolili aby k nám vstupoval len HTTP a HTTPS prenos. Okrem týchto dvoch je tam však aj pravidlo pre SSH, ktoré je potrebné povoliť na to, aby sme mohli túto VPSku jednoduchšie spravovať. Čo sa týka vystupných pravidiel, tak u tých povolíme všetky TCP a UDP prenosy a iba ICMP, ktorý slúži hlavne na sieťové účely.

4 ZÁVĚR

Cieľom práce bolo navrhnúť a implementovať systém, ktorý by pomohol k zefektívneniu a digitalizovaniu procesu testovania pacientov s neurologickými ochoreniami. Na základe tohto problému bola navrhnutá architektúra systému, ktorá by mala umožňiť efektívnejšiu prácu pri vytváraní testovacích dotazníkov a ich následným testovaním pacientov. Pri návrhu bol kladený aj dôraz na bezpečnosť systému a ochrane dát pacientov. V celkovej architektúre boli využité aj služby komerčných riešení, ktoré sú v súčasnej dobe už neoddeliteľnou časťou softwarových projektov.

V prvej kapitole práce bolo vysvetlené k čomu slúžia klinické testy, aké je ich využitie, a taktiež proces, ako sú pacienti testovaní. V druhej kapitole, bola popísaná a navrhnutá celková architektúra systému, spolu s detailným opisom jednotlivých častí systému, ich implementácie a opisom technológií ktoré v nich boli použité. V poslednej kapitole bola analyzovaná oblasť bezpečnosti systému, v ktorej boli detekované zraniteľnosti a následná implementácia riešení, ktoré tieto problémy eliminovali.

V systéme boli použité rôzne technológie, ktoré pokrývajú všetky oblasti aplikačného vývoja, keďže systém sa skladá z mobilnej, backendovej a frontendovej časti. Na tieto aplikácie boli použité jazyky: Java verzie 8 na vývoj backendovej aplikácie, ďalej webový framework Angular verzie 5, ktorý bol použitý na vývoj frontendovej časti a poslednou technológiou, ktorá bola použitá na vývoj bol framework Android na mobilnú aplikáciu. Okrem týchto technológií boli použité aj služby komerčných riešení, a to konkrétne služba Cloudflare, ktorá nám pomohla s rýchlym nasadením systému do ostrej prevádzky a taktiež poskytla dôležité funkcie na zabezpečenie systému. Ďalšou službou bol cloudový poskytovateľ virtuálnych prostriedkov, ktorým je Digital Ocean. U tohto poskytovateľa sme nasadili finálne riešenie, kde sme si prenajali virtuálne prostriedky ich datového centra aby sme mohli spustiť serverovú aplikáciu, bez ktorej by mobilná a webová aplikácia nemohli správne fungovať. Systém na ktorom bežala serverová aplikácia bol linuxový operačný systém Ubuntu verzie 18.04. Na tomto operačnom systéme sme nainštalovali a nakonfigurovali aj webový server NGINX, ktorým sme vyriešili niektoré problémy s bezpečnosťou, ktorých riešenie by v inakšom prípade, zabralo oveľa viac času.

Hlavným prínosom celkovej práce bolo zoznámenie sa a vyskúšanie si rôznych technológií, ktoré boli použité na vyriešenie problémov v konkrétnom zadaní. Okrem toho aj potreba analyticky myslieť, aby sme prišli s komplexným riešením, ktoré sa na začiatku zdalo byť priamočiare.

Súčasná implementácia je v stave, ktoré je funkčné, ale potrebovalo by ešte nejaký čas, aby bolo dotiahnuté do lepšieho stavu. Hlavne na mobilnej aplikácii neboli dotiahnuté úplne všetky časti a aj dizajn by si pýtal viac času. Bohužiaľ tento čas musel byť použitý aj na vývoj zvyšných častí systému, bez ktorých by mobilná

aplikácia nefungovala. Ako to už ale býva s vývojom softwérových riešení, vždy sa nájde niečo, čo by sa dalo zlepšiť.

LITERATÚRA

[1] ZIEMOŃSKI, Grzegorz.: *Layered Architecture Is Good.* In: Dzone.com [online]. Grzegorz Ziemoński 2017 [cit. 2017-12-14]. Dostupné z URL: <https://dzone.com/articles/layered-architecture-is-good>.

[2] NEUHAUS, Jens.: *Angular vs. React vs. Vue: A 2017 comparison.* In: Medium.com [online]. Palo Alto: Jens Neuhaus 2017 [cit. 2017-12-09]. Dostupné z URL: <https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176>.

[3] *DRAFT: JSX Specification.* [online]. Kalifornia, USA: Facebook Inc 2014 [cit. 2017-12-09]. Dostupné z URL: <https://github.com/facebook/jsx>.

[4] SCHWARZMÜLLER, Maximilian.: *SPAs vs MPAs/MVC - Are Single Page Apps always better?* In: Youtube.com [online]. Nemecko: Maximilian Schwarzmüller 2017 [cit. 2017-12-07]. Dostupné z URL: <https://www.youtube.com/watch?v=F_BYg2QGsC0>.

[5] *Neurologie pro praxi: Screeningové škály pro hodnocení demence* [online] Olomouc: Solen, 2011, **2011**(12) [cit. 21. 11. 2017]. ISSN 1803-5280. Dostupné z URL: <https://www.neurologiepropraxi.cz/pdfs/neu/2011/92/11.pdf>.

[6] *HTTP request methods.* [online]. Florian Scholz 2017 [cit. 2017-12-14]. Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>.

[7] SKÓLSKI, Paweł.: *Single-page application vs. multiple-page application.* In: Neoteric.eu [online]. Gdańsk: Paweł Skólski 2016 [cit. 2017-12-07]. Dostupné z URL: <https://neoteric.eu/single-page-application-vs-multiple-page-application?utm_source=medium.com&utm_medium=social&utm_content=neo&utm_campaign=blog>.

[8] *TypeScript Language Specification.* [online]. Redmond, USA: Microsoft Corporation 2016 [cit. 2017-12-09]. Dostupné z URL: <https://github.com/Microsoft/TypeScript/blob/master/doc/spec.md>.

[9] WYSE, Josh.: *WHY JSON IS BETTER THAN XML.* In: Blog.cloud-elements.com [online]. SAN FRANCISCO: Josh Wyse 2014 [cit. 2017-12-09]. Dostupné z URL: <https://blog.cloud-elements.com/json-better-xml>.

[10] BUGAYENKO, Yegor.: *Stop Comparing JSON and XML.* In: Yegor256.com [online]. Palo Alto: Yegor Bugayenko 2015 [cit. 2017-12-09]. Dostupné z URL: <http://www.yegor256.com/2015/11/16/json-vs-xml.html>.

[11] *Spring Security Reference: 6.3 Java Configuration and Form Login* [online]. San Francisco, USA: Ben Alex, Pivotal Software c2004-2017 [cit. 2019-05-25]. Dostupné z URL: <https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#jc-form>.

[12] *Routing & Navigation* [online]. Mountain View, USA: Google c2010-2019 [cit. 2019-05-25]. Dostupné z URL: <https://angular.io/guide/router>.

[13] INTERNET ENGINEERING TASK FORCE (IETF). RFC 4122:*A Universally Unique IDentifier (UUID) URN Namespace* [online]. Edited by P. Leach. July 2005 [cit. 2019-05-25]. Dostupné z URL: <https://www.ietf.org/rfc/rfc4122.txt>.

[14] *What is a CDN Edge Server?* [online]. San Francisco, USA: Cloudflare [cit. 2019-05-25]. Dostupné z URL: <https://www.cloudflare.com/learning/cdn/glossary/edge-server/>.

[15] *What Is DNS? | How DNS Works: What are the steps in a DNS lookup?* [online]. San Francisco, USA: Cloudflare [cit. 2019-05-26]. Dostupné z URL: <https://www.cloudflare.com/learning/dns/what-is-dns/>.

[16] *Cloudflare Support | How does Cloudflare work?* [online]. San Francisco, USA: Cloudflare [cit. 2019-05-26]. Dostupné z URL: <https://support.cloudflare.com/hc/en-us/articles/205177068-How-does-Cloudflare-work->.

[17] *C10k problem* [online]. San Francisco, USA: Wikimedia Foundation 2019 [cit. 2019-05-26]. Dostupné z URL: <https://en.wikipedia.org/wiki/C10k_problem>.

[18] *Embedded Web Servers* [online]. San Francisco, USA: Phillip Webb, Pivotal Software c2012-2018 [cit. 2019-05-26]. Dostupné z URL: <https://docs.spring.io/spring-boot/docs/current/reference/html/howto-embedded-web-servers.html>.

[19] PROVOS, Niels, Markus FRIEDL a Peter HONEYMAN.: *Preventing Privilege Escalation* [online]. Michigan, USA: Wikimedia Foundation 2017 [cit. 2019-05-26]. Dostupné z URL: <http://niels.xtdnet.nl/papers/privsep.pdf.>. University of Michigan.

[20] INTERNET ENGINEERING TASK FORCE (IETF). RFC 2818:*HTTP Over TLS* [online]. Edited by E. Rescorla. May 2000 [cit. 2019-05-26]. Dostupné z URL: <https://tools.ietf.org/html/rfc2818>.

[21] *How the Iptables Firewall Works* [online]. San Mateo, USA: Justin Ellingwood, DigitalOcean, LLC [cit. 2019-05-26]. Dostupné z URL: <https://www.digitalocean.com/community/tutorials/how-the-iptables-firewall-works>.

ZOZNAM PRÍLOH

A ZOZNAM PRÍLOH

- Obsah CD - praktická časť práce

B OBSAH PRILOŽENÉHO CD

Na priloženom CD nosiči je možné nájsť dve zložky. Jednu s názvom **clinical-questionnaire-server** na ktorej sú zdrojové kódy serverovej a webovej aplikácie. V serverovej aplikácii je dôležitý súbor *application.properties*, kde je potrebné nastaviť hodnotu **spring.security.user.password**, ktorá definuje heslo ktorým sa je možné prihlásiť do webovej a mobilnej aplikácie. Aby mohla byť serverová aplikácia spustená, je potrebné si nainštalovať aj na systém, databázovú aplikáciu postgreSQL, môže byť najnovšia verzia 11. Kde je potrebné vytvoriť užívateľa s názvom **postgres** a heslom **postgres** a taktiež je potrebné vytvoriť aj databázu s názvom **clinic**. S týmito údajmi sa bude snažiť pripojiť serverová aplikácia na databázu. Tieto údaje môžu byť zmenené v application.properties súbore u prvých troch hodnotách. Na vytvorenie týchto nastavení v databáze je odporúčané použiť program PgAdmin, cez ktorý je to jednoduchšie. Aby mohla byť spustená samotná serverová aplikácia je potrebné mať na systéme nainštalovanú javu verzie 8. Samotná aplikácia vie byť potom spustená príkazom **java -jar /clinical-questionnaire-server/clinical-questionnaire-server/backend/target/backend-0.0.1-SNAPSHOT.jar**

Druhá zložka má názov **clinical-questionnaire-mobile**, kde sú uložené zdrojové kódy android aplikácie. Táto aplikácia nie je bohužiaľ otestovaná na reálnom zariadení a preto ju treba spúšťať cez Android studio v emulátore. Aplikácia je písaná pre minimálnu verziu Android SDK 22.

Čo sa týka ukážky nasadenia ostrej verzie, tam by bolo dobré dohodnúť sa s tvorcom systému, aby spustil aplikáciu na riadnom serveri, lebo ináč systém nie je v stálej prevádzke.