TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Programování webového rozhraní pro ovládání přístroje Myotonometru

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Jakub Hadač**
Vedoucí práce: Ing. Martin Kysela

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Myotonometer Web Control Interface Programming

Master thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Jakub Hadač**
Supervisor: Ing. Martin Kysela

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jakub Hadač**
Osobní číslo:	**M15000164**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Programování webového rozhraní pro ovládání přístroje Myotonometru**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Nastudujte problematiku měření svalového napětí měkkých tkání a principu měření přístrojem Myotonometr.
2. Navrhněte uživatelské webové rozhraní k přístroji Myotonometr s využitím HTML, CSS, JavaScriptu a SSL zabezpečením datových zpráv.
3. Naprogramujte uživatelské webové rozhraní k přístroji Myotonometr dle zadání.
4. Ověřte systém z hlediska bezpečnosti uživatelských dat, stability a dostupnosti.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **40–50 stran**

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

Seznam odborné literatury:

[1] **Kadlec V. Ucíme se programovat v jazyce C. Computer Press, 2002. ISBN: 80- 7226-715-9.**

[2] **Šifta, P., Otáhal, S., Süssová, J.: MEASUREMENT OF VISCOELASTIC PROPERTIES OF SOFT TISSUE IN SPASTIC SYNDROME. Neurorehabilitation and Neural Repair, Vol. 20, n. 1, March 2006, 4th Congress for NeuroR**

[3] **Castro E., Hyslop B. HTML5 a CSS3 Názorný průvodce tvorbou WWW stránek. Computer Press, 2012.**

Vedoucí diplomové práce: **Ing. Martin Kysela**
Ústav mechatroniky a technické informatiky

Konzultant diplomové práce: **Ing. Matěj Kolář**
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **10. října 2016**

Termín odevzdání diplomové práce: **15. května 2017**

Technická univerzita v Liberci - 14

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2016

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

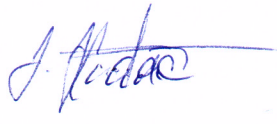
Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15. 5. 2017

Podpis: 

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé diplomové práce, Ing. Martinu Kyselovi za veškeré rady, bohaté zkušenosti a poznatky, které mi pomohly k úspěšné realizaci této práce. Dále bych chtěl poděkovat své rodině za jejich nekončící podporu a možnost studovat na vysoké škole. Nakonec i všem svým blízkým přátelům, kteří mně při studiu velice pomohli a motivovali.

Abstrakt

Cílem této diplomové práce je vytvořit webové uživatelské rozhraní pro ovládání měřicího přístroje Myotonometr s využitím moderních technologií pro vývoj webových aplikací. Na základě nastavených podmínek a definovaných funkcionalit bylo rozhraní rozděleno do samostatných komponent. Pomocí těchto komponent jsou realizovány samostatné části (stránky) rozhraní poskytující definované funkce nebo funkční mechanismy. Výsledné webové uživatelské rozhraní pro ovládání měřicího přístroje a práci se získanými a uloženými daty je vytvořeno propojením všech samostatných částí. Podařilo se vytvořit součást unikátního systému, který umožňuje v rámci webového prohlížeče ovládat měřicí přístroj, nastavit parametry a spouštět měření na měřicím přístroji a také poskytuje mnohé možnosti pro práci s daty, které jsou uloženy v databázi přístroje. Usnadňuje a urychluje práci obsluze přístroje spojenou s prováděním měření a zhotovováním výsledného záznamu o měření.

Klíčová slova

Měření; webová aplikace, jednostránková aplikace; uživatelské rozhraní; ReactJs; Myotonometr

Abstract

This diploma thesis aims at and this document deals with development of web user interface for controlling measuring device Myotonometer with the use of modern technologies for web application development. Based on the set conditions and defined functionalities, the user interface was divided into separate components. These components create separate parts (pages) of the user interface that provide defined functions and functional mechanisms. The resulting web user interface for controlling the meter and working with the acquired or the saved data from the measuring device is realize out by linking all the separate parts (pages). All the effort clearly led to new part of the unique system, allowing creation of measurements at measuring device Myotonometer and work with acquired or saved data every the patient, running easily in a web browser and speeding up the activities and tasks connected to the creation of all the measurements documents or protocols.

Keywords

Measurement; web application; single-page application; user interface; ReactJs; Myotonometer

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam zkratek

- AJAX (Asynchronous JavaScript and XML) – obecné označení pro asynchronní technologie, které mění obsah webových stránek bez nutnosti kompletního znovu načítání
- API (Application Programming Interface) – rozhraní pro programování aplikací
- CSS (Cascading Style Sheet) – jazyk pro popis grafického zobrazení prvků na webové stránce
- DOM (Document Object Model) – objektově orientovaná reprezentace HTML dokumentu
- HTML (HyperText Markup Language) – značkovací jazyk pro tvorbu webových stránek
- HTTP (HyperText Transfer Protocol) – je internetový protokol k výměně hypertextových dokumentů
- JSON (JavaScript Object Notation) – formát dat určený pro přenos dat
- PDF (Portable Document Format) – přenosný formát dokumentů od firmy Adobe
- SPA (Single-Page Application) – jednostránkový model webové aplikace
- XML (Extensible Markup Language) – standardní formát pro výměnu dat

Úvod

Tuto diplomovou práci jsem si zvolil, protože během mého studia na vysoké škole mě zaujaly některé technologie pro vývoj webových aplikací. Tato práce spojovala nejen využití nových technologií pro vývoj webových aplikací, ale i jejich propojení s nově vyvíjeným měřicím přístrojem Myotonometr.

Dříve (někdy stále i dnes) pro měření svalového napětí lékaři používali tak zvanou palpační metodu, kdy pomocí prstů poklepávali na pokožku nad daným svalem a tak jej testovali. Bohužel se tato metoda ukázala jako velice subjektivní a začaly vznikat první přístroje označované jako Myotonometr. Dnešní varianty přístroje jsou koncipovány spíše jako přenositelná zařízení umožňující rychle přeměřit svalové napětí měkkých tkání u daného pacienta a na základě získané hodnoty vyhodnotit stav pacienta. Bohužel tyto přístroje většinou neposkytují možnost nastavovat parametry prováděného měření a jejich výsledkem je jedna hodnota získaná buď jako průměr z naměřených hodnot nebo jako nejčastější hodnota. Jelikož tyto přístroje neumožňují nastavovat parametry měření a poskytují pouze jednu výslednou hodnotu, nejsou vhodné pro měření v laboratořích. Z těchto důvodů je na Technické univerzitě v Liberci vyvíjen přístroj, který má sloužit k realizaci profesionálních měření. Tato varianta přístroje umožňuje provádět mnohem komplexnější měření a tak měřit nejen svalové napětí ale rozeznat i o jaký typ měkké tkáně se jedná. Bohužel není vybavena displejem pro zobrazení naměřených dat jako jiné varianty, proto byl přístroj rozšířen o server a wi-fi modul, pomocí něhož se k němu může uživatel připojit.

Cílem této práce je vytvořit webové uživatelské rozhraní, které umožní obsluze nově vyvíjeného přístroje ho efektivně a jednoduše ovládat a také obsluze poskytne funkcionality pro správu již naměřených dat.

Cíle a omezení práce

Cílem této diplomové práce je navrhnout a realizovat webové rozhraní pro ovládání měřicího přístroje Myotonometru, které umožní uživateli vytvořit, nastavit a spustit měření na přístroji. Dále pak poskytne uživateli funkcionality pro manipulaci s naměřenými daty nebo pacienty, či již uloženými měřeními.

Dalším cílem práce je navržení a vytvoření vhodných podpůrných funkcí, které by uživateli umožnily pracovat s přístrojem rychleji nebo vhodně manipulovat s naměřenými daty. Dále s tímto cílem souvisí i jednoduché a přehledné ovládání rozhraní, které by nemělo uživatele zdržovat dlouhým vyhledáváním v aplikaci.

Třetím a posledním cílem této práce je otestování a ošetření výsledné aplikace proti všem nalezeným chybám nebo možným útokům na aplikaci. Tak aby bylo docíleno co nejstabilnějšího a nejbezpečnějšího chodu výsledné aplikace.

Protože cílem této práce není realizace rozsáhlého komplexního problému v rámci celého systému, ale pouze realizace dílčí části související s vývojem měřicího přístroje Mytonometer, jsou zde určitá omezení.

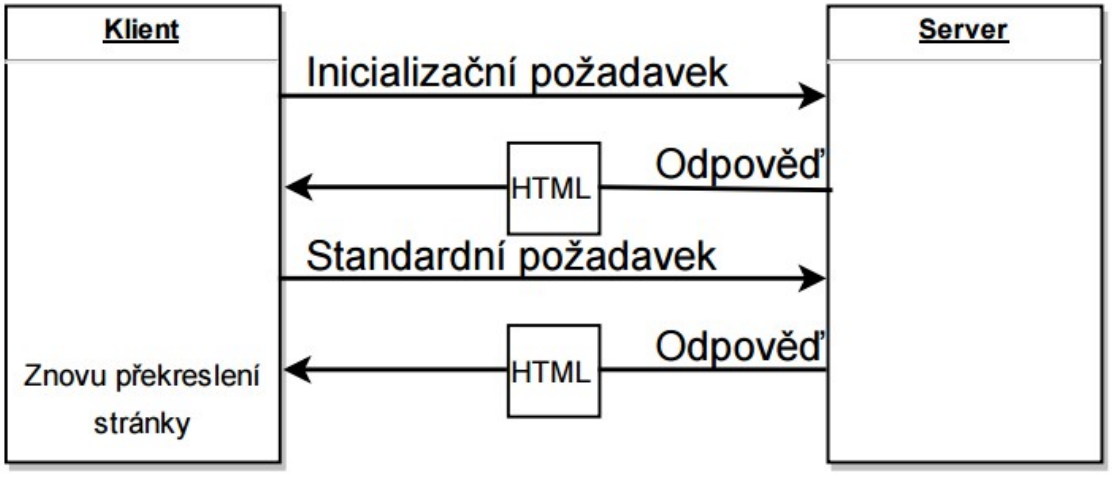
Prvním omezením je samostatný měřicí přístroj, který byl vyvíjen a zdokonalován souběžně s realizací této práce. Protože přístroj nebyl v zadání této diplomové práce zcela hotový, bylo umožněno jeho vývoj ovlivňovat z přidávaných návrhů na implementace funkcionalit přímo do systému, které umožní realizovat výsledné uživatelské ovládací rozhraní. Druhým omezením práce je serverová část přístroje a databáze, které byly přidány pro rozšíření systému a nebylo možné tak využít jiných voleb při návrhu výsledného rozhraní.

1 Výběr použitých technologií

V této části diplomové práce budou rozebrány a zdůvodněny technologie a techniky použité při návrhu a realizaci ovládacího uživatelského rozhraní pro přístroj Myotonometr, například asynchronní komunikace mezi klientem a serverem nebo jednostránkový typ webových aplikací. Dále také samotný měřicí přístroj Myotonometr, pro který bylo uživatelské rozhraní vytvářeno.

1.1 Jednostránková webová aplikace

Tradiční webovou aplikací rozumíme aplikaci, která rozmisťuje funkcionality aplikace do více stránek, tak aby každá stránka vždy odpovídala nějakému určitému stavu. Problémem takovéto aplikace je nutnost zatěžování serveru vytvářením výsledné HTML stránky. Typickým zástupcem vícestránkové webové aplikace jsou statické webové stránky nebo dynamické webové aplikace napsané například ve skriptovacím jazyku PHP nebo Python.

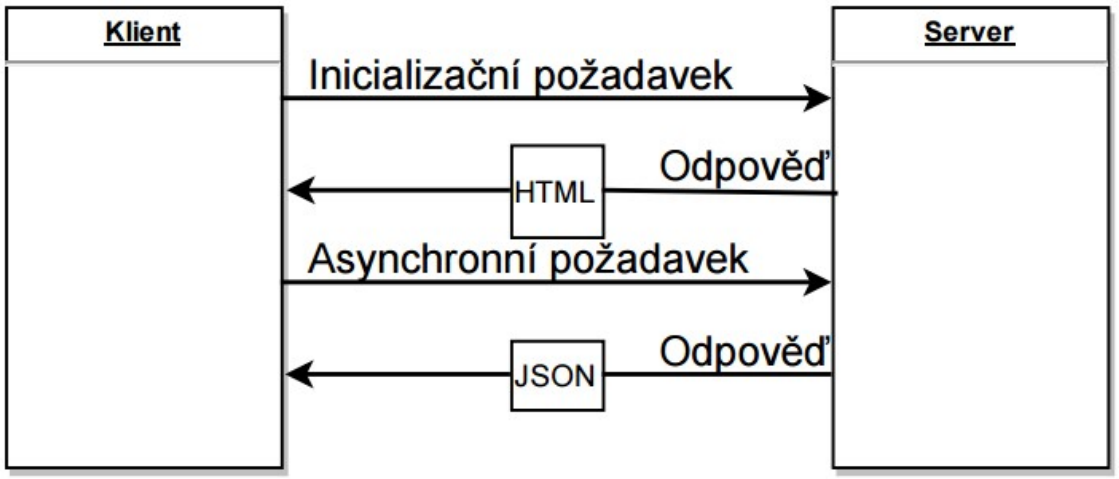


Obrázek 1: Životní cyklus vícestránkové aplikace

Princip, na kterém tento typ webových aplikací funguje je opakované zasílání dotazu na stranu serveru za účelem získání nové HTML stránky. Uživatel si na základě HTTP dotazu s metodou GET vyžádá HTML stránku

s určitým statickým obsahem, daty dle svého výběru, nebo kombinací obojího. Pokud je to nutné, server získá data ze zdrojových souborů nebo použije vhodný databázový dotaz, aby data získal z databáze. Dále pak na základě předem nadefinované logiky sestaví kompletní HTML stránku, kterou odešle zpět klientovi. Tento postup se opakuje stále dokola (Shimanovsky, 2015).

Jednostránková aplikace (z angl. Single-Page Application - SPA), která je ve své definici označována za aplikace s podobnými vlastnostmi a chováním jako aplikace určená pro osobní počítač nebo mobilní zařízení. Z této definice tedy vyplývá, že při prvním načtení webové aplikace dojde nejen k načtení statické HTML stránky, ale také se načtou nezbytné JavaScriptové soubory obsahující veškerou funkcionalitu a logiku obsaženou v aplikaci, dále také soubory kaskádových stylů CSS a další potřebné zdrojové soubory.



Obrázek 2: Životní cyklus jednostránkové aplikace

Veškeré podoby výsledných stránek jsou již přeneseny na straně klienta a ten svými akcemi zvolí, která ze stránek bude aktuálně zobrazena. Je tedy komunikace se serverem omezena pouze na správu dat skrze API umožňující manipulaci s daty. Například jde o přidání, odstranění nebo modifikace dat na straně serveru nebo získání dat z databáze, která jsou následně vložena

do uživatelem vybrané HTML stránky bez nutnosti přehrání webové stránky (Takada, 2012).

1.1.1 Výhody

Mezi výhody tohoto řešení patří rychlost, kterou aplikace překresluje výsledné stránky. Ačkoliv první vykreslení (nebo načtení) stránky může být pomalejší než u klasických vícestránkových webů, protože musí dojít ke stáhnutí a zpracování všech potřebných zdrojových souborů, následná navigace skrz web je již rychlejší.

Další výhodou, která přímo souvisí s předchozí výhodou, je snížení výkonnostních a paměťových nároků na server. Server nyní slouží pouze jako zdroj dat, která uživatel v aplikaci potřebuje zobrazit, a nemusí svůj výpočetní výkon využívat na vygenerování výsledné stránky. Není tedy nutné na stranu klienta posílat zbytečná a duplicitní data jako je hlavička nebo patička stránky, které se většinou v rámci dané aplikace nemění (Ingram-Westover, 2015).

Další z mnoha výhod je lepší práce s cachováním webu. Pokud je potřeba provést výkonovou nebo časovou optimalizaci kódu na straně serveru, je tak nutno se zaměřit pouze na API pro získávání dat z databáze, se kterou SPA komunikuje. Styly, skripty, šablony webu a další statické soubory se mohou cachovat v prohlížeči, spojovat, komprimovat, atp. U klasických vícestránkových aplikací je serverová optimalizace mnohem složitější. Například šablony webu nelze ukládat do cache prohlížeče a při každém požadavku se zpracují na straně serveru a jsou znova zaslány (Štrauch, 2012).

1.1.2 Nevýhody

SPA mají také několik nevýhod, které především vyplývají z toho, že při vývoji webových technologií, prohlížečů a nástrojů se nepočítalo s aplikacemi

tohoto typu. U některých typů webových aplikací tyto nevýhody ale nemusí vadit, případně je převýší výhody.

Asi nejvíce postřehnutelnou nevýhodou je pomalé počáteční načtení, protože při prvním načtení SPA se před samotným vykreslením HTML stránky musí načíst veškeré soubory obsahující funkcionality aplikace, případně další podpůrné knihovny a zdrojové soubory. Tyto soubory (mnohdy sloučené pouze do jednoho JavaScriptového souboru) bývají rozsáhlé jak délkou, tak i velikostí. Z tohoto důvodu může proto první načtení SPA trvat i o několik sekund déle než u podobné více-stránkové aplikace (Wasson, 2013).

Další nevýhodou je historie webového prohlížeče, protože SPA je definována jako jednostránková, není možné tedy využívat funkcí tlačítek „Vpřed a Zpět" sloužící k pohybu v historii prohlížeče. Tento problém může být vyřešen např. použitím HTML5 standartu a to přesněji pomocí funkcí *pushState()* a *replaceState()* vycházející z History API, které umožňují skrze objekt *window.history* uložený v DOM měnit historii prohlížeče. Další možností jak vyřešit tento problém je využití některého z JavaScriptových Frameworků (samotného nebo spolu s některou rozšiřující knihovnou), který zpřístupní svoje funkce nebo objekty pro manipulaci s historií webového prohlížeče (Oshiro, 2014).

1.2 AJAX

Nejrozšířenější technologie pro asynchronní odesílání a přijímání dat na pozadí webové aplikace mezi stranou klienta a serveru bez nutnosti znovu načítání celé HTML stránky se nazývá Asynchronní JavaScript a XML (z angl. Asynchronous JavaScript and XML - AJAX). Jako praktičtější se ukázalo nahradit datový formát XML formátem JSON, který je nativně umístěn v JavaScriptu (Garrett, 2005).

AJAX je kombinací několika technologií, které dohromady vytvářejí základ pro webové aplikace typu SPA. JavaScript a objekt XMLHttpRequest (Aubourg, a další, 2016) poskytují metody pro asynchronní výměnu dat mezi prohlížečem klienta a serverem, aby se zabránilo úplnému načtení stránek.

1.2.1 Websocket

WebSocket je technologie, která vytváří plně-obousměrné spojení mezi klientem a serverem, kteří si potom mohou vyměňovat data v reálném čase. Na základě protokolu HTTP nebo HTTPS (Lampa, 2002) pomocí příkazu CONNECT se otevře TCP/IP spojení s určeným serverem na určeném portu (například https://*www.exaple.com:8080*). Jakmile je takto vybudován tunel, mohou být zprávy volně přenášeny z jedné strany na druhou. Technologie WebSocket vychází částečně z technologie AJAX a Comet (nestandardizované řešení na principu dlouhodobě vyhrazeném HTTP spojení). Jedná se o řešení pro vytváření real-timových aplikací jako např. chatů, aplikací zobrazujících ceny komodit obchodovaných na burze, nebo právě pro některé SPA.

Výhoda WebSocketů je v tom, že na straně klienta je tato technologie implementována přímo ve webovém prohlížeči a přístupná pomocí jazyka Javascript. Na straně serveru je to už komplikovanější, protože nestačí pouze jednoduchý HTTP server, ale specializovaný, který podporuje tuto technologii, nicméně dnes existují knihovny téměř pro každý vývojový jazyk, ve kterém je možné napsat webový server (Hickson, 2012).

1.3 Framework ReactJs

ReactJs je JavaScriptový otevřený framework od firmy Facebook pro vytváření jednostránkových aplikací, který přinesl mnoho novinek a inovací, které mohou za změnu způsobu, jakým dnes vývojáři píší svoje webové aplikace. ReactJs odbočuje od využívání návrhového vzoru MVC (Model-View-Controller), který je používán při vytváření webových aplikací

a je implementován u většiny ostatních frameworků jako například v AngularJS od firmy Google, nebo v EmeberJS. React představuje pouze prezentační vrstvu View, ale takto přináší změnu ve psaní kódu, kterou se odlišuje od ostatních frameworků, protože vývojář už pouze kódem popisuje, jak má výsledek vypadat (Facebook, 2016).

Framework využívá nástroje, které umožňují zvýšit rychlost vykreslování i u složitějších aplikací. Modifikace DOMu ručně nebo automaticky po každé změně může být velmi pomalá i přesto, že JavaScript je velice rychlý, a proto ReactJs přišel se svojí vlastní zjednodušenou verzi DOMu nazývanou virtuální DOM (z angl. VirtualDOM).

Velkou výhodou frameworku ReactJs je samotná komunita vývojářů, která vyvíjí a zpřístupňuje mnoho užitečných knihoven. Díky tomuto systému balíčků (modulů či knihoven) je možné vyvíjet výslednou aplikaci mnohem rychleji a efektivněji.

1.3.1 Komponenta

Základním stavebním kamenem frameworku ReactJs je komponenta, která reprezentuje určitou část výsledné aplikace. V komponentě dochází k nadefinování struktury, tedy k popisu výsledného vzhledu komponenty a její aplikační logiky. V rámci definice výsledného vzhledu jedné komponenty je možné vkládat (využívat) i jiné komponenty, což umožňuje efektivně navrhovat výsledný vzhled stránky a také redukovat množství přenášeného kódu na stranu klienta ve výsledném JavaScriptovém souboru.

Proto vzniká nové označení některých komponent a to výrazem kontejner. Kontejner označuje takovou komponentu, která vybírá data ze svých sub-komponent nebo z různých zdrojů, jako například z komunikace pomocí WebSocketů. Nebo se na základě vlastních dat a nastavené logiky rozhoduje, které komponenty vykreslí a jestli do těchto komponent má předat nějaká data

nebo nastavující parametry. Označení komponenta pak zůstává pro takové komponenty, které nezobrazují žádné jiné komponenty a řeší (nebo zobrazují) tak nedělitelný systém.

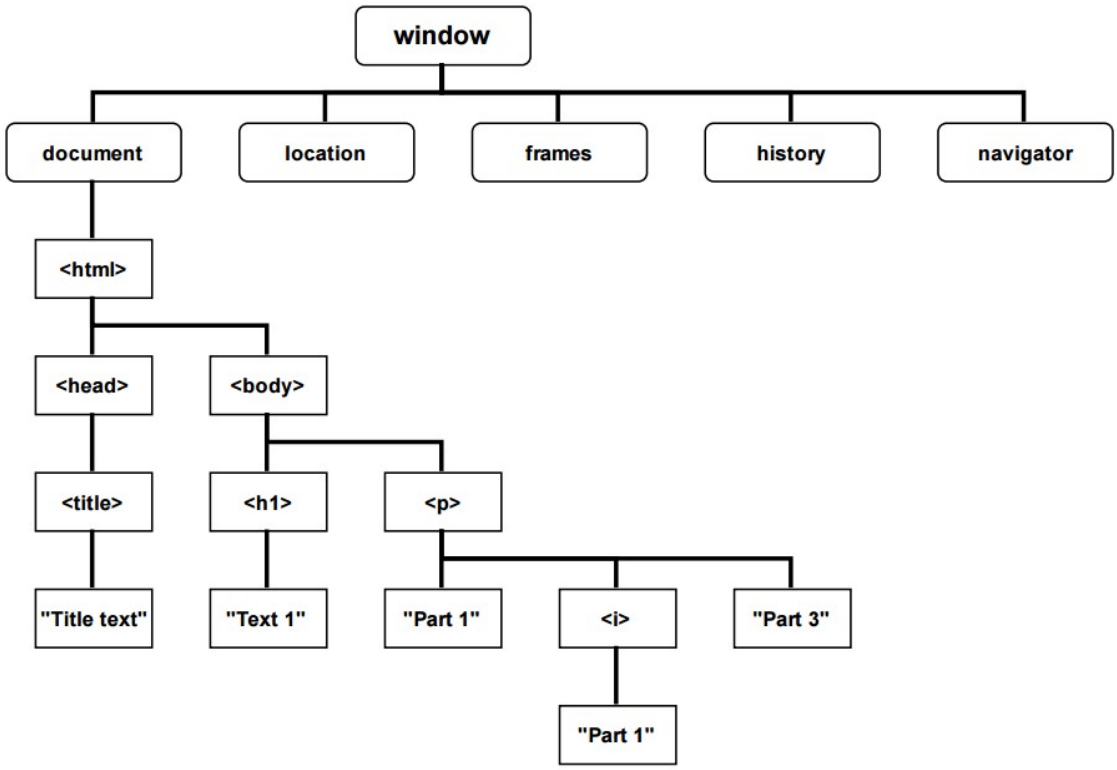
Framework ReactJs vyžaduje, aby komponenta měla vždy implementovanou funkci *render*, protože bez této funkce by komponenta postrádala smysl. Neboť by komponenta nezobrazovala žádnou část výsledné stránky, ale reprezentovala by pouze jednotlivé funkce, které pak pro svoji činnost nepotřebují komponentu. ReactJs v rámci komponenty a jejího tzv. životního cyklu (fáze ve kterých se komponenta nachází) vybudoval hned několik funkcí, které se volají automaticky. Například při inicializaci komponenty se zavolá funkce *render* a po jejím dokončení je zavolána funkce *componentDidMount*, takže došlo k vykreslení komponenty a bezprostředně na to se komponenta může například dotazovat serveru na data nebo může uživatele informovat, že je komponenta připravena k použití.

1.4 DOM

Objektový model dokumentu (z angl. Document Object Model - DOM) je objektově orientovaná reprezentace XML nebo HTML dokumentu ve formě stromu. Dále reprezentuje také vlastnosti samotného prohlížeče nebo používaného zařízení, například objekt **navigator**, který obsahuje informace o pozici, typu připojení a stavu baterie zařízení. DOM je také API umožňující přístup nebo modifikaci obsahu v něm uloženého. Je možné měnit vlastnosti samotných atributů HTML elementů, obsah uložený mezi nimi, nebo vlastnosti kaskádových stylů a atd (Koch, 2001).

Původní reprezentace DOM v jednotlivých prohlížečích záležela pouze na tvůrcích daného prohlížeče, ale tuto nejednotnost vyřešila standardizace od W3C (World Wide Web Consorcium), která je platformě i jazykově nezávislá. Tento krok měl za následek zjednodušení vývoje webových aplikací

využívajících JavaScript, protože nebylo nutné psát tu samou aplikaci pro několik různých reprezentací DOMu.



Obrázek 3: Ukázka reprezentace struktury DOM

Vlivem vývoje webových technologií a nástrojů došlo k tomu, že DOM obsahuje data (informace) nejen o reprezentaci výsledného HTML dokumentu ale i nástroje k jejich manipulaci, které umožňují efektivně vyvíjet webové aplikace typu SPA. (Le Hégaret, a další, 2002)

1.4.1 Virtualní DOM

Virtuální DOM funguje identicky jako standardizovaný DOM obsažený ve webovém prohlížeči, ale jeho smyslem a cílem je zefektivnit vykreslování výsledné HTML stránky při jakékoliv informaci uložené v DOM. Proto není nutné, aby virtuální DOM byl zcela identickou kopií standardního DOMu, ale stačí, aby se jednalo o abstraktní verzi, přesněji zjednodušenou kopií (Freed, 2016).

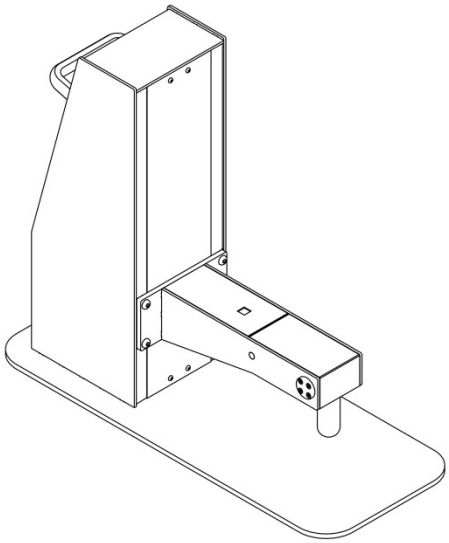
Standardně je při jakékoliv změně vykreslována celá stránka tak, že stávající DOM je nahrazen novou strukturou obsahující provedenou změnu. Při využití virtuálního DOMu je kopie stávající struktury uložena právě do něj a dojde-li ke změně některého parametru, hodnoty nebo části struktury, je automaticky vyvolané porovnání obou struktur. Výsledkem jsou rozdíly od původního DOMu včetně jejich umístění, které jsou pak do něj zaneseny. Prohlížeč tak překreslí stránku rychleji a efektivněji (Krajka, 2015).

1.5 Měřicí přístroj Myotonometr

Měřicí přístroj Myotonometr je neinvazivní měřicí přístroj pro měření svalového napětí měkkých tkání, který vykonává měření pomocí řízeného měřícího ramene s vhodně zvolenou měřící hlavou (indentorem). Řídicí systém přístroje je rozdělen na dva HW moduly.

První modul s čipem AT91sam9g25 od firmy Atmel s jádrem ARM9 o frekvenci 400MHz a 128 MB DDR2 RAM pamětí, který zajišťuje podporu pro rozhraní Ethernet 100 Mbps, sběrnice USB, UART, SPI a I2C, desetibitový AD převodník a SD kartu. Dále je zaváděn operační systém Unix, který zjednodušuje vývoj přístroje o možnost využívat stávající již ověřené SW balíčky, jako je HTTPS server, DNS a DHCP server, SW emulovaný WiFi AP, správa souborového systému anebo databáze fungující na principu adresářové struktury.

Druhý modul zajišťuje ovládání všech periferií, které nemohou být řízené pomocí prvního modulu kvůli nepřímému přístupu do periferií. Proto je nutné použít externí mikroprocesor (STM32F301), který komunikuje s prvním modulem po sběrnici typu SPI pomocí vyrovnávací paměti. Mikroprocesor pak zajišťuje řízení motorů měřicího přístroje, přesné a synchronizované čtení dat z AD převodníku (Kysela, a další, 2016).



Obrázek 4: Drátový model přístroje Myotonometr

2 Návrh

V této části diplomové práce budou popsány specifikace požadavků na funkcionalitu výsledného uživatelského rozhraní. Dále zde bude popsána struktura aplikace a adresářová struktura souborů, ze kterých je aplikace vytvořena. Další část se zabývá navrženým formátem komunikace mezi klientem a serverem a rozvržením databáze pro ukládání dat. Nakonec je v této části práce popsán datový tok v aplikaci s využitím rozšiřující knihovny Redux.

2.1 Specifikace požadavků

Protože aplikace bude sloužit jako uživatelské rozhraní k ovládání měřicího přístroje Myotonometr, zobrazení naměřených dat nebo manipulaci s daty uloženými v databázi, bylo nutné zformulovat a definovat některé požadavky na danou aplikaci.

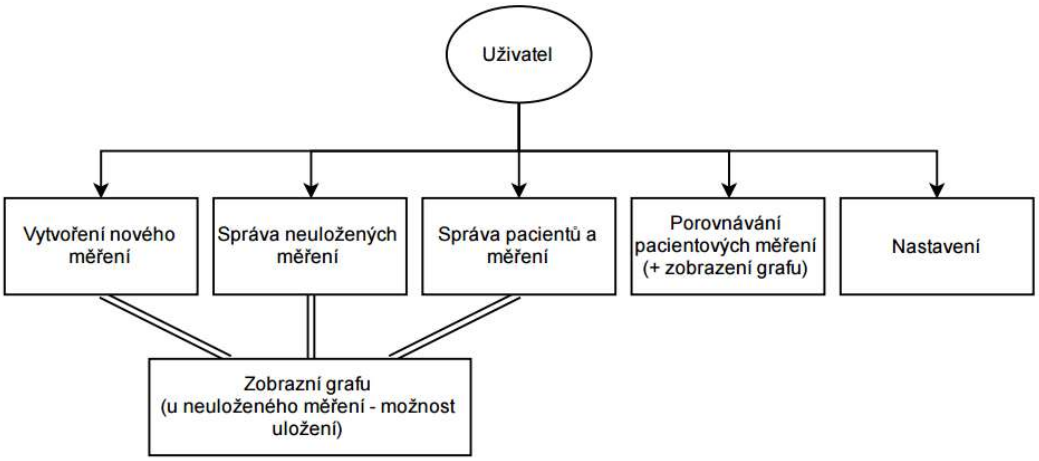
Aplikace umožní uživateli (lékaři nebo pověřenému pracovníku) přihlásit se ke svému profilu a tak zpřístupnit nástroje pro nastavení a vytvoření měření na přístroji Myotonometr. Dalšími zpřístupněnými nástroji jsou nástroje pro správu databáze pacientů a k nim náležejících provedených měření. Uživatel má možnost nastavit si vlastnosti webového rozhraní jako je jazyková lokalizace nebo tvar a rozměry použité měřící hlavy přístroje.

Uživatelské rozhraní musí být přehledné a jednoduché, tak aby uživateli umožnilo velmi rychle nalézt a vybrat si zvolenou funkcionalitu, kterou chce momentálně využít. Není nutné vytvářet složitá a hodně členěná menu, která by uživatele omezovala v práci se samotným přístrojem nebo uživatelským rozhraním.

Dalším požadavkem na uživatelské rozhraní je intuitivní ovládání. Přesněji, aby ovládání jednotlivých funkcionalit nebylo složité na nastavení nebo nezdržovalo uživatele od jeho skutečné práce s přístrojem.

2.2 Diagram případů využití aplikace

Tento typ diagramu zobrazuje chování systému z pohledu uživatele, tedy popisuje strukturu funkcionalit, ke kterým má uživatel přístup. Zároveň zobrazuje výslednou dekompozici samotné aplikace do jednotlivých stránek, které umožní uživateli pracovat s danou funkcionalitou.



Obrázek 5: Diagram užití pro přihlášeného uživatele

Jednou z těchto výsledných stránek je stránka Měření, která umožní uživateli nastavit parametry a následně spustit měření na přístroji. V případě, že měření proběhne v pořádku, automaticky dojde k zobrazení výsledného protokolu obsahujícího informace o pacientovi, parametrech měření a graf z naměřených hodnot.

2.3 Struktura projektu

Při vytváření webové aplikace se programátor (nebo kodér) nespoléhá pouze na jednu technologii a vlastní úsilí, ale většinou jich využívá hned několik dohromady. Tento přístup mu ulehčuje vývoj výsledné aplikace nejen po časové stránce ale i možností využít znovupoužitelného a otestovaného kódu. Podobné řešení je použito i u této práce.

Výsledný systém můžeme rozdělit na část klientskou, která realizuje grafické uživatelské rozhraní (prezentační vrstvu) pro ovládání měřicího přístroje a všech funkcionalit (aplikační vrstvu) nabízených rozhraním, běžící v uživatelově webovém prohlížeči. A dále na část serverovou, která je realizovaná samotným přístrojem a umožňuje skrze komunikační rozhraní nastavit a spustit měření, nebo manipulovat s daty uloženými v databázi přístroje.

Obě části systému si mezi sebou vyměňují data skrze bezdrátovou síť vytvářenou WiFi modulem, který je součástí přístroje, pomocí asynchronní komunikační technologie WebSocket, která umožňuje odesílat data ze serveru na klienta i bez nutnosti aby klient o data musel zažádat. Tento způsob komunikace nezatěžuje server (měřicí přístroj) neustálým dotazováním ohledně dat a tak server může svůj výpočetní výkon využívat pro realizaci požadavků ostatních uživatelů, nebo řízení samotného měření.

2.3.1 Adresářová struktura projektu

Adresářová struktura projektu je závislá pouze na autorovi dané aplikace, ale pokud se jedná o webovou aplikaci typu SPA využívající JavaScriptového frameworku ReactJs jsou dány určité části, které struktura musí obsahovat a další uspořádání je na autorovi (autorech) práce. Jednou z možností je danou strukturu vytvářet ručně nebo lze využít instalační projekt s názvem create-react-app. Tento projekt umožní přes NPM (Node Package Manager je balíčkovací systém pro JavaScript) nainstalovat nutnou strukturu a veškeré potřebné balíčky (moduly nebo knihovny) pro správné fungování SPA postavené na frameworku ReactJS.

V kořenovém adresáři projektu (viz Příloha A) se nachází soubor **package.json**, který musí obsahovat každý balíček nebo projekt. Obsahuje totiž základní informace o projektu jako např. jméno autora, verze nebo popis

aplikace. Dále obsahuje velmi důležitou informaci, kterou je seznam závislostí, tj. seznam balíčků (modulů nebo knihoven), které daný projekt potřebuje pro své správné fungování, které jsou stáhnuty a nainstalovány automaticky pomocí NPM.

Povinou součástí struktury pro projekt je adresář *node_modules*, ve kterém jsou instalované balíčky včetně jejich závislostí pomocí NPM. V tomto adresáři je nainstalovaný samotný framework ReacJS doplnění o další knihovny potřebné k realizaci výsledné SPA. Dalším balíčkem instalovaným v adresáři je Babel, který provádí překlad z tzv. next-generation JavaScript označovaného taky jako ECMAScript 6 (rozšíření jazyka JavaScript o chybějící syntaxi jako např. *class* nebo *import*, kterou je nutné překládat) z roku 2015 do JavaScriptu kompatibilního s prohlížeči (jedná se o funkce, které jsou podporovány všemi prohlížeči). Posledním důležitým balíčkem je modul Webpack, který slouží k sestavení a přípravě k distribuci výsledné aplikace v podobě jednoho JavaScriptového souboru. Tento výsledný soubor je umístěn v adresáři *build* společně se statickým HTML a CSS souborem.

V adresáři *src* je umístěný kompletně celý projekt, který je rozdělený do jednotlivých souborů kvůli jednodušší orientaci během vývoje, než v případě jediného souboru obsahujícího veškerý kód. Projekt je rozdělen do jednotlivých skupin obsahujících vždy stejné prvky, jako například skupina komponent obsahující zdrojové soubory všech komponent, dále například skupina reduktorů obsahující zdrojové kódy a logiku všech použitých reduktorů.

2.4 Komunikační API

Komunikace mezi klientem a serverem je řešena pomocí zabezpečené technologie WebSocket přes SSL/TLS, která umožňuje obousměrnou komunikaci. Tedy není potřeba, aby se klient opakovaně dotazoval, zda

má server pro něj připravena nová data, ale server data automaticky odešle klientovi, když jsou data pro klienta připravena. Jedním z hlavních důvodů komunikace mezi klientem a serverem je umožnit přístup k datům uloženým v databázi nebo umožnit se klientovi přihlásit do systému. Proto je nutné definovat formát zpráv pro volbu události, která se na straně serveru má vykonat.

Klient odesílá na stranu serveru zprávy s pevně daným formátem, který obsahuje čtyři důležité informace. První informací je akce, kterou chce klient na straně serveru provést uložená v proměnné **action**. Tato hodnota nabývá názvů funkcí používaných v databázi jako je funkce **get**, nebo volby pro přihlášení jako je **login**. Další informací je cesta, definující v jaké části databáze má být provedena zvolená akce, která je uložena v proměnné **location**. Třetí informací jsou samotná data uložená v proměnné **content**, se kterými se zvolená akce na určeném místě má provést. Poslední informací je upřesňující volba uložená v proměnné **pattern**, která definuje, jaké části z výsledku akce se mají vrátit zpět na stranu klienta. Výsledný řetězec dat ve formátu JSON tvořící požadavek vypadají takto: {"action":"get", "location":"/data/users/1", "content":{}, "pattern":"username"}.

Server odesílá na stranu klienta dva různé formáty zprávy. První formát slouží jako odpověď na klientem vyžádanou akci, která ho informuje o výsledku provedené akce. Tento výsledek je uložen jako číselná hodnota v proměnné **status**, pokud je hodnota rovna nule akce proběhla v pořádku, pokud je hodnota proměnné **status** větší než nula znamená to, že při provádění dané akce došlo k chybě a potom odpověď obsahuje proměnnou **error_message**, ve které je uložena správa pro klienta o dané chybě. Poslední proměnnou obsaženou v odpovědi je **content**, který obsahuje získaná data na straně serveru. Výsledný získaný řetězec na předchozí dotaz vypadá takto: {"status":0, "content":{"username":"username1"}}.

Druhý formát, který server odesílá na stranu klienta je zpráva, obsahující informace o změně obsahu dat v části databáze, na které klient naslouchá. Tento formát je složen z proměnných **action** a **location**, které byly předány jako součást dotazu klienta. Dále obsahuje proměnnou **content**, ve které se nachází celá oblast databáze, na které uživatel naslouchá s nově modifikovanými daty. A jako poslední je proměnná **status**, předávající informace o výsledku zvolené akce.

2.5 Databáze

Součástí výsledného systému je i databáze, která uživateli slouží jako úložný prostor pro jeho data. Uživatel využívá data uložená ve svém webovém prohlížeči, ale v případě, že chce uložit nového pacienta nebo získané měření k danému pacientovi, je nutné tuto změnu uložit i do příslušné části v databázi.

Databáze na straně serveru je tvořena jednoduchým úložným systémem ve formě adresářové struktury. Tento jednoduchý databázový systém poskytuje funkce *add*, *get*, *modify* a *remove*. Jedná se o základní operace nad trvalým úložištěm označované jako CRUD (Create, Read, Update, Delete), které slouží vytváření datové struktury, vkládání, úpravu nebo mazání dat. Dále databáze poskytuje funkci *bind* pro naslouchání určité části databázové struktury, která vrací změny provedené v této části všem, kteří naslouchají.

Navržený datový model (viz Příloha B), který slouží k ukládání nebo předávání dat je rozložen na čtyři části. V první části jsou uloženi uživatelé a slouží především k ověření uživatele při požadavku na manipulaci s daty nebo při pokusu o přihlášení uživatele do systému. Ve druhé části jsou uložena data o jednotlivých pacientech tak, že každý pacient má pouze jednoho lékaře. Ve třetí části jsou uložena veškerá měření pacientů.

Poslední část slouží jako rozhraní pro předávání zpráv mezi uživatelem a měřicím přístrojem Myotonometr. Skrze tuto část se spouští nové měření

na přístroji tak, že jsou do ní uloženy veškeré nastavující parametry, které si přístroj vyzvedne a provede zadané měření. Po skončení měření zapíše do databáze výsledek, který je automaticky odeslán uživateli, protože uživatel poslouchá změny provedené v této části databáze.

Nevýhodou tohoto návrhu je možný vznik duplicitních dat, který je způsobený vlastností databáze. Přesněji v databázi není možné provádět vyhledávání na základě zvolených podmínek jako například vyhledat veškeré pacienty, u kterých pole lékařů obsahuje identifikátor daného uživatele.

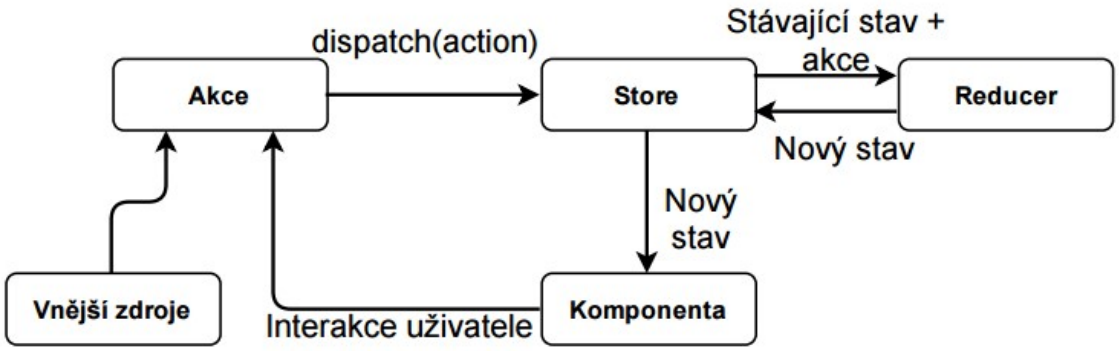
2.6 Datový tok

Přenos dat v aplikacích využívající framework ReactJs je velice jednoduchý, protože potřebná data si mohou předávat komponenty mezi sebou díky proměnným *props*. Využívají k tomu dvě metody. První metodou je shora dolů, kde jedna komponenta přes parametr předává data při volání druhé komponenty např. **[paramName]={data}**. Druhá komponenta má přístup k datům přes proměnnou **this.props.[paramName]**. Druhou metodou je zezdola nahoru, kdy místo dat je předána tzv. funkce zpětného volání (z angl. callback function), kterou druhá komponenta zavolá, když bude předávat data do první komponenty.

Uvedené způsoby předávání dat jsou vhodné jen v malém měřítku, nebo když nejsou využívána stejná data ve více komponentách. Z toho tedy vyplývá, že pokud se tento způsob využívá při přesunu mezi mnoha komponentami, je nutné v každé komponentě nastavovat, jaká data se musí předat a jaká ne. Takovéto řešení je neefektivní, protože při jakékoliv změně potřebných dat je nutné upravit tyto omezení ve všech nadřazených komponentách.

Zmíněný problém lze vyřešit za pomoci knihovny Redux, která funguje jako API pro práci s knihovnou Flux od firmy Facebook (Abramov, a další, 2017). Jedná se o návrhový vzor, který se aplikuje při návrhu jednosměrného

toku dat a vychází se z předpokladu, že úložným místem všech přenášených dat v rámci aplikace bude jeden JavaScriptový objekt nazývaný Store.



Obrázek 6: Datový tok s knihovnou Redux

Store je objekt reprezentující úložiště všech sdílených dat a v celé aplikaci je vždy jeden, to ale neznamená, že je nutné vytvářet jeden obrovský Store, který by uchovával veškerá data včetně logiky pro manipulaci s těmito daty. Je možné ho definovat po určitých menších částech, které pak následně sloučíte do jednoho výsledného úložiště.

Komponenta má možnost si data z tohoto úložiště vyzvednout pomocí funkce *store.getState()*, nebo provedením tzv. propojení si naváže určená data na svoji proměnnou **props**. Takovým řešením se docílí automatického překreslení komponenty při změně dat ve Store jako v případě volání funkcí *this.setState()* nebo *this.forseUpdate()* ve vnitřní logice komponenty.

Díky této vlastnosti je možné předávat daným komponentám pouze ta data, která potřebují pro svoji funkcionalitu. Dále tato vlastnost umožňuje zpřístupnit funkce pro manipulaci s daty ve Store jak samotným komponentám, tak například i komunikačnímu API, určenému pro komunikaci přes technologii WebSocketů.

3 Realizace

Na základě zadání práce byla vytvořena výsledná aplikace realizující ovládací uživatelské rozhraní pro měřicí přístroj. K vytvoření této aplikace byly použity technologie a techniky popsané v kapitole 2, také byly použity veškeré návrhy popsané v předchozí kapitole. Proto v této části diplomové práce bude popsána vlastní realizace některých dílčích částí výsledné aplikac, jako je směrování v rámci aplikace a realizace objektu Store. V poslední části budou popsány jednotlivé výsledné stránky, které aplikace obsahuje.

3.1 Směrování aplikací pomocí URL

Každé přesměrování v aplikaci je podmíněno použitím URL jako zdroje informace. Tedy, vypadá-li URL adresa *https://www.exaple.com,* dojde k přesměrování na domovskou stránku, obsahuje-li navíc cestu, například */settings*, přesměruje se na stránku s nastavením.

Framework ReactJs sám o sobě umožňuje toto směrování v rámci aplikace vytvořit, ale protože je nutné nejen vytvořit směrovací mechanismus, který by na základě parametru rozhodoval jaká komponenta (hierarchie komponent) reprezentující danou stránku bude zobrazena, ale také řešit problematiku historie prohlížeče, vznikla knihovna React-Router, která poskytuje API usnadňující práci se směrováním.

Reac-Router knihovna zpřístupňuje několik komponent, které umožňují velmi jednoduše definovat vlastní směrovací pravidla a současně ukládat každou změnu URL do historie prohlížeče, aby uživatel mohl využívat mechanismus tlačítek „Zpět a vpřed“ (Dorr, a další, 2017).

Zdrojový kód 1: Realizace směrování

```
1. <Router history={browserHistory} >
2.  <Route path="/" component={App} >
3.   <IndexRoute component={Home} />
4.   <Route path="measurement" component={Measurement} onEnter={authCheck} />
5.   <Route path="unsaved" component={Unsaved} onEnter={authCheck} />
6.   <Route path="evidence" component={Evidence} onEnter={authCheck} />
7.   <Route path="settings" component={Settings} onEnter={authCheck} />
8.   <Route path="comparison" component={Comparison} onEnter={authCheck} />
9.   <Route path="measshow" component={ShowMeas} onEnter={authCheck} />
10.  <Route path="login" component={Login} onEnter={ifNotLogged} />
11. </Route>
12. <Route path="*" component={App} >
13.   <IndexRoute component={NoF} />
14. </Route>
15.</Router>
```

Komponenta Router vytvoří porovnávací mechanismus obsahující objekty (reprezentující pravidla) směrování, vkládané pomocí komponent Route. Také začne naslouchat na proměnné *window.history.location.pathname* uloženém v DOM, při každé změně této proměnné vlivem upravení URL, dojde k porovnání shody s uloženými cestami v pravidlech. Dalším parametrem komponenty je **history**, ve kterém se předává způsob, jak se bude pracovat s URL adresou nebo historií prohlížeče. Jedna z poskytovaných definic práce s historií přímo knihovnou React-Router, které se od sebe moc neliší, je *browerHistory*, která vytváří standardní vzhled výsledné URL bez použití oddělovacího symbolu „#", protože URL v podobě *https://www.exaple.com#/login* není uživatelsky přívětivá.

Komponenta Route, přidává do výsledného rozhodovacího mechanismu další objekt (pravidlo), který je definován parametrem **path**, ve kterém je uvedena podoba cesty. Je možné vytvářet skupiny, jak je vidět na předchozím zdrojovém kódu, kde každé pravidlo mimo své vlastní cesty přebírá navíc i cestu skupiny. Dalším možným parametrem je **onEnter**, do kterého je vložena funkce. Tato funkce se provede v případě, že cesta u daného objektu (pravidla) bude shodná s cestou uloženou v DOM parametru *window.history.location.pathname*. Posledním parametrem této komponenty

je **component,** ve kterém je nastavena komponenta reprezentující danou stránku nebo funkcionalitu. I s předávanými komponentami platí obdobná vlastnost jako u parametru **path**, avšak v tomto případě funguje trochu odlišně, a to tak, že komponenta definovaná pro skupinu v sobě zobrazí komponentu definovanou u neskupinového pravidla. Tedy při cestě */login* je v komponentě App zavolána komponenta Login. Komponenta IndexRoute funguje zcela shodně jako komponenta Route, ale její parametr **path** je automaticky nastavený na prázdný řetězec.

Funkce *authCheck*, která je umístěna v parametru **onEnter** u komponenty Route, ověří, jestli proměnná **isLogged,** uložená v proměnné objektu **store.user**, má nastavenou hodnotu na **True**. Pokud je tato podmínka splněna, dojde k zobrazení zvolené stránky, ale pokud nedojde ke splnění podmínky, je uživatel přesměrován na stránku s přihlášením. Funkce *ifNotLogged* slouží k zamezení přístupu přihlášeného uživatele na stránku s přihlášením. V případě, že je uživatel přihlášen, dojde k jeho přesměrování na stránku Home.

3.2 Store

Store je nejdůležitějším objektem v aplikaci, který slouží k ukládání a předávání dat v rámci celé aplikace, a to nejen komponentám, ale i komunikačnímu API pro sestavení výsledné cesty v databázi.

Pro každou část výsledného objektu Store je nutné vytvořit akce a reduktor (z angl. actions and reducer). Pomocí reduktoru se definuje změna výsledného stavu objektu Store, v závislosti na zvolené události. Reduktor je funkce vložená ve výsledném objektu Store, která na základě požadavku a dat předaných pomocí objektu akce (**actions**) bezpečně modifikuje data uložená uvnitř objektu Store.

Zdrojový kód 2: Realizace reduktoru typu user

```
const userState = {
  id: '', fullname: '', isLogged: false, token: ''
};
export default const user = (state = userState, action) => {
  switch (action.type){
    case 'LOGIN_USER':
      return { id: action.user.id, isLogged: true,
                  fullname: action.user.fullname,
                  token: action.user.token };
    case 'LOGOUT_USER':
      return { id: '', fullname: '', isLogged: false,
                  token: '' };
    case 'CHANGE_USER_TOKEN':
      return {...state, token: action.token };
    default:
      return state;
  }
};
```

Objekt akce je návratovou hodnotou definované funkce, který obsahuje požadovanou změnu a potřebná data pro modifikaci objektu Store. Jediným jeho povinným atributem je **type**, který slouží pro jednoznačnou identifikaci způsobu modifikace dat, kterou chceme provést v objektu Store. Další atributy objektu jsou nepovinné a slouží k předání dat potřebných k provedení zvolené modifikace.

Zdrojový kód 3: Realizace akcí pro reduktor typu user

```
export const userLogIn = (user)=>{
  return {type:'LOGIN_USER', user: user};
};
export const userLogOut = ()=>{
  return {type:'LOGOUT_USER'};
};
export const editUserToken = (token)=>{
  return {type:'CHANGE_USER_TOKEN', token: token};
};
```

Před vytvořením výsledného objektu Store je nutné sloučit veškeré reduktory do jednoho reduktoru pomocí funkce *combineReducers()* poskytované knihovnou Redux, kde je jako parametr funkce předán objekt, obsahující veškeré použité reduktory. Poté může být pomocí funkce *createStore()* vytvořen samotný objekt Store, kde vstupním parametrem funkce musí být sloučený

reduktor. V případě, že u jednotlivých reduktorů nejsou uvedeny inicializační stavy, mohou být předány jako další parametr funkce *createStore()*.

Zdrojový kód 4: Vytvoření objektu Store

```
1. const myotonometerApp = combineReducers({
2.   myotonometer, patient, settings, measurement, user
3. });
4. let store = createStore( myotonometerApp );
```

V této aplikaci je objekt Store tvořen z několika reduktorů:

- **patient** - obsahuje seznam objektů reprezentující pacienty a umožňuje přidávat nové pacienty do seznamu, odebírat nebo modifikovat stávající pacienty.
- **measurement** - obsahuje seznam objektů reprezentujících všechna uložená a neuložená měření. Umožňuje přidávat a odebírat měření z obou seznamů.
- **settings** - obsahuje informace (nastavující hodnoty) o nastavení, jako je jazyk webového rozhraní nebo použitý tvar a rozměr měřící hlavy. Umožňuje modifikovat jednotlivé hodnoty.
- **user** - obsahuje informace o přihlášeném uživateli, jako je jméno, identifikátor nebo token. Umožňuje změnit veškeré hodnoty po přihlášení a odhlášení, nebo změnit token kvůli vypršení jeho platnosti.
- **myotonometer** - obsahuje veškeré informace přenášené z měřicího přístroje, jako je stav přístroje nebo data získaná z měření. Umožňuje měnit veškeré hodnoty na základě jejich změny v databázi.

3.3 Jazyková lokalizace rozhraní

Důležitou částí rozhraní je jazyková lokalizace, která umožní zobrazovat webové rozhraní v několika jazycích. Pro změnu jazykové sady existuje několik různých knihoven, ale některé jsou komplikované na nastavení a zavedení

do aplikace, proto bylo vytvořeno vlastní řešení, které funguje na velice jednoduchém principu.

Pro uchování zvoleného jazyka je využita proměnná **settings.language**, která se nachází v objektu Store. Při inicializaci této proměnné je využita vlastnost DOMu *window.navigator.language*, která vrací textový řetězec obsahující zkratku nastavení jazyka systému. DOM poskytuje i vlastnost *window.navigator.languages*, která vrací pole textových řetězců reprezentujících preferované jazyky uživatele. Vrácené pole jazyků může vypadat například takto *["en", "en-US", "cs"]*, ale protože tato vlastnost není podporována u všech prohlížečů, musí být k volbě jazyka využito systémové nastavení jazyka.

Získaný jazyk je porovnán spolu s vytvořeným polem **languageData**, který se nachází v souboru **language** uloženém v adresáři */src/common/data*, obsahujícím překlady jednotlivých výrazů pro předdefinované jazyky. V případě, že získaný jazyk splní podmínku *languageData[systemLang] === undefined*, je automaticky nastaven jazyk anglický, protože daný jazyk nemá nadefinovanou jazykovou sadu. V opačném případě je nastaven jazyk, který byl získán od uživatele.

Uživateli nemusí automaticky nastavený jazyk vyhovovat a proto má možnost si jej manuálně nastavit na jazyk, který má definovanou jazykovou sadu v proměnné **languageData** pomocí komponenty **languageSwitcher** umístěné na stránce pro nastavení rozhraní.

3.3.1 Komponenta languageSwitcher

Jak bylo popsáno v předešlé kapitole, komponenta umožňuje uživateli změnit jazykovou sadu, kterou chce používat při práci s uživatelským rozhraním pro ovládání přístroje Myotonometr.

Komponenta pro svoji funkčnost využívá tzv. spojení nebo propojení (z angl. connection) s objektem Store, ve kterém je uložena hodnota aktuálně používaného jazyka v proměnné **settings.language**. Propojení je nutné provést, protože objekt Store neumožňuje přímý přístup k datům, která jsou v něm uložena. Pro spojení komponenty s objektem Store je nutné předat jako parametr funkce *connect* mapující funkce, které propojí proměnné ze Store nebo funkce pro vkládání dat do Store na proměnné umístěné v proměnných **this.props**.

První mapující funkcí je funkce *mapStateToProps(store)*, která vrací objekt popisující navázání proměnné **store.settings.language** na proměnnou **this.props.lang**. Druhá mapující funkce *mapDispatchToProps(dispatch)* vrací objekt popisující navázání funkce *dispatch(actions.changeLang(lang))* na proměnnou (poté proměnná funguje jako funkce) **this.props.changeLang(lang)**.

Samotná komponenta je tvořena jedním *html* elementem *select*, který uživateli umožňuje vybrat svůj preferovaný jazyk, obsahující dva ovládací atributy. Prvním atribut je **value,** do kterého je vložena hodnota z proměnné **this.props.lang**. Tento atribut automaticky při každém překreslení komponenty vybere možnost ze seznamu v *html* elementu *select*, která odpovídá hodnotě předané do atributu.

Druhým atributem je reakce na událost změny **onChange** obsahující funkci, která se vykoná v případě, že uživatel vybere jinou možnost ze seznamu. Tato funkce předá získanou hodnotu z vybrané možnosti (jazyka) funkci, uložené pod proměnnou **this.props.changLang**, která uloží nově vybranou hodnotu do proměnné **settings.language** ve Store. Po uložení nového jazyka je automaticky vyvoláno překreslení všech komponent, které mají připojenou proměnnou **store.settings.language**.

Obrázek 7: Ukázka komponenty languageSwitcher

3.4 Autentizace uživatele

Aplikace funguje ve dvou režimech, kdy podle toho zda je uživatel přihlášen, rozhoduje jaké funkcionality mu budou zpřístupněny. Proto je nutné provádět ověřování uživatelů na základě přihlašovacích údajů.

Přihlašování uživatelů není jenom bezpečnostním opatřením aplikace pro zabránění přístupu neoprávněným uživatelům, ale je také mechanismem pro identifikaci konkrétního uživatele, protože jeho identifikátor je součástí cesty k datům, které má uložená v databázi.

3.4.1 Na straně serveru

Server využívá pro přihlášení uživatele základní funkci *get*, kterou poskytuje databáze. Protože databáze neumožňuje vyhledat klienta na základě jeho přihlašovacích údajů jako například SQL databáze pomocí vložené podmínky v dotazu. Musí dojít k periodickému procházení jednotlivých uživatelů a porovnávání jejich uživatelského jména. Pokud není příslušné uživatelské jméno v databázi, na stranu klienta je odeslána odpověď obsahující informace o této chybě a přihlašování se ukončí. V případě, že se uživatelské jméno v databázi nachází, dochází k porovnání hesel. Pokud hesla nejsou stejná, je přihlašování ukončeno a uživateli je odeslána zpráva obsahující informaci o chybě.

V opačném případě je vytvořená odpověď, která obsahuje nejen informace jako je identifikátor uživatele, vytvořený token, ale také veškeré

informace o pacientech uložených v uživatelově databázi spolu se všemi uloženými měřeními.

3.4.2 Na straně klienta

Uživatel se přihlašuje skrze stránku s přihlášením popsanou v kapitole 3.5.1. Vyplněné přihlašovací údaje klienta jsou odeslány na server jako žádost o přihlášení, následně klient vyčká na odpověď serveru.

V případě, že přihlášení proběhne v pořádku, odpověď obsahuje veškeré potřebné informace, které jsou uloženy do příslušných proměnných v objektu Store. Současně je automaticky aktivována funkce, která každých patnáct minut požádá server o vytvoření nového tokenu. V neposlední části přihlašování je do proměnné **user.isLogged** ve Store uložena logická hodnota **True**, která při směrování v aplikaci slouží pro ověření, zda je uživatel přihlášen. Nakonec je uživatel přesměrován na stránku pro vytvoření nového měření, která je přístupná pouze v režimu pro přihlášeného uživatele.

V opačném případě, odpověď obsahuje informaci (chybovou zprávu), proč se přihlášení nepodařilo, která je uložena do proměnné **user.status** v objektu Store a zobrazena uživateli.

3.5 Vytvořené rozhraní

Výsledné rozhraní pro ovládací měřicího přístroje Myotonometr je tvořeno několika stránkami, které reprezentují jednotlivé funkcionality znázorněné pomocí grafu užití. Každá stránka je tvořena kombinací několika jednotlivých komponent, které realizují vždy nějakou dílčí část výsledné stránky a její funkcionality. Proto v následující části práce budou popsány jednotlivé výsledné stránky a jejich aplikační logika bez podrobnějšího popisu všech dílčích komponent.

3.5.1 Přihlášení

Stránka realizuje funkcionalitu pro přihlášení uživatele k samotnému systému a tím mu zpřístupní možnost využívat další funkcionality a nástroje poskytované vytvořeným uživatelským rozhraním, jako je například manipulace s databází na straně serveru.

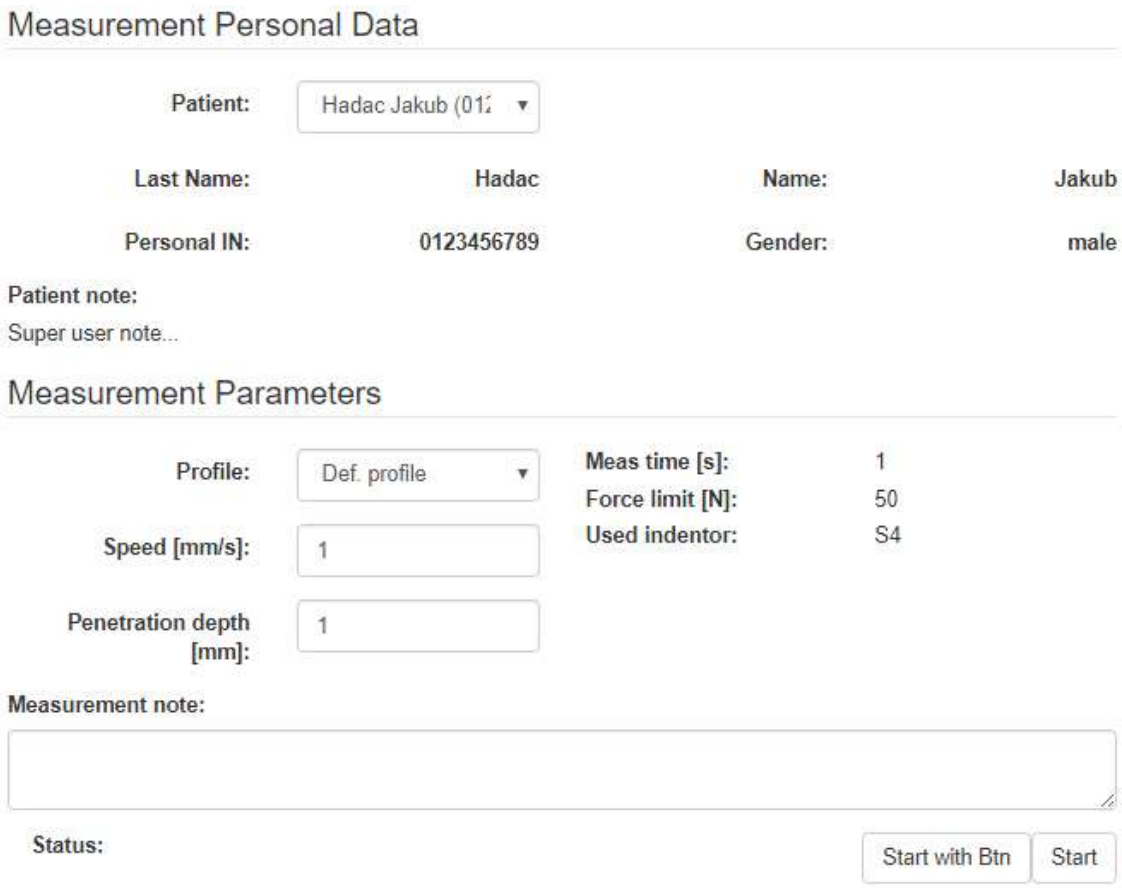
Stránka obsahuje pouze jednu komponentu, která je tvořena dvěma *html* elementy *input*. První vstupní element slouží k zadání uživatelského jména a druhý k zadání příslušného hesla. Dále obsahuje tlačítko, pomocí kterého uživatel vyvolá odeslání žádosti na server. Poslední částí stránky je informační řádek označený textem „Status:“, ve kterém jsou zobrazeny veškeré chybové zprávy.

3.5.2 Měření

Jedna z nejdůležitějších stránek celé aplikace, která zajišťuje možnost jak nadefinovat potřebné parametry měření, to je následně odesláno na server a tam provedeno. Poskytuje i možnost snadno přidat nového pacienta tak, aby uživatel nemusel přepínat mezi funkcionalitami aplikace.

Stránka obsahuje dva kontejnery, které umožní jednoduše a velice rychle nastavit potřebné parametry k vykonání měření. Prvním kontejnerem je komponenta umožňující výběr pacienta pomocí *html* elementu *select* ze seznamu pacientů. Pokud pacient není v seznamu, stačí v seznamu vybrat položku nový pacient (v ang. New patient), která automaticky povolí editační prostředí komponenty. Pro vytvoření nového pacienta je nutné vyplnit jméno a příjmení pacienta, jeho rodné číslo pomocí *html* elementů *input* a pohlaví pomocí *html* elementu *select*. Nepovinným údajem je poznámka k pacientovi, která se vkládá pomocí *html* elementu *texarea*, a umožňuje uživateli doplnit k pacientovi doplňující informace. Po stisknutí tlačítka pro uložení je ověřena validita dat a jsou odstraněny veškeré nepovolené znaky vložené do textu.

Pokud jsou data v pořádku a ve správném formátu, jsou odeslána na server, kde jsou uložena v databázi pacientů na příslušné místo pro daného uživatele. Je-li pacient v pořádku uložen do databáze, provede se jeho automatický výběr, který opět uzamkne editovací prostředí komponenty. V případě, že data nesplňují veškeré podmínky pro jejich správnost nebo se nepodařilo uložit pacienta do databáze, dojde k informování uživatele o dané chybě pomocí chybové zprávy zobrazené ve spodní části komponenty označené textem „Status:“.



Obrázek 8: Ukázka stránky pro vytvoření nového měření

Druhý kontejner, který je zobrazen pouze v případě, že uživatel zvolil pacienta, umožňuje nastavit parametry samotného měření. Pomocí *html* elementu *select* je možné vybrat jeden z předem nadefinovaných profilů pohybu, které ovlivňují způsob pohybu měřící sondy přístroje. Dalšími

parametry, které je nutné nastavit pomocí *html* elementů *input*, jsou rychlost pohybu a maximální hloubka průniku měřící hlavy. Současně jsou zobrazeny další informace, které jsou získány z nastavení, jako je typ a rozměry použité měřící sondy a maximální použitá síla přístroje, nebo jsou dopočítány na základě nastavených hodnot, jako je například doba měření.

Po nastavení všech hodnot potřebných pro vytvoření měření jsou uvolněna tlačítka, kterými lze zvolit, jestli se má měření spustit automaticky nebo vyčkat na stisknutí tlačítka umístěného na měřicím přístroji.

Ze strany serveru dostává uživatel informace o stavu měření, které se zobrazují ve spodní části stránky označené textem „Status:“. V případě že měření skončí chybou, může uživatel měření spustit opětovně. Pokud měření proběhne v pořádku, jsou veškerá potřebná a získaná data reprezentující dané měření uložena v objektu Store. Následně je uživatel automaticky přesměrován na stránku se zobrazeným výsledkem měření obsahující i výsledný graf.

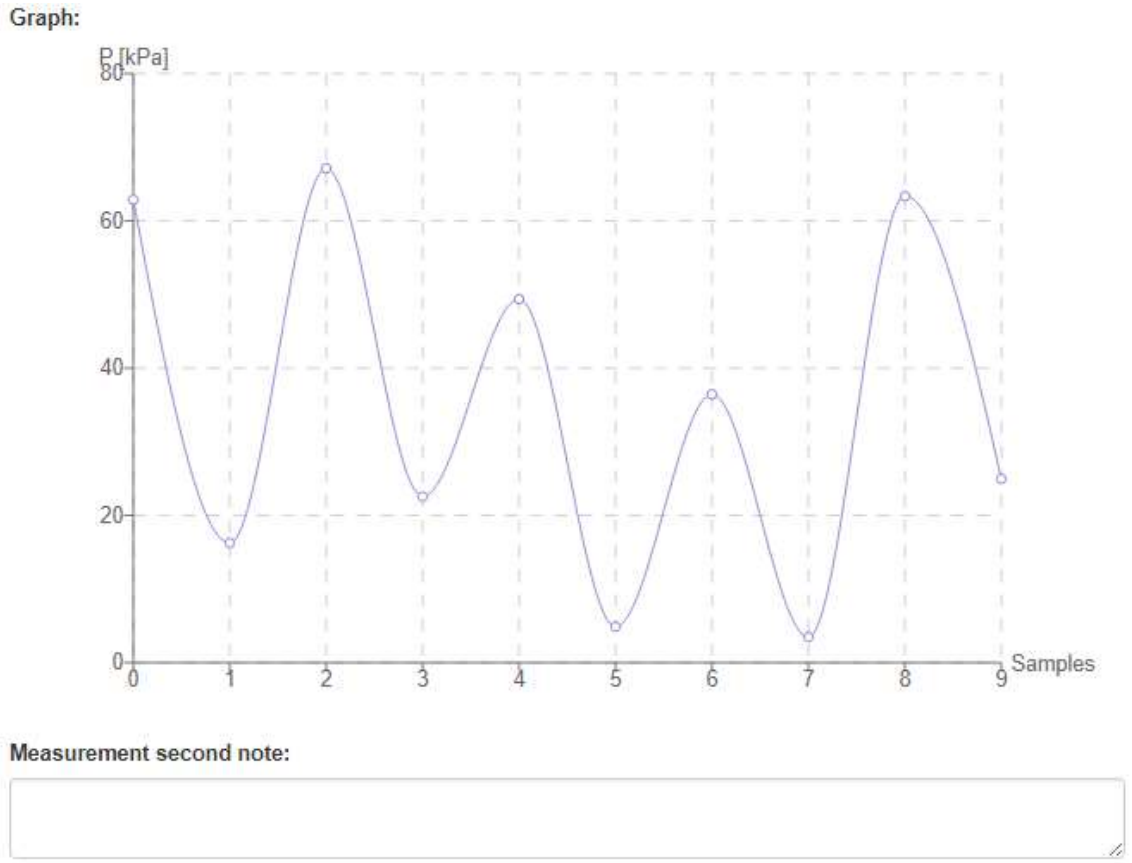
3.5.3 Zobrazení výsledného měření

Stránka realizuje funkcionalitu pro zobrazení výsledků provedeného měření do jednoduchého protokolu o měření. Tato stránka pracuje ve dvou režimech, kde první režim slouží pro zobrazení a možnost uložení neuloženého měření a druhý režim pro zobrazení již uloženého měření.

Kontejner reprezentující stránku je tvořen pomocí základních *html* elementů, pomocí kterých jsou uživateli zobrazeny informace o pacientovi a nastavující parametry daného měření. Dále je na stránce umístěna komponenta realizující vykreslení grafu z naměřených hodnot do *html* elementu *canvas* pomocí knihovny Recharts (RechartsGroup, 2016).

V případě, že se jedná o režim zobrazení neuloženého měření, je na stránce umístěn *html* element *textarea* pro možnost vložení poznámky k měření. Ve druhém případě je tato poznámka pouze zobrazena jako textová informace.

Ve spodní části stránky se nacházejí tlačítka pro uložení neuloženého měření do databáze k danému pacientovi, dále pak tlačítko pro možnost vyexportování protokolu o měření ve formátu PDF. Poslední tlačítko slouží k opakování zobrazeného měření. Dojde k přesměrování uživatele na stránku pro vytvoření nového měření, ale současně jsou automaticky nastaveny veškeré parametry měření stejně, jako jsou nastaveny v případě zobrazeného měření.



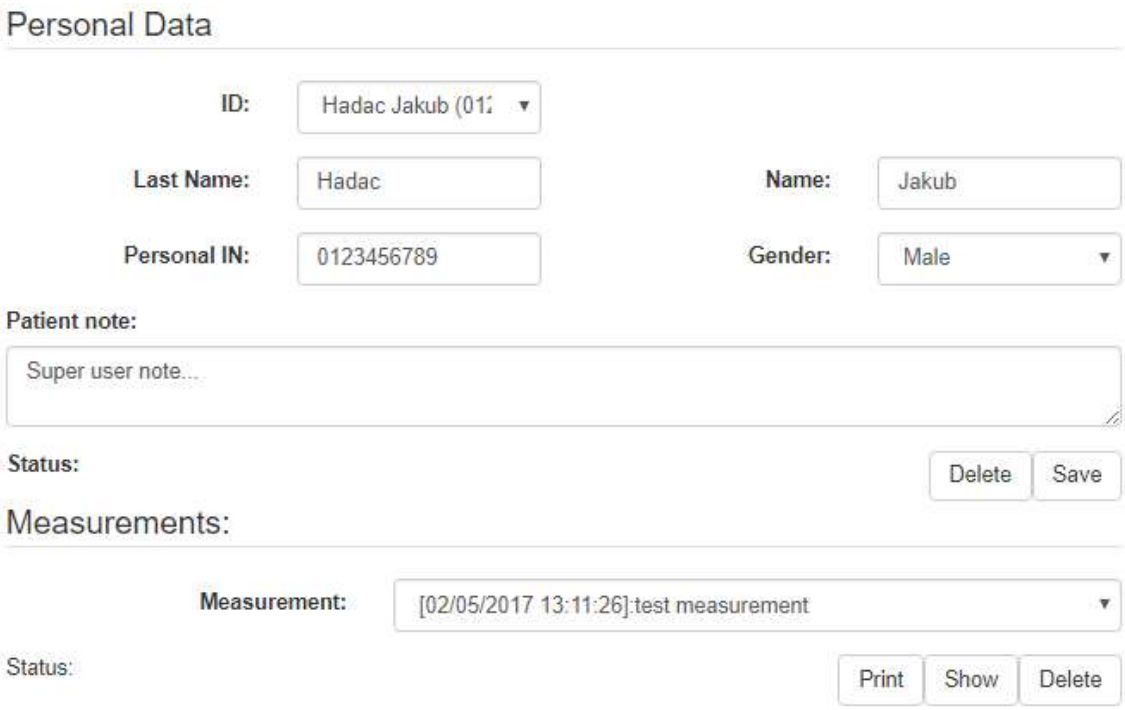
Obrázek 9: Část stránky pro zobrazení neuloženého měření

3.5.4 Evidence

Stránka realizuje funkcionalitu pro správu pacientů a jejich měření. Umožňuje tedy uživateli přidávat nové pacienty, upravovat anebo mazat stávající pacienty. Dále umožňuje u vybraného pacienta mazat měření nebo přesměrovat se na stránku popsanou v předchozí kapitole, kde si dané měření může prohlédnout.

Stránka je tvořena ze dvou kontejnerů, které ji rozdělují na část pro správu pacientů a část pro správu měření vybraného pacienta. První kontejner je komponenta realizující správu pacientů, která umožňuje výběr pacienta ze seznamu pomocí *html* elementu *select*. V případě, že uživatel zvolil ze seznamu pacienta, je komponenta nastavena do editačního režimu. V tomto režimu jsou do jednotlivých html elementů pro zadávání informací ohledně pacienta vloženy jednotlivé hodnoty uložené u vybraného pacienta. Následně pak mohou být upraveny a uloženy pomocí tlačítka „Uložit“, nebo může být pacient odstraněn společně se všemi měřeními pomocí tlačítka „Smazat“. Pokud je v seznamu pacientů vybrána položka „Nový pacient“ chová se komponenta stejně jako v případě stránky pro vytvoření měření popsané v kapitole 3.5.2.

Při ukládání nového pacienta nebo změn u stávajícího pacienta na server dochází k ověření správného formátu dat a odstranění všech nevhodných znaků vložených do vyplněných hodnot. V případě, že data nemají správný formát nebo při ukládání na server dojde k nějaké chybě, je o dané situaci uživatel informován ve spodní části komponenty označené textem „Status:“. Do stejného místa jsou vložena oznámení, která informují uživatele, že provedená akce proběhla v pořádku.



Obrázek 10: Ukázka stránky pro správu pacientů a jejich měření

Druhý kontejner je komponenta zobrazená pouze v případě, že v kontejneru pro správu pacientů je zvolen pacient. Tato komponenta na základě identifikátoru vybraného pacienta z prvního kontejneru, získá veškerá provedená a uložená měření daného pacienta. Pokud pacient nemá žádná uložená měření, je o tom uživatel informován pomocí zobrazené zprávy. Ale v případě, že pacient nějaká uložená měření má, je uživateli zobrazen seznam všech těchto měření pomocí seznamu realizovaným *html* elementem *select*. Po zvolení měření dojde k zobrazení tlačítek, která umožňují uživateli dané měření zobrazit detailněji, vyexportovat do přenositelného protokolu ve formátu PDF nebo ho odstranit.

3.5.5 Porovnání měření

Stánka realizuje funkcionalitu pro porovnání dvou měření od jednoho pacienta, které se uživateli zobrazí do grafu, ve kterém si může obě měření porovnat.

Stránka je rozdělena na dvě části, které realizují výslednou funkcionalitu. První část stránky je tvořena komponentou pro výběr pacienta, která je tvořena jediným interaktivním *html* elementem *select*. Pomocí tohoto elementu uživatel vybere pacienta ze seznamu pacientů, u kterého chce provést porovnání. Po zvolení pacienta je automaticky zobrazen kontejner realizující druhou část výsledné funkcionality stránky.

Obrázek 11: Část stránky pro porovnání měření

Kontejner, který tvoří druhou část výsledné funkcionality stránky, je tvořen pomocí dvou komponent. První komponenta umožňuje vybrat dvě různá měření, která chce uživatel porovnat, ze seznamu obsahujícího veškerá měření uložená u vybraného pacienta, pomocí dvou *html* elementů *select*. Je-li v jednom elementu vybráno měření, je automaticky odstraněno z možností druhého elementu, aby nemohlo dojít k tomu, že uživatel bude porovnávat stejné měření. Po zvolení obou měření dochází automaticky k vykreslení

výsledného grafu pomocí knihovny Recharts a zobrazení informací o obou zvolených měřeních. V případě, že zvolený pacient nemá alespoň dvě uložená měření, je uživatel informován o tom, že u tohoto pacienta nemůže provést porovnání měření.

3.5.6 Neuložená měření

Stránka s neuloženými měřeními zobrazuje všechna provedená měření, která nebyla uložena do databáze k daným pacientům. Tato neuložená měření jsou jen dočasná, respektive pokud se uživatel odhlásil nebo ukončil aplikaci, jsou tato měření automaticky odstraněna.

Na stránce je zobrazen seznam všech neuložených měření ve formě tabulky obsahující jméno pacienta, čas a datum, kdy dané měření proběhlo a dvě tlačítka pro volbu operace, kterou chce uživatel s měřením provést. První tlačítko umožňuje odstranit dané měření ze seznamu neuložených měření zobrazeného na stránce. Druhé tlačítko přesměruje uživatele na stránku, kde je vybrané měření zobrazeno společně s výsledným grafem, kde si jej může uživatel prohlédnout a případně uložit k danému pacientovi.

3.5.7 Nastavení

Stránka realizuje funkcionalitu pro nastavení samotného ovládacího uživatelského rozhraní, jako je například preferovaný jazyk nebo maximální hodnoty parametrů měření, které bude možné použít v rámci stránky pro vytvoření nového měření.

Stránka je tvořena několika *html* elementy *select*, které uživateli umožňují vybrat vhodnou variantu daného nastavení a tu následně uložit do příslušné proměnné z proměnné **settings** v objektu Store.

4 Ověření systému

V této části diplomové práce budou nastíněny některé koncepce testování výsledné aplikace, realizující uživatelské rozhraní pro měřicí přístroj Myotonometr. Jde například o testování funkčnosti a použitelnosti aplikace, dále také testování bezpečnosti aplikace a dat v ní uložených.

4.1 Funkčnost aplikace

Testování funkčnosti aplikace bylo soustředěno na otestování, zda jsou splněny veškeré požadavky kladené v kapitole 2.1, které aplikace musí splňovat.

Během vývoje aplikace probíhalo testování nejen požadavků kladených na aplikaci ale také veškerých funkcionalit, které aplikace uživateli umožňuje využívat. Dále docházelo k testování správnosti odesílaných dat na stranu serveru, nebo správnosti formátu dat předávaných dané funkcionalitě.

4.1.1 Použitelnost aplikace

Testováním použitelnosti aplikace znamená, že aplikace byla odzkoušena nejen během samotného vývoje, ale také malou skupinou uživatelů, kteří měli vyzkoušet řešení několika úkolů podle předem definovaného scénáře.

Vytvoření scénáře, který by pokryl všechny problematiky aplikace, je poměrně obtížné, protože je nutné nejen otestovat veškeré hlavní cíle aplikace, ale také možné kroky, který by mohl chtít uživatel vykonat (Churm, 2012).

Výsledný scénář pro testování použitelnosti vypadal takto:

- Přihlásit se pomocí zadaných přihlašovacích údajů
- Vytvořit a spustit měření pro pacienta „Jakub Hadac“
- Uložit výsledné měření
- Porovnat dvě různá měření u pacienta „Jakub Hadac“

S těmito úkoly nenastaly žádné problémy, pravděpodobně protože přihlášení spočívalo ve vyplnění získaných informací do dvou vstupních polí. V případě, že uživatel opsal tyto informace špatně, byl o dané chybě informován a okamžitě ji opravil.

Vytvoření, spuštění a uložení nového měření nedělalo žádnému uživateli problémy, neboť uživatel je mezi těmito volbami automaticky přesměrováván a tak mu stačilo klikat na vhodná tlačítka. Některým uživatelům chyběl ukazatel o stavu probíhajícího měření, který byl na základě tohoto testu zpětně přidán. Protože aplikace umožňuje velice rychlé a jednoduché ovládání, porovnání měření nedělalo problém žádnému uživateli a většině z nich nezabralo ani třicet sekund.

4.2 Stabilita aplikace

Největším problémem z hlediska stability systému jsou nevhodná data, tedy přesněji data, která neodpovídají potřebnému formátu. Taková data by mohla způsobit zamrznutí aplikace (stav kdy aplikace zaznamená kritickou chybu, která zabrání dalšímu pohybu v aplikaci) nebo nezobrazení požadované stránky. Testování tohoto problému probíhalo během celého vývoje aplikace a docházelo k ošetřování veškerých možných chyb, které v průběhu testování nastaly.

Například ze strany serveru byly úmyslně zaslány neúplné informace ohledně nově přidaného pacienta. Tyto upravené informace neobsahovaly identifikátor pacienta, v případě, že by byla tato data uložena do proměnné

patient.patients v objektu Store, neznamenalo by to okamžitě ohrožení stability aplikace. Ale pokud by se uživatel snažil zobrazit si získané měření, kde je daný identifikátor uložen a slouží k vyhledání informací o daném pacientovi, aplikace by skončila chybou a nezobrazila by žádný výsledek, protože by nebyl nalezen pacient s daným identifikátorem.

Proto dochází ke kontrole všech příchozích dat ze strany serveru na správnost a úplnost všech informací, která data musí obsahovat. Pokud data neobsahují veškeré potřebné informace, nedojde k jejich uložení do objektu Store.

4.3 Dostupnost

Výsledná aplikace je přístupná pouze z privátní sítě vytvářené přístrojem pomocí WiFi modulu, ale i přesto proběhlo testování aplikace na její dostupnost. Standardně se pro tyto testy využívá několik různých metod jako například testování pomocí příkazu ping, nebo testy navázání spojení přes HTTP/HTTPS atp. Pro otestování této aplikace byl vytvořen jednoduchý test, který spočíval v periodickém dotazování serveru o data.

Spuštěný klient po dobu dvanácti hodin automaticky odesílal každých pět minut požadavek na stranu serveru (přístroje) o poskytnutí příjmení zvoleného pacienta. Každým odeslaným dotazem se navýšila hodnota pro odchozí zprávy a s každou správnou odpovědí na daný požadavek, tedy odpověď obsahující správné příjmení, se navýšila hodnota pro správné odpovědi. Výsledkem toho testu byl poměr, udávající z kolika procent se systému podařilo vyřídit daný požadavek. Během testu bylo odesláno 145 požadavků a zpět přišlo stejné množství správných odpovědí, tedy systém byl dostupný ze 100 procent.

4.4 Bezpečnost

Na základě analýzy bezpečnostních hrozeb výsledné aplikace došlo k prozkoumání některých typů útoků. Tyto útoky pak prošly otestováním nebo byly zamítnuty kvůli jejich nemožnosti provedení. Například útok SQLi (SQL injection), který slouží k napadení databázové vrstvy systému pomocí pozměnění prováděného SQL dotazu skrze neošetřené vstupy, nemůže být v této aplikaci proveden, protože databáze není řízena SQL dotazy. Jediným možným způsobem jak provést obdobu toho útoku v rámci aplikace, je pozměnit hodnoty parametru **location** předávaného jako součást dotazu na server. Největším bezpečnostním rizikem byl vyhodnocen útok typu XSS.

4.4.1 Útoky typu XSS

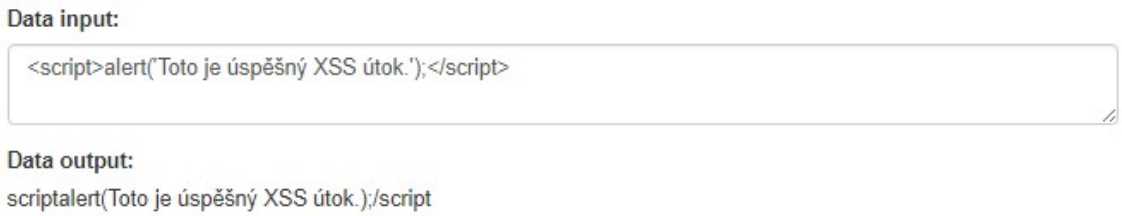
Útok spočívá v principu vkládání tzv. cizích skriptů (Cross-site scripting - XSS) do aplikace pomocí neošetřených vstupních datových polí, nebo neošetřených vstupních dat z databáze, kdy útočník podstrčí svůj vlastní JavaScriptový kód, který se po načtení nebo překreslení stánky provede a tak může například poškodit výsledný vzhled stránky nebo v nejhorším případě získat přístup k citlivým datům uloženým v aplikaci (Assis, 2016).

Během samotného vývoje aplikace docházelo k testování toho typu útoku pomocí vkládání řetězce „<script> alert('Toto je úspěšný XSS útok.'); </script>“ do jednotlivých vstupních polí, který by při svém úspěšném provedení způsobil zobrazení dialogového okna s textem „Toto je úspěšný XSS útok.“. Takto navržený skript je dostatečný aby prokázal, že aplikace nese bezpečnostní rizika z hlediska XSS útoku.

Proti tomuto útoku je ošetřen každý *html* element vstupních polí formulářů v celé aplikaci pomocí HTML5 atributů, například pomocí **type="number",** který umožní do daného pole vkládat pouze číslice. Další atribut sloužící k ošetření vstupních polí proti tomuto útoku je **maxLength**,

který nastavuje maximální počet znaků, které je možné vepsat do příslušného vstupního *html* elementu (Hunt, 2010).

Dalším způsobem ošetření je využití regulárního výrazu společně s JavaScriptovou funkcí *replace(/<|'|>/g, "")*, která z řetězce odstraní veškeré znaky nebo části řetězce odpovídající danému regulárnímu výrazu. Tímto jednoduchým způsobem dojde k zamezení vložení škodlivého kódu do výsledné aplikace skrze vstupní formulářové prvky nebo skrze neošetřená vstupní data získaná ze strany serveru (Zakas, 2009).



Obrázek 12: Ošetřený vstupní element proti útoku typu XSS

Dalším možným způsobem jak provést XSS útok na aplikaci, je využít předávání parametrů v URL adrese. V případě této aplikace by škodlivá URL adresa vypadala takto: „*https://www.expample.com/measurement/<script> alert('Toto je úspěšný XSS útok.'); </script>*“. Tento způsob předání škodlivého kódu pomohla vyřešit knihovna React-Router popsaná v kapitole 3.1. Směrování v aplikaci je řešeno pomocí absolutních cest, kde nejsou předávány žádné parametry. Proto v tomto případě je útočník přesměrován na stránku informující ho, že žádná stránka (funkcionalita) s takto zadanou cestou v systému není a proto se vykonání škodlivého kódu neprovede.

4.4.2 SPA aplikace

Samotnou bezpečností hrozbou je i webová aplikace typu SPA, přesněji za tuto hrozbu z hlediska bezpečnosti může interpretovaný skriptovací jazyk JavaScript. Protože útočník si může daný JavaScriptový soubor obsahující

veškerou funkcionalitu dané aplikace uložit a následně upravovat podle své potřeby a tak získávat informace o fungování nejen samotné aplikace, ale také fungování serveru.

V případě, že je daný soubor distribuován bez jakéhokoliv zásahu, tedy soubor je výsledkem sloučení všech zdrojových souborů a překladem na kompatibilní funkce, útočník není nijak omezen v možnostech, jak s daným kódem zacházet a upravovat ho. Soubor obsahuje veškeré názvy proměnných, funkcí a také veškeré komentáře k dané logice, a proto poskytuje útočníkovi dobré možnosti jak efektivně vytvořit přesně cílený útok na danou aplikaci.

V této práci je pro výsledné sestavení aplikace využit balíčkovací model Webpack, který umožní sloučit zdrojové kódy aplikace do jednoho výsledného souboru, ale také smaže veškeré vložené komentáře a bílé mezery. Dále pak s využitím nastavení pro minimalizaci (kompresi) výsledného kódu dochází k nahrazení všech názvů proměnných a neintegrovaných funkcí na co nejkratší strojově generované řetězce. Výsledný soubor je tvořen jediným dlouhým řetězcem kódu, který obsahuje nic neříkající označení pro funkce a proměnné.

Tento způsob ochrany kódu nezaručuje ochranu proti jeho úpravě, ale prodlouží čas, který útočník potřebuje k vytvoření vhodné úpravy kódu. Útočník musí nejdříve provést analýzu získaného kódu pro získání logiky aplikace a až pak může realizovat vhodné úpravy kódu. Pro zabezpečení tohoto útoku je využita privátní šifrovaná WiFi síť opatřená heslem, kterou přistroj vytváří pomocí WiFi modulu. Tímto způsobem útočník musí nejdříve překonat zabezpečení sítě, než se dostane k samotnému zdrojovému kódu aplikace poskytovanému serverem měřicího přístroje.

5 Zhodnocení řešení

Navržený systém realizující ovládací rozhraní pro měřicí přístroj Myotonometr procházel během celého vývoje testováním, které mělo zaručit správnou funkčnost systému. Nejprve docházelo k testování komunikace mezi klientem a serverem, za účelem ověření správného dodržení navrženého formátu předávaných dat v jednotlivých zprávách. Následně docházelo k samostatnému testování jednotlivých dílčích funkcionalit navrženého rozhraní kvůli snadnějšímu nalezení a odladění chyb v aplikační logice pro danou funkcionalitu. Testováno bylo i správné zobrazení jednotlivých komponent jakožto celku nebo zobrazení daných komponent vzhledem k stavu aplikační logiky dané stránky, ve kterém se uživatel nacházel. Například aby došlo k automatickému skrytí komponenty pro zadání nastavujících parametrů vytvářeného měření v případě, že uživatel přepne vybraného pacienta na položku pro vytvoření nového pacienta. Velkou částí všech testů bylo testování bezpečnosti aplikace a to především proti možným útokům, které by mohly nejen poškodit pouhý vzhled výsledného rozhraní, ale především získat důvěrná data.

U původního návrhu webového rozhraní se nepočítalo s využitím rozšíření frameworku ReactJs o knihovnu Redux, která umožňuje uchovávat data v jednom úložišti a následně tato data poskytovat potřebným komponentám nebo nástrojům, jako například komunikace pomocí WebSocketů. Tento návrh se ukázal jako nepraktický z hlediska údržby, protože výsledný kód byl nepřehledný a předávání dat bylo závislé na datech, které daná komponenta získala z jiné komponenty. V případě rozšiřování komponenty o nová data, bylo nutné projít veškeré komponenty, které danou komponentu zobrazují a zajistit, aby byla potřebná data předána. Proto bylo nutné přepracovat návrh aplikace do podoby využívající knihovnu Redux,

výsledný systém je mnohem jednodušeji rozšířitelný a komponenty mají přístup k datům bez ohledu na to, kde jsou zobrazeny.

Velkou výhodou při vytváření rozhraní byl samotný měřicí přístroj Myotonometr, který byl vylepšován souběžně s vývojem samotného ovládacího rozhraní. Proto bylo možné jednoduše rozšířit server o navržené komunikační rozhraní nebo mechanismy pro práci s databází při přihlašování.

V neposlední řadě je samotné ovládání výsledného ovládacího uživatelského rozhraní a jednotlivých funkcionalit, které je navrženo jako jednoduché a intuitivní. Snahou bylo minimalizovat počet úkonů nebo množství zadávacích parametrů nutných pro provedení zvolené funkcionality, tak aby uživatel neztrácel čas zbytečným a zdlouhavým manipulováním s jednotlivými nastaveními. Nebylo by vhodné, aby uživatel musel několik minut nastavovat veškeré potřebné i nepotřebné parametry a poté provedl několika- sekundové měření. Uživateli jsou zpřístupněny pomocné funkce, které mají za úkol urychlit jeho práci s rozhraním a ovládáním jednotlivých funkcionalit, například možnost zobrazit si provedené měření. Při stisku tlačítka pro opakování měření dojde k automatickému nastavení všech parametrů pro vytvoření nového měření tak, jako jsou u zobrazeného měření.

Závěr

Hlavním cílem této práce bylo navrhnout a realizovat výsledné grafické uživatelské rozhraní pro ovládání přístroje, které v rámci webového prohlížeče umožní uživateli nastavit a vytvořit nové měření. Dále pak ovládat funkcionality umožňující uživateli manipulovat s daty uloženými v databázi přístroje. Uživatelské rozhraní je navrženo s využitím nástrojů pro tvorbu webových stránek JavaScript, HTML a CSS, které byly stanoveny v rámci zadání práce. Na základě stanovených podmínek a analýzy potřebných funkcionalit bylo realizováno výsledné rozhraní, které je rozděleno na jednotlivé stránky, reprezentující dílčí funkcionality, poskytované uživateli pro ovládání přístroje a práci s daty uloženými v databázi. Navržené rozhraní umožňuje nastavit a vytvořit měření, zobrazit měření, uložit nebo odstranit provedené měření, vyexportovat provedené měření do protokolu ve formátu PDF, provádět správu pacientů a jejich měření a správu nastavení samotného uživatelského rozhraní.

Dalším cílem práce bylo vytvoření jednoduchého a přehledného ovládání webového rozhraní, které neomezuje uživatele v jeho práci se samotným přístrojem nebo získanými daty. Proto má rozhraní velmi jednoduché menu, které uživateli umožní okamžitě nalézt a využít zvolenou funkcionalitu. Dále každá funkcionalita obsahuje minimální počet nastavujících parametrů pro své ovládání, uživatel tedy nebude zdlouhavě vyplňovat všechny potřebné parametry. Součástí výsledné aplikace jsou i funkce, které uživateli ulehčí práci, jako je automatické doplnění nastavujících parametrů pro nové měření při opakování měření.

Posledním cílem práce bylo provést otestování výsledného webového rozhraní a tak předejít mnoha různým chybám, které by nastaly z důvodu zavedení špatných nebo neúplných dat do aplikace a také díky útokům, které

by mohl útočník na dané webové rozhraní využít za účelem poškození aplikace nebo získání informací.

Výsledný systém skládající se z měřicího přístroje a webového uživatelského rozhraní je navržený pro měření svalového napětí (odporu) měkkých tkání v lékařství, ale tento systém by mohl být uplatněn i v jiných oborech jako je například textilní průmysl, kde by mohl být celý systém využit pro měření elastických vlastností látek.

Dále by systém mohl být vylepšen o databázi poskytující mnohem sofistikovanější možnosti vyhledávání, jako například využívání dotazů s podmínkami. Tak by bylo možné přiřadit jednoho pacienta k úpravám více lékařům (uživatelům) a zabránit vzniku duplicitních dat v databázi.

Seznam použité literatury

Abramov, Dan, Dorr, Timm a Clark, Andrew. 2017. Redux. [Online] 2. únor 2017. [Citace: 2. únor 2017.] http://redux.js.org/.

Assis, Rudolfo. 2016. Ask Sucuri: What is an XSS Vulnerability? *Securi blog.* [Online] Sucuri Inc. , 20. duben 2016. [Citace: 14. březen 2017.] https://blog.sucuri.net/2016/04/what-is-an-xss-vulnerability.html.

Aubourg, Julian, a další. 2016. XMLHttpRequest Level 1. *w3.org.* [Online] W3C, 6. říjen 2016. [Citace: 16. prosinec 2016.] https://www.w3.org/TR/XMLHttpRequest/.

Dorr, Tim, Florence, Ryan a Jackson, Michael. 2017. GitHub - ReactTraining/react-router: Declarative routing for React. *GitHub.* [Online] GitHub, Inc, 13. leden 2017. [Citace: 13. leden 2017.] https://github.com/ReactTraining/react-router.

Facebook. 2016. React - A JavaScript library for building user interfaces. *facebook.github.io.* [Online] Facebook Inc, 17. prosinec 2016. [Citace: 17. prosinec 2016.] https://facebook.github.io/react/.

Freed, Tony. 2016. What is Virtual Dom - Cardlife Blog. *cardlife.blog.* [Online] 11. červen 2016. [Citace: 5. prosinec 2016.] https://cardlife.blog/what-is-virtual-dom-c0ec6d6a925c.

Garrett, Jasse James. 2005. Ajax: A New Approach to Web Applications. *Internet Archive: Wayback Machine.* [Online] Wayback Machine, 18. únor 2005. [Citace: 2. prosinec 2016.] https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php.

Hickson, Ian. 2012. The WebSocket API. *w3.org.* [Online] W3C, 20. září 2012. [Citace: 23. prosinec 2016.] https://www.w3.org/TR/websockets/.

Hunt, Lachlan. 2010. HTML5 Reference. *www.w3c.org.* [Online] W3C, 10. srpna 2010. [Citace: 20. února 2017.] http://dev.w3.org/html5/html-author/.

Churm, Thomas. 2012. An Introduction To Website Usability Testing. *UsabilityGeek.com.* [Online] UsabilityGeek, 9. červenec 2012. [Citace: 18. březen 2017.] http://usabilitygeek.com/an-introduction-to-website-usability-testing/.

Ingram-Westover, Anthony. 2015. Technically Speaking: The Pros and Cons of Single Page Applications (SPAs). *Dialog Tech.* [Online] DialogTech, 10. červen 2015. [Citace: 13. prosinec 2016.] https://www.dialogtech.com/blog/technically-speaking/technically-speaking-the-pros-and-cons-of-single-page-applications-spas.

Koch, Peter-Paul. 2001. The Document Object Model: an Introduction. *http://www.digital-web.com.* [Online] Digital Web Magazine, 14. května 2001. [Citace: 3. listopad 2016.] http://www.digital-web.com/articles/the_document_object_model/.

Krajka, Bartosz. 2015. The difference between Virtual DOM and DOM. *React Kung Fu.* [Online] 12. říjen 2015. [Citace: 13. prosinec 2016.] http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/.

Kysela, Martin a Kolář, Matěj. 2016. Myotonometer – Device for Measurements of Viscoelastic Characteristics of Soft Tissues. *ELEKTRO 2016 – 11th International Conference, Proceedings.* 2016, stránky 556 – 560.

Lampa, Petr. 2002. Protokol HTTP. *www.fit.vutbr.cz.* [Online] FIT VUTBR, 26. září 2002. [Citace: 17. února 2017.] http://www.fit.vutbr.cz/~lampa/WWW/http.html.cs.

Le Hégaret, Philippe, Whitmer, Ray a Wood, Lauren. 2002. Document Object Model (DOM). *www.w3.org.* [Online] W3C, 19. leden 2002. [Citace: 4. prosinec 2016.] https://www.w3.org/DOM/.

Oshiro, Rafael. 2014. HTML5 pushState and Single-Page apps. *FrontEnd Journal.* [Online] 14. leden 2014. [Citace: 9. leden 2017.] https://www.frontendjournal.com/html5-pushstate-and-single-page-apps/.

RechartsGroup. 2016. Recharts - A composable charting library built on React components. *Recharts.* [Online] Recharts Group, 29. leden 2016. [Citace: 15. únor 2017.] http://recharts.org/#/en-US.

Shimanovsky, Serge. 2015. Multi page web applications vs. single page web applications. *eikospartners.com.* [Online] Eikos Partners, 9. červenec 2015. [Citace: 15. prosinec 2016.] http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications.

Štrauch, Adam. 2012. Memcached: zrychlete svůj web pomocí cache. *root.cz.* [Online] Internet Info, s. r. o., 7. prosinec 2012. [Citace: 11. leden 2017.] https://www.root.cz/clanky/memcached-zrychlete-svuj-web-pomoci-cache/.

Takada, Mikito. 2012. Single page apps in depth. *singlepageappbook.* [Online] 18. ledna 2012. [Citace: 18. listopad 2016.] http://singlepageappbook.com/.

Wasson, Mike. 2013. ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. *msdn.microsoft.com.* [Online] Microsoft, Listopad 2013. [Citace: 25. listopad 2017.] https://msdn.microsoft.com/en-us/magazine/dn463786.aspx.

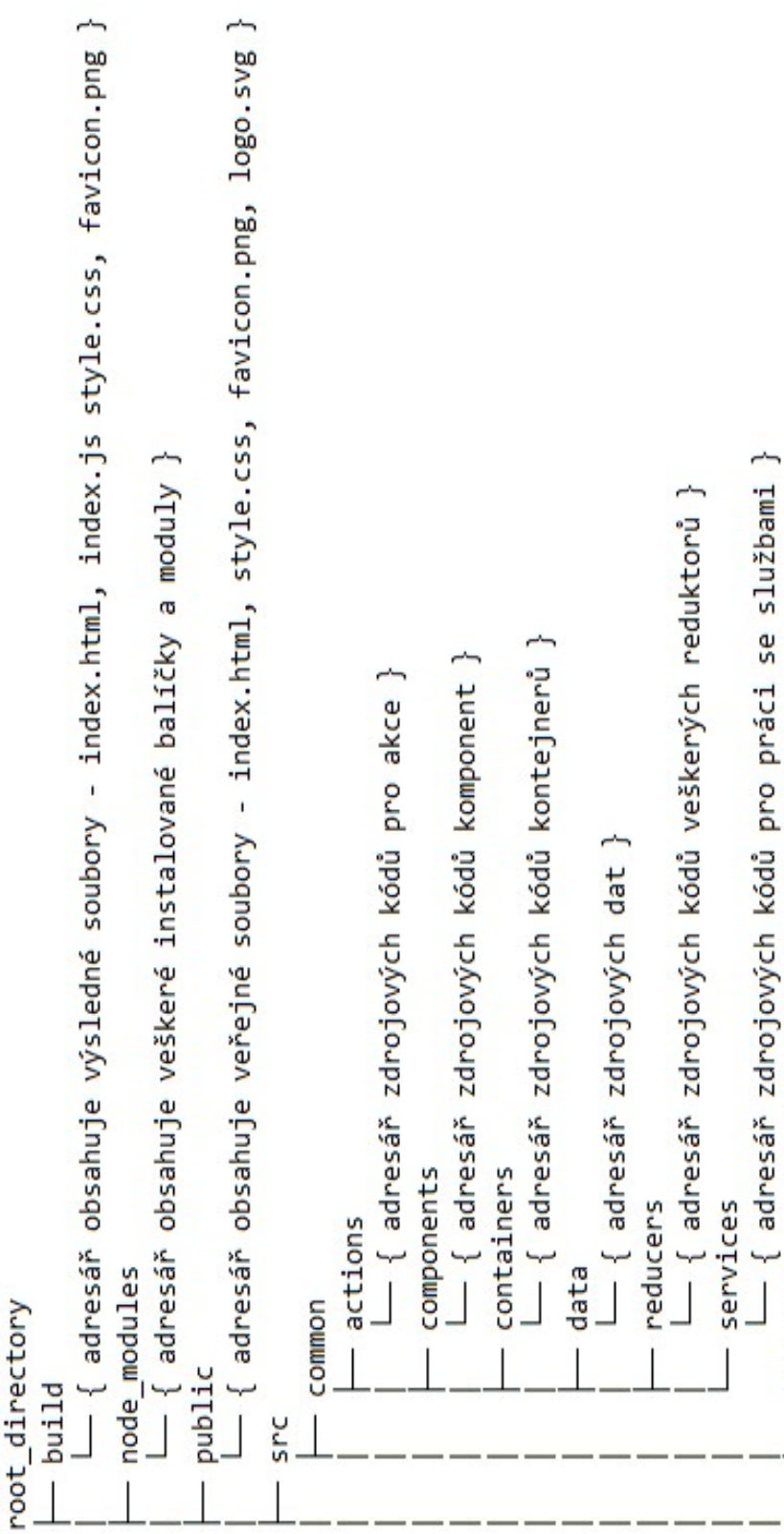
Zakas, Nicholas Z. 2009. *JavaScript pro webové vývojáře (programujeme profesionálně).* 1. vydání. Brno : COMPUTER PRESS, 2009. str. 832. 978-80-251-2509-0.

Příloha na CD-ROM

Elektronická podoba dokumentu ve formátu PDF.

Zdrojové kódy uživatelského rozhraní.

Příloha A


root_directory
build
{ adresář obsahuje výsledné soubory - index.html, index.js style.css, favicon.png }
node_modules
{ adresář obsahuje veškeré instalované balíčky a moduly }
public
{ adresář obsahuje veřejné soubory - index.html, style.css, favicon.png, logo.svg }
src
common
actions
{ adresář zdrojových kódů pro akce }
components
{ adresář zdrojových kódů komponent }
containers
{ adresář zdrojových kódů kontejnerů }
data
{ adresář zdrojových dat }
reducers
{ adresář zdrojových kódů veškerých reduktorů }
services
{ adresář zdrojových kódů pro práci se službami }
index.js - { kořenový soubor typu JavaScript }
package.json - { soubor pro definici závislostí pro NPM }

Příloha B

Struktura dat ukládaných v databázi přistroje.