FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Prototyp interaktivního modelu oka pro vizualizaci nitrooční čočky - Front-end
Student:	Tomáš Vondra
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Projekt Model oka je aplikace vyvíjena ve spolupráci s Fakultou biomedicínského inženýrství. Aplikace se zaměřuje na úpravu modelu oka vzhledem k reálným parametrům naměřených před operací kataraktu (operace šedého zákalu). Manipulace s modelem má usnadňovat očním lékařům orientaci umělé čočky v oku při této operaci.

* Analyzujte aktuální stav projektu, zaměřte se zejména na front-endovou část aplikace a její komunikaci s Rest API.
* Analyzujte technologie pro tvorbu webových a desktopových uživatelských rozhraní.
* Analyzujte návrhové vzory pro zvolené technologie.
* Pomocí metod softwarového inženýrství navrhněte front-end aplikace.
* Na základě provedené analýzy proveďte implementaci prototypu.
* Hotový prototyp podrobte vhodným testům (akceptační, uživatelské, ...).

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 3. února 2023 v Praze.

Bakalářská práce

PROTOTYP INTERAKTIVNÍHO MODELU OKA PRO VIZUALIZACI NITROOČNÍ ČOČKY – FRONT-END

Tomáš Vondra

Fakulta informačních technologií
Katedra softwérového inženýrství
Vedoucí: Ing. Jiří Chludil
10. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat vedoucímu práce Ing. Jiřímu Chludilovi za vedenní a rady při psaní této práce. Dále bych chtěl poděkovat Ing. Martinu Fůsovi, Ph.D za to, že mi umožnil testování aplikace na oční klinice a všem zaměstnancům kliniky co se testování účstnili. Nakonec bych chtěl poděkovat své rodině za neustálou podporu nejen při psaní této práce, ale i po celé době studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 10. května 2023 .....

Abstrakt

Tato bakalářská práce se zabývá tvorbou front-endové části prototypu interaktivního modelu oka pro vizualizaci nitrooční čočky. Jde o webovou aplikaci pro správu pacientů a jejich vyšetření. Umožňuje tvořit, upravovat a vyhledávat pacienty. K pacientům pak zaznamenávat informace o jejich vyšetřeních.

Konkrétně se práce zabývá analýzou současného řešení, návrhem a implementací řešení nového a následným testování tohoto řešení.

Analýza sestává z rozboru technologií pro vytváření webových aplikací a souvisejících návrhových vzorů. V rozboru se zaměřuji na hodnocením javascryptových frameworků pro tvorbu front-endu. S nejlépe hodnoceným frameworkem pak dále pracuje v návrhu a implementaci.

Návrh se zabývá hlavně architekturou vhodnou pro použití s vybraným frameworkem, pokrytím funkčních požadavků a designem uživatelského rozhraní v podobě wireframů.

Z návrhu vychází implementační část, kde práce rozebírá implementační detaily a výsledky praktické části práce. Rovněž uvádí přiručku pro instalaci aplikace na vlastní zařízení.

Práce je zakončena testováním aplikace. Aplikace byla testována na oční klinice v praze, kde by i někdy v budoucnu mohla být uvedena do opravdové praxe. Práce zde uvádí přípravu testování, jeho průběh a nakonec závěrečné zhodnocení.

Klíčová slova webová aplikace, interaktivní model oka, front-end framework, React, návrhové vzory, uživatelské rozhraní, Flux, interaktivní model oka

Abstract

This bachelor thesis deals with the development of the front-end part of a prototype interactive eye model for visualization of the intraocular lens. It is a web application for managing patients and their examinations. It allows to create, edit and search patients. Information about their examinations is then recorded for the patient.

Specifically, the thesis deals with the analysis of the current solution, the design and implementation of a new solution and the subsequent testing of the new solution.

The analysis consists of an exploration of technologies for creating web applications and associated design patterns. In the analysis, I focus on the evaluation of javascrypt frameworks for front-end development. It then continues to work with the best evaluated framework in the design and implementation.

The design mainly deals with the architecture suitable for use with the selected framework, coverage of functional requirements, and user interface design in the form of wireframes.

From the design comes the implementation part, where the thesis discusses the implementation details and results of the practical part of the work. It also presents a tutorial for installing the application on a personal device.

The thesis concludes by testing the application. The application was tested in an eye clinic in Prague, where it could be put into real practice sometime in the future. The thesis presents here the preparation of the testing, its course and in the end the final review.

Keywords web application, interactive eye model, front-end framework, React, design patterns, user interface, Flux, interactive eye model

Seznam zkratek

API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Development
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DRY	Don't repeat yourself
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
SPA	Single page application

Úvod

Oční chirurg musí při operaci šedého zákalu spoléhat především na své ruce a na svůj vlastní zrak. S jeho prací mu pomáhají, ale i další lidé jako sestry nebo medicínský technik. Medicínský technik má při operaci za úkol sledovat přístroje a na vyžádání předat chirurgovi údaje o pozici nitrooční čočky v pacientově oku. Pak již záleží na chirurgovi, aby si za pomocí zjištěných informací udělal představu o tom, jak to v oku vypadá a jak má na základě toho s čočkou hýbat. Interkativní model oka tuto komunikaci výrazně zjednoduší a zároveň zlepší úroveň technikem předávaných informací.

Další motivací pro vytvoření takové aplikace je, že medicínský technik sbírá při předoperačním vyšetření o pacientovi velké množství dat, jako jsou například parametry jeho oka. V současnosti neexistuje přístroj, který by zvládl změřit vše potřebné najednou a tak je technik nucen oběhnout s pacientem více přístrojů, data si zaznamenávat na papír a později přepisovat. S webovým rozhraním této aplikace, by technik mohl jednoduše vzít tablet a rovnou data z přístrojů zadávat do formuláře, tím je rovnou přiřazovat k danému pacientovi a ukládat do databáze.

Projekt vznikl ve spolupráci s Fakultou Biomedicínského Inženýrství na ČVUT. Na vytváření této aplikace jsem se podílel v rámci předmětu týmový projekt a je tedy už nějaký čas ve vývoji. Výstupem týmového projektu byl Blender plugin, který upravuje model oka podle zadaných parametrů. Na to následně navázala práce, jejíž výsledkem byl interaktivní mockup webové aplikace. Mým úkolem je vytvořit z mockupu prototyp webové aplikace, která bude spravovat profily pacientů a jejich vyšetření pomocí přívětivého a jednoduchého uživatelského rozhraní.

Nejdříve provedu analýzu současného stavu frontendu aplikace a navrhnu změny, které by se měly implementovat. Poté se zaměřím na dostupné technologie a knihovny používané k vytváření takovýchto webových aplikací. Na základě hodnocení z nich pak vyberu ideální pro moje potřeby.

K vybraným technologiím se také vážou návrhové vzory, které mi říkají jak řešit běžné problémy při vývoji a pomáhají udržovat kód čitelný a zajišťovat jeho kvalitu. Zjistím které návrhové vzory jsou vhodné pro mojí problematiku a uplatním je v implementační části mé práce.

Další částí práce bude navrhnout aplikaci, tak aby splňovala mé poznatky z provedené analýzy. Hlavním uživatelem aplikace má být lékař, a tak při návrhu budu klást důraz hlavně na jednoduché rozhraní a snadné používání. Většina současných doktorů je totiž ještě ze staré školy a proto je důležité, aby se s aplikací dalo naučit intuitivně pracovat.

Nakonec podrobým aplikaci uživatelským testům.

Cíle práce

Hlavním Cílem práce je implementovat prototyp front-endové části aplikace interaktivního modelu oka pro vizualizaci nitrooční čočky.

Analyzuji současný stav aplikace a technologie použité pro její vývoj. Konkrétně analyzuji kvalitu kódu a využití návrhových vzorů. U technologií se budu soustředit na analýzu použitého framewowrku. Poté analyzuji nové technologie potřebné pro vlastní implementaci a popíšu zásady pro tvorbu uživatelského rozhraní. Nakonec vypíšu funkční a nefunkční požadavky aplikace.

Na základě této analýzy zhotovím návrh aplikace. Popíšu architekturu, kterou budu používat, vytvořím doménový model a diagramy používání aplikace, pomocí kterých ověřím pokrytí funkčních požadavků. Nakonec navrhnu vzhled uživatelského rozhraní pomocí wireframů.

V neposlední řadě provedu implementaci, která bude vycházet z návrhu a analýzy. V rámci implementace vytvořím programátorskou, instalační a uživatelskou příručku. Nakonec podrobím implementaci uživatelskému testování, pro které vytvořím testovací scénáře a výsledky tohoto testování zhodnotím.

Kapitola 1

Analýza

V této kapitole zhodnotím současný stav projektu a analyzuji výstupy předmětů BI-SP.1 a BI-SP.2, v rámci kterých byla tato aplikace dosud vyvíjena. Konkrétně se zaměřím na použité technologie a kvalitu kódu.

Nakonec ukážu a popíšu všechny nasbírané funkční a nefunkční požadavky a vyberu z nich ty, které se týkají mojí bakalářské práce.

1.1 Týmový projekt

Jak již jsem zmiňoval v úvodu, tato aplikace byla vyvíjena v rámci předmětů týmový projekt 1 a týmový projekt 2. Součástí tohoto týmu jsem byl já, Andrej Ohrablo, Daniel Koreň, Dominik Žůrek a Marek Koubek.

V rámci týmového projektu jsme řešili analýzu funkčních a nefunkčních požadavků se zadavatelem projektu Ing. Martinem Fůsem, Ph.D, následně návrh a nakonec jsme vytvořili i velice jednoduchý mockup.

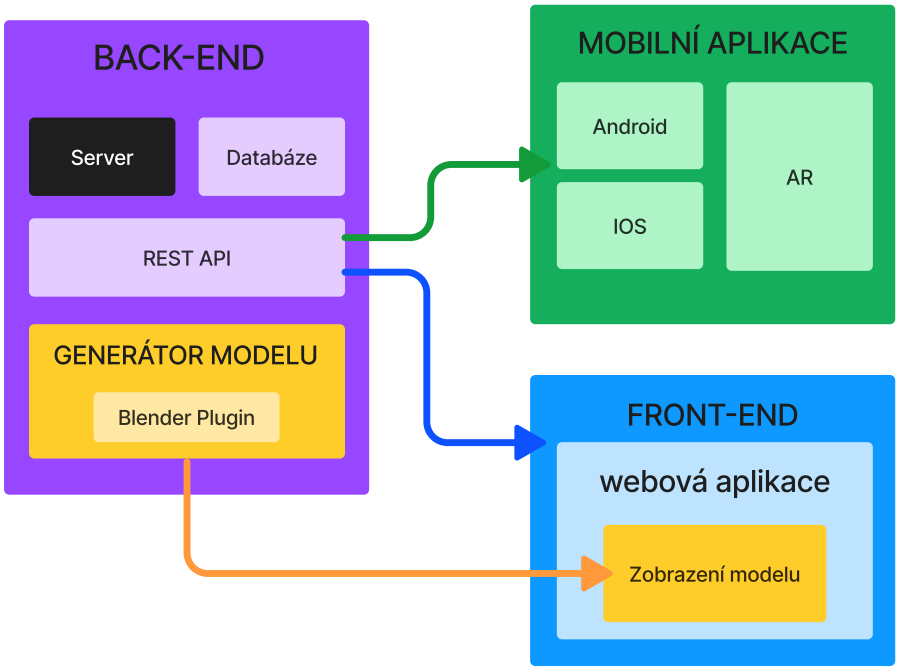
Čtyři z nás se pak dále věnují tématu ve své bakalářské práci. Témata jsme si rozdělili podle funkčních celků aplikace: Já – front-end, Daniel Koreň – back-end, Dominik Žůrek – mobilní aplikace, Marek Koubek – generování 3D modelu. Rozdělení je znázorněno na diagramu 1.1.

1.2 Použité technologie

Jako první věc analyzuji použité technologie. Protože je aplikace v ranném stádiu vývoje, mohu použití některých technologií přehodnotit a vyměnit za jiné vhodnější pro potřeby projektu.

Platforma

V současném stavu je aplikace koncipovaná jako webová stránka. Vzhledem k tomu, že aplikace má být dostupná jak na Windows tak i na android a IOS, je webová stránka dobrou volbou. Pokud existuje na dané platformě webový prohlížeč, pak tam lze aplikaci spustit. Nemusím tedy tvořit aplikaci speciálně pro každou platformu.



Obrázek 1.1 Diagram rozdělení práce

Framework

Uživatelské rozhraní webové aplikace bylo implementováno pomocí React.js frameworku. React je framework vyvájený společností Meta a jde o jeden z nejpopulárnějších frameworků pro tvorbu webových stránek již od roku 2016. [1, 2]

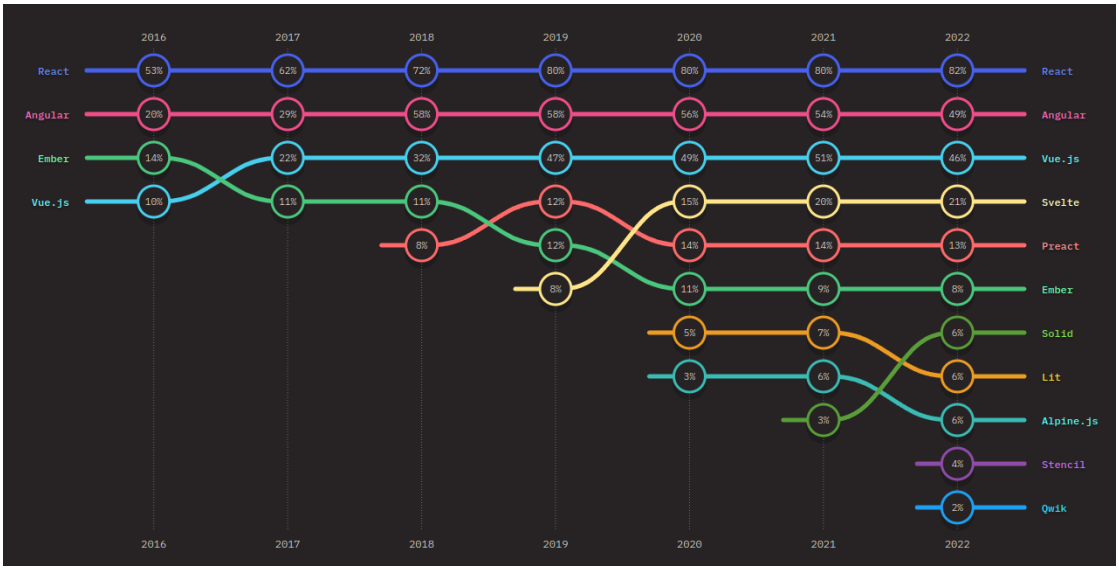
React se používá hlavně pro vývoj single page aplikací (SPA). SPA mají výhodu v tom, že na straně klienta se načte pouze jedna html stránka a veškeré změny co uživatel vidí probíhají pomocí javascriptu v pozadí. SPA se tak vyhne nutnosti obnovovat stránky, jako je tomu u vícestránkových aplikací. [3]

Takových frameworků existuje celá řada a jak již jsem zmiňoval, aplikace je ve velice ranném stádiu vývoje a přechod mezi frameworky by nebyl obtížný. Vyplatí se tedy udělat průzkum alternativ.

Alternativní frameworky

Nejdříve je nutné, abych vybral vhodné alternativy schopné Reactu konkurovat. React využívá programovací jazyk JavaScript. Ten je v dnešní době jeden z nejpopulárnějších jazyků pro vytváření webových aplikací, proto se v průzkumu zaměřím na další frameworky napsané v tomto jazyce.

Tyto další frameworky jsem vybral na základě průzkumu jejich popularity [2]. Výsledky průzkumu jsou vidět na grafu 1.2 níže. Frameworky které budu hodnotit jsou: **React**, **Angular 2**, **Vue** a **Svelte**.



Obrázek 1.2 Graf používání front-end frameworků [2]

Hodnotící kritéria

Abych mohl frameworky zhodnotit a mezi sebou porovnat, musím stanovit hodnotící kritéria. U každého kritéria vysvětlím, proč je důležité a za co konkrétně framewrok může získat body v dané kategorii. Nakonec vytvořím přehlednou tabulku s výsledky.

Jednoduchost

Framework by měl být nástrojem pro ulehčení programátorovi práce při vývoji. Proto je toto kritérium velmi důležité. Zároveň je ale velice těžké toto kritérium ohodnotit a porovnat jednoduchost frameworků mezi sebou, protože neexisute žádný měřitelný faktor jednoduchosti. Jde navíc o velmi subjektivní zálžitost, kterou každý člověk může vnímat trochu jinak. Proto jednoduchost frameworků budu hodnotit následovně.

S každým frameworkem vytvořím „Hellow World“ aplikaci, která bude obsahovat jedno tlačítko. Po zmáčknutí tlačítka se objeví na další stránce (tzn. Změna v url) nápis „Hello World“. Frameworky potom seřadím podle času, (jak dlouho mi trvalo Hello World aplikaci s daným frameworkem vytvořit) a bodově je ohodnotím (4 body nejrychlejší – 1 bod nejpomalejší). Časy vypíšu v hodnocení konkrétních frameworků.

Dokumentace
Základem dobrého frameworku je dobrá dokumentace. Z dlouhodobého hlediska, pokud používám nějaký framework, vždy narazím na nějaký problém. V ten moment by měla přijít do hry dokumentace, která mi pomůže můj problém vyřešit. Dokumentace by měla být rozsáhlá, ale přesto přehledná a mělo by se v ní dát snadno vyhledávat.

Pokud něco nenajdu v dokumentaci, často budu hledat odpověď někde na fórech. Proto v rámci dokumentace ohodnotím i to jak má framework aktivní komunitu. Dokumentaci budu hodnotit následovně:

- Obsahuje dokumentace praktické ukázky? – 1b
- Existuje dokumentace k API frameworku? – 1b
- Obsahuje dokumentace tutorial na jednoduchou aplikaci? – 1b
- Exitují dokumentace pro starší verze frameworku? – 2b
- Dá se v celé dokumentaci vyhledávat(nepočítá se vyhledávání na stránce)? – 1b
- Má framework aktivní komunitu na diskuzních fórech? – 1b

Aktuálnost
Pokud chci aplikaci vyvíjet dlouhodobě je dobré, aby se spolu s ní vyvíjel i framework. Proto je důležité, aby vývojáři frameworku vydávali pravidlné patche, a tím reagovali na měnící se situaci na trhu (například pokud by se vyskytla nějaká bezpečnostní ryzika). Pokud se vývojáři o framework starají, zančí to růst kvality daného frameworku do budoucna a vyplatí se učit se s ním pracovat. Aktuálnost budu hodnotit následovně:

- Dostává framework menší aktualizace a opravy aspoň jednou do měsíce? – 1b
- Dostává framework velké aktualizace, které ho posunují technologicky dopředu aspoň jednou do roka? – 1b
- Podporují nové velké aktualizace některé technologie z minulých verzí? – 1b

Rozšiřitelnost
Často se stává, že framework v základu nabízí jen některé funkcionality a pokud potřebuji něco specifického musím si nainstalovat rozšiřující balíček nebo knihovnu. Proto je důležité, abych hledáním, instalováním a sprovozňováním chybějicí funkcionality nestrávil zbytečně moc času. Další věcí je, že pokud bych byl nucen další knihovny instalovat ručně, bylo by sprovoznění CI/DI s touto aplikací zbytečně složité. Rozšiřitelnost budu hodnotit následovně:

- Jsou framework a rozšiřující knihovny součástí nějakého package manageru? – 1b
- Je jednoduché najít knihovnu s potřebnou funkcionalitou? Například v nějakém seznamu? – 1b
- Hlídá mi něco kompatibilitu verzí knihoven? – 1b

Předchozí zkušenost
Při výběru frameworku musím myslet i na to jestli už s ním mám nějaké zkušenosti. Framework může mít nejlepší vlastnosti, ale pokud jsem s ním nikdy nepracoval, je velká šance, že stejně odvedu lepší práci s frameworkem, který už znám. Moji zkušenost s frameworkem budu hodnotit následovně:

- Již jsem s frameworkem pracoval na větším projektu v minulosti? – 2b
- Již mám zkušenosti s programovacím jazykem se kterým framework pracuje? – 1b

1.2.1 React

React je knihovna pro vývoj uživatelských rozhraní za pomocí komponent napsaná v jazyce JavaScript. Je vyvíjena a udržována společností Meta. React je vhodný pro vývoj jednostránkových aplikací, mobilních aplikací a aplikací vykreslovaných na serveru. React se stará pouze o vykreslování, to znamená že je obvykle s knihovnou React potřeba použití dalších knihoven (například pro směrování). [1]

Narozdíl od ostatních hodnocených frameworků, nemá React svůj vlastní šablonovací jazyk pro vytváření html souborů a vyvíjí se čistě v javascriptu nebo typescriptu. Veškerý HTML výstup se tedy tvoří ve funkcích a třídách, což v klasickém javascriptu nevypadá moc hezky. K tomu aby bylo HTML v javascriptu pohodlné psát slouží formát JSX. Syntax JSX pak vypadá podobně jako klasické HTML. [4]

Hodnocení

Jednoduchost
Celkově mi trvalo demo vytvořit 22 minut a 41 sekund. Uděluji tři body – 3b.

Dokumentace
Dokumentaci [5, 6] jsem hodnotil následovně:

- Obsahuje dokumentace praktické ukázky? – 1b
- Existuje dokumentace k API frameworku? – 1b
- Obsahuje dokumentace tutorial na jednoduchou aplikaci? – 1b
- Exitují dokumentace pro starší verze frameworku? – 2b
- Dá se v celé dokumentaci vyhledávat(nepočítá se vyhledávání na stránce)? – 1b
- Má framework aktivní komunitu na diskuzních fórech? – 1b

Aktuálnost

- Dostává framework menší aktualizace a opravy aspoň jednou do měsíce? – 1b [7]
- Dostává framework velké aktualizace, které ho posunují technologicky dopředu aspoň jednou do roka? – 1b [7]
- Podporují nové velké aktualizace podoporu některých technologií z minulých verzí? – 0b[8]

Rozšiřitelnost
React je dostupný je dostupný pomocí npm [9].

- Jsou framework a rozšiřující knihovny součástí nějakého package manageru? – 1b
- Je jednoduché najít knihovnu s potřebnou funkcionalitou? Například v nějakém seznamu? – 0b
- Hlídá mi něco kompatibilitu verzí knihoven? – 1b

Předchodzí zkušenost

- Již jsem s frameworkem pracoval na větším projektu v minulosti? – 2b
- Již mám zkušenosti s programovacím jazykem se kterým framework pracuje? – 1b

Shrnutí

React je stejně jako ostatní frameworky zde zmíněné component-based. Komponenty se vytvářejí pomocí funkcí nebo tříd, které vrací ReactNode. Kde ReactNode může být cokoliv ve formátu JSX, nebo řetězec. [1]

Díky tomuto způsobu vytváření komponent, pro mě bylo v porovnání s ostatními frameworky, které každý představily svůj šablonovací jazyk a syntax, velmi intuitivní s komponenty pracovat. Komponenty v Reactu se totiž nijak neliší od jiných funkcí v jakémkoliv jiném programovacím jazyce.

Jedinou překážkou v psaní kódu je tedy výše zmiňovaný formát JSX, který je ale velice jednoduchý a pokud již programátor předtím znal HTML, s JSX si snadno poradí.

Největší problém při vytváření dema mi dalo routování (přepínání stránek). Narozdíl od ostatních zde prezentovaných frameworků, React sám nenabízí možnost routování, takže jsem musel sáhnout po externím řešení. Dokumentace k tomuto řešení navíc není úplně ideální. [10]

Zhodnocení na závěr: React je velice příjemný nástroj pro tvorbu uživatelského rozhraní pro programátory, kteří rádi dělají s čistým javascriptem/typescriptem a nechtějí se učit další šablonovací jazyk. Je to dobrý nástroj pro tvorbu středních až větších projektů, ale u malých projektů představuje zbytečný overhead.

Celkově jsem ohodnotil React sedmnácti body – 17b

1.2.2 Angular

Angular je framework pro tvorbu webových stránek postavený na jazyce TypeScript a HTML vyvýjený společností Google. Stejně jako React je i Angular vhodný pro tvorbu jednostránkových aplikací a je založený na principu tvorby komponent. Na rozdíl od Reactu Angular nabízí integrované knihovny, které pokrývají veškeré potřeby pro tvorbu webových aplikací. Používá se hlavně při vývojí velkých aplikací na podnikové úrovni. [11]

Hodnocení

Jednoduchost
Celkově mi trvalo demo vytvořit 44 minut. Uděluji jeden bod – 1b.

Dokumentace
Dokumentaci [12] jsem hodnotil následovně:

- Obsahuje dokumentace praktické ukázky? – 1b
- Existuje dokumentace k API frameworku? – 1b
- Obsahuje dokumentace tutorial na jednoduchou aplikaci? – 1b
- Exitují dokumentace pro starší verze frameworku? – 2b
- Dá se v celé dokumentaci vyhledávat(nepočítá se vyhledávání na stránce)? – 1b
- Má framework aktivní komunitu na diskuzních fórech? – 1b

Aktuálnost

- Dostává framework menší aktualizace a opravy aspoň jednou do měsíce? – 1b [13]
- Dostává framework velké aktualizace, které ho posunují technologicky dopředu aspoň jednou do roka? – 1b [13]
- Podporují nové velké aktualizace podoporu některých technologií z minulých verzí? – 1b [13]

Rozšiřitelnost
Angular je dostupný je dostupný pomocí npm [14].

- Jsou framework a rozšiřující knihovny součástí nějakého package manageru? – 1b
- Je jednoduché najít knihovnu s potřebnou funkcionalitou? Například v nějakém seznamu? – 0b
- Hlídá mi něco kompatibilitu verzí knihoven? – 1b

Předchodzí zkušenost

- Již jsem s frameworkem pracoval na větším projektu v minulosti? – 0b
- Již mám zkušenosti s programovacím jazykem se kterým framework pracuje? – 0b

Shrnutí

Kdybych měl práci s Angularem popsat tak si určitě nevyberu slova přímočarý a jednoduchý. Jen abych mohl začít na demu pracovat, musel jsem si přečíst kolem dvaceti stránek dokumentace. I když je demo opravdu velice jednoduché, jde jen o dvě stránky s tlačítkem a nápisem, dlouho jsem nevěděl, kde bych měl začít.

Angular má spoustu užitečných nástrojů, jak programátorovi, zjednodušit práci. Nejdříve je tu šikovné CLI, které se nainstaluje společně s knihovnou Angularu a potom všudypřítomné odkazy na dokumentaci. CLI mi pomohlo s inicializací pracovního adresáře a poté i při generování nových komponent.

Komponenta má představovat jednu část uživatelského rozhraní (například tlačítko, nebo stylizovaný seznam), která se potom dá používat na libovolných místech po celé stránce. V Angularu se komponenty skládají z několika souborů: soubor pro styly, soubor pro šablonu a soubor pro typescript. Soubor pro typescript obsahuje metadata komponenty a spojuje všechny soubory dohromady. [12]

Toto dělení na více souborů, vede k velice striktnímu rozdělení různých částí kódu, což je najednu stranu krásně přehledné pro větší projekty, ale nadruhou stranu pokud mám v každém souboru deset řádků kódu na přehlednosti to nepřidá a je opravdu otravné pořád muset mezi soubory přepínat. Nemluvě o tom pokud upravuji komponent více najednou.

Angular má opravdu kvalitně zpracovanou dokumentaci a věřím, že kdybych se celý týden věnoval studiu základních principů práce s angularem, které jsou v dokumentaci velice detailně popsány, byl bych schopen v dokumentaci najít odpověď na každou svou otázku. Protože jsem však takto připraven nebyl, narážel jsem konstantně na to, že nejsem schopen z dokumentace vyčíst řešení i těch nejzákladnějších problémů.

Obecně se tedy dá říct, že angular je velice robustní framework připravený pro vývoj velkých single page aplikací, napěchovaný vším co při vývoji budete potřebovat, ale pro tvorbu menších projektů je absolutně nevhodný.

Celkově jsem ohodnotil Angular čtrnáctí body – 14b

1.2.3 Vue

Vue je framework postavený na jazyce JavaScript, používaný pro vývoj uživatelských rozhraní. Je to nadstavba nad standartním HTML, CSS a Javascriptem a nabízí deklarativní programování s pomocí komponent. Jde o flexibilní framework pro vývoj malých i komplexních webových aplikací. [15]

Hodnocení

Jednoduchost
Celkově mi trvalo demo vytvořit 24 minut a 31 sekund. Uděluji dva body – 2b.

Dokumentace
Dokumentaci [15] jsem hodnotil následovně:

- Obsahuje dokumentace praktické ukázky? – 1b
- Existuje dokumentace k API frameworku? – 1b
- Obsahuje dokumentace tutorial na jednoduchou aplikaci? – 1b
- Exitují dokumentace pro starší verze frameworku? – 2b
- Dá se v celé dokumentaci vyhledávat (nepočítá se vyhledávání na stránce)? – 1b
- Má framework aktivní komunitu na diskuzních fórech? – 1b

Aktuálnost

- Dostává framework menší aktualizace a opravy aspoň jednou do měsíce? – 1b [16]
- Dostává framework velké aktualizace, které ho posunují technologicky dopředu aspoň jednou do roka? – 1b [16]
- Podporují nové velké aktualizace podoporu některých technologií z minulých verzí? – 1b [17]

Rozšiřitelnost

Vue je dostpné přez npm. [18]

- Jsou framework a rozšiřující knihovny součástí nějakého package manageru? – 1b
- Je jednoduché najít knihovnu s potřebnou funkcionalitou? Například v nějakém seznamu? – 0b
- Hlídá mi něco kompatibilitu verzí knihoven? – 1b

Předchodzí zkušenost

- Již jsem s frameworkem pracoval na větším projektu v minulosti? – 0b
- Již mám zkušenosti s programovacím jazykem se kterým framework pracuje? – 0b

Shrnutí

Vue framework si zakládá na flexibilitě. Dá se použít jak na malé tak i na velké projekty, nebo dokonce zakomponovat Vue do již rozdělaného projektu společně s jiným frameworkem. [15]

Vue se stejně jako Angular nainstalovala s CLI, které mi značně ulehčilo práci při vytváření základní struktury projektu. Po zapnutí serveru mě čekala vygenerovaná helloWorld stránka, která mě odkázala na dokumentaci.

Dokumentace mě provedla základními principy, což stačilo k tomu abych se v předpřipraveném projektu vyznal. Pak už jsem dokumentaci prakticky nepotřeboval, neboť knihovna dostála svému slibu a opravdu mě nechala dělat jen to co jsem potřeboval.

Vue stejně jako Angular dělí aplikaci na znovupoužitelné komponenty. Narozdíl od Angularu, je ale komponenta v základním nastavení složena pouze z jednoho souboru s příponou .vue. V souboru komponenty se pak nachází oddělené bloky kódu pro Javascript, html a css. To přispíva k přehlednosti a zároveň mám vše co potřebuji k úpravě komponenty na jednom místě.

S jistotou mohu říct, že Vue dostál svému slovu a jde v zásadě o velmi flexibilní framework, na jehož workflow a syntax se dá snadno zvyknout. Je vhodný pro začátečníky s component-based frameworky, kteří se chtějí pustit do vývoje webových aplikací. Zároveň, ale nabízí i spoustu pokročilých funkcí ve formě rozšiřujících knihoven, se kterými se můžete pustit do velkých projektů.

Celkově jsem ohodnotil Vue čtrnácti body – 14b

1.2.4 Svelte

Svelte je stejně jako ostatní výše zmíněné frameworky, používaný pro vývoj jednostránkových webových aplikací s pomocí komponent založený na jazyce JavaScript, HTML a CSS. Narozdíl od ostatních ale většinu práce, kterou ostatní frameworky dělají až při vykreslování v prohlížeči, provádí v kompilačním kroku při stavění aplikace. [19]

Hodnocení

Jednoduchost
Celkově mi trvalo demo vytvořit 15 minut a 48 sekund. Uděluji čtyři body – 4b.

Dokumentace
Dokumentaci[20] jsem hodnotil Následovně:

- Obsahuje dokumentace praktické ukázky? – 1b
- Existuje dokumentace k API frameworku? – 0b
- Obsahuje dokumentace tutorial na jednoduchou aplikaci? – 1b
- Exitují dokumentace pro starší verze frameworku? – 0b
- Dá se v celé dokumentaci vyhledávat(nepočítá se vyhledávání na stránce)? – 0b
- Má framework aktivní komunitu na diskuzních fórech? (V porovnání s ostatními frameworky) – 0b

Aktuálnost

- Dostává framework menší aktualizace a opravy aspoň jednou do měsíce? – 1b [21]
- Dostává framework velké aktualizace, které ho posunují technologicky dopředu aspoň jednou do roka? – 1b [21]
- Podporují nové velké aktualizace podoporu některých technologií z minulých verzí? – 0b

Rozšiřitelnost
Svelte je dostupný přez npm. [22]

- Jsou framework a rozšiřující knihovny součástí nějakého package manageru? – 1b
- Je jednoduché najít knihovnu s potřebnou funkcionalitou? Například v nějakém seznamu? – 1b
- Hlídá mi něco kompatibilitu verzí knihoven? – 1b

Předchodzí zkušenost

- Již jsem s frameworkem pracoval na větším projektu v minulosti? – 0b
- Již mám zkušenosti s programovacím jazykem se kterým framework pracuje? – 0b

Shrnutí

Svelte je jako ostatní frameworky component-base a zakládá na jednoduchosti a rychlosti. V obou těchto kategoriích Svelte exceluje. Unikátní vlastností Svelte je, že kompiluje kód do obyčejného javascriptu, který neobsahuje žádné stopy po použitém frameworku. Výsledná velikost balíčku dat, co uživatel pro zobrazení stránky musí stahovat je tedy velice malá. [19]

Samotný framework je také velice úsporný co se struktury projektu týče. Pro pochopení základů mi stačilo prohlédnout si jednu stránku v dokumentaci. Aby framework správně fungoval, stačí aby se pod hlavní složkou s kódem nacházel soubor se jménem app.html a složka s názvem routes. [20]

Princip frameworku je velmi jednoduchý. Pokud chci vytvořit novou stránku například s adresou home/detail stačí vytvořit odpovídající složkovou strukturu pod složkou routes. Ve složce detail se pak musí nacházet soubor s názvem +page.svelte. Jednotlivé komponenty jsou tak snadno dohledatelné. Pokud bych, ale chtěl mít v url nějaký parametr, musím i pro něj vytvořit prázdnou složku s názvem „[slug]“. To může při velkých projektech vést k spoustě zbytečných prázdných složek. [20]

Samotný soubor +page.svelte je stejně jako soubory ve frameworku Vue, rozdělený na javascriptovou část, část s html a část s css styly, hezky úsporně a přehledně. [20]

Svelte je velice jednoduchý framework, se kterým se dá pohodlně pracovat na malých až středních projektech. Na větších projektech bych Svelte používal pokud váš největší zájem není udržitelnost kódu, ale rychlost se kterou se dostane stránka k uživateli (ať už po stránce programování, tak i načítání stránky).

Celkově jsem ohodnotil Svelte jedenácti body – 11b

1.2.5 Shrnutí hodnocení

Všechny čtyři frameworky, jsou si velice podobné a pro vývoj této aplikace by vyhovoval kterýkoliv z nich. Z mého šetření však vyplívá, že pro mě nejideálnější bude pokračovat ve vývoji aplikace s frameworkem React. Celkové rozdělení bodů je vidět v tabulce 1.1.

Tabulka 1.1 Shrnutí hodnocení frameworků

	React	**Angular**	**Vue**	**Svelte**
Jednoduchost	3	1	2	4
Dokumentace	7	7	7	2
Aktuálnost	2	3	3	2
Rozšiřitelnost	2	3	2	3
Moje vlastní zkušenost	3	0	0	0
Celkem	17	14	14	11

1.3 Analýza kódu

V této části analyzuji kód prototypu vytvořeného v rámci týmového projektu. Zaměřím se na zvyklosti programování v daném frameworku a jazyce a na celkovou strukturu projektu.

Zhodnocení

Hned ze začátku musím pochválit využití Linteru pro typescript, který provádí statickou kontrolu kódu za účelem nalezení stylistický chyb a pochybných konstrukcí [23]. O to horší, ale musím udělit výtku za ignorování námitek tohoto nástroje. Kód má velice daleko od principů jako je DRY a je například plný zbytečných importů.

Kromě toho jsem narazil i na několik komponent, které až na pár řádků měly stejný kód, což určitě není ideální stav při možnosti dané komponenty poskládat z menších znovupoužitelných komponent.

Hlavní problém současného prototypu je, že data a byznisová logika jsou definovány přímo v komponentách. Stejně tak i volání na back-end se znovu konstruuje pokaždé přímo v komponentě, kde jsou data potřeba (komunikaci s backendem se pak konkrétněji věnuji v části 1.3).

Aby byla aplikace dobře rozšiřitelná a snadno se v ní hledaly chyby, bude nutné abych jednotlivé části dat, logiky a prezentace rozdělil do samostatných částí podle nějaké architektury. Architektuře se více budu věnovat později v návrhu 2.1.

Poslední moje výtka je mířená proti struktuře adresářů. Nejhůře vypadá podadresář s .css soubory, jehož struktura se nedrží žádeného jednotného formátu a některé css soubory jsou rozházné i mimo tuto složku. Dále i statické soubory jako například obrázky, nejsou všechny na jednom místě. Jinak adresářová struktura více či méně následuje praktiky pro práci s React frameworkem.

Adresářová struktura

Adresářovou strukturou je důležité se zabývat ve chvíli kdy vím, že velikost mého projektu bude růst. Dobrá adresářová struktura mi pomůže hledat související soubory a celkově organizovat mojí práci. Špatná adresářová struktura může vést k frustrující práci na projektu, kde neustále musím hledat ve změti složek soubor, který zrovna potřebuji.

Pro práci s Reactem exituje několik standardů adresářových struktur v závislosti na velikosti projektu. Já si vybral strukturu pro středně velký projekt. Tato struktura vychází z článku [24].

Hlavní složkou této adresářové struktury je složka s názvem pages. Ta by měla obsahovat složku za každou stránku aplikace. V každé této složce by pak měl být hlavní soubor představující stránku (nejčastěji s názvem index) a všechny další soubory, které obsahují kód úzce související s touto stránkou.

Další důležitou složkou je složka components. Ta obsahuje vschny komponenty aplikace, které se používají na více místech.

Předposlední složkou této adresářové struktury je složka hooks, která obsahuje hooky používané na více místech aplikace. Poslední složkou je složka assests, která slouží pro ukládání statických souborů jako jsou obrázky.

Já jsem přidal ještě složku context, která slouží pro soubory, které pracují s daty. Například by tam patřil soubor, kde jsou definovaná volání na back-end.

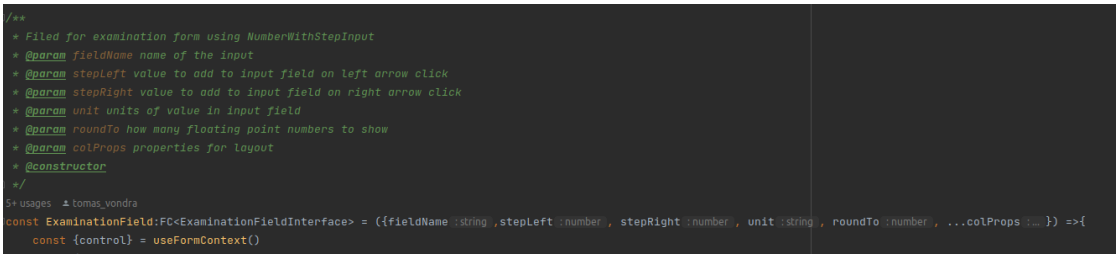
Dokumentace

Dokumentace kódu je další velká slabina projektu, protože kód parkticky žádnou dokumentaci neobsahuje. Rád bych v rámci této práce doplnil dokumentaci ve formě komentářů kódu. Pro javascript a typescript existuje formát komentářů, který umožňuje označit parametry, návratové hodnoty a mnoho dalších podobných atributů.[25]

Příklad tohot formátu komentářů je vidět na obrázku 1.3. Pomocí těchto komentářu pak lze například vgenerovat HTML dokumentaci.

Komunikace s back-endem

Webové aplikace komunikují s back-endem pomocí takzvaných HTTP dotazů. Tento dotaz se skládá ze tří částí: požadavek, hlavička a tělo. V požadavku jsou obsaženy informace o jeho druhu, cestě k požadovanému zdroji na back-endu ve formátu URL a verzi HTTP. Požadavek je následován hlavičkami. [26]



Obrázek 1.3 Příklad JSDoc komentářů v Javascriptu

Hlavičky poskytují další informace o požadavku, jako kdo je žadatel a v jakém formátu chce komunikovat. Poslední část dotazu je jeho tělo. [26]

V těle dotazu se nacházejí data, která chci na back-end poslat (například pokud chci něco vložit do databáze). Na základě dotazů pak back-end posílá žadateli odpověď také ve formátu HTTP. [26]

Existuje spoustu knihoven, které s HTTP dotazy pracují. Knihovna, se kterou pracuje prototyp se jmenuje Axios. Axios je jednoduchá knihovna, která umí pracovat s asynchroními HTTP dotazy. [27]

Místo čekání na odpověď, což by mohlo v některých případech vést k zamrznutí celé aplikace, vrátí Axios slib. Slib má tří stavy do kterých se může dostat: úspěch, chyba, konec. K těmto stavů můžeme připojit volání jakýchkoliv dalších funkcí. Například funkci, která zpracuje doručená data pokud se HTTP dotaz podařil (slib je ve stavu úspěch). Nebo funkci která upozorní uživatele na neúspěch dotazu (slib je ve stavu chyba). Ke koncovému stavu se většinou připojují funkce, které mají po dotazu „uklidit“. [28]

V prototypu se zatím vyskytuje HTTP dotazů jen pár. Jejich volání, ale probíhá pokaždé přímo v komponentách. Jak bude aplikace růst, může se stát že ve více různých komponentách bude úplně stejné volání. Pokud by se pak například změnila struktura zdrojů na back-endu, musel bych toto volání opravit jednotlivě ve všech komponentách. To rozhodně není v souladu s udržitelností a zavání velkým technickým dluhem. Proto přenesu definice veškerých volání na jedno místo. Tím docílím znovupoužitelnosti a kód bude lehce rozšiřitelný a opravitelný.

Návrhové vzory

K tvorbě jakéhokoliv programu, ať za pomocí nějakého frameworku nebo bez něj, patří návrhové vzory.

Návrhové vzory jsou taková kuchařka, která mi říká jak řešit běžné problémy, na které při programování narazím. Spousta návrhových vzorů se nevztahuje jen na určitý framework ani na konkrétní programovací jazyk, ale dají se aplikovat obecně nezávisle na platformě. Já se, ale zaměřím na ty, které jsou přímo určené pro vývoj v mnou zvoleném frameworku. [29, 30]

Návrhové vzory byly při tvorbě prototypu používány velice sporadicky a kód tak rozhodně nepoužívá nejlepší praktiky spojené s prací s Reactem. Obzvláště špatné využití principů Reactu je vidět na komponentách. Například místo jejich kompozice, nebo úpravě stávající komponenty byla prostě vytvořena nová a velká část kódu se zkopírovala.

Takovýmto nepěkným manévrům bych se chtěl vyvarovat a dodržování návrhových vzorů je jedna z věci, která mi v tom pomůže.

Analýzou návrhových vzorů a jejich následném použití v praktické částí této práce, dosáhnu využití všech silných stránek frameworku. Návrhové vzory mě také pomůžou vyhnout se problémům, které by mohly nastat až později ve vývoji, které bych pak musel složitě opravovat předěláváním velkých částí aplikace.

Obecně mi tedy návrhové vzory pomohou psát čistčí a efektivnější kód, vyplatí se tedy jejich studiu věnovat.

Návrhové vzory komponentů
Hlavním stavebním kamenem frameworku React jsou komponenty. Dobrý design těchto komponent je kritický pro správné fungování celé aplikace a pomůže mi organizovat kód tak, aby byl maximálně znovupoužitelný.

The provider pattern
Komponenty se v Reactu dají parametrizovat. Parametry které dostane komponenta na vstupu, pak mění její stav a tím i její výstup. Pokud však mám nějakou rodičovskou komponentu a chci data z ní poslat do nějaké hluboko zanořené komponenty, musel bych data poslat pomocí parametrů všech jejích rodičovských komponent až k ní.

Řešení na tento problém dává právě návrhový vzor provider. Zde figuruje nový druh komponenty takzvaný provider, který pomocí kontextu poskytuje data všem jeho vnořeným komponentám, nezávisle na tom jak hluboko jsou v hierarchii pod ním. Díky tomu se vyhnu nutnosti posílání parametrů přez komponenty, se kterými posílaná data nesouvisí. [29]

Controlled Components
Kontrolovaná komponenta je komponenta, jejíž stav je zcela určen skrz její parametry. O změnách, jako třeba onChange u textových polích, informuje svojí rodičovskou komponentu, která jí kontroluje. Rodičovská komponenta pak na základě těchto informací aktualizuje stav kontrolované komponenty. Díky tomuto přístupu se dá například snadno validovat uživatelský vstup v reálném čase. Tento návrhový vzor přichází vhod hlavně u vytváření formulářů, kterých je aplikace plná. [29, 30]

Compound components
Jde asi o nejběžnější návrhový vzor, který řeší komunikaci mezi rodičovskými a vnořenými komponentami. Tyto komponenty, které spolu úzce souvisí (například <select> a <option>), spolu sdílí svůj stav a navzájem se ovlivňují. Chování je podobné jako v případě kontrolovaných komponent, zde ale není jedna komponenta hlavní a druhá kontrolovaná. Jak rodičovská tak i vnořená zastávají obě role. [29, 30]

State reducer pattern
State reducer pattern je návrhový vzor, který pomáhá aktualizovat stav komponenty na základě nějaké akce. Když uživatel vyvolá akci, místo toho aby změnu stavu řešila přímo komponenta, zavolá reducer a poskytne mu informaci o provedené akci. Uvnitř reduceru je pak definovaná logika, která rozhodne jak stav změnit na základě výsledku provedené akce (akce mohou být například login, logout a podobně). [29]

Tento vzor pomáhá koncentrovat logiku komponenty na jedno místo a navíc jí využít i v dalších komponentách.

The Hooks pattern
V Reactu existují dva způsoby jak vyrábět komponenty, pomocí tříd a pomocí funkcí. Programování komponent pomocí tříd často vyústí v jednu obří komponentu rozdělenou na spoustu menších ve snaze rozdělit její logickou část a zobrazovací. Zároveň se komponenty vyrobené pomocí tříd špatně kompozicují. Funkcionální komponenty tyto problémy nemají a představují tedy lepší volbu při tvorbě komponent.

Hooky jsou funkce, které se dají volat jen v komponentách a díky kterým se vůbec vyplatí funkcionální komponenty tvořit. Díky nim můžu sledovat vnitřní stav komponenty a dávají mi přístup k jejímu životnímu cyklu (nasazení, změna, smazání). Zároveň React umožňuje vytvářet hooky vlastní. [29, 30]

1.4 Analýza uživatelského rozhraní

Uživatelksé rozhraní je jednou z nejdúležitějších částí každé aplikace. Musí být co nejjednoduší a hlavně co nejvíce intuitivní. Pokud bude uživatel muset každou funkcionalitu složitě hledat, nebude chtít aplikaci používat.

Další věcí na kterou je třeba si dávat pozor, je konzistentnost a zvyklosti v uživatelských rozhraní. Uživatelé předpokládají chování prvků na základě předchozích zkušeností. To znamená, že není dobré příliš měnit již zažité ikonky, které se pro dané akce běžně používají (například hamburger pro expanzi navigace).

Při tvorbě UI se budu řídit Nielsenovými kritérii [31], které tady stručně popíšu.

Nielsenova kritéria

1. **Viditelnost stavu systému**
   Systém by měl vždy uživatele informovat o tom co dělá v co nejmenším časovém rozmezí.

2. **Komunikace systému by měla odpovídat reálnému světu**
   Design by měl s uživatelem komunikovat jeho řečí. Měl by používat slova a koncepty uživateli známé. Stejně tak zobrazovat informace na přirozených místech a v logickém pořadí.

3. **Uživatelská volnost**
   Uživatelé často provedou nějakou akci omylem, proto je potřeba vždy jasně označit způsob jak se z dané akce vrátit zpět do původního stavu, například tlačítkem zpět.

4. **Konzistence a standarty**
   O tomto bodu jsem mluvil již v úvodu. Uživatelé jsou zvyklí na nějaké označení a neměli bysme ho tedy měnit. Zároveň musí být veškeré značení napříč aplikací konzistentní.

5. **Prevence chyb**
   Systém by se měl snažit co nejvíce vyvarovat chybám. Například pomocí „jste si jistí, že chcete danou akci provést“ vyskakovacím oknům.

6. **Rozpoznání před vzpomínáním**
   Veškeré akce a možnosti by měli být jasně viditelné, radši než nutit uživatele si je pamatovat. Design by neměl nutit uživatele pamatovat si informaci z jiné části uživatelského rozhraní v části další.

7. **Flexibilita a efektivnost**
   Systém by měl nabízet i pokročilé možnosti, které umožní pokročilým uživatelům ovládat aplikaci rychleji a efektivněji.

8. **Estetika a minimalistický design**
   Rozhraní by nemělo obsahovat irelevantní informace, nebo informace, které jsou málokdy potřeba.

9. **Pomoc uživateli rozpoznat, diagnostikovat a vzpamatovat se z chyb**
   Chybové hlášky by měly být v prostém jazyce (nepoužívat chybové kódy), měly by popisovat problém co nejpřesněji a ideálně navrhnout řešení.

10. **Pomoc a dokumentace**
    V ideálním případě by němela být další dokumentace potřeba. Někdy je ale nutné poskytnout uživateli manuál, který mu některé principy vysvětlí.

Technologie

K vytváření uživatelského rozhraní budu používat knihovny, které mi ušetří práci s programováním responzivních rozvržení stránek a jednotlivých uživatelských prvků. Jsou to knihovny Material UI a Bootstrap.

Material UI

Material UI je open-source knihovna komponent pro React založená na principech Google Material Design. Tato knihovna nabízí předem nadesignované uživatelské prvky jako tlačítka, nebo třeba pole. Výhoda použití takovéto knihovny je, že mi umožní využívat již uživatelsky otestované prvky s hotovými animacemi a já se můžu soustředit převážně na user experience. [32]

I kdyby se mi předpřipravený design nelíbil, nemusím prvky designovat od nuly. Stačí lehce upravit základní design a využít už připravené animace a přechody.

Bootstrap

Bootstrap je CSS, javascriptová knihovna obsahující nástroje pro tvorbu uživatelských rozhraní. Knihovna nabízí nepřeberný počet předpřipravených stylů a šablon, které se dají aplikovat na HTML. [33]

Já budu používat Bootstrap především na rozvržení obsahu na stránce. Využívat k tomu budu grid systém.

Hlavní výhodou tohoto systému je, že při správnému využití, jsou stránky automaticky responzivní. To znamená, že při změně velikosti stránky, bude obsah na ní stále dobře čitelný a použitelný.

Bootstrap toho dosahuje tím, že dělí obsah na stránce do řádek a sloupců. Při zmenšení stránky se pak sloupce, které se vedle sebe už na řádce nevejdou, přeskupí pod sebe.

1.5 Mockovaný back-end

Né vždycky se podaří vyvíjet back-end a front-end stejným tempem. Proto je někdy potřeba při vývoji front-endu back-end mockovat. Mockovaný back-end se chová z pohledu front-endu stejně jako normální. Data která se na front-end posílají jsou, ale statická nebo náhodná.

V úvodní sekci jsem zmiňoval, že souběžně s mojí prací běží i vývoj back-endu pro tuto aplikaci. Bohužel k tomu nakonec nedošlo a beck-end nebude v provozu. Musím si tedy vyrobit mockovaný backend.

Na mockování budu využívat aplikaci Mockoon. Mockoon je open-source program pro vytváření mockovaných API. Mockoon umožňuje vytvářet rozhraní pro CRUD oprece a posílat HTTP odpovědi. Dále dovoluje posílat statická data a s pomocí knihovni faker.js generovat data náhodná. Navíc se dají použít i hotové API, které můžu modifikovat a poskládat jak potřebuji. [34]

1.6 Funkční a nefunkční požadavky

Po seznámení se současným stavem projektu je čas analyzovat požadavky zadavatele projektu. Zde analyzuji požadavky na všechny části systému dohromady a poté z nich vyberu ty, které se přímo týkají mého zadání.

Pro názornost označím funkční požadavky FX a nefunkční požadavky NX, seřadím podle důležitosti od nejdůležitějšího po nejméně důležitý (zvlášť funkční a zvlášť nefunkční) a rozdělím je do skupin *nutné*, *důležité* a *zlepšující*.

Nutné požadavky jsou ty, bez kterých by aplikace nemohla správně fungovat, nebo by bez nich neměla smysl.

Do důležitých požadavků spadají ty, které bych označil za podpůrné, přesto potřebné pro správný běh aplikace v reálném provozu.

Zlepšující požadavky jsou nejméně důležité, budou implementovány s nejmenší prioritou a jde spíš o nepatrné zlepšováky, nebo naopak věci, které by sami obsáhly celé téma bakalářské práce, ale nejde o hlavní funkčnost aplikace.

1.6.1 Nutné funkční požadavky

F1 – Renderování oka
Zde leží celé srdce projektu, je to tedy nejhlavnější funkční požadavek. Aplikace by měla být schopná na základě zadaných parametrů vygenerovat 3D model oka i s umělou čočkou. Tyto parametry jsou znázorněny na obrázcích 1.4.
Paramtery oka:

- cylindrické zakřivení přední plochy rohovky (r1)
- průměr rohovky ve frontální rovině (WtW)
- tloušťka v centru rohovky (CCT)
- axiální délka oka (AL)
- hloubka přední komory (ACD)
- aproximace velikosti čočkového pouzdra (aproximace z parametru WtW)
- průměr zornice (PD)

Parametry čočky:

- model a síla umělé čočky a model korekčního cylindru
- axiální poloha uložení čočky a pouzdra
- náklon čočky vůči horizontální a vertikální rovině
- decentrace čočky vůči středu WtW
- úhlová poloha čočky dle centračních bodů

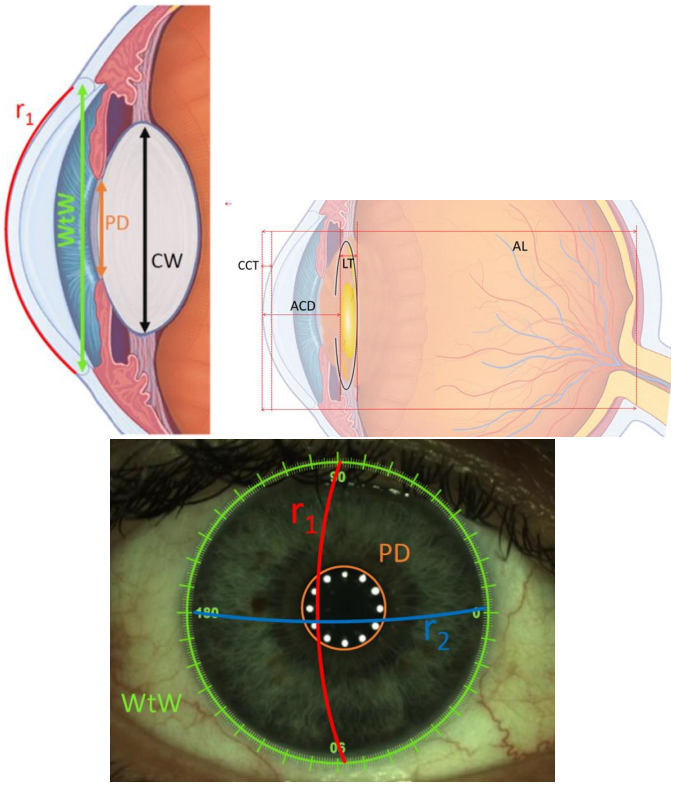
F2 – Manipulace s vygenerovaným modelem
Další velice klíčový požadavek. Poté co se model vygeneruje, by s ním mělo být možné hýbat. Tedy přesněji hýbat s pohledem na model a to jak po osách X, Y, Z tak ho i rotovat, přibližovat nebo oddalovat.

F3 – Ukládání měření
V rámci aplikace bude možnost vytvořit u pacientova profilu nové měření. Data potřebné k vytvoření nového měření obsahují (* – povinný údaj):

- Parametry oka a čočky specifikované ve funkčním požadavku F1.
- Strana (levé/pravé oko) *
- Doktor provádějící měření (přihlášený doktor) *
- Datum vytvoření / datum úpravy (automaticky) *

Vygenerovat model půjde jen z měření se všemi vyplněnými parametry.

F4 – Úprava a smazání měření
Po vytvoření měření ho bude možné upravit a smazat.



Obrázek 1.4 Parametry oka. Převzato z [35].

F5 – Ukládání pacientů
V rámci aplikace bude možnost vytvořit pacientův profil. Pacientův profil bude obsahovat tyto informace (* – povinný údaj):

- Jméno a příjmení pacienta *
- Datum narození
- Rondé číslo *
- Kontaktní údaje: e-mail, adresa, telefoní číslo

F6 – Úprava a smazání pacienta
Po vytvoření profilu pacienta ho bude možné upravit a smazat.

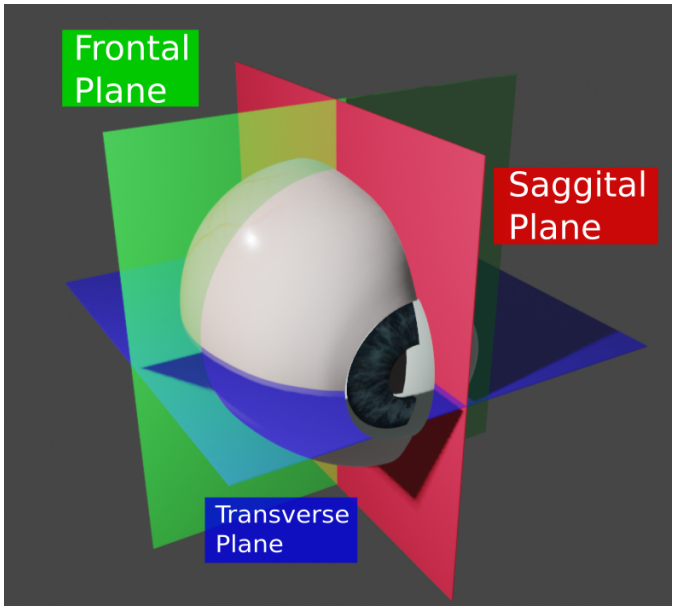
1.6.2 Důležité funkční požadavky

F7 – Autentizace doktorů
Pro přístup do aplikace se bude muset doktor přihlásit svým e-mailem a heslem. Toto je důležité, protože aplikace bude ukládat citlivé údaje pacientů, ke kterým by měl mít přístup jenom lékař. Zároveň se díky tomu automaticky identifikuje doktor, který vytváří nové měření.

F8 – Vyhledávání pacientů
V aplikaci bude možné vyhledat konrétního pacienta pomocí vyhledávacího pole.

F9 – Řezy oka
Po vytvoření nového meření nebo jeho úpravě, se automaticky vygenerují obrázky řezů modelem oka. Řezy okem jsou znázorněny na obrázku 1.5.
Generovat se budou tyto řezy:

- Frontální (Frontal plane)
- Sagitální (Saggital plane)
- Transverzální (Transverse Plane)



Obrázek 1.5 Řezy modelem oka

F10 – Poznámky
U profilu pacienta a u jednotlivých měření bude možné si dělat poznámky. Textový editor poznámek bude mít funkce:

- Zvýraznění písma – **Bold**, *Italic*
- Seznam – Vytvoření bullet points seznamu

F11 – QR kód jako odkaz na měření / model oka
Z měření bude možné vygenerovat QR kód, který bude odkazovat na toto konkrétní měření, nebo na model oka z tohoto měření vygenerovaný.

F12 – Report
Z měření bude možné vygenerovat Report, na kterém bodou základní informace o pacientovi, obrázky řezů viz. F9 a QR kód daného měření viz. F11. Tento požadavek je důležitý zejména kvůli tomu, že řada doktorů je zvyklá pracovat s papírovými dokumenty. Příprava na operaci skrz tuto tištěnou formu shrnutí měření, by pro ně měla být příjemnější.

F13 – Ukládání snímků
Během manipulace s okem viz. F2, bude možnost pořídit snímek současného pohledu na model oka, který se uloží k měření ze kterého byl model vygenerován.

F14 – Ukládání pohledů
Během manipulace s okem viz. F2, bude možnost si pohled uložit. Pohled se bude ukládat jako souřadnice kamery (x, y, z) plus quaternion (natočení v 3D prostoru). Pohledy se budou ukládat k měření, z něhož byl prohlížený model vygenerován. Uložené pohledy se budou zobrazovat při manipulaci s modelem. Po kliknutí na pohled se kamera přesune do polohy pohledu.

F15 – Skrývání částí modelu
Během manipulace s okem viz. F2, bude možnost upravovat viditelnost jednotlivých částí modelu.
Skrývatelné části:

- Rohovka
- Bulva
- Čočka
- Duhovka

F16 – Nastavitelná velikost zorničnky
Během manipulace s okem viz. F2, bude možnost upravovat rozšíření zorničky. Doktor si tímto způsobem může ověřit, jestli při daném průměru uvidí orientační body na umělé čočce.

1.6.3 Zlepšující funkční požadavky

F17 – Šipky u zadávání parametrů
U zadávání parametrů budou šipky, které budou posouvat zadanou hodnotu o jednotku nejmenší části zadávaného čísla.

F18 – Google login
Do aplikace se bude doktor moci přihlásit pomocí autorizovaného google účtu.

F19 – Zobrazení správné polohy čočky
Správná poloha čočky se bude zobrazovat jako její transparentní kopie. Zároveň se budou zobrazovat rotace a vzdálenosti, o které se čočka musí posunout, aby se dostala do správné polohy.

1.6.4 Nutné nefunkční požadavky

N1 – Dostupnost na PC, Tabletu a mobilu
Aplikace bude dostupná na počítači s operačním systémem Windows a na tabletu a mobilu s operačním systémem android i IOS.

N2 – Renderování skrz Blender
Model oka se bude generovat pomocí Blender render enginu. Blender umožňuje generování pomocí skriptů napsaných v jazyce python. Je tedy možné generování automatizovat.

N3 – Data pacientů a měření bude možné vyplnit jen částečně
Založit profil pacienta, nebo vytvořit nové měření bude možné po vyplnění jen některých (povinných) údajů.

N4 – Zadávání parametrů do textových polí
Parametry oka se do aplikace budou zadávat do textových polí

N5 – Řezy oka
Řezy oka z F9 se budou zobrazovat u dat z měření

N6 – Snímky oka
Snímky oka z F13 se budou zobrazovat u dat z měření

1.6.5 Důležité nefunkční požadavky

N7 – Autentizace pomocí protokolu Oauth2.0
Přihlášení doktorů bode probíhat pomocí protokolu Oauth2.0

N8 – Textury Bělma-cévy
Bělmo modelu oka bude mít texturu vytvořenou z fotografie pacienta oka. Důležité na textuře budou cévy na bělmu. Pomocí těchto cév se bude dát identifikovat pacient, nebo namapovat model přímo na pacientovo oko.

1.6.6 Zlepšující nefunkční požadavky

N9 – Omezený pohyb umělé čočky
Haptiky čočky a její pohyb budou omezeny prostorem ohraničeným kapsulou oka, kde je umělá čočka umístěna.

N10 – Raytracing
Model bude simulovat paprsky světla, aby mohl vypočítat z odrazů ideální polohu čočky v oku.

N11 – Textury duhovky
Model oka by měl mít k dispozici několik textur duhovky, které se budou náhodně vybírat při generování modelu. Nebo se textura duhovky bude vyrábět přímo z fotky pacientova oka.

N12 – Tmavý mód
Aplikace bude mít tmavý mód, který šetří uživatelův zrak.

1.6.7 Funkční a nefunkční požadavky pro frontend

Zde jen vypíšu ty funkční a nefunkční požadavky, které se přímo týkají mé části bakalářské práce.

- **F3** – Ukládání měření
- **F4** – Úprava a smazání měření
- **F5** – Ukládání pacientů
- **F6** – Úprava a smazání pacienta
- **F7** – Autentizace doktorů
- **F8** – Vyhledávání pacientů
- **F10** – Poznámky
- **F11** – QR kód jako odkaz na měření / model oka
- **F12** – Report
- **F17** – Šipky u zadávání parametrů
- **F18** – Google login
- **N1** – Dostupnost na PC, Tabletu a mobilu
- **N3** – Data pacientů a měření bude možné vyplnit jen částečně
- **N4** – Zadávání parametrů do textových polí
- **N5** – Řezy oka
- **N6** – Snímky oka
- **N7** – Autentizace pomocí protokolu Oauth2.0
- **N12** – Tmavý mód

Kapitola 2
Návrh

Na začátku návrhu bych se chtěl věnovat architektuře, podle které budu aplikace vyvíjet a popíšu technologie, které se při vývoji používají. Následně vytvořím workflow diagramy, které mi pomůžou ověřit splnění funkčních požadavků a navrhnu domain diagram, kterým popíšu rozvržení dat v jednotlivých částech aplikace.

Dále bych se chtěl zaměřit v návrhu na authentikaci uživatelů a protokol oAuth2, neboť se v aplikaci bude pracovat s citlivými daty pacientů.

Nakonec vytvořím návrh uživatelského rozhraní v podobě wireframů.

2.1 Architektura

Architektura je jednou z nejdůležitějších věcí při návrhu aplikace. Architekturou se myslí set pouček a pravidel, kterými by se měl vývojář při návrhu a implementaci řídit. Výběr vhodné architektury a dodržování jejich principů, pomáhá docílit čistého, snadno rozšířitelného a hlavně funkčního kódu. [36]

Jak už jsem naznačil v analýze současného stavu kódu, je nutné rozdělit komponenty, data a byznysovou logiku aplikace. V budoucnu navíc bude potřeba, kvůli nedokončenému backendu, měnit adresu back-endu, ze kterého si aplikace bere data. Kvůli tomu je potřeba jí navrhnout tak, aby byla vrstva dat a logiky od uživatelského rozhraní co nejvíce oddělená a na sobě nezávislá. K tomu účelu přesně slouží architektury MVC a MVP. Více o těchto architekturách řeknu v následujíci sekci 2.1.1.

Protože už vím, že budu vyvíjet aplikaci ve frameworku React, mohu se při výběru architektury omezit na řešení speciálně mířené právě na React. Společnost Meta, přišla s vlastním řešením architektury pro vývoj aplikací mířený přímo na vývoj v Reactu. Této architektuře se říká Flux.

Flux staví na principech definovaných v architekturách MVC a MVP a upravuje je specificky pro potřeby Reactu. Proto dám přednost při vývoji právě architektuře Flux. Více do hloubky se architektuře Flux věnuji v podsekci 2.1.2. [37]

2.1.1 MVC a MVP

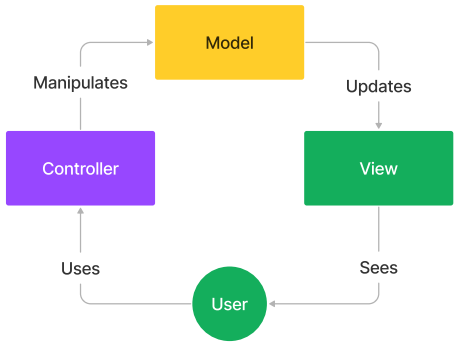
MVC a MVP neboli Model View Controller a Model View Presenter jsou oba architektonické vzory sloužící k oddělení logiky a dat od vrsty uživatelského rozhraní. Což velice zjednodušuje testování a skrývá přístup k datům. To znamená, že pokud se mi nějakým způsobem změní přístup k datům, stačí změnit vrstvu s daty a uživatelského rozhraní by se změna vůbec neměla dotknout. Nevýhodou tohoto přístupu je přidaná komplexita při oddělování vrstev. [38]

MVC

Jak již jméno napovídá, rozděluje aplikaci na tři části: Model, View a Controller. Vrstva model se stará o veškerá data aplikace a logiku s nimi spojenou. Pohled (View) si bere data z modelu a stará se o vykreslování uživatelského rozhraní a prezentační logiku. Poslední částí je controller, který je zodpovědný za aktualizování modelové vrstvy. Proud dat, který aplikací prochází se dá popsat následovně:

1. Uživatel provede akci.
2. Controller akci zaznamená a změní data v modelu.
3. Model aktualizuje pohled novými daty.
4. Pohled vykreslí uživatelské rozhraní s aktualizovanými daty.

Tento cyklus je znázorněn na diagramu 2.1.



Obrázek 2.1 MVC diagram. Vychází z diagramu na stránce [38].

Implementace této architektury s sebou, ale může přinést některé problémy. První problém je, že potřeby pohledové vrstvy se můžou snadnou přenést do modelové vrstvy, čímž se zcela naruší jejich oddělenost. Nutnost předělání jedné vrstvy pak automaticky zanemná nutnost předělání druhé.

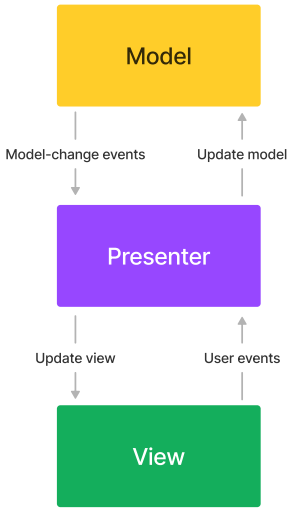
Dalším problémem je, že implementace pohledové vrstvy bývá úzce závislá na frameworku uživatelského rozhraní (v mém případě React). Což znamená, že pohledovou vrstvu je těžké testovat. [38]

MVP

MVP architektura vychází z MVC a stejně jako MVC dělí aplikaci na tři části: Model, View, Presenter. Narozdíl od MVC, ale nespecifikuje jak strukturovat celý systém. MVP se zaměřuje hlavně na strukturu pohledu.

Pohled je zodpovědný za vykreslování UI elementů. Rozhraní pohledu je pak použito k propojení presenteru s pohledem. Skrz presenter pak komunikuje pohled s modelem. Model je stejně jako v MVC zodpovědný za spravování dat a byznysovou logiku. Navíc oproti MVC modelová vrstva obsahuje současný stav aplikace. Po změně stavu Presenter inicializuje překreslení pohledové vrstvy. Komunikace vrstev je znázorněna v diagramu 2.2.

Stejně jako MVC si implementace MVP nese jisté problémy. Protože MVP nemá kontrolní vrstvu, musí se o kontrolu akcí starat prezentační vrstva. To znamená, že je zodpovědná za dvě rozdílné věci: aktualizování modelu a prezentování modelu. [38]



Obrázek 2.2 MVP diagram. Vychází z diagramu na stránce [38].

Shrnutí

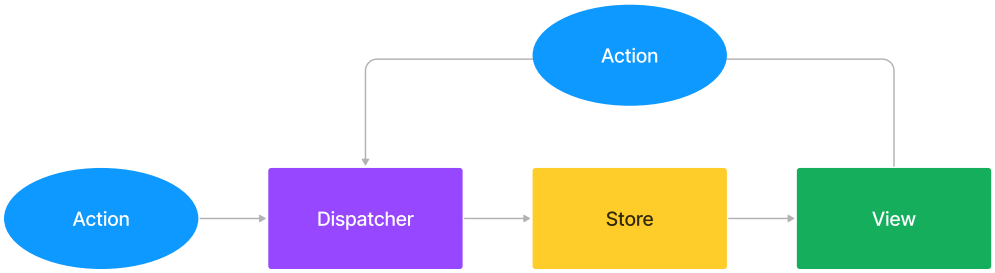
MVC a MVP jsou v jádru stejné architektury. Obě se snaží rozdělit zodpovědnosti do samostatných komponent a tím docílit nízké provázanosti UI, byznysové logiky a dat. Největší rozdíl mezi architekturami je až v implementaci. Hlavními rozdíly jsou:

1. V MVC jsou model a pohled vysoce provázané zatímco v MVP nízce.
2. V MVP komunikace probíhá User -> View -> Presenter -> Model -> Presenter -> View. V MVC komunikace probíhá User -> Controller -> Model -> View.
3. V MVC řeší uživatelský vstup controller, který ovládá model. V MVP je uživatelský vstup řešen přímo pohledem, který dále volá presenter.
4. V MVC je controller zodpovědný za kontrolování více pohledů. V MVP je jeden presenter zodpovedný za jeden pohled.
5. V MVC je hlavní komponentou Controller, zatímco v MVP je hlavním prvkem pohled.

2.1.2 Flux

Jak jsem říkal v úvodu této sekce 2.1 při vývoji budu používat architekturu Flux. Flux je architektura navržená společností Meta, speciálně pro vývoj aplikací s frameworkem React.

Stejně jako MVC nebo MVP dělí aplikaci do tří vrstev: Dispatcher, Stores, Views. Hlavním rozdílem mezi výše popsanými architekturami a architekturou Flux je, že v MVC a v MVP jedna uživatelská interakce, vede až k několika voláním skrz různé komponenty. Jinými slovy závislosti mezi komponentami se větví. Chyba v takvém kódu se špatně hledá, proto se architektura Flux soustředí na to, aby cesta v grafu volání byla vždy jen jedna (single direction flow). Jak vypadá komunikace mezi vrstvami je vidět na diagramu 2.3. Dále konkrétněji popíšu zodpovědnosti jednotlivých vrstev. [39, 40, 41]



Obrázek 2.3 Diagram Flux architektury. Vychází z diagramu na stránce [41].

Dispatcher

Dispatcher funguje jako rozcestník pro data v celé aplikaci. V této vrstvě se registrují zpětná volání do datové vrstvy (Stores) a slouží jako mechanismus pro distribuování a ovládání akcí, které data mění. V podstatě se jedná o frontu akcí, které se postupně vykonávají. Dispatcher poskytuje jednoduché rozhraní [39, 40, 41]:

1. Register() – metoda pomocí které se registrují zpětná volání
2. Unregister() – metoda pro odhlašování zpětných volání
3. WaitFor() – metoda pomocí které můžeme říct dispatcheru, aby vyčkal na zavolání jiného zpětného volání
4. Dispatch() – metoda pro zavolání akce
5. isDispatching() – metoda která říká, jestli dispatcher volá nějakou akci

Stores

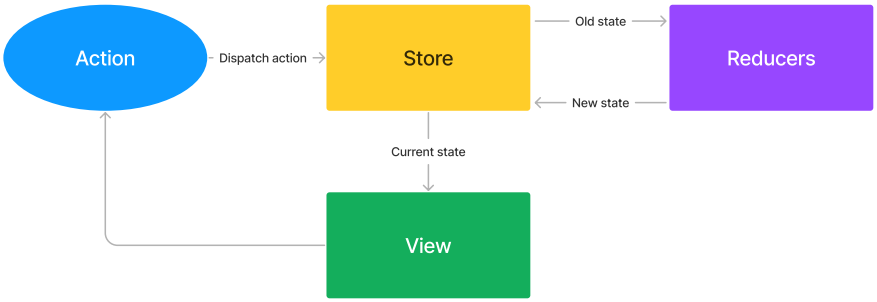
Tato vrstva slouží pro ukládání stavu aplikace. Podobá se Modelu v architektuře MVC. Store se aktualizuje pomocí zpětných volání vyvolaných nějakou akcí. Pokud akce změní stav aplikace, upozorní store na změnu pohled, který data používá. Aplikace může mít více než jeden store. [39, 40, 41]

Views

Views, nebo také controller-views. Hlavní účel této vrstvy je zobrazovat data. Tato vrstva ale obsahuje i logiku pro generování akcí a příjmání dat ze store vrstvy. Kdykoliv uživatel interaguje s pohledem, vyšle pohled pomocí dispatcheru akci. Pokud pohled zaznamená změnu ve stavu aplikace, překreslí se. [39, 40, 41]

Actions

Akce představují události a data posílané pohledem. Akce typicky obsahuje dva druhy informací: O jaký typ akce se jedná a payload. Payload obsahuje data a informace o změně stavu aplikace. V akci se také například mohou nacházet volání na back-end. Akce jsou z pohledů odesílány pomocí dispatcher vrstvy do vrstvy store. [39, 40, 41]



Obrázek 2.4 Diagram Reduxu. Vychází z daimgramu na stránce [43].

2.1.3 Redux

Redux je knihovna implementující Flux architekturu. Stará se o udržování a manipulování stavu aplikace. Pomáhá rozdělit úzké spojení mezi uživatelskými interakcemi a změnou stavu. To se hodí zejména pokud nějaká změna stavu ovlivňuje více komponent pohledu najednou. Knihovna také existuje ve verzi, která je určená přímo pro použití s Reactem, což ušetří spoustu práce při integrování nové architektury do projektu. [42, 43]

Redux implementuje Flux architekturu téměř přesně tak, jak jsem jí popsal. Změna v Reduxu oproti Flux architektuře je, že v ní přibyla vrstva reducers. Reducery přebírají zodpovědnost za rozhodování o změnách dat na základě typu akce (Flux tuto zodpovědnost nechává přímo na store vrstvě).

Další důležitou změnou je, že Redux většinou pužívá jeden velký neměnný store. Každá akce dostane na vstup starý stav aplikace, aplikuje na něj změny a vrátí stav nový, který ten starý nahradí. To je výhodné pokud bych chtěl implementovat například zpětné tlačítko. To by šlo udělat snadno ukládáním starých stavů aplikace. Změny komunikace mezi vrsvami v Reduxu oproti Fluxu jsou nejlépe vidět na diagramu 2.4. [43]

2.2 Protokol oAuth2

Protože aplikace bude využívat pro autentizaci protokol oAuth2, vysvětlím zde jak funguje a jak se sním pracuje.

Předtím než začnu s popisem oAuth2 musím zmínit, že tento protkol neslouží k autentizaci, ale k autorizaci. Autenitzace je ale nedílnou součástí autorizačního procesu, ze kterého uživatel vidí pouze tu autentizační část (přihlašovací formulář). Proto jsem doposud zmiňoval oAuth2 protokol ve spojení s autentizací. Dále už o něm ale budu mluvit jako o protokolu autorizačním.

Role

Protkol oAuth2 definuje aktéry a jejich role, které se v autorizačním procesu vyskytují. Aby jsem lépe mohl popsat samotný autorizační proces, vysvětlím nejdříve význam těchto rolí. Popis vychází z [44, 45].

Majitel zdrojů: Představuje systém nebo uživatele vlastnící zdroje, ke kterým chci přistupovat.

Klient: Klient je systém požadující přístup ke chráněným zdrojům. Aby k nim mohl klient přistupovat, musí vlastnit patřičný přístupový token.

Autorizační server: Server, který vyřizuje požadavky na přístupové tokeny od klienta. Aby od serveru token dostal musí se klient autentizovat. Autorizační server se skládá ze dvou částí: Autorizační – která řeší autentizaci klienta a tokenovou – která se stará o tokeny a komunikaci mezi servery.

Server se zdroji: Server poskytující klientovi na vyžádání zdroje. Zdroje klientovi poskytuje na základě přístupových tokenů připojených k žádosti.

Komunikace

Teď popíšu, jak mezi sebou jednotlivé role během autorizace komunikují. Podle [44, 45] komunikace probíhá následovně:

1. Celý proces zahájí klient, který si vyžádá autorizaci od autorizačního serveru. Klient předá autorizačnímu serveru informace o rozsahu práv a cílovou adresu kam má autorizační server poslat přístupvý token, nebo autorizační kód.
2. Předtím než autorizační server pošle klentovi přístupový token nebo autorizační kód, požádá klienta, aby se identifikoval (v mém případě se bude autentizovat samotný uživatel svými přihlašovacími údaji ke svému Google účtu). Autorizační server ještě ověří, jestli majitel zdrojů povoluje přístup klienta ke zdrojům v daném rozsahu práv.
3. Server ze zdroji komunikuje s autorizačním serverem (tokenová část), ze kterého získá informace o uživateli.
4. Autorizační server přesměruje klienta na adresu specifikovanou v kroku jedna a předá mu autorizační kód nebo přístupvý token v závislosti na typu předání. Typu předání se budu věnovat v následující sekci. Autorizační server také může předat klientovi obnovovací token. Tímto tokenem lze zažádat o další přístupový token, pokud by současný vypršel a byl neplatný.

Předání typu PKCE

PKCE neboli Proof Key for Code Exchange upravuje základní práci s oAuth2 protokolem. Tato verze řeší problém s uložením Autorizačního kódu přímo v prohlížeči klienta u SPA. Protože je celý zdrojový kód stránky prohlížeči k dispozici, uložením autorizačního kódu riskuji jeho vyzrazení třetím stranám.

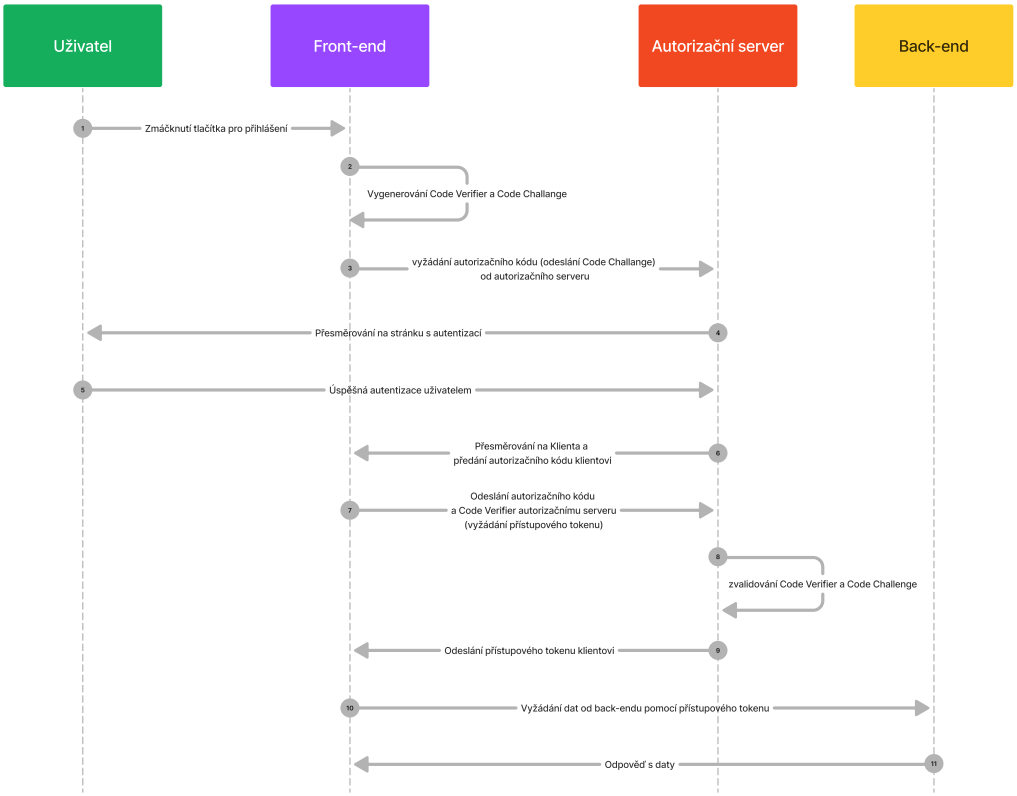
Předání typu PKCE odstraňuje tuto slabinu tím, že si klient vytvoří kód ověřitelný autorizačním serverem. Tento kód se nazývá code verifier. K tomu, aby autorizační server mohl kód ověřit, vytvoří klient další kód jako transformaci prvního kódu. Tento druhý kód se nazývá code challenge. Code challenge pošle klient autorizačnímu serveru společně s požadavkem v bodě jedna z 2.2.

Tímto způsobem nemůže útočník získat přístupový token pomocí ukradeného atorizačního kódu, protože k získání uživatelského tokenu potřebuje kromě autorizačního kódu ještě code verifier. Celá komunikace s přdáním typu PKCE je zachycená na diagramu 2.5. [46]

2.3 Workflow diagramy

Pomocí workflow diagramů ukážu, jak bude probíhat použití aplikace a jaké funkce bude mít. Nakonec na základě těchto diagramů odvodím pokrytí funkčních požadavků.

Každý diagram reprezentuje jednu stránku, kterou aplikace bude mít. Pro rozdělení na více stránek jsem se rozhodl, protože se tím celkové množství informací na každé stránce zmenší a zpřehlední. Zároveň mi toto rozdělení dovolí umístit akce a funkcionality, které spolu souvisí na jedno místo.



Obrázek 2.5 diagram oAuth2 typu PCKE. Vychází z obrázku na stránce [47].

Přechody mezi stránkami

Na diagramu 2.6 můžeme vidět, jak se uživatel dostane z jedné stránky aplikace na další. Všechny stránky kromě přihlašovací budou navíc obsahovat navigační panel, který bude fungovat jako rozcestník mezi stránkami. Pomocí panelu se bude moct uživatel dostat na stránku pro tvorbu nového pacienta, na stránku se seznamem pacientů, nebo se odhlásit.

Přihlášení

Aplikace by měla na tuto stránku uživatele přesměrovat pokaždé, když uživatel nebude přihlášený. Po přihlášení aplikace uživatele pustí dále na požadovanou stránku. Práce se stránkou je znázorněna na diagramu 2.7. Tato stránka pokrývá funkční požadavky F7 a F18.

Výběr pacienta

Tato stránka slouží primárně pro vyhledání a zvolení konkrétního pacienta. Dále se odtud bude uživatel moct dostat na stránku pro vytváření nového pacienta. Práce se stránkou je znázorněna na diagramu 2.8. Tato stránka pokrývá funkční požadavky F5 a F8.

Nový pacient / Upravit pacienta

Toto je jednoduchá stránka s formulářem s údaji o pacientovi s možností formulář odeslat, nebo ho zrušit a vrátit se zpět. Práce se stránkou je znázorněna na diagramu 2.9. Tato stránka pokrývá funkční požadavky F5, F6.

Detail pacienta

Tato stránka slouží jako rozcestník s akcemi týkajících se konkrétního pacienta. Nachází se zde informace o pacientovi s možností jejich úpravy. Zde se budou zobrazovat seznam s měřeními podstoupené pacientem a možnost přejít na stránku pro vytvoření dalších. Budou zde poznámky k pacientovi s možností přejít k jejich úpravě. Práce se stránkou je znázorněna na diagramu 2.10. Tato stránka pokrývá funkční požadavky F5, F6, F10.

Detail měření

Tato stránka slouží jako rozcestník s akcemi týkajících se konkrétního měření. Nachází se zde informace o měření s možností jejich úpravy. Také se zde zobrazují poznámky o měření s možností jejich úpravy. Dále jsou tu tlačítka pro provedení akcí: Vygenerovat report, Vygenerovat model oka a zpět na detial pacienta. Práce se stránkou je znázorněna na diagramu 2.11. Tato stránka pokrývá funčkní požadavky F3, F4, F11, F12.

Nové měření / Upravit měření

Toto je další jednoduchá stránka, tentokrát s formulářem s údaji o měření. Práce s formulářem je stejná jako s formulářem pro vytvoření nového pacienta a popisuje ho stejný diagram 2.9. Tato stránka pokrývá funkční požadavky F3, F4 a F17.

Poznámky

Na této stránce může uživatel upravovat poznámky. K dispozici má funkce pro úpravu textu: Bold, Italic, Underline a vytvořit seřazený a neseřazený seznam. Práce se stránkou je znázorněna na diagramu 2.12. Tato stránka pokrývá funkční požadavek F10.

Shrnutí požadavků

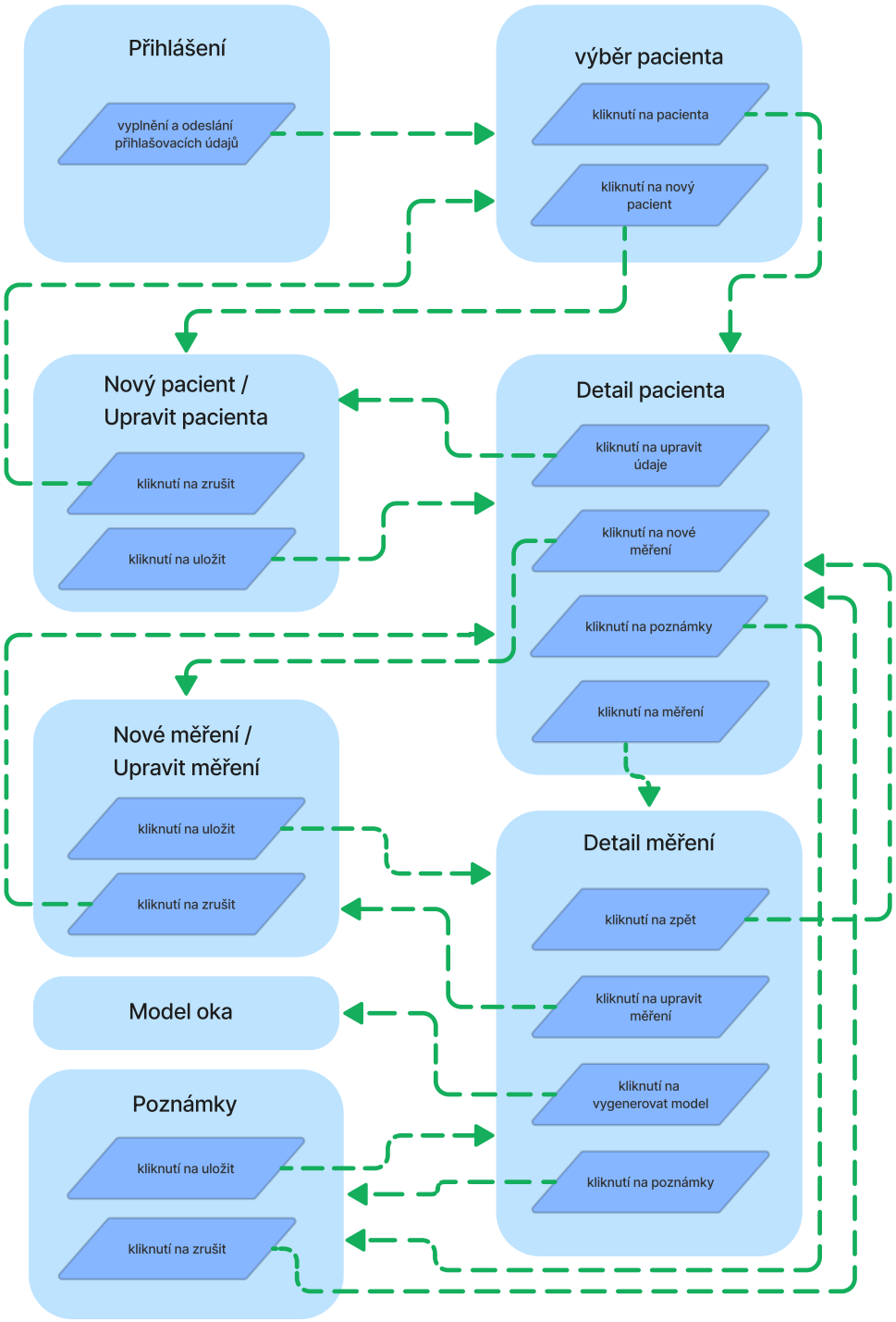
Zde pomocí tabulky 2.1 shrnu pokrytí funkčních požadavků stránkami popsanými v minulé sekci. Z tabulky je jasně vidět, že diagramy pokrývají všechny funkční požadavky.

Tabulka 2.1 Přehled pokrytí funkčních požadavků

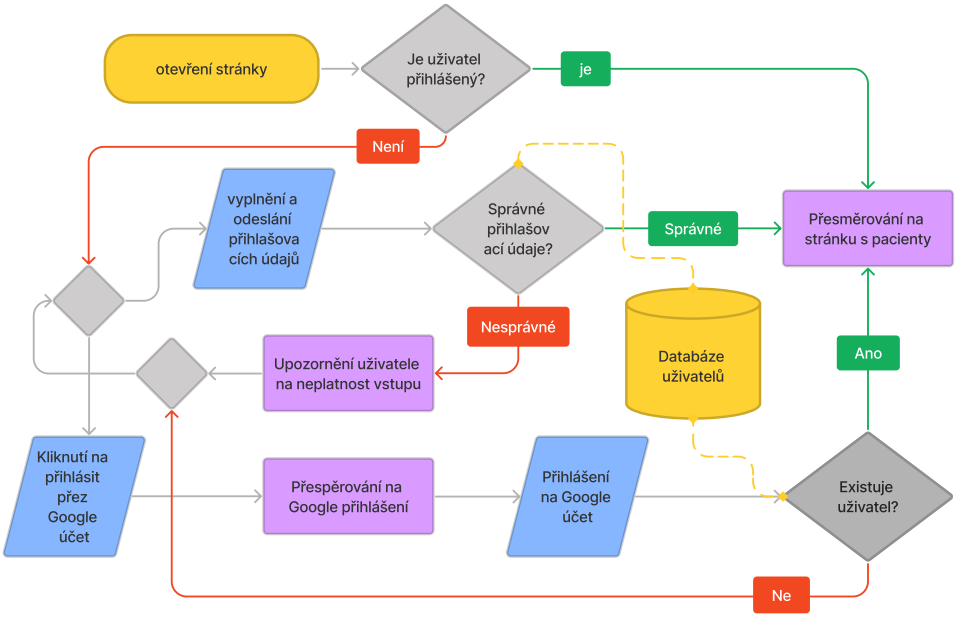
	F3	F4	F5	F6	F7	F8	F10	F11	F12	F17	F18
Přihlášení					-						-
Výběr pacienta			-			-					
Nový pacient			-	-							
Detail pacienta		-	-			-					
Detail měření	-	-						-	-		
Nové měření	-	-								-	
Poznámky							-				

2.4 Doménový model

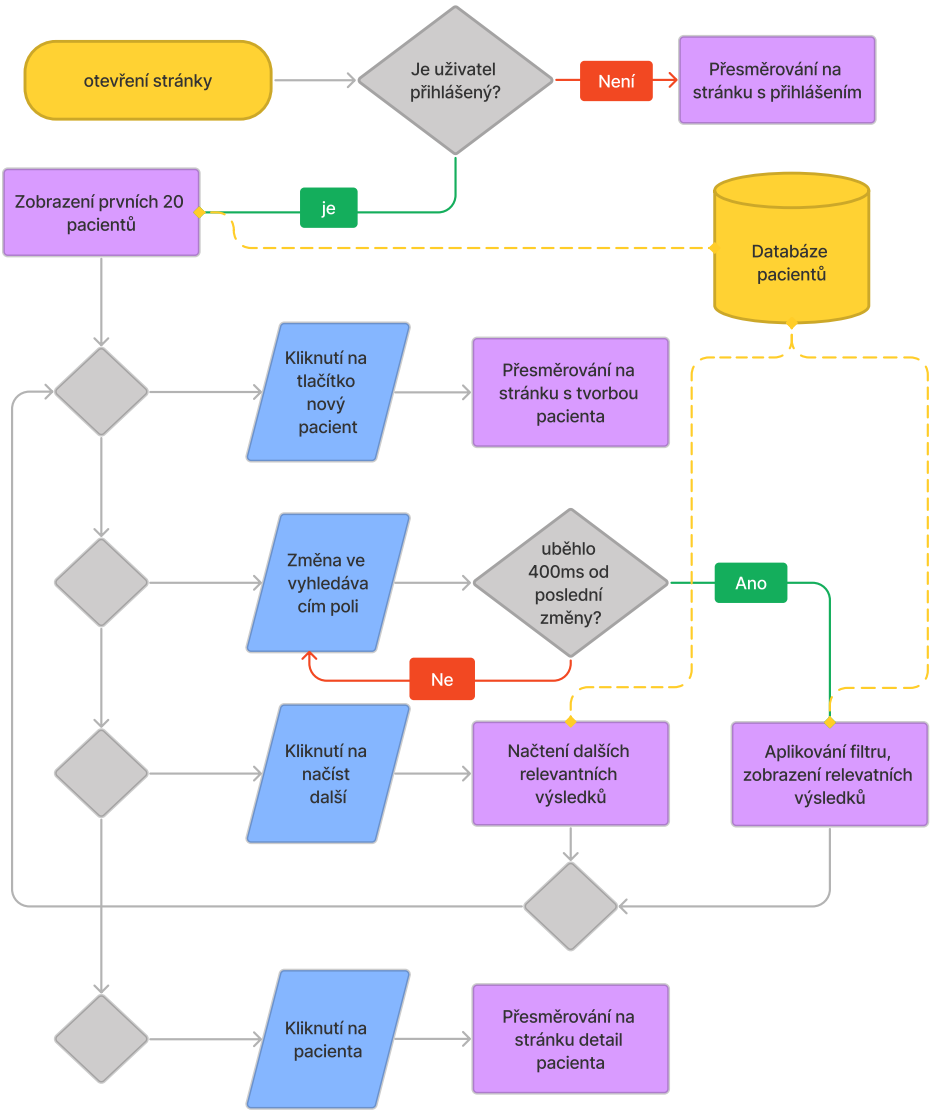
V minulé části jsem ukázal jak by měl vypadat průběh použití aplikace a jednotlivých stránek. Teď pomocí doménového modelu přiblížím s jakými daty bude aplikace pracovat a jaké jsou mezi nimi vazby. Diagram 2.13 je modelován v jazyce UML.



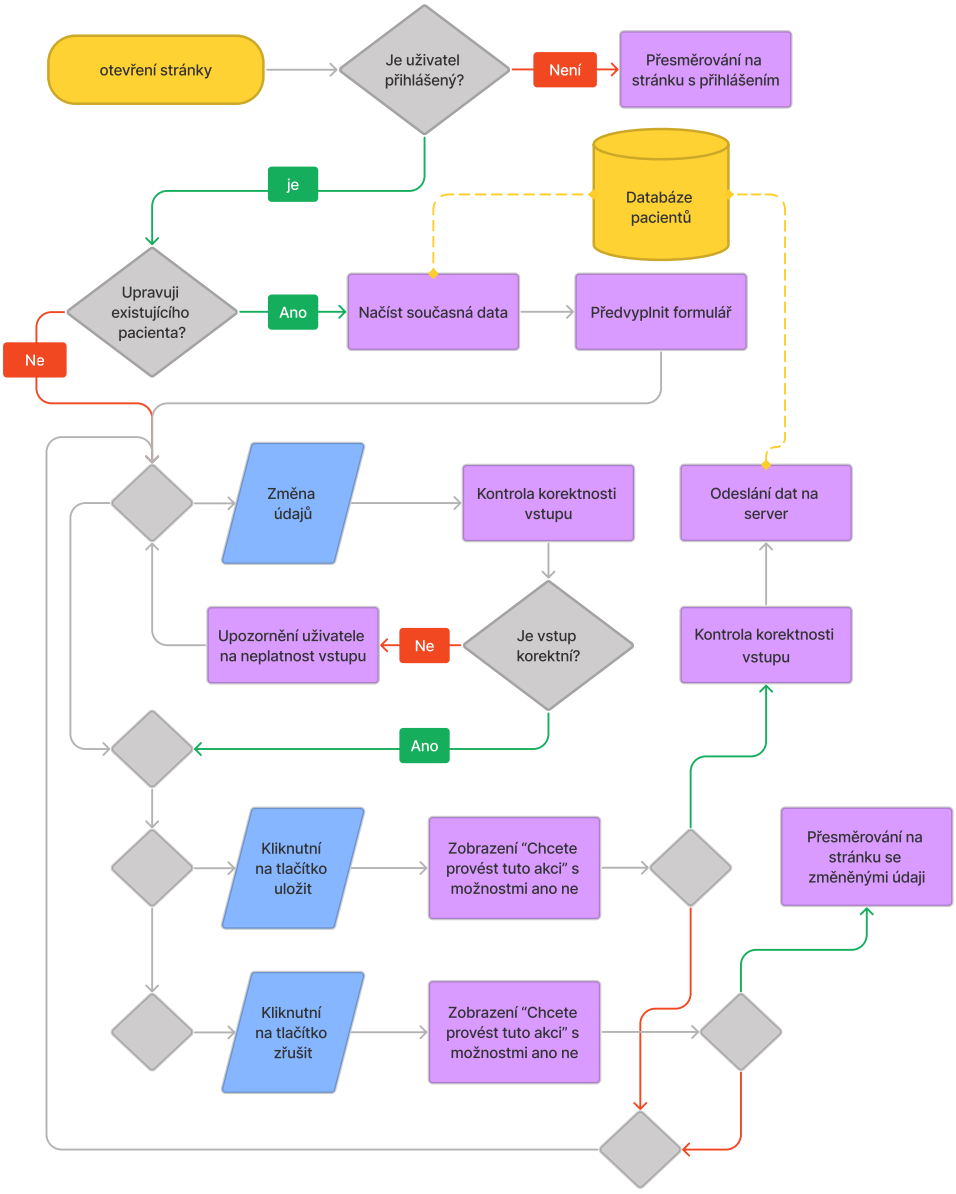
Obrázek 2.6 Přechodový diagram stránek



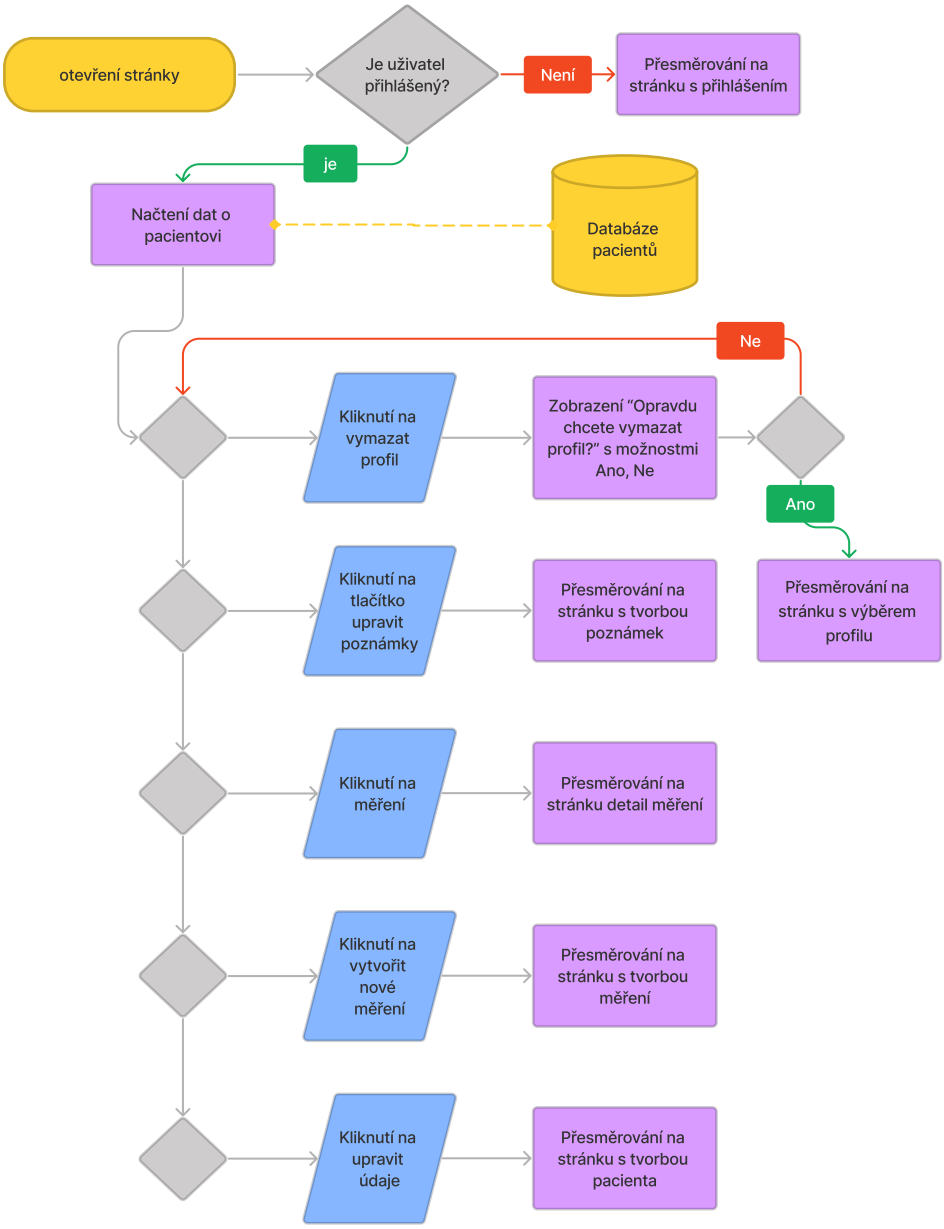
Obrázek 2.7 FlowChart diagram přihlášení



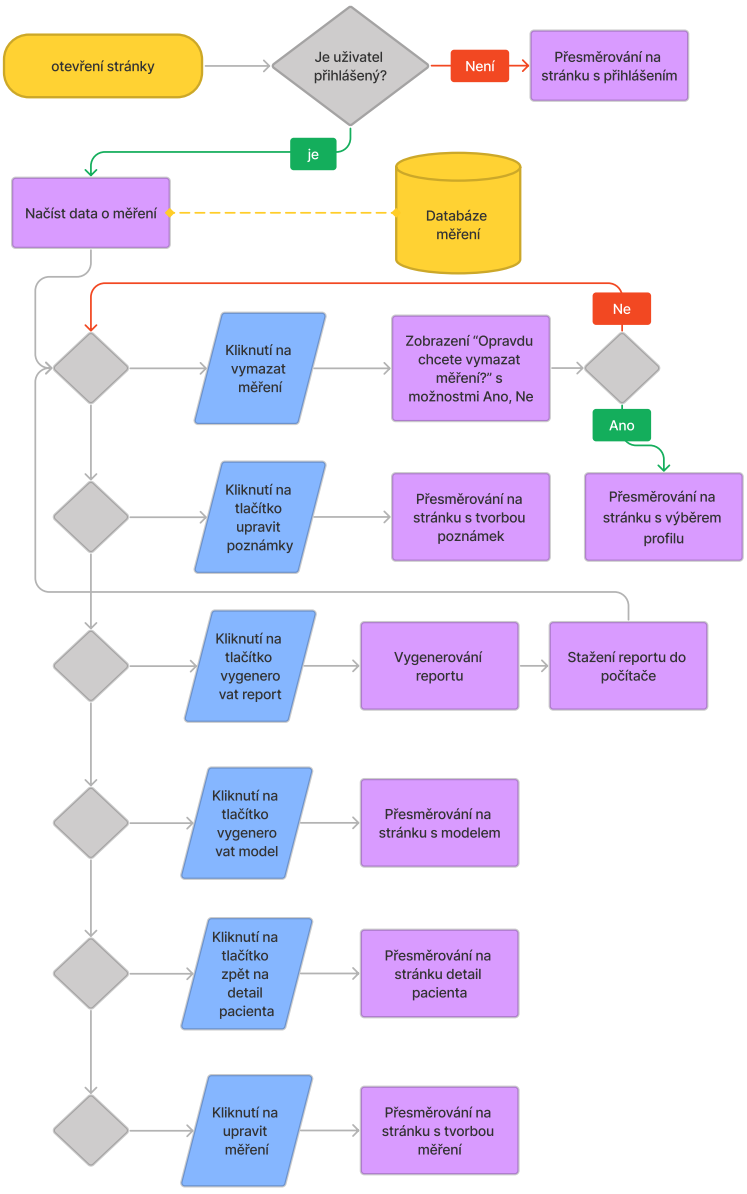
Obrázek 2.8 FlowChart diagram výběru pacienta



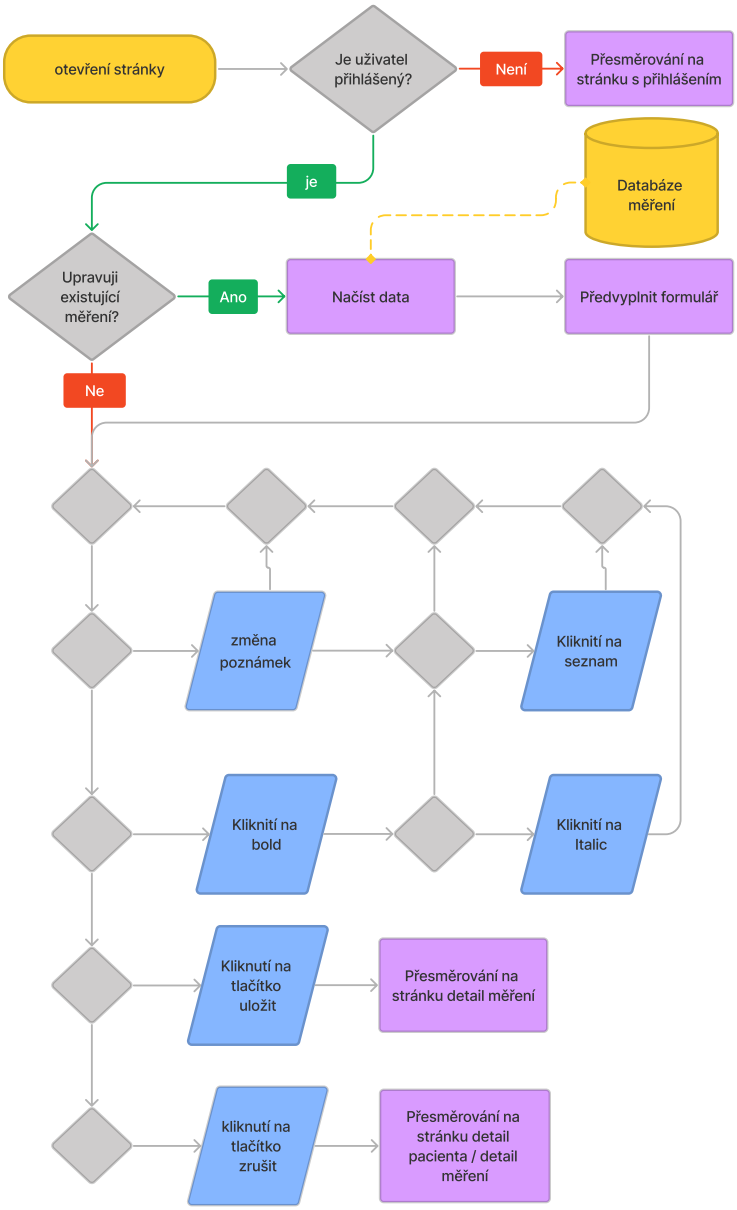
Obrázek 2.9 FlowChart diagram formulářů nový pacient a nové měření



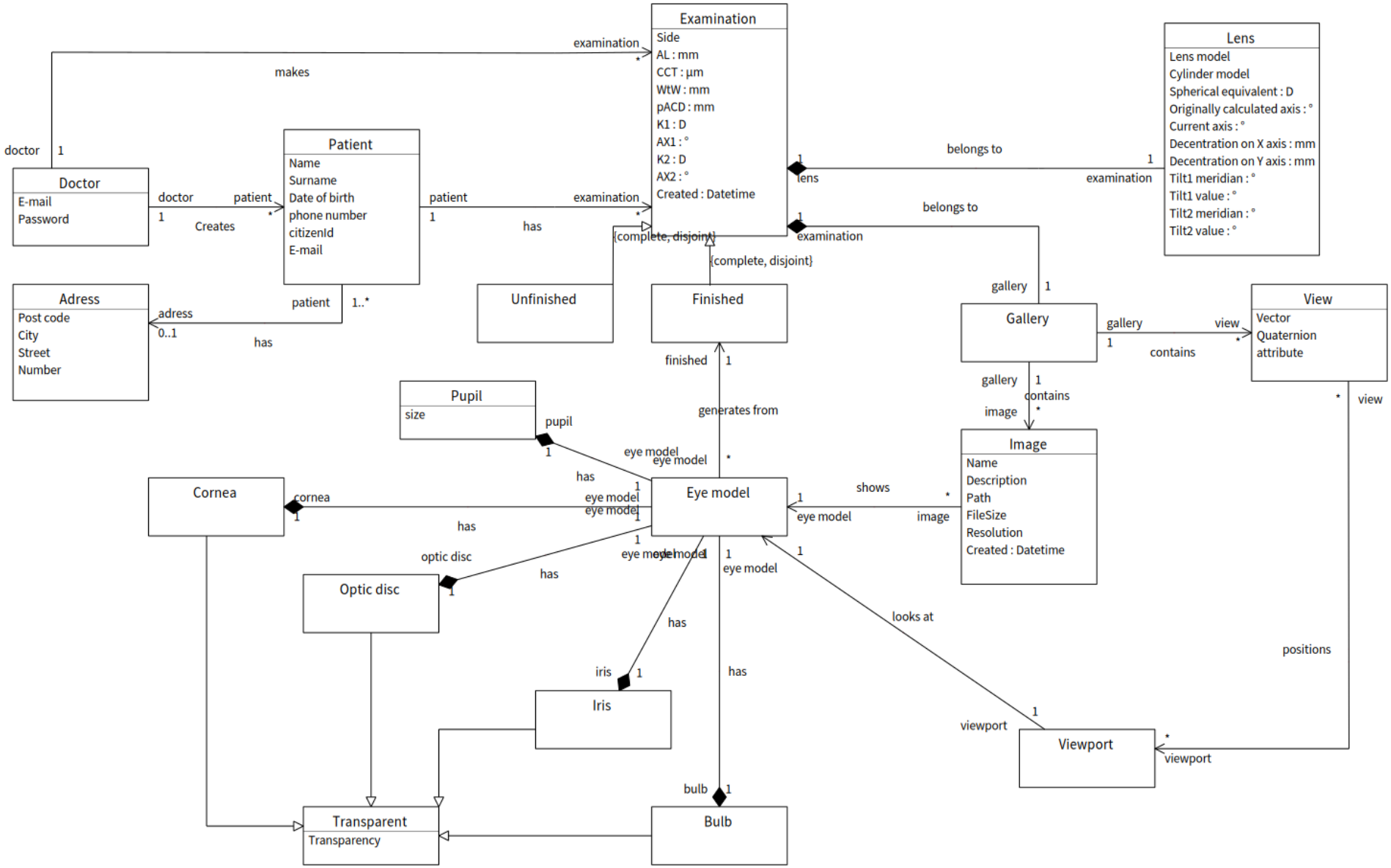
Obrázek 2.10 FlowChart diagram detailu pacienta



Obrázek 2.11 FlowChart diagram detailu měření



Obrázek 2.12 FlowChart diagram úpravy poznámek



Obrázek 2.13 Doménový diagram

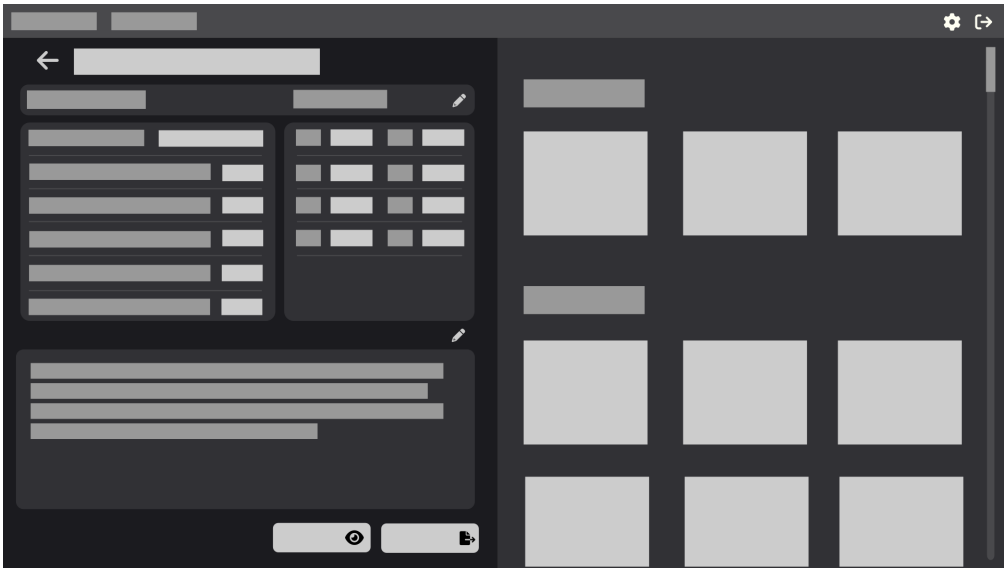
2.5 Wireframe

Jako poslední věc jsem provedl návrh uživatelského rozhraní aplikace v podobě wireframů. Wireframy jsem nadesignoval bez barev pouze v černobílé.

Myšlenka za tím je taková, že mi stačí naznačit úroveň kontrastu pomocí různé úrovně světlosti prvků. Barvy pak už můžu přidat snadno. Při tvorbě barevného schématu se budu zaměřovat na tmavý režim.

Jako ukázku zde uvedu wireframe stránek detail měření 2.15 a detail pacienta 2.14, zbytek wireframů se nachází v příloze.

Obrázek 2.14 Wireframe stránky detail pacienta



Obrázek 2.15 Wireframe stránky detail měření

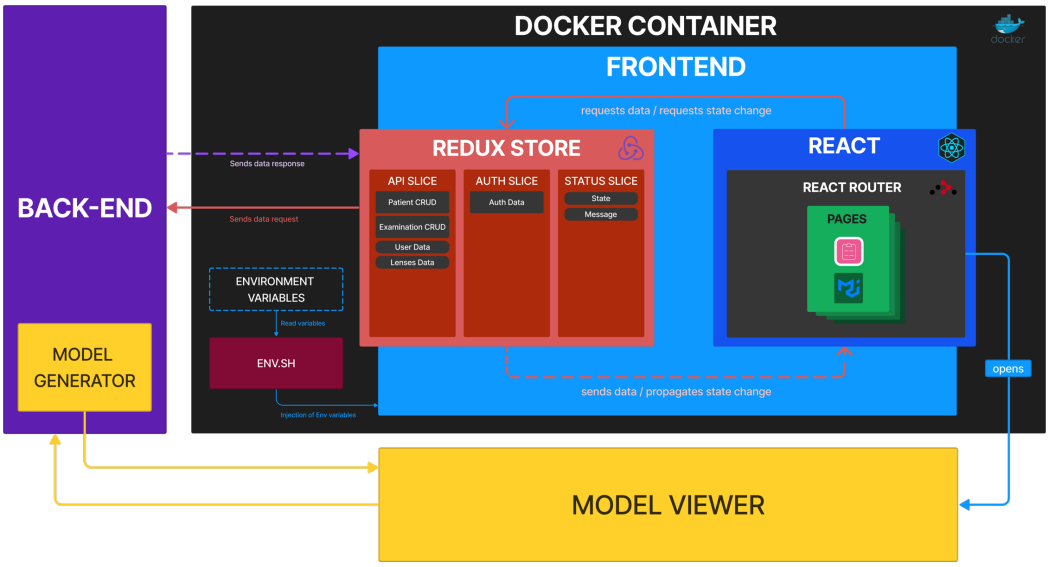
Kapitola 3

Implementace

V této kapitole se budu věnovat implementaci prototypu a poukážu na změny oproti návrhu. Začnu programátorskou příručkou, kde se budu zabývat zasazením prototypu do celého systému a nejdůležitějšími částmi kódu. Nakonec v ní uvedu jak funguje nasazování aplikace na vývojářský a produkční server.

V další části se zaměřím na instalační příručku, ve které vysvětlím proces instalace prototypu na vlastním zařízení.

Kapitolu zakončím uživatelskou příručkou, kde ukážu výsledný vzhled aplikace a popíšu její uživatelské rozhraní.



Obrázek 3.1 Diagram zasazení aplikace do celkové struktury systému

3.1 Programátorská příručka

V této sekci se zaměřím hlavně na kód aplikace a poukážu na jeho fundamentální části, tak aby se budoucí programátor, který k projektu přijde, snadno zorientoval.

Popis všech metod a komponent se pak nachází přímo v kódu v podobě TSDoc komentářů. Původně jsem plánoval z těchto komentářů vygenerovat HTML dokumentaci. Vygenerovaná dokumentace ale nedávala pro funkcionální React komponenty moc smysl, protože je považovala za obyčejné funkce a neukazovala jejich vnitřní procesy. Proto jsem se rozhodl nechat dokumentaci přímo v kódu, kde jsou dobře vidět vzájemné závislosti mezi komponentami a hlavně celá jejich implementace.

Nakonci této sekce budu mluvit o nasazování aplikace na server a o prostředcích co k tomu využívám.

Poznámka: Tato sekce obsahuje značné množství slangových a anglických výrazů, neboť to popis výrazně zjednodušuje a přispívá k srozumitelnosti textu ve spojení s kódem.

3.1.1 Obecná struktura

Jak jsem popisoval v návrhu, k implementaci jsem použil architekturu Flux. Na implementaci vrstev Dispatcher, Actions a Stores jsem použil knihovnu Redux.

Redux se stará o data a o jejich distribuci napříč aplikací. Srdcem této knihovny je Store, který představuje úložiště a správce dat. Data z tohoto úložiště jsou dostupná pro všechny komponenty a ze všech komponent můžu provádět jejich úpravu.

Roli vrstvy Views zastává React. React se stará o vykreslování a manipulaci s uživatelským rozhraním. Uživatelské rozhraní je členěné na stránky, kde každá stránka má svůj vlastní účel. Tyto stránky a jejich funkcionality kopírují diagramy ze sekce 2.3.

Struktura aplikace je dobře vidět na obrázku 3.1. Na obrázku je také vidět, jak prototyp zapadá do celého systému a jak s ostatními částmi komunikuje. Prototyp je navržen tak, aby s ostatními částmi šel propojit snadno a bez potřeby větších úprav. Co se týče změny back-endu, tak jde pouze o změnu parametru při spouštění a aplikaci ani není potřeba znovu sestavovat. Více o tom řeknu v části nasazení na server.

Pro sprovoznění komunikace mezi částí Model viewer a mojí aplikací už bude zásah do kódu potřeba, nepůjde však o nijak zásadní změny. Pokud bude mít Model viewer vlastní server, bude stačit se na něj přesměrovat s vhodnými parametry. Jinak je možné vytvořit pro Model viewer vlastní stránku přímo na front-endu a vyrobit z něj React komponentu.

3.1.2 Redux

Zde řeknu něco víc o tom jak jsem knihovnu Redux použil a nakonfiguroval.

Hlavní store jsem rozdělil na několik částí: api, auth, status. Každá z tětcho částí je zodpovědná za spravování jednoho druhu dat aplikace a to včetně jejich stahování z back-endu. Konfigurace a rozdělení úložiště je vidět na výstřižku z kódu 3.2.

V komponentách přistupuji k datům z úložiště pomocí funkcí *useSelect* a *useDispatch*. UseSelect se používá pro vybírání konkrétních dat a pomocí *useDispatch* se odesílají požadavky na změny dat. Aby jsem u těctho funkcí nemusel pokaždé specifikovávat typ, vytvořil jsem hooky, které mi vrátí funkce již správně otypované. Jde o hooky *useAppSelect* a *useAppDispatch*.

Dále řeknu něco víc o jednotlivých částech, na které je úložiště rozdělené. Popíšu jaká data definují a kde se používají.

Auth

Část Auth se stará o autentizaci uživatele. Uživatel se může autentizovat dvěma způsoby. První způsob je svým emailem a heslem. K tomu slouží akce *loginUser*, která odešle tyto údaje na

```
const store = configureStore( options: {
    reducer:{
        status: statusSlice,
        auth: authSlice,
        [apiSlice.reducerPath]:apiSlice.reducer
    },
    middleware: getDefaultMiddleware =>
        getDefaultMiddleware().concat(apiSlice.middleware)
})
```

Obrázek 3.2 Konfigurace vrstvy Store

server a zpátky dostane user token, který se pak používá k autentizaci při další komunikaci s back-endem. User token se kormě do redux úložiště navíc ještě ukládá přímo do lokálního úložiště prohlížeče. Toto je nutné pro udržení user tokenu v paměti i při znovunačtení stránky. Pokud by se token lokálně neukládal, uživatel by se musel přihlašovat znovu po každém načtení aplikace.

Druhou možností je přihlášení pomocí poskytovatele identity, zde konkrétně pomocí Google účtu. O tento druh přihlášení se stará funkce *loginWithJWTToken*. Po kliknutní na tlačítko „login with Google“ na stránce /login, se na Google api odešle požadavek k přístupu k datům uživatele. Google pak ve vyskakovacím okně požádá uživatele, aby se autentizoval. Po tom co se uživatel úspěšně přihlásí ke svému Google účtu, dostane aplikace ve zpětném volání JWT token obsahující údaje o uživateli. Zde bych rád podotknul, že Google nepodporuje protokol OAuth2 v módu PKCE o kterém jsem psal v návrhu v sekci 2.2 a prototyp tedy používá jen variantu s autorizačním kódem. V tuto chvíli přichází do hry *loginWithJWTToken* funkce, která odešle JWT token na server. Server token ověří a vrátí userToken a informace o uživateli. S těmi se pak nakládá stejně jako v případě funkce loginUser.

Api

Tato část obstarává téměř všechny akce spojené s back-endem. Její úlohou je stahovat data z back-endu a ukládat je do redux úložiště. Nachází se zde data pacientů a jejich vyšetření, také galerie a dostupné modely čoček.

Protože stahování dat ze serveru má pokaždé stejnou strukturu, použil jsem funkci createApi z knihovny redux-toolkit, která za mě vygenerovala akce a reducery potřebné pro komunikaci s úložištěm. Další velkou výhodou použití createApi je automatické cachování stahovaných dat.

Ke každému druhu dat můžu připojit tag, který se dá později, například při nějaké mutaci, invalidovat a tím vyvolat jejich automatickou aktualizaci. To mi zajistí, že aplikace zobrazuje vždy validní data.

K tomu aby mohlo úložiště s back-endem komunikovat, musí v store existovat userToken (uživatel musí být přihlášený). Token se posílá společně s dotazem na server v hlavičce jako Bearer token. Pomocí tokenu si back-end ověří, že požadavek posíla opravdový uživatel a ne útočník.

Status

Část Status je velice jednoduchá a slouží k indikaci stavu aplikace. Aplikace může být buď ve stavu Idle, Error nebo Success. Informace o stavu využívá komponenta *<StoreAlerts/>*, která zobrazuje oznámení o procesech. Když se změní stav aplikace, *<StoreAlerts/>* zobrazí vyskakovací okno a po uplynutí určtého intervalu, nebo po interakci uživatele vrátí stav zpět na Idle a oznámení zavře.

3.1.3 Stránky

Teď budu mluvit o stránkách aplikace. Stránka představuje to co se uživateli zobrazí na určité adrese aplikace (více o adresování budu mluvit v další sekci). Zde jsou stránky velké komponenty, které spojují spoustu menších dohromady.

Stránku většinou tvoří komponenty s názvem <LeftHalf/> a <RightHlaf/>. Každá stránka kromě stránky pro přihlášení je totiž rozdělena na dva sloupce pomocí komponenty *<VerticalHalfSplit/>*. Při zobrazení na zařízení s menší obrazovkou se sloupce přeskupí do jednoho. Díky tomu je obsah na stránce dobře čitelný a přístupný i na mobilu a tabletu.

Formuláře

Kromě jednoduchých komponent pro vizualizaci dat obsahují některé stránky formuláře. Na manipulaci s formulářovými daty využívám knihovnu rect-hook-form [48]. Pomocí funkce *register* lze komponenty s rozhraním vstupního pole registrovat. Knihovna pak spravuje hodnoty registrovaných komponent a předá je funkci *handleSubmit* při odeslání formuláře.

Další užitečná funkce knihovny je validace registrovaných polí. Já v kódu použil validaci povinného pole a validaci pomocí vzoru.

Pro vizualizaci formulářových polí jsem většinou použil komponentu *<TextField/>* z knihovny Material UI. Tato komponenta umí při specifikaci paramterů *error* a *helperText* zobrazit pod polem chybovou hlášku, což se hodí, když chci uživatele informovat o neplatnosti nějaké zadané hodnoty. Aby jsem nemusel pokaždé ručně tyto paramtery k poli zadávat, vytvořil jsem si hook *useRegisterWrapper* který tuto práci udělá za mě při registraci pole.

Hook *useRegisterWrapper* bere jako vstupní argumenty funkci *register* a objekt se stavem formulářových polí. Jeho návratová hodnota je funkce *customRegister*, která se dá použít pro registraci formulářových polí stejně jako funkce *register*. Funkce *customRegister* ale narozdíl od *register* pro komponentu *<TextField/>* specifikuje i parametry error a helperText. To jak funkce *customRegister* parametr helperText specifikuje, se dá ovlivnit pomocí jejího parametru *helperTextReducer*. Tento parametr by měla být reducer funkce, která na základě typu eroru vrátí požadovaný text. Pokud *helperTextReducer* není specifikovaný, použije se základní reducer.

3.1.4 Router

Směrování se používá, aby jsem mohl ovládat na jaké url adrese se objeví jaká stránka. React sám o sobě žádnou funkcionalitu na přepínaní stránek nenabízí, proto jsem sáhnul po knihovně React Router [10].

Protože je aplikace SPA jedná se pouze o virtuální přepínání stránek. Stránky se ve skutečnosti nepřepínají, jen se vykreslý jiný komponent na té samé stránce a změní se URL v prohlížeči.

V souboru App.tsx je definovaná hierarchie stránek, podle které se určuje na jaké adrese se zobrazí jaký komponent. Na obrázku 3.3 je vidět, že například stránku detail pacienta s parametrem id=12345 najdetena adrese pacients/12345.

Co je potřeba ještě u směrování zmínit jsou komponenty RequireAuth a PersistentLogin. Komponenta PersistentLogin se stará o udržování dat o přihlášeném uživateli v úložišti. Komponenta v pravidelných intervalech vyvolává nové stahování dat o uživateli ze serveru. To způsobí překreslení stránky čímž se kontroluje platnost user tokenu.

Komponenta RequireAuth pak rozděluje aplikaci na části dostupné bez autentizace (v této aplikaci se jedná jen o stránku přihlášení) a s autentizací. Pokud by se nepřihlášený uživatel zkoušel dostat na stránku která pro něj není přístupná, RequireAuth ho místo zobrazení požadované stránky přesměruje na stránku kde se může autentizovat.

Kombinace těchto dvou komponent zajišťuje, že pokud by uživatel zůstal přihlášený, po vypršení platnosti user tokenu se sám automaticky odhlásí.

```
<BrowserRouter>
  <Routes>
      <Route path={"/login"} element={<LogInPage/>}/>
      <Route element={<PersistentLogin/>}>
          <Route path={"/"} element={<ShowNavbar><RequireAuth/></ShowNavbar>}>
              <Route path={"patients"}>
                  <Route path={""} element={<PatientSelectionPage/>}/>
                  <Route path={"create"} element={<PatientNewPage/>}/>
                  <Route path={":id"}>
                      <Route path={""} element={<PatientDetailPage/>}/>
                      <Route path={"examinations"}>
                          <Route path={":exId"} element={<ExaminationDetailPage/>}/>
                          <Route path={"create"} element={<ExaminationNewPage/>}/>
                      </Route>

                      <Route path={"edit"}>
                          <Route path={"patient"} element={<PatientUpdatePage/>}/>
                          <Route path={"notes"} element={<EditPatientNotesPage/>}/>
                      </Route>
                  </Route>
              </Route>

              <Route path={"/examinations"}>
                  <Route path={":id/edit/notes"} element={<EditExaminationNotesPage/>}/>
                  <Route path={":id/edit/examination"} element={<ExaminationUpdatePage/>}/>
              </Route>
          </Route>
          <Route path={"/"} element={<RequireAuth/>}>
              <Route path={"/logout"} element={<Logout/>}/>
              <Route path={"/model"} element={<ModelPage/>}/>
          </Route>
      </Route>
  </Routes>
</BrowserRouter>
```

Obrázek 3.3 Hierarchie stránek v Router kompnentě

3.1.5 Nasazení

Teď budu mluvit o tom, jak vypadá proces nasazení aplikace na server. K nasazování na server jsem použil Gitlab CI/CD a Docker. Samotný proces je tedy plně automatický a provede se vždy, když dojde k aktualizaci repozitáře. Aplikace s mockovaným back-endem je dostupná na adrese `http://oko.sic.cz`.

Docker

Abych mohl proces nasazení lépe popsat, musím nejdřív vysvětlit co to je Docker. Pomocí Dockeru se dají vytvářet takzvané kontejnery. Tyto kontejnery se chovají podobně jako virtuální stroje, nemají však takové nároky na výkon. Aplikaci můžu do tohoto kontejneru zabalit a zapnout jí. Aplikace pak bude žít v kontejneru stejně jako kdybych jí pustil přímo na svém zařízení. Výhodou takto zabalené aplikace je, že pokud na zařízení lze spustit Docker, lze spustit i kontejner s aplikací. Aplikace je tak snadno přenositelná a víceméně nezávislá na platformě.

Aby mohl Docker kontejner spustit, musí nejdřív znát jeho předpis. Tomuto předpisu se říka docker image. Image lze vytvořit pomocí takzvaného Dockerfile. Jde o soubor, ve kterém je napsané, jak se má image sestavit. Říká co je obsahem image a jak jí nastavit a spustit. Na obrázku 3.4 je vidět, jak vypadá dockerfile této aplikace.

Na prvním řádku je specifikovaná image, která funguje jako systém, na kterém se budou instalovat závislosti aplikace. Druhý až šestý řádek pak říká, že se mají zkopírovat soubory package.json a package-lock.json a pomocí příkazu npm install nainstalovat všechny potřebné závislosti. Na sedmém řádku je už příkaz pro sestavení aplikace.

Po tom co se aplikace sestaví, se spustí nový image, tentokrát webový server nginx. Tento image slouží jako prostředí, na kterém aplikace v kontejneru poběží. Do tohoto nového prostředí se nakopíruje aplikace, sestavená v předchozím image a soubor nginx.conf, ve kterém je definovaná configurace nginx serveru.

Řádky 21 až 25 slouží k tomu, abych mohl dovnitř kontejneru a do sestavené aplikace propagovat proměnné. Pomocí těchto proměnných lze nastavit adresu back-endu a klientské id pro přihlašování přes google identitu. Tato funkcionalita je postavená na scriptu env.sh. Tento skript přečte všechny proměnné v souboru .env a vytvoří z nich soubor env-config.js. Tento soubor přiřadí proměnné k objektu window._env_, ze kterého je může už aplikace číst. Pokud jsou ty

```
FROM node:19-alpine as builder
WORKDIR ./
COPY package*.json ./
RUN npm install

COPY . .
RUN npm run build

FROM nginx

RUN rm -r /usr/share/nginx/html/*

RUN rm /etc/nginx/conf.d/default.conf

COPY --from=builder ./build /usr/share/nginx/html/

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

WORKDIR /usr/share/nginx/html
COPY ./env.sh .env* ./
RUN test -e ./.env || $(touch .env && echo "API_URL=http://localhost:3001" > ./.env && echo "CLIENT_ID=undefined" >> ./.env)

RUN chmod +x env.sh

CMD ["/bin/bash", "-c", "/usr/share/nginx/html/env.sh && nginx -g \"daemon off;\""]

```

Obrázek 3.4 Dockerfile aplikace

samé proměnné navíc definované přímo v prostředí kontejneru, skript dá přednost jim a přepíše proměnné v souboru .env. Díky tomu lze proměnné nastavit při spouštění kontejneru.

Gitlab CI/CD

Gitlab kromě verzování a repozitáře nabízí i automatizaci nasazování aplikace pomocí takzvaných pipeline. Pipeliny jsou takové kroky, které se vykonají když proběhne nějaká změna v repozitáři a je splněná podmínka pro jejich spuštění. Aby gitlab věděl, jaké kroky provést, musí být popsány v souboru s názvem „.gitlab-ci.yml“. Můj .gitlab-ci.yml soubor definuje tyto kroky: lint, build, publish, publish-mock, deploy-dev, deploy-prod a deploy-back-mock. Všechny kroky se spouštějí na takzvaném gitlab runneru.

Gitlab runner je nějaký server, na který se gitlab připojí a krok tam vykoná ve svém docker kontejneru. V základu gitlab nabízí runner zadarmo, protože ale při nasazování pracuji s citlivými informacemi, jako runner využívám vlastní server poskytnutý vedoucím mé práce.

V každém kroku specifikuji v jakém prostředí (docker kontejneru) se bude vykonávat, skript který se v prostředí spustí a nakonec za jakých podmínek se má daný krok spustit.

První krok lint spustí statickou kontrolu kódu. Tento krok by mohl být odstraněn, protože se kontrola stejně provádí i v kroku build. Krok lint se ale narozdíl od kroku build spouští při každé změně repozitáře, jde tedy jen o preferenci jak často chcete kontrolu provádět.

Druhý krok build spustí sestavení aplikace. V tomto kroku ověřuji, jestli lze aplikaci sestavit. Sestavenou aplikaci lze pak stáhnout jako artefakt. Tento krok se spustí pouze v případě, že dojde ke změně ve složce src.

V třetím kroku sestavuji docker image aplikace. Image se sestavuje podle výše popsaného dockerfilu a poté se nahraje do docker repozitáře. Gitlab nabízí také svůj vlastní docker repozitář, ten ale není na školní verzi gitlabu spuštěný. Tento krok se stejně jako krok build spustí v případě, že dojde ke změnám ve složce src. Zároveň ale musí být změny na větvi development nebo master.

V kroku publish-mock sestavuji docker image mockovaného back-endu. Tato image je pak stejně jako image aplikace nahrána do docker repozitáře. Tento krok se spouští ve stejném případě jako krok předešlý, ale ke změně musí dojít ve složce mockoon ne src.

Dalším je krok deploy-back-mock. V tomto kroku dojde k přihlášení na server pomocí příkazu ssh a privátního klíče. Přítomnost privátního klíče je důvod, proč je potřeba využít vlastního gitlab runneru. Po přihlášení se stáhne docker image mockovaného back-endu z docker repozitáře a pomocí příkazu docker run spustí kontejner. Tento krok se spustí ve stejném případě jako krok předešlý.

Posledními dvěma kroky jsou deploy-dev a deploy-prod. Oba kroky spouští aplikaci na serveru, podobně jako krok deploy-back-mock spouští mockovaný back-end. Rozdíl mezi těmito kroky je v tom, na jakém portu aplikaci spustí. Deploy-dev se vykoná, pokud došlo ke změnám na větvi development a spustí server na protu 8080. Deploy-prod se vykoná, pokud došlo ke změnám na větvi master a spustí server na klasickém portu 80, který se normálně pro webové aplikace používá. Při spouštení kontejneru aplikace se taky specifikují proměnné API_URL a CLIENT_ID.

3.2 Instalační příručka

V této části podrobně popíšu jak nainstalovat a spustit server s aplikací na svém zařízení. Pro spuštění vývojářského serveru na zařízení linux stačí mít nainstalovaný node package manager. Potom už jen nainstalovat závislosti aplikace příkazem npm install (musíte se nacházet v kořenové složce projektu) a spustit jí příkazem npm start.

Instalace docker engine

Pro instalaci již sestavené aplikace je potřeba mít na zařízení sprovozněný docker engine (v době psaní využívám verzi 23.06, ale aplikace pravděpodobně půjde spustit i na novějších verzích). Na zařízení s operačním systémem Ubuntu můžete docker engine nainstalovat pomocí následujících příkazů:

1. sudo apt install ca-certificates curl gnupg
2. sudo install -m 0755 -d /etc/apt/keyrings
3. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg –dearmor -o /etc/apt/keyrings/docker.gpg
4. sudo chmod a+r /etc/apt/keyrings/docker.gpg
5. echo ”deb [arch=”$(dpkg –print-architecture)”signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu” $(. /etc/os-release && echo ”$VERSION_CODENAME”)”stable”| \ sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
6. sudo apt update
7. sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Podrobnější informace k instalaci najdete na stránce [49].

Image

Aby mohl Docker kontejner s aplikací spustit musí nedjříve vidět její image. Samotnou image aplikace si můžete stáhnout na adrese [50]. Jde o soubor s názvem „interactiveModelEye-frontend-image.tar“.

Aby docker image viděl, musíte jí přidat příkazem *docker load –input „path/to/image“*. Poté by se měla image objevit pod názvem eye-front:latest ve výpisu příkazu *docker images*. Podrobnější informace o příkazu docker load jsou v Docker dokumentaci na stránce [51].

Spuštění

Server spustíte příkazem *docker run -d -p 80:80 -e API_URL=”address of your back-end”-e CLIENT_ID=”google api client id”–name eye-front-server eye-front:latest*.

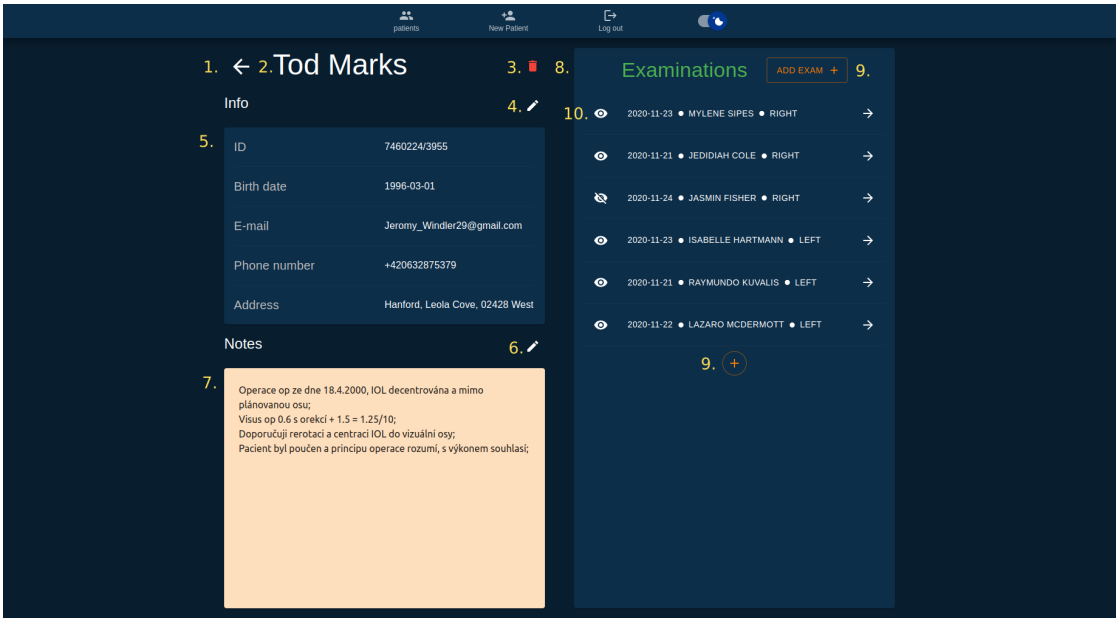
V příkazu je možné nespecifikovat parametr *API_URL*, potom bude aplikace stahovat data ze serveru s adresou *http://localhost:3001*. CLIENT_ID je povinný parametr a jedná se o klientské id od Google potřebné pro využití Google jako poskytovatele identity. Pokud parameter při spuštění nespecifikujete, nebude přihlášení přez Google účet fungovat. Více o clientId na stránce [52].

Po spuštění by se měl na adrese *http://localhost/login* objevit formulář pro přihlášení uživatele. Pro vypnutí serveru stačí zadat příkaz *docker stop eye-front-server*. Pokud budete chtít server opětovně zapnout můžete ho už spustit jednodušeji příkazem *docker start eye-front-server*.

3.3 Uživatelská příručka

V této částí ukážu a popíšu uživatelské rozhraní aplikace. Zde budu popisovat pouze stránky detail pacienta a detail měření. Celá uživatelská příručka obsahující popis všech stránek je v příloze.

Detail pacienta



Obrázek 3.5 Detail pacienta

Na této stránce se nacházejí veškeré informace o pacientovi. Obsah této stránky je vidět na obrázku 3.5. Jednotlivé prvky na této stránce jsou:

1. Tlačítko zpět. Po kliknutí vás přenese zpět na stránku s výběrem profilu.

2. Jméno pacienta.

3. Tlačítko vymazat. Slouží k vymazání tohoto profilu. Po kliknutí se objeví dialogové okno s prázdným polem a možnostmi Cancel a Ok. Možnost Cancel zruší mazání profilu. Možnost Ok potvrdí mazání profilu. Abyste mohli zvolit možnost Ok, musíte do pole vyplnit jméno pacienta.

4. Tlačítko upravit profil. Po kliknutí vás přenese na stránku, kde budete moct upravit informace o pacientovi.

5. Tabulka s informacemi o pacientovi

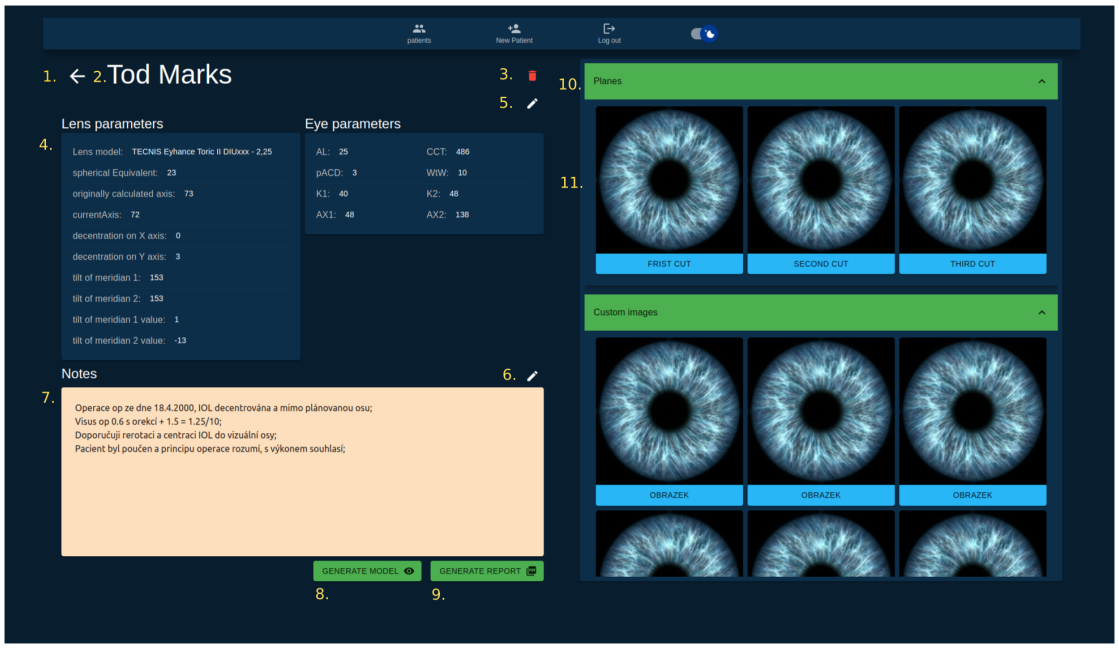
6. Tlačítko upravit poznámky. Po kliknutí vás přenese na stránkuk kde budete moct upravit pacientovi poznámky.

7. Poznámky k pacientovi.

8. Seznam pacientových vyšetření.

9. Tlačítko nové vyšetření. Po kliknutí vás přenese na stránku, na které budete moct vytvořit nové vyšetření k tomuto pacientovi.

10. Tlačítko vyšetření. Po kliknutí vás přenese na stránku s detail vyšetření.



Obrázek 3.6 Detail vyšetření

Detail vyšetření

Na této stránce se nacházejí veškeré informace o vyšetření. Obsah této stránky je vidět na obrázku 3.6.

Jednotlivé prvky na této stránce jsou:

1. Tlačítko zpět. Po kliknutí vás přenese zpět na stránku detail pacienta.
2. Jméno pacienta.
3. Tlačítko vymazat. Slouží k vymazání tohoto vyšetření. Po kliknutí se objeví dialogové okno s prázdným polem a možnostmi Cancel a Ok. Možnost Cancel zruší mazání vyšetření. Možnost Ok potvrdí mazání vyšetření. Abyste mohli zvolit možnost Ok, musíte do pole vyplnit jméno pacienta.
4. Tabulka s parametry vyšetření.
5. Tlačítko upravit vyšetření. Po kliknutí vás přenese na stránku kde budete moct upravit parametry vyšetření.
6. Tlačítko upravit poznámky. Po kliknutí vás přenese na stránku kde budete moct upravit poznámky k vyšetření.
7. Poznámky k vyšetření.
8. Tlačítko vygenerovat model. Po kliknutí vás přenese na stránku s vygenerovaným modelem.
9. Tlačítko vygenerovat Report. Po kliknutí aplikace vygeneruje model a stáhne ho do počítače do složky pro stahování (nastavení prohlížeče).
10. Tlačítko s názvem sekce galerie. Po kliknutí se tato sekce zavře, nebo otevře.
11. Obrázek. Po kliknutí se obrázek zvětší.

Kapitola 4
Testování

V této kapitole se zaměřím na testování aplikace. Hotový prototyp podrobým uživatelskému testování. Cílem tohoto testování je odkrýt problémy designu, které by mohly koncovým uživatelům stěžovat používání aplikace [53]. Aplikaci jsem testoval na očkní klinice v Praze, kde pracuje zadavatel projektu.

4.1 Testovací scénáře

Abych mohl testování provést musím zhotovit vstupní a výstupní dotazník a takzvané testovací scénáře. Pomocí těchto scénářů má vedoucí testování instruovat uživatele, ten se poté bude snažit instrukce splnit. Zde uvedu pouze scénáře a u dotazníků pouze uvedu, jaké otázky obsahují. Oba dotazníky jsem nakonec zkombinoval do jednoho, aby se mi lépe tisknul. Dotazník pro tisk se nachází v příloze.

4.2 Úvod

Aplikace Interaktivního modelu oka má být využita při operaci šedého zákalu jako pomoc při orientaci nitrooční čočky v pacientově oku.

Součástí této aplikace je i systém pro správu pacientů a jejich předoperačních vyšetření. Tato část aplikace bude předmětem tohoto testování.

Moderátor testování bude nahlas předčítat instrukce. Vaším úkolem je pokusit se je splnit. Pokud to bude možné, říkejte nahlas co si myslíte. Instrukce nezkouší vás a vaší schopnost úkoly splnit, ale testuje implementaci aplikace a její funkcionality.

4.3 Scénář 1

Cílem tohot scénáře je vyzkoušet proces vytváření nového profilu, nového měření a přihlášení uživatele do aplikace pomocí e-mailu a hesla.

4.3.1 Přihlášení

Zaměření scénáře

Otestování funkčnosti přihlášení uživatele pomocí e-mailu a hesla.

Výchozí stav

Aplikace na stránce /login zobrazující přihlašovací formulář. Uživatel nebyl předtím přihlášen.

Finální stav

Uživatel je úspěšně přihlášen a nachází se na stránce /patients. Aplikace úspěšně stáhla všechna data a je vidět seznam profilů.

Instrukce pro testera

1. Přihlašete se pomocí e-mailu „right@example.com“ a hesla „123“.

Očekávané kroky

1. Zadání e-mailu do pole s názvem e-mail.
2. Zadání hesla do pole s názvem password.
3. Kliknutí na tlačítko Log in, nebo stisknutí klávesy enter.

4.3.2 Vytvoření nového profilu

Zaměření scénáře

Otestování funkčnosti vytvoření nového profilu.

Výchozí stav

Aplikace na stránce /patients.

Finální stav

Uživatel úspěšně vytvořil nového pacienta a nachází se na stránce /patients/id. Na této stránce se zobrazují informace zadané při vytváření profilu.

Instrukce pro testera

1. Na stránce najděte tlačítko New patient.
2. Klikněte na něj.
3. Zkontrolujte, že se zobrazil formulář.
4. Do formuláře vyplňte následující informace:
   - Jméno: Petr
   - Příjmení: Nový
   - Datum narození: 6.3.1990
   - Rodné číslo: 9900306/4567
   - Telefoní číslo: +420702800400
   - E-mail: petr.novy@gmail.com
   - Adresa: Zlín, Cimrmanova 14, 760 01
5. Odešlete formulář.
6. Zkontrolujte, jestli se informace na stránce shodují s informacemi zadávanými do formuláře.

Očekávané kroky

1. Kliknutí na tlačítko New patient.
2. Zadání údajů do správných polí formuláře.
3. Kliknutí na tlačítko submit nebo zmáčknutí klávesy enter.

4.3.3 Vytvoření nového měření

Zaměření scénáře

Otestování funkčnosti vytvoření nového měření.

Výchozí stav

Aplikace na stránce /patients/id.

Finální stav

Uživatel úspěšně vytvořil nové měření a nachází se na stránce /patients/id/examinations/id. Na této stránce se zobrazují informace zadané při vytváření měření.

Instrukce pro testera

1. Na stránce najděte tlačítko Add exam a klikněte na něj.
2. Do formuláře vyplňte následující informace:
   - model čočky: enVista Toric MX60T125
   - spherical equivalent: 17,5
   - Originally calculated axis: 75
   - Current Axis: 83
   - Decentration on X: 0,5
   - Decentration on Y: 0,5
   - Tilt1 meridian: 180
   - Tilt1 value: 4,5
   - Tilt2 meridian: 90
   - Tilt2 value: 1
   - AL: 25,64
   - CCT: 534
   - WtW: 12,27
   - pACD: 4,75
   - K1: 40,38
   - AX1: 171
   - K2: 41,99
   - AX2: 81
3. Odešlete formulář.
4. Zkontrolujte, jestli se informace na stránce shodují s informacemi zadávanými do formuláře.

Očekávané kroky

1. Kliknutí na tlačítko Add examination nebo tlačítko plus pod seznamem.
2. Zadání údajů do správných polí formuláře.
3. Kliknutí na tlačítko submit nebo zmáčknutí klávesy enter.

4.3.4 Vygenerování reportu a stránka model

Zaměření scénáře

Otestování funkčnosti generování reportu a přepínání na stránku s modelem.

Výchozí stav

Aplikace na stránce /patients/id/examinations/id.

Finální stav

Uživatel úspěšně vygeneroval a stáhl report a nachází se na stránce /model.

Instrukce pro testera

1. Na stránce najděte tlačíto generate report a klikněte na něj.
2. Zkontrolujte, že se report stáhl a otevřete ho.
3. Na stránce najděte tlačítko generate model a klikněte na něj.

Očekávané kroky

1. Kliknutí na tlačítko Generate report.
2. Otevření staženého souboru přez stažené soubory v prohlížeči.
3. Kliknutí na tlačítko Generate model.

4.4 Scénář 2

Cílem tohoto scénáře je otestovat úpravu profilu a měření a to včetně jejich mazání. Zároveň testuje přihlašování přez Google identitu.

4.4.1 Přihlášení

Zaměření scénáře

Otestování funkčnosti přihlášení uživatele pomocí e-mailu a hesla.

Výchozí stav

Aplikace na stránce /login zobrazující přihlašovací formulář. Uživatel nebyl předtím přihlášen.

Finální stav

Uživatel je úspěšně přihlášen a nachází se na stránce /patients. Aplikace úspěšně stáhla všechna data a je vidět seznam profilů.

Instrukce pro testera

1. Přihlašete se pomocí e-mailu „right@example.com“ a hesla „123“.

Očekávané kroky

1. Zadání e-mailu do pole s názvem e-mail.
2. Zadání hesla do pole s názvem password.
3. Kliknutí na tlačítko Log in, nebo stisknutí klávesy enter.

4.4.2 Vyhledání a zvolení profilu pacienta

Zaměření scénáře

Otestování funkčnosti vyhledávání podle jména nebo rodného čísla a odkazů na profil.

Výchozí stav

Aplikace na stránce /patients zobrazující seznam profilů.

Finální stav

Uživatel úspěšně vyhledal profil pacienta pomocí rodného čísla, nebo jména a nachází se na stránce /patients/:id.

Instrukce pro testera

1. Na stránce najděte vyhledávací pole.
2. Vyhledejte profil pacienta jménem „Robert Malý“ s rodným číslem „0001232/1128“.
3. Klikněte vyhledanou položku.

Očekávané kroky

1. Kliknutí do vyhledávacího pole.
2. Zadání pacientova rodného čísla nebo jména do vyhledávacího pole.
3. Kliknutí na položku ze seznamu, která představuje profil pacienta.

4.4.3 Úprava profilu

Zaměření scénáře

Otestování funkčnosti úpravy profilu.

Výchozí stav

Aplikace na stránce /patients/id.

Finální stav

Uživatel úspěšně upravil pacientův profil a nachází se na stránce /patients/id. Na této stránce jsou vidět změny provedené úpravou.

Instrukce pro testera

1. Najděte na stránce tlačítko pro úpravu profilu.
2. Ve formuláři změňte pacientovu adresu na Olomouc, Hlavní 55, 779 01.
3. Odešlete formulář.
4. Zkontrolujte, že byl profil změněn.

Očekávané kroky

1. Kliknutí na tlačítko tužky nad tabulkou s informacemi o pacientovi.
2. Přepsání informací v polích City, Street, House number, Post code.
3. Kliknutí na tlačítko Submit nebo zmáčknutí klávesy enter.
4. Prohlédnutí tabulky s informacemi o pacientovi.

4.4.4 Úprava poznámek

Zaměření scénáře

Otestování funkčnost tvorby a úpravy poznámek. Otestování poznámkových nástrojů: List, Bold.

Výchozí stav

Aplikace na stránce /patients/id, resp. /patients/id/examinations/exid.

Finální stav

Uživatel úspěšně upravil pacientovy poznámky a nachází se na stránce /patients/id resp. /patients/id/examinations/exid. Na této stránce jsou vidět změny provedené úpravou.

Instrukce pro testera

1. Najděte na stránce tlačítko pro úpravu poznámek.
2. Do poznámek napište: „Pacient byl poučen.“ a tučně zvýrazněte slovo „byl“.
3. Na dalším řádku vytvořte seznam s položkami: „položka jedna“, „položka dvě“, „položka tři“.
4. Poznámky uložte.
5. Zkontrolujte, že se poznámky úspěšně uložili.

Očekávané kroky

1. Kliknutí na tlačítko tužky nad poznámkami.
2. Napsání textu dle zadání.
3. Kliknutí na tlačítko Submit nebo zmáčknutí klávesy enter.
4. Prohlédnutí poznámek zobrazených na stránce. Výsledný text by měl vypadat takto:
   „ Pacient **byl** poučen.
   - položka jedna
   - položka dvě
   - položka tři

   “

4.4.5 Zvolení a úprava měření

Zaměření scénáře

Otestovat přehlednost seznamu měření a funkčnost úpravy parametrů měření.

Výchozí stav

Aplikace na stránce /patients/id/examinations/id.

Finální stav

Uživatel se úspěšně dostal na stránku s měřením a provedl úpravu parametrů.

Instrukce pro testera

1. Najděte v seznamu měření, měření které bylo dnes vytvořené (dnešní datum).
2. Na stránce měření najděte tlačítko pro úpravu parametrů a zmáčkněte ho.
3. Změňte model čočky na model TECNIS a model cylindru na 300.
4. Potvrďte provedené úpravy.
5. V tabulce s parametry zkontrolujte, že byly změněny.

Očekávané kroky

1. Vyhledání měření v pravém seznamu podle data vytvoření.
2. Kliknutí na tlačítko tužky nad tabulkou s parametry.
3. Změnění údajů ve formuláři.
4. Kliknutí na tlačítko submit nebo stisknutí klávesy enter.

4.4.6 Smazání měření

Zaměření scénáře

Otestování funkčnosti smazání měření.

Výchozí stav

Aplikace na stránce /patients/id/examinations/id.

Finální stav

Uživatel úspěšně smazal pacientův profil a nachází se na stránce /patients/id. V seznamu měření chybí smazané měření.

Instrukce pro testera

1. Najděte na stránce tlačítko pro vymazání měření a klikněte na něj.
2. Držte se pokynů v novém dialogovém okně.
3. Klikněte na tlačítko Ok.
4. Zkontrolujte, že bylo měření smazáno.

Očekávané kroky

1. Kliknutí na tlačítko popelnice.
2. Vyplnění pacientova jména v dialogovém okně.
3. Kliknutí na tlačítko Ok.
4. Prohlédnutí seznamu.

4.4.7 Smazání profilu

Zaměření scénáře

Otestování funkčnosti smazání profilu.

Výchozí stav

Aplikace na stránce /patients/id.

Finální stav

Uživatel úspěšně smazal pacientův profil a nachází se na stránce /patients. V seznamu profilů chybí profil smazaného pacienta.

Instrukce pro testera

1. Najděte na stránce tlačítko pro vymazání profilu a klikněte na něj.
2. Držte se pokynů v novém dialogovém okně.
3. Klikněte na tlačítko Ok.
4. Zkontrolujte, že byl profil smazán.

Očekávané kroky

1. Kliknutí na tlačítko popelnice.
2. Vyplnění pacientova jména v dialogovém okně.
3. Kliknutí na tlačítko Ok.
4. Prohlédnutí seznamu, případně pokus o vyhledání profilu pomocí vyhledávacího pole.

4.5 Vstupní dotazník

Tento dotazník slouží k získání informací o testerovi, ke keterým se pak přihlíží při anlaýze sesbíraných dat při testovaní. Dotazník se ptá na: pohlaví, věk, zaměstnání, předchozí zkušenost s desktopovými a webovými aplikacemi.

4.6 Výstupní dotazník

Tento dotazník vyplňuje tester po skončení testování. Má za úkol sesbírat informace o tom, jaký má tester z aplikace pocit a jeho připomínky a poznámky k implementaci. Otázky v dotazníku:

- Měli jste problém nalézt nějaký funkční prvek (jako třeba tlačítko)? Pokud ano napište jaký.
- Byli jste spokojeni s barevným schématem aplikace a kontrastem jednotlivých prvků? Pokud ne, jaké barevné schéma byste zvolili a které prvky by mohly být kontrastnější nebo naopak.
- Chyběla vám při testování nějaká funkcionalita (jako třeba možnost uložit rozpracovanou část formuláře, aniž bych skočil na další stránku)? Pokud ano popište jí.
- Přišlo vám rozvržení informací na stránkách logické? Dalo se v informacích na stránce zorientovat?
- Přidali byste nějakou funkcionalitu do horní navigace? Pokud ano napište jakou.

4.7 Průběh testování

Jak jsem říkal v úvodu testování proběhlo na oční klinice v Praze. Celkově aplikaci testovalo pět lidí, z toho dvě sálové sestry, jedna ambulatní sestra a dvě doktorky. Protože není jednoduché zaměstnat tolik lidí ze zdravotnictví nepracovní aktivitou, museli mi věnovat svůj čas z obědové pauzy. I tak testování probíhalo relativně v rozumném tempu. Testování probíhalo v kuchyňce kliniky na mém notebooku. Z každého testování jsem pořídil záznam obrazovky. Poslední záznam se ale bohužel poškodil, protože jsem se z kuchyňky musel přesunout, aby se zaměstnatnci kliniky mohli najíst a já zavřel notebook, aniž bych zastavil nahrávání.

Po dokončení obou scénařů jsem vždy testera vyzpovídal na základě otázek z výstupního dotazníku. Do dotazníku jsem za testera psal já a dělal si do něj poznámky během testování, protože jsem potřeboval uspořit čas, abych testování stihl.

Věk testerů se pohyboval od třiceti do třicetidevíti let, kromě Paní doktorky které je už sedmdesátpět. Téměř všichni testeři měli spíše malé zkušenosti s počítačem a aplikacemi, což se na testování ukazovalo. Testování také stěžovala skutečnost, že někteří testeři neuměli anglicky.

4.8 Zhodnocení

Celkově byli všichni testeři s uživatelským rozhraním spokojení. Všem vyhovovalo barevné schéma aplikace a možnost přepnutí z tmavého na světlý mód. Také byli spokojení s umístěním prvků na stránce a s jejich kontrastem. Přesto se při testování podařilo odhalit nějaké problémy.

Nejvíce problémů bylo s formulářem pacienta. Největší problém bylo pole s rodným číslem. V současné době se do pole musí číslo zadat i s lomítkem. Testeři na lomítko často zapomínali, nebo ho nemohli na klávesnici najít. Bylo by tedy lepší, aby se do pole lomítko automaticky doplňovalo. Podobný problém byl s telefoním číslem a předvolbou. Navrhovaná změna byla dát pevnou předvolbu a nebo výběr z předvolby české a slovenské.

Následně byl ještě problém s datumem a adresou. Pole pro zadávání datumu bylo rohraním v pořádku, testery ale mátl anglický formát datumu. Proto by bylo lepší změnit formát datumu na český. U adresy by bylo dobré sloučit pole s názvem a číslem ulice. Paní doktorka pak měla ještě poznámku, že bych měl k adrese přidat pole pro orientační číslo.

Další větší problém byl s vyhledáváním pacientů. Situace vypadala následovně. Tester se snažil vyhledat pacienta, který ještě nebyl načtený. Aplikace zobrazila text, který oznamoval, že hledání nemá žádné výsledky. Testeři pak nabyli dojmu, že hledaný pacient neexistuje. Tento problém vznikl hlavně kvůli tomu, že nemůžu použít vyhledávání na back-endu, protože to ten mockovaný neumí. Normální řešení by bylo, poslat dotaz na back-end s query parameterm se vstupem z vyhledávacího pole. Back-end by pak poslal jen relevantní výsledky. Finální řešení tedy bude vypadat tak, že uživatel něco zadá do vyhledávacího pole, front-end odešle požadavek na back-end a ten vrátí všechny relevantní pacienty, které se uživateli zobrazí.

Nakonec ještě uvedu nějaké poznámky k funkcionalitám, které jsem dostal ve výstupním dotazníku.

- Pacienti by mohli být rozlišení barvami podle pohlaví. To má přispět k rychlejšímu vyhledání pacienta.
- Porovnání měření mezi sebou. Možnost porovnat hodnoty a obrázky dvou měření.
- Řazení měření. Možnost řadit měření podle datumu. To je zase lepší řešit již při dotazování do databáze na back-endu.
- Není potřeba při mazání pacienta a měření zadávat jméno pacienta. Podle většiny testerů jde o příliš velké zabezpečení proti nechtěnému smazaní a stačilo by větší dialogové okno s více informacemi o pacienotvi nebo měření. Případně by mohla být dvě dialogová okna zasebou.
- Nejasnost jestli mažu pacienta nebo měření. Jeden tester měl problém odlišit situaci, kdy maže měření a kdy profil pacienta. To se dá vyřešit upřesněním v dialogovém okně.
- Ve výpisu informací o měření není vidět o jaké oko se jedná.
- U zadávání desetinných čísel by mělo být možné použít tečku i čárku.
- Report by se měl uložit pod jménem pacienta a datumem měření.

Kapitola 5
Závěr

Cílem práce bylo analyzovat současný stav front-endu projektu interkativního modelu oka pro vizualizaci nitrooční čočky, zhotovit prototyp front-endu a podrobit ho uživatelskému testování.

Analýzu současného stavu projketu jsem provedl v první části práce, kde jsem zhodnotil kód a technologie použité při dosavadním vývoji. Rozebral jsem možné alternativy technologií pro vývoj webových aplikací a vybral jsem vhodné pro potřeby projektu. Poté jsem vypsal funkční a nefunkční požadavky na celou aplikaci a vybral ty které se týkají front-endové části.

Na základě vybraných technologií a požadavků jsem provedl návrh aplikace. Konkrétně jsem vybral její architekturu, vypracoval diagram domény a workflow diagramy a navrhl její uživatelské rozhraní v podobě wireframů.

Následně jsem s pomocí návrhu a vybraných technologí provedl implementaci prototypu. Vytvořil jsem webovou aplikaci stojící na frameworku React a knihovně Redux. Oproti původnímu prototypu jsem změnil a přidal následující: generování PDF reportů, galerii obrázků, kompletně celý design a vzhled aplikace. Moje aplikace nabízí celkově lepší uživatelskou zkušenost z jejího používání.

Kvůli absenci back-endu jsem v rámci implementace vytvořil také mockovaný back-end, který se dá použít pro testování a vývoj v offline prostředí.

Hlavní přínos mojí implementace se ale skrývá pod povrchem a jde o zlepšení celkové kvality kódu. Moje práce dobře poslouží jako základ pro další úpravy a rozšíření, které projekt do budoucna jistě bude mít. V současné podobě prototyp splňuje všechny požadavky, ale není napojen na ostatní části projektu, které ale zatím ani nebyly naimplementovány. Data a prostředí jsou tedy zatím čistě testovací. Aplikaci jsem ale připravil tak, aby spojení s ostatními částmi a přechod do produkčního prostředí byl co nejjednodušší.

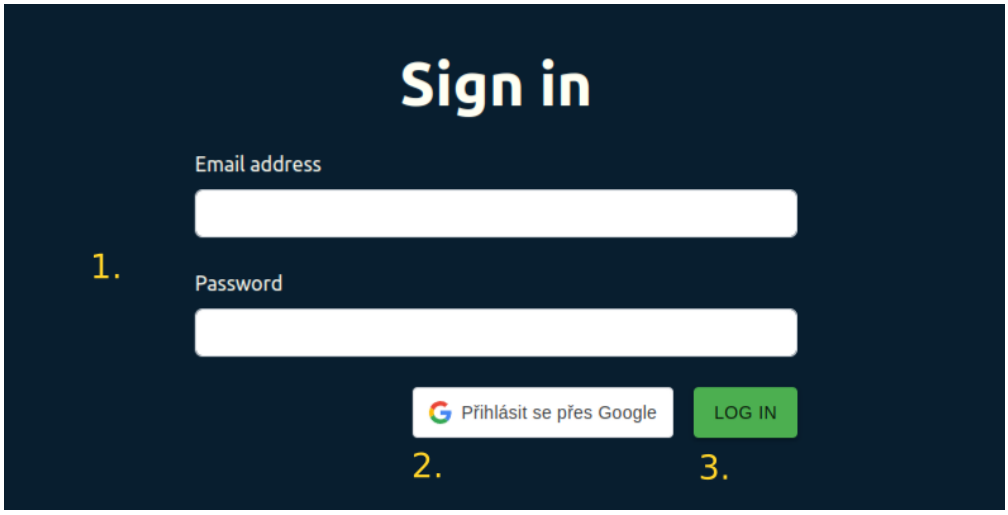
Nakonec jsem provedl uživatelksé testování, které přineslo užitečné poznámky pro vývoj dalších iterací tohoto projektu. Testování neodhalilo žádné vážné nedostatky, ale navrhlo spoustu drobných změn a úprav pro zlepšení kvality uživatelské zkušenosti. Také vyplynuly návrhy na pár dalších funkcionalit, které by aplikace v budoucnu mohla mít.

Příloha A

Uživatelská příručka

Tato přiručka má pomoct uživateli zorientovat se v uživatelském rozhraní aplikace. Konkrétně popisuje jednotlivé funkční prvky a části každé stránky.

A.1 Přihlašovací stránka



Obrázek A.1 Přihlašovací stránka

Tato stránka slouží k přihlášení uživatele. Obsah stránky je vidět na obrázku A.1. Nachází se zde přihlašovací formulář (1) a dvě tlačítka (2, 3).

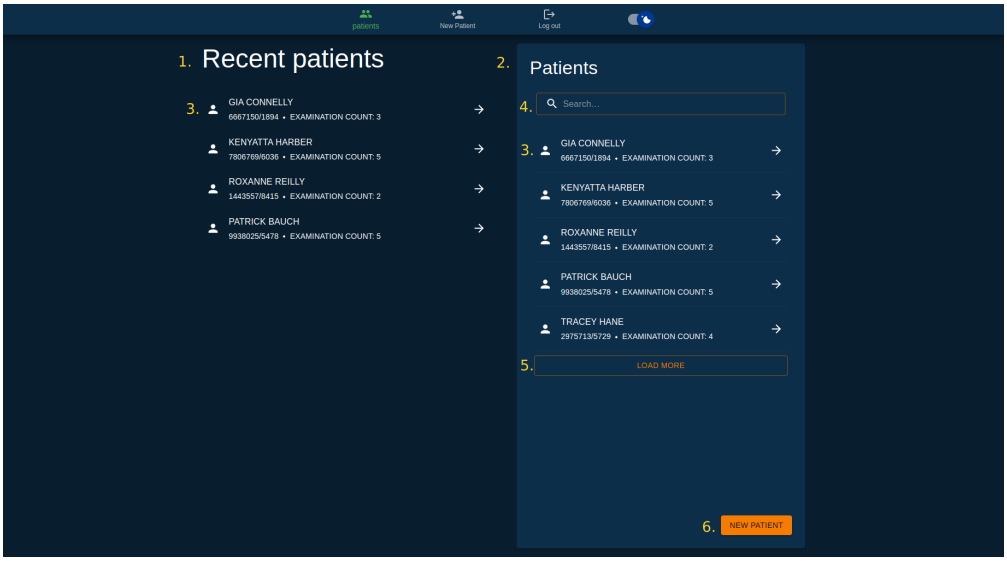
Po vyplnění formuláře ho můžete odeslat tlačítkem Log in (3). Pokud jsou zadané údaje platné dojde k přihlášení a přesměrování na stránku s výběrem pacienta. Pokud platné nejsou zobrazí se u formuláře hlášení o chybě „Invalid email or password“.

Tlačítko Login with Google (2) slouží k přihlášení svým Google účtem. Po jeho zmáčknutí se objeví vyskakovací okno s možnostmi přihlášení.

Pokud se vyskakovací okno neoběví, zkontrolujte jestli máte v prohlížeči povolené cookies a nebo nemáte zapnutou službu adblock, poté akci opakujte.

Po úspěšném přihlášení na váš Google účet dojde k přesměrování na stránku s výběrem pacienta.

A.2 Výběr pacienta



Obrázek A.2 Stránka s výběrem pacienta

Tato stránka slouží pro vyhledání pacientova profilu. Stránka je vidět na obrázku A.2.

Na levé straně (1) se nacházejí profily, které byly nedávno vyhledávané přihlášeným uživatelem.

Po načtení stránky se na pravé straně (2) zobrazí prvních deset profilů. Pomocí tlačítka Load more (5) můžete načíst další profily pacientů.

Nad tímto seznamem je vyhledávací pole (4), které slouží pro filtrování profilů. Profily lze filtrovat podle jména pacientů a podle jejich rodného čísla.

Pod seznamem se nachází tlačítko New patient (6). Toto tlačítko vás přesměruje na stránku pro tvorbu nového profilu pacienta.

Po Kliknutí na tlačítko s pacientem (3) vás aplikace přesměruje na stránku detail pacienta.

A.3 Nový profil

Tato stránka slouží k tovrbě nového profilu pacienta. Nachází se na ní formulář s informacemi o pacientovi a dvě tlačítka. Políčka name a surname jsou povinná ostatní můžou zůstat prázdná.

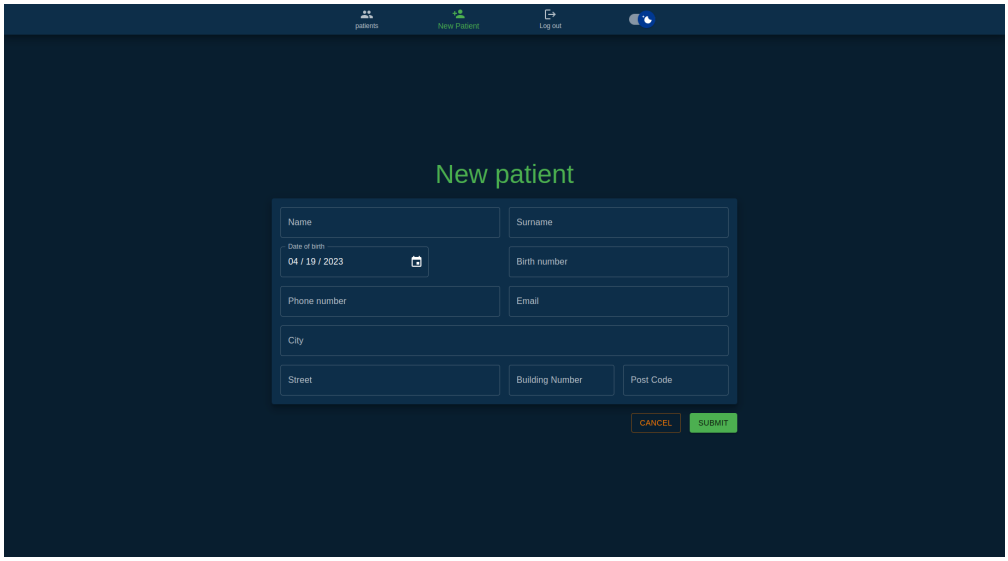
Tlačítkem submit odešlete formulář a vytvoříte nového pacienta. Pokud nejsou zadané informace validní (špatný formát), zobrazí se u invalidních políček text s chybou.

Po zmáčknutí tlačítka Cancel nedojde k vytvoření nového profilu a aplikace vás přeměruje na stránku s výběreme pacienta.

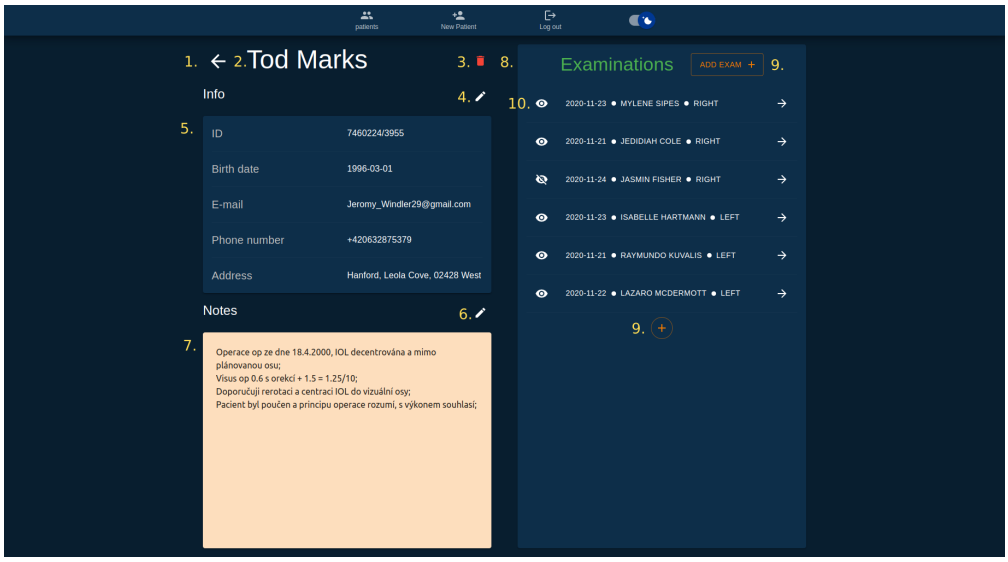
A.4 Detail pacienta

Na této stránce se nacházejí veškeré informace o pacientovi. Obsah této stránky je vidět na obrázku A.4. Jednotlivé prvky na této stránce jsou:

1. Tlačítko zpět. Po kliknutí vás přenese zpět na stránku s výběrem profilu.
2. Jméno pacienta.



Obrázek A.3 Stránka nový profil



Obrázek A.4 Detail pacienta

3. Tlačítko vymazat. Slouží k vymazání tohoto profilu. Po kliknutí se objeví dialogové okno s prázdným polem a možnostmi Cancel a Ok. Možnost Cancel zruší mazání profilu. Možnost Ok potvrdí mazání profilu. Abyste mohli zvolit možnost Ok, musíte do pole vyplnit jméno pacienta.

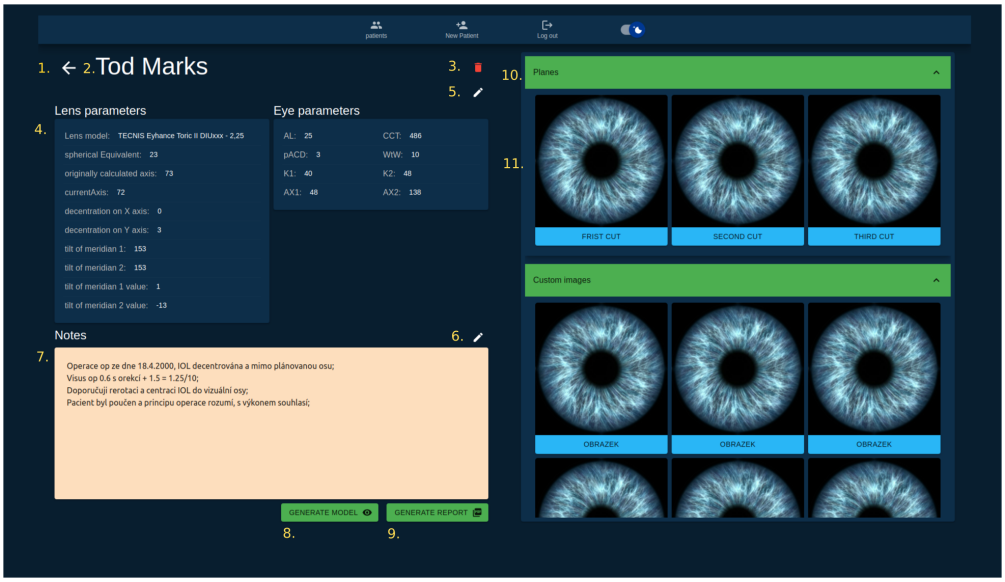
4. Tlačítko upravit profil. Po kliknutí vás přenese na stránku kde budete moct upravit informace o pacientovi.

5. Tabulka s informacemi o pacientovi

6. Tlačítko upravit poznámky. Po kliknutí vás přenese na stránku kde budete moct upravit pacientovi poznámky.

7. Poznámky k pacientovi.
8. Seznam pacientových vyšetření.
9. Tlačítko nové vyšetření. Po kliknutí vás přenese na stránku, na které budete moct vytvořit nové vyšetření k tomuto pacientovi.
10. Tlačítko vyšetření. Po kliknutí vás přenese na stránku s detail vyšetření.

A.5 Detail vyšetření



Obrázek A.5 Detail vyšetření

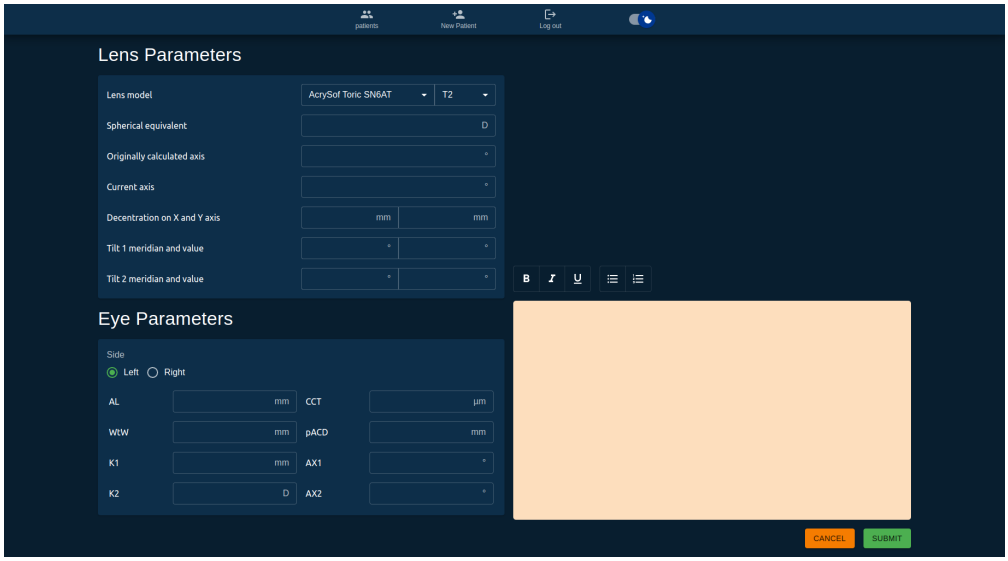
Na této stránce se nacházejí veškeré informace o vyšetření. Obsah této stránky je vidět na obrázku A.5.

Jednotlivé prvky na této stránce jsou:

1. Tlačítko zpět. Po kliknutí vás přenese zpět na stránku detail pacienta.
2. Jméno pacienta.
3. Tlačítko vymazat. Slouží k vymazání tohoto vyšetření. Po kliknutí se objeví dialogové okno s prázdným polem a možnostmi Cancel a Ok. Možnost Cancel zruší mazání vyšetření. Možnost Ok potvrdí mazání vyšetření. Abyste mohli zvolit možnost Ok, musíte do pole vyplnit jméno pacienta.
4. Tabulka s parametry vyšetření.
5. Tlačítko upravit vyšetření. Po kliknutí vás přenese na stránku kde budete moct upravit parametry vyšetření.
6. Tlačítko upravit poznámky. Po kliknutí vás přenese na stránku kde budete moct upravit poznámky k vyšetření.
7. Poznámky k vyšetření.

8. Tlačítko vygenerovat model. Po kliknutí vás přenese na stránku s vygenerovaným modelem.
9. Tlačítko vygenerovat Report. Po kliknutí aplikace vygeneruje model a stáhne ho do počítače do složky pro stahování (nastavení prohlížeče).
10. Tlačítko s názvem sekce galerie. Po kliknutí se tato sekce zavře, nebo otevře.
11. Obrázek. Po kliknutí se obrázek zvětší.

A.6 Nové vyšetření



Obrázek A.6 Nové vyšetření

Tato stránka slouží pro vytvoření nového vyšetření. Nachází se na ní formulář s parametry měření, formulář na poznámky k měření a dvě tlačítka. Obsah stránky je vidět na obrázku A.6.

Tlačítkem submit odešlete formulář a vytvoříte nové měření. Pokud nejsou zadané informace validní (špatný formát), zobrazí se u invalidních políček text s chybou.

Po zmáčknutí tlačítka Cancel nedojde k vytvoření nového vyšetření a aplikace vás přeměruje na stránku detail pacienta.

A.7 Horní navigace

Obrázek A.7 Horní navigace

Tato navigace se zobrazuje na každé stránce kromě přihlašovací a na stránce s modelem. Na liště jsou čtyři tlačítka. Jsou to zleva:

- Patients. Po zmáčknutí vás přenese na stránku s výběrem pacienta.
- New Patient. Po zmáčknutí vás přenese na stránku pro vytvoření nového pacienta.

- Log out. Po zmáčknutí vás odhlásí a přesměruje na stránku přihlášení
- Dark/light mode. Po zmáčknutí přepne na druhý barevný mód (tmavý, světlý)

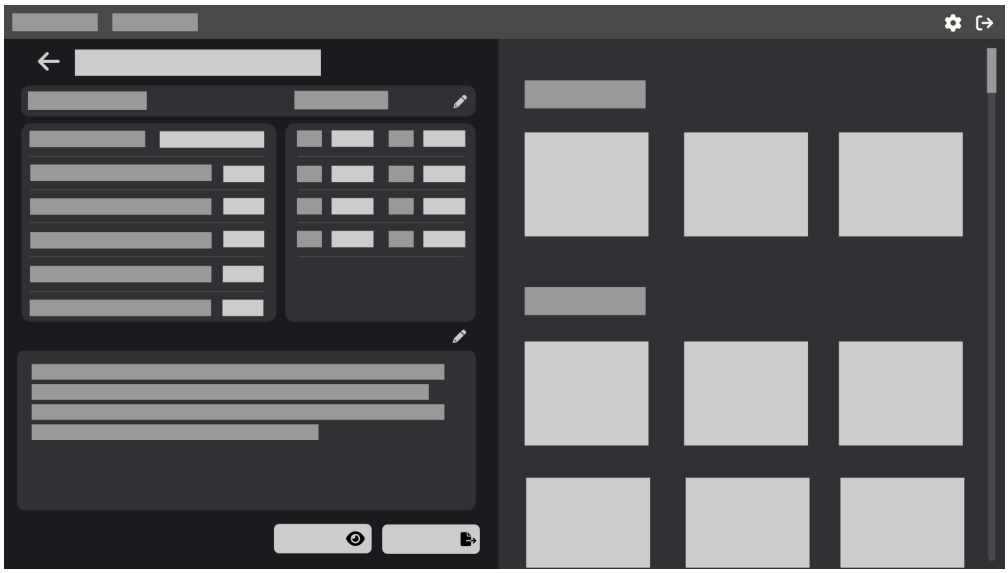
Příloha B

Wireframy

Obrázek B.1 Wireframe přihlašovací stránky

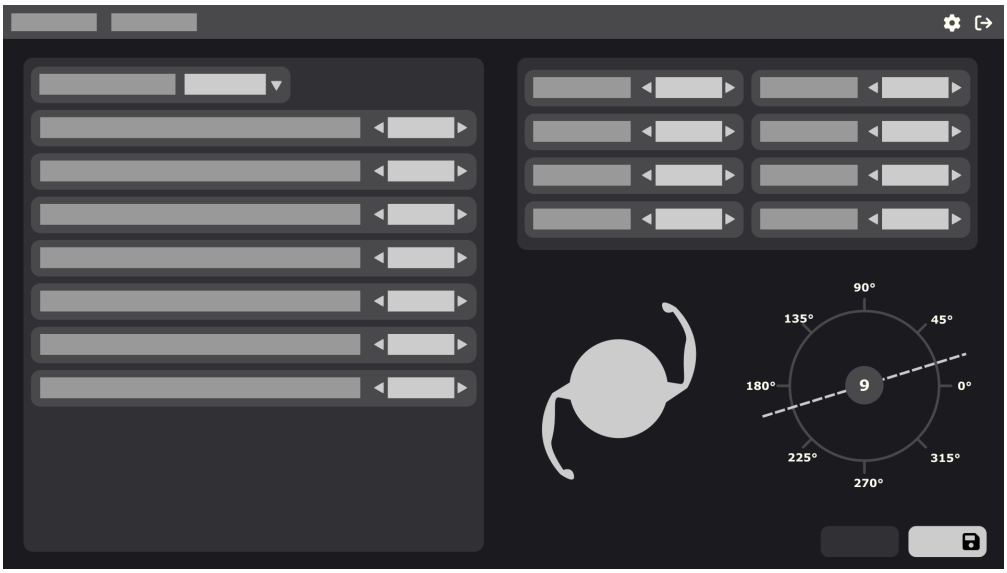
Obrázek B.2 Wireframe stránky s výběrem pacienta

Obrázek B.3 Wireframe stránky s detailem pacienta



Obrázek B.4 Wireframe stránky s detailem měření

Obrázek B.5 Wireframe stránky s úpravou poznámek



Obrázek B.6 Wireframe stránky s úpravou měření

Obrázek B.7 Wireframe stránky s úpravou pacienta

Příloha C

Dotazník

Pohlaví:
Věk:
Zaměstnání:
Zkušenost s desktopovými a webovými aplikacemi:
Měli jste problém nalézt nějaký funkční prvek (jako třeba tlačítko)? Pokud ano napište jaký.
Byli jste spokojeni s barevným schématem aplikace a kontrastem jednotlivých prvků? Pokud ne, jaké barevné schéma byste zvolili a které prvky by mohly být výraznější nebo naopak méně?
Chyběla vám při testování nějaká funkcionalita (jako třeba možnost uložit rozpracovanou část formuláře aniž bych skočil na další stránku)? Pokud ano popište jí.
Přišlo vám rozvržení informací na strákách logické? Dalo se v informacích na stránce zorientovat?
Přidali byste nějakou funkcionalitu do horní navigace? Pokud ano napište jakou.

Obrázek C.1 Vstupní a výstupní dotazník pro testování – tisk

Bibliografie

1. *React* [online]. Meta open source, 2023. [cit. 2023-04-21]. Dostupné z: `https://react.dev/`.
2. GREIF, Sacha. *State of JS Front-end framewroks* [online]. 2022. [cit. 2023-04-21]. Dostupné z: `https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/`.
3. MFUJI09. *SPA (Single page application)* [online]. 2023. [cit. 2023-04-21]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Glossary/SPA`.
4. *Writing markup with JSX* [online]. Meta open source, 2023. [cit. 2023-04-21]. Dostupné z: `https://react.dev/learn/writing-markup-with-jsx`.
5. *Learn React* [online]. Meta open source, 2023. [cit. 2023-04-21]. Dostupné z: `https://react.dev/learn`.
6. *React API reference* [online]. Meta open source, 2023. [cit. 2023-04-21]. Dostupné z: `https://react.dev/reference/react`.
7. *React API reference* [online]. GitHub, Inc., 2023. [cit. 2023-04-21]. Dostupné z: `https://github.com/facebook/react/releases`.
8. *React version upgrade guide* [online]. Meta open source, 2023. [cit. 2023-04-21]. Dostupné z: `https://react.dev/blog/2022/03/08/react-18-upgrade-guide`.
9. *React npm* [online]. npm, Inc. [cit. 2023-04-21]. Dostupné z: `https://www.npmjs.com/package/react`.
10. *React router* [online]. Remix Software, Inc. [cit. 2023-04-21]. Dostupné z: `https://reactrouter.com/en/main`.
11. *What is Angular* [online]. Google, 2010. [cit. 2023-04-21]. Dostupné z: `https://angular.io/guide/what-is-angular`.
12. *Angular documentation* [online]. Google, 2010. [cit. 2023-04-21]. Dostupné z: `https://angular.io/docs`.
13. *Angular releases* [online]. Google, 2010. [cit. 2023-04-21]. Dostupné z: `https://angular.io/guide/releases`.
14. *Angular npm* [online]. npm, Inc. [cit. 2023-04-21]. Dostupné z: `https://www.npmjs.com/package/@angular/cli`.
15. *Vue introduction* [online]. Evan You, 2014. [cit. 2023-04-21]. Dostupné z: `https://vuejs.org/guide/introduction.html`.
16. *Vue releases* [online]. Evan You, 2014. [cit. 2023-04-21]. Dostupné z: `https://vuejs.org/about/releases.html`.
17. *Vue FAQ* [online]. Evan You, 2014. [cit. 2023-04-21]. Dostupné z: `https://vuejs.org/about/faq.html#what-s-the-difference-between-vue-2-and-vue-3`.

18. *Vue Quick start* [online]. Evan You, 2014. [cit. 2023-04-21]. Dostupné z: https://vuejs.org/guide/quick-start.html#try-vue-online.
19. *Svelte* [online]. [cit. 2023-04-21]. Dostupné z: https://svelte.dev/.
20. *Svelte documentation* [online]. [cit. 2023-04-21]. Dostupné z: https://svelte.dev/docs.
21. *Svelte Blog* [online]. [cit. 2023-04-21]. Dostupné z: https://svelte.dev/blog.
22. *Svelte npm* [online]. npm, Inc. [cit. 2023-04-21]. Dostupné z: https://www.npmjs.com/package/svelte.
23. BELLAIRS, Richard. *What Is Lint Code? And What Is Linting and Why Is Linting Important?* [online]. Perforce Software, Inc., 2019-03-19. [cit. 2023-04-21]. Dostupné z: https://www.perforce.com/blog/qac/what-lint-code-and-what-linting-and-why-linting-important.
24. *How To Structure React Projects From Beginner To Advanced* [online]. Web Dev Simplified, 2022-07-11. [cit. 2023-04-21]. Dostupné z: https://blog.webdevsimplified.com/2022-07/react-folder-structure/.
25. *What is TSDoc?* [online]. Microsoft, 2022. [cit. 2023-04-21]. Dostupné z: https://tsdoc.org/.
26. *HTTP requests* [online]. IBM Corporation, 2015. [cit. 2023-04-21]. Dostupné z: https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests.
27. SARJEANT, John. *Axios - Promise based HTTP client for the browser and node.js* [online]. Mozilla, 2020. [cit. 2023-04-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.
28. *Promise - JavaScript* [online]. Mozilla, 1998. [cit. 2023-04-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.
29. EAGLES, Lawrence. *A guide to React design patterns* [online]. LogRocket, 2020-12-07. [cit. 2023-04-21]. Dostupné z: https://blog.logrocket.com/react-design-patterns/.
30. *The Best React Design Patterns You Should Know About* [online]. UXPin Sp. z o.o., 2022-12-09. [cit. 2023-04-21]. Dostupné z: https://www.uxpin.com/studio/blog/react-design-patterns/.
31. NIELSEN, Jakob. *10 Usability Heuristics for User Interface Design* [online]. Nielsen Norman Group, 2020-11-15. [cit. 2023-04-21]. Dostupné z: https://blog.logrocket.com/react-design-patterns/.
32. *Material UI - Overview* [online]. Material UI SAS., 2023. [cit. 2023-04-21]. Dostupné z: https://mui.com/material-ui/getting-started/overview/.
33. *Build fast, responsive sites with Bootstrap* [online]. [cit. 2023-04-21]. Dostupné z: https://getbootstrap.com/.
34. *Create mock APIs in seconds* [online]. Mockoon, 2017. [cit. 2023-04-21]. Dostupné z: https://mockoon.com/.
35. PH.D, Ing. Martin Fůs. *Projekt: Interaktivní model oka pro vizualizaci polohy nitrooční čočky v oku* [online]. 2022-03-05. [cit. 2023-04-21]. Dostupné z: https://campuscvut.sharepoint.com/:b:/s/Team-BI-SP1ModelOka/EcuxWsSKS35OukOyBK_eTmEBvjVJoIPYzXZlA8ltv5O3og?e=lzYeTB.
36. *Gartner Glossary - Architecture* [online]. Gartner, Inc., 2023. [cit. 2023-04-21]. Dostupné z: https://www.gartner.com/en/information-technology/glossary/architecture.
37. BAUMGARTNER, Ricardo. *10 Usability Heuristics for User Interface Design* [online]. Rangle.io, 2021-04-27. [cit. 2023-04-21]. Dostupné z: https://rangle.io/blog/how-react-and-redux-brought-back-mvc-and-everyone-loved-it.

38. BAELDUNG. *Difference Between MVC and MVP Patterns* [online]. Tarnum Java SRL, 2022-11-09. [cit. 2023-04-21]. Dostupné z: https://www.baeldung.com/mvc-vs-mvp-pattern.
39. WAMBUA, William. *Understanding Flux Architecture* [online]. Medium, 2023-11-23. [cit. 2023-04-21]. Dostupné z: https://medium.com/swlh/understanding-flux-architecture-9060e5a0399c.
40. SHAH, Sharvin. *How to Use Flux to Manage State in ReactJS - Explained with an Example* [online]. FreeCodeCamp, 2020-04-20. [cit. 2023-04-21]. Dostupné z: https://www.freecodecamp.org/news/how-to-use-flux-in-react-example/.
41. *React Flux Concept* [online]. Javatpoint, 2011. [cit. 2023-04-21]. Dostupné z: https://www.javatpoint.com/react-flux-concept.
42. ABRAMOV, Dan. *Getting Started with Redux* [online]. Redux, 2022-08-18. [cit. 2023-04-21]. Dostupné z: https://redux.js.org/introduction/getting-started.
43. ASIRI, Sidath. *Flux and Redux* [online]. Medium, 2018-04-13. [cit. 2023-04-21]. Dostupné z: https://medium.com/@sidathasiri/flux-and-redux-f6c9560997d7.
44. *What is OAuth 2.0?* [online]. Okta, Inc., 2023. [cit. 2023-04-21]. Dostupné z: https://auth0.com/intro-to-iam/what-is-oauth-2.
45. HARDT, Dick. *RFC FT-IETF-oauth-V2: The oauth 2.0 authorization framework* [online]. 2012-10. [cit. 2023-04-21]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc6749.
46. SAKIMURA, Nat; BRADLEY, John; AGARWAL, Naveen. *RFC FT-IETF-OAUTH-spop: Proof key for code exchange by OAuth public clients* [online]. 2015-09. [cit. 2023-04-21]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc7636.
47. *Authorization Code Flow with Proof Key for Code Exchange (PKCE)* [online]. Okta, Inc., 2023. [cit. 2023-04-21]. Dostupné z: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-proof-key-for-code-exchange-pkce.
48. *React hook form* [online]. BEEKAI. [cit. 2023-04-21]. Dostupné z: https://react-hook-form.com/.
49. *Install Docker Engine on Ubuntu* [online]. Docker Inc., 2023. [cit. 2023-04-21]. Dostupné z: https://docs.docker.com/engine/install/ubuntu/.
50. *Přiložené soubory – Google disk*. Dostupné také z: https://drive.google.com/drive/folders/1BaT2P0fUNe5IMoM6XdjO6Nco0wt7QRgC?usp=sharing.
51. *Docker load* [online]. Docker Inc., 2023. [cit. 2023-04-21]. Dostupné z: https://docs.docker.com/engine/reference/commandline/load/.
52. *Sign in with Google for web – Get you Google API client ID* [online]. Google, 2022-11-11. [cit. 2023-04-21]. Dostupné z: https://developers.google.com/identity/gsi/web/guides/get-google-api-clientid.
53. BASTIEN, J.M. Christian. Usability testing: a review of some methodological and technical aspects of the method. *International Journal of Medical Informatics*. 2010, roč. 79, č. 4, e18–e23. ISSN 1386-5056. Dostupné z DOI: https://doi.org/10.1016/j.ijmedinf.2008.12.004. Human Factors Engineering for Healthcare Applications Special Issue.

Obsah přiloženého média

- `interactiveModelEye-frontend-image.tar` ..... docker image implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis.zip` ..... zkomprimovaná zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF